Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Ciência de Dados

DEEP LEARNING PARA RECONHECIMENTO DE FONEMAS CONCATENADOS

ESTUDANTE PEDRO SILVA VARELA COSTA

Leiria, Setembro de 2024

[15 de setembro de 2024 at 13:13 – versão 0.1]

Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Ciência de Dados

DEEP LEARNING PARA RECONHECIMENTO DE FONEMAS CONCATENADOS

ESTUDANTE PEDRO SILVA VARELA COSTA
Número: 2223244

Dissertação realizada sob orientação do Professor Doutor João da Silva Pereira (joao.pereira@ipleiria.pt).

Leiria, Setembro de 2024

[15 de setembro de 2024 at 13:13 – versão 0.1]

AGRADECIMENTOS

A elaboração desta dissertação não teria sido possível sem o apoio e a colaboração de várias pessoas, às quais gostaria de expressar o meu mais profundo agradecimento.

Em primeiro lugar, quero agradecer a todas as pessoas que, de alguma forma, contribuíram para a recolha de dados. A vossa disponibilidade e generosidade foram essenciais para o desenvolvimento deste trabalho, e sem o vosso contributo, esta dissertação não seria uma realidade.

Gostaria também de deixar um agradecimento muito especial aos meus pais. O vosso apoio incondicional, paciência e encorajamento ao longo de todo este processo foram fundamentais para que eu pudesse concluir este trabalho. Agradeço-vos, não só pelo suporte emocional, mas também por todos os sacrifícios que fizeram para que eu pudesse alcançar os meus objetivos.

Por fim, mas não menos importante, expresso o meu mais sincero agradecimento ao Professor João Pereira (Professor coordenador do departamento de Engenharia Informática da ESTG do Instituto Politécnico de Leiria e investigador no Instituto de Telecomunicações), o meu orientador, pela sua orientação, paciência e conselhos ao longo deste percurso. A sua sabedoria e o seu apoio constante foram fundamentais para a realização desta dissertação, e sou extremamente grato por todo o tempo e dedicação que me dispensou.

RESUMO

Esta dissertação teve como objetivo criar um modelo capaz de classificar fonemas em português, utilizando técnicas avançadas de Deep Learning. Um fonema é a menor unidade de som na linguagem, e a sua correta identificação é essencial para a compreensão da fala. Ao focar na classificação de fonemas, em vez de palavras inteiras, este projeto busca superar desafios relacionados a variações de sotaques ou deficiências na fala, permitindo um reconhecimento mais preciso e inclusivo. Foram exploradas várias técnicas de Deep Learning, que foram aplicadas na análise de Mel-Espectrogramas — representações visuais das frequências dos sons ao longo do tempo. Esses espectrogramas serviram como base para o treino do modelo, permitindo que ele classificasse fonemas com boa precisão em testes de validação. No entanto, o desempenho do modelo foi inferior ao esperado quando testado em novos dados e amostras de áudio, destacando a necessidade de melhorias na sua robustez e capacidade de generalização para diferentes contextos linguísticos.

Palavras-chave: Deep Learning, Fonemas, Mel-Espectrogramas, Reconhecimento de Fala

ABSTRACT

This dissertation aimed to create a model capable of classifying phonemes in Portuguese, using advanced Deep Learning techniques. A phoneme is the smallest unit of sound in a language, and its correct identification is essential for speech comprehension. By focusing on the classification of phonemes instead of whole words, this project seeks to overcome challenges related to accent variations or speech impairments, enabling more accurate and inclusive recognition. Various Deep Learning techniques were explored and applied to the analysis of Mel-Spectrograms — visual representations of sound frequencies over time. These spectrograms served as the foundation for training the model, allowing it to classify phonemes with good accuracy in validation tests. However, the model's performance was below expectations when tested on new data and audio samples, highlighting the need for improvements in its robustness and generalization capability for different linguistic contexts.

Keywords: Deep Learning, Phonemes, Mel-Spectrograms, Speech Recognition

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

BERT	Bidirectional Encoder Representations from Transformers.
CNN	Convolutional Neural Networks.
cuDNN	CUDA Deep Neural Network Library.
DBN	Deep Belief Network.
DNNs	Deep Learning Networks.
GPT	Generative Pre-trained Transformer.
GPU	Unidades de Processamento Gráfico.
GRU	Gated Recurrent Unit.
GUI	Graphical User Interface.
HMMs	Hidden Markov Models.
IVR	Resposta Vocal Interativa.
LSTM	Long Short-Term Memory.
MVA	Mean Validation Accuracy.
MVL	Mean Validation Loss.
RBM	Restricted Boltzmann Machine.
ReLU	Unidade Linear Retificada.
RNA	Redes Neuronais Artificiais.

RNN	Recurrent Neural Network.
SVM	Máquinas de Vetores de Suporte.
tanh	Tangente Hiperbólica.
TPU	Unidades de Processamento Tensor.
UL	Unfrozen Layers.

1

INTRODUÇÃO

A fala é um dos pilares fundamentais da comunicação humana, permitindo a transmissão de ideias, emoções e intenções de forma imediata e fluida. Dentro deste complexo sistema, os fonemas desempenham um papel essencial, servindo como os blocos de construção da linguagem. Um fonema é a menor unidade de som que pode alterar o significado de uma palavra, e a sua combinação cria palavras, frases e pensamentos completos, que estão muitas vezes ausentes na comunicação escrita. Contudo, embora a fala seja uma habilidade inata dos seres humanos, o reconhecimento e a interpretação dos fonemas pelas máquinas continuam a ser um desafio. A diversidade linguística e as variações de pronúncia representam obstáculos significativos para o reconhecimento automático da fala (Graves et al., 2013). As diferenças de sotaque, entoação e até mesmo o ruído de fundo torna o reconhecimento da fala uma tarefa complexa. Neste contexto, o reconhecimento de fonemas tem-se destacado como uma área crucial da inteligência artificial, ao permitir que as máquinas compreendam a estrutura subjacente da fala (Deng e Li, 2013). O objetivo é que as máquinas não apenas identifiquem palavras, mas também sejam capazes de captar as subtilezas e variações que surgem na linguagem natural.

O progresso nas tecnologias de reconhecimento de fala tem sido impulsionado, sobretudo, pelos avanços em Inteligência Artificial, nomeadamente no campo do Deep Learning. Esta abordagem permite às máquinas aprenderem diretamente a partir de uma abundância de dados, sem a necessidade de pré-processamento manual intensivo. Em particular, redes neurais convolucionais e recorrentes destacam-se pela sua capacidade de captar padrões complexos em sinais acústicos, facilitando o reconhecimento de fala com uma precisão cada vez maior (Goodfellow et al., 2016).

O Deep Learning revolucionou a forma como os sistemas de reconhecimento de fala funcionam, permitindo uma adaptação a uma variedade de contextos e utilizadores. Mediante um treino com vastos volumes de dados, estas redes neurais conseguem identificar características subtis nos fonemas e nas palavras, melhorando substancialmente a eficácia dos sistemas de reconhecimento de fala. Esta tecnologia tem sido aplicada em várias áreas, desde assistentes de voz, como a Siri ou a Alexa, até soluções de acessibilidade que permitem a pessoas com deficiências motoras

ou auditivas interagir de forma mais natural com os dispositivos tecnológicos (G. Hinton et al., 2012).

O reconhecimento de fonemas centra-se na identificação e classificação automática de fonemas no sinal acústico. Essencialmente, procura desvendar o complexo código auditivo subjacente à linguagem, permitindo que as máquinas compreendam palavras e frases faladas (L. Rabiner e B.-H. Juang, 1993). No contexto da tecnologia, a capacidade de uma máquina compreender e responder à fala humana tem implicações profundas. Isto vai desde a automação de tarefas quotidianas, até avanços na acessibilidade para pessoas com limitações físicas, abrindo novas possibilidades de inclusão. Para que esta interação seja eficaz, os sistemas de reconhecimento de fala precisam ser robustos, precisos e capazes de lidar com uma vasta gama de variabilidades linguísticas.

Este projeto insere-se no campo do reconhecimento de fonemas em português utilizando técnicas de Deep Learning, visando explorar novas abordagens para o reconhecimento de comandos de voz em tempo real. Por uma análise detalhada dos modelos de redes neurais aplicados ao reconhecimento de fonemas, o estudo foca-se na identificação de características relevantes dos espectrogramas — representações visuais da energia das frequências sonoras ao longo do tempo (Graves et al., 2013). Estes dados servirão de base para treinar modelos de Deep Learning que consigam classificar corretamente fonemas.

As próximas secções deste trabalho estão organizadas da seguinte forma:

- Capítulo 2: Trabalho Relacionado, detalhando os avanços e metodologias ao longo da história para o reconhecimento de voz;
- Capítulo 3: Enquadramento Teórico, detalhando metodologias existentes usadas, como *Machine Learning*, redes neuronais, *deep learning* e '*frameworks*' e bibliotecas;
- Capítulo 4: Desenvolvimento, revisão das técnicas utilizadas, ferramentas e práticas, como a recolha e preparação de dados, implementação dos modelos, estrutura dos testes e resultados obtidos;
- Capítulo 5: Conclusões;

2

TRABALHO RELACIONADO

Este capítulo procura estabelecer o enquadramento teórico que suportará a análise e discussão do tema central abordado neste projeto. Para construir uma base conceptual sólida, iniciamos com uma revisão histórica do estado da arte, seguida de uma análise dos principais conceitos e teorias que caracterizam este vasto campo de estudo. Ao destacar as perspetivas teóricas predominantes e as lacunas existentes no conhecimento, fornecemos um contexto que sublinha a relevância e a originalidade deste trabalho. Este enquadramento teórico será fundamental para a compreensão do desenvolvimento do projeto, bem como das técnicas e metodologias utilizadas.

2.1 HISTÓRIA

2.1.1 *1950-1960: Primeiros passos*

Nas décadas de 1950 e 1960, as primeiras tentativas de reconhecimento de fala baseavam-se principalmente em sistemas baseados em regras. Estes sistemas focam-se em conjuntos explicitamente programados de regras linguísticas e fonéticas e eram desprovidos de aprendizagem automática ou de modelação estatística. Os linguistas e foneticistas estavam fortemente envolvidos na criação destas regras (B. H. Juang e Lawrence R Rabiner, 2004).

Estes sistemas eram compostos por um manual de regras que descrevia as relações entre a linguagem falada e a sua correspondente forma escrita. Também incorporaram um vasto conhecimento fonético e linguístico para definir a forma como os diferentes fonemas e palavras devem ser reconhecidos com base nas suas características acústicas. As regras abrangem aspetos como as variações de pronúncia, os padrões de acentuação e o contexto fonético. Infelizmente estes sistemas eram limitados na sua capacidade de escalonamento para acomodar a imensa complexidade e variabilidade da linguagem natural (Jelinek, 1999). A manutenção e a expansão dos conjuntos de regras revelaram ser uma tarefa difícil. Também a inflexibilidade destes sistemas tornava-os menos adaptáveis a diferentes sotaques,

dialetos e línguas. Os sistemas baseados em regras tinham muitas vezes dificuldades em lidar com a variação dos oradores. Ao contrário das abordagens modernas de aprendizagem automática, os sistemas baseados em regras não tinham a capacidade de aprender com os dados. Em vez disso, dependiam da experiência humana para codificar manualmente as regras linguísticas e fonéticas. Devido às suas limitações só conseguiam atingir uma precisão modesta em tarefas de reconhecimento de fala restritas. Eram frequentemente utilizados em aplicações de palavras isoladas ou de vocabulário limitado (Jelinek, 1999).

2.1.2 *1960-1970: Modelação Acústica*

A modelação acústica é um componente crítico dos sistemas de reconhecimento de fala e nas décadas de 1960 e 1970 girava principalmente em torno de técnicas tradicionais de processamento de sinal e extração manual de características.

Como técnicas de processamento de sinal, a modelação acústica envolvia a aplicação de filtragem e a análise espectral para extrair características relevantes do sinal de áudio. Características como coeficientes espectrais foram usadas para representar as características acústicas da fala. Engenheiros e linguistas conceberam e selecionaram manualmente características que se acreditava serem relevantes para o reconhecimento de fonemas e palavras. Estas características não foram aprendidas a partir de dados, mas sim com base em conhecimentos especializados. Os dados para a modelação acústica eram frequentemente limitados, tanto em termos de quantidade como de variedade. Esta escassez de dados limitou a capacidade de desenvolver modelos sofisticados (Allen, 1991).

Os modelos acústicos eram muitas vezes independentes do contexto, o que significa que não tinham em conta a influência dos fonemas vizinhos nas propriedades acústicas do fonema atual. Capturar a variabilidade na fala devido a fatores como a velocidade de fala, a identidade do locutor e as condições ambientais foi um desafio. Os modelos acústicos não estavam bem equipados para lidar com essas variações. A modelação acústica centrava-se principalmente na modelação de fonemas individuais ou unidades subfonéticas.

Apesar das limitações da modelação acústica nas décadas de 1960 e 1970, esta lançou as bases para os desenvolvimentos subsequentes no reconhecimento da fala, acabando por conduzir à adoção de técnicas estatísticas e de aprendizagem automática, como os *Hidden Markov Models*, que melhoraram significativamente a precisão e a robustez dos sistemas modernos.

2.1.3 *1970-1980: Hidden Markov Models (HMMs)*

Na década de 1970, estes modelos surgiram como a abordagem dominante para o reconhecimento da fala, marcando um momento crucial na história deste domínio e lançando as bases para futuros avanços. Os HMMs representaram uma mudança significativa na forma como os investigadores abordavam o problema da conversão da linguagem falada em texto escrito. Na sua essência, os HMMs são modelos matemáticos que nos ajudam a dar sentido a dados sequenciais, como a fala. São chamados "Hidden"(ou ocultos) porque lidam com informação que não podemos observar diretamente, mas que podemos inferir a partir dos dados que podemos ver (B. H. Juang e ; L. R. Rabiner, 1991).

O *Hidden Markov Model* pode ser explicado em cinco conceitos que são apresentados a seguir:

- Estados: Imagine a fala como uma sequência de sons, ou fonemas, passando de um para outro. Os HMMs dividem esta sequência em diferentes "estados". Cada estado representa um determinado som ou fonema.
- Transições: Os HMMs utilizam probabilidades para descrever a probabilidade de um estado (fonema) se seguir a outro. Por exemplo, a probabilidade de transição do som "S"para o som "P".
- Observações: Dentro de cada estado, existem características observáveis. No contexto do reconhecimento da fala, estas características podem incluir características acústicas como o tom e a duração de um som.
- Variáveis ocultas: A parte "hidden"dos HMMs refere-se ao facto de não sabermos diretamente em que estado nos encontramos num dado momento. Em vez disso, fazemos suposições educadas com base nos dados observados.
- Descodificação: Ao observar as características do áudio, podemos usar HMMs para descodificar a sequência provável de estados (fonemas) que geraram as palavras faladas.

Assim, resumidamente, os HMMs são uma forma de modelar a estrutura subjacente da fala, tendo em conta os diferentes sons, a forma como se sucedem e as características que os definem. Estes modelos foram inovadores na sua capacidade de lidar com dados sequenciais, tornando-os a pedra angular dos sistemas de reconhecimento de fala nos anos 80 e seguintes.

2.1.4 *1980-1990: Avanços nos Hidden Markov Models e as suas aplicações*

Durante este período, os HMMs introduzidos na década de 1970, sofreram aperfeiçoamentos significativos. Os investigadores trabalharam para aumentar a eficácia no reconhecimento de uma gama mais alargada de padrões de fala. Algoritmos de treino foram melhorados e combinados com conjuntos de dados mais extensos. Isto permitiu uma melhor modelação de fonemas e vocabulário complexo, tornando os sistemas de reconhecimento da fala mais versáteis e capazes de lidar com aplicações do mundo real.

Esta década também testemunhou a integração da tecnologia de reconhecimento da fala nas primeiras aplicações práticas. Os sistemas de resposta vocal interativa (IVR) foram introduzidos no sector comercial, permitindo interações automatizadas de serviço ao cliente. O sector da saúde também beneficiou do reconhecimento da fala, uma vez que os profissionais de saúde recorreram cada vez mais a esta tecnologia para a transcrição médica, simplificando os seus processos de documentação.

2.1.5 *1990-2000: Avanços em Modelagem Acústica e Processamento de Linguagem*

De 1990 a 2000 a tecnologia de reconhecimento da fala registou um progresso contínuo. A modelação acústica tornou-se mais refinada e sofisticada, aplicando análise espetral e métodos estatísticos para captar as complexidades da linguagem falada. Estes avanços melhoraram muito a precisão dos sistemas de reconhecimento da fala, tornando-os mais fiáveis para uma gama mais vasta de aplicações.

Além disso, as técnicas de modelação da linguagem evoluíram durante este período. Os modelos linguísticos estatísticos, como os n-gramas, desempenharam um papel crucial na melhoria da compreensão contextual da linguagem falada. Com modelos linguísticos melhorados os sistemas de reconhecimento de voz tornaram-se mais hábeis na distinção entre palavras e frases com características acústicas semelhantes, aumentando significativamente o seu desempenho global (Jurafsky e Martin, 2000).

2.1.6 *2000-2009: A ascensão das Redes Neurais e a adoção comercial*

Os anos entre 2000 e 2009 representaram uma fase de transformação para o reconhecimento de fala. Durante este período, as primeiras experiências com redes neurais ganharam destaque, preparando o terreno para a revolução do *Deep Learning* que

iria surgir no futuro. Embora as redes neuronais ainda não fossem a abordagem dominante, lançaram as bases para avanços substanciais nos anos seguintes (Bengio et al., 2003).

Neste período, assistiu-se também à adoção comercial da tecnologia de reconhecimento da fala em vários sectores. Os sistemas de reconhecimento da fala tornaram-se parte integrante de aplicações como os serviços telefónicos ativados por voz, os serviços de transcrição automática e a tecnologia de comando por voz. O sucesso destas aplicações no mundo real demonstrou a praticabilidade e o potencial do reconhecimento vocal, dando início a uma nova era de interação homem-computador (Lawrence R. Rabiner e Schafer, 2007).

2.1.7 *2009: O início do Deep Learning*

Em 2009, ocorreu uma mudança significativa no campo do reconhecimento de fala quando as *Deep Learning Networks*(DNNs) começaram a ganhar destaque. Este momento marcou o início do impacto substancial para o *Deep Learning*. As DNNs introduziram uma abordagem nova e mais poderosa à modelação acústica, acabando por revolucionar a forma como compreendemos e processamos os dados da fala. As DNNs são uma classe de redes neurais artificiais com várias camadas entre as camadas de entrada e saída. Ao contrário das redes neurais superficiais com apenas uma ou duas camadas ocultas, as DNNs consistem em várias camadas ocultas(*hidden layers*), muitas vezes referidas como arquiteturas "profundas". Estas camadas ocultas permitem que as DNNs aprendam representações complexas e hierárquicas a partir dos dados, tornando-as altamente eficazes para tarefas como o reconhecimento de voz.

As *Deep Neural Networks* têm cinco conceitos-chave que são apresentados a seguir:

- Camada de entrada: esta camada recebe os dados em bruto, tais como características de áudio no caso do reconhecimento de voz.
- Camadas ocultas: as DNNs têm várias camadas ocultas, sendo cada camada constituída por numerosos neurónios artificiais ou nós. Estas camadas são responsáveis pela aprendizagem de padrões e características complexas nos dados. A arquitetura profunda permite às DNNs captar informação hierárquica, com as camadas inferiores a aprenderem características básicas e as camadas superiores a aprenderem representações mais abstratas.

- Pesos e ligações: cada ligação entre neurónios está associada a um peso, que determina a força da ligação. Estes pesos são ajustados durante o treino para otimizar o desempenho da rede.
- Funções de ativação: os neurónios de cada camada aplicam funções de ativação às entradas ponderadas, introduzindo não linearidade no modelo. As funções de ativação comuns incluem as funções sigmoide, Unidade Linear Retificada (ReLU) e a tangente hiperbólica (tanh).
- Camada de saída: A camada final da rede produz a saída, que pode representar vários aspetos dos dados, dependendo da tarefa. No contexto do reconhecimento de fala, podem ser fonemas, palavras ou outras unidades linguísticas.

As DNNs são excelentes na aprendizagem automática e na extração de características relevantes a partir de dados em bruto, eliminando a necessidade de engenharia manual de características que era predominante nas abordagens anteriores de características baseadas em regras e criadas manualmente. Esta capacidade de descobrir automaticamente padrões e representações complexas nos dados fez das DNNs um fator de mudança no reconhecimento de voz, conduzindo a melhorias significativas na precisão e preparando o terreno para novos avanços neste domínio.

2.1.8 *2010: Introdução das Convolutional Neural Networks*

Em 2010, o campo do reconhecimento da fala testemunhou um avanço significativo com a introdução das *Convolutional Neural Networks* (CNN) para extração de características. Este desenvolvimento desempenhou um papel crucial no aumento da precisão e da robustez dos sistemas de reconhecimento da fala. As CNN, originalmente concebidas para o processamento de imagens, foram adaptadas para extrair características significativas de dados de áudio, oferecendo uma abordagem nova e eficaz.

As CNN são uma classe de modelos de *Deep Learning* conhecidos principalmente pelo seu desempenho excecional em tarefas de reconhecimento de imagem. Foram inicialmente inspirados pela forma como o sistema visual humano processa a informação. No contexto do reconhecimento de fala, as CNN foram adaptadas para trabalhar com espectrogramas de áudio, que são representações visuais de dados de som ao longo do tempo. As CNN contêm camadas convolucionais que fazem deslizar pequenos filtros (também chamados *kernel*) pelos dados de entrada, normalmente uma imagem ou um espectrograma no reconhecimento de voz. Estes filtros captam

padrões localizados como arestas ou pequenas características nos dados. Após este processo, as CNN utilizam frequentemente camadas de *pooling* para reduzir as dimensões espaciais dos dados. O agrupamento ajuda a simplificar a representação e a reter características importantes, descartando informações menos relevantes. As CNN concluem normalmente com uma ou mais camadas totalmente ligadas, que são camadas de redes neuronais convencionais. Estas camadas utilizam as características abstraídas e comprimidas das camadas anteriores e fazem previsões com base nelas.

No contexto do reconhecimento da fala, as CNN são utilizadas para a extração de características. Identificam automaticamente padrões e estruturas relevantes nos dados do espectrograma, que representam o conteúdo de frequência do áudio ao longo do tempo. Este processo de extração de características ajuda a captar pistas acústicas essenciais para o reconhecimento da fala. A capacidade das CNN para discernir padrões hierárquicos e estruturas locais no espectrograma contribui para a sua eficácia no reconhecimento da fala.

Ao introduzir as CNN para a extração de características, os investigadores puderam tirar partido do poder da *Deep Learning* na análise de áudio, melhorando a precisão dos sistemas de reconhecimento da fala. Este marco abriu caminho para a integração das CNN com outras técnicas de *Deep Learning*, como as Recurrent Neural Networks (RNNs) e os mecanismos de atenção, para desenvolver modelos mais avançados e precisos para a compreensão da linguagem falada.

2.1.9 *2010-2012: Avanços Tecnológicos*

A década de 2010 marcou um período de transformação no domínio do reconhecimento da fala com a introdução de técnicas avançadas de *Deep Learning*. O reconhecimento da fala deu um salto significativo com a incorporação de técnicas de *Deep Learning*. Destacam-se dois desenvolvimentos fundamentais, nomeadamente a *Restricted Boltzmann machine* (RBM) e as *Deep Belief Network* (DBN). Estes modelos melhoraram significativamente o desempenho, introduzindo uma forma mais sofisticada de representar e processar os dados da fala.

As *Restricted Boltzmann machines* são um tipo de modelo de rede neural utilizado para a aprendizagem não supervisionada. São constituídas por camadas visíveis e ocultas e são treinadas para aprender características a partir de dados de entrada. No reconhecimento da fala, as RBM foram utilizadas para modelar relações complexas entre características acústicas, melhorando a representação dos fonemas. A sua aplicação conduziu à melhora da extração de características de alto nível dos dados

de áudio, ajudando o sistema a captar melhor as complexidades da linguagem. A sua utilização como modelos de pré-treino em Deep Neural Networks (DNNs) reduziu as taxas de erro e aumentou a precisão geral dos sistemas de reconhecimento de fala.

As Deep Belief Networks são um tipo de modelo generativo que empilha várias RBM para formar redes profundas. No reconhecimento da fala, as DBN foram utilizadas para modelar representações hierárquicas complexas de características de áudio. As suas contribuições incluem a aprendizagem da representação de dados da fala em vários níveis de abstração, permitindo uma modelação mais eficaz dos fonemas. Também a integração de DBN em pipelines de modelagem acústica aumentou significativamente a precisão dos sistemas de reconhecimento de fala. Embora as melhorias percentuais exatas na precisão possam variar com base em implementações e conjuntos de dados específicos, estas técnicas, em particular a utilização de RBM e DBN, deram início a uma nova era de sistemas de reconhecimento da fala mais precisos e robustos.

Em 2012, os sistemas baseados em Deep Learning começaram a dominar o campo do reconhecimento de fonemas. A utilização de DNNs e CNN na modelação acústica marcou uma mudança significativa. Estes desenvolvimentos reduziram significativamente as taxas de erro de reconhecimento de fonemas, conduzindo a sistemas de reconhecimento da fala mais fiáveis e eficazes na década de 2010. Mais uma vez, as melhorias exatas da precisão dependem de implementações e conjuntos de dados específicos, mas foram muitas vezes substanciais, ultrapassando os limites da tecnologia de reconhecimento da fala.

2.1.10 *2013-2018: O surgimento das Recurrent Neural Networks e Long Short-Term Memory*

Nesta época o panorama do reconhecimento da fala sofreu uma mudança transformadora com a adoção generalizada de Recurrent Neural Networks (RNNs), particularmente as revolucionárias Long Short-Term Memory (LSTM). Estas arquiteturas avançadas de redes neuronais foram fundamentais para a introdução de uma abordagem mais dinâmica e consciente do contexto à modelação de sequências, melhorando significativamente a precisão do reconhecimento de fonemas e do processamento da fala na totalidade.

As Recurrent Neural Networks são uma classe de redes neurais concebidas para lidar com dados sequenciais, o que as torna especialmente adequadas para tarefas dependentes do tempo, como o reconhecimento da fala. Ao contrário das redes

neurais básicas, que processam os dados de forma independente, as RNN possuem um *hidden state* que captura informações de etapas de tempo anteriores. Este *hidden state* permite que as RNN modelem as dependências temporais e o contexto dentro das sequências. No entanto, as RNN tradicionais podem sofrer de gradientes que desaparecem ou explodem, o que prejudica a sua capacidade de captar dependências de longo alcance. Essa limitação levou ao desenvolvimento de variantes mais avançadas de RNN, incluindo as redes LSTM.

As redes Long Short-Term Memory são um tipo especializado de RNN que supera o problema do desaparecimento de gradiente. Elas apresentam uma arquitetura mais complexa, com células de memória e portas que controlam o fluxo de informações. Os principais componentes das redes LSTM são:

- Células de memória: estas células armazenam informações sobre longas sequências, permitindo que os LSTM capturem dependências de longo alcance nos dados.
- Porta de esquecimento: decide qual informação deve ser descartada da célula de memória e qual deve ser mantida.
- Porta de entrada: determina que nova informação deve ser adicionada à célula de memória.
- Porta de saída: regula a informação passada para o próximo intervalo de tempo e para a saída.

As RNN, incluindo as LSTM, trouxeram melhorias significativas para o reconhecimento da fala, aqui estão alguns dos pontos mais fortes que contribuíram para o aumento da precisão do reconhecimento de fonemas:

- Modelação de contexto: a captura de contexto em dados sequenciais. Aprender a reconhecer não apenas fonemas individuais, mas também como eles se relacionam com fonemas vizinhos, melhorando a precisão geral do reconhecimento de fonemas.
- Dependências de longo alcance: abordaram o desafio de capturar dependências de longo alcance na fala. Isto é crucial para tarefas como a deteção de transições de fonemas que podem estender-se ao longo do tempo.
- Robustez: tornaram os sistemas mais robustos às variações de pronúncia, velocidade de fala e ruído de fundo, considerando o contexto mais alargado da língua falada.

O advento das RNN e das LSTM no reconhecimento da fala lançou as bases para sistemas de reconhecimento da voz mais conscientes do contexto, precisos e naturais, contribuindo para a adoção generalizada de tecnologias da fala em várias aplicações, desde assistentes virtuais a serviços de transcrição automática.

2.1.11 *2018-Atualidade: Avanços Modernos*

Nos últimos anos, registaram-se progressos notáveis no domínio do reconhecimento da fala e dos fonemas, impulsionados por tecnologias inovadoras e novas abordagens. Estes avanços aproximaram-nos da obtenção de interações homem-máquina naturais e precisas em várias aplicações. Eis um resumo de alguns destes desenvolvimentos recentes:

Em 2018, a introdução de modelos baseados em *transformers*, exemplificados pelo modelo *Wave2Vec*, marcou um avanço significativo no reconhecimento de fala e fonemas. A arquitetura *transformer*, originalmente desenvolvida para o processamento de linguagem natural, foi adaptada para processar dados de áudio, permitindo uma aprendizagem de representação mais eficaz e uma melhor precisão. O *Wave2Vec* é um modelo baseado na arquitetura *Transformer*, excelente na captação de informações contextuais de sinais de áudio. Segmenta o áudio em pequenos pedaços, ou "*frames*"e processa-os em paralelo, facilitando um reconhecimento mais eficiente e preciso dos fonemas. A capacidade do modelo *Wave2Vec* de trabalhar diretamente com dados de áudio em bruto conduziu a resultados impressionantes, reduzindo a dependência da engenharia de características tradicional.

Em 2019, a integração da aprendizagem por transferência, especificamente a partir de modelos pré-treinados como o *Bidirectional Encoder Representations from Transformers* (BERT) e o *Generative Pre-trained Transformer* (GPT), começou a transformar as tarefas de reconhecimento de fala e fonemas. Estes modelos foram originalmente desenvolvidos para dados de texto, mas aproveitaram o seu conhecimento pré-treinado para a compreensão do áudio. As técnicas de aprendizagem por transferência permitiram aos investigadores aproveitar as vastas quantidades de dados de texto em que estes modelos foram treinados. Esta transferência de conhecimentos ajudou a melhorar a modelação acústica e fonética, inicializando os modelos com uma base sólida na compreensão da linguagem.

Na década de 2020, o trabalho em curso na integração de sistemas de reconhecimento da fala *End-to-end* é um ponto focal de investigação e desenvolvimento. Os sistemas *End-to-end* combinam a modelação acústica e a modelação linguística

numa arquitetura única e unificada. O mapeamento direto do áudio para fonemas ou palavras é uma abordagem mais holística, permitindo um reconhecimento mais preciso e consciente do contexto. Estes sistemas podem potencialmente reduzir as complexidades da formação e aumentar a eficiência do reconhecimento da fala.

Estes recentes avanços sublinham a evolução do reconhecimento da fala e dos fonemas, oferecendo modelos mais eficientes e precisos e abrindo caminho a '*interfaces*' e aplicações de voz sem descontinuidades. A fusão dos modelos mais avançados, a aprendizagem por transferência e as abordagens *End-to-end* representam uma nova era na busca de uma tecnologia de reconhecimento da fala mais humana e contextualmente consciente.

3

ENQUADRAMENTO

Neste capítulo, serão discutidos os diferentes conceitos teóricos que serviram para o desenvolvimento deste projeto. Assim, tornou-se fundamental realizar uma pesquisa detalhada sobre as tecnologias existentes e essenciais, bem como analisar diversos estudos conduzidos ao longo dos anos. Esse processo foi crucial para a construção das bases que sustentaram a investigação realizada no campo do *Deep Learning* voltado ao reconhecimento de voz.

3.1 *MACHINE LEARNING*

Nos últimos anos, a área de Machine Learning (ML) tem-se destacado como uma das disciplinas mais influentes na ciência de dados e na tecnologia da informação. Com a crescente disponibilidade de grandes volumes de dados (Big Data) e o aumento do poder computacional, as técnicas de Machine Learning têm sido cada vez mais aplicadas em diversas áreas. O objetivo principal desta disciplina é desenvolver algoritmos e modelos que permitam aos sistemas computacionais aprenderem com os dados e melhorarem o seu desempenho ao longo do tempo, sem a necessidade de serem explicitamente programados para realizar tarefas específicas.

Machine Learning pode ser definido como um subcampo da inteligência artificial que se preocupa com a construção de sistemas que aprendem a partir de dados (Alpaydın, 2014). Estes sistemas utilizam métodos estatísticos para identificar padrões e fazer previsões ou decisões baseadas em dados novos, ou dados desconhecidos. A aprendizagem em Machine Learning pode ser categorizada em três principais tipos:

- Aprendizagem Supervisionada: Neste tipo de aprendizagem, os modelos são treinados com dados rotulados, ou seja, o algoritmo recebe entradas (‘*inputs*’) e saídas (‘*outputs*’) conhecidas e aprende a mapear as entradas para as saídas. Exemplos comuns incluem classificação e regressão. Um exemplo clássico de um problema de classificação é o reconhecimento de dígitos manuscritos, onde o objetivo é classificar imagens de dígitos (Alpaydın, 2014).

- Aprendizagem Não Supervisionada: Aqui, o algoritmo é treinado sem rótulos. O objetivo é encontrar padrões ocultos ou estrutura nos dados. Os exemplos mais comuns são o *clustering* (agrupamento) e a redução de dimensionalidade. O *clustering* é frequentemente utilizado em '*marketing*' para segmentar clientes com base em comportamentos semelhantes (Goodfellow et al., 2016).
- Aprendizagem por Reforço (*Reinforcement Learning*): Este tipo de aprendizagem envolve a interação de um agente com o seu ambiente, onde o agente aprende a tomar ações que maximizam uma recompensa cumulativa ao longo do tempo. É frequentemente utilizada em áreas como jogos e robótica.

Os métodos de Machine Learning são vastos e variados, mas podem ser categorizados em algumas abordagens principais. Um dos métodos mais utilizados são as Árvores de Decisão, que são modelos de previsão capazes de mapear observações sobre um 'item' para conclusões acerca do valor alvo desse 'item'. A construção destas árvores envolve a divisão recursiva dos dados em subconjuntos homogéneos. Outro método importante são as Máquinas de Vetores de Suporte (SVM). O SVM é um algoritmo de classificação poderoso que tenta encontrar um hiperplano que separa os dados de diferentes classes com a maior margem possível. Este método é amplamente utilizado em problemas de classificação e regressão (Bishop M. Christopher, 2006). As Redes Neurais Artificiais são outra abordagem fundamental, inspirada na estrutura do cérebro humano. Estas redes são compostas por camadas de nós (neurónios) que processam e transmitem informações. Têm sido a base para os recentes avanços em Deep Learning, especialmente no reconhecimento de imagem e no processamento de linguagem natural (Murphy, 2012). Devido à sua importância para este projeto, as Redes Neurais Artificiais serão abordadas em maior detalhe no próximo capítulo. Por fim, temos os *'Ensembles'*, métodos que combinam múltiplos modelos para melhorar a precisão das previsões. Exemplos incluem *Random Forests* e *Gradient Boosting*, que são especialmente eficazes em problemas de classificação e regressão.

As aplicações de Machine Learning são amplas e impactantes. Alguns exemplos incluem:

- Medicina: Em saúde, o Machine Learning é utilizado para prever surtos de doenças, diagnosticar condições a partir de imagens médicas, e até personalizar tratamentos baseados no perfil genético do paciente.
- Finanças: No setor financeiro, modelos de Machine Learning são usados para prever movimentos de mercado, detetar fraudes e otimizar carteiras de investimentos.

- Indústria Automóvel: Com a evolução dos veículos autónomos, o Machine Learning tem sido fundamental no desenvolvimento de sistemas de perceção, tomada de decisão e controlo, que permitem que veículos operem sem intervenção humana.
- Marketing e Vendas: Empresas utilizam Machine Learning para analisar o comportamento dos consumidores, segmentar mercados, e personalizar campanhas de marketing, aumentando assim a eficácia e retorno sobre o investimento.

Apesar dos avanços significativos, o Machine Learning enfrenta vários desafios. Um dos principais é a necessidade de uma abundância de dados etiquetados para treinar modelos eficazes, o que nem sempre está disponível. Além disso, a interpretação e explicabilidade dos modelos são uma preocupação crescente, especialmente em setores críticos como a saúde e finanças, onde as decisões tomadas por modelos de Machine Learning têm um impacto significativo na vida das pessoas. Outro desafio importante é a questão ética. Algoritmos de Machine Learning têm o potencial de perpetuar preconceitos e discriminações se forem alimentados com dados tendenciosos. Por isso, é essencial que os profissionais na área adotem práticas de desenvolvimento responsáveis e estejam atentos às implicações sociais das tecnologias que criam (Russell e Norvig, s.d.). O futuro do Machine Learning promete ser excitante, com avanços contínuos na capacidade de processamento, novas arquiteturas de redes neurais e melhores técnicas de aprendizagem. A integração de Machine Learning com outras áreas emergentes, como a Internet das Coisas (IoT) e a computação quântica, poderá abrir novas possibilidades e aplicações. No entanto, é imperativo que o desenvolvimento nesta área seja acompanhado de uma reflexão profunda sobre as suas implicações éticas e sociais, garantindo que a tecnologia seja utilizada para o bem comum.

3.2 REDES NEURONAIS ARTIFICIAIS

As redes neuronais artificiais (RNA) são uma classe de algoritmos de aprendizagem automática que têm a capacidade de modelar e resolver problemas complexos, inspirando-se nas redes neuronais biológicas. As RNA são compostas por unidades de processamento simples chamadas neurónios, que estão organizados em camadas e conectados entre si por meio de sinapses artificiais. Estas redes têm sido amplamente utilizadas em várias aplicações, como reconhecimento de padrões, processamento de linguagem natural, visão computacional, e muitas outras. O conceito de redes

neuronais remonta aos estudos de neurociência dos anos 1940 e 1950, quando Warren McCulloch e Walter Pitts propuseram um modelo matemático de neurónio (Mcculloch e Pitts, 1990). Desde então, as RNA têm evoluído significativamente, especialmente com o advento de técnicas de *Deep Learning* que permitem a construção de redes muito mais complexas e profundas.

As redes neuronais consistem tipicamente em três tipos de camadas: camada de entrada (*input layer*), camada oculta (*hidden layer*) e camada de saída (*output layer*). A camada de entrada recebe os dados brutos e esses dados são transmitidos às camadas ocultas, onde ocorrem a maioria dos cálculos e transformações. Finalmente, a camada de saída produz o resultado final da rede, que pode ser uma classificação, uma previsão, ou outra forma de saída numérica, ou categórica. Cada neurónio na rede realiza uma soma ponderada das suas entradas e aplica uma função de ativação (*activation function*) ao resultado. As funções de ativação são fundamentais para introduzir não-linearidades no modelo, o que permite que a rede aprenda relações complexas nos dados. Algumas das funções de ativação mais comuns incluem a função sigmoide, a tangente hiperbólica (tanh), e a unidade linear retificada (ReLU) (Glorot e Bengio, s.d.). A função ReLU, em particular, tem sido amplamente adotada devido à sua simplicidade computacional e ao seu bom desempenho em modelos de *Deep Learning*. A ReLU é definida como $f(x) = \max(0, x)$, o que significa que qualquer entrada negativa é convertida em zero, enquanto as entradas positivas permanecem inalteradas. Esta característica permite que a rede trate de forma eficaz as questões de desaparecimento do gradiente, um problema comum em redes profundas (Nair e G. E. Hinton, s.d.).

