Deep Learning para Reconhecimento de Voz em Aplicações Médicas (Deepvoice)

Mestrado em Engenharia Informática – Computação Móvel

Pedro Henrique Proença Carreira – 2210491

Leiria, Setembro de 2023

Deep Learning para Reconhecimento de Voz em Aplicações Médicas (Deepvoice)

Mestrado em Engenharia Informática – Computação Móvel

Pedro Henrique Proença Carreira – 2210491

Dissertação de Mestrado do curso de Engenharia Informática – Computação Móvel realizada sob a orientação do Professor Doutor João da Silva Pereira

Leiria, Setembro de 2023

Agradecimentos

A realização desta dissertação contou com importantes apoios e incentivos sem os quais não se teria tornado realidade.

Ao professor João Pereira, pela sua orientação, total apoio e disponibilidade, pelo saber que me incutiu e transmitiu, pelas opiniões e críticas construtivas, e plena colaboração na resolução de problemas que foram surgindo aquando do desenvolvimento deste projeto.

Aos meus colegas que estiveram do meu lado durante a realização de todo este trabalho, pelo companheirismo e força em todos os momentos.

Por último um especial agradecimento aos meus pais que tornaram tudo isto possível, sendo estes um modelo para mim, pelo seu apoio incondicional, e mais que tudo pela sua paciência demonstrada ao longo destes anos académicos.

Resumo

Esta dissertação teve como objetivo trabalhar na melhoria e acessibilidade de uma aplicação médica desenvolvida o ano passado, recorrendo ao uso de redes neuronais em *Deep Learning*. A aplicação final tem como objetivo, permitir o controlo por voz da aplicação de diagnóstico de Covid-19, controlando o rato com recurso a comandos de voz.

Dessa forma, foi inicialmente efetuado um estudo de diversas formas, algoritmos e estruturas das Redes Neuronais em *Deep Learning* de forma a apurar qual das diversas opções seria mais viável, precisa e rápida. Após isso, foi implementado um sistema de reconhecimento de voz utilizando um método inovador baseado em espectrogramas de Mel que posteriormente são analisados pela rede neuronal de forma a ser efetuado o treino do modelo usado pela aplicação.

O uso de técnicas de *Deep Learning* e a abordagem inovadora com espectrogramas proporcionaram resultados muito bons e que encaminham esta metodologia para posterior investigação, possivelmente com *datasets* que incluam um maior número de amostras com mais oradores.

Palavras-chave: *Deep Learning, CNN, EfficientNet, Resnet50, Espetrogramas*

Abstract

This dissertation aimed to work on the improvement and accessibility of a medical application developed last year, using neural networks in Deep Learning. The final application aims to allow voice control of the Covid-19 diagnostic application, controlling the mouse using voice commands.

Thus, a study of various forms, algorithms and structures of Deep Learning Neural Networks was initially carried out in order to determine which of the various options would be most viable, accurate and fast. After that, a voice recognition system was implemented using an innovative method based on Mel spectrograms that are later analysed by the neural network in order to train the model to be used by the application.

The use of Deep Learning techniques and the innovative approach with spectrograms provided very good results that refer this methodology for further investigation, possibly with datasets that include a larger number of samples from more speakers.

Keywords: *Deep Learning, CNN, EfficientNet, Resnet50, Spectrograms*

Índice

Lista de Figuras

Lista de Tabelas

Lista de siglas e acrónimos

DNN	Deep Neural Network
CNN	*Convolutional Neural Network*
RNN	*Recurrent Neural Network*
LSTM	*Long Short Term Memory*
GRU	*Gated Recurrent Unit*
CPU	*Central Processing Unit*
CSV	*Comma-Separated Values*
DL	*Deep Learning*
ESTG	Escola Superior de Tecnologia e Gestão
GPU	*Graphics Processing Unit*
IA	Inteligência Artificial
ML	*Machine Learning*
TPU	*Tensor Processing Unit*

1 Introdução

Nas últimas décadas, temos testemunhado um notável crescimento e desenvolvimento no campo do reconhecimento de voz, uma conquista que se deve, em grande parte, à revolução do *Deep Learning*. As aplicações dessa tecnologia estão cada vez mais presentes na nossa vida cotidiana, desde assistentes de voz em dispositivos móveis até sistemas de segurança e acessibilidade que respondem a comandos de voz. Como resultado, o reconhecimento de voz estabeleceu-se como uma ferramenta central que permite interações naturais e intuitivas entre humanos e máquinas. Além dessas inúmeras vantagens, o desenvolvimento nesse campo visa também esbater cada vez mais a impossibilidade de indivíduos com debilidades motoras realizar determinadas ações no seu dia a dia, possibilitando assim mais autonomia seja em atividades de lazer ou em ambiente profissional.

É neste contexto que surge a necessidade de obter mecanismos de compreensão de comandos de voz e palavras que sejam precisos e rápidos, reduzindo assim a taxa de erro das aplicações e transmitindo uma melhor experiência de utilização.

Dentro deste cenário dinâmico e diversificado das redes neurais, este estudo almeja alcançar três objetivos fundamentais:

- Extração de Características Relevantes dos Espectrogramas, ou seja, extrair características e padrões dos gráficos de intensidade e amplitude das frequências sonoras das palavras.
- Desenvolvimento de Modelos de *Deep Learning* capazes de classificar, a partir das características obtidas os diversos comandos de voz/palavras.
- Desenvolvimento de uma aplicação capaz de identificar comandos de voz/palavras com base no modelo obtido

Considerando a diversidade de arquiteturas que se pode obter com redes de *Deep Learning*, optou-se por implementar um modelo estruturado por nós, analisar o seu desempenho e procurar melhorá-lo através do uso da técnica *Augmentation* e, por fim, compará-lo com dois modelos pré treinados, um que implementa a arquitetura ResNet50 e outro que implementa a arquitetura EfficientNet.

As secções seguintes deste documento encontram-se estruturadas da seguinte forma:

- **Capítulo 2** – Enquadramento Teórico, onde são explicados e detalhados diversos conceitos essenciais para a compreensão do trabalho desenvolvido neste projeto;
- **Capítulo 3** – Reconhecimento de Voz/Palavras, explica o progresso efetuado até agora nessa área e as diversas metodologias utilizadas para fazer esse reconhecimento;
- **Capítulo 4** – Modelos desenvolvidos, apresenta a estrutura dos modelos desenvolvidos e as metodologias de treino executadas e os resultados obtidos, assim como foram obtidos os Espectrogramas utilizados.
- **Capítulo 5** – Discussão de resultados, realiza uma comparação entre os resultados obtidos neste projeto com resultados obtidos em artigos relevantes;
- **Capítulo 6** – Aplicação, descreve o processo de elaboração e utilização da aplicação desenvolvida
- **Capítulo 7** – Trabalhos futuros, indica alterações futuras que a implementar no projeto, desde melhorias no processo a adição de funcionalidades na aplicação;
- **Capítulo 8** – Conclusões

2 Enquadramento Teórico

No presente capítulo irão ser abordados os diversos conceitos teóricos, trabalhos pioneiros e recentes que por sua vez foram a base do desenvolvimento deste projeto. Desta forma foi importante efetuar uma pesquisa aprofundada sobre todas as tecnologias existentes e necessárias e de vários estudos efetuados aos longos dos anos. Só assim foi possível construir os fundamentos necessários para o estudo e investigação que foram realizados no âmbito do *Deep Learning* aplicado ao reconhecimento de voz.

2.1. *Machine Learning*

Machine Learning é um campo na área da Inteligência Artificial (IA) que consiste no desenvolvimento e aplicação de algoritmos e modelos estatísticos, que permitem a sistemas computorizados aprender sem ser necessária à sua programação explícita, tirando partido da informação presente no grande volume de dados utilizados no seu desenvolvimento. Permite a criação de sistemas que, de forma independente, analisam e interpretam padrões ou realizam previsões com base nos dados.

Em *Machine Learning* é possível realizar uma diferenciação dos algoritmos existentes com base no processo de aprendizagem. Os três tipos de aprendizagem principais são [1]:

- Aprendizagem Supervisionada (*Supervised Learning*): o algoritmo é treinado recorrendo a *datasets* onde o atributo objetivo está identificado. A aprendizagem é realizada através da estimação de dependências entre os dados de entrada e os dados de saída, através do ajuste dos parâmetros e pesos associados ao algoritmo ou modelo. Este tipo de aprendizagem é utilizado em problemas de classificação, onde se pretende identificar classes, e regressão, onde se pretende prever valores contínuos.
- Aprendizagem Não Supervisionada (*Unsupervised Learning*): o algoritmo é utilizado com *datasets* onde não existe um objetivo definido. Utilizada para estimar dependências a partir dos dados de entrada, tem como objetivo descobrir padrões e estruturas previamente desconhecidas, presentes nos dados. Problemas de *Clustering* e Associação de dependências são exemplos de problemas de Aprendizagem Não Supervisionada.
- *Reinforcement Learning*: é uma metodologia de aprendizagem onde um agente é treinado num ambiente interativo e aprende com base nas suas ações e

experiências, recebendo feedback positivo ou negativo na forma de *rewards* ou *penalties*, respetivamente. O agente tem como objetivo maximizar o efeito cumulativo das *rewards* enquanto explora e atua sobre o ambiente [2].

Atualmente, a aplicação de algoritmos de *Machine Learning* pode ser encontrada em sistemas como reconhecimento de imagens e de fala, processamento de linguagem natural, sistemas de recomendação, deteção de fraude e *Market Basket Analysis* [3].

No presente projeto, foi utilizado uma subcategoria do *Machine Learning*, conhecida como *Deep Learning* num contexto de aprendizagem supervisionada. O *Deep Learning* é construído recorrendo a redes neuronais. Os conceitos necessários à compreensão desta temática podem ser explorados nos subcapítulos seguintes.

2.1.1 Neurónio Artificial

A estrutura elementar que constitui as redes neuronais é o neurónio artificial, que visa modelar através de funções matemáticas o comportamento de neurónios biológicos. A sua composição, indicada na Figura 1, consiste nas seguintes variáveis:

- valores de entrada, representados pelas variáveis x;
- pesos das entradas, representados pelas variáveis *w*, determinam o significado que uma entrada terá na saída [4];
- *offset* ou *bias*, representado pela variável β, é uma variável independente de outros neurónios que ajuda a controlar o valor a que a função de ativação é acionada;
- soma pesada dos valores de entrada;
- função de ativação que produz o valor de saída.

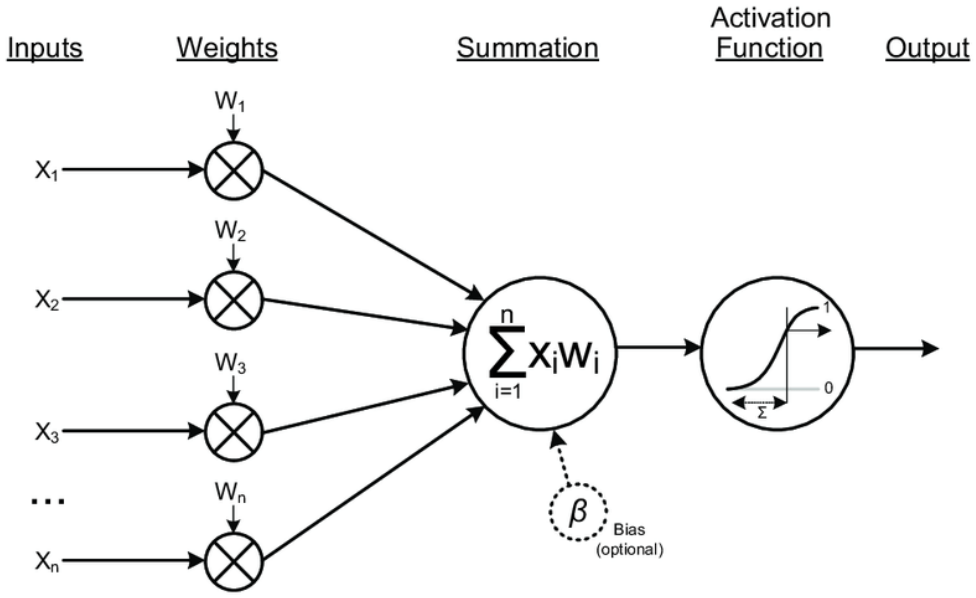


Figura 1 - Diagrama de um neurónio artificial [5]

A computação do valor de saída de um neurónio é realizada em duas fases, a primeira calcula o somatório dos valores de entrada multiplicados pelos respetivos pesos, soma pesada, tendo em conta o *bias*, e a segunda é utilização da função de ativação com base no valor previamente calculado [6].

Funções de ativação, utilizadas para transformar o valor de entrada num valor de saída, têm um impacto significativo no desempenho da rede neuronal [9] pelo que a sua escolha deve ser realizada de acordo com o caso em estudo. Estas podem ser organizadas em duas categorias distintas, funções de ativação descontínuas, como a *Step* e a *Signal*, e funções de ativação contínuas como por exemplo a *Sigmoid*, *Tanh*, *SoftMax* e a *ReLu*. Na Figura 2 e 3 encontram-se representadas as funções *ReLu* e *Sigmoid* respetivamente [6].

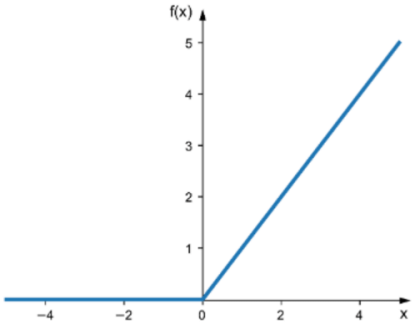


Figura 2 - Gráfico da função *Relu* [7]

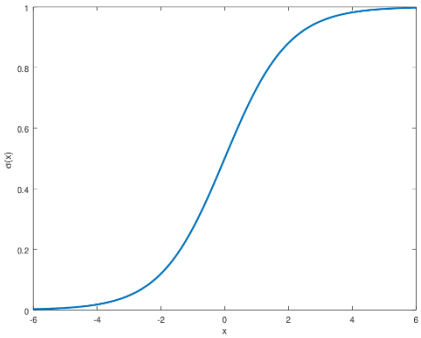


Figura 3 - Gráfico da função *Sigmoid* [8]

2.1.2 Redes Neuronais

Enquanto um neurónio artificial tenta replicar a funcionalidade de um neurónio biológico, as redes neuronais foram inspiradas pela estrutura do cérebro humano e tentam replicar a forma como os neurónios comunicam entre si. As redes neuronais são compostas por camadas de neurónios artificiais, em que cada neurónio de uma camada comunica com todos os neurónios da camada seguinte. Uma rede neuronal é composta por uma camada de entrada (*Input Layer*), uma ou mais camadas ocultas (*Hidden Layers*) e uma camada de saída (*Output Layer*), como pode ser visto na Figura 4. Os pesos associados aos neurónios que compõem as diferentes camadas de uma rede neuronal são tipicamente inicializados de forma aleatória. A precisão destas redes é melhorada recorrendo a processos de treino que permitem ajustar os pesos mencionados anteriormente. A modificação dos pesos durante o processo de treino pode ser realizada através de redes neuronais com *backpropagation* ou através de algoritmos de otimização que aplicam *Stochastic Gradient Descent* [11].

Um dos desafios da utilização de redes neuronais, surge durante o processo de treino onde podem ocorrer fenómenos de *overfitting* e *underfitting*, que indicam que o processo de aprendizagem ficou aquém do desejado. *Overfitting*, também designado por erro de generalização, indica que o modelo "memorizou" os dados utilizados no treino e é incapaz de generalizar para novos dados, por sua vez, *underfitting* indica que rede foi incapaz de aprender os padrões presentes nos dados.

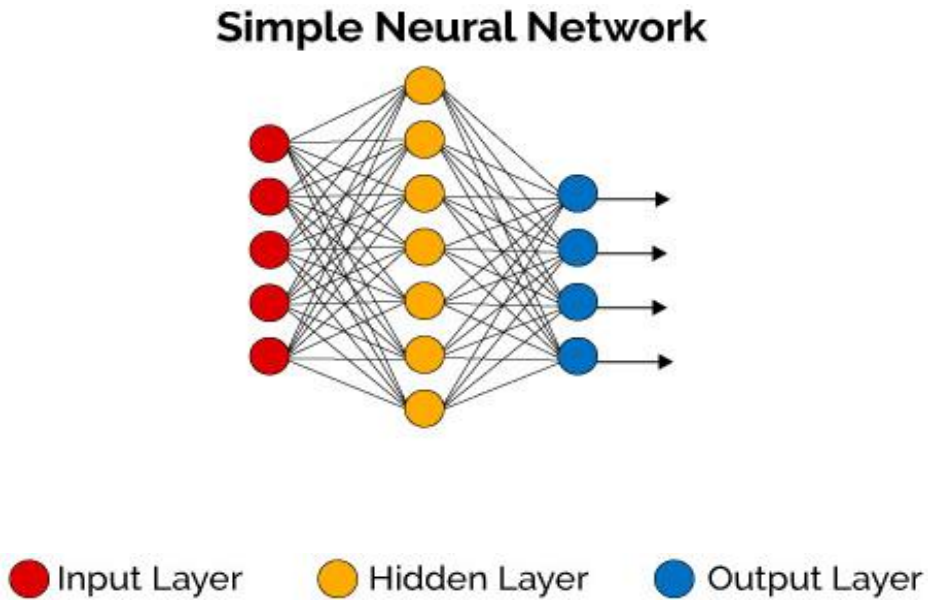


Figura 4 - Rede Neuronal Simples [10]

2.1.2.1 Redes Neuronais Convolucionais

As *Convolutional Neural Networks* (CNNs) são um tipo de redes de *deep learning* desenvolvido especificamente para o processamento e análise de dados num formato de grelha, como por exemplo imagens. As camadas convolucionais são usadas frequentemente com camadas de agrupamento e com camadas densamente ligadas, abordadas anteriormente. A Figura 5 representa a arquitetura geral de uma CNN.

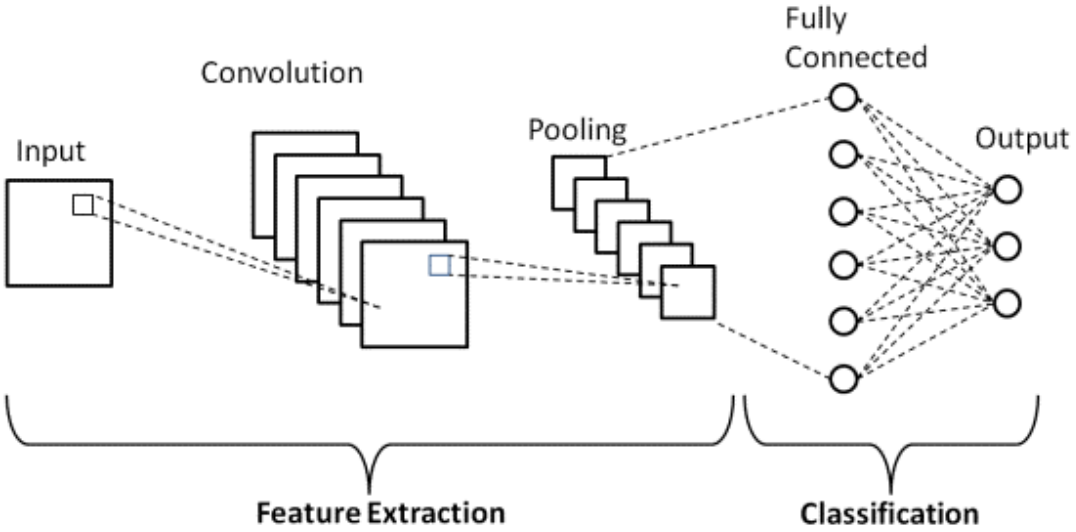


Figura 5 – Arquitectura básica de uma CNN [13]

As camadas convolucionais executam a principal operação das CNNs, a convolução. Esta consiste num conjunto de filtros (também chamados de *kernels*) que deslizam sobre os dados introduzidos. Cada filtro extrai características especificas realizando multiplicações e somas por elemento. Este processo captura padrões locais e relações, como arestas, vértices ou texturas. De seguida, encontram-se um conjunto de figuras que representam o funcionamento de uma camada convolucional, em que a matriz azul ilustra os dados introduzidos, a matriz verde exemplifica um filtro e a matriz vermelha representa o mapa de características extraídas.

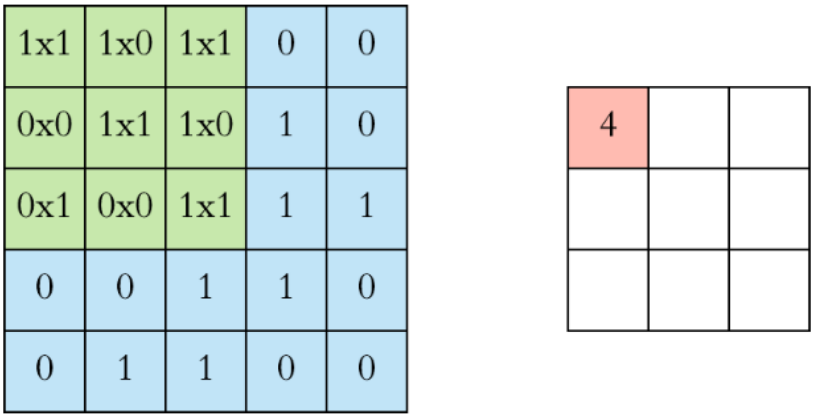


Figura 6 - Exemplo de convolução (Parte 1) [14]

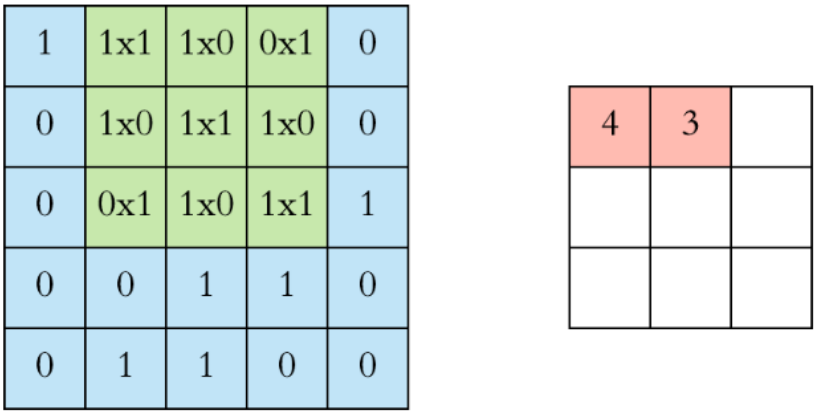


Figura 7 - Exemplo de convolução (Parte 2) [14]

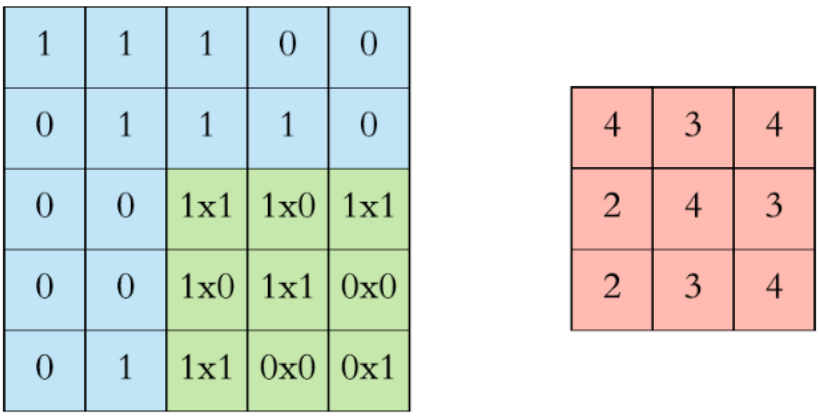


Figura 8 - Exemplo de convolução (Parte 3) [14]

As camadas de agrupamento (*Pooling*), por outro lado, têm como objetivo reduzir as dimensões espaciais (altura e largura) do mapa de características, mantendo a maioria da informação relevante. A operação de agrupamento mais comum é o agrupamento máximo (*Max Pooling*) que seleciona o valor máximo dentro de uma região do mapa de características e descarta o resto. Desta forma é possível reduzir a complexidade computacional e tornar a rede mais robusta a pequenas translações espaciais.

As Redes Neuronais Convolucionais (*Convolutional Neural Networks*) (CNNs) apresentam grande aplicabilidade em deteção de padrões em imagens. As CNNs são redes que empregam operações conhecidas como convoluções, um tipo especial de operação linear [40]. Os principais conceitos introduzidos pelas CNNs são: campos recetivos locais, que capturam características numa região específica; pesos compartilhados, que possibilitam o varrimento em todo o espaço e que otimizam o treino de uma maneira geral, e *Pooling*, que diminui a dimensionalidade de um tensor.

Uma CNN é uma subclasse de redes neuronais que têm pelo menos uma camada de convolução. São ótimas para capturar informação local (por exemplo, pixéis vizinhos numa imagem ou palavras circundantes num texto), bem como a redução da complexidade do modelo (treino mais rápido, requer menos amostras, reduz a hipótese de *overfitting*). Podemos pensar numa Rede Neuronal Convolucional como uma Rede Neuronal Artificial (ANN) que tem algum tipo de especialização para poder retirar ou detetar padrões no input fornecido e atribuir-lhes uma *label* ou um resultado. Esta capacidade de captar padrões é o que faz as CNN tão úteis para analisar imagens ou sons, tais como classificação de imagem ou reconhecimento de uma única palavra.

Primeiro uma CNN tem camadas ocultas chamadas camadas convolucionais, abordadas anteriormente. As mesmas são responsáveis por permitir a estas redes especializarem-se na deteção de padrões. Embora uma CNN possa também ter, como muitas vezes acontece, outros tipos de camadas, estas camadas convolucionais são as suas principais constituintes. Como qualquer outro tipo de camada, uma camada convolucionais também recebe um input, transforma-o de alguma forma, e envia o resultado da transformação para a camada seguinte.

As camadas de convolução são os principais blocos de construção de uma rede convolucional. A principal função dessas camadas é identificar “padrões locais” dentro dos dados aos quais a rede está exposta, assim como as células simples do córtex visual. Esses

padrões são localizados através de operações de convolução dos dados com camadas de filtros que são ajustados aos dados à medida que a rede neuronal é treinada.

Os filtros Convolucionais, também chamados de *kernels*, varrem os dados em busca de características simples, por exemplo, no caso de uma imagem, um filtro poderá ser ajustado para procurar contornos verticais. Esses mapas são passados para camadas posteriores da rede que, através das características simples que foram aprendidas pelas primeiras camadas, serão capazes de aprender características mais complexas. Essa propriedade chama-se hierarquia espacial de padrões e é o que torna redes Convolucionais ótimas para processamento de imagem.

O segundo bloco de construção introduzido nas redes convolucionais é a camada de *pooling*. A operação de *pooling* varre o mapa de características gerado pela camada de convolução com filtros de *pooling* e agrupa os valores mais importantes para comprimir as informações extraídas. Existem duas formas principais de *pooling*: por média e por valor máximo. *Pooling* por valor máximo (*MaxPooling*), consiste em selecionar apenas o valor mais alto encontrado pelo filtro na região analisada (Figura 9). A operação de *pooling* por média (*AvgPooling*) calcula a média aritmética dos valores dentro do campo do filtro e regista esse valor. A operação de MaxPooling geralmente é a mais utilizada no contexto das CNNs.