O processo de aprendizagem em redes neuronais envolve a minimização de uma função de custo ou erro, que mede a discrepância entre as predições da rede e os valores reais. A minimização é realizada via um processo chamado retropropagação (*backpropagation*), que utiliza o método do gradiente descendente para ajustar os pesos da rede (Rumelhart et al., 1986). A retropropagação funciona calculando o gradiente da função de custo relativamente a cada peso da rede, e depois atualizando os pesos na direção oposta ao gradiente, na tentativa de reduzir o erro. Este processo é repetido iterativamente, passando várias vezes por todo o conjunto de dados de treino, até que a função de custo seja minimizada.

Com o aumento da capacidade computacional e a disponibilidade de grandes conjuntos de dados, possibilitou-se treinar redes neuronais com muitas camadas ocultas. Estas redes, conhecidas como redes neuronais profundas (*deep neural networks*), têm mostrado desempenhos impressionantes em diversas tarefas, superando métodos tradicionais em muitas aplicações (Lecun et al., 2015). As redes profundas são capa-

zes de aprender representações hierárquicas dos dados, onde cada camada sucessiva captura padrões mais abstratos e complexos. Por exemplo, numa rede treinada para reconhecimento de imagens, as camadas iniciais podem aprender a identificar bordas e texturas simples, enquanto as camadas mais profundas identificam formas, objetos, e até mesmo classes específicas de objetos (Krizhevsky et al., s.d.). Apesar dos avanços, as redes neuronais profundas também apresentam vários desafios. O treino de redes muito profundas pode ser computacionalmente caro e pode exigir uma quantidade significativa de dados. Além disso, redes profundas são suscetíveis ao sobreajuste (*overfitting*), onde o modelo aprende detalhes e ruídos do conjunto de treino ao invés de generalizar para novos dados (Srivastava et al., 2014). Diversas técnicas têm sido desenvolvidas para mitigar esses desafios, incluindo regularização (como *dropout* e L2), aumento de dados (*data augmentation*), e utilização de redes pré-treinadas por meio de transferência de aprendizagem (*transfer learning*). A escolha cuidadosa da arquitetura da rede e das técnicas de otimização continua a ser um campo ativo de pesquisa (Pan e Yang, 2010).

As redes neuronais têm sido aplicadas com sucesso em muitas áreas. No reconhecimento de fala, as RNA são usadas para transcrever áudio em texto com alta precisão. Em visão por computador, as Redes Neurais Convolucionais (*Convolutional Neural Network* - CNN), um tipo especializado de RNA, são usadas para identificar objetos, rostos, e até mesmo diagnosticar doenças a partir de imagens médicas. No processamento de linguagem natural, Redes Neurais Recorrentes (*Recurrent Neural Network* - RNN) e as suas variantes, como LSTM e Gated recurrent unit (GRU), têm sido utilizadas para tradução automática, geração de texto, e análise de sentimentos (Hochreiter e Schmidhuber, 1997). Além disso, as redes neuronais também são aplicadas em sistemas de recomendação, jogos, finanças, e muitas outras áreas, demonstrando a sua versatilidade e poder. Para concluir, estas redes representam uma das ferramentas mais poderosas no campo da aprendizagem automática. A sua capacidade de modelar relações complexas e de aprender com grandes volumes de dados tornou-as indispensáveis em muitas aplicações modernas. No entanto, a complexidade do treino e a necessidade de grandes quantidades de dados e recursos computacionais apresentam desafios significativos. A pesquisa contínua neste campo está focada em tornar estas redes mais eficientes, interpretáveis e generalizáveis.

3.3 DEEP LEARNING

Deep Learning é uma subárea do *Machine Learning* que tem ganho destaque significativo nos últimos anos devido ao seu sucesso em diversas aplicações complexas, como reconhecimento de imagem, processamento de linguagem natural e jogos. Esta abordagem de aprendizagem automática caracteriza-se pela utilização de redes neurais profundas, ou seja, redes compostas por várias camadas de neurónios, que são capazes de aprender representações hierárquicas dos dados. A capacidade de modelar e aprender a partir de grandes volumes de dados tem permitido ao *Deep Learning* alcançar resultados impressionantes em tarefas que antes eram consideradas difíceis ou até impossíveis para sistemas computacionais. A evolução do *Deep Learning* está intimamente ligada aos avanços em poder computacional e à disponibilidade de grandes conjuntos de dados, fatores que possibilitaram a aplicação de redes neurais com muitas camadas. A ideia fundamental por trás do *Deep Learning* é que, ao aumentar o número de camadas numa rede neural, esta se torna capaz de aprender representações cada vez mais abstratas dos dados, o que resulta num modelo com maior capacidade de generalização (Lecun et al., 2015). As arquiteturas de redes neurais profundas são variadas, mas entre as mais populares estão as CNN e as RNN.

As CNN são especialmente eficazes em tarefas de processamento de imagem e visão computacional. Estas redes são compostas por camadas convolucionais, que aplicam filtros para extrair características locais das imagens, como bordas, texturas e formas. Uma das principais vantagens das CNN é a sua capacidade de capturar as relações espaciais nas imagens, o que permite que estas redes sejam altamente eficazes na deteção e reconhecimento de objetos em imagens (Krizhevsky et al., s.d.). A arquitetura das CNN inclui geralmente camadas convolucionais seguidas por camadas de *'pooling'*, que reduzem a dimensionalidade das características extraídas, e camadas totalmente conectadas (*fully connected layers*) que realizam a classificação final. A introdução das CNN revolucionou o campo da visão computacional, como demonstrado pelo sucesso da arquitetura AlexNet na competição ImageNet em 2012, onde superou largamente as abordagens tradicionais (Szegedy et al., 2014).

As RNN, por outro lado, são projetadas para lidar com dados sequenciais, como texto, áudio ou séries temporais. A característica distintiva das RNN é a sua capacidade de manter uma memória interna que permite que a rede processe sequências de dados, onde a ordem dos elementos é importante. As RNN são amplamente utilizadas em tarefas de processamento de linguagem natural, como tradução automática, geração de texto e análise de sentimentos (Mikolov et al.,

2010). No entanto, as RNN tradicionais enfrentam problemas com a aprendizagem de dependências de longo prazo devido ao fenómeno do gradiente que desaparece, o que dificulta a capacidade da rede de aprender padrões em sequências longas. Para mitigar este problema, foram desenvolvidas variantes como as LSTM e GRU, que introduzem mecanismos internos para controlar o fluxo de informações através das camadas, melhorando assim a capacidade da rede de aprender dependências de longo prazo (Hochreiter e Schmidhuber, 1997).

O treino de redes neurais profundas segue uma metodologia semelhante ao treino de redes neurais mais simples, mas com algumas complexidades adicionais devido ao grande número de parâmetros envolvidos. O processo de treino envolve a minimização de uma função de custo, como o erro quadrático médio para tarefas de regressão ou a entropia cruzada para tarefas de classificação. A minimização é realizada utilizando o algoritmo de retropropagação combinado com otimizações como o gradiente descendente estocástico (SGD), que atualiza os pesos da rede de forma iterativa (Goodfellow et al., 2016). Devido à complexidade das redes profundas, é comum enfrentar problemas como o sobreajuste (*overfitting*), onde a rede se torna excessivamente ajustada aos dados de treino e não generaliza bem para novos dados. Para combater o sobreajuste, técnicas como *dropout*, regularização L2, e *data augmentation* são frequentemente utilizadas. O *dropout*, por exemplo, é uma técnica que desativa aleatoriamente uma fração dos neurónios durante o treino, forçando a rede a aprender representações mais robustas (Srivastava et al., 2014).

Outro desafio no treino de redes profundas é o tempo e o poder computacional necessários. O treino de redes com milhões de parâmetros pode ser extremamente lento em hardware convencional, por isso, a utilização de Unidades de Processamento Gráfico (GPU) e Unidades de Processamento Tensor (TPU) tem-se tornado comum, pois estas unidades são altamente otimizadas para operações matriciais abundantes no treino de redes neurais (Dean et al., s.d.). Os avanços no *Deep Learning* têm impulsionado inovações em várias áreas. Uma das aplicações mais notáveis é a criação de modelos de linguagem como o GPT (Generative Pre-trained Transformer), que são capazes de gerar texto coerente e relevante com base em entradas de texto (Radford et al., s.d.). Estes modelos são treinados em grandes corpora de texto e utilizam arquiteturas de transformadores que permitem capturar dependências de longo alcance no texto, superando as limitações das RNN.

Outra área que tem beneficiado enormemente do *Deep Learning* é a condução autónoma. As redes neurais profundas são utilizadas para processar as imagens captadas pelas câmaras dos veículos e identificar objetos, sinalização rodoviária, e outros veículos. Combinadas com sensores adicionais, como radares, estas redes

fornecem aos veículos a capacidade de perceber o ambiente e tomar decisões em tempo real (Bojarski et al., 2016). No campo da saúde, o *Deep Learning* tem sido utilizado para analisar imagens médicas e ajudar na deteção precoce de doenças. Por exemplo, redes CNN são utilizadas para analisar radiografias e identificar sinais de doenças como o cancro do pulmão com uma precisão muitas vezes superior à de especialistas humanos.

Embora o *Deep Learning* tenha alcançado feitos notáveis, ainda enfrenta desafios significativos. Um dos principais desafios é a necessidade de grandes quantidades de dados rotulados para treinar modelos eficazes, o que pode ser difícil de obter em muitas áreas. Além disso, as redes neurais profundas são frequentemente vistas como "caixas pretas", onde a interpretação das decisões do modelo é difícil, o que levanta questões sobre a confiança e transparência dos modelos em aplicações críticas (Lipton, 2016). Apesar destes desafios, a pesquisa continua a avançar em direção ao desenvolvimento de modelos mais eficientes, interpretáveis que requerem menos dados. Novas técnicas como *transfer learning*, onde um modelo treinado numa tarefa pode ser adaptado para outra tarefa com menos dados, estão a abrir novas possibilidades para o *Deep Learning* (Pan e Yang, 2010). O futuro do *Deep Learning* parece promissor, com a expectativa de que continue a ser uma força motriz na inteligência artificial e nas tecnologias emergentes. A sua integração com áreas como a Internet das Coisas (IoT) e a computação quântica poderá levar a novas descobertas e aplicações que continuam por explorar.

Em suma, o *Deep Learning* representa um marco na evolução da aprendizagem automática, trazendo consigo uma nova era de inovação e capacidade de modelar problemas complexos. As suas aplicações abrangem várias indústrias e têm o potencial de transformar a forma como a sociedade interage com a tecnologia. No entanto, é crucial continuar a investigar formas de tornar o *Deep Learning* mais acessível, interpretável e ético, para que os seus benefícios possam ser amplamente distribuídos e compreendidos.

3.4 MEL-ESPECTROGRAMAS

Os mel-espectrogramas são uma representação essencial no processamento de áudio, especialmente em tarefas relacionadas ao reconhecimento de fala e classificação de áudio. Esta representação visual do espectro de frequências ao longo do tempo é ajustada para melhor refletir a perceção auditiva humana, o que a torna uma ferramenta poderosa para analisar características acústicas de sinais de áudio. Nesta

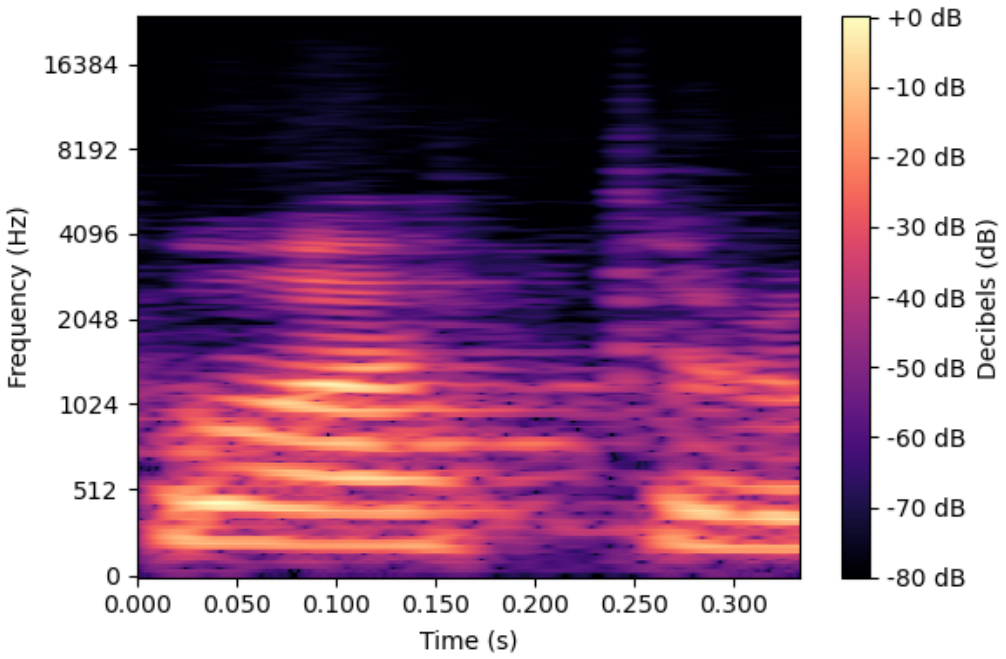


Figura 1: Imagem de exemplo de um mel-espectrograma

secção, exploramos o conceito de mel-espectrogramas, a sua construção, e como são utilizados no treino de modelos de *machine learning*, com especial ênfase em redes neurais convolucionais (Zhang et al., s.d.).

Um mel-espectrograma é uma transformação do espectrograma de um sinal de áudio, projetado para capturar melhor as características percetíveis pelo ouvido humano. O espectrograma convencional é uma representação em 2D que mostra como a energia de um sinal de áudio se distribui em diferentes frequências (escala vertical) ao longo do tempo (escala horizontal), como demonstra a figura 1. Para calcular um espectrograma, o sinal de áudio é dividido em pequenos segmentos temporais, chamados janelas, e uma Transformada de Fourier é aplicada a cada janela para obter o espectro de frequências desse segmento. No entanto, o ouvido humano não percebe as frequências de maneira linear. A nossa sensibilidade auditiva segue uma escala logarítmica, o que significa que somos mais sensíveis a mudanças de frequência em tons mais baixos do que em tons mais altos. A escala Mel é uma escala não-linear de frequências, baseada em como os humanos percebem o som, em que as frequências mais baixas têm uma resolução mais alta em comparação com as frequências mais altas. Ao mapear as frequências do espectrograma para a escala Mel, obtemos o mel-espectrograma, que oferece uma representação mais adequada para tarefas como o reconhecimento de padrões em sinais de áudio.

Os mel-espectrogramas oferecem várias vantagens em comparação com outras representações de áudio em tarefas de reconhecimento de fala e classificação de som. Uma das principais vantagens é que eles capturam tanto as características temporais quanto as espectrais do sinal de áudio, o que é fundamental para distinguir diferentes fonemas e padrões de fala. Além disso, a escala Mel alinha-se melhor com a forma como os humanos percebem o som. Ao ajustar as frequências do espectro conforme a sensibilidade humana, os mel-espectrogramas reduzem a complexidade dos dados ao mesmo tempo que mantêm a informação relevante para a interpretação auditiva. Isto permite que os modelos de machine learning se concentrem em padrões significativos, ignorando informações irrelevantes ou redundantes. Outro benefício é a possibilidade de utilizar técnicas de visão computacional em conjunto com mel-espectrogramas. Uma vez que estas representações são essencialmente imagens, redes neurais convolucionais (CNN) podem ser treinadas para identificar padrões nas imagens, como mudanças na frequência e intensidade que correspondem a diferentes fonemas ou palavras. As CNN têm-se mostrado extremamente eficazes em tarefas de classificação de imagens, e a utilização de mel-espectrogramas permite que esta abordagem seja aplicada a dados de áudio. O uso de mel-espectrogramas também facilita o diagnóstico e a interpretação do desempenho do modelo. Uma vez que as entradas são imagens, é possível visualizar diretamente os padrões que o modelo está a aprender a reconhecer, o que pode ajudar a identificar possíveis problemas nos dados ou no próprio modelo (Salamon e Bello, 2016).

Apesar das suas vantagens, o uso de mel-espectrogramas não está isento de desafios. Uma das principais limitações é que, ao converter o sinal de áudio numa representação visual, perde-se parte da informação temporal exata, o que pode ser crítico em algumas aplicações. Além disso, o cálculo e a geração de mel-espectrogramas podem ser computacionalmente intensivos, especialmente para grandes volumes de dados de áudio. Outro fator a considerar é a qualidade dos dados de áudio originais. Se os áudios contiverem muito ruído ou distorções, isso será refletido nos mel-espectrogramas, o que pode prejudicar o desempenho do modelo. Por isso, etapas de pré-processamento, como a filtragem de ruído, são cruciais para garantir que os mel-espectrogramas capturam de forma precisa as características acústicas desejadas. Por fim, a escolha dos parâmetros para a geração dos mel-espectrogramas, como o tamanho da janela e o número de bandas Mel, pode ter um impacto significativo no desempenho do modelo. Parâmetros inadequados podem levar à perda de informações importantes ou à introdução de artefactos indesejados, dificultando o aprendizado do modelo.

Neste projeto, utilizaremos mel-espectrogramas como a principal forma de entrada para os nossos modelos de machine learning, aproveitando a sua capacidade de capturar características relevantes do áudio de forma compacta e eficiente. Através da combinação de mel-espectrogramas com CNN e técnicas de aumento de dados, esperamos desenvolver um modelo robusto e preciso para o reconhecimento de fonemas, capaz de generalizar bem para novos exemplos de áudio.

3.5 FRAMEWORKS E BIBLIOTECAS

A crescente demanda por sistemas de aprendizagem automática e, em particular, de *Deep Learning*, impulsionou o desenvolvimento de diversas ferramentas que facilitam a implementação, treino e avaliação de modelos complexos. Entre as mais destacadas estão *TensorFlow, Keras e Scikit-Learn*, que se tornaram componentes essenciais no ecossistema de *machine learning*. Estas bibliotecas e frameworks oferecem um conjunto robusto de funcionalidades que simplificam o processo de desenvolvimento de modelos, desde a manipulação de dados até à implementação de algoritmos avançados.

3.5.1 *CUDA Toolkit*

O *CUDA Toolkit*, desenvolvido pela NVIDIA, é uma plataforma de computação paralela e um modelo de programação que permite a utilização da GPU para realizar operações computacionais de forma acelerada. O CUDA Toolkit inclui compiladores, bibliotecas e ferramentas essenciais para o desenvolvimento de aplicações que utilizam GPUs NVIDIA. Este 'kit' permite que tarefas intensivas em computação sejam distribuídas entre vários núcleos da GPU, acelerando significativamente o processo de treino de modelos de *deep learning*. A sua instalação é fundamental para que frameworks como TensorFlow possam utilizar a GPU de maneira eficiente no treino de modelos complexos. A versão utilizada neste projeto foi a 11.8.0, compatível com o hardware e software instalados.

3.5.2 *cuDNN (CUDA Deep Neural Network Library)*

O *cuDNN* é uma biblioteca acelerada por GPU, desenvolvida pela NVIDIA, projetada especificamente para melhorar o desempenho das operações em redes neurais

profundas. Esta biblioteca é altamente otimizada para funcionar em conjunto com o CUDA, proporcionando acelerações substanciais em operações como convoluções, *pooling* e funções de ativação em redes neurais. Neste projeto, foi utilizada a versão 8.6.0 do cuDNN. Após o download e extração dos ficheiros, foi necessário copiar o conteúdo para o diretório do CUDA, garantindo uma integração suave entre as ferramentas. A presença do cuDNN é crucial para maximizar o desempenho de frameworks como TensorFlow, reduzindo o tempo de treino de modelos complexos ao aproveitar o poder computacional das GPUs.

3.5.3 *Conda e Criação de Ambiente com TensorFlow GPU*

O *Conda* é um gestor de pacotes e ambientes muito popular, especialmente no desenvolvimento em Python. Permite a criação de ambientes isolados com dependências específicas, evitando conflitos entre bibliotecas e assegurando a reprodutibilidade dos projetos. Neste projeto, foi criado um ambiente Conda com o nome *tensorflow-gpu*, utilizando a versão 3.9 do Python. Este ambiente foi configurado para suportar o uso de GPU com TensorFlow, permitindo que o treino dos modelos de *deep learning* ocorresse de forma acelerada e eficiente. A criação de ambientes específicos com Conda facilita a gestão de diferentes configurações de software, que podem ser ajustadas conforme as necessidades do projeto.

3.5.4 *TensorFlow*

O *TensorFlow* é uma das mais populares bibliotecas de *deep learning*, desenvolvida pela Google Brain Team e lançada em 2015 como um projeto *'open-source'*. Projetada para a computação numérica, TensorFlow é amplamente utilizada para implementar e treinar redes neuronais profundas. A sua flexibilidade permite que seja utilizada tanto em dispositivos móveis quanto em infraestruturas distribuídas em larga escala, como *'clusters'* de servidores e GPUs. A arquitetura do TensorFlow é baseada em grafos computacionais, onde os nós representam operações matemáticas e as arestas representam os dados (tensores) que fluem entre essas operações. Esta abordagem facilita a visualização e a depuração de modelos, além de permitir a execução eficiente em diferentes dispositivos. O TensorFlow oferece suporte nativo para GPU e TPU, essenciais para acelerar o treino de modelos de *deep learning*. Uma das características mais poderosas do TensorFlow é o seu ecossistema abrangente, que inclui várias bibliotecas auxiliares, como *TensorFlow Lite* para dispositivos móveis,

TensorFlow Serving para a implementação de modelos em produção, e *TensorBoard* para a visualização de métricas de treino. Este ecossistema extenso permite que o TensorFlow seja uma escolha versátil tanto para pesquisa quanto para aplicações de produção em larga escala (Abadi et al., s.d.).

3.5.5 *Keras*

O *Keras* foi originalmente desenvolvido como uma API de alto nível para redes neuronais, concebida para ser modular, extensível e fácil de usar. Em 2017, foi integrada ao TensorFlow, tornando-se a API padrão para a criação de modelos nesta plataforma. A simplicidade de Keras contrasta com a complexidade inerente de frameworks como TensorFlow, oferecendo uma interface amigável que permite a criação rápida de protótipos de modelos, sem sacrificar a flexibilidade necessária para o ajuste fino de redes profundas (Chollet et al., 2015). Keras abstrai a complexidade do TensorFlow, permitindo que os programadores definam e treinem redes neuronais em apenas algumas linhas de código. Esta simplicidade torna o Keras uma excelente escolha para iniciantes, mas também se mostrou suficientemente poderosa para suportar projetos complexos de *deep learning* em ambientes de produção. A API de Keras é completamente modular, permitindo a fácil combinação e reutilização de camadas, funções de perda, otimizadores e métricas. Além disso, a integração nativa com o TensorFlow faz com que o Keras seja uma ferramenta essencial para este projeto, já que combina simplicidade e poder computacional numa única solução.

3.5.6 *Scikit-Learn*

O *Scikit-Learn* é uma biblioteca de *machine learning* em Python que se destaca pela sua simplicidade e eficiência. Desenvolvida a partir da biblioteca SciPy, o Scikit-Learn oferece uma vasta gama de algoritmos de *machine learning*, incluindo regressão, classificação, *clustering*, e redução de dimensionalidade. Além disso, fornece ferramentas para a validação cruzada, seleção de hiperparâmetros e *pipelines* de processamento de dados (Pedregosa FABIANPEDREGOSA et al., 2011). Embora o Scikit-Learn não seja especificamente projetado para *deep learning*, é amplamente utilizado para tarefas de *machine learning* tradicional e é frequentemente integrado com frameworks como TensorFlow e Keras para o pré-processamento de dados e para a avaliação de modelos. A simplicidade da sua API, combinada com a sua

documentação detalhada, faz com que seja uma escolha popular tanto em ambientes académicos quanto industriais (Müller e Guido, s.d.).

3.5.7 *Firebase*

O *Firebase* é uma plataforma de desenvolvimento de aplicações criada pela Google, que oferece uma ampla gama de ferramentas e serviços, facilitando o desenvolvimento, a gestão e a análise de aplicações móveis e web. Desde o seu lançamento em 2011, inicialmente como uma solução de banco de dados em tempo real, o Firebase evoluiu para um ecossistema completo, abrangendo funcionalidades como hospedagem, autenticação de utilizadores, notificações *push*, entre outros serviços. Esta plataforma é composta por diversos módulos, cada um voltado para atender diferentes necessidades de desenvolvimento. No contexto deste projeto, destacam-se dois componentes principais: o *Firestore* e o *Firebase Hosting*. O *Firestore*, similar ao *Realtime Database*, oferece uma estrutura de dados mais flexível e suporta consultas avançadas, sendo ideal para aplicações que necessitam de um banco de dados escalável. Já o Firebase Hosting é um serviço de hospedagem estática, caracterizado pela sua rapidez e segurança. Neste projeto, o Firebase desempenhou um papel crucial, servindo tanto como solução de hospedagem quanto como banco de dados. O Firebase Hosting foi escolhido pela sua simplicidade e eficiência na implementação e gestão do website. Através dele, foi possível garantir uma entrega ágil e segura dos conteúdos, com suporte integrado ao protocolo SSL (Secure Sockets Layer). Por sua vez, o *Firestore* foi selecionado como a principal base de dados da aplicação, devido à sua flexibilidade e à facilidade de integração com o front-end da aplicação web. A escolha pelo *Firestore*, em detrimento do Realtime Database, deveu-se à sua estrutura mais organizada, além da sua capacidade de escalar conforme o crescimento do projeto.

3.5.8 *Outras Ferramentas e Bibliotecas*

Além de TensorFlow, Keras e Scikit-Learn, foram utilizadas outras bibliotecas e frameworks relevantes no domínio de *deep learning* e *machine learning*:

- **PyTorch**: O *PyTorch* é um framework de *deep learning* muito popular, conhecido pela sua flexibilidade e capacidade de realizar computação dinâmica. É amplamente utilizado em pesquisa e prototipagem rápida, mas também

suporta implementações em larga escala. A simplicidade na depuração, graças à execução dinâmica, é um dos principais diferenciais relativamente ao TensorFlow (Paszke et al., 2019).

- **Caffe**: Originalmente desenvolvido pela Berkeley Vision and Learning Center (BVLC), o *Caffe* é um framework de *deep learning* voltado para a velocidade e a eficiência. É especialmente eficaz para tarefas de visão computacional e foi uma das primeiras plataformas a oferecer suporte robusto para GPUs. No entanto, o Caffe tem sido gradualmente superado por TensorFlow e PyTorch devido à sua menor flexibilidade (Jia et al., 2014).
- **Theano**: O *Theano* é uma biblioteca Python que permite a definição, otimização e avaliação de expressões matemáticas envolvendo tensores multidimensionais. Foi uma das primeiras bibliotecas de *Deep Learning* amplamente adotadas, mas o seu desenvolvimento foi descontinuado em 2017. No entanto, o Theano foi fundamental para o desenvolvimento de outras bibliotecas como Keras e Lasagne (Bergstra et al., 2010).
- **MXNet**: Apoiada pela Apache Software Foundation, o *MXNet* é uma biblioteca de *deep learning* que combina a eficiência do Caffe com a flexibilidade do PyTorch. Oferece suporte para uma vasta gama de linguagens de programação, incluindo Python, Scala e Julia, e é particularmente popular na Ásia (Chen et al., 2015).
- **SciPy**: O *SciPy* é uma biblioteca científica em Python que fornece ferramentas para computação técnica e científica, incluindo otimização, álgebra linear, integração, e estatísticas. No projeto, o SciPy foi utilizado para realizar cálculos avançados, necessários durante o processamento de dados e treino de modelos.
- **Librosa**: O *Librosa* é uma biblioteca Python especializada em análise de áudio e música. Oferece funções para carregar, processar e analisar sinais de áudio, sendo muito utilizada em tarefas de processamento de linguagem natural e análise de dados de som. Nesta aplicação, o *Librosa* foi utilizado para manipular e extrair características de áudio, essenciais para o treino de modelos que envolvem processamento de áudio (Mcfee et al., s.d.).
- **PyAudio**: O *PyAudio* é uma biblioteca em Python que permite a captura e reprodução de áudio. Ela facilita a interface com o sistema de som do computador, sendo utilizada para gravar ou reproduzir áudio diretamente da aplicação.

- **SoundDevice**: O *SoundDevice* é outra biblioteca que permite a manipulação de áudio, com uma API simples e eficiente para gravação e reprodução de som. Ambas as bibliotecas foram utilizadas para realizar operações de entrada e saída de áudio durante o desenvolvimento e testes dos modelos.
- **Tkinter**: O *Tkinter* é a biblioteca padrão do Python para a criação de interfaces gráficas (GUI). Nesta aplicação, o *Tkinter* foi utilizado para construir interfaces simples que facilitam a interação do utilizador com a aplicação. A instalação foi feita através do Conda, integrando essa funcionalidade ao ambiente de desenvolvimento.
- **Pydub**: O *Pydub* é uma biblioteca de Python usada para manipulação de áudio, facilitando tarefas como conversão entre formatos, corte, junção e aplicação de efeitos em arquivos de áudio. Ela suporta formatos populares como MP3, WAV e OGG, e permite operações simples com uma sintaxe intuitiva. Além disso, o *Pydub* depende do *FFmpeg* ou *libav* para manipulação de arquivos de áudio, o que amplia a sua capacidade de lidar com uma grande variedade de formatos e codificações.
- **Imgaug**: A biblioteca *imgaug* é uma ferramenta poderosa e flexível para aplicar técnicas de aumento de dados (*data augmentation*) em imagens. Amplamente utilizada em tarefas de visão computacional, a *imgaug* permite gerar variações artificiais nos dados de treino, como rotações, cortes, alterações de brilho, entre outras transformações. Isto ajuda a aumentar a robustez dos modelos de machine learning ao expô-los a diferentes versões dos mesmos dados, reduzindo o risco de sobreajuste e melhorando a generalização em dados novos. A biblioteca destaca-se pela sua simplicidade e pela capacidade de criar pipelines complexos de *augmentation* de maneira eficiente.

3.5.9 *Comparação e escolha das ferramentas utilizadas*

A escolha das ferramentas e bibliotecas para um projeto de *machine learning* e *deep learning* depende de uma série de fatores, incluindo a natureza do problema a ser resolvido, o ambiente de desenvolvimento, os recursos computacionais disponíveis e o nível de familiaridade da equipa com as tecnologias. No contexto deste projeto, a decisão de utilizar TensorFlow, Keras e Scikit-Learn foi baseada em critérios claros de desempenho, flexibilidade e eficiência.

O *TensorFlow* foi escolhido pela sua robustez e pelo seu ecossistema abrangente, que facilita a implementação de soluções tanto em fase de desenvolvimento quanto em ambientes de produção. A sua capacidade de escalabilidade, com o suporte a GPU e TPU, torna-o ideal para o treino de modelos complexos de *deep learning*. A integração nativa com o *Keras* permitiu uma maior facilidade na prototipagem rápida, sem sacrificar o poder de processamento subjacente oferecido pelo TensorFlow.

O *Keras*, devido à sua simplicidade e modularidade, provou ser a ferramenta ideal para o desenvolvimento rápido de protótipos e ajustes finos nos modelos de redes neuronais. A sua interface intuitiva e a forte integração com o TensorFlow foram decisivos para a sua escolha neste projeto, reduzindo a complexidade no desenvolvimento e permitindo uma abordagem mais iterativa e ágil.

Por outro lado, o *Scikit-Learn* destacou-se pela sua simplicidade e eficiência em tarefas de *machine learning* tradicional. A sua utilização foi essencial no pré-processamento dos dados e na validação dos modelos, onde a sua vasta gama de algoritmos e ferramentas de avaliação facilitaram a obtenção de resultados precisos de forma rápida.

A escolha por frameworks como *PyTorch* e *Caffe* foi considerada, mas o TensorFlow demonstrou maior adequação ao contexto deste projeto, principalmente pela sua capacidade de integração com outros serviços da Google, como o Firebase, e pela vasta documentação e suporte da comunidade. Assim, a decisão por utilizar TensorFlow, Keras e Scikit-Learn foi orientada por uma análise das necessidades específicas do projeto e pela capacidade dessas ferramentas em oferecer um equilíbrio entre flexibilidade, desempenho e facilidade de uso.

3.6 AVALIAÇÃO DE MODELOS

O objetivo principal desta fase é garantir que o modelo criado não apenas apresente um bom desempenho sobre o conjunto de dados de treino, mas também tenha a capacidade de generalizar bem para novos dados, ou seja, dados que o modelo não viu durante o processo de treino. Um modelo de qualidade deve ser capaz de equilibrar precisão e generalização, evitando o sobre ajustamento (*overfitting*) ao conjunto de treino e garantindo um bom desempenho em cenários reais.

Para avaliar a eficácia de um modelo, utilizam-se diferentes técnicas e métricas. Entre as técnicas mais comuns estão os métodos de reamostragem, que dividem o conjunto de dados em subconjuntos para testar o desempenho do modelo em

diferentes partes dos dados. Este processo permite estimar o desempenho real do modelo ao simular a sua aplicação em diferentes situações. Além disso, métricas de desempenho, como a *accuracy*, *precision*, *recall*, *F1-score*, entre outras, ajudam a quantificar a eficácia do modelo em resolver o problema proposto.

3.6.1 *Holdout*

Entre os diversos métodos de avaliação, o método *Holdout* foi o escolhido para este projeto devido à sua simplicidade e eficiência computacional. Este método consiste em dividir o conjunto de dados em dois subconjuntos: um conjunto de treino e um conjunto de teste. O conjunto de treino é utilizado para construir o modelo, enquanto o conjunto de teste é utilizado para avaliar o desempenho do modelo em dados não vistos. Neste projeto o conjunto de treino vai representar 80% do total dos dados e o conjunto de teste vai representar os 20% restantes dos dados.

O método Holdout apresenta algumas vantagens claras. Por ser um processo direto, ele é rápido e computacionalmente eficiente, já que o modelo é treinado e testado apenas uma vez. Esta simplicidade torna-o particularmente útil quando se trabalha com conjuntos de dados relativamente pequenos ou quando o tempo de computação é uma consideração importante. Contudo, o método também tem as suas limitações. Como o desempenho do modelo depende fortemente de como o conjunto de dados foi dividido, existe o risco de que a divisão não seja representativa do comportamento geral do modelo. Se a divisão dos dados não for feita de maneira aleatória e equilibrada, o modelo pode apresentar resultados otimistas ou pessimistas, subestimando ou superestimando o seu verdadeiro desempenho.

3.6.2 *Métricas de Avaliação*

No domínio de *Machine Learning* e Redes Neurais Artificiais, a avaliação e validação dos modelos são passos cruciais para assegurar o desempenho e a fiabilidade das previsões. Diferentes métricas de avaliação fornecem uma compreensão detalhada sobre como um modelo se comporta relativamente a diferentes aspetos dos dados, como a precisão, a capacidade de deteção de classes minoritárias, e a robustez geral. Neste capítulo, exploramos detalhadamente várias métricas de avaliação comuns, nomeadamente a Matriz de Confusão, *Accuracy*, *Precision*, *Recall*, *F1 Score*, *Mean Validation Loss* e *Mean Validation Accuracy*, que são frequentemente utilizadas para avaliar o desempenho de modelos de classificação e redes neurais.

3.6.2.1 *Matriz Confusão*

A Matriz de Confusão é uma ferramenta fundamental para a avaliação do desempenho de algoritmos de classificação. Trata-se de uma tabela que permite visualizar as predições feitas por um modelo relativamente aos valores reais dos dados de teste. A matriz é composta por quatro elementos principais:

- Verdadeiros Positivos (TP): Casos em que o modelo previu corretamente a classe positiva.
- Falsos Positivos (FP): Casos em que o modelo previu a classe positiva erroneamente.
- Falsos Negativos (FN): Casos em que o modelo não conseguiu prever a classe positiva.
- Verdadeiros Negativos (TN): Casos em que o modelo previu corretamente a classe negativa.

A análise da Matriz de Confusão permite identificar onde o modelo está a falhar, por exemplo, se está a gerar muitos falsos positivos ou falsos negativos, o que é crítico em áreas como a deteção de fraudes ou diagnósticos médicos (Powers e Ailab, s.d.).

3.6.2.2 *Accuracy*

A *Accuracy* (ou Precisão Global) é uma das métricas mais simples e frequentemente utilizadas para avaliar o desempenho de um modelo. É calculada como a razão entre o número de predições corretas (soma de TP e TN) e o número total de predições:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

Embora seja uma métrica intuitiva, a *Accuracy* pode ser enganadora em cenários de classes desbalanceadas. Por exemplo, num conjunto de dados onde 95% das amostras pertencem à classe negativa, um modelo que prevê sempre a classe negativa terá uma *Accuracy* de 95%, mas será ineficaz para identificar a classe positiva (Sokolova e Lapalme, 2009).

3.6.2.3 *Precision*

A *Precision* (ou Precisão) é uma métrica que avalia a capacidade do modelo em prever a classe positiva de forma correta, minimizando os falsos positivos. É calculada como a proporção de predições corretas da classe positiva em relação a todas as predições feitas para essa classe:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2}$$

Esta métrica é especialmente útil em cenários onde o custo de um falso positivo é alto, como diagnósticos médicos ou filtragem de '*spam*', onde é preferível ter menos falsos positivos (Davis e Goadrich, s.d.).

3.6.2.4 *Recall*

O *Recall* (ou Sensibilidade/Revocação) mede a capacidade do modelo em identificar corretamente todas as instâncias da classe positiva, minimizando os falsos negativos. É calculado como a proporção de predições corretas da classe positiva relativamente ao total de verdadeiros casos positivos:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3}$$

Esta métrica é particularmente importante em aplicações onde a não deteção de um caso positivo pode ter consequências severas, como na deteção de doenças graves ou em sistemas de segurança (Manning et al., 2009).

3.6.2.5 *F1 Score*

O *F1 Score* é a média harmónica entre a *Precision* e o *Recall*, proporcionando uma métrica única que equilibra os dois fatores. Esta métrica é especialmente útil quando se pretende alcançar um equilíbrio entre a capacidade de prever corretamente a classe positiva e minimizar tanto os falsos positivos quanto os falsos negativos:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

O *F1 Score* é preferível à *Accuracy* em situações de desequilíbrio de classes, uma vez que proporciona uma visão mais equilibrada do desempenho do modelo (Van Rijsbergen, s.d.).