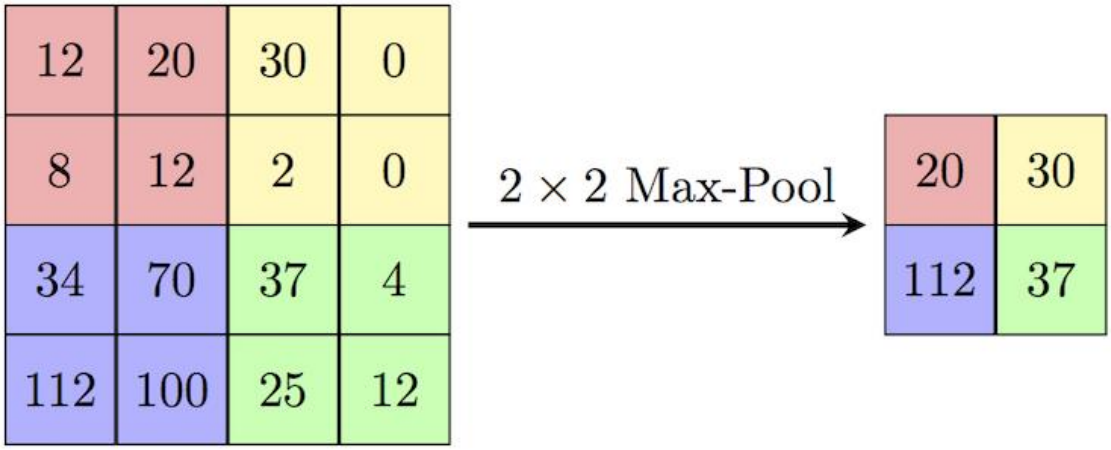


Figura 9 - Exemplo de Max Pooling [48]

A camada de *pooling*, portanto, tem por objetivo reduzir a quantidade de amostras de um mapa de características, de modo que só os pontos da imagem onde as características ficaram mais evidentes são considerados. O ato de realizar essa redução na dimensionalidade dos dados, dando prioridade apenas para os pontos mais significativos, proporciona tanto

uma melhora capacidade de organizar padrões hierarquicamente quanto reduz a tendência da rede ao *overfitting* uma vez que diminui a quantidade de parâmetros da rede.

Outras duas camadas que, embora não sejam características de redes convolucionais, geralmente estão presentes em modelos de *Deep Learning* pelos benefícios que trazem ao treino são as camadas de *Batch Normalization* e de *Dropout*. A camada de *Batch Normalization* é uma camada de normalização das entradas e geralmente é aplicada antes de cada camada de convolução e contribui muito para melhorar a performance em geral da rede neuronal. A camada de *Dropout*, por sua vez, faz com que cada neurónio da camada possua uma probabilidade de ser ignorado em etapas do treino. A camada de *Dropout* possui um impacto significativo para prevenir o overfitting do modelo.

2.1.2.2 Redes Neuronais Recorrentes

Além das redes Convolucionais, outra classe de rede neuronal que tem vindo a ser utilizada em sistemas de reconhecimento de fala são as chamadas Redes Neuronais Recorrentes, uma classe de redes especialmente projetada para lidar com dados sequenciais, como séries temporais, documentos de texto e amostras de áudio [41]. Tradicionalmente as redes recorrentes são mais utilizadas em tarefas de processamento de linguagem natural (NLP) e previsão de etapas futuras de séries temporais, porém existem projetos que têm como objetivos a sua utilização em tarefas de classificação de sinais de áudio [42].

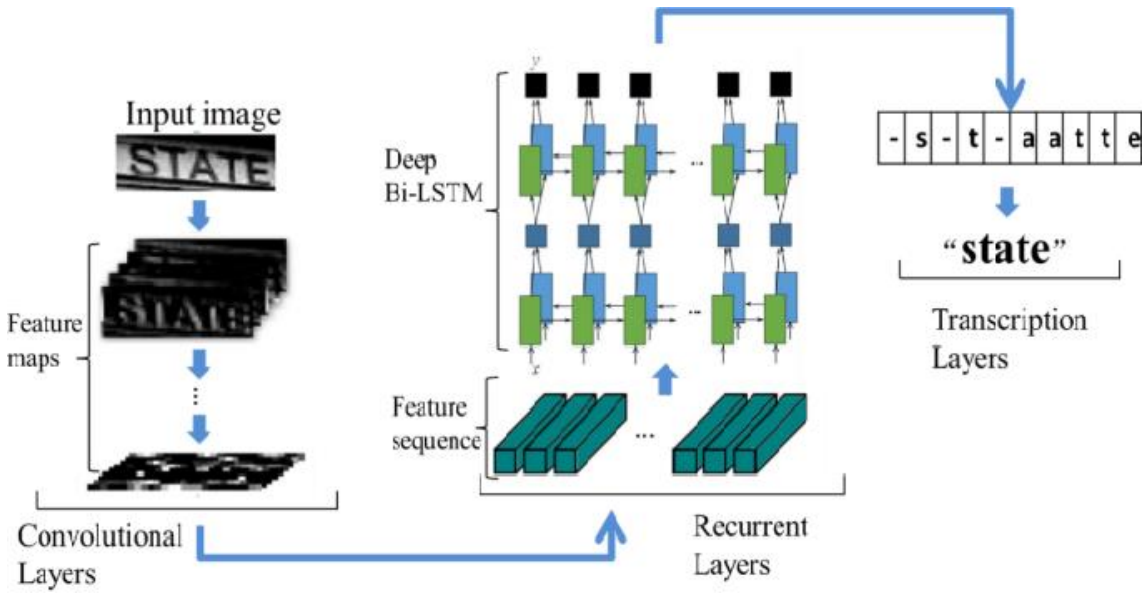


Figura 10 - Arquitectura básica de uma RNN [49]

As RNN são estruturalmente similares às redes *feedforward* (onde a informação flui apenas numa direção), porém com a inclusão de conexões no sentido contrário ao sentido em que a informação flui, de modo que os neurónios que processam as sequências recebem a cada intervalo de tempo, tanto a informação de saída do neurónio produzida no instante de tempo anterior quanto a informação de entrada no tempo. A resposta do neurónio no instante de tempo anterior é utilizada para se calcular a saída no instante atual, daí o nome rede recorrente.

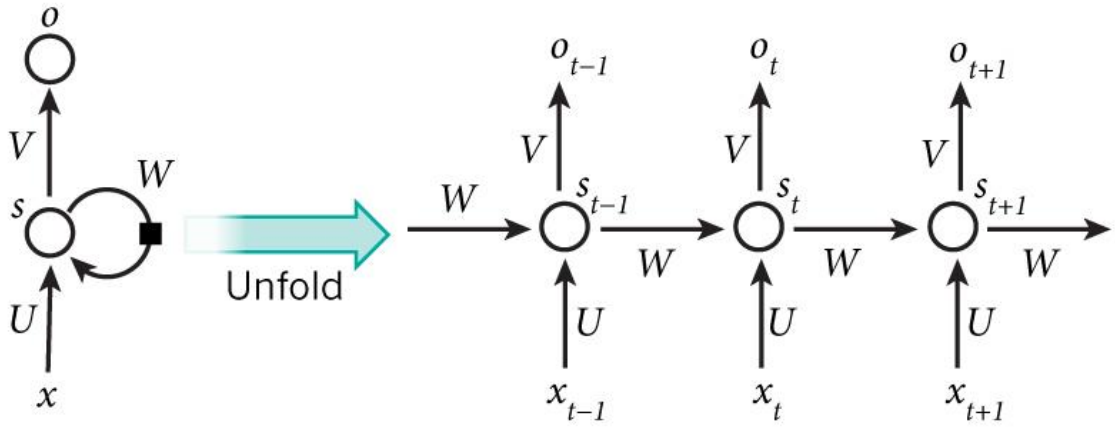


Figura 11 - Diagrama de funcionamento de um neurónio recorrente ao longo do tempo [50]

Nos últimos dois anos, as RNN estão a ser aplicadas a uma variedade de problemas, tais como tradução automática e o reconhecimento da fala. Ao falarmos iniciamos um processo dinâmico, e assim, as RNN parecem uma boa escolha em detrimento das tradicionais arquiteturas de *feedforward*. No entanto, a aplicabilidade das RNN é limitada devido a algumas razões.

A primeira é que as RNNs requerem um treino pré-segmentado e um pós-processamento do resultado de forma a convertê-lo numa sequência de dados corretamente etiquetada. De forma a resolver essa limitação a *Connectionist Temporal Classification* (CTC) pode ser usada em conjunto com as RNN para etiquetar a sequência de dados. O método CTC tem provado ser útil em situações onde o alinhamento entre as *labels* de entrada e saída são desconhecidas [37].

Segundo, para as dependências entre dados onde o intervalo entre a informação relevante e o local onde são necessários é grande, as RNNs têm uma utilização limitada.

Além desses dois problemas ainda nos podemos deparar com a questão de desaparecimento e explosão de gradiente (*gradient vanishing* e *gradient exploding*) e a questão de retenção de memória em sequências longas, quando a rede tende a "esquecer" as informações aprendidas nas primeiras etapas do treino. A questão da explosão e desaparecimento de gradiente é um problema intrínseco de redes neuronais muito complexas e afeta especialmente redes recorrentes porque no contexto dessa classe de redes neuronais não é possível utilizar técnicas que lidam com esses problemas em redes *feedforward*, como a simples adição de camadas de *BatchNormalization* e *Dropout*, conforme demonstrado por Laurent [38].

No caso das redes recorrentes, a solução mais comum para os problemas falados anteriormente é a substituição dos neurónios recorrentes simples por neurónios com *gates* que irão controlar tanto a propagação do gradiente quanto a retenção de memórias de longo prazo. Existem duas estruturas que são mais comumente abordadas e utilizadas sendo elas os neurónios LSTM (*Long Short-Term Memory*) e os neurónios GRU (*Gated Recurrent Units*).

O primeiro, proposto por Hochreiter e Schmidhuber em 1997, adiciona um estado a mais na célula que carrega memórias de longo prazo, enquanto outro estado se encarrega das memórias de curto prazo [40]. A estrutura completa pode ser observada na figura 12.

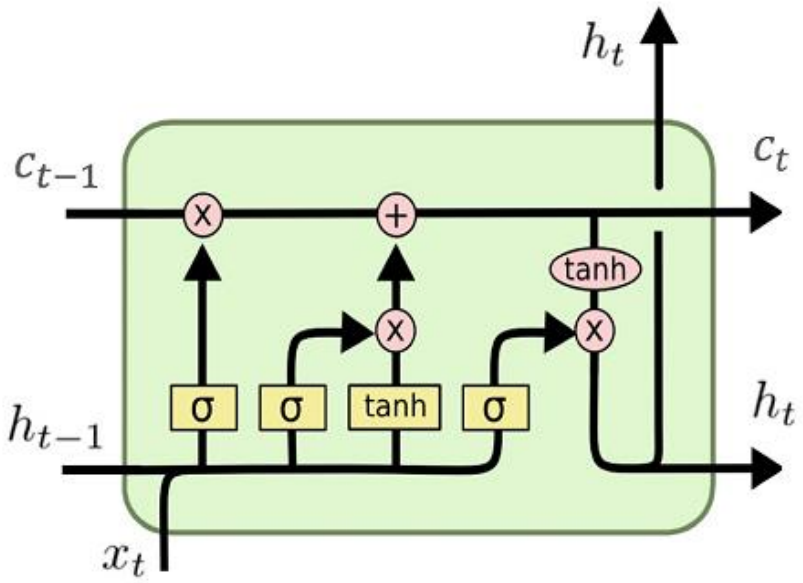


Figura 12 - Diagrama de um neurónio LSTM [52]

A estrutura de um neurónio LSTM é composta por três portas: entrada, saída e esquecimento (*input*, *output* e *forget*, respetivamente), sendo que a porta de esquecimento irá controlar quando as memórias de longo-prazo devem ser substituídas por memórias de curto-prazo. As portas de um LSTM em si também funcionam como camadas de uma rede neuronal e, ao longo do treino irão aprender quais memórias de longo-prazo e quais memórias de curto-prazo devem ser mantidas, ignoradas ou esquecidas [33].

O segundo tipo, os neurónios GRU, apesar de também serem usados com a mesma finalidade, apresentam algumas diferenças. Entre elas podemos destacar a sua menor complexidade, visto possuírem menor número de parâmetros, o que por sua vez se traduz num treino mais fácil e num menor uso de poder computacional. Além disso, e derivado da sua simplicidade, podem apresentar mais dificuldade em modelar dependências de longo prazo, logo, em casos onde a dependência de dados de longo prazo é critica, as mesmas podem não ser tão eficazes quanto as LSTM. Na figura 13 é mostrado um diagrama de um neurónio GRU.

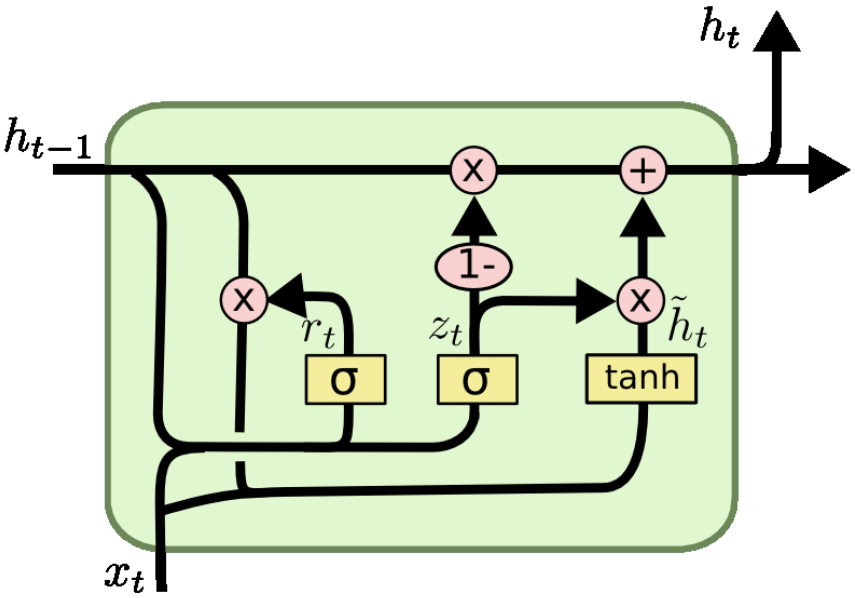


Figura 13 - Diagrama de um neurónio GRU [51]

2.1.3 Deep Learning

Atualmente, com a evolução rápida e constante das tecnologias ao nosso redor, uma das áreas com maior crescimento e evolução, é a área de *Deep Learning* (uma subárea de *Machine Learning*) e todas as áreas intrinsecamente ligadas à mesma. Os diversos algoritmos

disponíveis têm o potencial para estarem profundamente envolvidos em todos os campos do nosso quotidiano, desde a tomada de decisões e obtenção de conclusões a partir de conjuntos massivos de dados, até à possibilidade de nos permitirem controlar grande parte dos nossos dispositivos e aplicações apenas com recurso a comandos de voz.

Deep Learning é um termo utilizado para descrever uma classe específica de redes neuronais artificiais que são compostas por diversas camadas. Uma rede neuronal constituída por menos de três camadas é considerada uma rede neuronal básica, se esta for constituída por mais de três camadas é considerada um algoritmo em *Deep Learning*. Figura 14.

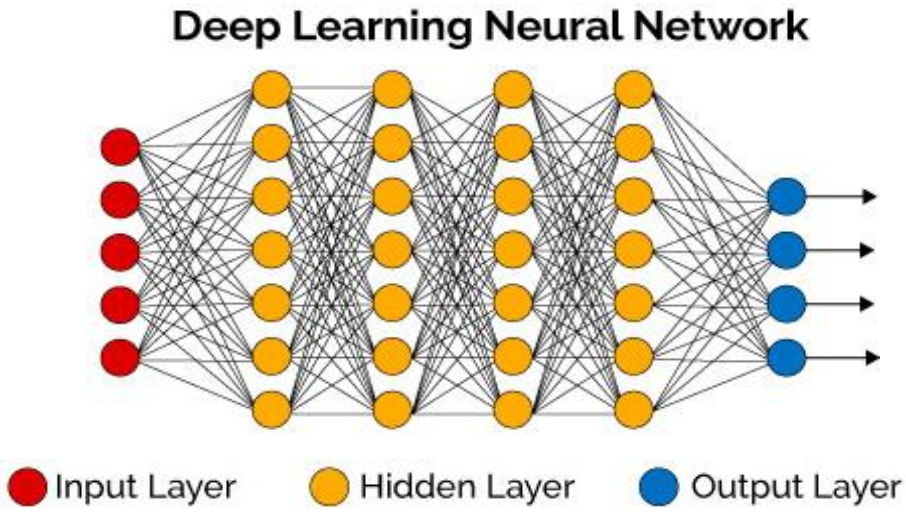


Figura 14 - Rede neuronal profunda (deep learning) [10]

As redes neuronais são compostas por neurónios que simulam como o cérebro humano trabalha e os neurónios são treinados de modo a ficarem mais "inteligentes", ou seja, terem mais capacidade para acertarem no resultado. Quantas mais camadas uma rede neuronal possuir melhor, pois vai ter mais neurónios para treinar e consequentemente mais ligações entre estes, tornando-os assim mais capacitados para prever situações futuras. As *Deep Neural Networks* existem há muitas décadas. No entanto, o treino das mesmas falhava até ao trabalho inovador de Geoffrey Hinton em 2006 [35].

No artigo escrito pelo mesmo, podemos observar que a sua principal contribuição, é a apresentação de um método de treino chamado *Contrastive Divergen*ce (CD) que é uma técnica de aprendizagem não supervisionada eficiente para *Deep Belief Networks* (DBN), que são uma outra classe de redes neuronais *deep learning*. O algoritmo CD simplifica o processo de treino em comparação com métodos anteriores, como o *Restricted Boltzmann*

Machine Learning (RBM), tornando o treino de DBNs mais rápido e escalável e menos computacionalmente intensivo. Dessa forma, o artigo em causa é considerado um marco importante no desenvolvimento de técnicas de treino eficientes para DNN's, contribuindo para a sua popularização e sucesso.

O *Deep Learning* permite aos modelos computacionais compostos de múltiplos níveis, aprender informações e representações com múltiplos níveis de abstração. Estes métodos melhoraram o estado da arte no reconhecimento da fala, reconhecimento visual de objetos, deteção de objetos e muitos outros domínios tais como a descoberta de medicamentos. O *Deep Learning* é capaz da descoberta de padrões complexos em grandes conjuntos de dados, utilizando o algoritmo de retro propagação, o que permite à rede modificar os seus parâmetros internos utilizados para calcular a representação em cada nível a partir da representação no nível anterior [12].

Os métodos de *deep learning* possuem muitas áreas aplicacionais, e diversos sucessos têm sido observados em particular na área do processamento de imagem. Um dos exemplos é a arquitetura de *deep learning* usada na aplicação que serviu como base para esta dissertação. Essa arquitetura é chamada de *Convolucional Neural Network* (CNN) e as mesmas são concebidas com o intuito de emular o comportamento do córtex visual. As CNNs desempenham muito bem qualquer tarefa de reconhecimento visual visto que a arquitetura da CNN consiste em camadas especiais chamadas camadas convolucionais e camadas de *pooling* que permitem ao algoritmo codificar certas propriedades das imagens.

Outra área onde o *deep learning* é aplicado com sucesso é o reconhecimento da voz ou da fala. Mas, neste caso e em comparação ao reconhecimento visual possuímos em mãos uma dificuldade adicional, a necessidade de trabalhar também dados temporais, tais como métricas de fala diferentes (velocidade do diálogo), e dependências temporais visto que as *Deep Neural Networks* só conseguem modelar um período acústico fixo não sendo muito eficazes com velocidades de diálogo diferentes.

2.2 Frameworks e Bibliotecas

Ao longo dos anos e de forma a que a investigação nesta área fosse mais acessível e simples de implementar, foram sendo desenvolvidas diversas bibliotecas, especialmente em Python, que permitem aos curiosos, estudantes ou pesquisadores, implementar diversos modelos de redes neuronais de diversas formas [20].

Em seguida falaremos sobre algumas das utilizadas no projeto e algumas das mais populares atualmente. Falaremos também das suas vantagens, características e como podem ser utilizadas.

2.2.1 Tensorflow

A *framework* TensorFlow, amplamente utilizada em ambientes académicos e profissionais, disponibiliza uma serie de ferramentas para computação numérica, desenvolvidas e melhoradas para aplicações de *Machine Learning* de grande escala como por exemplo classificação de imagens, processamento de linguagem natural, sistemas de recomendação, entre outros.

O TensorFlow cria gráficos de fluxo de dados (Figura 15), ou seja, aceita entradas como *arrays* multidimensionais sobre o qual é construído um gráfico de fluxo de dados que representam as operações às quais os dados de entrada serão submetidos [21].

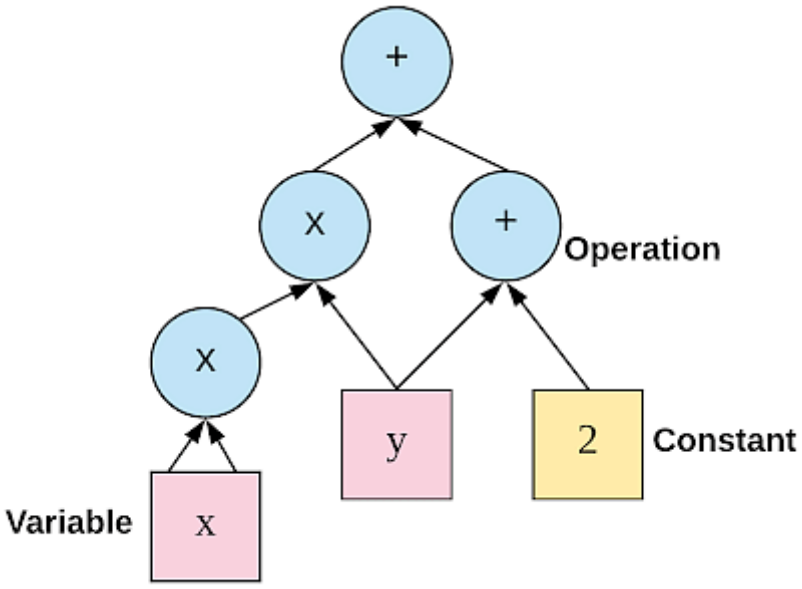


Figura 15 - Gráfico de fluxo de dado [21]

É uma plataforma de código aberto *end-to-end* concebida para *Machine Learning*. A biblioteca TensorFlow contém funções capazes de executar redes neuronais da forma mais precisa e eficiente possível. Possui também excelente documentação e um bom conjunto de exemplos que a tornam uma das melhores escolhas para este tipo de projeto. Tem um ecossistema abrangente e flexível de ferramentas, bibliotecas e recursos comunitários que

permitem aos investigadores e aos programadores construir e implementar facilmente aplicações alimentadas por ML.

A aplicação da arquitetura do TensorFlow é feita em três etapas distintas:

- Pré-processamento dos dados, que estrutura os dados de entrada.
- Construção do modelo onde se aplicaram os dados.
- Treino e avaliação do modelo.

TensorFlow permite realizar cálculos utilizando o *Central Processing Unit* (CPU) ou o *Graphics Processing Unit* (GPU), sendo para isso necessário verificar o sistema operativo a utilizar e a placa gráfica disponível antes de selecionar a versão do TensorFlow a instalar. Este projeto foi desenvolvido com a versão 2.9.1 pois, apesar da versão 2.10.7 ser a última versão a suportar GPU NVidia em Windows nativo, a mesma possuía um *bug* ao tentar guardar o modelo durante o treino, no nosso caso, usando o modelo pré-treinado da EfficientNetB7 [22] [23] [24].

2.2.2 Keras

Com o número crescente de utilizadores a entrar no mundo das DNNs, cada vez mais comuns no funcionamento dos programas atuais, a complexidade das mesmas tem sido sempre uma barreira para novos programadores. Tem havido muitas propostas de APIs simplificadas para a modelação de redes neuronais, e o Keras é uma delas. Keras é uma API de redes neuronais de alto nível, desenvolvida em Python.

2.2.3 Scikit-Learn

Scikit-learn é uma biblioteca Python que fornece muitos algoritmos de aprendizagem não supervisionados e supervisionados e recorre ao uso de outras bibliotecas já desenvolvidas como tais como NumPy, pandas e Matplotlib.

Algumas das características fornecidas por esta biblioteca são: Regressão (como Regressão Linear e Logística), Classificação, *Clustering*, Seleção de Modelos e Pré-Processamento (como Normalização Min-Max).

Scikit-learn é uma popular biblioteca de código aberto para Python. Esta é construída com base noutras bibliotecas como NumPy, SciPy e Matplotlib, que são usadas comumente em computação científica [25]. A biblioteca Scikit-learn oferece vários algoritmos para

problemas de aprendizagem supervisionada e não supervisionada, bem como ferramentas para processamento de dados e para avaliação e seleção de modelos.

Esta biblioteca é conhecida pela facilidade de uso, pelo seu desempenho e escalabilidade. Tal como o TensorFlow, o Scikit-learn é usado tanto em ambientes académicos como profissionais e tem uma comunidade grande e ativa que contribui com o seu desenvolvimento e documentação.

No âmbito deste projeto, esta biblioteca foi usada maioritariamente pelo seu amplo leque de métricas para problemas de classificação e pela facilidade de implementação de técnicas como o *Cross Validation*.

2.2.4 Pyautogui

A Pyautogui é uma biblioteca em Python que fornece uma maneira simples e fácil de automatizar tarefas em sistemas operativos Windows, macOS e Linux, através do rato e do teclado. É especialmente útil para a automação de tarefas repetitivas, testes de software, simulação de interações do utilizador e outras atividades que envolvem a manipulação da *Graphical User Interface* (GUI).

Ao usarmos esta biblioteca no desenvolvimento do programa pudemos assim possibilitar o movimento do cursor do rato, clicar em elementos do ecrã e também controlar a sua velocidade de movimento.

2.2.5 Multiprocessing

A biblioteca multiprocessing é um módulo em Python que facilita a programação concorrente (execução simultânea de tarefas) e paralela (execução de tarefas em múltiplos núcleos de CPU). A mesma fornece um conjunto de classes e funções para criar processos paralelos, permitindo-nos assim aproveitar eficientemente os múltiplos núcleos do CPU.

Posteriormente neste artigo será explicado em maior detalhe como as potencialidades desta biblioteca foram usadas.

2.3 Seleção de Modelos

O conceito de um bom modelo preditivo consiste em desenvolver um modelo robusto, com boa capacidade de generalização e que desempenha a sua função com rendimento desejado. É neste sentido que o processo de seleção de modelos se revela importante. No

contexto deste projeto aplicaram-se métodos de reamostragem, com o objetivo de averiguar a robustez dos modelos gerados, e métricas de desempenho baseadas em avaliação de hipóteses.

2.3.1 Métodos de Reamostragem

Métodos de reamostragem são ferramentas que permitem extrair uma ou mais amostragens de um *dataset* aos quais se aplicam processos de modelação ou estatísticos, com o intuito de obter métricas de desempenho mais robustas [15].

Existem diferentes métodos baseados no conceito de reamostragem, cada um com as suas características próprias, vantagens, desvantagens e cenários de aplicação [16]. Os métodos são os seguintes:

- Resubstituição
- Holdout
- Leave-one-out
- Cross-Validation
- Bootstrap

No contexto deste projeto utilizou-se apenas o método *Holdout* visto os resultados obtidos serem melhores do que o esperado. O seu funcionamento é abordado de seguida.