3.6.2.6 *Mean Validation Loss*

A *Mean Validation Loss* é uma métrica utilizada para avaliar o desempenho do modelo durante o processo de treino, mais especificamente em relação ao conjunto de validação. A Loss, ou perda, refere-se à medida do erro entre as predições do modelo e os valores reais dos dados. Durante o treino de redes neurais, a função de perda é minimizada mediante técnicas como o gradiente descendente. A *Mean Validation Loss* é a média das perdas calculadas em cada iteração sobre o conjunto de validação e é utilizada para monitorizar se o modelo está a melhorar ao longo do tempo. Um valor crescente de Validation Loss pode indicar problemas como sobreajuste, onde o modelo está a aprender demasiado bem os dados de treino, mas não generaliza adequadamente para novos dados (Goodfellow et al., 2016).

3.6.2.7 *Mean Validation Accuracy*

A Mean Validation *Accuracy* é semelhante à *Accuracy*, mas é medida especificamente no conjunto de validação durante o processo de treino do modelo. Esta métrica indica a proporção de predições corretas feitas pelo modelo no conjunto de validação e é utilizada para avaliar a capacidade de generalização do modelo. Enquanto a *Mean Validation Loss* dá uma ideia do erro absoluto, a Mean Validation *Accuracy* proporciona uma noção de quão bem o modelo está a prever corretamente as classes no conjunto de validação. Juntas, estas métricas permitem aos investigadores monitorizar e ajustar o treino de modelos, evitando problemas como o sobreajuste e garantindo que o modelo mantém um bom desempenho em dados não vistos (Bishop M. Christopher, 2006).

4

DESENVOLVIMENTO

Neste capítulo, detalhamos o desenvolvimento do nosso projeto sobre a aplicação de *Deep Learning* para reconhecimento de fonemas concatenados. Desde a obtenção de dados de áudio, passando pelo seu tratamento e a aplicação de diversas ferramentas e metodologias para alcançar os resultados pretendidos.

4.1 RECOLHA DE DADOS

A qualidade e quantidade dos dados são cruciais no desenvolvimento de modelos de *Deep Learning*, pois os dados servem como a base sobre a qual os modelos são treinados. Dados ricos e diversificados permitem que os modelos aprendam padrões complexos. Além disso, dados limpos e bem anotados garantem que o treino seja preciso e que os resultados obtidos sejam fiáveis. Sem dados adequados, mesmo os algoritmos mais sofisticados podem falhar em alcançar um desempenho satisfatório. Para tal, a criação de um repositório robusto de dados de áudio foi um passo crucial para garantir a eficácia dos modelos. Para facilitar a recolha de dados, foi desenvolvido um website, oferecendo uma interface intuitiva e de fácil utilização para que qualquer pessoa possa voluntariamente participar nos dados a serem usados no desenvolvimento deste projeto.

O website, intitulado "Deep Learning para Fonemas Concatenados", sobe o endereço `https://fonemasweb.web.app`, foi desenvolvido com o propósito de recolher gravações de áudio individuais de voz humana articulando números de 0 a 9 em português, permitindo a criação de um 'corpus' de dados significativo. Foi criado uma plataforma que permite aos utilizadores gravar diretamente os seus fonemas através do seu navegador de Internet, sem a necessidade de software adicional. Os utilizadores podem gravar as suas vozes de forma anónima, apenas especificando o seu género (masculino ou feminino).

Esta plataforma foi concebida para ser intuitiva e fácil de utilizar, incentivando a participação dos utilizadores. A estrutura foi desenvolvida em HTML como base do conteúdo e JavaScript para a interatividade e funcionalidade. O '*design*' visual

Figura 2: Imagem retirada da página inicial do website `https://fonemasweb.web.app`

foi aprimorado utilizando CSS e a framework Bootstrap para melhor resposta e consistência do layout. O código referente a este desenvolvimento está na listagem 8 do apêndice K.

O website contém três páginas principais: Início, Gravação de Áudio e Contactos.

- Início: Esta página apresenta uma introdução do projeto, do autor e a instituição pela qual o website foi criado, como demonstra a Figura 2.
- Gravação de Áudio: A página de gravação permite aos utilizadores gravar números de 0 a 9. Cada gravação é feita ao clicar num botão numerado, ativando a gravação de áudio por 2 segundos. Após a gravação, os utilizadores podem ouvir e validar os áudios antes de submetê-los, como demonstra a Figura 3.
- Contactos: Esta página fornece um formulário para os utilizadores entrarem em contacto, caso tenham dúvidas ou necessitem de suporte, como demonstra a Figura 4.

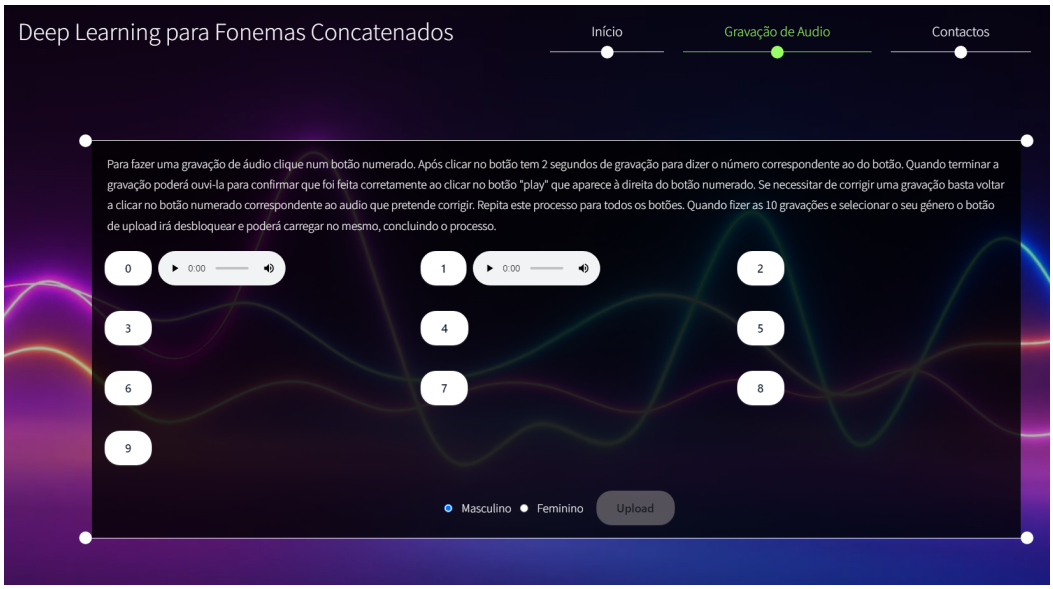


Figura 3: Imagem retirada da página de gravação de áudio do website https://fonemasweb.web.app

A gravação de áudio foi implementada utilizando a *API de Media Devices* do navegador, que permite o acesso ao microfone do dispositivo do utilizador. Ao clicar num botão numerado, a gravação é iniciada e, após 2 segundos, o áudio é automaticamente armazenado localmente para revisão. Os utilizadores podem reproduzir as gravações para garantir que foram realizadas corretamente antes de prosseguir para o '*upload*'.

Para garantir uma hospedagem fiável e segura, optou-se por utilizar o Firebase, uma plataforma robusta que oferece uma série de serviços para o desenvolvimento web. O Firebase foi escolhido devido à sua capacidade de armazenamento em tempo real e à facilidade de integração com tecnologias web. O armazenamento foi organizado por índice e género, garantindo que os dados recolhidos sejam facilmente acessíveis e organizados.

Figura 4: Imagem retirada da página de contactos do website https://fonemasweb.web.app

A privacidade dos utilizadores foi uma prioridade no desenvolvimento do website. As gravações são anónimas, armazenando apenas a informação do género. Nenhum dado pessoal é recolhido durante o processo de gravação. O formulário de contacto, embora colete nome e endereço eletrónico, é opcional. Durante o desenvolvimento houve a necessidade de garantir a qualidade das gravações e a compatibilidade com diferentes dispositivos e navegadores.

A recolha de dados teve a participação de 22 voluntários (13 masculinos e 9 femininos) dando um total de 220 amostras de áudio. O sucesso desta recolha de dados destaca a importância de ferramentas bem desenvolvidas na pesquisa e desenvolvimento de sistemas de inteligência artificial, permitindo também o contínuo enriquecimento da base de dados para futuras melhorias do modelo.

4.2 PREPARAÇÃO DOS DADOS

Após a recolha dos dados, procedeu-se à preparação dos mesmos para o treino dos modelos. A preparação dos dados foi dividida em cinco etapas principais: conversão de formato, filtragem de ruído, construção de imagens de mel-espectrograma, seleção de imagens representantes de fonemas e aumento de dados (augmentation). A seguir, detalhamos cada uma dessas etapas.

4.2.1 *Conversão de Formato de Áudio*

A primeira etapa envolveu a conversão dos arquivos de áudio capturados do website. Os áudios estavam originalmente no formato WebM (*.webm*), que embora seja eficiente para *'streaming'* e armazenamento, não é ideal para processamento em Python devido à falta de suporte robusto em bibliotecas populares como *librosa*. Portanto, foi necessário converter estes arquivos para o formato WAV(*.wav*), que oferece melhor compatibilidade e desempenho para as operações de processamento de áudio. Para realizar esta conversão, foi utilizada a biblioteca Pydub, que oferece uma interface simples para manipulação de arquivos de áudio, como podemos ver no código apresentado na Listagem 1 do Apêndice A.

4.2.2 *Aplicação do Filtro de Wiener*

Após a conversão dos áudios, foi necessário melhorar a qualidade através da filtragem de ruído. Para isso, aplicámos o filtro de Wiener, uma técnica eficaz para a redução de ruído em sinais que apresentam distorções. No código implementado apresentado na Listagem 2 do Apêndice A, utilizou-se a biblioteca *librosa* para carregar os arquivos de áudio em formato WAV e o módulo *Wiener* da biblioteca *scipy.signal* para aplicar o filtro de Wiener. A escolha destas bibliotecas deve-se à sua robustez e eficiência no processamento de sinais de áudio. Este processo garantiu que os áudios utilizados para o treino estivessem limpos e adequados, reduzindo a influência de ruídos indesejados nos resultados do modelo.

4.2.3 *Construção de imagens de mel-espectrograma*

Após a filtragem dos áudios, a etapa seguinte consistiu na construção de imagens de mel-espectrograma a partir dos sinais de áudio. Para esta tarefa, utilizámos a biblioteca *librosa*, que oferece funcionalidades avançadas para o processamento de sinais de áudio, em conjunto com o *Matplotlib* para a visualização dos mel-espectrogramas. O código implementado, conforme apresentado na Listagem 3 no Apêndice A, carrega os arquivos de áudio, computa o mel-espectrograma e converte-o para uma escala logarítmica de decibéis, o que facilita a visualização de variações dinâmicas no espectro de frequências. Após o cálculo do mel-espectrograma, este é segmentado em janelas de tamanho fixo, com deslocamentos sobrepostos, gerando

uma série de imagens que representam diferentes partes do áudio original. Cada uma destas imagens foi então guardada em formato PNG, numa pasta dedicada. Este processo automatizado permitiu a criação eficiente de um grande conjunto de dados visuais.

A criação de mel-espectrogramas requer a definição de parâmetros-chave que afetam diretamente a forma como o áudio é segmentado em janelas temporais e processado. Os parâmetros mais relevantes neste contexto são o *SHIFT*, o *WINDOW_SIZE* e o *HOP_LENGTH*. A seguir, vamos explorar detalhadamente o significado e o impacto de cada um deles.

O parâmetro *SHIFT* refere-se ao deslocamento aplicado ao longo do eixo do tempo para definir a próxima janela de áudio a ser processada. Este deslocamento indica quantas amostras de áudio a janela se move antes de ser calculado um novo mel-espectrograma. Este parâmetro controla a sobreposição das janelas consecutivas. Por exemplo, se o valor de *SHIFT* for pequeno, haverá uma maior sobreposição entre as janelas, o que pode permitir uma maior resolução temporal, mas também aumentará a redundância entre as imagens geradas. Como exemplo, podemos ver a diferença entre imagens geradas com diferentes *SHIFT* na figura 5.

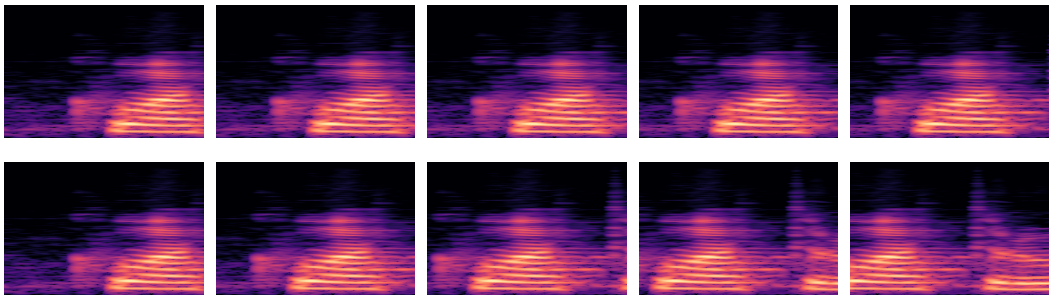


Figura 5: Comparação de sequências de imagens de mel-espectrogramas com diferentes *SHIFT*. Primeira linha com um *SHIFT* de 100 e a segunda linha com *SHIFT* de 300.

O *WINDOW_SIZE* define o tamanho, em amostras, da janela de áudio utilizada para calcular cada mel-espectrograma. Uma janela maior permite capturar mais informações ao longo do tempo, mas também reduz a resolução temporal, já que eventos sonoros rápidos podem ser diluídos dentro de uma janela muito longa. O equilíbrio entre o tamanho da janela e a resolução temporal é crucial. A fórmula que relaciona o tamanho da janela com o tempo em segundos é dada por:

$$T_{window} = \frac{WINDOW\_SIZE}{SR} \tag{5}$$

onde T_{window} é o tempo correspondente ao tamanho da janela em segundos e SR é a taxa de amostragem (*sample rate*) do áudio. Como exemplo, podemos ver a diferença entre imagens geradas com diferentes *WINDOW_SIZE* na figura 6.

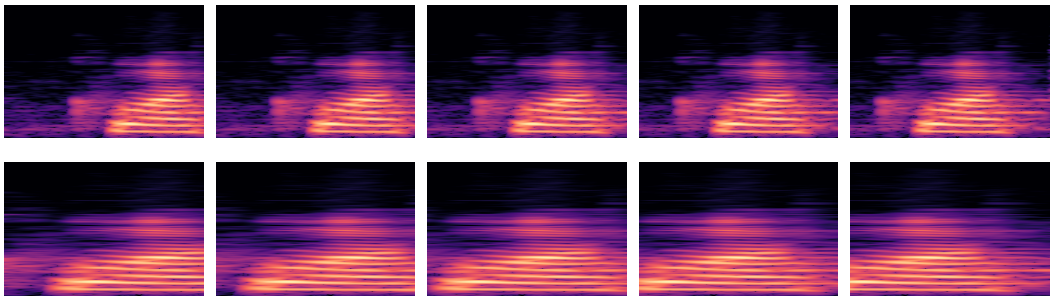


Figura 6: Comparação de sequências de imagens de mel-espectrogramas com diferentes *WINDOW_SIZE*. Primeira linha com um *WINDOW_SIZE* de 2000 e a segunda linha com *WINDOW_SIZE* de 1000.

O *HOP_LENGTH* é o parâmetro que determina o número de amostras que a janela se desloca ao calcular os coeficientes do mel-espectrograma ao longo do tempo. Este parâmetro controla o grau de sobreposição entre as janelas dentro do cálculo do mel-espectrograma, similar ao conceito de *SHIFT*, mas numa escala mais fina. Valores mais pequenos de *HOP_LENGTH* resultam numa maior densidade de coeficientes temporais, o que aumenta a resolução temporal do espectrograma final, mas também aumenta a complexidade computacional. A fórmula para calcular o tempo correspondente ao *HOP_LENGTH* é:

$$T_{hop} = \frac{HOP\_LENGTH}{SR} \tag{6}$$

onde T_{hop} é o intervalo de tempo entre dois coeficientes consecutivos no mel-espectrograma. Este parâmetro afeta diretamente a suavidade temporal do espectrograma e a quantidade de informação visual gerada. Como exemplo, podemos ver a diferença entre imagens geradas com diferentes *HOP_LENGTH* na figura 7.

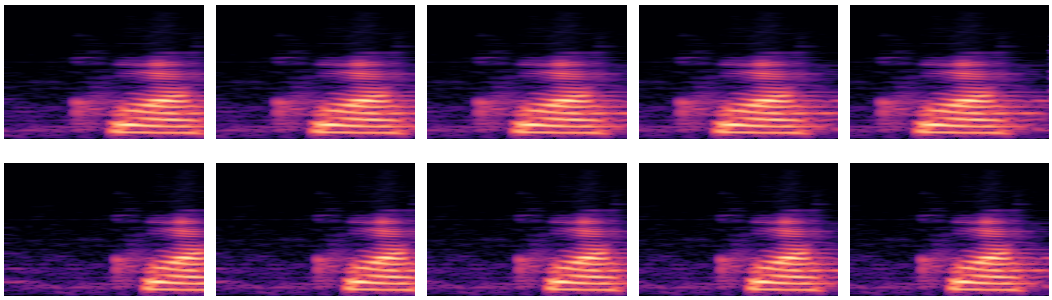


Figura 7: Comparação de sequências de imagens de mel-espectrogramas com diferentes *HOP_LENGTH*. Primeira linha com um *HOP_LENGTH* de 8 e a segunda linha com *HOP_LENGTH* de 10.

4.2.4 *Seleção de imagens representantes de fonemas*

Após a geração das imagens de mel-espectrograma, foi necessário selecionar manualmente imagens representativas de fonemas de maneira a criar as diferentes classes que o modelo irá utilizar. Esta etapa visa garantir que o modelo receba exemplos de alta qualidade de cada fonema específico. A seleção manual permite identificar as representações visuais mais claras e adequadas, eliminando imagens que possam conter ruídos ou distorções.

Para facilitar o processo de seleção, foi desenvolvida uma interface gráfica simples utilizando a biblioteca *Tkinter*. Esta interface, representada como exemplo na Figura 8, permite visualizar as imagens de cada amostra de áudio e selecionar aquelas que melhor representam os fonemas desejados. O código implementado, conforme mostrado na Listagem 4 no Apêndice A, carrega as imagens geradas, apresentando-as num formato compacto para revisão manual. Após a seleção, a imagem escolhida é copiada para uma pasta de destino, onde será usada para o treino e validação do modelo.

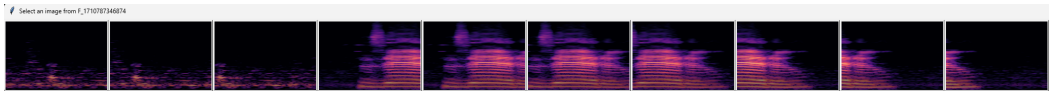


Figura 8: Exemplo de interface gráfica para seleção de imagens representativas dos fonemas da palavra "zero". Neste exemplo as imagens selecionadas seriam a quarta imagem para o fonema 'ze' e a nona imagem para o fonema 'ro'.

Este processo, embora manual, é fundamental para garantir a qualidade dos dados que serão utilizados no treino. A seleção cuidadosa de imagens representativas de fonemas ajuda a minimizar a introdução de erros no processo de aprendizagem, contribuindo para o aumento da precisão do modelo nos testes futuros.

4.2.5 *Aplicação de Augmentation nos Dados*

A última etapa do processo de preparação dos dados foi a aplicação de técnicas de aumento de dados (data augmentation). Esta etapa é fundamental para aumentar a robustez do modelo, criando variações nos dados de treino já obtidos sem a necessidade de coletar novos dados. Utilizando a biblioteca *imgaug*, foi possível aplicar diversas transformações nos dados de áudio, simulando variações que o modelo pode encontrar em produção. O código implementado encontra-se apresentado na Listagem 5 do Apêndice A.

4.3 IMPLEMENTAÇÃO DOS MODELOS

4.3.0.1 *Dataset Utilizado*

Os modelos desenvolvidos procuram classificar fonemas que compõe palavras em português de Portugal de zero a nove. Após a análise dos fonemas e sílabas da fonética portuguesa fornecida pelo Portal da Língua Portuguesa (www.portaldalinguaportuguesa.org) podemos decompor os fonemas de cada palavra da maneira expressa na Tabela 1.

PALAVRA	NÚMERO CARDINAL	SÍLABAS	FONÉTICA
zero	0	ze.ro	z'ɛ.ɻu
um	1	um	'u
dois	2	dois	d'ojʃ
três	3	três	tɻ'ɛ ʃ
quatro	4	qua · tro	kw'aɻ.tu
cinco	5	cin · co	s'i .ku
seis	6	seis	s'ɒj ʃ
sete	7	se · te	s'ɛ.tɨ
oito	8	oi · to	'oj.tu
nove	9	no · ve	n'ɔ.vɨ

Tabela 1: Silabas e fonética de palavras

Como podemos observar, existe uma semelhança nos fonemas finais das palavras 'dois', 'três' e 'seis'. Por este motivo, ao definir o número de classes para o *dataset*, foi decidido criar inicialmente dois *datasets* distintos. O primeiro, com 19 classes, inclui todos os fonemas de cada palavra de forma independente; o segundo, com 17

classes, agrupa os fonemas finais das palavras 'dois', 'três' e 'seis' numa única classe, devido à sua semelhança fonética. Após a análise dos resultados obtidos com estes dois *datasets*, foi criado um terceiro *dataset*, com 18 classes, que é semelhante ao de 17 classes, mas com a inclusão de uma classe adicional que representa uma categoria desconhecida ('unknown'). Esta classe foi adicionada com o intuito de permitir que o modelo identifique momentos de silêncio ou de ruído numa amostra de áudio, melhorando assim a robustez do sistema. Para ampliar ainda mais o espectro de testes e avaliar o desempenho do modelo sob diferentes condições, foram geradas duas versões de cada *dataset*: uma com *augmentation* e outra sem *augmentation*. Esta abordagem permitiu explorar o impacto da diversificação dos dados no processo de treino do modelo, contribuindo para uma análise mais abrangente dos resultados.

4.3.1 *Modelos Utilizados*

Neste abordaremos a utilização de diferentes arquiteturas de redes de *deep learning*, todas baseadas em modelos pré-treinados disponíveis na biblioteca *TensorFlow Keras*, uma das mais reconhecidas e amplamente utilizadas para o desenvolvimento de aplicações de aprendizagem automática e redes neurais. Para garantir uma maior diversidade nos experimentos realizados, foram exploradas tanto versões dos modelos com pré-treino utilizando o dataset "ImageNet", que fornece uma base sólida de reconhecimento de padrões, como versões sem qualquer pré-treino, permitindo uma avaliação mais abrangente do desempenho em diferentes cenários. Estes modelos, conhecidos pela sua eficácia e pelo papel fundamental no avanço da classificação de imagens, serão discutidos em mais detalhe nos próximos capítulos, onde exploraremos as suas respetivas arquiteturas.

4.3.1.1 *Resnet50*

O ResNet50 foi um dos modelos mais usados, a sua arquitetura foi desenvolvida para facilitar o treino de redes muito profundas, superando o problema do desaparecimento do gradiente *(Vanishing gradient problem)*. O modelo faz uso de blocos residuais, onde a entrada original é somada à saída de uma série de camadas convolucionais, permitindo que o modelo mantenha características importantes ao longo de várias camadas. A ResNet50 tem 50 camadas de profundidade e foi um dos primeiros modelos a mostrar que redes extremamente profundas podiam ser treinadas de forma eficaz, alcançando resultados de ponta.

4.3.1.2 *InceptionV3*

O InceptionV3 é uma versão melhorada da série de modelos Inception, também conhecida como GoogleNet. Esta arquitetura é conhecida por sua capacidade de capturar informações em várias escalas ao mesmo tempo, graças ao uso de módulos Inception, que combinam convoluções de diferentes tamanhos. O InceptionV3 introduz várias melhorias, incluindo técnicas como convoluções e rótulos auxiliares, que ajudam no treino e na generalização do modelo. Este modelo oferece um bom equilíbrio entre precisão e eficiência computacional, sendo amplamente utilizado em aplicações de visão por computador.

4.3.1.3 *VGG16*

A VGG16 é uma arquitetura de rede neural convolucional profunda que se destaca pela simplicidade da sua estrutura. Desenvolvida pela equipa do Visual Geometry Group da Universidade de Oxford, a VGG16 consiste em 16 camadas de profundidade, todas baseadas em convoluções 3x3 e camadas de pooling. Esta arquitetura tornou-se popular devido à sua robustez e à capacidade de transferir bem o aprendizado para outras tarefas, mesmo que seja relativamente pesada em termos de parâmetros e requisitos de memória.

4.3.1.4 *MobileNetV2*

O MobileNetV2 foi desenvolvido visando ser um modelo leve e eficiente, adequado para dispositivos móveis e aplicações com recursos limitados. A arquitetura faz uso de convoluções separáveis em profundidade e um novo bloco denominado "inverted residuals"com conexões lineares entre blocos. Esta combinação permite que o modelo seja compacto e eficiente, mantendo uma boa precisão, o que o torna ideal para aplicações que requerem baixo consumo de energia e processamento rápido.

4.3.1.5 *Xception*

A Xception é uma arquitetura que expande a ideia das convoluções separáveis em profundidade, popularizadas pelo Inception. O nome Xception significa "Extreme Inception", e o modelo é construído inteiramente em torno de convoluções separáveis em profundidade, substituindo os módulos Inception tradicionais por esta abordagem. Essa estrutura permite que a Xception capture melhor as características espaciais e

de profundidade das imagens, levando a um desempenho superior em várias tarefas de classificação de imagens, mantendo a eficiência computacional.

4.3.1.6 *DenseNet121*

A DenseNet121 é uma rede neural convolucional profunda conhecida pela sua conectividade densa, onde cada camada é diretamente conectada a todas as camadas subsequentes. Essa estrutura única permite que o gradiente flua mais facilmente pelas camadas, facilitando o treino de redes muito profundas e melhorando a reutilização das características aprendidas. A DenseNet121 é uma das variantes da família DenseNet, com 121 camadas, e é altamente eficiente em termos de parâmetros, enquanto oferece uma precisão competitiva em tarefas de classificação.

4.4 ESTRUTURA DOS TESTES

Nesta secção, abordaremos a estrutura e os procedimentos aplicados durante a fase de testes dos modelos, que foram cruciais para otimizar o desempenho e a eficiência dos sistemas de aprendizagem automática utilizados. Por meio de um conjunto de fases bem definidas, realizámos uma análise detalhada das diferentes variáveis que impactam o sucesso do processo de treino e a capacidade dos modelos em gerar previsões precisas. As fases de teste foram organizadas para abordar desde a avaliação inicial dos dados, passando pela seleção e adaptação de modelos pré-treinados, até a afinação fina dos mesmos. A seguir, detalharemos cada uma dessas fases para compreender como contribuíram para a construção de um sistema robusto e eficaz.

4.4.1 *Avaliação de Datasets*

Nesta fase, foram conduzidos testes para avaliar os *datasets* com variação no número de *batches* e *augmentation*, para identificar qual *dataset* apresentava os melhores resultados em termos de desempenho e eficiência. Este passo foi crucial para assegurar que os dados utilizados eram otimizados e adequados para o processo de treino dos modelos. A análise focou na qualidade e na quantidade de dados, assegurando que o conjunto escolhido fosse o mais apropriado para o treino e validação dos modelos.

O código apresentado na listagem 6 do apêndice B, foi desenvolvido para permitir a fácil alteração dos principais parâmetros de treino. Como exemplo, temos a figura

9, onde podemos observar que o número de *batches*, o número de classes e o *path* para a localização do *dataset* são definidos como constantes. Esta abordagem facilita a modificação destes valores, permitindo uma rápida adaptação do modelo a diferentes configurações de treino.

```
12  # Define constants
13  MODEL_NAME = "RESNET50_16_T"
14  CLASSES = 18
15  BATCHES = 16
16  DATABASE_DIR = "PhonemesManualUnknownAug"
17
18  # Get the current directory
19  current_directory = os.getcwd()
20
21  # Specify the path to the "Database" folder
22  folder_path = os.path.join(current_directory, "Database")
23  dataset_folder_path = os.path.join(folder_path, DATABASE_DIR)
24
25  # Define image dimensions
26  img_height = 400
27  img_width = 600
```

Figura 9: Código exemplo pertencente à listagem 6 que representa parâmetros de treino de 18 classes e 16 *batches*

4.4.2 *Testes com Modelos Pré-Treinados*

A segunda fase envolveu a realização de testes com diferentes modelos pré-treinados, para determinar qual modelo apresentava o melhor desempenho na classificação de imagens de espectrogramas que representam fonemas. Os modelos usados foram os modelos anteriormente mencionados na secção 4.3.1. A seleção do modelo base é um fator determinante no sucesso do processo de machine learning, pois influencia diretamente a capacidade do sistema de generalizar e realizar previsões precisas. As arquiteturas dos modelos pré-treinados foram carregadas utilizando a biblioteca TensorFlow Keras e, posteriormente, uma camada final densa com o número específico de classes do dataset foi adicionada, com uma função de ativação *softmax*, conforme ilustrado na Figura 10.

```
46  # Model Architecture
47  # Load the pre-trained ResNet50 model without the top layers
48  pretrained_model = tf.keras.applications.ResNet50(
49      include_top=False,
50      weights='imagenet',
51      input_shape=(img_height, img_width, 3),
52      pooling='avg',
53  )
54  pretrained_model.trainable = False
55
56  # Define the model
57  model = Sequential([
58      pretrained_model,
59      Dense(CLASSES, activation='softmax')
60  ])
```

Figura 10: Código exemplo pertencente à listagem 6 que representa a importação do modelo *ResNet50* com uma camada densa

4.4.3 *Personalização da Arquitetura e Ajuste de Hiperparâmetros*

Durante esta fase, foram realizados testes para personalizar a arquitetura dos modelos InceptionV3 e ResNet50. Alteraram-se diferentes estruturas e configurações de camadas visando otimizar o desempenho. Além disso, foi efetuado o ajuste dos hiperparâmetros do modelo, incluindo parâmetros como a taxa de aprendizagem (*learning rate*) e a aplicação de diferentes *optimizers*, para refinar o processo de treino. Também se modificou o nível de data *augmentation* aplicado aos dados, para avaliar o impacto da diversificação dos dados na robustez do modelo. Esta abordagem permitiu ajustar os modelos pré-treinados para as especificidades do conjunto de dados atual, através da modificação de partes da arquitetura ou da atualização de pesos em camadas adicionais. Na Tabela 2 podemos ver a descrição das alterações feitas para cada modelo nesta fase.

MODELO	DESCRIÇÃO
InceptionV3_1024_05	Camada densa de 1024 neurónios e uma camada Dropout com 50%
InceptionV3_64_02	Camada densa de 64 neurónios e uma camada Dropout com 20%
RESNET50_1024_05	Camada densa de 1024 neurónios e uma camada Dropout com 50%
RESNET50_64_02	Camada densa de 64 neurónios e uma camada Dropout com 20%
InceptionV3_8batch	O modelo foi treinado com 8 batches
InceptionV3_T	Camadas do modelo pré-treinado congeladas
RESNET50_16_T	Camadas do modelo pré-treinado congeladas
InceptionV3_SGD	*Optimizer* SGD
InceptionV3_EXPD	*Optimizer* com taxa de aprendizagem exponencial
InceptionV3_AUG_00	Sem *augmentation* adicional
InceptionV3_AUG_03	*Augmentation* com shifts of 30%

Tabela 2: Descrição dos modelos treinados na fase Personalização da Arquitetura e Ajuste de Hiperparâmetros

4.4.4 *Testes de Fine-Tuning*

A última fase de treinos focou-se na aplicação de técnicas de fine-tuning, um processo crucial para ajustar modelos pré-treinados a novas tarefas e conjuntos de dados. O fine-tuning envolve o descongelamento de camadas específicas da rede neural, que até então permaneciam inalteradas durante a fase inicial de treino. Estas camadas são então retreinadas, permitindo ao modelo adaptar-se melhor à tarefa em questão. Além disso, foi feito um ajuste cuidadoso da taxa de aprendizagem (learning rate), um parâmetro essencial que controla o ritmo com que o modelo atualiza os seus pesos durante o treino. Este ajuste é particularmente importante na fase de fine-tuning, pois um learning rate muito elevado pode levar a uma sobrescrita indesejada do conhecimento previamente adquirido, enquanto um learning rate demasiado baixo pode resultar em convergência lenta.

A combinação do descongelamento de camadas e da otimização do learning rate visa refinar a capacidade do modelo para se especializar no novo conjunto de dados, maximizando a sua precisão e eficácia na tarefa de classificação. Este processo permite ao modelo aproveitar o conhecimento adquirido durante o pré-treino em grandes bases de dados, como o ImageNet, ao mesmo tempo que ajusta as camadas mais profundas para reconhecer padrões específicos da nova tarefa. Como resultado, a utilização do fine-tuning pode proporcionar melhorias substanciais no desempenho

do modelo, tornando-o mais robusto e eficiente para lidar com as 'nuances' dos novos dados.

4.4.5 *Testes de aplicação*

Para testar o melhor modelo treinado, foi criado um ambiente de teste que permite gravar novos áudios e classificar os fonemas capturados. O objetivo desta aplicação é avaliar a eficácia do modelo no reconhecimento de fonemas de números de 0 a 9 em novas amostras de áudio. A interface foi projetada para ser simples e intuitiva, permitindo ao utilizador gravar a sua voz por meio de um microfone. Após a gravação, o áudio é analisado pelo modelo, que classifica os fonemas e apresenta os resultados de forma clara, permitindo uma avaliação imediata do seu desempenho. Cada número foi gravado duas vezes para testar a consistência do modelo, garantindo que ele mantém um desempenho estável mesmo com variações de tom, velocidade da fala e outros fatores.

O código utilizado neste teste encontra-se na listagem 7 do apêndice B. A aplicação começa por gravar o áudio captado pelo microfone do utilizador, que é armazenado num ficheiro. Em seguida, o áudio é convertido em mel-espectrogramas, que são posteriormente guardados como imagens. Essas imagens são depois analisadas por um modelo de deep learning, treinado para reconhecer e classificar fonemas. O modelo prevê a classe correspondente a cada espectrograma, identificando o fonema provável. As previsões são registadas num ficheiro de '*log*' e agrupadas para formar palavras ou números completos a partir dos fonemas detetados.

4.4.6 *Callbacks e Gráficos gerados*

Em adição, foram implementados 2 tipos *callbacks* no processo de treino dos modelos, `EarlyStopping` e `ModelCheckpoint`. A função `EarlyStopping` é utilizada para monitorizar a *loss* na validação (`val_loss`) e interromper o treino de forma automática se não houver melhorias após 10 *epochs*, prevenindo o *overfitting*. Já a função `ModelCheckpoint` permite guardar o modelo com o melhor desempenho ao longo do treino, garantindo que a versão final do modelo corresponde ao ponto de menor *loss* na validação. Este procedimento assegura que o modelo treinado reflete a melhor configuração possível, ao mesmo tempo que reduz o risco de degradação do desempenho em *epochs* subsequentes conduzindo a um modelo final que maximiza o desempenho em cenários reais.

O processo de treino foi acompanhado pela geração de gráficos que mostram a evolução da *accuracy* e da *loss* durante as *epochs*, conforme ilustrado no exemplo da figura 11. Estes gráficos são fundamentais para entender o comportamento do modelo ao longo do tempo e identificar possíveis problemas como *overfitting*.

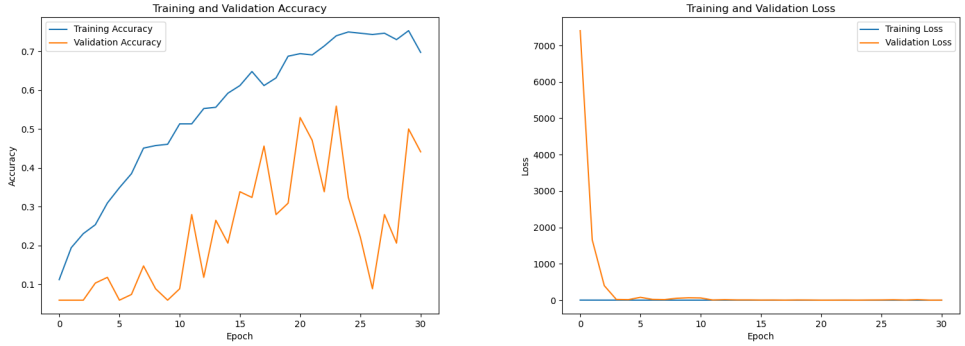


Figura 11: Imagens exemplo dos gráficos gerados da evolução da *accuracy* e da *loss* ao longo do treino do modelo

Após o processo de treino de cada modelo, foi calculada a sua matriz de confusão para avaliar o seu desempenho em prever cada classe individualmente. A matriz de confusão gerada, conforme apresentada no exemplo da figura 12, oferece uma visualização detalhada da classificação real em comparação com a classificação prevista do modelo, permitindo uma análise mais granular dos resultados.

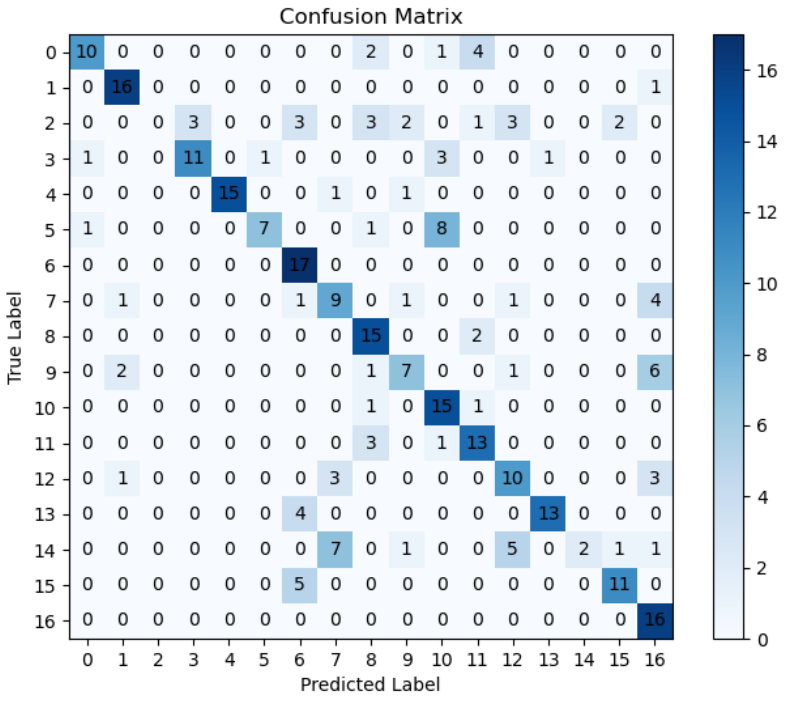


Figura 12: Imagem de exemplo da matriz confusão resultante da avaliação após treino de um modelo

4.5 RESULTADOS OBTIDOS

Esta secção apresenta e analisa os resultados obtidos a partir das diferentes fases de teste e otimização dos modelos. Após a realização dos testes de avaliação dos *datasets*, seleção de modelos pré-treinados, personalização da arquitetura e ajustes de hiperparâmetros, e fine-tuning, os resultados obtidos foram avaliados com base em métricas de desempenho e eficiência. A análise detalhada destes resultados visa proporcionar uma compreensão clara das melhorias alcançadas e dos impactos de cada fase no desempenho dos modelos, permitindo uma reflexão crítica sobre a eficácia das abordagens implementadas e as suas implicações para futuros desenvolvimentos no campo.