2.3.1.1 Holdout

O método *Holdout* consiste em dividir o dataset em dois conjuntos independentes, um conjunto de treino e um conjunto de teste. O conjunto de treino utiliza-se para realizar o processo de treino do modelo, e o conjunto de teste permite avaliar o desempenho do modelo

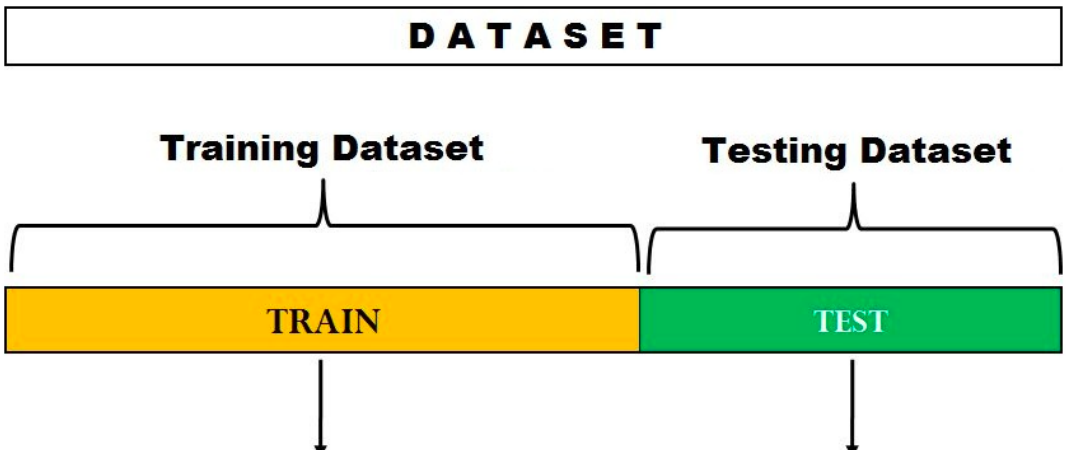


Figura 16 - Método *Holdout* [17]

com dados novos. Apesar de não existir uma regra definida para a divisão do *dataset*, é frequente encontrar divisões de 80% e 70% para o conjunto de treino e 20% e 30% para o conjunto de teste, respetivamente.

É prática frequente dividir o conjunto de teste em dois, de modo a se poder realizar uma validação do modelo com o objetivo de afinar os parâmetros do mesmo. No entanto, neste cenário a divisão do *dataset* original deve ser realizada de forma a que os três conjuntos possuam um número de amostras adequado, ou seja, essa abordagem, só deve ser efectuada caso o *dataset* em uso seja constituído por um número de amostras elevados pois, caso seja reduzido, os três conjuntos resultantes irão ser demasiado limitados podendo afetar o desempenho do treino e validação. No caso deste projeto, visto possuirmos apenas 1100 amostras, optamos pela abordagem de apenas dividir as nossas amostras em dois conjuntos.

Este método possui a vantagem de ser rápido e computacionalmente menos exigente visto que o processo de treino apenas se realiza uma vez, no entanto o desempenho do modelo está dependente da forma como o *dataset* é dividido podendo por vezes perder-se alguma informação na divisão.

2.3.2 Métricas de Avaliação

As métricas de avaliação têm como objetivo dar uma indicação do desempenho do modelo, e devem ser selecionadas de acordo com o tipo de problemas, classificação ou regressão, que o modelo visa a responder. De seguida são abordadas as métricas, referentes a problemas de classificação, utilizadas no desenvolvimento deste projeto.

2.3.2.1 Avaliação de hipóteses

Permite definir para cada classe do atributo objetivo o tipo de acerto ou o tipo de erro obtido pelo modelo para cada amostra. Disponibiliza quatro hipóteses distintas que possuem o seguinte significado:

- VP (Verdadeiro Positivo): amostra positiva e classificada como positiva
- VN (Verdadeiro Negativo): amostra negativa e classificada como negativa
- FP (Falso Positivo): amostra negativa e classificada como positiva
- FN (Falso Negativo): amostra positiva e classificada como negativa

É possível derivar da avaliação de hipóteses diversas métricas de desempenho que dão enfâse a hipóteses distintas, e cuja seleção depende do que se pretende avaliar. É de realçar que o significado dos FP e dos FN é distinto e difere para o tipo de problemas que se analisa.

2.3.2.2 Matriz de Confusão

A matriz de confusão permite obter uma representação visual das avaliações de hipóteses, relacionando a previsão com a realidade. Para além de indicar o número de amostras classificadas corretamente e incorretamente, também indica o número total de amostras classificadas e o número de amostras por classe. A Figura 17 representa uma matriz para um problema binário.

		Realidade	
		Sim	Não
Previsão	Sim	VP	FP
	Não	FN	VN

Figura 17 - Matriz de confusão com duas classes

Um bom modelo apresenta valores de VP e VN elevados, e valores de FP e FN o mais próximo de zero possível.

A representação de um problema com três ou mais classes também pode ser realizada com matrizes de confusão, apesar de a sua interpretação ser mais complexa. A Figura 18 representa uma matriz com três classes, A, B e C, onde as variáveis a verde representam previsões corretas e as variáveis a vermelho as previsões erradas.

		Realidade		
		A	B	C
Previsão	A	Vaa	Fab	Fac
	B	Fba	Vbb	Fbc
	C	Fca	Fcb	Vcc

Figura 18 - Matriz de confusão com três classe

2.3.2.3 *Accuracy*

Uma das métricas mais utilizadas em problemas de classificação é a *accuracy*, que descreve a forma como o modelo se comporta para todas as classes medindo a razão entre todas as previsões verdadeiras e todas as previsões obtidas. É uma métrica útil quando utilizada em problemas em que as classes do problema se encontram equilibradas, caso isto não se verifique deve-se utilizar métricas adicionais [18] [19].

A *accuracy* é calculada aplicando a Equação (1), onde i representa o número de classes.

$$\text{Accuracy} = \frac{\sum_1^i VP + \sum_1^i VN}{\sum_1^i VP + \sum_1^i VN + \sum_1^i FP + \sum_1^i FN} \quad (1)$$

2.3.2.4 *Precision*

Para obter informação da capacidade que o modelo apresenta na classificação de amostras positivas utiliza-se a métrica *precision*. A fórmula para a determinação da *precision* num problema binário está representada em (2).

$$Precision = \frac{VP}{VP + FP} \quad (2)$$

Num cenário onde o número de classes é superior a dois, é possível calcular a *precision* de duas formas distintas: *macro*, onde se realiza o cálculo da precisão por classe e no fim se calcula a média correspondente, e *micro* onde é feito o cálculo da precisão geral usando o somatório dos VP e o somatório dos VP e FP de todas as classes. Em suma, a *Precision Macro* tem como foco a *precision* média de cada classe, considerando-as todas de igual forma. Já a *Precision micro* considera a *precision* geral, agregando todos os casos de VP e FP para todas as classes.

2.3.2.5 *Recall*

A métrica *recall* analisa a forma como as amostras da classe positiva são classificadas pelo modelo. A fórmula base, indicada abaixo, para o *recall* é a razão entre os VP e o somatório dos VP e FN (3).

$$Recall = \frac{VP}{VP + FN} \tag{3}$$

De forma semelhante à métrica anterior, para problemas com múltiplas classes, o *recall* global pode ser calculado utilizando metodologias *micro* e *macro*. Sendo o *micro* calculado usando uma abordagem mais generalizada onde se utiliza a soma de todos os VP e a soma de todos os VP e FN de todas as classes e o *macro* sendo calculado para cada classe fazendo se posteriormente a média entre todos os resultados.

2.3.2.6 F1 *Score*

O F1 *Score* permite combinar as métricas *precision* e *recall* num só valor, sendo a média harmónica destas o resultado. A fórmula do F1 *Score* é encontrada em (4).

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

Da mesma forma que as métricas *precision* e *recall*, o F1 *Score* também pode ser calculado usando metodologias *micro* e *macro*, em problemas com mais de duas classes.

2.4 Performance de diferentes modelos

Através de um estudo de performance efetuado em 2019 entre três tipos diferentes de DNNs avaliadas num cenário igual (RNN, LSTM, GRU) o grupo responsável pelo desenvolvimento do estudo chegou a conclusão de que através de uma implementação de rede neuronal com uma arquitetura LSTM, os resultados obtidos eram mais promissores, obtendo uma menor percentagem de WER (*Word Error Rate*), assim como valores de perda (*Loss*) inferiores [36].

Table 1. Results for 500-node layer architecture

Model	WER (%)	Loss	Mean edit distance
RNN	87.02	186.61	0.4484
LSTM	77.55	160.51	0.3853
GRU	79.39	162.22	0.3939

Table 2. Results for 1,000-node layer architecture

Model	WER (%)	Loss	Mean edit distance
RNN	78.66	164.60	0.3991
LSTM	65.04	134.35	0.3222
GRU	67.42	136.89	0.3308

Figura 19 - Resultados dos testes efetuados entre RNN, LSTM e GRU [36]

Num segundo estudo efetuado para comparar a performance de diversos tipos de arquiteturas de redes neuronais na deteção de determinadas emoções na voz, foram usados três tipos de arquiteturas, CNN, LSTM e um modelo híbrido (TimeDistributed CNN). Após 535 iterações, chegou-se a conclusão de que os modelos baseados na arquitetura CNN foram bastante mais eficazes do que o LSTM, tendo o modelo em LSTM atingido aproximadamente 79.9% de precisão, o CNN, 87.7% de precisão e por fim o modelo híbrido

Parameters	Proposed Model	System [4] pt. 1
Architecture	Time Distributed CNN + LSTM	CNN
Dataset	RAVDESS + SAVEE + TESS	EmoDB
Features Used	Log Mel-Scale Filterbanks	Mel Spectrogram
Dataset Distribution	7720 and 1450	271 and 68
Accuracy	**89.26 %**	**78.16 %**

Parameters	Proposed Model	System [4] pt. 2
Architecture	Time Distributed CNN + LSTM	Bi-LSTM
Dataset	RAVDESS + SAVEE + TESS	Iemocap
Features Used	Log Mel-Scale Filterbanks	Mel Frequency Cepstral Coefficient
Dataset Distribution	7720 and 1450	4424 and 1107
Accuracy	**89.26 %**	**46.21 %**

Figura 20 - Resultados dos testes efetuados entre CNN, LSTM e Arquitetura Híbrida [45]

com 88%. Apesar de o foco de estudo neste caso em concreto ser a extração de emoção,

visto ainda que o tipo de input é vocal, consideramos resultados interessantes a ter em conta em futuras implementações [45].

Já em 2020, foi proposto por um grupo de investigadores, o uso do modelo EfficientNet, desta vez aplicado ao reconhecimento de voz. Para tal eles recorreram ao uso de diversos espectrogramas de Mel obtidos a partir de um *Dataset* da Google e constituído por dezenas de comandos de voz. Após o treino do modelo e a sua testagem, foram obtidos resultados de precisão na ordem dos 96%, sendo assim o estudo com maior percentagem de precisão encontrado durante a pesquisa por trabalhos relacionados [46].

Epochs	Training accuracy	Training loss	Testing accuracy	Testing loss
1	0.3482	1.8030	0.6000	1.2660
18	0.9423	0.1869	0.9644	0.1118

Figura 21 - Resultados da investigação que recorreu ao uso da arquitetura EfficientNet [46]

3 Reconhecimento de Voz/Palavras

De um modo geral e simplificado, um sistema de *Automatic Speech Recognition* (ASR) pode ser interpretado como uma função que recebe como entrada uma frase ou um comando simples e tem como saída uma palavra ou um conjunto de palavras que correspondam ao que foi dito.

3.1 História

De todas as formas que o ser humano é capaz de se expressar sem dúvidas a fala é a mais natural e intuitiva. Desde o momento em que nascemos somos expostos a um mundo de sons e símbolos linguísticos que começam a formar a nossa habilidade de nos comunicarmos através da imitação e associação. A língua é um elemento de interação entre o indivíduo e a sociedade a que ele pertence. Produzir, compreender sinais sonoros e atribuir a eles significados é uma das habilidades mais fundamentais que desenvolvemos para o convívio em sociedade.

Visto essa ser uma forma tão natural para o ser humano se expressar e comunicar, era de se esperar que desde o início do desenvolvimento da tecnologia da informação fossem procuradas formas de utilizar a fala com o intuito de interagir com os diversos dispositivos e aplicações existentes. Já no início dos anos 50, alguns pesquisadores dos Laboratórios Dell desenvolveram um sistema capaz de reconhecer dígitos isolados falados por um único orador através da correlação de frequências especificas produzidas pelo trato vocal humano ao pronunciar os números. Nos anos 60 e 70 foram feitos avanços que permitiram o aumento do vocabulário desses sistemas, porém ainda com técnicas baseadas em reconhecer padrões no domínio das frequências [32].

Nos anos 80 ocorreram duas mudanças que viriam a mostrar-se muito importantes para o desenvolvimento da tecnologia. A primeira delas foi uma mudança na forma de abordagem do problema em causa, sendo a mesma alterada para uma forma mais estatística, recorrendo à utilização dos *Hidden Markov Models* (HMM). Assumindo que a voz é uma junção de diversos sinais fixos de curta duração, os HMMs podem ser usados para prever as transições entre esses mesmos sinais e, através disso, realizar o reconhecimento da fala. A segunda grande mudança veio colmatar uma das "fraquezas" dos HMM: a ineficiência para lidar com não linearidades. Desde o momento de descoberta das Redes Neuronais, as mesmas foram utilizadas devido à sua capacidade de aproximar qualquer função, tanto lineares como não-

lineares, tornando se assim o complemento perfeito para os HMM. Essas duas técnicas em conjunto estabeleceram-se como um dos modelos clássicos para aplicações de reconhecimento de fala, até nos dias de hoje [33].