4.5.1 *Avaliação de Datasets*

Tal como referido anteriormente, os primeiros testes realizados focaram-se na avaliação dos *datasets*, considerando a variação no número de *batches* e a aplicação

ou não de *augmentation* adicional durante a preparação dos dados. Nas tabelas 3,4 e 5, que apresentam os resultados, a coluna com o cabeçalho "AUG"indica se o *dataset* em questão foi sujeito a *augmentation*. As colunas com os cabeçalhos "MVL"e "MVA"correspondem, respetivamente, à perda (*loss*) média de validação das 10 menores perdas (*Mean Validation Loss*) e à precisão (*accuracy*) média de validação das 10 maiores precisões (*Mean Validation Accuracy*).

Ao analisar o desempenho dos modelos treinados em três conjuntos de dados distintos, cada um com um número diferente de classes, observam-se variações significativas nos resultados, tanto entre os diferentes conjuntos de dados quanto entre as configurações dos próprios modelos.

NR	BATCHE	AUG	F1	RECALL	PRECISION	MVL	MVA
1	8	No	77.79%	79.45%	85.86%	1.3439	57.89%
2	8	Yes	86.72%	86.94%	89.89%	1.5461	63.73%
3	16	No	75.97%	76.01%	80.23%	1.8434	56.71%
4	16	Yes	91.10%	91.25%	91.87%	1.7971	61.32%

Tabela 3: Resultados dos modelos treinados com o *dataset* de 19 classes

No conjunto de dados com 19 classes (ver tabela 3), o uso de *Data Augmentation* demonstrou ser bastante eficaz. Por exemplo, o modelo 1, que não utilizou *augmentation*, obteve um *F1 Score* de 77,79%, um *Recall* de 79,45% e uma *Precision* de 85,86%. Estes resultados melhoraram significativamente com a aplicação de *Data Augmentation*, onde o modelo 2 obteve um *F1 Score* de 86,72%, um *Recall* de 86,94% e uma *Precision* de 89,89%. Esta tendência continuou com os modelos 3 e 4, onde o aumento do tamanho do *batch* e a introdução de *augmentation* elevaram o *F1 Score* de 75,97% para 91,10%.

No entanto, é importante observar que a *loss* média de validação aumentou ligeiramente com o uso de *augmentation*, apesar da *accuracy* média de validação ter melhorado. O aumento na *loss* média, combinado com a melhoria na *accuracy*, sugere que o modelo pode estar a fazer previsões menos precisas para um subconjunto específico dos dados. Esta suposição é confirmada ao analisar os dados do Apêndice C, onde a matriz de confusão (*confusion matrix*) revela que os modelos enfrentam maior dificuldade na classificação das *labels* 4 e 6, correspondendo respetivamente ao segundo fonema das palavras "dois"e "três", fonema este que está associado ao som da letra "S". Em decorrência desses resultados, decidiu-se criar um conjunto de dados com 17 classes para evitar este problema.

NR	BATCHES	AUG	F1	RECALL	PRECISION	MVL	MVA
5	8	No	61.81%	65.29%	72.68%	3.0306	42.79%
6	8	Yes	93.17%	93.08%	93.88%	1.1051	72.12%
7	16	No	69.55%	70.09%	78.20%	2.1654	56.18%
8	16	Yes	94.14%	94.09%	94.55%	1.5692	67.63%

Tabela 4: Resultados dos modelos treinados com o *dataset* de 17 classes

No conjunto de dados com 17 classes (ver tabela 4), o impacto da *augmentation* foi ainda mais pronunciado. Sem *augmentation*, o modelo 5 apresentou uma *F1 Score* de 61,81%, *Recall* de 65,29% e *Precision* de 72,68%. Com a aplicação de *augmentation*, o *F1 Score* disparou para 93,17%, com o *Recall* e a *Precision* a subirem para 93,08% e 93,88%, respetivamente. Este conjunto de dados mostrou-se o mais maleável para os modelos, especialmente quando a *augmentation* foi utilizada, com a *Accuracy* média de validação a atingir os 72,12%, o que é superior ao observado no *dataset* anterior. Os restantes dados dos testes a estes modelos encontra-se no Apêndice D.

NR	BATCHES	AUG	F1	RECALL	PRECISION	MVL	MVA
9	8	No	62.19%	64.97%	73.69%	2.5312	44.23%
10	8	Yes	78.40%	79.01%	82.38%	1.3082	67.02%
11	16	No	31.06%	24.23%	54.09%	10.9344	19.86%
12	16	Yes	95.26%	95.17%	95.78%	1.1738	74.24%

Tabela 5: Resultados dos modelos treinados com o *dataset* de 18 classes

No conjunto de dados com 18 classes(ver tabela 5), os resultados sem *augmentation* foram significativamente inferiores aos dos outros conjuntos de dados. Por exemplo, o modelo 11 sem *augmentation* apresentou uma *F1 Score* de apenas 31,06%, uma *Recall* de 24,23% e uma *Precision* de 54,09%. Estes valores indicam que o modelo teve grandes dificuldades em lidar com este conjunto de dados, possivelmente devido à complexidade ou à distribuição das classes. Contudo, a introdução de *augmentation* trouxe uma melhoria dramática. O modelo 12 atingiu um *F1 Score* de 95,26%, *Recall* de 95,17% e *Precision* de 95,78%, demonstrando que a *augmentation* foi fundamental para superar os desafios apresentados por este conjunto de dados. A *Accuracy* média de validação também melhorou consideravelmente, passando de 19,86% para 74,24% com a aplicação de *augmentation* e o aumento do tamanho do *batch*. Através deste conjunto de dados, foi possível desenvolver o modelo 12, que se destacou como o melhor modelo até ao momento no projeto.

Em termos gerais, observou-se que a *augmentation* desempenha um papel crucial na melhoria do desempenho dos modelos. O conjunto de dados com 17 classes mostrou-se o mais favorável aos modelos em geral, alcançando altos valores de *F1 Score* e *Accuracy* com a *augmentation*. Por outro lado, o conjunto de dados com 18 classes, embora inicialmente problemático, beneficiou enormemente da aplicação de *augmentation*, construindo o melhor resultado.

Em conclusão, estes resultados sugerem que para tarefas com elevada complexidade a utilização de *augmentation* é essencial para melhorar a capacidade do modelo em generalizar e obter resultados mais precisos. Além disso, a escolha do conjunto de dados e a configuração do modelo são determinantes para o sucesso do modelo, sendo necessário ajustar estas variáveis conforme a especificidade de cada problema. É importante notar que, embora a utilização de *augmentation* tenha melhorado os resultados, isto indica que o nosso modelo, por si só, deveria ter sido treinado com um conjunto de dados maior para obter resultados mais estáveis e menos dependentes de *augmentation*.

4.5.2 *Testes com Modelos Pré-Treinados*

Nos testes com modelos pré-treinados, foram avaliados seis novos modelos. Com base nos resultados prévios, decidiu-se manter as configurações do melhor modelo dos testes anteriores (ResNet50/ modelo 12) que inclui o uso de 16 *batches*, *data augmentation* e o *dataset* de 18 classes. Esses parâmetros foram aplicados uniformemente em todos os modelos pré-treinados desta fase de testes.

MODELO	F1	RECALL	PRECISION	MVL	MVA
ResNet50	95.26%	95.17%	95.78%	1.1738	74.24%
InceptionV3	96.54%	96.58%	96.68%	0.7718	80.03%
VGG16	76.83%	76.70%	80.08%	1.5791	48.79%
MobileNetV2	66.73%	66.79%	71.44%	2.0614	34.14%
Xception	47.05%	47.64%	57.37%	2.698	37.71%
ResNet101V2	29.50%	32.49%	62.20%	8.3531	32.29%
DenseNet121	36.70%	38.88%	50.97%	2.1524	36.45%

Tabela 6: Resultados dos treinos de modelos usando diferentes modelos pré-treinados

Após a realização dos testes com os modelos pré-treinados apresentados na Tabela 6 e com a informação suplementar no Apêndice F, podemos observar uma variabilidade significativa no desempenho entre os diferentes modelos. O modelo

InceptionV3 destacou-se como o melhor modelo desta fase, superando o ResNet50, que anteriormente tinha sido o melhor. O InceptionV3 obteve um *F1 Score* de 96,54%, um *Recall* de 96,58% e uma *Precision* de 96,68%, com uma *Mean Validation Loss* (MVL) de 0,7718 e uma *Mean Validation Accuracy* (MVA) de 80,03%. Estes resultados indicam que o InceptionV3 foi mais eficaz na generalização dos padrões do *dataset*, conseguindo não só melhorar a precisão, mas também reduzir a perda média de validação.

Por outro lado, o VGG16, apesar de ser um modelo robusto, apresentou um desempenho inferior, com um *F1 Score* de 76,83% e uma MVA de apenas 48,79%. Isso pode indicar que, embora o VGG16 seja eficaz em tarefas mais simples, ele pode não ser tão adequado para a complexidade do *dataset* utilizado.

Modelos como o MobileNetV2, Xception, ResNet101V2, e DenseNet121 mostraram resultados consideravelmente inferiores, com *F1 Scores* e MVA bem abaixo do esperado. O ResNet101V2, em particular, obteve os piores resultados, com um *F1 Score* de 29,50% e uma MVA de 32,29%. Isso sugere que, apesar de serem modelos poderosos em termos de arquitetura, eles podem não estar suficientemente otimizados ou adaptados ao *dataset* específico e às configurações utilizadas neste projeto.

A análise dos resultados sugere que, embora os modelos pré-treinados ofereçam uma base sólida, a sua eficácia depende fortemente da compatibilidade entre a arquitetura do modelo e as características do *dataset*. O sucesso do InceptionV3 destaca a importância de experimentar diferentes arquiteturas para identificar aquela que melhor se adapta às necessidades específicas da tarefa.

4.5.3 *Personalização da Arquitetura e Ajuste de Hiperparâmetros*

Dado o desempenho promissor dos modelos ResNet50 e InceptionV3 na fase anterior, decidiu-se explorar mais profundamente o potencial desses modelos.

4.5.3.1 *Adição de novas camadas*

A análise dos gráficos no Apêndice G e dos resultados apresentados na Tabela 7 evidencia o impacto de cada abordagem no desempenho dos modelos. Observa-se que os resultados variam significativamente de acordo com as alterações feitas na arquitetura dos modelos e nas estratégias de treino. Essas modificações influenci-

aram o desempenho de maneira distinta, destacando como diferentes ajustes nas arquiteturas e no processo de treino afetam os resultados obtidos.

MODELO	F1	RECALL	PRECISION	MVL	MVA
InceptionV3_1024_05	89.06%	88.82%	90.26%	1.3125	67.07%
InceptionV3_64_02	68.88%	70.36%	73.28%	1.4934	53.93%
RESNET50_1024_05	87.95%	87.94%	88.94%	1.8589	60.51%
RESNET50_64_02	83.96%	83.84%	86.17%	1.7894	60.41%
InceptionV3_8batch	91.59%	91.56%	92.23%	1.2135	68.89%
InceptionV3_T	77.16%	77.20%	78.97%	2.2384	46.43%
RESNET50_16_T	78.11%	77.79%	81.13%	1.5825	49.77%

Tabela 7: Resultados dos treinos de modelos onde houve alteração da arquitetura do modelo

Quando analisamos as modificações, o modelo InceptionV3_8batch, que utilizou apenas 8 *batches*, conseguiu o melhor desempenho entre as versões modificadas. Este resultado sugere que a redução do número de *batches* pode ter ajudado o modelo a convergir de forma mais eficiente, aproximando-se mais do comportamento do modelo original. No entanto, apesar de ter melhorado em relação às outras versões modificadas, ainda ficou aquém dos resultados do InceptionV3 original, indicando que a arquitetura inicial já estava otimizada para este problema específico. Por outro lado, os modelos InceptionV3_1024_05 e InceptionV3_64_02, que utilizaram diferentes tamanhos de camadas densas, apresentaram uma redução de desempenho significativa em comparação com o modelo original. Embora o Dropout possa ter contribuído para uma melhor regularização, estas alterações não compensaram a perda de desempenho em termos de precisão e generalização. A camada densa de 1024 neurónios, por exemplo, não conseguiu replicar a eficácia da arquitetura original, o que sugere que o modelo original estava bem equilibrado em termos de complexidade e capacidade de generalização. O modelo InceptionV3_T, que teve as camadas congeladas, demonstrou ser a configuração menos eficaz, com uma clara degradação de desempenho. Isto pode ser atribuído à incapacidade do modelo de se adaptar ao novo conjunto de dados, visto que as camadas congeladas não permitiram a atualização dos pesos para captar melhor as características dos novos dados. Comparado com o modelo original, o InceptionV3_T apresentou uma perda de flexibilidade, o que resultou numa generalização muito mais fraca. Em resumo, o modelo InceptionV3 original continuou a ser o mais eficaz em todos os parâmetros, e entre as versões modificadas, o InceptionV3_8batch foi o que mais se aproximou do desempenho esperado. As modificações na arquitetura, como o aumento das camadas densas e o congelamento de camadas, não foram benéficas, sugerindo que

o modelo pré-treinado na sua configuração original já estava bem ajustado para esta tarefa específica.

Nos modelos ResNet50, observamos uma tendência semelhante àquela encontrada nos modelos InceptionV3. O modelo original ResNet50, sem alterações, apresentou o melhor desempenho em todas as métricas, destacando-se pela sua alta precisão e generalização. Este resultado reforça a robustez da arquitetura pré-treinada, que já estava bem otimizada para lidar com o conjunto de dados. Entre os modelos modificados, o RESNET50_1024_05 e o RESNET50_64_02 foram os que mais se aproximaram do desempenho do modelo original, embora com uma ligeira perda de eficácia. Assim como nos modelos InceptionV3, a adição de camadas densas de diferentes tamanhos, com o uso de Dropout, não resultou em melhorias significativas. Na verdade, estas alterações parecem ter tornado o modelo mais complexo sem aumentar a sua capacidade de generalização, o que sugere que as camadas adicionais podem ter levado a um excesso de parametrização. O modelo RESNET50_16_T apresentou um desempenho significativamente inferior ao modelo original. A limitação imposta pelo congelamento das camadas impediu o modelo de ajustar os seus pesos para se adaptar ao novo conjunto de dados, resultando num desempenho muito abaixo do esperado. Esta abordagem, similar ao que foi observado no modelo InceptionV3_T, demonstrou que o congelamento de camadas não é uma estratégia eficaz para este conjunto de dados, pois reduz a flexibilidade do modelo.

4.5.3.2 *Ajustes nos hiperparâmetros*

Ao analisar os resultados apresentados na Tabela 8 e os gráficos do Apêndice H, notam-se diferenças consideráveis no desempenho. Comparando estes modelos com o InceptionV3 original, observamos que as alterações nos hiperparâmetros, como a taxa de aprendizagem e os *optimizers*, influenciaram significativamente os resultados.

MODELO	F1	RECALL	PRECISION	MVL	MVA
InceptionV3_RMS_001	63.73%	65.27%	74.21%	3.0071	41.85%
InceptionV3_SGD	32.78%	38.30%	67.25%	22.9057	35.37%
InceptionV3_EXPD	75.51%	75.23%	80.87%	2.2561	46.40%

Tabela 8: Resultados dos treinos de modelos onde houve ajuste dos hiperparâmetros do modelo

O InceptionV3_EXPD foi o que apresentou o melhor desempenho entre os modelos com ajustes de hiperparâmetros. O uso do *optimizer* com taxa de aprendizagem

exponencial conseguiu aproximar-se razoavelmente do desempenho do modelo original, embora ainda tenha ficado aquém. Este resultado sugere que, ao ajustar a taxa de aprendizagem de forma exponencial, o modelo foi capaz de melhorar a sua convergência sem comprometer muito a generalização. Por outro lado, o InceptionV3_RMS_001, que utilizou o *optimizer RMSprop* com uma taxa de aprendizagem de 0.001, teve um desempenho significativamente inferior. Isso indica que, apesar de o *RMSprop* ser um *optimizer* eficiente em muitos casos, nesta configuração específica, não conseguiu capturar bem as características do conjunto de dados. A elevada perda de validação (MVL) observada neste modelo é um indicativo de que o modelo teve dificuldades em generalizar, possivelmente devido a uma taxa de aprendizagem inadequada para este problema. O InceptionV3_SGD apresentou o pior desempenho entre todos os modelos analisados. O *optimizer* SGD, conhecido pela sua simplicidade e eficiência em problemas com grandes volumes de dados, não conseguiu produzir bons resultados neste cenário específico. A combinação de uma baixa precisão e uma elevada perda de validação sugere que o modelo teve dificuldades tanto para aprender quanto para generalizar, resultando num desempenho global muito insatisfatório. Em comparação com o modelo original, os ajustes de hiperparâmetros não trouxeram melhorias significativas. O modelo original InceptionV3 manteve-se como o mais eficaz, demonstrando que, para este conjunto de dados, a configuração original de hiperparâmetros já estava bem ajustada.

4.5.3.3 *Alteração do nível de data augmentation*

Nesta subsecção, foram testadas diferentes configurações de *augmentation* para avaliar como a alteração deste parâmetro influencia o desempenho dos modelos baseados na arquitetura InceptionV3. Foram treinados 2 modelos: InceptionV3_AUG_00 (sem *data augmentation*), e InceptionV3_AUG_03 (com *data augmentation* mais agressivo de 0,3). Para termos de comparação também foi adicionado na tabela o melhor modelo InceptionV3 (com *data augmentation* moderado de 0.1) que vem de treinos anteriores. Os valores de 0,1 e 0,3 mencionados para o *data augmentation* referem-se à magnitude das transformações aplicadas às imagens durante o treino dos modelos. Esses valores representam a percentagem máxima de variação aplicada em três parâmetros principais: width_shift_range, height_shift_range e zoom_range. Quando o valor é 0,1, significa que as imagens podem ser deslocadas horizontalmente ou verticalmente até 10% das suas dimensões originais, ou aumentadas/reduzidas até 10% do sei tamanho mediante técnicas de *'zoom'*. Da mesma forma, o valor de 0,3 indica variações de até 30% nesses mesmos parâmetros. Estas transformações

visam criar versões ligeiramente diferentes das imagens originais, ajudando o modelo a generalizar melhor para novos dados.

MODELO	F1	RECALL	PRECISION	MVL	MVA
InceptionV3_AUG_00	93.96%	93.89%	94.50%	1.4249	72.21%
InceptionV3	96.54%	96.58%	96.68%	0.7718	80.03%
InceptionV3_AUG_03	64.43%	66.09%	72.37%	2.2862	44.58%

Tabela 9: Resultados dos treinos de modelos onde houve modificação do nível de *augmentation*

Ao analisar os resultados obtidos da Tabela 9 podemos observar que o modelo InceptionV_AUG_00, que foi treinado sem qualquer *data augmentation*, apresentou um desempenho sólido. A pontuação F1 foi de 93,96%, o Recall de 93,89% e a Precision de 94,50%, o que sugere que o modelo conseguiu capturar bem os padrões nos dados de treino sem exagerar nas previsões incorretas. No entanto, a *Mean Validation Loss* (MVL) foi relativamente alta, atingindo 1,4249, e a *Mean Validation Accuracy* (MVA) foi de 72,21%. Esses números indicam que, embora o modelo tenha se ajustado bem ao conjunto de treino, ele pode ter sofrido de *overfitting*, isto é, teve dificuldades em generalizar para os dados de validação. Por outro lado, o modelo InceptionV3_AUG_03, que foi treinado com um *data augmentation* mais agressivo (com deslocamentos e '*zoom*' de até 30%), teve um desempenho significativamente inferior em comparação com os outros dois modelos. A pontuação F1 caiu para 64,43%, o Recall para 66,09% e a Precision para 72,37%. A MVL aumentou drasticamente para 2,2862, e a MVA desceu para 44,58%, indicando que o modelo teve dificuldades em aprender os padrões dos dados. O uso excessivo de *data augmentation* pode ter introduzido ruído nos dados de treino, prejudicando a capacidade do modelo de capturar características importantes.

Em suma, os resultados mostram que o *data augmentation* pode ser uma ferramenta poderosa para melhorar a generalização de um modelo, mas o nível de augmentation deve ser ajustado cuidadosamente. O modelo InceptionV3 com *data augmentation* moderado de 0,1 demonstrou que continua a ser o mais eficiente, obtendo o melhor equilíbrio entre a capacidade de aprender com os dados de treino e a capacidade de generalizar para novos dados. Por outro lado, o excesso de *data augmentation*, como no InceptionV3_AUG_03, pode ter um efeito adverso, deteriorando o desempenho do modelo.

4.5.4 *Testes de Fine-Tuning*

Na Tabela 10, são apresentados os resultados dos modelos ajustados durante a fase do fine-tuning, onde foi feita uma modificação no número de camadas descongeladas (UL - *Unfrozen Layers*) e o modelo Inception_RLR teve a aplicação do método *ReduceLROnPlateau*. Este método ajusta a taxa de aprendizagem automaticamente quando o modelo não apresenta melhorias significativas durante o treino, o que pode servir para evitar a estagnação em mínimos locais.

MODELO	F1	RECALL	PRECISION	MVL	MVA
Inception_UL10	97.13%	97.14%	97.28%	0.4779	87.84%
Inception_UL20	97.14%	97.14%	97.26%	0.4654	87.97%
Inception_UL30	96.93%	96.95%	97.03%	0.4781	87.63%
Inception_UL50	96.84%	96.82%	96.99%	0.4695	87.48%
ResNet50_UL10	95.58%	95.54%	95.96%	0.7952	83.06%
Inception_RLR	95.06%	95.06%	95.34%	0.8385	79.72%

Tabela 10: Resultados dos treinos de Fine-Tuning

Os resultados indicam que, em geral, os modelos com camadas descongeladas (Inception_UL10, Inception_UL20, Inception_UL30 e Inception_UL50) apresentaram um desempenho superior ao modelo ResNet50_UL10 e ao modelo Inception_RLR. Os modelos da família Inception com camadas descongeladas alcançaram F1-scores superiores a 96.8%, com o melhor resultado sendo obtido pelo modelo Inception_UL20, que obteve um F1-score de 97.14%. O modelo ResNet50_UL10 apresentou um F1-score inferior (95.58%) quando comparado aos modelos Inception, sugerindo que a arquitetura ResNet50 pode não ter sido tão eficaz neste contexto de fine-tuning com 10 camadas descongeladas. Já o modelo Inception_RLR, apesar de utilizar a estratégia ReduceLROnPlateau, obteve o pior desempenho da tabela, com um F1-score de 95.06%. Isso sugere que, neste caso específico, o ajuste automático da taxa de aprendizagem não foi capaz de superar a abordagem tradicional de descongelamento de camadas.

Ao comparar os modelos da arquitetura Inception com diferentes números de camadas descongeladas (UL10, UL20, UL30 e UL50), observa-se que o aumento progressivo do número de camadas descongeladas não resultou em melhorias significativas no desempenho. De facto, o modelo com 20 camadas descongeladas (Inception_UL20) apresentou o melhor desempenho global, tanto em termos de F1-score (97.14%) quanto em termos de Mean Validation Accuracy (MVA) com

87.97%. Este resultado sugere que a descongelação de um número moderado de camadas pode ser suficiente para otimizar o desempenho do modelo, enquanto descongelar muitas camadas (como no UL50) pode levar a uma ligeira queda de desempenho. O Mean Validation Loss (MVL) foi também mais baixo nos modelos Inception com camadas descongeladas, com os valores variando de 0,4654 a 0,4781, indicando que esses modelos tiveram uma boa capacidade de generalização durante o treino. O modelo com o melhor desempenho, Inception_UL20, registou o menor MVL (0.4654) e a maior MVA (87.97%), reforçando a ideia de que a descongelação de um número intermediário de camadas pode ser uma estratégia eficaz para este tipo de tarefa.

Em síntese, os resultados desta fase de fine-tuning demonstram que a descongelação de um número controlado de camadas, particularmente 20 camadas no modelo Inception, resultou no melhor desempenho geral, sendo o modelo com melhores resultados até este momento no treino. Embora a aplicação do método ReduceLROnPlateau não tenha gerado os melhores resultados neste caso específico, esta estratégia pode ainda ser útil em outros contextos onde o ajuste manual da taxa de aprendizagem não é ideal. Portanto, a escolha cuidadosa do número de camadas a descongelar, com uma adequada gestão da taxa de aprendizagem, revelou-se essencial para a otimização do modelo neste estudo.

4.5.5 *Testes de aplicação*

```
For 20240901145749_3600.png:
Predicted Class: tre
Unk: 0.00% ze: 0.00% ro: 0.00% um: 0.00% do: 0.02% is: 0.00% tre: 99.28% qua: 0.00% tro: 0.04%
cin: 0.01% co: 0.00% sei: 0.04% se: 0.00% te: 0.00% oi: 0.00% to: 0.01% no: 0.60% ve: 0.00%
```

Figura 13: Imagem de exemplo de um resultado de classificação de um mel-espectrograma

Para esta fase, foram realizadas novas gravações de áudio para avaliar o desempenho do modelo Inception_UL20 na classificação dos fonemas correspondentes aos números de 0 a 9. Cada imagem de mel-espectrograma foi classificada, obtendo-se resultados semelhantes aos apresentados na figura 13. Após a classificação de todas as imagens de uma amostra de áudio, obtém-se um resultado semelhante ao mostrado na figura 14, onde é exibido o fonema previsto para cada imagem. Os valores "N/a"indicam situações em que o modelo não previu nenhuma classe com confiança superior a 85%.

```
Predicted Classes for all images:
['N/a', 'N/a', 'N/a', 'N/a', 'N/a', 'N/a', 'N/a', 'qua', 'N/a', 'tre', 'N/a', 'tre',
'tre', 'tre', 'tre', 'tre', 'tre', 'tre', 'tre', 'tre', 'tre', 'N/a', 'tre', 'tre',
'N/a', 'N/a', 'N/a', 'se', 'se', 'N/a', 'N/a', 'N/a', 'N/a', 'N/a', 'N/a', 'N/a',
'N/a', 'tro', 'tro', 'tro', 'tro', 'N/a', 'N/a', 'N/a', 'tro', 'tro', 'tro',
'tro', 'tro', 'tro', 'tro', 'tro', 'tro', 'N/a']
```

Figura 14: Imagem de exemplo de um resultado de classificação de uma amostra de som

Após obter os resultados, foi necessário filtrar as previsões redundantes ou inválidas, de modo a garantir que apenas previsões válidas e não repetidas fossem adicionadas à lista de fonemas previstos. Para isso, foi utilizada uma lógica que compara cada previsão com a anterior, adicionando à lista apenas as previsões que representem uma mudança efetiva de fonema e que possuam um nível de confiança superior ao limiar estabelecido. Previsões repetidas consecutivamente ou com uma confiança insuficiente (marcadas como "N/a") foram descartadas, o que assegura uma sequência final de fonemas mais limpa e precisa. Obtém-se um resultado final semelhante ao mostrado na figura 15.

```
Predicted Phonemes:
['qua', 'tre', 'se', 'tro']
```

Figura 15: Imagem de exemplo de um resultado final de classificação de uma amostra de som

Após esta breve explicação do processo de obtenção dos resultados, apresenta-se abaixo uma análise detalhada dos resultados obtidos para cada som gravado.

4.5.5.1 *Resultados para a palavra 'zero'*

Para o som "zero"(fonemas 'ze' e 'ro') cada uma das gravações obteve:

- 'qua', 'no', 'ze', 'se', 'um', 'tro'
- 'qua', 'tro', 'qua', 'ze', 'qua', 'se', 'qua', 'ro', 'tro', 'qua', 'tro'

Para os áudios com o som "zero", os resultados foram dispersos e incluíram classes inesperadas como 'qua', 'no', 'se', e 'um', além dos fonemas corretos 'ze' e 'ro'. Embora os fonemas corretos ('ze' e 'ro') tenham sido identificados, a presença de muitos fonemas incorretos sugere dificuldades do modelo em reconhecer consistentemente o som "zero".

4.5.5.2 *Resultados para a palavra 'um'*

Para o som 'um' (fonema 'um') cada uma das gravações obteve:

- 'no', 've', 'tro', 'qua', 'tro', 've'
- 'tro', 'qua', 'no', 'um', 've', 'tro', 've', 'tro'

Para o som 'um', que contém apenas o fonema 'um', os resultados também indicaram uma mistura de previsões corretas e incorretas. Em uma das gravações, o fonema correto 'um' foi identificado, mas em conjunto com fonemas como 'no', 've', e 'tro'.

4.5.5.3 *Resultados para a palavra 'dois'*

Para o som "dois"(fonemas 'do' e 'is') cada uma das gravações obteve:

- 'co', 'qua', 'oi', 'sei', 'is', 'tro'
- 'qua', 'tro', 'qua', 'sei', 'is', 'tro', 'qua'

Nos testes com o som "dois", o modelo conseguiu identificar os fonemas corretos 'is', mas falhou em identificar 'do' consistentemente. Em vez disso, produziu previsões como 'qua', 'oi', e 'tro', sugerindo um desempenho limitado para este número.

4.5.5.4 *Resultados para a palavra 'três'*

Para o som "três"(fonemas 'tre' e 'is') cada uma das gravações obteve:

- 'qua', 'tre', 'se', 'tro'
- 'tro', 'qua', 'tre', 'is', 'tro'

Para o som "três", os resultados mostraram uma identificação relativamente melhor do fonema 'tre', enquanto 'is' foi identificado com menos frequência. Apesar de alguma variação, o fonema 'tre' foi reconhecido em ambas as gravações, o que demonstra um desempenho moderado do modelo para este som.

4.5.5.5 *Resultados para a palavra 'quatro'*

Para o som "quatro"(fonemas 'qua' e 'tro') cada uma das gravações obteve:

- 'Unk', 'tro', 'qua', 'co', 'Unk', 'qua', 'Unk', 'qua', 'tro'
- 'qua', 'tro', 'qua', 'tro', 'qua', 'tro'

O som "quatro"apresentou um desempenho razoável, com o modelo identificando os fonemas corretos na maioria das previsões. Neste caso, o modelo conseguiu reconhecer 'qua' e 'tro' consistentemente, sugerindo uma boa capacidade de classificação para este número.

4.5.5.6 *Resultados para a palavra 'cinco'*

Para o som "cinco"(fonemas 'cin' e 'co') cada uma das gravações obteve:

- 'ro', 'tro', 'Unk', 'tre', 'ze', 'tro', 'cin', 'tro'
- 'tro', 'cin', 'tro', 'co', 'qua', 'tro'

Os testes com o som "cinco"mostraram que o modelo identificou corretamente os fonemas 'cin' e 'co', embora também tenha gerado algumas previsões irrelevantes como 'ro' e 'tre'. Apesar de algumas previsões erróneas, o modelo foi capaz de detetar os fonemas corretos em ambas as gravações.

4.5.5.7 *Resultados para a palavra 'seis'*

Para o som "seis"(fonemas 'sei' e 'is') cada uma das gravações obteve:

- 'co', 'tre', 'ze', 'sei', 'is', 'tro'
- 'Unk', 'tre', 'se', 'sei', 'is', 'tro', 'Unk'

Para o som "seis", os resultados foram mistos, com o modelo identificando corretamente os fonemas 'sei' e 'is', mas também incluindo previsões incorretas como 'co' e 'tre'. Apesar das inconsistências, os fonemas corretos foram reconhecidos em ambas as tentativas.

4.5.5.8 *Resultados para a palavra 'sete'*

Para o som "sete"(fonemas 'se' e 'te') cada uma das gravações obteve:

- 'tre', 'Unk', 'ze', 'tre', 'se', 'tre', 'se', 'tro', 'tre', 'co', 'ro', 'tro', 'Unk'
- 'qua', 'ze', 'qua', 'ze', 'se', 'qua', 'se', 'cin', 'tro', 'qua', 'tro', 'qua', 'co'

O som "sete"apresentou uma grande variabilidade nos resultados, com o modelo frequentemente falhando em identificar ambos os fonemas 'se' e 'te'. Embora o fonema 'se' tenha sido reconhecido, 'te' não foi identificado consistentemente, mostrando um desempenho fraco para este número.

4.5.5.9 *Resultados para a palavra 'oito'*

Para o som "oito"(fonemas 'oi' e 'to') cada uma das gravações obteve:

- 'Unk', 'tro', 'Unk', 'qua', 'oi', 'tro', 'tre', 'tro'
- 'qua', 'Unk', 'qua', 'oi', 'tro'

Para o som "oito", o modelo identificou o fonema 'oi', mas apresentou dificuldades em reconhecer 'to' de forma consistente, misturando previsões como 'Unk' e 'tro'. O reconhecimento parcial dos fonemas corretos sugere um desempenho moderado para este som.

4.5.5.10 *Resultados para a palavra 'nove'*

Para o som "nove"(fonemas 'no' e 've') cada uma das gravações obteve:

- 'qua', 've', 'tro'
- 'qua', 'tro', 'qua', 'no', 'qua', 've', 'tro', 'qua', 'tro', 'qua', 'tro'

Nos testes com o som "nove", o modelo teve algum sucesso em identificar os fonemas corretos 'no' e 've', apesar de algumas previsões erradas. O fonema 've' foi reconhecido consistentemente, mas o fonema 'no' apresentou um desempenho misto.

4.5.5.11 *Conclusão*

Em geral, o modelo conseguiu identificar alguns fonemas corretamente, especialmente para números como "quatro"e "cinco". No entanto, houve uma considerável variabilidade nos resultados, com previsões incorretas ou irrelevantes aparecendo frequentemente, o que indica que o modelo pode ter dificuldades em lidar com diferentes tonalidades, velocidades de fala, ruído, sotaques, entre outros fatores. O desempenho foi particularmente inconsistente para números como "sete"e "zero", sugerindo que melhorias no treino do modelo ou no pré-processamento dos dados podem ser necessárias para aumentar a precisão nas previsões.

5

CONCLUSÕES

O desenvolvimento de modelos de classificação de fonemas baseados em mel-espectrogramas apresentou desafios significativos devido à complexidade do reconhecimento de padrões acústicos em dados de áudio. Este projeto teve como objetivo criar um sistema robusto e eficiente para classificar fonemas, utilizando redes *deep leraning* e técnicas de machine learning. Durante o trabalho, foram explorados vários modelos pré-treinados, estratégias de personalização de arquitetura e técnicas de ajuste de hiperparâmetros para maximizar a eficácia do sistema, sendo a fase de testes crucial para avaliar o desempenho e identificar melhorias.

Um dos principais desafios foi a seleção e preparação dos dados, que se mostraram insuficientes e comprometeram a robustez dos resultados. A limitação na amostra dificultou a obtenção de conclusões confiáveis, e a dificuldade em diferenciar fonemas no português, devido à sua semelhança sonora, agravou o problema. Muitos fonemas no português apresentam variações subtis, tornando o processo de identificação e discriminação sonora mais desafiante. Embora técnicas de data augmentation tenham sido essenciais para aumentar a diversidade dos dados e melhorar a capacidade dos modelos de generalizar, a falta de um dataset mais amplo ainda limitou o desempenho.

Os testes com modelos pré-treinados, como ResNet50 e InceptionV3, destacaram a importância da escolha da arquitetura base. O InceptionV3, em particular, mostrou uma notável capacidade de adaptação e superou outros modelos em métricas como F1 Score e precisão. Isso reforça a ideia de que uma arquitetura adequada pode ser crucial para o sucesso em tarefas complexas de classificação. No entanto, as personalizações nas arquiteturas e nos hiperparâmetros revelaram-se delicadas, com pequenas alterações resultando em variações significativas no desempenho. Isso sugere que modificações excessivas podem não trazer os resultados esperados e que a complexidade adicional deve ser justificada por ganhos de desempenho.

O fine-tuning também foi um aspeto crítico, onde descongelar um número controlado de camadas e ajustar a taxa de aprendizagem permitiu uma adaptação eficaz ao novo conjunto de dados. O modelo InceptionV3 com 20 camadas descongeladas se destacou como o mais robusto, evidenciando a importância de uma abordagem

equilibrada no fine-tuning para evitar tanto a subutilização do modelo quanto a sobrecarga de complexidade. No entanto, o modelo ainda enfrentou dificuldades em cenários desafiantes, como variações tonais, sugerindo a necessidade de melhorias no pré-processamento dos dados ou no aumento da diversidade das amostras.

Para futuros trabalhos, recomenda-se explorar novas técnicas de pré-processamento de áudio, como métodos avançados de redução de ruído e normalização, e expandir o dataset com gravações em diferentes ambientes e sotaques. A ampliação do dataset permitirá ao modelo aprender uma gama mais ampla de variações e melhorar a sua robustez em cenários reais. Além disso, o uso de técnicas avançadas, como redes neurais recorrentes, transformers ou LSTM, pode ser explorado para aprimorar o desempenho em tarefas de reconhecimento de fala.

Em suma, este trabalho representa um avanço importante na aplicação de machine learning ao reconhecimento de fonemas, destacando tanto o potencial das redes neurais profundas quanto os desafios que permanecem. O modelo final demonstrou capacidades promissoras, mas evidenciou a necessidade contínua de otimizações e ajustes, especialmente para melhorar a generalização em cenários variados. Com as melhorias recomendadas, espera-se que o sistema possa ser aplicado com sucesso em reconhecimento de fala, contribuindo para avanços nesta área de estudo.

BIBLIOGRAFIA

Abadi, Martín et al. (s.d.). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. Rel. téc. URL: www.tensorflow.org..

Allen, James F (1991). *Natural Language, Knowledge Representation, and Logical Form*. Rel. téc.

Alpaydın, Ethem (2014). «Introduction to Machine Learning - Third Edition». Em.

Bengio, Yoshua et al. (2003). *A Neural Probabilistic Language Model*. Rel. téc., pp. 1137–1155.

Bergstra, James et al. (2010). *Theano: A CPU and GPU Math Compiler in Python*. Rel. téc., p. 1. URL: http://groups.google.com/group/theano-users.

Bishop M. Christopher (2006). *Pattern Recognition and Machine Learning*. Rel. téc.

Bojarski, Mariusz et al. (abr. de 2016). «End to End Learning for Self-Driving Cars». Em: URL: http://arxiv.org/abs/1604.07316.

Chen, Tianqi et al. (dez. de 2015). «MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems». Em: URL: http://arxiv.org/abs/1512.01274.

Chollet, François et al. (2015). *keras*.

Davis, Jesse e Mark Goadrich (s.d.). *The Relationship Between Precision-Recall and ROC Curves*. Rel. téc.

Dean, Jeffrey et al. (s.d.). *Large Scale Distributed Deep Networks*. Rel. téc.

Deng, Li e Xiao Li (2013). «Machine learning paradigms for speech recognition: An overview». Em: *IEEE Transactions on Audio, Speech and Language Processing* 21.5, pp. 1060–1089. ISSN: 15587916. DOI: 10.1109/TASL.2013.2244083.

Glorot, Xavier e Yoshua Bengio (s.d.). *Understanding the difficulty of training deep feedforward neural networks*. Rel. téc. URL: http://www.iro.umontreal..

Goodfellow, Ian, Yoshua Bengio e Aaron Courville (2016). *Deep Learning*. Rel. téc.

Graves, Alex, Abdel-Rahman Mohamed e Geoffrey Hinton (2013). *Speech recognition with deep recurrent neural networks*. Rel. téc.

Hinton, Geoffrey et al. (2012). «Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups». Em: *IEEE Signal Processing Magazine* 29.6, pp. 82–97. ISSN: 10535888. DOI: 10.1109/MSP.2012.2205597.

Hochreiter, Sepp e Jürgen Schmidhuber (nov. de 1997). «Long Short-Term Memory». Em: *Neural Computation* 9.8, pp. 1735–1780. ISSN: 08997667. DOI: 10.1162/neco.1997.9.8.1735.

Jelinek, Frederick (1999). *Statistical methods for speech recognition*. eng. 2nd printing. Language, Speech and Communication. Cambridge, Mass. ; MIT Press. ISBN: 0262100665.

Jia, Yangqing et al. (jun. de 2014). «Caffe: Convolutional Architecture for Fast Feature Embedding». Em: URL: http://arxiv.org/abs/1408.5093.

Juang, B H e ; L R Rabiner (1991). *Hidden Markov Models for Speech Recognition*. Rel. téc. 3, pp. 251–272.

Juang, B H e Lawrence R Rabiner (2004). *Automatic Speech Recognition-A Brief History of the Technology Development*. Rel. téc. URL: http://www.recording-history.org/.

Jurafsky, Dan. e James H.. Martin (2000). *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall, p. 934. ISBN: 0130950696.

Krizhevsky, Alex, Ilya Sutskever e Geoffrey E Hinton (s.d.). *ImageNet Classification with Deep Convolutional Neural Networks*. Rel. téc. URL: http://code.google.com/p/cuda-convnet/.

Lecun, Yann, Yoshua Bengio e Geoffrey Hinton (mai. de 2015). *Deep learning*. DOI: 10.1038/nature14539.

Lipton, Zachary C. (jun. de 2016). «The Mythos of Model Interpretability». Em: URL: http://arxiv.org/abs/1606.03490.

Manning, Christopher D., Prabhakar Raghavan e Hinrich Schütze (2009). «An Introduction to Information Retrieval». Em.

Mcculloch, Warren S e Walter Pitts (1990). *A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY* n*. Rel. téc. 2, pp. 99–115.

Mcfee, Brian et al. (s.d.). *librosa: Audio and Music Signal Analysis in Python*. Rel. téc., p. 1. URL: https://github.com/bmcfee/librosa.

Mikolov, Tomáš et al. (2010). *Recurrent neural network based language model*. Rel. téc.

Müller, Andreas C e Sarah Guido (s.d.). *Introduction to Machine Learning with Python A GUIDE FOR DATA SCIENTISTS Introduction to Machine Learning with Python*. Rel. téc.

Murphy, Kevin P. (2012). *Machine learning: A Probabilistic Perspective*. MIT Press. ISBN: 9780262018029.

Nair, Vinod e Geoffrey E Hinton (s.d.). *Rectified Linear Units Improve Restricted Boltzmann Machines*. Rel. téc.

Pan, Sinno Jialin e Qiang Yang (2010). *A survey on transfer learning*. DOI: 10.1109/TKDE.2009.191.

Paszke, Adam et al. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Rel. téc.

Pedregosa FABIANPEDREGOSA, Fabian et al. (2011). *Scikit-learn: Machine Learning in Python Gaël Varoquaux Bertrand Thirion Vincent Dubourg Alexandre Passos PEDREGOSA, VAROQUAUX, GRAMFORT ET AL. Matthieu Perrot*. Rel. téc., pp. 2825–2830. URL: http://scikit-learn.sourceforge.net..

Powers, D M W e Ailab (s.d.). *EVALUATION: FROM PRECISION, RECALL AND F-MEASURE TO ROC, INFORMEDNESS, MARKEDNESS & CORRELATION*. Rel. téc.

Rabiner, Lawrence e Biing-Hwang Juang (1993). «Fundamentals of Speech Recognition». Em.

Rabiner, Lawrence R. e Ronald W. Schafer (2007). «Introduction to digital speech processing». Em: *Foundations and Trends in Signal Processing* 1.1-2, pp. 1–194. ISSN: 19328346. DOI: 10.1561/2000000001.

Radford, Alec et al. (s.d.). *Language Models are Unsupervised Multitask Learners*. Rel. téc. URL: https://github.com/codelucas/newspaper.

Rumelhart, David E., Geoffrey E. Hinton e Ronald J. Williams (1986). «Learning representations by back-propagating errors». Em: *Nature* 323.6088, pp. 533–536. ISSN: 00280836. DOI: 10.1038/323533a0.

Russell, Stuart e Peter Norvig (s.d.). «Artificial Intelligence A Modern Approach 4th Edition». Em: ().

Salamon, Justin e Juan Pablo Bello (ago. de 2016). «Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification». Em: DOI: 10.1109/LSP.2017.2657381. URL: http://arxiv.org/abs/1608.04363%20http://dx.doi.org/10.1109/LSP.2017.2657381.

Sokolova, Marina e Guy Lapalme (jul. de 2009). «A systematic analysis of performance measures for classification tasks». Em: *Information Processing and Management* 45.4, pp. 427–437. ISSN: 03064573. DOI: 10.1016/j.ipm.2009.03.002.

Srivastava, Nitish et al. (2014). *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Rel. téc., pp. 1929–1958.

Szegedy, Christian et al. (set. de 2014). «Going Deeper with Convolutions». Em: URL: http://arxiv.org/abs/1409.4842.

Van Rijsbergen, C J (s.d.). *INFORMATION RETRIEVAL*. Rel. téc.

Zhang, Boyang, Jared Leitner e Sam Thornton (s.d.). *Audio Recognition using Mel Spectrograms and Convolution Neural Networks*. Rel. téc.

APÊNDICES

APÊNDICE A

Neste apêndice podemos observar os 5 ficheiros de código que constituem todo o processo de tratamento de dados.

Listagem 1: Código-fonte do ficheiro `0_Webm_to_Wav.py` com sintaxe colorida

```
import os
from pydub import AudioSegment

# Directory containing raw audio files in webm format
RAW_DIR = "./Database/fonemasweb.appspot.com"

# Directory to store converted audio files in wav format
AUDIOS_DIR = "./Database/AudioWav"

def convert_webm_to_wav(webm_file, wav_file):
    """
    Convert a webm audio file to wav format.

    Parameters:
    webm_file (str): Path to the input webm file.
    wav_file (str): Path to the output wav file.
    """
    audio = AudioSegment.from_file(webm_file, format="webm")
    audio.export(wav_file, format="wav")

def process_audio_folders():
    """
    Process all audio files in RAW_DIR, converting them from webm to wav format.
    The converted files are saved in the corresponding folder structure in
    ↪  AUDIOS_DIR.
    """
    # Iterate over each folder in the raw audio directory
    for folder_name in os.listdir(RAW_DIR):
        folder_path = os.path.join(RAW_DIR, folder_name)
        output_folder_path = os.path.join(AUDIOS_DIR, folder_name)

        # Create the output folder if it doesn't exist
        if not os.path.exists(output_folder_path):
            os.makedirs(output_folder_path)

        # Iterate over each file in the current folder
        for file_name in os.listdir(folder_path):
            # Process only .webm files
            if file_name.endswith(".webm"):
                webm_file = os.path.join(folder_path, file_name)
                wav_file = os.path.join(output_folder_path,
                ↪  os.path.splitext(file_name)[0] + ".wav")
                convert_webm_to_wav(webm_file, wav_file)

if __name__ == "__main__":
    process_audio_folders()
```

Listagem 2: Código-fonte do ficheiro `1_WienerFilter.py` com sintaxe colorida

```python
import os
import librosa
import numpy as np
from scipy.signal import wiener
import soundfile as sf

# Define directories for input and output audio files
AUDIOS_DIR = "./Database/AudioWav"
OUTPUT_DIR = "./Database/AudioWiener"

class AudioProcessor:
    def process_wav_file(self, filepath, output_filepath):
        # Load the audio file with its original sample rate
        data, rate = librosa.load(filepath, sr=None)

        # Apply Wiener filtering to the audio data
        data_filtered = wiener(data)

        # Save the filtered audio data to the output file
        sf.write(output_filepath, data_filtered, rate)

def trim_audios():
    # Create output directory if it doesn't exist
    if not os.path.exists(OUTPUT_DIR):
        os.makedirs(OUTPUT_DIR)

    audio_processor = AudioProcessor()

    # Iterate over each folder in the input directory
    for number_folder in os.listdir(AUDIOS_DIR):
        number_folder_path = os.path.join(AUDIOS_DIR, number_folder)

        # Create a corresponding folder in the output directory
        output_number_folder = os.path.join(OUTPUT_DIR, number_folder)
        if not os.path.exists(output_number_folder):
            os.makedirs(output_number_folder)

        # Process each audio file in the current folder
        for audio_file in os.listdir(number_folder_path):
            audio_path = os.path.join(number_folder_path, audio_file)
            output_path = os.path.join(output_number_folder, audio_file)
            audio_processor.process_wav_file(audio_path, output_path)

if __name__ == "__main__":
    trim_audios()
```

Listagem 3: Código-fonte do ficheiro `2_MakeMels.py` com sintaxe colorida

```python
import librosa
import numpy as np
import matplotlib.pyplot as plt
import os

INPUT_FOLDER = 'INPUT_FOLDER'  # Input folder containing .wav files
OUTPUT_FOLDER = 'OUTPUT_FOLDER'  # Output folder to save images
SHIFT = 100
WINDOW_SIZE = 2000
HOP_LENGTH = 8
N_MELS = 256
```

```
12
13  def create_mel_spectrogram(wav_file, output_folder, window_size=WINDOW_SIZE):
14      # Create output folder if it doesn't exist
15      os.makedirs(output_folder, exist_ok=True)
16
17      # Load the audio file
18      y, sr = librosa.load(wav_file, sr=None)
19
20      # Compute the Mel spectrogram
21      mel_spectrogram = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=N_MELS,
        ↪  hop_length=HOP_LENGTH)
22
23      # Convert to decibels (log scale)
24      norm_melspec = librosa.power_to_db(mel_spectrogram, ref=np.max)
25
26      # Get number of time steps
27      num_time_steps = norm_melspec.shape[1]
28
29      # Generate spectrogram images with the specified window size and hop length
30      for i in range(0, num_time_steps - window_size, SHIFT):
31          # Extract the segment
32          segment = norm_melspec[:, i:i+window_size]
33
34          # Plot and save the segment as an image
35          plt.figure(figsize=(6, 4))
36          librosa.display.specshow(segment, sr=sr)
37          plt.axis('off')
38          plt.subplots_adjust(left=0, right=1, top=1, bottom=0)
39          plt.savefig(os.path.join(output_folder, f'mel_spec_{i}.png'))
40          plt.close()
41
42  # Process all .wav files in the input folder
43  for filename in os.listdir(INPUT_FOLDER):
44      if filename.endswith('.wav'):
45          wav_file = os.path.join(INPUT_FOLDER, filename)
46          # Create a corresponding output folder for each .wav file
47          output_folder = os.path.join(OUTPUT_FOLDER,
            ↪  os.path.splitext(filename)[0])
48          create_mel_spectrogram(wav_file, output_folder)
```

Listagem 4: Código-fonte do ficheiro `3_ImageSelector.py` com sintaxe colorida

```
1  import os
2  import random
3  import string
4  import shutil
5  import tkinter as tk
6  from tkinter import filedialog
7  from PIL import Image, ImageTk
8
9  def generate_random_string(length=8):
```

```
    return ''.join(random.choices(string.ascii_letters + string.digits,
    ↪  k=length))

def select_image(folder, output_subfolder):
    # Create a simple Tkinter window
    root = tk.Tk()
    root.title(f"Select an image from {os.path.basename(folder)}")

    # To keep a reference to image objects
    image_refs = []

    def on_image_click(image_path):
        random_name = generate_random_string() + ".png"
        dest_path = os.path.join(output_subfolder, random_name)
        shutil.copy(image_path, dest_path)
        print(f"Copied {image_path} to {dest_path}")
        root.destroy()

    for img_name in os.listdir(folder):
        if img_name.endswith(".png"):
            img_path = os.path.join(folder, img_name)
            img = Image.open(img_path)
            img.thumbnail((200, 200))
            img = ImageTk.PhotoImage(img)
            image_refs.append(img)  # Keep a reference to avoid garbage
            ↪  collection

            btn = tk.Button(root, image=img, command=lambda path=img_path:
            ↪  on_image_click(path))
            btn.image = img  # Also keep a reference in the button
            btn.pack(side=tk.LEFT)

    root.mainloop()

def main():
    src_base_folder = './Database/PhonemeSequences'
    dest_base_folder = './PhonemesManual'

    # Create the destination base folder if it doesn't exist
    if not os.path.exists(dest_base_folder):
        os.makedirs(dest_base_folder)

    # Iterate through each numbered folder
    for i in range(10):
        src_folder = os.path.join(src_base_folder, str(i))
        dest_folder = os.path.join(dest_base_folder, str(i))

        # Create the destination numbered subfolder if it doesn't exist
        if not os.path.exists(dest_folder):
            os.makedirs(dest_folder)

        # Iterate through each subfolder in the numbered folder
        for subfolder_name in os.listdir(src_folder):
            subfolder_path = os.path.join(src_folder, subfolder_name)
```

```
61              if os.path.isdir(subfolder_path):
62                  print(f"Opening subfolder: {subfolder_path}")
63                  select_image(subfolder_path, dest_folder)
64
65  if __name__ == "__main__":
66      main()
```

Listagem 5: Código-fonte do ficheiro `4_Augment.py` com sintaxe colorida

```
1   import os
2   import shutil
3   # Import imageio for reading and writing images
4   import imageio.v2 as imageio
5   # Import imgaug for image augmentation
6   import imgaug.augmenters as iaa
7
8   # Define input and output folders
9   INPUT_FOLDER = './Database/PhonemesManualUnknown'
10  OUTPUT_FOLDER = './Database/PhonemesManualUnknownAug'
11
12  # Augmentation sequence 1: Horizontal stretching
13  seq1 = iaa.Sequential([
14      iaa.Affine(scale={"x": (0.75, 1.25), "y": (1.0, 1.0)})
15  ])
16
17  # Augmentation sequence 2: Vertical stretching
18  seq2 = iaa.Sequential([
19      iaa.Affine(scale={"x": (1.0, 1.0), "y": (0.75, 1.25)})
20  ])
21
22  # Augmentation sequence 8: Shearing along X and Y axes
23  seq8 = iaa.Sequential([
24      iaa.ShearX((-3, 3)),
25      iaa.ShearY((-3, 3))
26  ])
27
28  # Augmentation sequence 9: Inverse shearing along X and Y axes
29  seq9 = iaa.Sequential([
30      iaa.ShearX((3, -3)),
31      iaa.ShearY((3, -3))
32  ])
33
34  # Function to augment images in a specified folder
35  def augment_images_in_folder(folder_path, output_folder):
36      # Create output folder if it doesn't exist
37      os.makedirs(output_folder, exist_ok=True)
38
39      # List all PNG files in the folder
40      png_files = [f for f in os.listdir(folder_path) if f.endswith('.png')]
41
42      for png_file in png_files:
```

```
        # Read the original image
        image_path = os.path.join(folder_path, png_file)
        original_image = imageio.imread(image_path)

        # Convert RGBA to RGB if the image has four channels
        if original_image.shape[2] == 4:
            original_image = original_image[:, :, :3]

        # Save the original image to the output folder
        shutil.copy(image_path, output_folder)

        # Apply augmentation sequences
        augmented_images = []
        augmented_images.extend(seq1.augment_images([original_image]))
        augmented_images.extend(seq2.augment_images([original_image]))
        augmented_images.extend(seq8.augment_images([original_image]))
        augmented_images.extend(seq9.augment_images([original_image]))

        # Save the augmented images to the output folder
        output_file_name = os.path.splitext(png_file)[0]
        for idx, augmented_image in enumerate(augmented_images):
            output_path = os.path.join(output_folder,
            ↪  f"{output_file_name}_aug_{idx}.png")
            imageio.imwrite(output_path, augmented_image)

        print(f"Augmented {png_file} saved successfully.")

if __name__ == "__main__":
    # List all subfolders in the input folder
    subfolders = [f for f in os.listdir(INPUT_FOLDER) if
    ↪  os.path.isdir(os.path.join(INPUT_FOLDER, f))]

    for subfolder in subfolders:
        subfolder_path = os.path.join(INPUT_FOLDER, subfolder)
        augment_images_in_folder(subfolder_path, os.path.join(OUTPUT_FOLDER,
        ↪  subfolder))
```

B

APÊNDICE B

Neste apêndice podemos observar o ficheiro de código que constituem o processo de treino e teste de modelos.

Listagem 6: Código-fonte do ficheiro `5_TrainModel.py` com sintaxe colorida

```
import os
import numpy as np
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint
import matplotlib.pyplot as plt
from sklearn.metrics import precision_score, recall_score, f1_score,
↪  confusion_matrix
from tensorflow import keras

# Define constants
MODEL_NAME = "TEST"
CLASSES = 19
BATCHES = 16
DATABASE_DIR = "PhonemesManualUnknownAug"

# Get the current directory
current_directory = os.getcwd()

# Specify the path to the "Database" folder
folder_path = os.path.join(current_directory, "Database")
dataset_folder_path = os.path.join(folder_path, DATABASE_DIR)

# Define image dimensions
img_height = 400
img_width = 600

#####################################################
# Callbacks
checkpoint_callback = [
    keras.callbacks.EarlyStopping(
        monitor="val_loss",
        patience=10
    ),
    keras.callbacks.ModelCheckpoint(
        filepath=MODEL_NAME + '_BEST.h5',
        save_best_only=True,
```

```
        monitor='val_loss',
        verbose=1,
        mode='min'
    )
]

################################################
# Model Architecture
# Load the pre-trained ResNet50 model without the top layers
pretrained_model = tf.keras.applications.ResNet50(
    include_top=False,
    weights='imagenet',
    input_shape=(img_height, img_width, 3),
    pooling='avg',
)
pretrained_model.trainable = True

# Define the model
model = Sequential([
    pretrained_model,
    Dense(CLASSES, activation='softmax')
])

# Display model architecture
model.summary()

# Compile the model
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# Data Augmentation
train_datagen = ImageDataGenerator(
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    validation_split=0.2
)

# Data Generators
train_generator = train_datagen.flow_from_directory(
    dataset_folder_path,
    target_size=(img_height, img_width),
    batch_size=BATCHES,
    class_mode='sparse',
    subset='training'
)
validation_generator = train_datagen.flow_from_directory(
    dataset_folder_path,
    target_size=(img_height, img_width),
    batch_size=BATCHES,
    class_mode='sparse',
```

```
    subset='validation'
)

# Training
history = model.fit(
    train_generator,
    epochs=100,
    validation_data=validation_generator,
    callbacks=checkpoint_callback
)

# Save the model
model.save(MODEL_NAME + ".h5")

# Extract validation losses and accuracies
validation_losses = np.array(history.history['val_loss'])
validation_accuracies = np.array(history.history['val_accuracy'])

# Identify the top 10 epochs with highest accuracy and lowest loss
sorted_accuracy_indices = np.argsort(-validation_accuracies)
best_accuracy_epoch_indices = sorted_accuracy_indices[:10]
sorted_loss_indices = np.argsort(validation_losses)
best_loss_epoch_indices = sorted_loss_indices[:10]

# Calculate the mean of the best 10 accuracies and losses
mean_best_accuracy = np.mean([validation_accuracies[i] for i in
↪  best_accuracy_epoch_indices])
mean_best_loss = np.mean([validation_losses[i] for i in best_loss_epoch_indices])

# Make predictions on the validation data
validation_generator.reset()
num_validation_samples = len(validation_generator.filenames)
y_val_true = []
y_val_pred_classes = []

# Iterate through the validation data and predict classes
for _ in range(num_validation_samples // validation_generator.batch_size + 1):
    x_val_batch, y_val_batch = validation_generator.next()
    y_val_true.extend(y_val_batch)
    y_val_pred_probs = model.predict(x_val_batch)
    y_val_pred_classes.extend(np.argmax(y_val_pred_probs, axis=1))

# Calculate precision, recall, and F1 score
precision = precision_score(y_val_true, y_val_pred_classes, average='macro')
recall = recall_score(y_val_true, y_val_pred_classes, average='macro')
f1 = f1_score(y_val_true, y_val_pred_classes, average='macro')

# Print evaluation metrics
print(f"Mean Validation Loss (10 Lowest Losses): {mean_best_loss:.4f}")
print(f"Mean Validation Accuracy (10 Highest Accuracies):
↪  {mean_best_accuracy:.2%}")
print(f"Precision: {precision:.2%}")
print(f"Recall: {recall:.2%}")
print(f"F1 Score: {f1:.2%}")
```

```
145
146 # Calculate the confusion matrix
147 conf_matrix = confusion_matrix(y_val_true, y_val_pred_classes)
148
149 # Plot the confusion matrix as a heatmap with numbers inside cells
150 plt.figure(figsize=(8, 6))
151 plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
152
153 # Annotate the confusion matrix with values
154 for i in range(len(conf_matrix)):
155     for j in range(len(conf_matrix[i])):
156         plt.text(j, i, str(conf_matrix[i, j]), ha='center', va='center',
          ↪  color='black')
157
158 plt.title('Confusion Matrix')
159 plt.colorbar()
160 tick_marks = np.arange(len(conf_matrix))
161 plt.xticks(tick_marks, tick_marks)
162 plt.yticks(tick_marks, tick_marks)
163 plt.xlabel('Predicted Label')
164 plt.ylabel('True Label')
165 plt.show()
166
167 # Plot the training and validation accuracy graph
168 plt.figure(figsize=(8, 6))
169 plt.plot(history.history['accuracy'], label='Training Accuracy')
170 plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
171 plt.xlabel('Epoch')
172 plt.ylabel('Accuracy')
173 plt.legend()
174 plt.title('Training and Validation Accuracy')
175 plt.show()
176
177 # Plot the training and validation loss graph
178 plt.figure(figsize=(8, 6))
179 plt.plot(history.history['loss'], label='Training Loss')
180 plt.plot(history.history['val_loss'], label='Validation Loss')
181 plt.xlabel('Epoch')
182 plt.ylabel('Loss')
183 plt.legend()
184 plt.title('Training and Validation Loss')
185 plt.show()
```

Listagem 7: Código-fonte do ficheiro `6_LiveTest.py` com sintaxe colorida

```
1 import os
2 from datetime import datetime
3
4 import librosa
5 import matplotlib.pyplot as plt
6 import numpy as np
```

```
import scipy.io.wavfile as wav
import sounddevice as sd
from tensorflow import keras
from tensorflow.keras.preprocessing import image

# Constants
PATH_MODEL = "./LIVETEST/LM_Inception_UL20_BEST.h5"
OUTPUT_RECORDING = "Recordings"
OUTPUT_IMAGES = "Images"
OUTPUT_LOGS = "Logs"
SHIFT = 200
WINDOW_SIZE = 3000
HOP_LENGTH = 8
N_MELS = 256
THRESHOLD = 0.8
DURATION = 2.5
# class_labels = ["0_0", "0_1", "0_2", "1_1", "2_1", "2_2", "3_1", "4_1", "4_2",
↪  "5_1",
# "5_2", "6_1", "7_1", "7_2", "8_1", "8_2", "9_1", "9_2"]
class_labels = ["Unk", "ze", "ro", "um", "do", "is", "tre", "qua", "tro", "cin",
"co", "sei", "se", "te", "oi", "to", "no", "ve"]
#CLASS 17
# class_labels = ["0_1", "0_2", "1_1", "2_1", "2_2", "3_1", "4_1", "4_2", "5_1",
#                 "5_2", "6_1", "7_1", "7_2", "8_1", "8_2", "9_1", "9_2"]

# Get the current directory
current_directory = os.getcwd()
# Specify the path to the "LIVETEST" folder
base_folder_path = os.path.join(current_directory, "LIVETEST")

def record_voice(duration):
    timestamp = datetime.now().strftime("%Y%m%d%H%M%S")
    directory = os.path.join(base_folder_path, OUTPUT_RECORDING)
    filename = os.path.join(directory, f"{timestamp}.wav")

    if not os.path.exists(directory):
        os.makedirs(directory, exist_ok=True)

    print("Recording...")
    fs = 44100
    recording = sd.rec(int(duration * fs), samplerate=fs, channels=2,
    ↪  dtype='int16')
    sd.wait()
    print("Recording finished")

    print("Saving to", filename)
    wav.write(filename, fs, recording)

    directory = os.path.join(base_folder_path, OUTPUT_IMAGES)
    output_folder = os.path.join(directory, timestamp)
    create_mel_spectrogram(filename, output_folder, timestamp)
    predict_images(output_folder, timestamp)

```

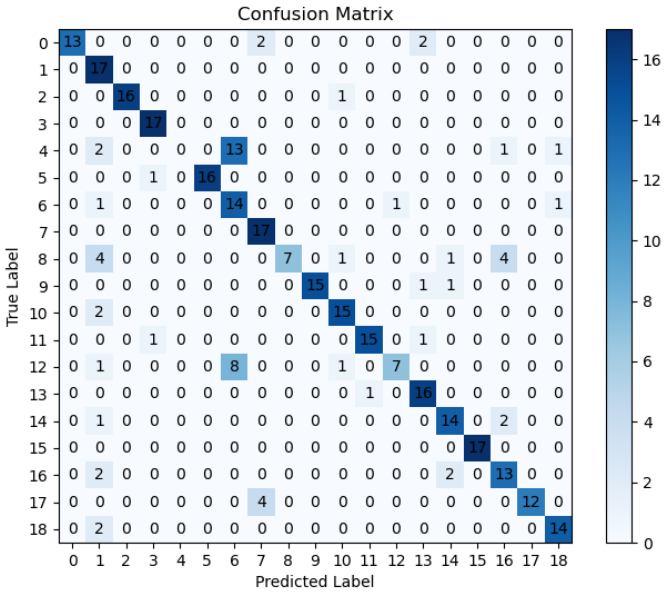
```
58  def create_mel_spectrogram(wav_file, output_folder, timestamp,
    ↪  window_size=WINDOW_SIZE):
59      if not os.path.exists(output_folder):
60          os.makedirs(output_folder, exist_ok=True)
61
62      y, sr = librosa.load(wav_file, sr=None)
63      mel_spectrogram = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=N_MELS,
        ↪  hop_length=HOP_LENGTH)
64      norm_melspec = librosa.power_to_db(mel_spectrogram, ref=np.max)
65
66      num_time_steps = norm_melspec.shape[1]
67      print("Saving images...")
68
69      for i in range(0, num_time_steps - window_size, SHIFT):
70          segment = norm_melspec[:, i:i + window_size]
71          plt.figure(figsize=(6, 4))
72          librosa.display.specshow(segment, sr=sr, hop_length=HOP_LENGTH,
            ↪  x_axis='time', y_axis='mel')
73          plt.axis('off')
74          plt.subplots_adjust(left=0, right=1, top=1, bottom=0)
75          plt.savefig(os.path.join(output_folder, f'{timestamp}_{i}.png'))
76          plt.close()
77
78  def preprocess_image(img_path, target_size=(400, 600)):
79      img = image.load_img(img_path, target_size=target_size)
80      img_array = image.img_to_array(img)
81      img_array = np.expand_dims(img_array, axis=0)
82      return img_array
83
84  def predict_image_class(log_file, model, img_path, class_labels):
85      img = preprocess_image(img_path)
86      probabilities = model.predict(img)[0]
87      predicted_class = np.argmax(probabilities)
88
89      if np.max(probabilities) > THRESHOLD:
90          predicted_class_label = class_labels[predicted_class]
91          print("Predicted Class:", predicted_class_label)
92          log_file.write(f"Predicted Class: {predicted_class_label}\n")
93
94          for i, prob in enumerate(probabilities):
95              print(f"{class_labels[i]}: {prob * 100:.2f}%")
96              log_file.write(f"{class_labels[i]}: {prob * 100:.2f}% ")
97
98          return predicted_class_label
99      else:
100         return "N/a"`
101
102 def predict_images(folder_path, timestamp):
103     print("Predicting images...")
104     predicted_classes = []
105     predicted_words = []
106     prev_predicted_class = ""
107
108     log_directory = os.path.join(base_folder_path, OUTPUT_LOGS)
```

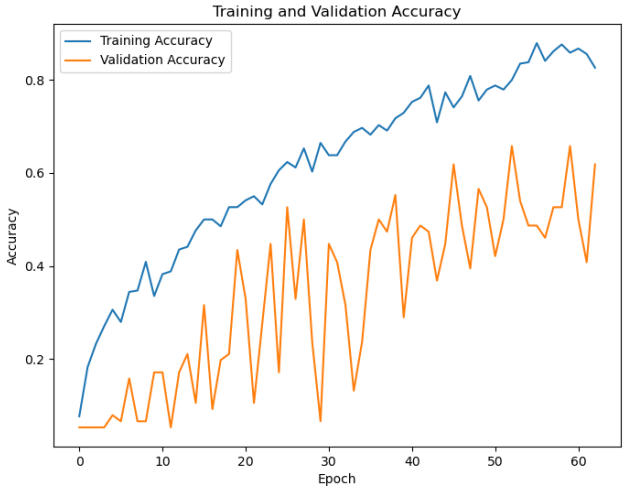
```
109     if not os.path.exists(log_directory):
110         os.makedirs(log_directory, exist_ok=True)
111 
112     log_file_path = os.path.join(log_directory, f"{timestamp}.txt")
113     with open(log_file_path, "w") as log_file:
114         for filename in os.listdir(folder_path):
115             if filename.endswith(".png") or filename.endswith(".jpg"):
116                 image_path = os.path.join(folder_path, filename)
117                 log_file.write(f"For {filename}:\n")
118                 predicted_class = predict_image_class(log_file, model,
                    ↪  image_path, class_labels)
119                 predicted_classes.append(predicted_class)
120 
121                 if predicted_class != prev_predicted_class and predicted_class
                    ↪  != "N/a":
122                     predicted_words.append(predicted_class)
123                     prev_predicted_class = predicted_class
124 
125                 log_file.write("\n")
126 
127         print("Predicted Classes for all images:")
128         print(predicted_classes)
129         log_file.write("Predicted Classes for all images:\n")
130         log_file.write(str(predicted_classes) + "\n")
131 
132         print("Predicted Words:")
133         print(predicted_words)
134         log_file.write("Predicted Words:\n")
135         log_file.write(str(predicted_words) + "\n")
136 
137 try:
138     model = keras.models.load_model(PATH_MODEL)
139 except OSError as e:
140     print(f"Error: {e}")
141     print("Failed to load the model. Please check the PATH_MODEL variable and
    ↪  ensure the model file exists.")
142     raise SystemExit
143 
144 # Example usage
145 record_voice(DURATION)
```

C

APÊNDICE C

Neste apêndice, apresentam-se figuras de gráficos com informações relativas aos 4 modelos treinados na análise do *dataset* de 19 classes. Cada modelo possui um conjunto de três gráficos: matriz de confusão, variação da *accuracy* durante o treino e a validação, e variação da *loss* ao longo do treino e da validação.





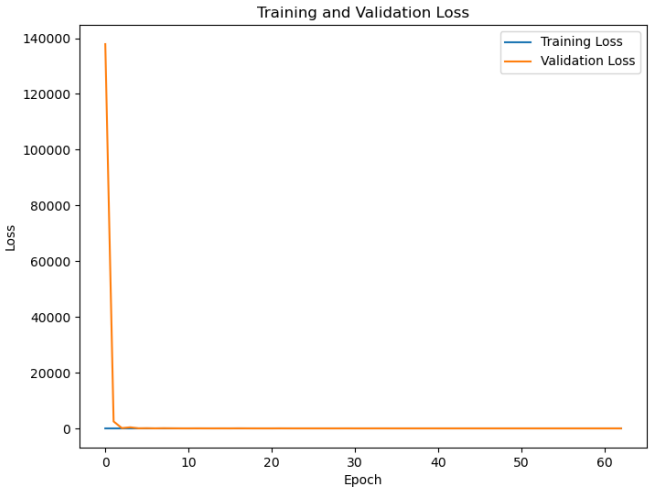
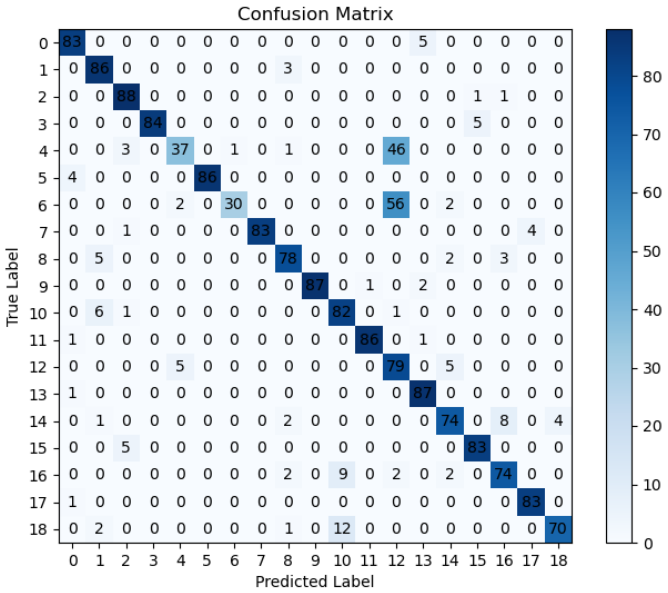


Figura 16: Figuras dos gráficos de treino do modelo 1



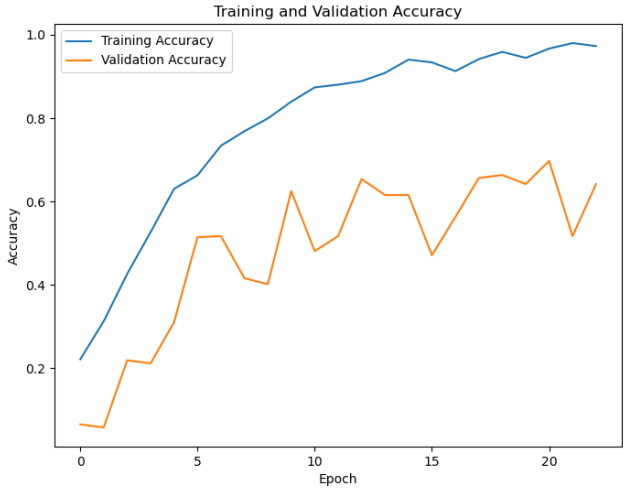
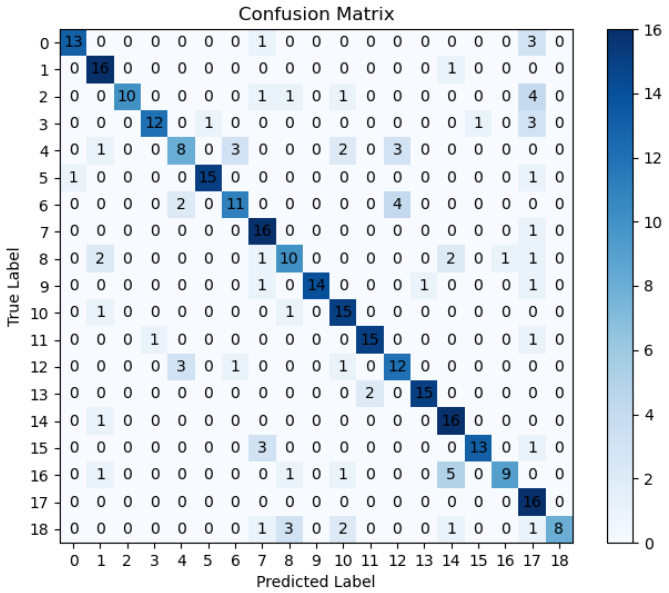
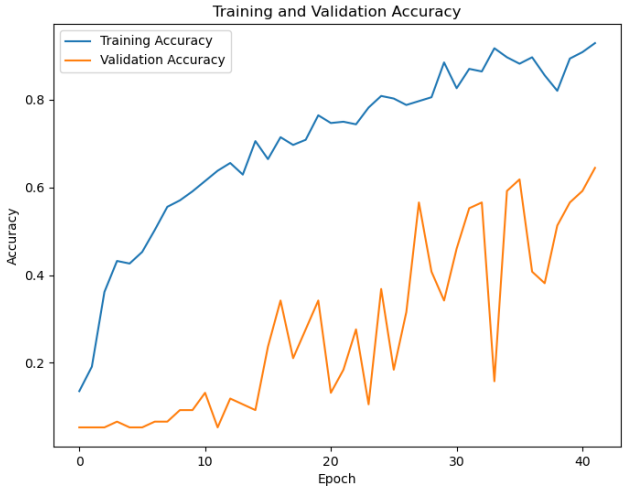


Figura 17: Figuras dos gráficos de treino do modelo 2





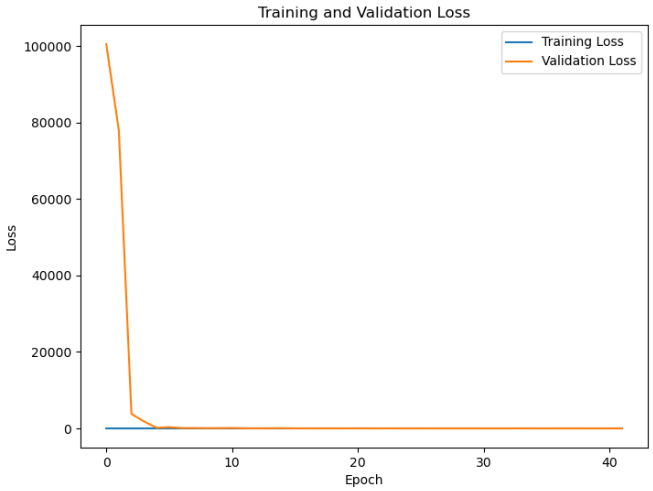
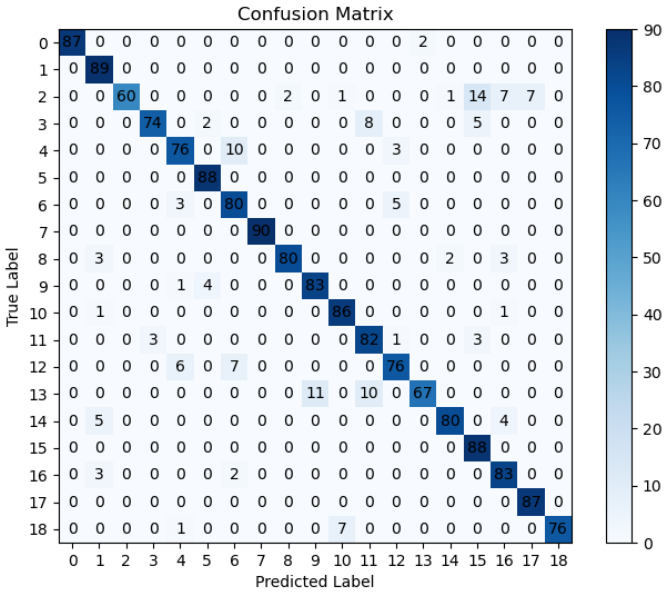


Figura 18: Figuras dos gráficos de treino do modelo 3



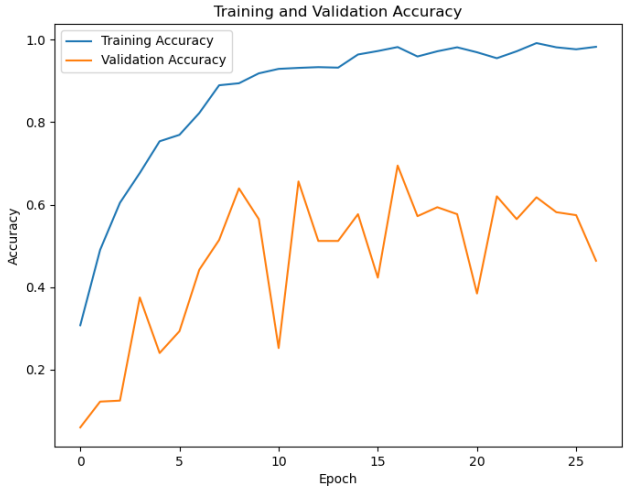
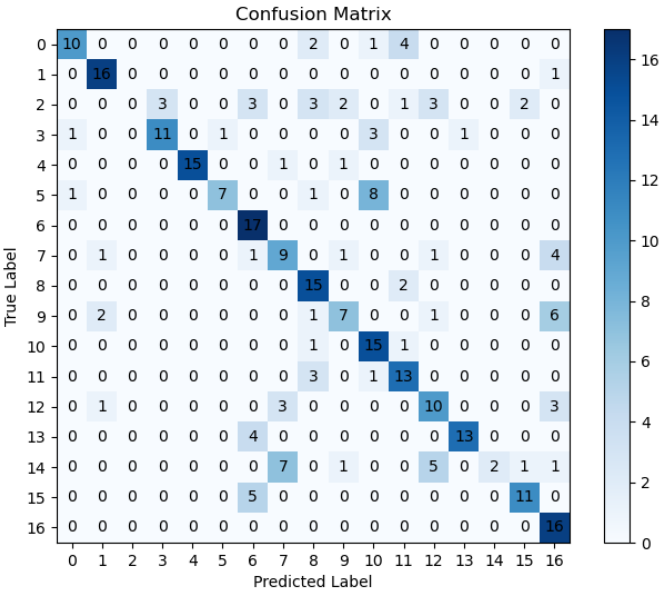


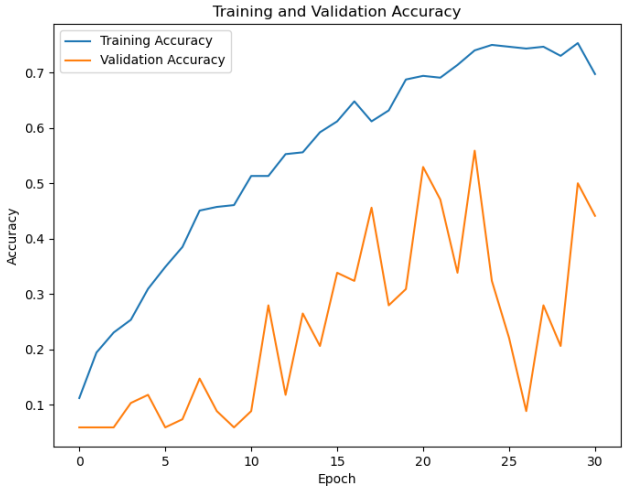
Figura 19: Figuras dos gráficos de treino do modelo 4

D

APÊNDICE D

Neste apêndice, apresentam-se figuras de gráficos com informações relativas aos 4 modelos treinados na análise do *dataset* de 17 classes. Cada modelo possui um conjunto de três gráficos: matriz de confusão, variação da *accuracy* durante o treino e a validação, e variação da *loss* ao longo do treino e da validação.