Porém, mais recentemente outras técnicas começaram a ganhar destaque, chegando mesmo a substituir a preferência pelos modelos híbridos de HMM/RN. A esse conjunto de técnicas foi dado o nome de *Deep Learning*. O conceito de *Deep Learning* foi apresentado por Hinton para mostrar a eficácia resultante de combinar técnicas de treino não-supervisionado de forma a treinar camadas interiores e realizar ajustes nas camadas finais de uma *Deep Neural Network* (DNN) [34].

As principais aplicações onde se observa o reconhecimento automático de fala atualmente dividem-se nos seguintes tópicos:

- Transcrição Automática de Áudio (para, por exemplo, gerar legendas automáticas para vídeos).

- Identificação de Palavras-Chave (muito utilizado para ativar Assistentes Digitais).

- Reconhecimento de comandos de voz, objetivo principal de estudo deste trabalho.

No caso de um modelo voltado para reconhecimento de comandos de voz, é necessário que o mesmo seja robusto, principalmente a diferentes oradores e possivelmente diferentes ambientes, uma vez que para cada ser humano a pronúncia de palavras varia em velocidade, volume, frequência, variações vocais, articulação e ainda assim devem ser identificadas como sendo a mesma palavra.

A tecnologia de reconhecimento de fala é avaliada através da sua taxa de precisão, ou seja, taxa de erro de palavra (WER) e velocidade. Assim, os fatores falados anteriormente podem impactar essa mesma taxa de erro. Alcançar a paridade humana, que significa uma taxa de erro equivalente à obtida numa conversa entre dois humanos, tem sido o principal objetivo dos sistemas de reconhecimento de fala [47].

3.2 Técnicas de Reconhecimento

Diversos algoritmos e técnicas de computação são usados para reconhecer a fala e melhorar a precisão da transcrição. A seguir estão breves explicações de alguns dos métodos mais usados:

- *Natural Language Processing* (NLP): embora o NLP não seja necessariamente um algoritmo específico usado no reconhecimento de fala, é a área de inteligência artificial que se concentra na interação entre humanos e máquinas por meio da linguagem. Muitos dispositivos móveis incorporam reconhecimento de voz no seu software de forma a realizarem procuras por voz.

- *Hidden Markov Models* (HMM): criam o modelo de cadeia de Markov, que estipula que a probabilidade de um determinado estado depende do estado atual, e não dos anteriores. São utilizados como modelos de sequência no reconhecimento de fala, atribuindo *labels* a cada unidade, ou seja, palavras, sílabas, frases etc., na sequência. Estas etiquetas criam um mapeamento com a entrada fornecida, permitindo que seja determinada a sequência de *labels* mais adequada.

- N-grams: este é o tipo mais simples de modelo de linguagem, que atribui probabilidades a frases. Um N-gram é sequência de N-words. Por exemplo, "encomenda a pizza" é um trigrama ou 3-gram e "encomenda a pizza grande" é um 4-gram. Gramática e a probabilidade de certas sequências de palavras são usadas para melhorar o reconhecimento e a precisão.

- Redes neuronais: as redes neuronais, em especial os modelos de *Deep Learning*, processam diferentes dados de treino, na tentativa de imitar a interconectividade do cérebro humano através de diversas camadas de nós. Cada nó é composto de entradas, pesos, um limite e uma saída. Se esse valor de saída exceder um determinado limite, ele "dispara" ou ativa o nó, passando dados para a camada seguinte na rede. As redes neuronais aprendem essa função de mapeamento por meio da aprendizagem supervisionada, ajustando-se com base na função de perda. Embora as redes neuronais tendam a ser mais precisas e possam aceitar mais dados, isso tem um custo de eficiência de desempenho, visto que tendem a ser mais lentas para treinar em comparação com os modelos de linguagem tradicionais.

4 Modelos Desenvolvidos

Durante a realização deste projeto para além do objetivo de obter uma elevada precisão no treino do nosso modelo de forma ao mesmo poder ser usado para potenciar o controlo de uma aplicação através da voz, tivemos também, como objetivo secundário, averiguar as diferenças, que três modelos distintos, têm no desempenho de aprendizagem de palavras por meio de deteção de padrões em espectrogramas.

Os três modelos desenvolvidos consistem num modelo baseado em camadas Conv2D que realizam a convolução espacial da entrada. Um deles, desenvolvido por nós. Outro, designado por Resnet50 e outro denominado EfficientNet baseado em blocos constituídos por diversas camadas.

Serve o presente capítulo para descrever os vários modelos desenvolvidos e a estrutura implementada para o treino, validação, teste dos modelos e também elucidar sobre o desenvolvimento do *Dataset* e como foram obtidos os diversos espetrogramas.

4.1 Estrutura dos modelos

4.1.1 Conv2D

A aplicação de camadas Conv2D em problemas de classificação de imagens, tem apresentado bons resultados recentemente [28]. O código implementado e a descrição do código podem ser observados na Figura 22.

```
23  model = Sequential()
24  model.add(Conv2D(4, (3,3), activation='relu', input_shape=(img_height, img_width,3)))
25  model.add(tf.keras.layers.LayerNormalization(axis=-1 , center=True , scale=True))
26  model.add(Flatten())
27  model.add(Dense(80, activation='relu'))
28  model.add(Dropout(0.3))
29  model.add(tf.keras.layers.LayerNormalization(axis=-1 , center=True , scale=True))
30  model.add(Dense(80, activation='relu'))
31  model.add(tf.keras.layers.LayerNormalization(axis=-1 , center=True , scale=True))
32  model.add(Dense(10, activation='softmax'))
```

Figura 22 - Modelo Conv2D

A camada Conv2D, cujo funcionamento é abordado na secção CNN, apresenta 4 filtros que representam a dimensionalidade do espaço de saída da camada. Em seguida são especificadas as dimensões da janela de convolução 2D, neste caso o par de valores (3,3) especifica uma janela 3x3 (Figura 23).

Já com o parâmetro *activation='relu'*, é especificada a função de ativação usada nessa mesma camada, neste caso a RELU, bastante utilizada para introduzir não linearidade dentro da rede neuronal. Por fim, definimos o *input_shape* consoante os pixéis de altura e largura das imagens a serem carregadas na rede neuronal.

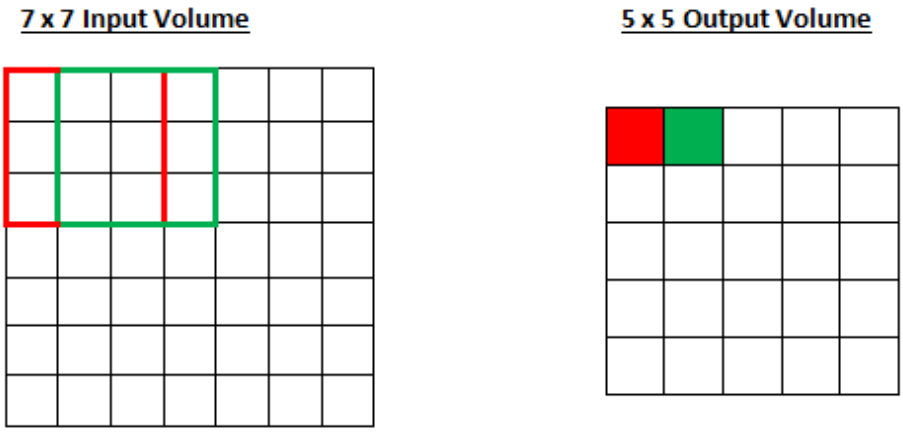


Figura 23 - Funcionamento da janela de convolução

Na camada seguinte é aplicada a normalização às saídas da camada anterior. Essa camada vai realizar a normalização ao longo do eixo especificado. Os argumentos *center=True* e *scale=True* permitem que as transformações afins aprendidas sejam aplicadas após a normalização.

A função da camada *Flatten* consiste em transformar as amostras que recebe, que neste cenário possuem uma forma de *img_height* x *img_width* x 3, num *array* unidimensional (Figura 24).

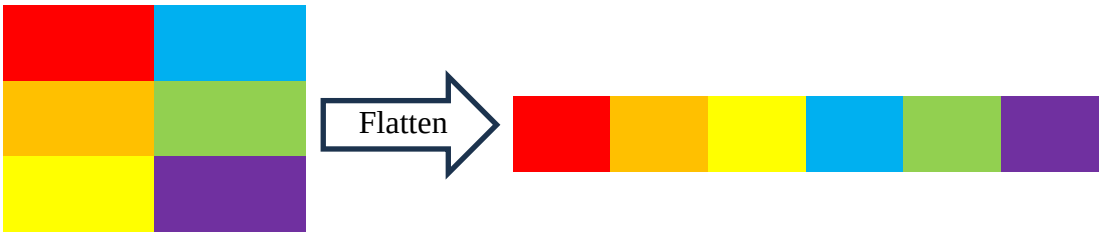


Figura 24 - Demonstração do funcionamento da função Flatten

O *array* unidimensional alimenta uma camada de redes neuronais densamente interligadas, designada *Dense*, onde se especifica a dimensão do espaço de saída e a função

de ativação que os neurónios desta camada implementam. Para esta camada esses valores são de 80 e “relu” respetivamente.

A utilização de uma camada de *Dropout* tem como objetivo minimizar o fenómeno de *overfitting*. Esta camada converte aleatoriamente um valor de entrada para 0, com base na taxa definida nos seus parâmetros de entrada. Os restantes valores de entrada são escalados aplicando a fórmula 1/(1-taxa), de forma a que o somatório dos valores de entrada se mantenha inalterado. A taxa definida para esta camada foi de 0,3.

De seguida é repetida a sequência de camadas *LayerNormalization*, *Dense* e *LayerNormalization*, com os mesmos parâmetros.

A última camada mapeia o espaço de saída para 10 classes distintas utilizando a função de ativação *softmax*. Esta função de ativação transforma os valores reais presentes no vetor de entrada numa distribuição de probabilidades que abrange todas as classes.

Depois de estruturar o modelo, é necessário proceder à sua compilação. Para isso é necessário definir o optimizador, a forma de calcular das perdas e a métrica a analisar. Selecionou-se o optimizador Adam que implementa uma variação do algoritmo Gradiente Descendente Estocástico [26], e que tem visto uma adoção significativa.

O cálculo das perdas realizou-se utilizando *SparseCategoricalCrossentropy* que permite ajustar os pesos do modelo durante o processo de treino e deve ser utilizado quando se tem duas ou mais classes como objetivo. [27]

A métrica utilizada na compilação do modelo foi a *accuracy* que permite observar quantas vezes o modelo esteve, na generalidade, correto (Figura 25).

```
36  model.compile(
37      optimizer='adam',
38      loss=tf.losses.SparseCategoricalCrossentropy(),
39      metrics=['accuracy'])
```

Figura 25 - Código da compilação dos modelos

O processo de compilação indicado é idêntico para os restantes modelos abordados neste capítulo.

4.1.2 Conv2D Com *Augmentation*

Este modelo foi estruturado exatamente como o anterior, visto que o objetivo era testar o desempenho do modelo aplicando diversas transformações as imagens do *Dataset*. Para tal foi necessário alterar a forma de carregamento das imagens e foi necessário também criar dois "geradores" que são os responsáveis por efetuar as transformações. Esse mesmo processo irá ser explicado em detalhe no subcapítulo seguinte.

4.1.2.1 *Augmentation*

Augmentation, em termos gerais no contexto do processamento de imagens, é uma técnica crucial para melhorar a diversidade e a qualidade dos dados de treino utilizados em algoritmos de *machine learning*.

Esse processo consiste em aplicar diversas transformações nas imagens originais, a fim de criar variações realistas que representem diferentes condições e perspetivas

Essas transformações podem incluir rotações, ampliações, reduções, reflexões, ajustes de brilho e contraste, entre outras. O objetivo principal da *augmentation* é mitigar o risco de *overfitting*, onde o modelo se adapta em excesso aos dados de treino específicos, não generalizando bem para novos exemplos. Ao introduzir essas variações, a *augmentation* ajuda o modelo a aprender características mais robustas e a se tornar mais capaz de lidar com variações naturais nos dados de entrada.

Para este trabalho em específico, utilizamos a classe *ImageDataGenerator* da biblioteca Keras. A mesma disponibiliza diversos parâmetros que podem ser utilizados para manipular as imagens. No entanto, visto estarmos a trabalhar com espectrogramas como dados de entrada, nem todos esses parâmetros são aplicáveis, pois iriam transformar as imagens de forma drástica, causando o efeito oposto do pretendido. Tais transformações seriam, por exemplo o zoom, pois ficaríamos sem parte da imagem, ou seja, estaríamos a cortar a palavra. A rotação com valores muito elevados pois estaríamos a alterar a frequência da palavra e a sua cadência temporal. E por fim qualquer tipo de espelhamento, que por sua vez modificaria por completo o espectrograma da palavra falada e por sua vez, a própria palavra a ser reconhecida.

Assim, mantivemo-nos por opções mais simples, que faziam sentido no contexto deste estudo e demonstraram ajudar ligeiramente no desempenho dos diversos modelos conforme será possível observar em capítulos futuros. Dentro desses parâmetros incluímos o

rotation_range = 2, que vai efetuar rotações ligeiras num conjunto de [-2,2] graus. Temos o parâmetro *height_shift_range* = 15, que vai efetuar transformações no eixo dos yy dentro do intervalo [-15,15] pixéis. Por fim temos os parâmetros *Samplewise_center* e *Samplewise_std_normalization*.

O primeiro baseia-se em calcular a média dos valores de cada amostra individualmente (imagem, matriz, etc.) e, em seguida, subtrair essa média de todos os valores dessa amostra. Isso tem o efeito de centrar cada amostra em torno do valor médio de zero, o que pode ser benéfico para melhorar a convergência durante o treino do modelo.