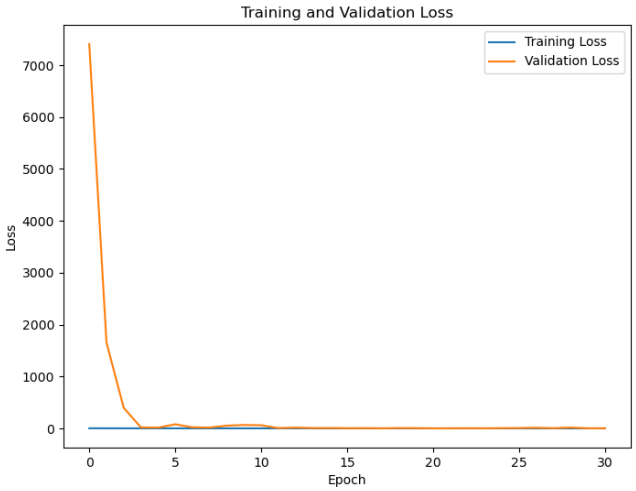
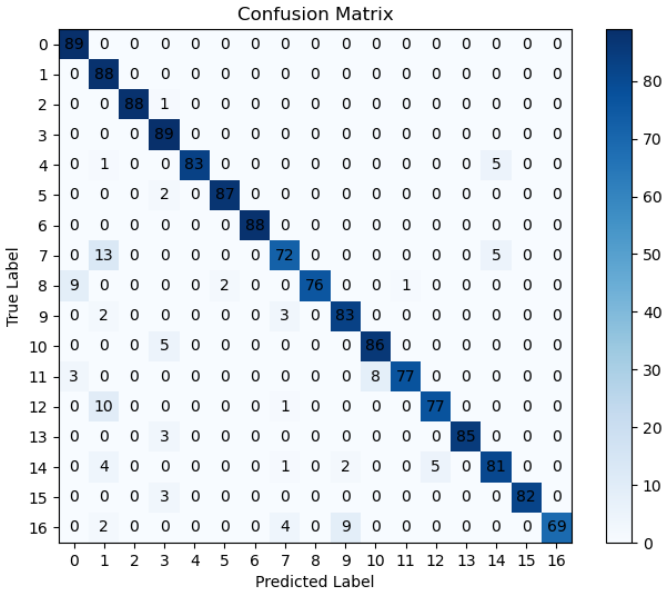


Figura 20: Figuras dos gráficos de treino do modelo 5



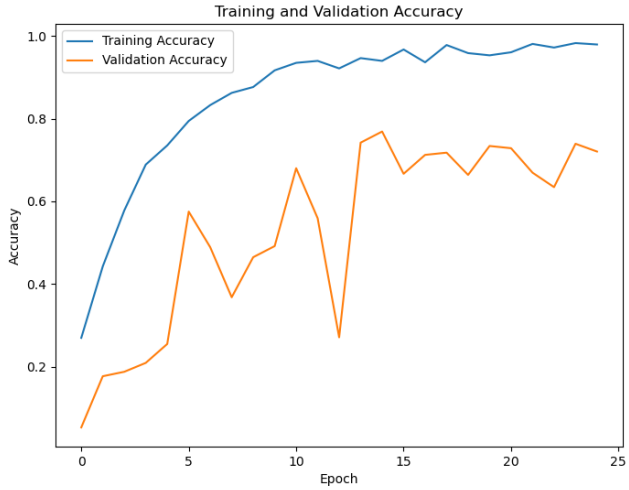
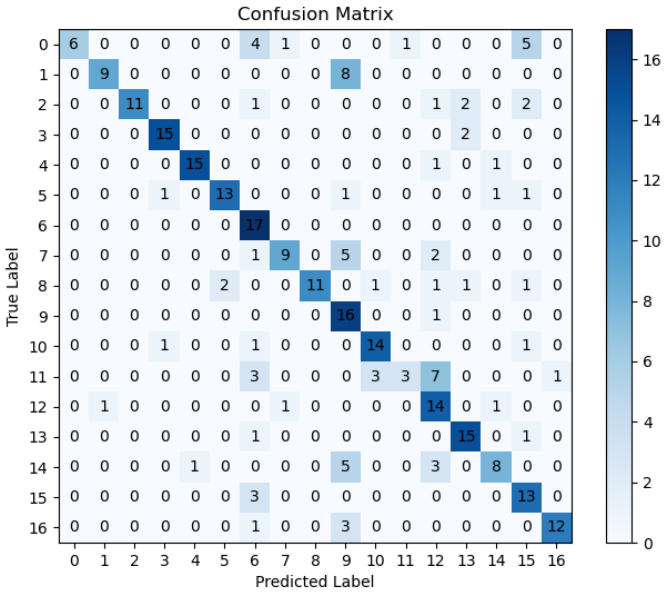
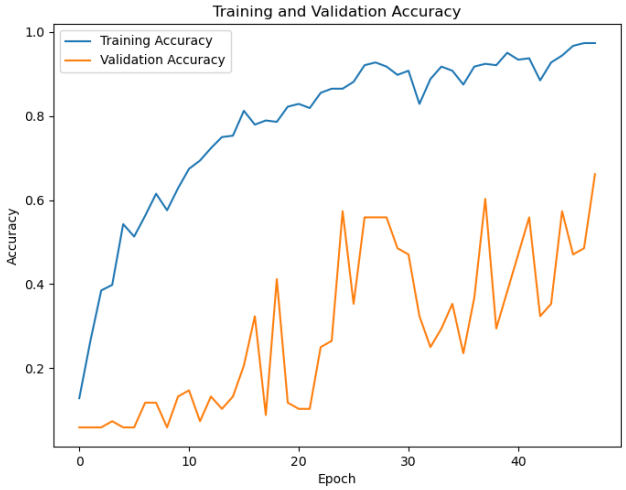


Figura 21: Figuras dos gráficos de treino do modelo 6





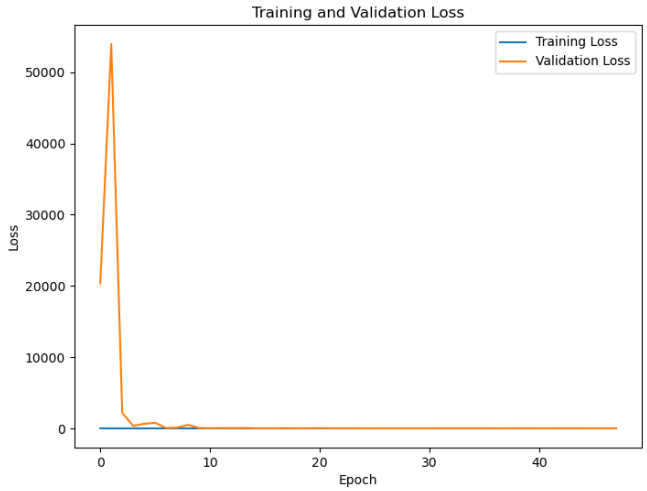
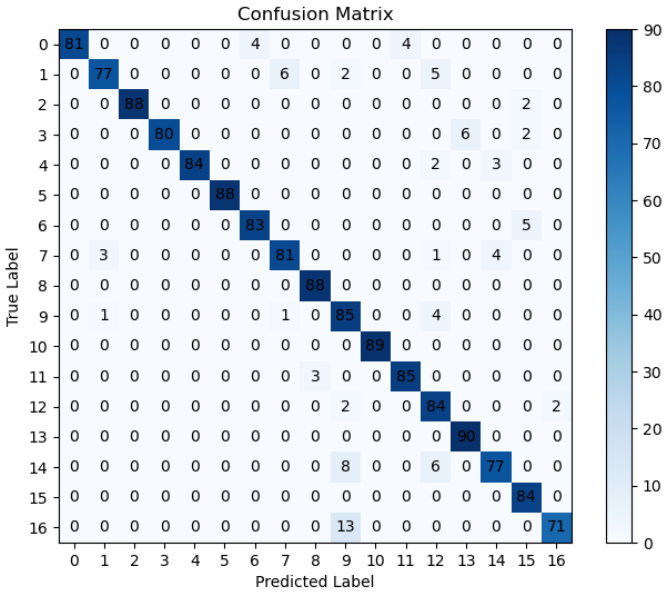


Figura 22: Figuras dos gráficos de treino do modelo 7



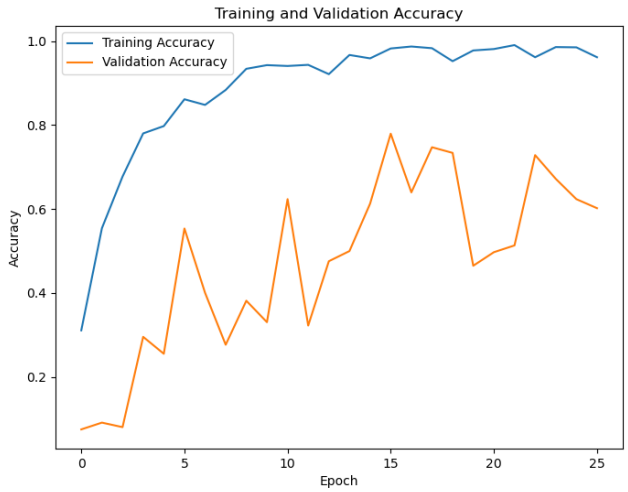
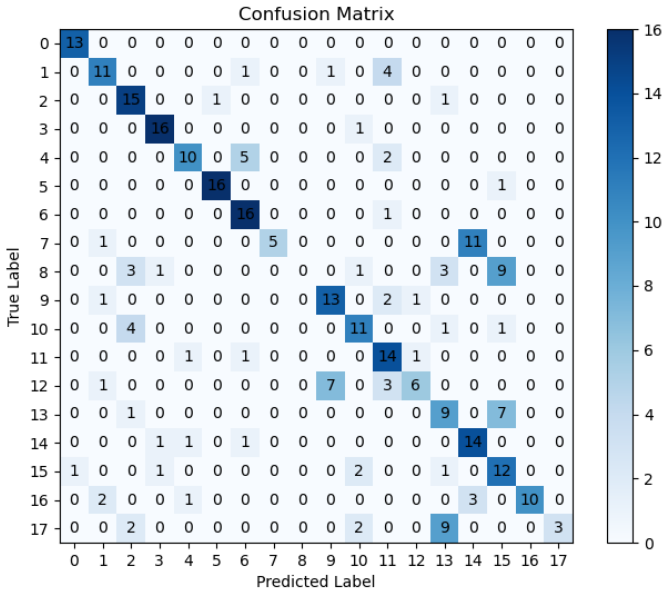


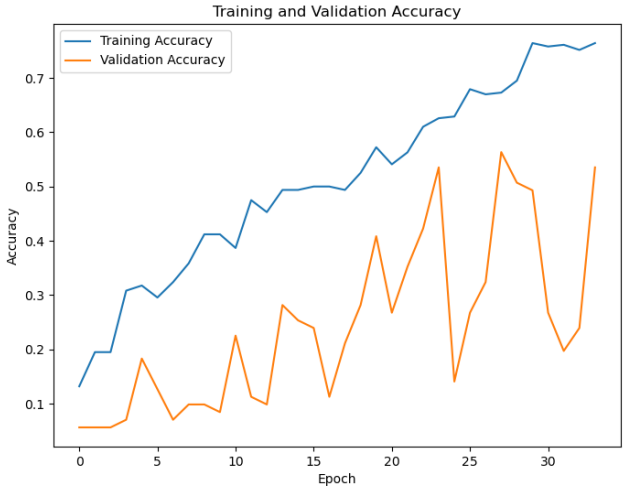
Figura 23: Figuras dos gráficos de treino do modelo 8

E

APÊNDICE E

Neste apêndice, apresentam-se figuras de gráficos com informações relativas aos 4 modelos treinados na análise do *dataset* de 18 classes. Cada modelo possui um conjunto de três gráficos: matriz de confusão, variação da *accuracy* durante o treino e a validação, e variação da *loss* ao longo do treino e da validação.





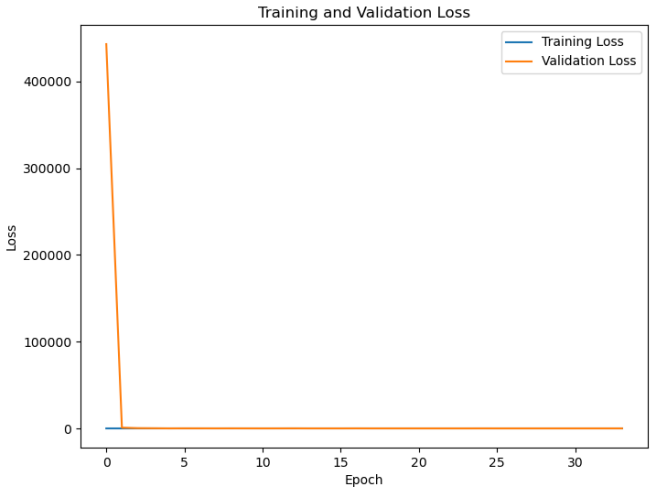
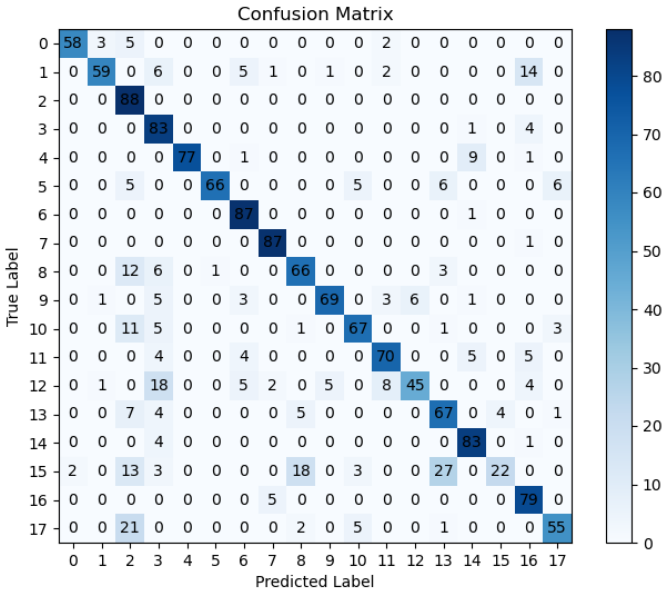
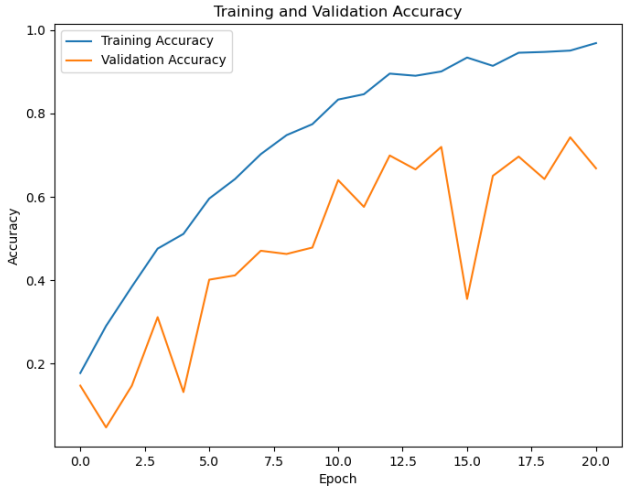


Figura 24: Figuras dos gráficos de treino do modelo 9





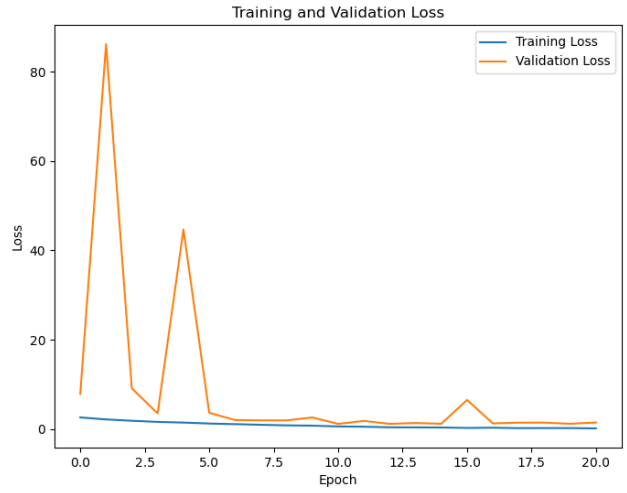
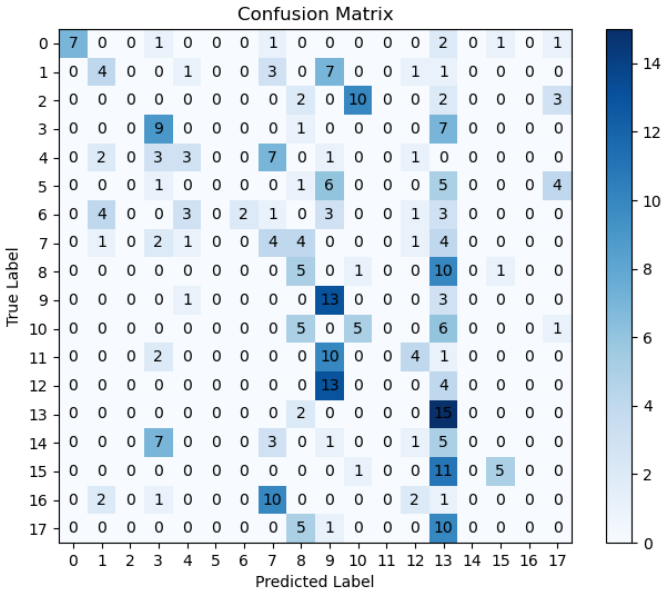
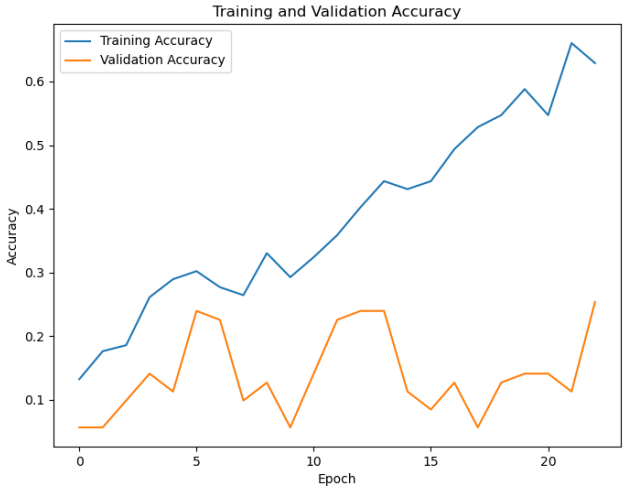


Figura 25: Figuras dos gráficos de treino do modelo 10





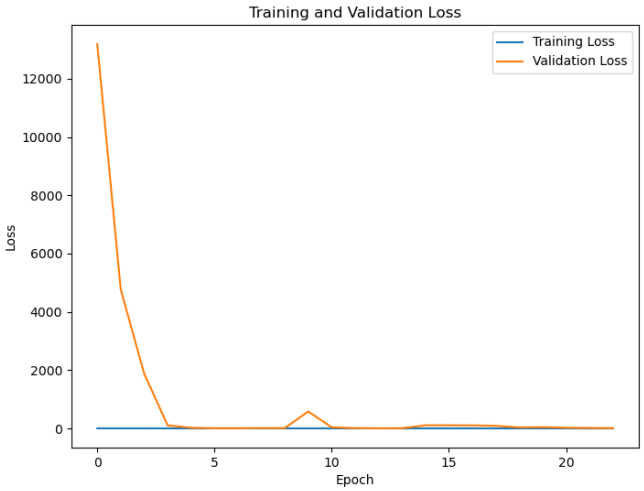
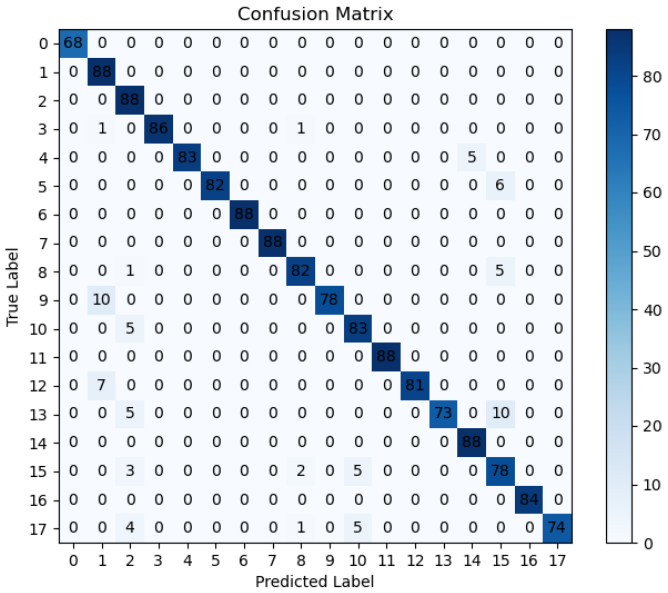


Figura 26: Figuras dos gráficos de treino do modelo 11



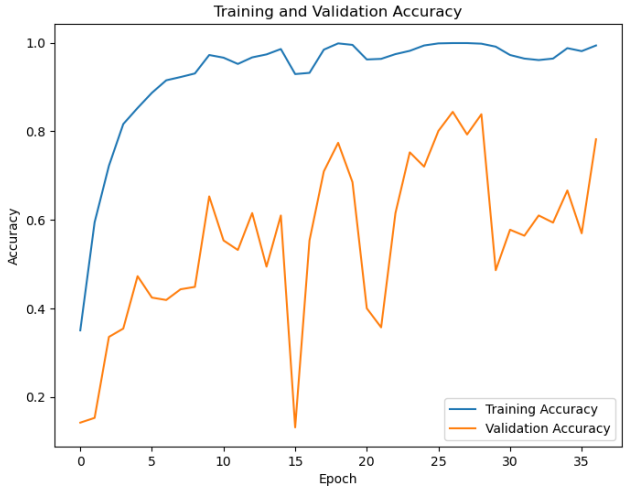
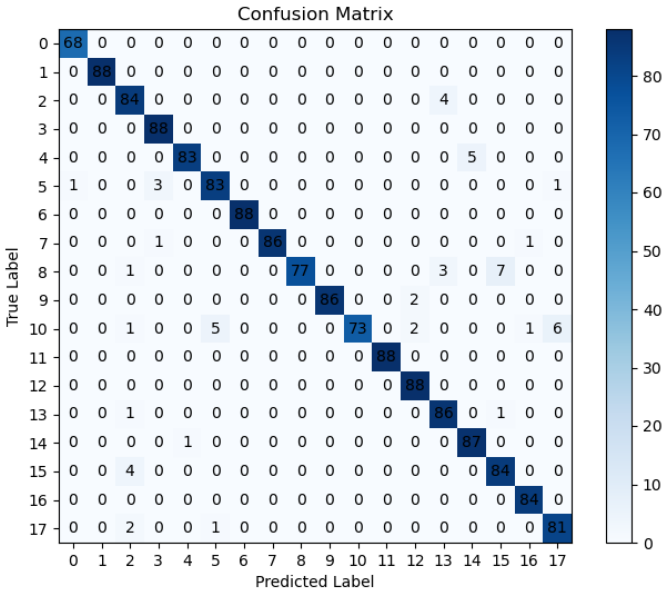


Figura 27: Figuras dos gráficos de treino do modelo 12

F

APÊNDICE F

Neste apêndice, apresentam-se figuras de gráficos com informações relativas aos 6 modelos treinados na análise de modelos pré-treinados. Cada modelo possui um conjunto de três gráficos: matriz de confusão, variação da *accuracy* durante o treino e a validação, e variação da *loss* ao longo do treino e da validação.



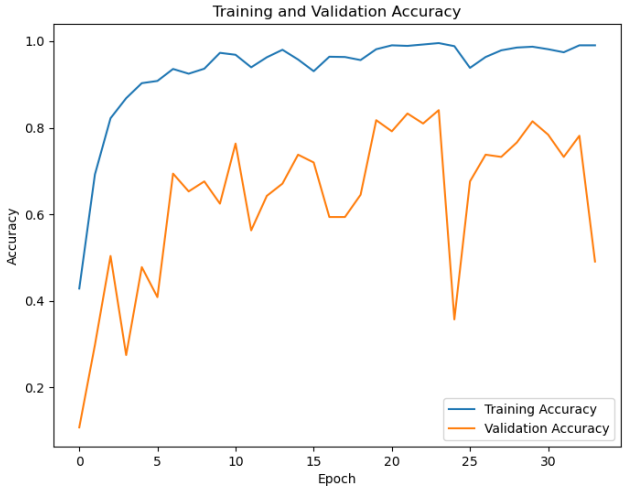
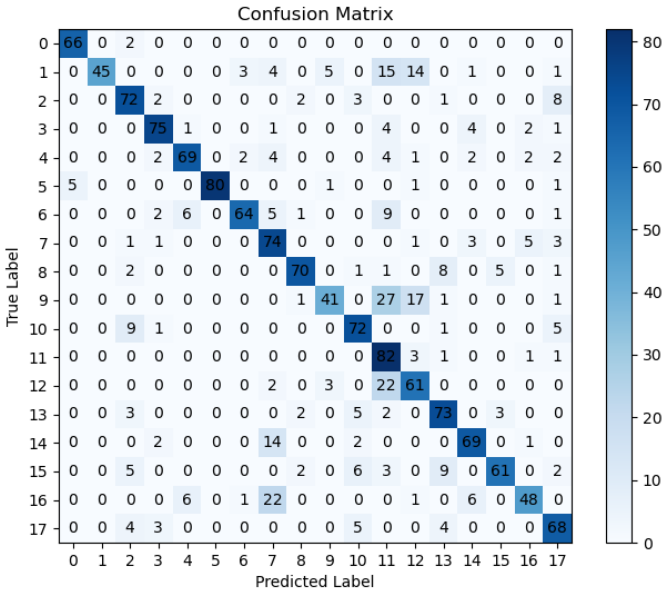


Figura 28: Figuras dos gráficos de treino do modelo InceptionV3



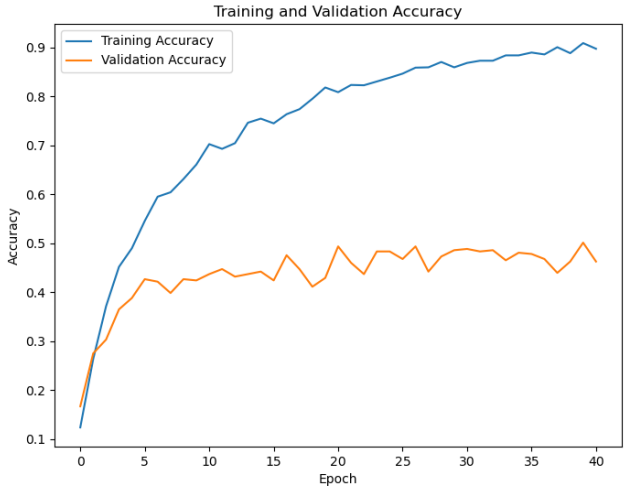
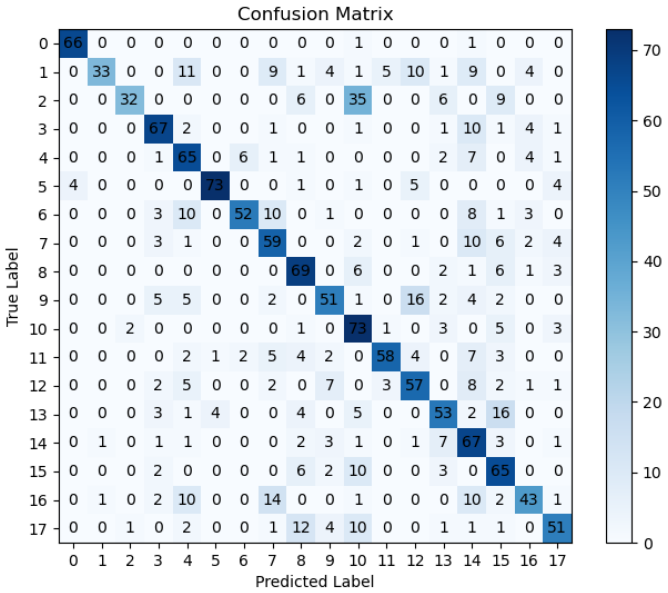
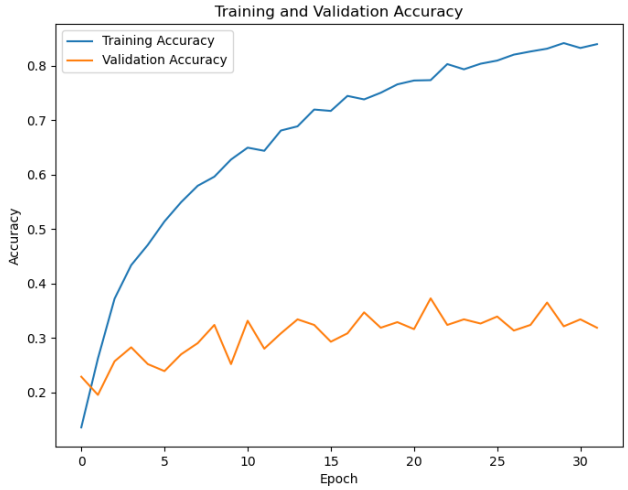


Figura 29: Figuras dos gráficos de treino do modelo VGG16





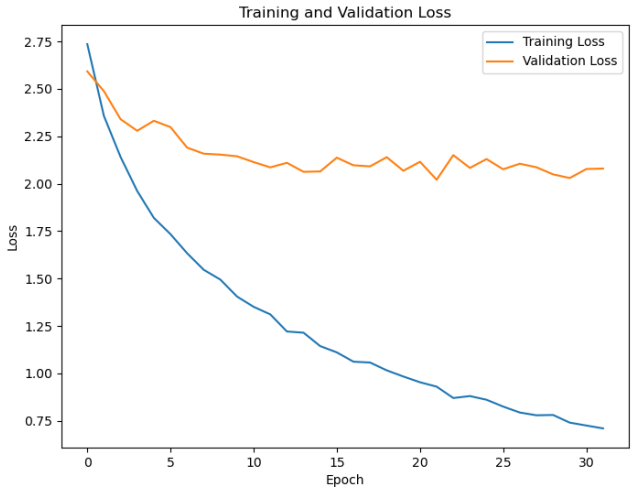
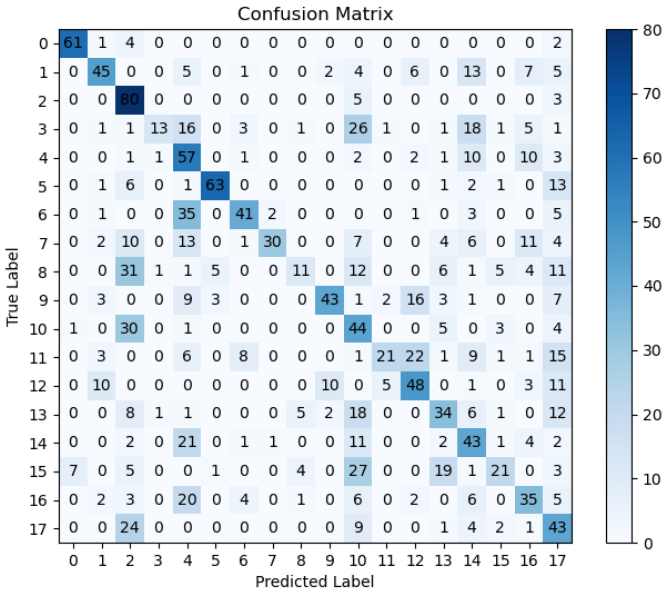
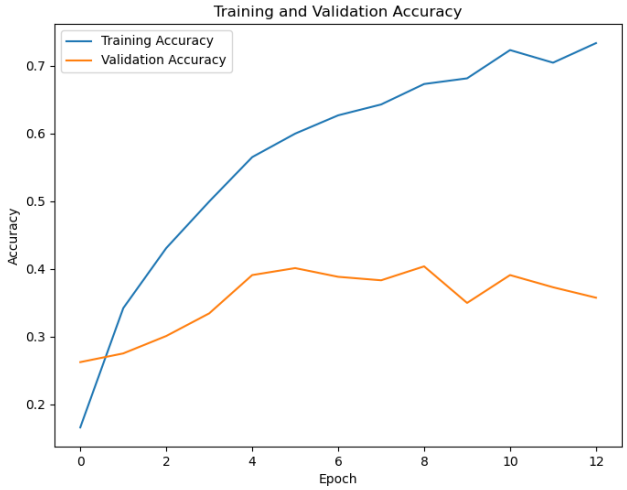


Figura 30: Figuras dos gráficos de treino do modelo MobileNetV2





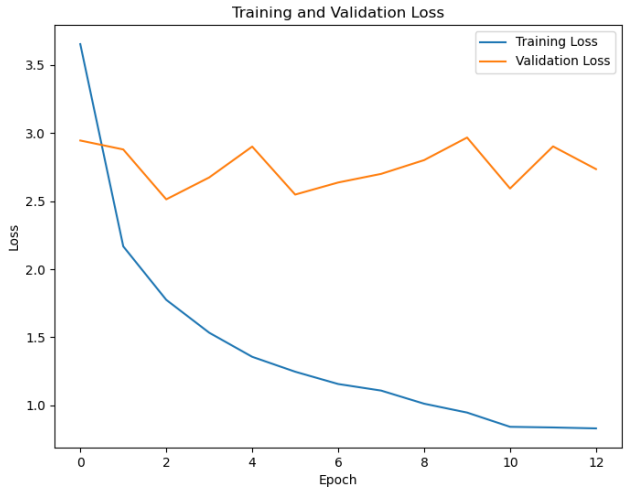
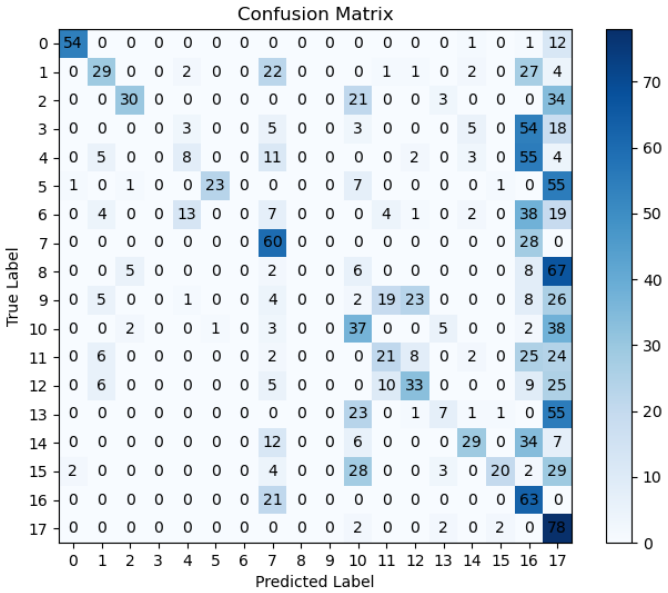