O segundo envolve dois passos: o primeiro é centrar a amostra (como explicado acima) e o segundo é dividir cada valor da amostra pelo desvio padrão daquela amostra específica. Como resultado cada amostra tem uma média zero e um desvio padrão de valor unitário.

```
train_datagen = ImageDataGenerator(height_shift_range=15,
                                   samplewise_center=True,
                                   samplewise_std_normalization=True,
                                   rotation_range=2)

validation_datagen = ImageDataGenerator(height_shift_range=15,
                                        samplewise_center=True,
                                        samplewise_std_normalization=
                                        rotation_range=2)
```

Figura 26 - Código da inicialização dos geradores de imagens

4.1.3 Efficient Net

Atualmente existem diversos modelos de redes neuronais pré-treinados de fácil implementação e que apresentam bons resultados. É neste contexto que se destaca um dos modelos pré-treinados que vamos abordar, o EfficientNetB7.

O EfficientNetB7 é um modelo de rede neuronal pré treinado que faz parte da família de modelos EfficientNet. Esse conjunto de modelos (B0-B7) foi projetada com o objetivo de alcançar um equilíbrio entre desempenho e eficiência computacional, permitindo treinar e implementar redes neurais maiores e precisas sem aumentar desproporcionalmente os recursos computacionais necessários [43].

Como pode ser observado na imagem abaixo o EfficientNet utiliza um bloco de construção chamado MBConv (*Convolutional Mobile Inverted Bottleneck*), que combina operações de convolução profunda com convoluções separáveis em camadas mais estreitas,

reduzindo drasticamente o número de parâmetros e operações. A medida que vamos aumentado o número de B, significa que estamos a aumentar o número de blocos que constituem esse modelo. Assim sendo o modelo utilizado, B7, e o que possui o maior número de blocos e por sua vez, maior poder de análise de características dos dados de input à custa de requerer mais poder computacional para o treino [44].

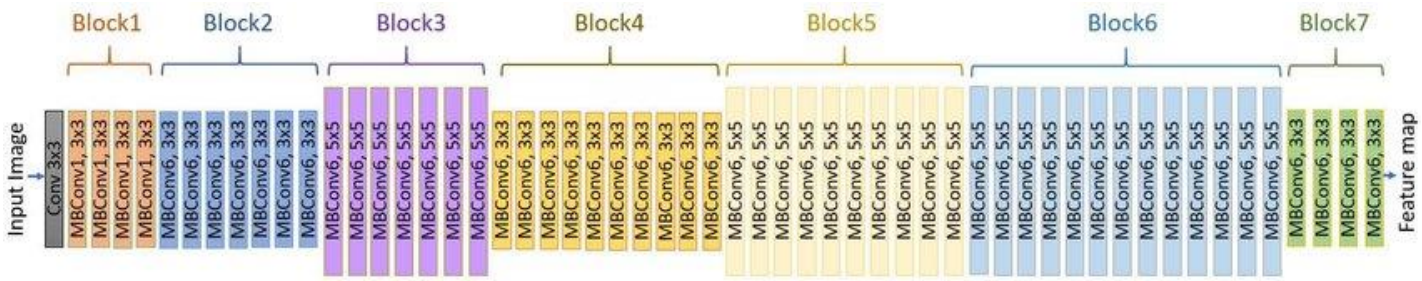


Figura 27 - Arquitectura do modelo EfficientNet [29]

O carregamento do modelo EfficientNetB7 realiza-se por intermédio da função do TensorFlow keras.applications.EfficientNetB7.

Tendo em conta que as amostras utilizadas neste processo possuem uma dimensão de *img_height* x *img_width* x 3, o parâmetro *include_top* deve ser inicializado a False. Esta opção remove a primeira camada neuronal, que está programada para receber como input dimensões diferentes das imagens do nosso *dataset*.

Em simultâneo à exclusão da camada inicial, é necessário especificar as dimensões das amostras com que se pretende treinar o modelo, através do parâmetro *input_shape* inicializado com *img_height* x *img_width* x 3, e o método de *pooling* para a extração de *features*, onde avg aplica o avg *pooling*.

O parâmetro *weights* especifica a forma como os pesos são inicializadas, o valor *imagenet* inicializa com os pesos obtidos durante o treino com *ImageNet*. O parâmetro classes permite especificar o número de classes para as quais se pretendem classificar as imagens. A parametrização deste modelo encontra-se na Figura 28.

```
23  pretrained_model=tf.keras.applications.EfficientNetB7(
24      include_top=False,
25      weights='imagenet',
26      input_tensor=None,
27      input_shape=(img_height, img_width,3),
28      pooling='avg',
29      classes=10)
30
31  model = Sequential()
32  model.add(pretrained_model)
33  model.add(Dense(10, activation='softmax'))
```

Figura 28 - Modelo EfficientNetB7

O modelo resultante é composto pelo modelo EfficientNetB7 com as configurações anteriores e uma camada dense que mapeia o espaço de saída para 10 classes distintas utilizando a função de ativação *softmax*.

4.1.4 ResNet50

Outro dos diversos modelos de redes neuronais pré-treinados abordados para fim de comparação foi o ResNet50 que é uma rede neuronal convolucional com 50 camadas de profundidade, capaz de classificar objetos em 1000 categorias distintas e treinado com o *dataset ImageNet*.

Devido ao seu desempenho, optou-se por treinar o modelo ResNet50 com o *dataset* utilizado neste projeto com intuito de ter uma métrica de comparação para os restantes modelos desenvolvidos. A arquitetura deste modelo pode ser observada na Figura 29.

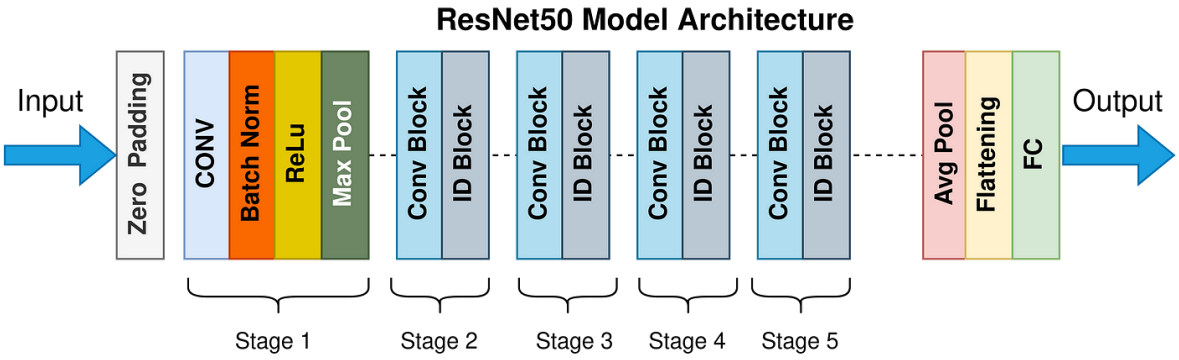


Figura 29 - Arquitetura do modelo ResNet50 [30]

O carregamento do modelo ResNet50 realiza-se por intermédio da função do TensorFlow keras.applications.ResNet50. Os parâmetros de entrada desta função são relevantes, em particular, quando as dimensões das imagens a analisar são diferentes das dimensões normais do modelo ResNet50 que são de 224x224x3.

Tendo em conta que as amostras utilizadas neste processo possuem uma dimensão de *img_height* x *img_width* x 3, o parâmetro *include_top* deve ser inicializado a False. Esta opção remove a primeira camada neuronal, que está programada para receber como input amostras 224x224x3.

Em simultâneo à exclusão da camada inicial, é necessário especificar as dimensões das amostras com que se pretende treinar o modelo, através do parâmetro *input_shape* inicializado com *img_height* x *img_width* x 3, e o método de *pooling* para a extração de *features*, onde avg aplica o avg *pooling*.

O parâmetro *weights* especifica a forma como os pesos são inicializadas, o valor *imagenet* inicializa com os pesos obtidos durante o treino com *ImageNet*. O parâmetro classes permite especificar o número de classes para as quais se pretendem classificar as imagens. A parametrização deste modelo encontra-se na Figura 30.

```
23  pretrained_model=tf.keras.applications.ResNet50(
24      include_top=False,
25      weights='imagenet',
26      input_tensor=None,
27      input_shape=(img_height, img_width,3),
28      pooling='avg',
29      classes=10)
30
31  model = Sequential()
32  model.add(pretrained_model)
33  model.add(Dense(10, activation='softmax'))
```

Figura 30 - Modelo ResNet50

O modelo final é composto pelo modelo ResNet50 com as configurações anteriores e uma camada dense que mapeia o espaço de saída para 10 classes distintas utilizando a função de ativação *softmax*.

4.2 Metodologia de treino e teste

O treino e a avaliação do modelo realizou-se sempre recorrendo ao método de *holdout*, com o intuito de obter uma métrica generalista do desempenho do modelo. Visto os resultados obtidos estarem quase sempre acima dos 95% (e chegando aos 100%) em várias métricas, não achamos necessidade de testar outras metodologias visto que a margem para melhoria era mínima.

De seguida apresenta-se a aplicação do método *holdout* apenas para o modelo Conv2D, visto que a sua implementação é feita de forma idêntica para os restantes modelos sendo apenas necessário alterar os modelos e as variáveis referentes à origem dos dados, nome dos modelos, diretório onde se gravam, nome dos mesmos e a função que carrega as diversas imagens do *dataset*.

4.2.1 *Dataset* Utilizado

Como falado anteriormente, o foco deste estudo foi encontrar uma solução inovadora para o reconhecimento de voz e para tal fugimos de *Datasets* baseados em idiomas extremamente estudados, abordados e para os quais já existem inúmeras soluções. Para tal decidimos realizar o treino do modelo com a nossa língua materna, o Português de Portugal.

Devido a escassez de *Datasets* nessa mesma língua foi necessário criar uma amostra do que seria um *Dataset* completo, recorrendo ao *script* que será abordado no subcapítulo seguinte de forma a gravar os diversos espectrogramas.

No final, foi obtido e utilizado um conjunto de espectrogramas dos números de 0 a 9, provenientes de 7 oradores (6 masculinos e 1 feminino), contendo 1100 imagens no total.

4.2.2 Criação de Espectrogramas e Pré-processamento de Áudio

De forma a criar o *dataset* utilizado ao longo de todos os testes, foi necessário desenvolver a nossa própria forma de criar espectrogramas a partir de áudio, quer seja ficheiro quer inputs diretamente no microfone. Para tal foi desenvolvido um *script* em Python que nos permitiu assim gerar os espectrogramas respetivos dos comandos/palavras falados.

Ao desenvolver o *script* tivemos em mente fazê-lo de forma que a sua utilização fosse o mais simples possível. Daí ao iniciar o mesmo é pedido ao utilizador um input do número a

ser falado, que para além de definir a nomenclatura dos ficheiros, irá também definir a diretoria a ser criada ou caso já exista, definir onde o espetrograma deverá ser gravado.

Após isso o microfone ficará à escuta de sinais sonoros. Caso os mesmos ultrapassem um determinado *threshold*, o processo de gravação será iniciado. Em seguida, assim que o *script* identificar silêncio por um determinado período a gravação é terminada. Posteriormente é efetuada a gravação da palavra em formato áudio para controlo (Figura 31) e é também gerado o espectrograma recorrendo a uma a uma biblioteca chamada Librosa (Figura 32).

```
scipy.io.wavfile.write(os.path.join(sampleDir,'{}_{}.wav'.format(palavra, sampleDir_n_files)),44100,r)
print ('Written to file')
```

Figura 31 - Código da gravação do áudio de controlo

```
melspec = librosa.feature.melspectrogram(y=r, sr=self.rate,
                                         n_mels=256,
                                         fmax=22000,
                                         hop_length=8)

norm_melspec = librosa.core.power_to_db(melspec, ref=np.max)
```

Figura 32 - Código para gerar o espectrograma correspondente ao áudio gravado

No final da conversão para espectrograma o mesmo é mostrado ao utilizador também para efeitos de controlo (Figura 33).

```
img = librosa.display.specshow(norm_melspec,
                               x_axis='time',
                               y_axis='mel',
                               sr=self.rate,
                               fmax=22000,
                               ax=ax)
```

Figura 33 - Código para mostrar o espetrograma gerado

Após ser fechada a janela que mostra a imagem gerada, o script irá novamente ficar à escuta de sinais sonoros e este ciclo irá ser efetuado infinitamente. À medida que isso acontece os novos espectrogramas irão ser gravados na diretoria mencionada até ser atingido

o limite de ficheiros definido para a diretoria de treino. Após isso os novos espectrogramas começam a ser gravados numa subdiretoria semelhante, mas desta vez na diretoria referente às amostras de validação. Figura 34 e 35

```
if not os.path.exists(os.path.join(trainingDir, palavra)):
  os.makedirs(os.path.join(trainingDir, palavra))
else:
  pass

if not os.path.exists(os.path.join(validationDir, palavra)):
  os.makedirs(os.path.join(validationDir, palavra))
else:
  pass
```

Figura 34 - Código para criar as diretorias caso as mesmas não existam

```
trainingDir_n_files = len(os.listdir(os.path.join(trainingDir, palavra)))
validationDir_n_files = len(os.listdir(os.path.join(validationDir, palavra)))

print(validationDir_n_files)

if trainingDir_n_files < trainingSamples:
  plt.savefig(os.path.join(os.path.join(trainingDir, palavra),
                           '{}_{}.png'.format(palavra,
                                              trainingDir_n_files)),
                                              transparent=True)
elif validationDir_n_files < validationSamples:
  plt.savefig(os.path.join(os.path.join(validationDir, palavra),
                           '{}_{}.png'.format(palavra,
                                              trainingDir_n_files+validationDir_n_files)),
                                              transparent=True)
```

Figura 35 - Código para gravar os espectrogramas nas respetivas diretorias

É também importante referir que relativamente ao áudio o mesmo foi captado com um rate de 44100 Hz e um *chunck size* de 1024 conforme a Figura 36.

```
def __init__(self):
  self.rate = 44100
  self.threshold = 0.014 # silence threshold
  self.chunk_size = 1024
  self.format = pyaudio.paFloat32
  self._pyaudio = pyaudio.PyAudio()
```

Figura 36 - Parâmetros de gravação dos sinais sonoros

Já relativamente aos espectrogramas, os mesmo foram gerados com 256 *Mels*, uma frequência máxima de 22000Hz e um *hop lenght* de 8 (Figura 37)

```
melspec = librosa.feature.melspectrogram(y=r, sr=self.rate,
                                         n_mels=256,
                                         fmax=22000,
                                         hop_length=8)
```

Figura 37 - Parâmetros de geração dos espectrogramas

O número de *Mels* determina a precisão com que o espetro de frequências é dividido em bandas distintas, ou seja, controla a resolução da representação.