Figura 31: Figuras dos gráficos de treino do modelo Xception



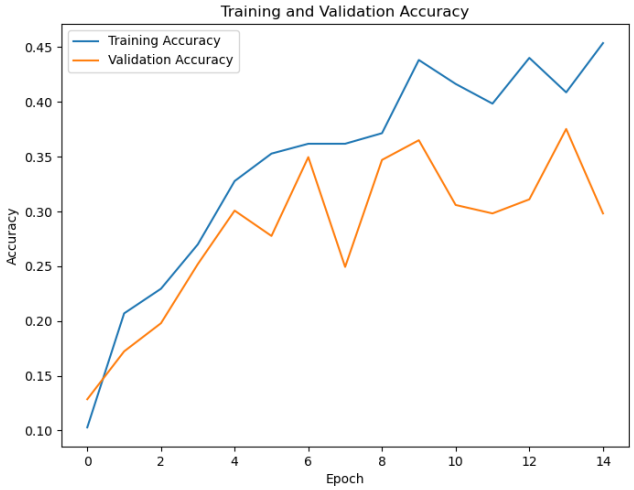
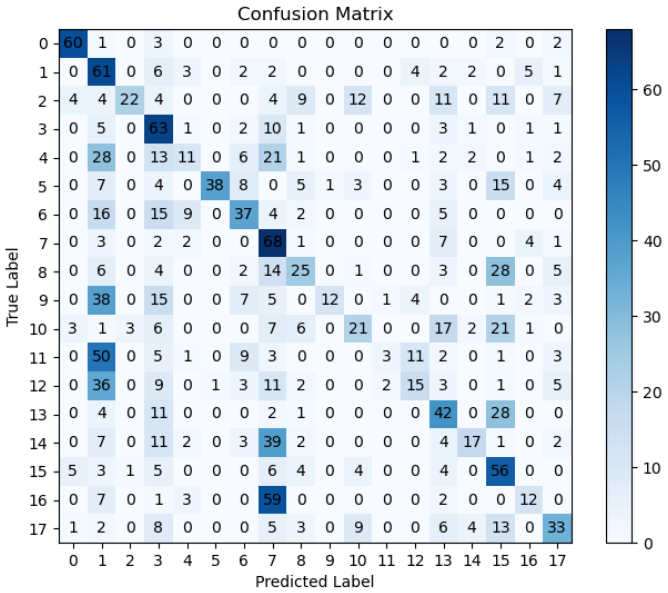
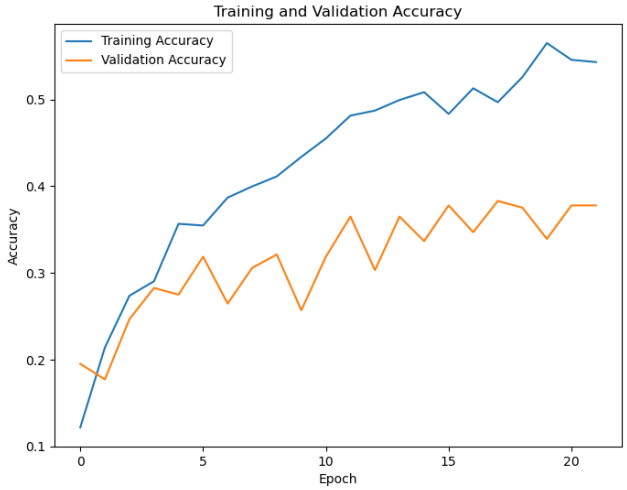


Figura 32: Figuras dos gráficos de treino do modelo ResNet101V2





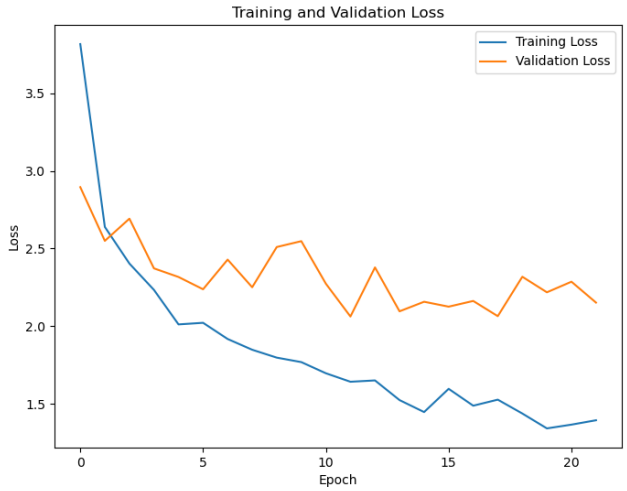
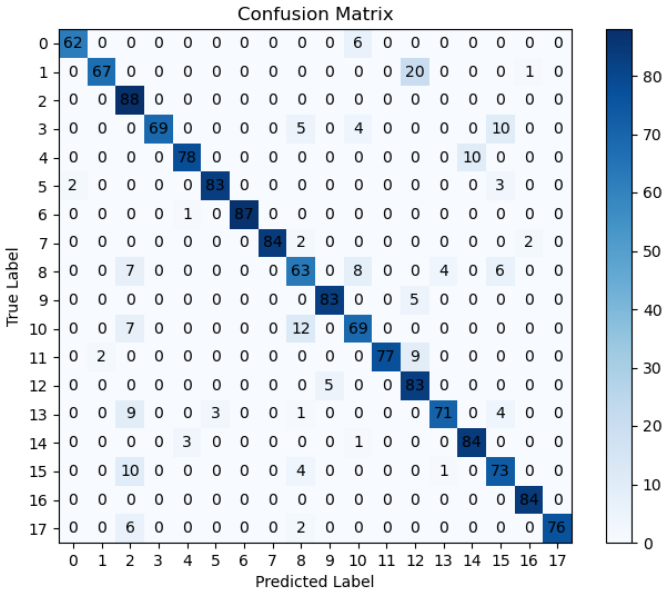


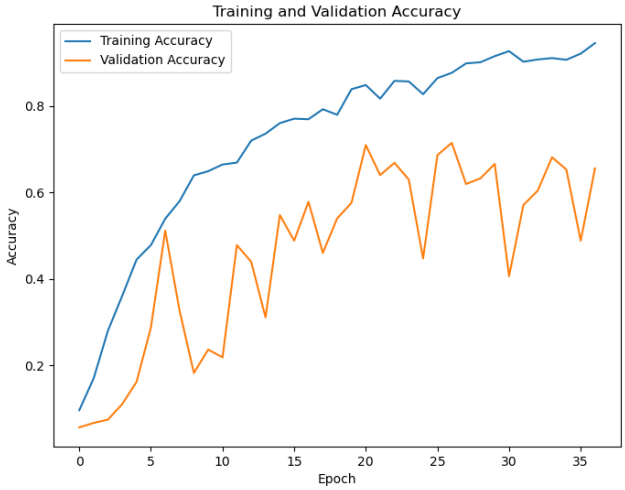
Figura 33: Figuras dos gráficos de treino do modelo DenseNet121

G

APÊNDICE G

Neste apêndice, apresentam-se figuras de gráficos com informações relativas aos 7 modelos treinados na análise de modelos no grupo 1 de *fine-tuning* relativo às alterações de arquitetura de modelos. Cada modelo possui um conjunto de três gráficos: matriz de confusão, variação da *accuracy* durante o treino e a validação, e variação da *loss* ao longo do treino e da validação.





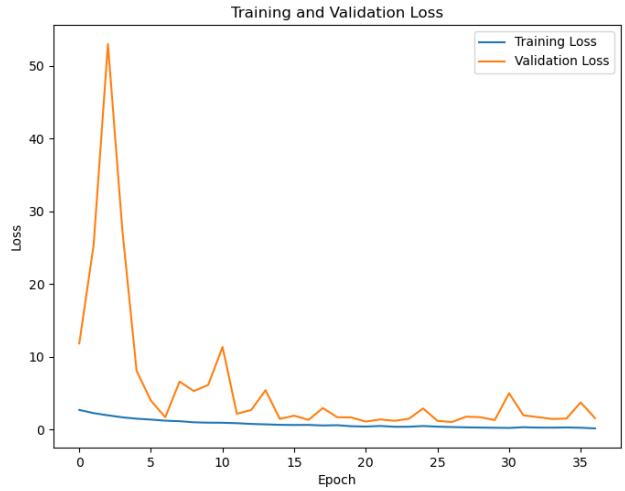
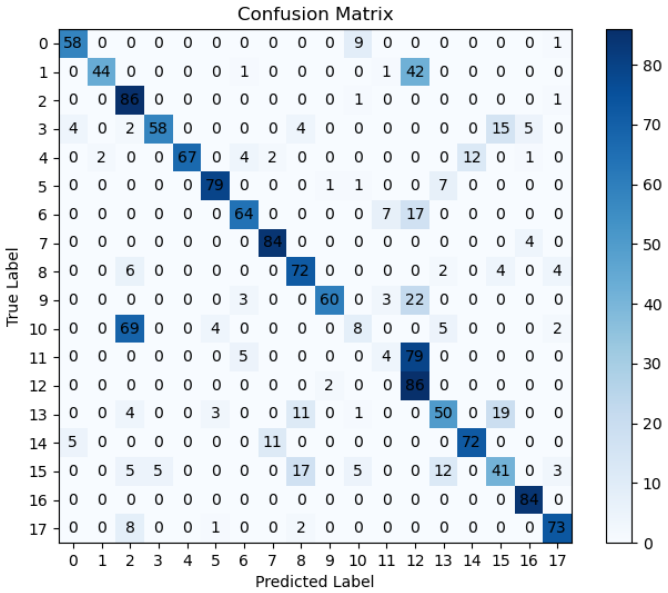


Figura 34: Figuras dos gráficos de treino do modelo InceptionV3_1024_05



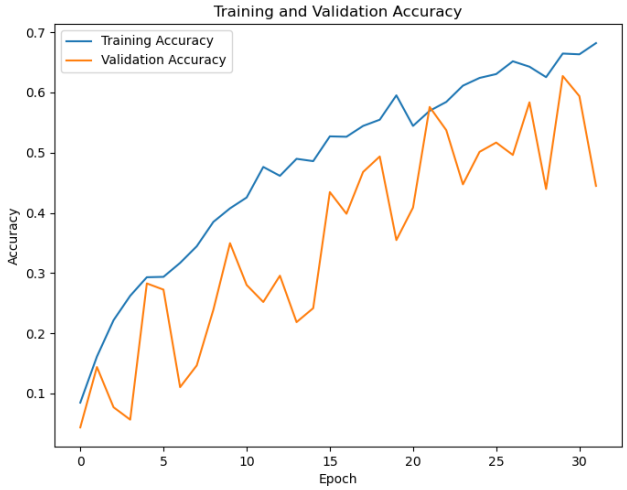
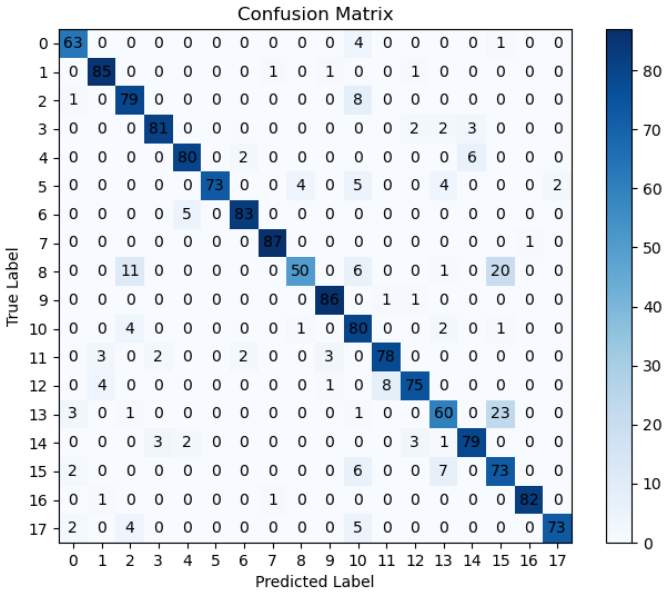


Figura 35: Figuras dos gráficos de treino do modelo InceptionV3_64_02



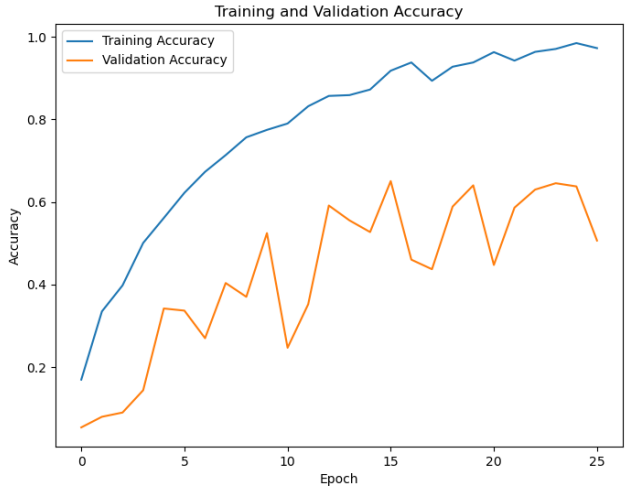
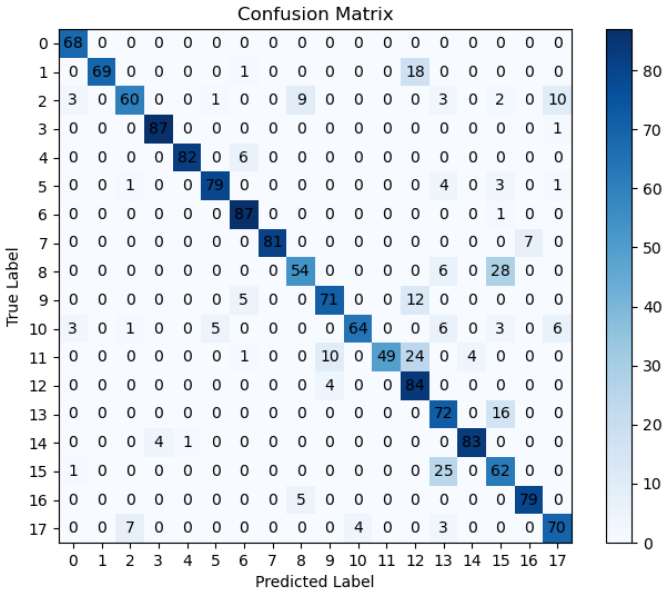
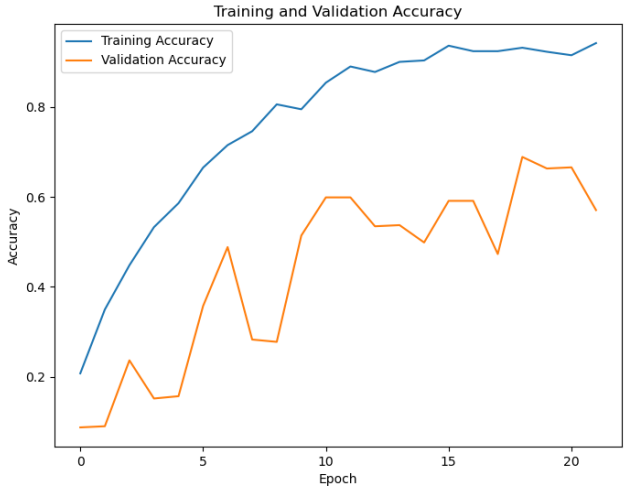


Figura 36: Figuras dos gráficos de treino do modelo RESNET50_1024_05





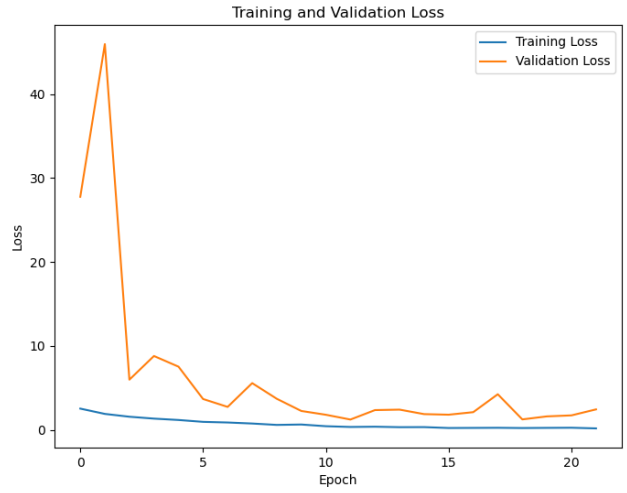
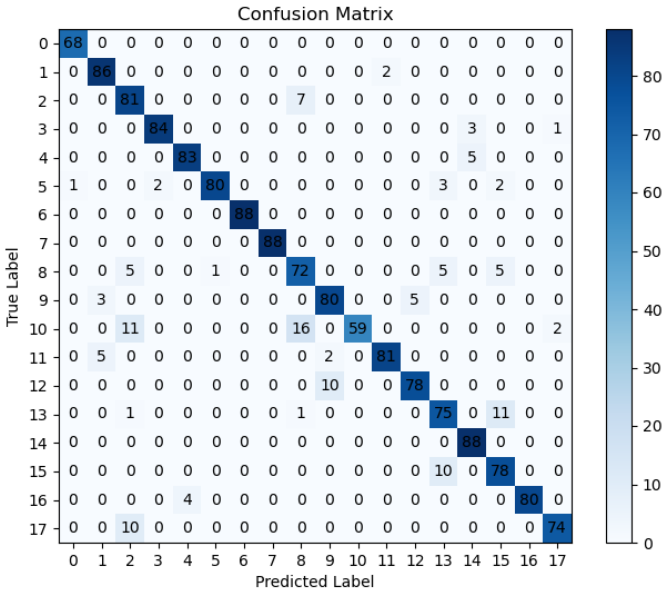
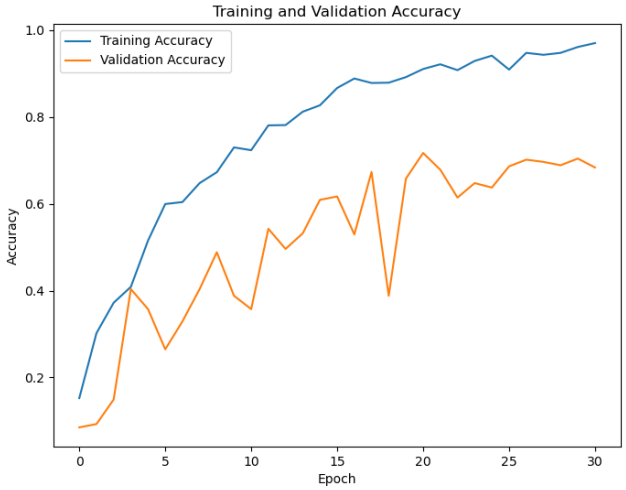


Figura 37: Figuras dos gráficos de treino do modelo RESNET50_64_02





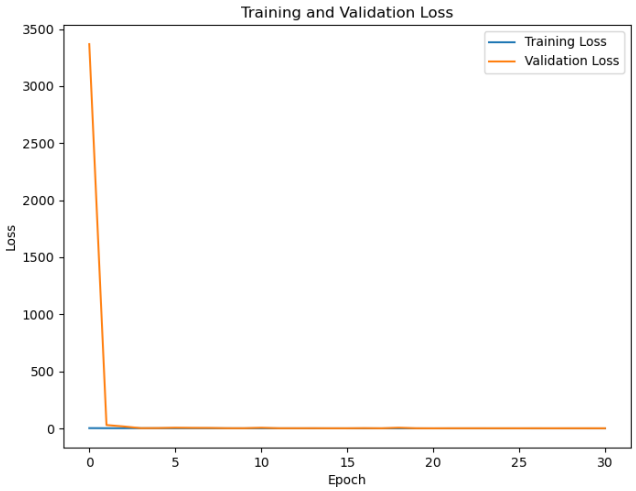
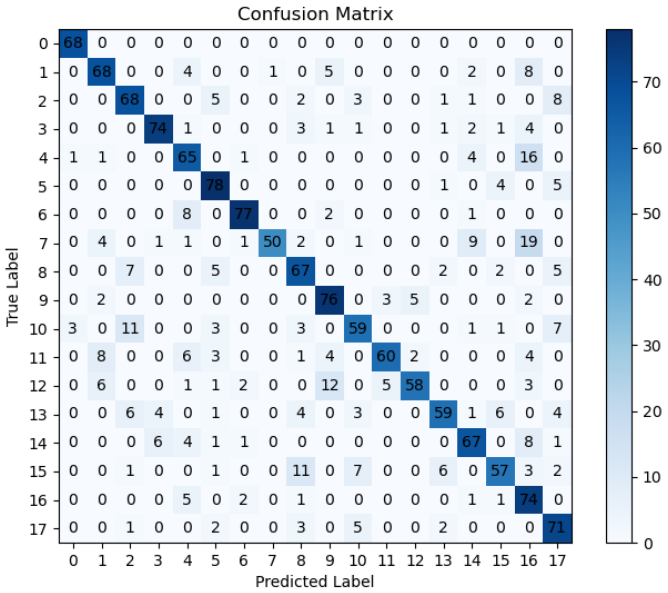
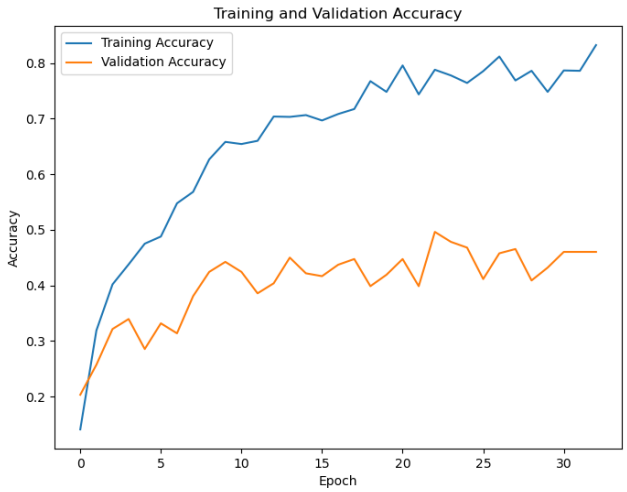


Figura 38: Figuras dos gráficos de treino do modelo InceptionV3_8batch





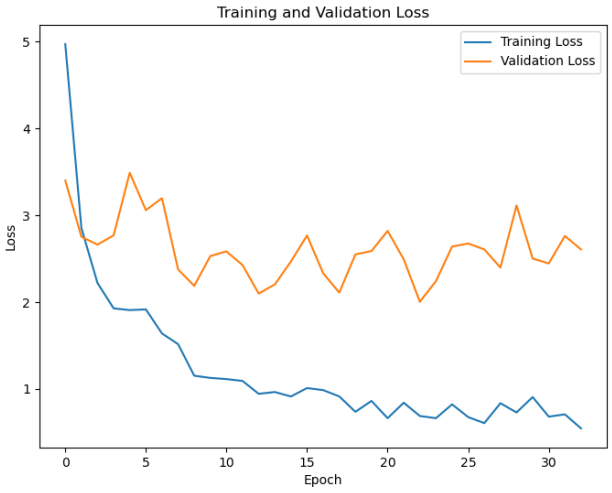
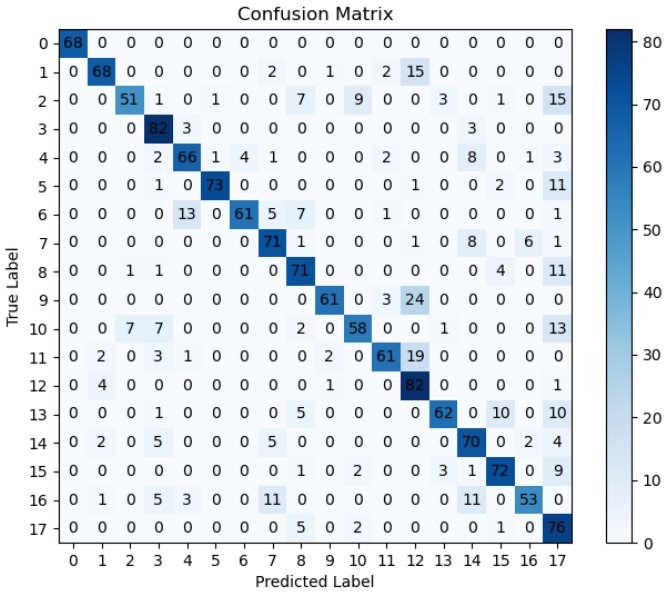


Figura 39: Figuras dos gráficos de treino do modelo InceptionV3_T



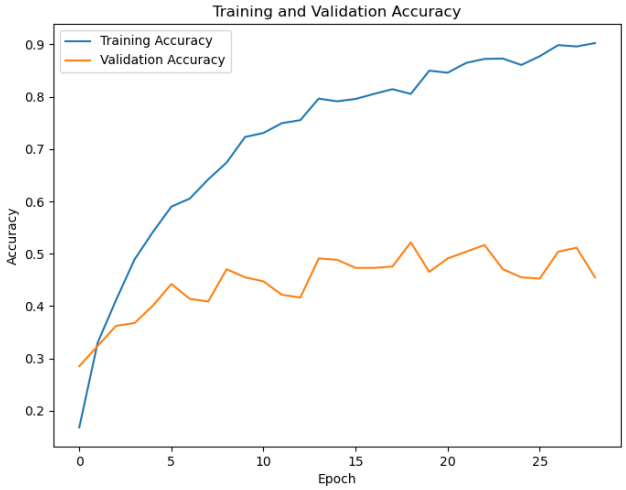
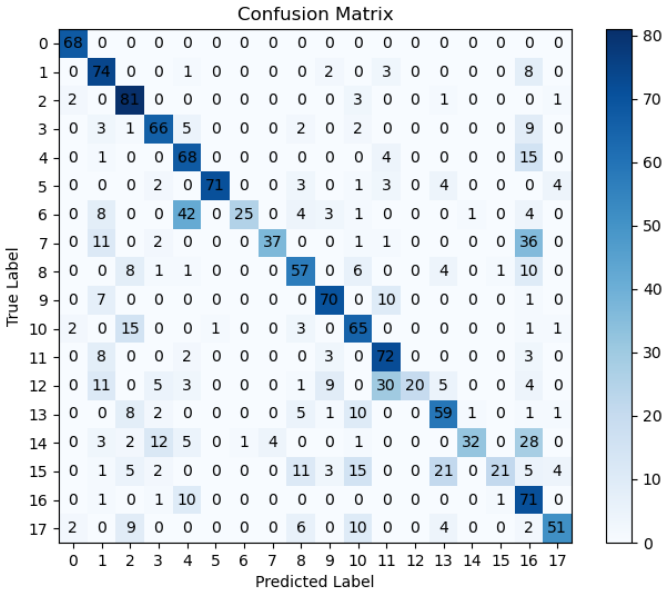


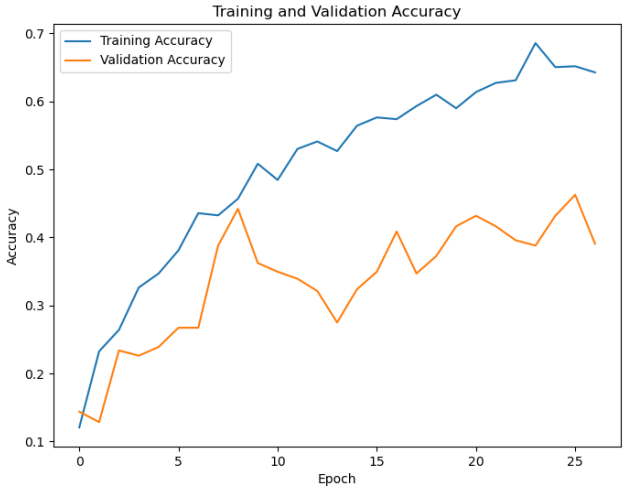
Figura 40: Figuras dos gráficos de treino do modelo RESNET50_16_T

H

APÊNDICE H

Neste apêndice, apresentam-se figuras de gráficos com informações relativas aos 3 modelos treinados na análise de modelos no grupo 1 de *fine-tuning* relativo à utilização de diferentes hiperparâmetros. Cada modelo possui um conjunto de três gráficos: matriz de confusão, variação da *accuracy* durante o treino e a validação, e variação da *loss* ao longo do treino e da validação.





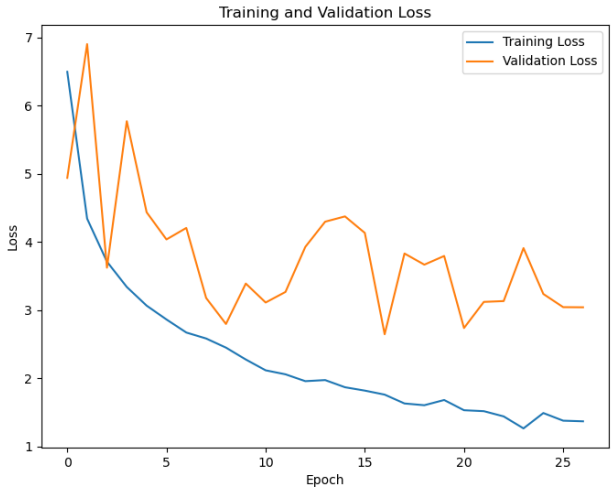


Figura 41: Figuras dos gráficos de treino do modelo InceptionV3_RMS_001

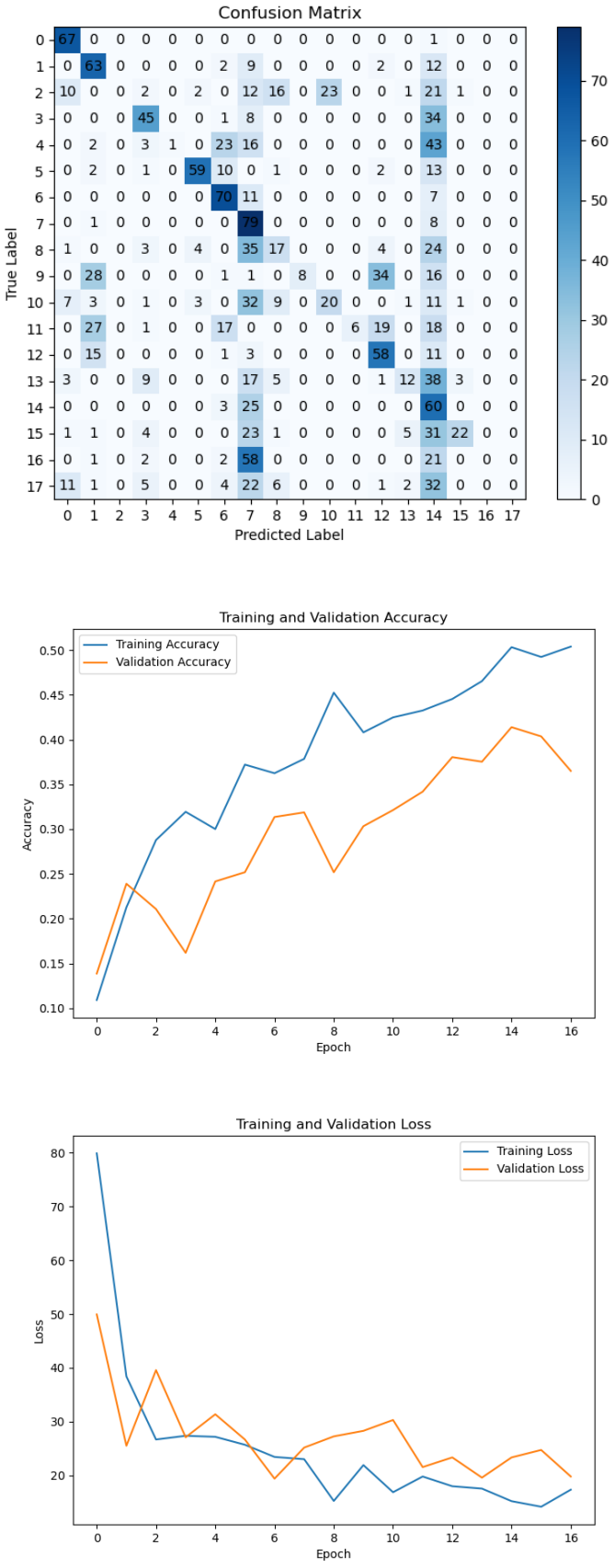
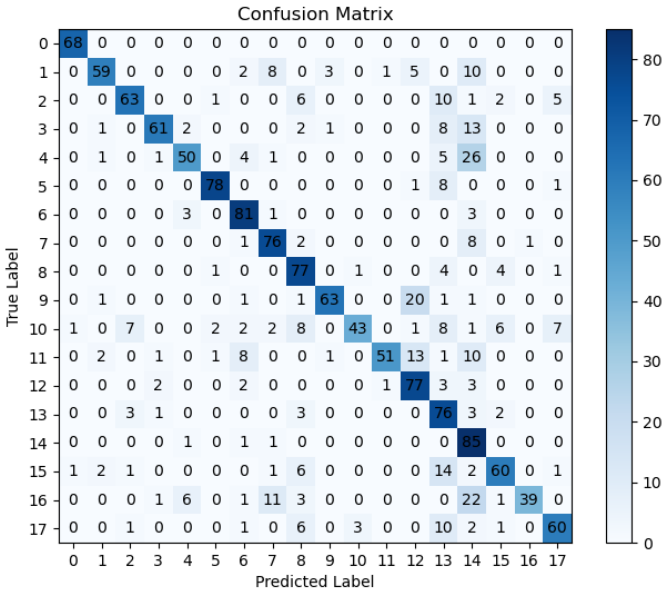


Figura 42: Figuras dos gráficos de treino do modelo InceptionV3_SGD



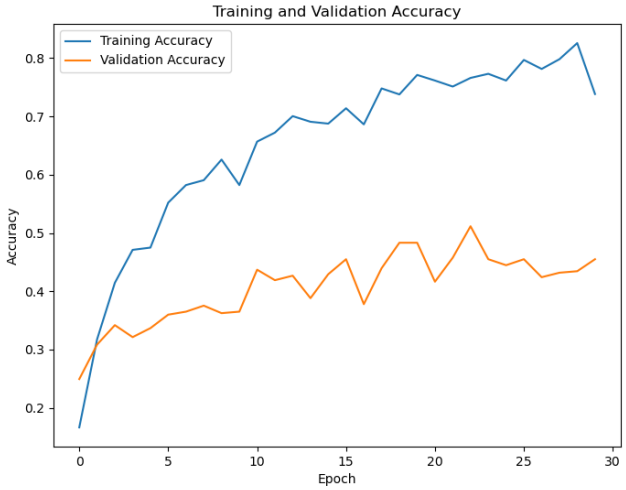
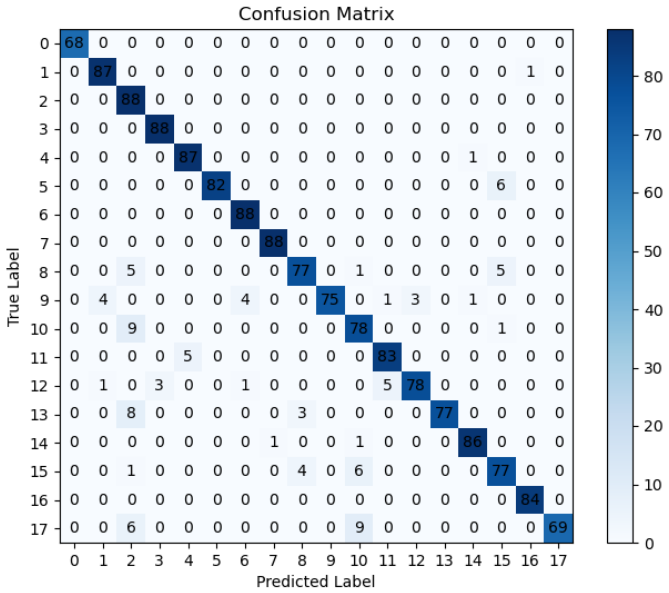


Figura 43: Figuras dos gráficos de treino do modelo InceptionV3_EXPD

I

APÊNDICE I

Neste apêndice, apresentam-se figuras de gráficos com informações relativas aos 2 modelos treinados na análise de modelos no grupo 1 de *fine-tuning* relativo a diferentes métodos de *augmentation*. Cada modelo possui um conjunto de três gráficos: matriz de confusão, variação da *accuracy* durante o treino e a validação, e variação da *loss* ao longo do treino e da validação.



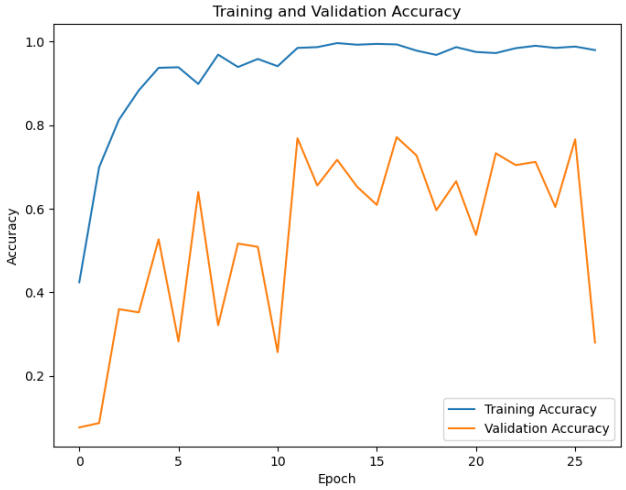
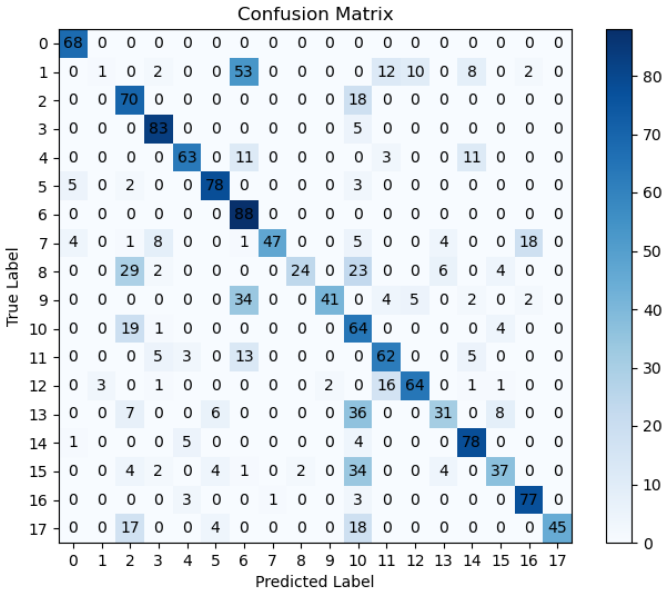
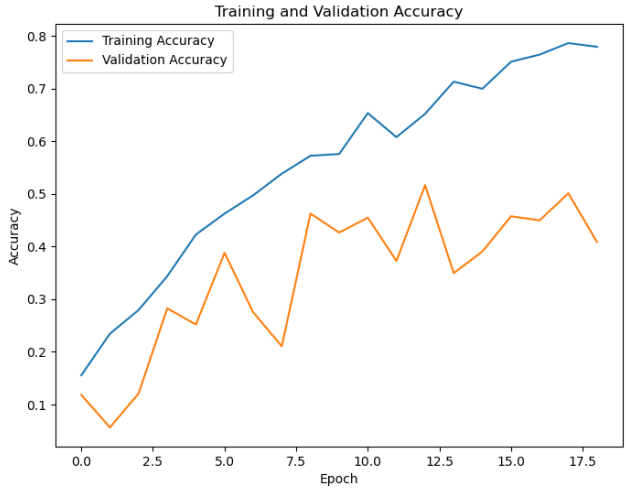


Figura 44: Figuras dos gráficos de treino do modelo InceptionV3_AUG_00





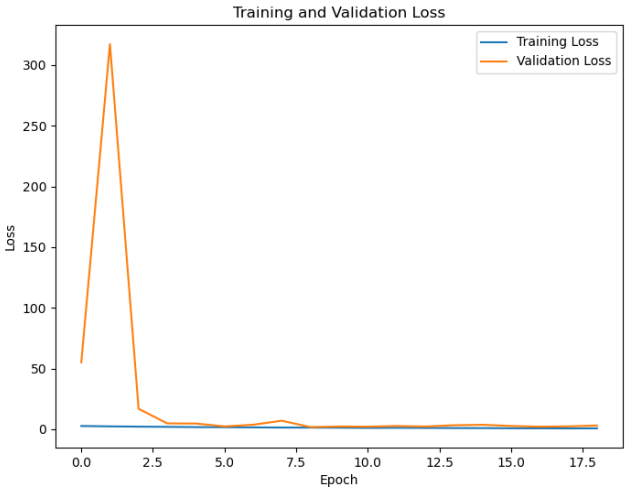


Figura 45: Figuras dos gráficos de treino do modelo InceptionV3_AUG_03

J

APÊNDICE J

Neste apêndice, apresentam-se figuras de gráficos com informações relativas aos 6 modelos treinados na análise de modelos no grupo 2 de *fine-tuning*. Cada modelo possui um conjunto de três gráficos: matriz de confusão, variação da *accuracy* durante o treino e a validação, e variação da *loss* ao longo do treino e da validação.

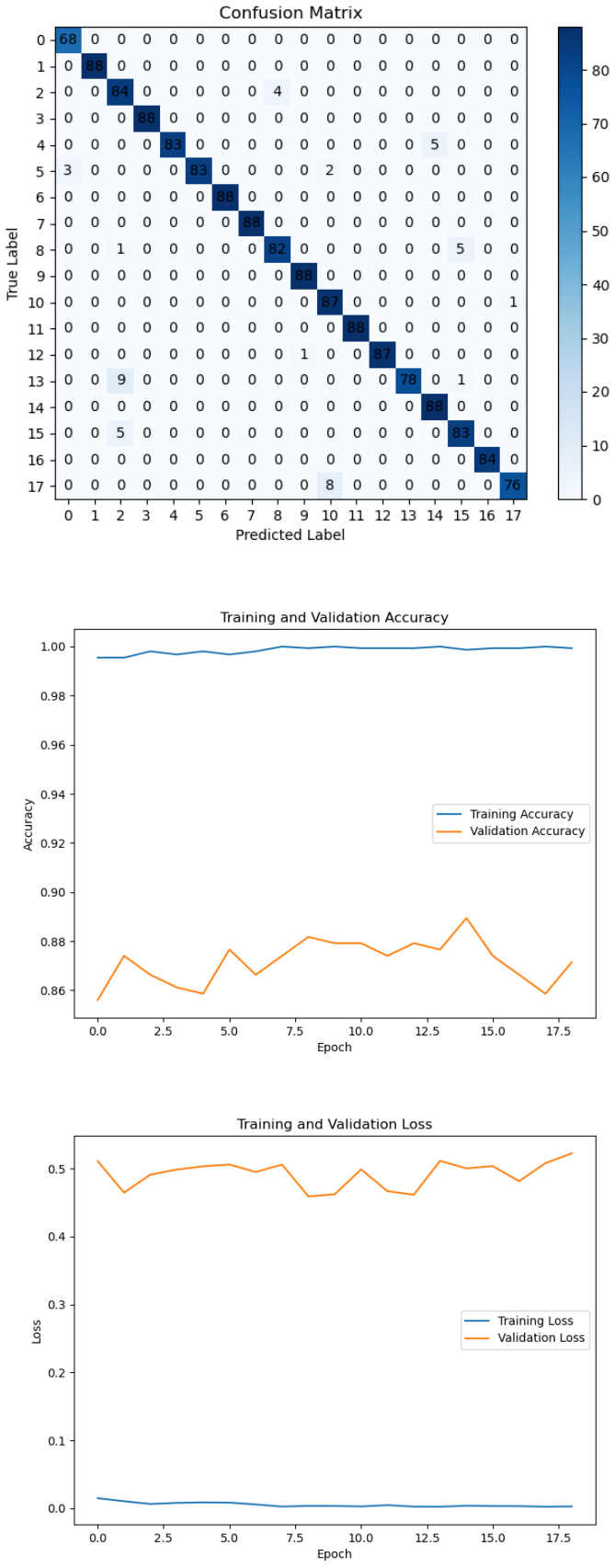
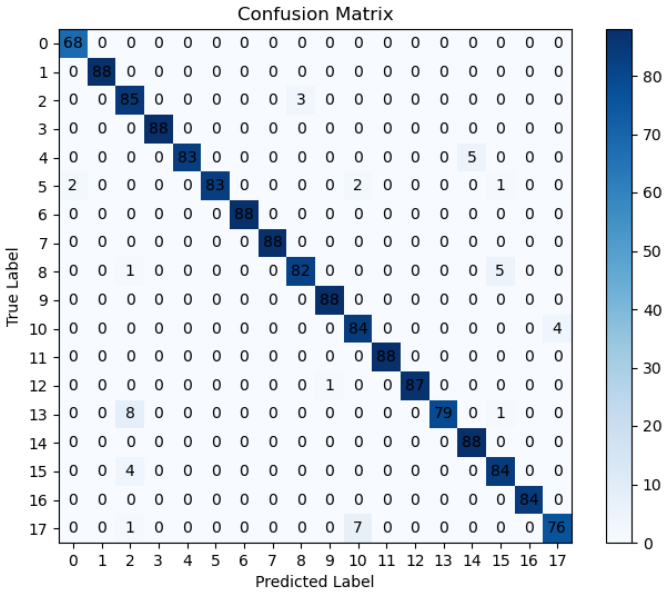


Figura 46: Figuras dos gráficos de treino do modelo Inception_UL10



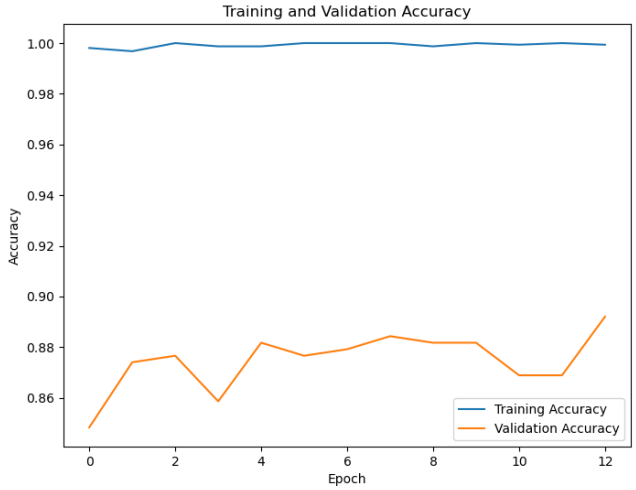
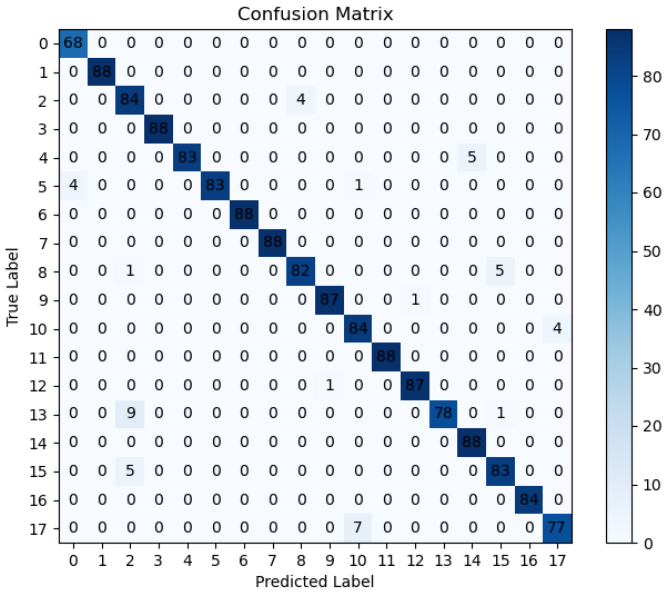


Figura 47: Figuras dos gráficos de treino do modelo Inception_UL20



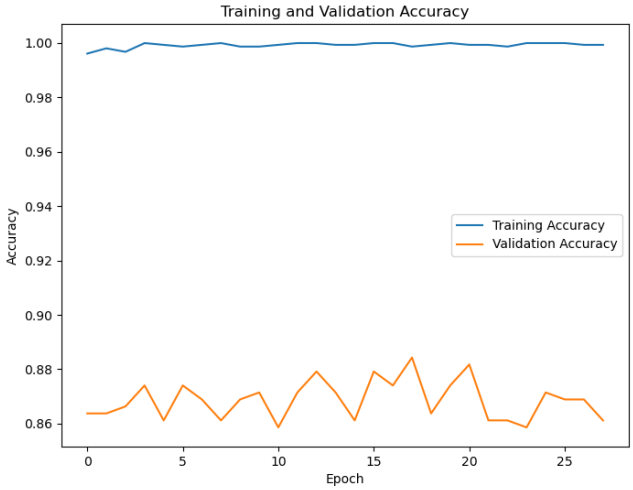
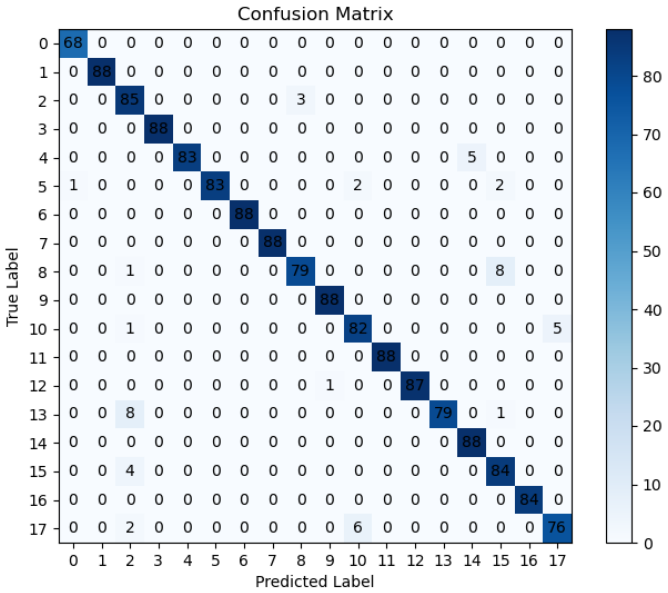
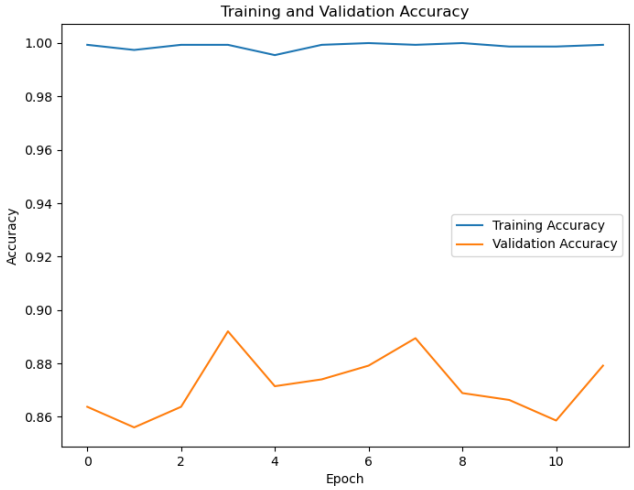


Figura 48: Figuras dos gráficos de treino do modelo Inception_UL30





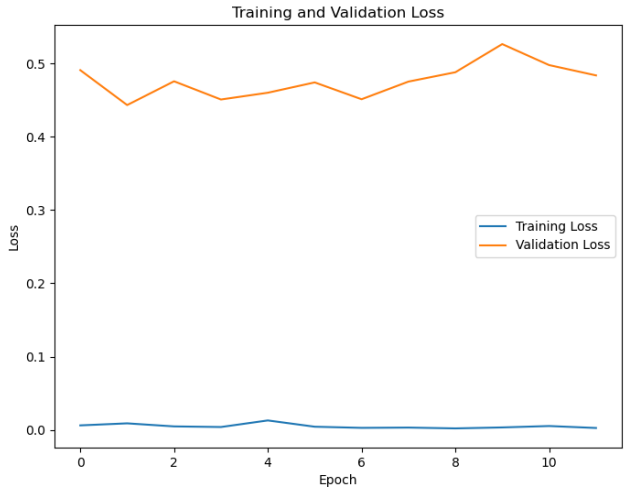
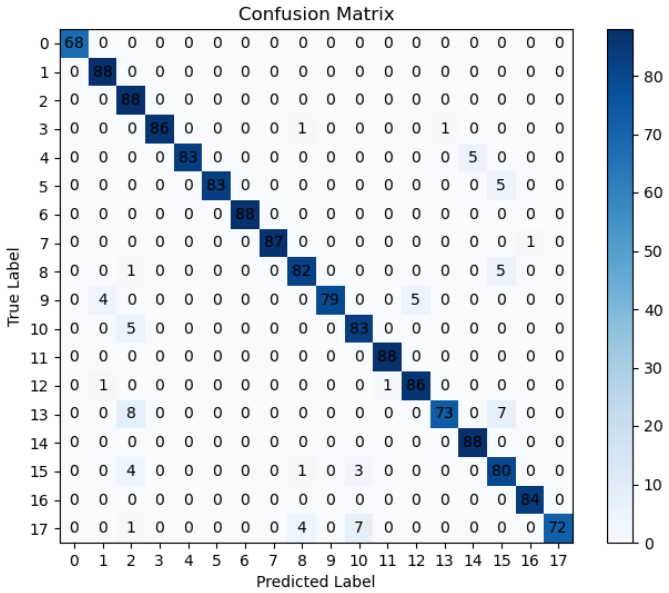
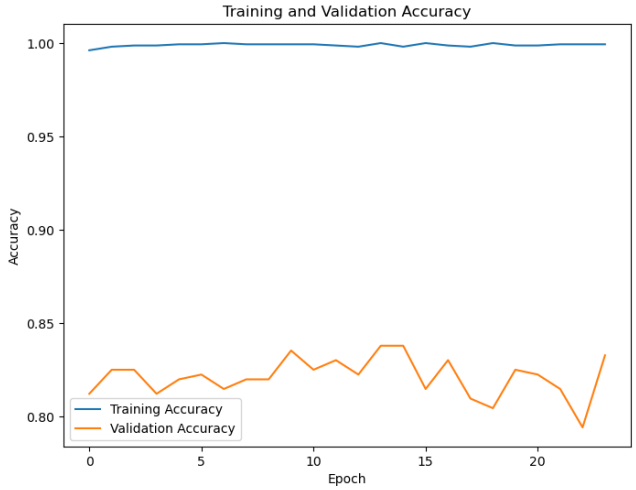


Figura 49: Figuras dos gráficos de treino do modelo Inception_UL50





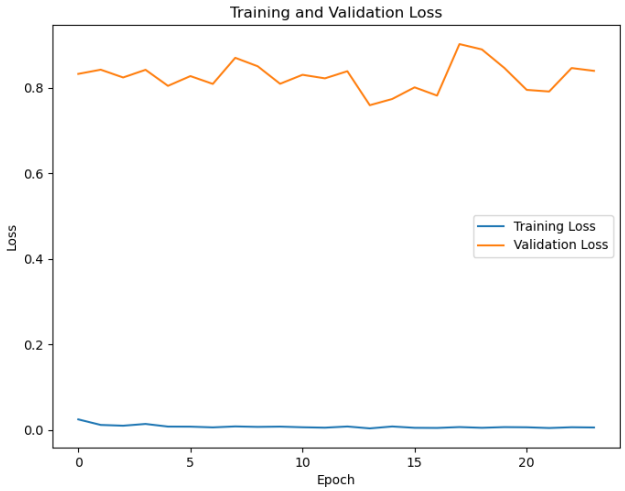
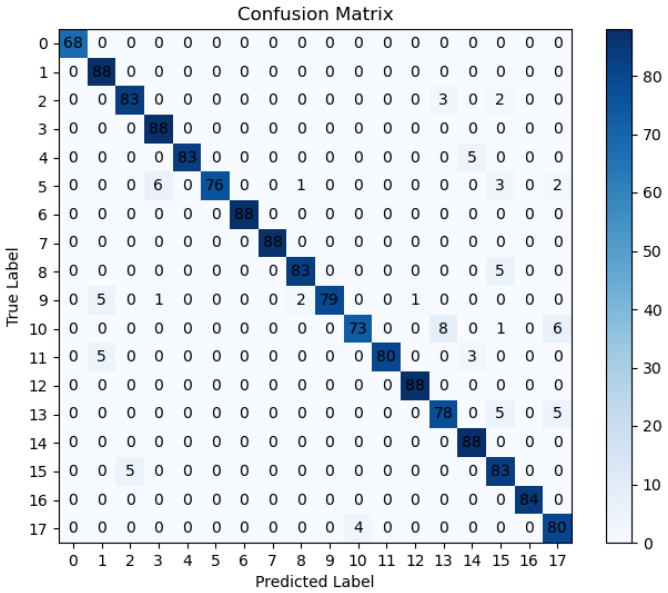
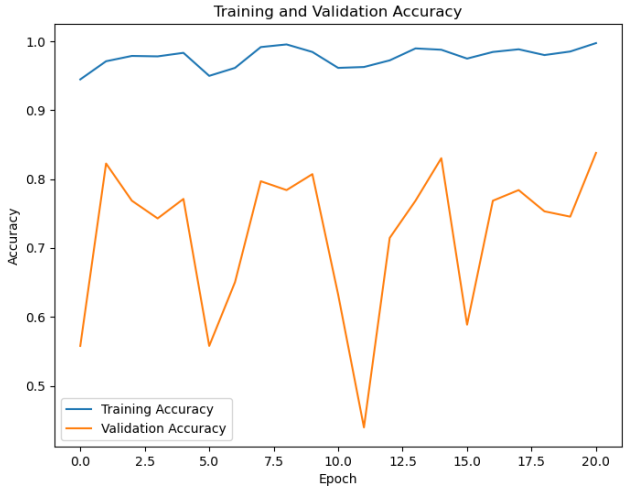


Figura 50: Figuras dos gráficos de treino do modelo ResNet50_UL10





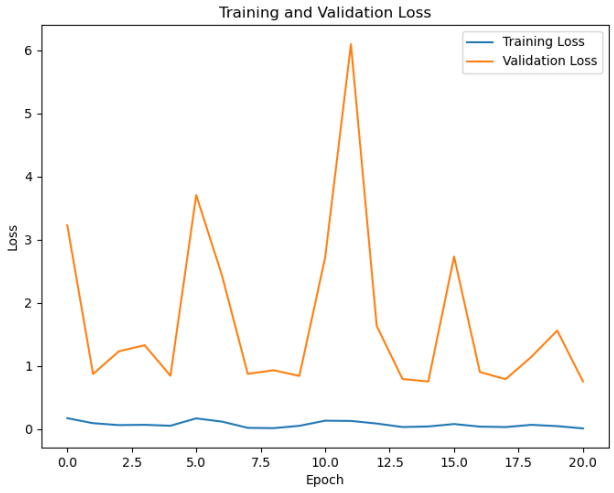


Figura 51: Figuras dos gráficos de treino do modelo Inception_RLR

APÊNDICE K

Neste apêndice podemos observar o ficheiro de código HTML que constitui o website fonemasweb.

Listagem 8: Código-fonte do ficheiro HTML relativo ao website fonemasweb com sintaxe colorida

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Fonemas Concatenados</title>
    <link href="https://fonts.googleapis.com/css2?family=Source+Sans+Pro:wght@30
    ↪ 0;400&display=swap"
    ↪ rel="stylesheet">
    <link rel="stylesheet" href="css/bootstrap.min.css">
    <link rel="stylesheet" href="css/slick.css" type="text/css" />
    <link rel="stylesheet" href="css/templatemo-style.css">
    <link rel="stylesheet" href="css/my-style.css">
    <script
    ↪ src="https://www.gstatic.com/firebasejs/8.6.8/firebase-app.js"></script>
    <script src="https://www.gstatic.com/firebasejs/8.6.8/firebase-storage.js"><
    ↪ /script>
    <link rel="icon" href="data:;base64,iVBORw0KGgo=">

</head>
<body>
    <video autoplay muted loop id="bg-video">
        <source src="video/gfp-astro-timelapse.mp4" type="video/mp4">
    </video>
    <div class="page-container">
      <div class="container-fluid header-container">
        <div class="row">
          <div class="col-xs-12">
            <div class="cd-slider-nav">
              <nav class="navbar navbar-expand-lg" id="tm-nav">
                <a class="navbar-brand" href="#">Deep Learning para Fonemas
                ↪ Concatenados</a>
                  <button class="navbar-toggler" type="button"
                  ↪ data-bs-toggle="collapse"
                  ↪ data-bs-target="#navbar-supported-content"
                  ↪ aria-controls="navbarSupportedContent"
                  ↪ aria-expanded="false" aria-label="Toggle navigation">
```

```
30                    <span class="navbar-toggler-icon"></span>
31                  </button>
32                  <div class="collapse navbar-collapse"
                    ↪  id="navbar-supported-content">
33                    <ul class="navbar-nav mb-2 mb-lg-0">
34                      <li class="nav-item selected">
35                        <a class="nav-link" aria-current="page" href="#0"
                          ↪  data-no="1">Início</a>
36                        <div class="circle"></div>
37                      </li>
38                      <li class="nav-item">
39                        <a class="nav-link" href="#0" data-no="3">Gravação de
                          ↪  Audio</a>
40                        <div class="circle"></div>
41                      </li>
42                      <li class="nav-item">
43                        <a class="nav-link" href="#0" data-no="4">Contactos</a>
44                        <div class="circle"></div>
45                      </li>
46                    </ul>
47                  </div>
48              </nav>
49            </div>
50          </div>
51        </div>
52      </div>
53      <div class="container-fluid tm-content-container">
54        <ul class="cd-hero-slider mb-0 py-5">
55          <li class="px-3" data-page-no="1">
56            <div class="page-width-1 page-center">
57              <div class="d-flex position-relative tm-border-top tm-border-bottom
                ↪  intro-container">
58                <div class="tm-bg-dark" style="
59                  padding-top: 20px;
60                  padding-left: 10px;
61                  padding-right: 10px;
62                  padding-bottom: 20px;
63                  ">
64                  <h2 class="mb-4">Fonemas Concatenados</h2>
65                  <p class="mb-4">
66                    Este website foi desenhado com o propósito de recolher
                      ↪  ficheiros áudio com a finalidade de criar uma base de
                      ↪  dados para a dissertação "Deep Learning para Fonemas
                      ↪  Concatenados" do mestrado
67                    em Ciência de Dados do <a rel="sponsored"
                      ↪  href="https://www.ipleiria.pt/"
                      ↪  target="_blank">Instituto Politécnico de Leiria</a>,
                      ↪  desenvolvido pelo aluno Pedro Costa.</p>
68                  <p class="mb-4">
69                    Este trabalho irá utilizar redes neurais artificiais em Deep
                      ↪  Learning que serão aplicadas no reconhecimento
70                    de fonemas concatenados para comunicação verbal com a API do
                      ↪  famoso Chat GPT.</p>
71                  <p class="mb-0">
```

```
72                  Se necessitar de um vídeo tutorial para saber como gravar o
                    ↪ seu áudio corretamente basta clicar <a rel="nofollow"
                    ↪ href="https://www.youtube.com/watch?v=c5b2GrjTHMY"
                    ↪ target="_blank">AQUI</a>.
73                  Para continuar clique no botão abaixo "Ir para Gravação de
                    ↪ Áudio".</p>
74                </div>
75                <div class="circle intro-circle-1"></div>
76                <div class="circle intro-circle-2"></div>
77                <div class="circle intro-circle-3"></div>
78                <div class="circle intro-circle-4"></div>
79              </div>
80              <div class="text-center">
81                <a href="#0" data-page-no="3" class="btn btn-primary tm-intro-btn
                  ↪ tm-page-link">
82                  Ir para Gravação de Áudio
83                </a>
84              </div>
85            </div>
86          </li>
87          <li data-page-no="3" class="px-3">
88            <div class="position-relative page-width-3 page-left tm-border-top
              ↪ tm-border-bottom tm-bg-dark">
89              <div class="circle intro-circle-1"></div>
90              <div class="circle intro-circle-2"></div>
91              <div class="circle intro-circle-3"></div>
92              <div class="circle intro-circle-4"></div>
93              <div>
94                <p class="m-4 mt-0">
95                  Para fazer uma gravação de áudio clique num botão numerado.
                    ↪ Após clicar no botão tem 2 segundos de gravação para dizer
                    ↪ o número correspondente ao do botão.
96                  Quando terminar a gravação poderá ouvi-la para confirmar que
                    ↪ foi feita corretamente ao clicar no botão "play" que
                    ↪ aparece à direita do botão numerado.
97                  Se necessitar de corrigir uma gravação basta voltar a clicar
                    ↪ no botão numerado correspondente ao audio que pretende
                    ↪ corrigir.
98                  Repita este processo para todos os botões.
99                  Quando fizer as 10 gravações e selecionar o seu género o botão
                    ↪ de upload irá desbloquear e poderá carregar no mesmo,
                    ↪ concluindo o processo.</p>
100             </div>
101             <div class="grid-container">
102               <div class="button-row">
103                 <div class="button-column">
104                   <button class="button-81 bnumber" role="button" id="button0"
                      ↪ onclick="recordAudio(0)">0</button>
105                   <div class="audio-player-container"
                      ↪ id="audioPlayerContainer0">
106                     <audio class="audio-player-custom" id="audioPlayer0"
                        ↪ controls></audio>
107                   </div>
108                 </div>
```

```
109                     <div class="button-column">
110                       <button class="button-81 bnumber" role="button" id="button1"
                          ↪  onclick="recordAudio(1)">1</button>
111                       <div class="audio-player-container"
                          ↪  id="audioPlayerContainer1">
112                         <audio class="audio-player-custom" id="audioPlayer1"
                            ↪  controls></audio>
113                       </div>
114                     </div>
115                     <div class="button-column">
116                       <button class="button-81 bnumber" role="button" id="button2"
                          ↪  onclick="recordAudio(2)">2</button>
117                       <div class="audio-player-container"
                          ↪  id="audioPlayerContainer2">
118                         <audio class="audio-player-custom" id="audioPlayer2"
                            ↪  controls></audio>
119                       </div>
120                     </div>
121                   </div>
122                   <div class="button-row">
123                     <div class="button-column">
124                       <button class="button-81 bnumber" role="button" id="button3"
                          ↪  onclick="recordAudio(3)">3</button>
125                       <div class="audio-player-container"
                          ↪  id="audioPlayerContainer3">
126                         <audio class="audio-player-custom" id="audioPlayer3"
                            ↪  controls></audio>
127                       </div>
128                     </div>
129                     <div class="button-column">
130                       <button class="button-81 bnumber" role="button" id="button4"
                          ↪  onclick="recordAudio(4)">4</button>
131                       <div class="audio-player-container"
                          ↪  id="audioPlayerContainer4">
132                         <audio class="audio-player-custom" id="audioPlayer4"
                            ↪  controls></audio>
133                       </div>
134                     </div>
135                     <div class="button-column">
136                       <button class="button-81 bnumber" role="button" id="button5"
                          ↪  onclick="recordAudio(5)">5</button>
137                       <div class="audio-player-container"
                          ↪  id="audioPlayerContainer5">
138                         <audio class="audio-player-custom" id="audioPlayer5"
                            ↪  controls></audio>
139                       </div>
140                     </div>
141                   </div>
142                   <div class="button-row">
143                     <div class="button-column">
144                       <button class="button-81 bnumber" role="button" id="button6"
                          ↪  onclick="recordAudio(6)">6</button>
145                       <div class="audio-player-container"
                          ↪  id="audioPlayerContainer6">
```

```
146                     <audio class="audio-player-custom" id="audioPlayer6"
                        ↪  controls></audio>
147                   </div>
148                 </div>
149                 <div class="button-column">
150                   <button class="button-81 bnumber" role="button" id="button7"
                      ↪  onclick="recordAudio(7)">7</button>
151                   <div class="audio-player-container"
                      ↪  id="audioPlayerContainer7">
152                     <audio class="audio-player-custom" id="audioPlayer7"
                        ↪  controls></audio>
153                   </div>
154                 </div>
155                 <div class="button-column">
156                   <button class="button-81 bnumber" role="button" id="button8"
                      ↪  onclick="recordAudio(8)">8</button>
157                   <div class="audio-player-container"
                      ↪  id="audioPlayerContainer8">
158                     <audio class="audio-player-custom" id="audioPlayer8"
                        ↪  controls></audio>
159                   </div>
160                 </div>
161               </div>
162               <div class="button-row">
163                 <div class="button-column">
164                   <button class="button-81 bnumber" role="button" id="button9"
                      ↪  onclick="recordAudio(9)">9</button>
165                   <div class="audio-player-container"
                      ↪  id="audioPlayerContainer9">
166                     <audio class="audio-player-custom" id="audioPlayer9"
                        ↪  controls></audio>
167                   </div>
168                 </div>
169               </div>
170             </div>
171             <div class="footer">
172               <div class="radio-button">
173                 <input type="radio" name="gender" value="M"
                    ↪  onclick="checkUploadButtonStatus()"> Masculino
174                 <input type="radio" name="gender" value="F"
                    ↪  onclick="checkUploadButtonStatus()"> Feminino
175               </div>
176               <div class="upload-button">
177                 <button class="button-81" role="button"
                    ↪  onclick="uploadAudios()" id="uploadButton"
                    ↪  disabled>Upload</button>
178               </div>
179             </div>
180           </li>
181           <li data-page-no="4">
182             <div class="mx-auto page-width-2">
183               <div class="row">
184                 <div class="col-md-6 me-0 ms-auto">
185                   <h2 class="mb-4">Contactos</h2>
```

```
186                 </div>
187               </div>
188               <div class="row">
189                 <div class="col-md-6 tm-contact-left">
190                   <form action="#" method="POST" class="contact-form"
                      ↪  id="uploadForm">
191                     <div class="input-group tm-mb-30">
192                         <input name="name" id="nameInput" type="text"
                            ↪  class="form-control rounded-0 border-top-0
                            ↪  border-end-0 border-start-0" placeholder="Name"
                            ↪  required>
193                     </div>
194                     <div class="input-group tm-mb-30">
195                         <input name="email" id="emailInput" type="text"
                            ↪  class="form-control rounded-0 border-top-0
                            ↪  border-end-0 border-start-0" placeholder="Email"
                            ↪  required>
196                     </div>
197                     <div class="input-group tm-mb-30">
198                         <textarea rows="5" id="messageInput" name="message"
                            ↪  class="textarea form-control rounded-0 border-top-0
                            ↪  border-end-0 border-start-0" placeholder="Message"
                            ↪  required></textarea>
199                     </div>
200                     <div class="input-group justify-content-end">
201                         <input type="submit" class="btn btn-primary
                            ↪  tm-btn-pad-2" value="Enviar">
202                     </div>
203                   </form>
204                 </div>
205                 <div class="col-md-6 tm-contact-right">
206                   <p class="mb-4">
207                     Se tiver alguma questão basta contactar preenchendo este
                      ↪  formulário ou enviar um email diretamente.
208                     Uma resposta será enviada para o seu email assim que
                      ↪  possível.
209                   </p>
210                   <div>
211                     Email: <a href="mailto:2223244@my.ipleiria.pt"
                      ↪  class="tm-link-white">2223244@my.ipleiria.pt</a>
212                   </div>
213                 </div>
214               </div>
215             </div>
216           </li>
217         </ul>
218     </div>
219   </div>
220   <!-- Preloader, https://ihatetomatoes.net/create-custom-preloading-screen/ -->
221   <div id="loader-wrapper">
222     <div id="loader"></div>
223     <div class="loader-section section-left"></div>
224     <div class="loader-section section-right"></div>
225   </div>
```

```
<script src="js/jquery-3.5.1.min.js"></script>
<script src="js/bootstrap.min.js"></script>
<script src="js/slick.js"></script>
<script src="js/templatemo-script.js"></script>
<script>
  const firebaseConfig = {
    apiKey: "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
    authDomain: "fonemasweb.firebaseapp.com",
    projectId: "fonemasweb",
    storageBucket: "xxxxxxxxxxxxxxxxxxx",
    messagingSenderId: "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
    appId: "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  };
  firebase.initializeApp(firebaseConfig);
  const storage = firebase.storage();

  let recordedAudios = Array(10).fill(null);
  let audioPlayers = [];

  function recordAudio(index) {
    // Disable all buttons
    document.querySelectorAll('.button-81').forEach(button => {
      button.disabled = true;
    });

    navigator.mediaDevices.getUserMedia({ audio: true })
      .then(function(stream) {
        const mediaRecorder = new MediaRecorder(stream, { mimeType:
        ↪  'audio/webm' });
        const chunks = [];

        mediaRecorder.start();
        document.getElementById(`button${index}`).style.backgroundColor =
        ↪  '#ff0000c5';

        setTimeout(() => {
          mediaRecorder.stop();
        }, 2500);

        mediaRecorder.ondataavailable = function(event) {
          chunks.push(event.data);
        }

        mediaRecorder.onstop = function() {
          const recordedBlob = new Blob(chunks, { type: 'audio/webm' });
          recordedAudios[index] = recordedBlob;
          const audioPlayer = document.getElementById(`audioPlayer${index}`);
          audioPlayer.src = URL.createObjectURL(recordedBlob);
          document.getElementById(`audioPlayerContainer${index}`).style.displa
          ↪  y =
          ↪  'block';
          audioPlayers.push(audioPlayer);
          checkUploadButtonStatus();
```

```
276             document.getElementById(`button${index}`).style.backgroundColor = '';
277             // Re-enable all buttons
278             document.querySelectorAll('.bnumber').forEach(button => {
279               button.disabled = false;
280             });
281           }
282         })
283         .catch(function(err) {
284           console.error('Error accessing microphone:', err);
285         });
286     }
287
288     function uploadAudios() {
289       let uploadPromises = [];
290       document.getElementById('uploadButton').disabled = true;
291       const time = Date.now();
292       recordedAudios.forEach((audioBlob, index) => {
293         if (audioBlob) {
294           const storageRef = storage.ref();
295           const gender =
          ↪  document.querySelector('input[name="gender"]:checked').value;
296           const audioRef = storageRef.child(`${index}/${gender}_${time}.webm`);
297           uploadPromises.push(audioRef.put(audioBlob));
298         }
299       });
300
301       Promise.all(uploadPromises)
302         .then((snapshots) => {
303           console.log('Uploaded all audios successfully:', snapshots);
304           alert('Audios carregados com sucesso, obrigado!');
305         })
306         .catch((error) => {
307           console.error('Error uploading audios:', error);
308           alert('Ocorreu um erro, tente mais tarde');
309         });
310     }
311
312     function checkUploadButtonStatus() {
313       const uploadButton = document.getElementById('uploadButton');
314       const maleRadio =
        ↪  document.querySelector('input[name="gender"][value="M"]');
315       const femaleRadio =
        ↪  document.querySelector('input[name="gender"][value="F"]');
316       if ((recordedAudios.every(audio => audio !== null)) && (maleRadio.checked
        ↪  || femaleRadio.checked)) {
317         uploadButton.disabled = false;
318       }
319     }
320
321
322     function handleResize() {
323       recordedAudios.forEach((audio, index) => {
324         if (audio !== null) {
325           if (window.innerWidth < 800) {
```

```
326                 document.getElementById(`audioPlayerContainer${index}`).style.disp⌋
                    ↪  lay =
                    ↪  'none';
327             } else if (window.innerWidth < 1200) {
328                 document.getElementById(`audioPlayer${index}`).style.width =
                    ↪  '100px';
329                 document.getElementById(`audioPlayerContainer${index}`).style.disp⌋
                    ↪  lay = 'block';
                    ↪
330             } else {
331                 document.getElementById(`audioPlayer${index}`).style.width =
                    ↪  '200px';
332                 document.getElementById(`audioPlayerContainer${index}`).style.disp⌋
                    ↪  lay = 'block';
                    ↪
333             }
334         }
335       });
336     }
337
338     window.addEventListener('resize', handleResize);
339
340     // Handle form submission
341     document.getElementById('uploadForm').addEventListener('submit',
        ↪  function(event) {
342         event.preventDefault();
343
344         // Get form inputs
345         const name = document.getElementById('nameInput').value;
346         const email = document.getElementById('emailInput').value;
347         const message = document.getElementById('messageInput').value;
348         const fileContent = `Name: ${name}\nEmail: ${email}\nMessage: ${message}`;
349
350         const blob = new Blob([fileContent], { type: 'text/plain' });
351         const storageRef = storage.ref();
352         const fileRef = storageRef.child(`messages/${Date.now()}.txt`);
353         fileRef.put(blob).then(function(snapshot) {
354             console.log('File uploaded successfully!');
355             alert('Mensagem enviada com sucesso, obrigado!');
356         }).catch(function(error) {
357             console.error('Error uploading file:', error);
358             alert('Ocorreu um erro, tente mais tarde');
359         });
360     });
361   </script>
362 </body>
363 </html>
```

D E C L A R A Ç Ã O

Declaro, sob compromisso de honra, que o trabalho apresentado nesta dissertação, com o título *"Deep Learning para Reconhecimento de Fonemas Concatenados"*, é original e foi realizado por Estudante Pedro Silva Varela Costa (2223244) sob orientação de Professor Doutor João da Silva Pereira (joao.pereira@ipleiria.pt).

Leiria, Setembro de 2024

Estudante Pedro Silva Varela Costa