Quando se escolhe um número mais elevado de filtros *Mel*, está essencialmente a criar uma representação mais detalhada e de maior resolução do conteúdo de frequência do áudio em termos da sua perceção de altura.

O parâmetro *fmax* define a frequência máxima a ser representada no gráfico.

Por fim o parâmetro *hop_lenght*, é utilizado no processamento de sinais de áudio ao criar representações de tempo-frequência, como os espectrogramas de *Mel*. Ele determina o intervalo de tempo entre janelas de tempo consecutivas usadas para analisar o sinal de áudio.

Um menor *hop lenght* resulta numa maior sobreposição entre fotogramas. Isto pode proporcionar uma maior resolução temporal porque está a captar informação sobre o sinal de áudio em pontos temporais mais espaçados a custa de maior processamento.

Um *hop lenght* maior resulta em menos sobreposição entre fotogramas. Isto pode proporcionar uma melhor resolução de frequência ao analisar partes distintas do sinal de áudio sem tanta sobreposição à custa de possível perda de eventos de curta duração ou mudanças rápidas no áudio.

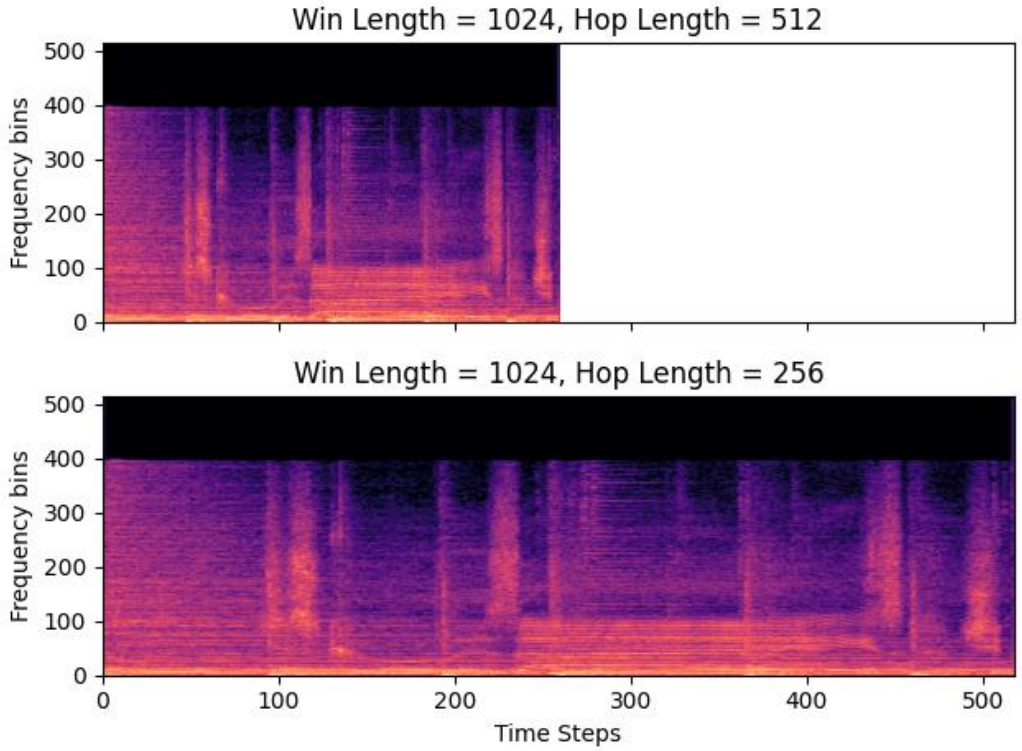


Figura 38 - Espectrograma resultante com hop lenght de 512 e 256

Visto que neste caso aplicacional ambas as vertentes tinham as suas vantagens, definimos o *hop lenght* num valor que pareceu oferecer um bom rácio entre resolução e menor necessidade de processamento.

4.2.3 Holdout

O processo que se desenvolveu para o treino dos modelos consiste em quatro etapas sequenciais: carregamento dos dados, preparação dos dados, treino e avaliação do modelo.

Para realizar o carregamento do *dataset* utilizou-se a função *image_dataset_from_directory* no caso inicial e nos casos onde foi utilizado *data augmentation* foi utilizada a função *flow_from_directory*, pois verificou se que a primeira função não era compatível com a classe geradora de imagens. Ambas as funções disponibilizam diversos parâmetros que permitem influenciar a forma como o *dataset* é carregado em memória (Figuras 39 e 40).

```
imagesTreino = tf.keras.utils.image_dataset_from_directory(
    training_dir,
    image_size=(img_height, img_width),
    batch_size=3)

imagesTest = tf.keras.utils.image_dataset_from_directory(
    validation_dir,
    image_size=(img_height, img_width),
    batch_size=3)
```

Figura 39 - Código do carregamento do *dataset*

```
train_generator = train_datagen.flow_from_directory(
    training_dir,
    target_size=(img_height, img_width),
    batch_size=3,
    class_mode='sparse',
)

validation_generator = validation_datagen.flow_from_directory
    validation_dir,
    target_size=(img_height, img_width),
    batch_size=3,
    class_mode='sparse',
)
```

Figura 40 - Código do carregamento do *dataset* para *Augmentation*

No presente cenário especificou-se o diretório que contem as imagens a carregar sejam elas para treino ou para validação. O valor específico de cada classe corresponde ao nome da pasta onde se encontram as amostras, como se pode observar pela estrutura do diretório na Figura 41.

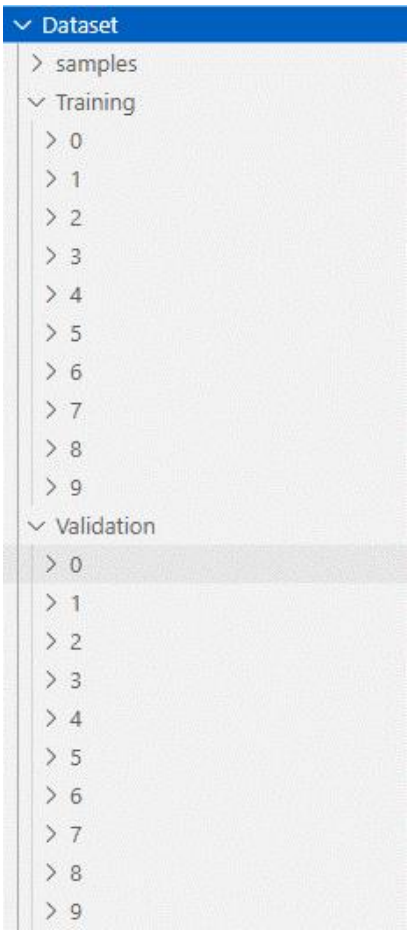


Figura 41 - Estrutura das diretorias

A especificação das dimensões das imagens, *image_size* ou *target_size*, é um parâmetro obrigatório e deve ser ajustado às imagens presentes no *dataset* a utilizar. Caso sejam introduzidas dimensões erradas, a função redimensiona as imagens o que se pode traduzir na perda de informação.

A importação dos dados pode ser realizada em lotes configurando o parâmetro *batch_size* com o valor pretendido. Neste caso foi definido um *batch_size* de 3.

As funções geram dois objetos *Dataset*. Um com 770 amostras para treino e outro com 330 amostras para validação. Ambos os conjuntos de amostras encontram se distribuídos por 10 classes distintas.

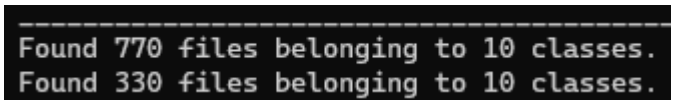


Figura 42 - Número de ficheiros e classes encontradas pelas funções de carregamento de dados

Para a implementação do método de avaliação *holdout* aplicou-se uma divisão de 70% dos dados para realizar o treino, e os restantes 30% dos dados para validação do processo de treino. Neste caso a própria divisão foi efectuada com recurso a diretórios diferentes.

Após os dados se encontrarem preparados, procede-se à definição e compilação do modelo a treinar. É nesta fase que se aplicam os modelos desenvolvidos mencionados na secção anterior.

A função de treino, *fit*, permite incluir uma lista de funções *callback*. Estas funções permitem realizar diversas operações durante o processo de treino como por exemplo interromper o treino quando o processo de aprendizagem estagna. Neste cenário implementou-se uma *callback* designada de *ModelCheckpoint* que permite, durante as épocas de treino, guardar o modelo ou os pesos que apresentam um desempenho superior ao atingido em épocas anteriores. Neste cenário o desempenho é referente à maximização da *validation accuracy* (*val_accuracy*), tendo-se contemplado a minimização da *validation loss* (*val_loss*).

Ao utilizar as funções de carregamento faladas anteriormente não precisamos de nos preocupar em informar o modelo das diversas classes caso os dados estejam já distribuídos pela estrutura de pastas corretas conforme as classes que necessitamos e, posteriormente apenas precisamos de passar ao modelo os conjuntos carregados anteriormente, neste caso os conjuntos imagesTreino e imagesTest, onde o primeiro corresponde aos dados das imagens e o segundo às imagens de validação.

O número de iterações que o processo de treino executa é definido através do parâmetro *epochs*. Os resultados obtidos por cada iteração podem ser observados definindo o parâmetro *verbose* a 1.

```
checkpoint_path = 'ModeloNormalFinal.h5'
checkpoint_callback = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', save_best_only=True, mode='max', verbose=1)

model = Sequential()
model.add(Conv2D(4, (3,3), activation='relu', input_shape=(img_height, img_width,3)))
model.add(tf.keras.layers.LayerNormalization(axis=-1 , center=True , scale=True))
model.add(Flatten())
model.add(Dense(80, activation='relu'))
model.add(Dropout(0.3))
model.add(tf.keras.layers.LayerNormalization(axis=-1 , center=True , scale=True))
model.add(Dense(80, activation='relu'))
model.add(tf.keras.layers.LayerNormalization(axis=-1 , center=True , scale=True))
model.add(Dense(10, activation='softmax'))

model.summary()

model.compile(
    optimizer='adam',
    loss=tf.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy'])

classes = sorted(os.listdir(training_dir))

imagesTreino = tf.keras.utils.image_dataset_from_directory(
    training_dir,
    image_size=(img_height, img_width),
    batch_size=3)

imagesTest = tf.keras.utils.image_dataset_from_directory(
    validation_dir,
    image_size=(img_height, img_width),
    batch_size=3)

history = model.fit(imagesTreino, epochs=100, validation_data=imagesTest, callbacks=[checkpoint_callback])
```

Figura 43 - Código do processo de treino

Terminado o processo de treino, o modelo que obteve melhor desempenho é carregado e submetido a um processo de validação para obter a *accuracy* e a *loss* correspondente, e a um processo de predição que classifica os dados de teste.

É com as classes obtidas na predição e as classes originais que se obtêm as métricas de desempenho: *accuracy*, *precision*, *recall*, F1_*score* e *confusion matrix*. As métricas calcularam-se recorrendo às funções disponibilizadas pela biblioteca sklearn que tem como valores de entradas as classes reais das amostras, *y_test*, e as classes previstas mencionadas acima, *y_pred*. O código relativo a esta etapa pode ser observado na Figura 44.

```
validation_losses = history.history['val_loss']
validation_accuracies = history.history['val_accuracy']

validation_losses = np.array(validation_losses)
validation_accuracies = np.array(validation_accuracies)

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]

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
y_val_true = []
y_val_pred_classes = []
for x_val_batch, y_val_batch in imagesTest:
    y_val_true.extend(y_val_batch.numpy())  # Directly extend the true class labels
    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')
```

Figura 44 - Código para calcular as métricas de avaliação

Os resultados que se obtiveram do modelo podem ser visualizados pelos blocos de código apresentados nas figuras seguintes (Figura 45, Figura 46 e Figura 47).

```
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%}")
```

Figura 45 - Código para apresentar os resultados (métricas)

```
89  # Calculate the confusion matrix
90  conf_matrix = confusion_matrix(y_val_true, y_val_pred_classes)
91
92  # Plot the confusion matrix as a heatmap with numbers inside cells
93  plt.figure(figsize=(8, 6))
94  plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)
95
96  for i in range(len(conf_matrix)):
97      for j in range(len(conf_matrix[i])):
98          plt.text(j, i, str(conf_matrix[i, j]), ha='center', va='center', color='black')
99
100 plt.title('Confusion Matrix')
101 plt.colorbar()
102 tick_marks = np.arange(len(conf_matrix))
103 plt.xticks(tick_marks, tick_marks)
104 plt.yticks(tick_marks, tick_marks)
105 plt.xlabel('Predicted Label')
106 plt.ylabel('True Label')
107 plt.show()
```

Figura 47 - Código para gerar a matriz de confusão

```
109   # Plot the training and validation accuracy graph
110   plt.figure(figsize=(8, 6))
111   plt.plot(history.history['accuracy'], label='Training Accuracy')
112   plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
113   plt.xlabel('Epoch')
114   plt.ylabel('Accuracy')
115   plt.legend()
116   plt.title('Training and Validation Accuracy')
117   plt.show()
118
119   # Plot the training and validation loss graph
120   plt.figure(figsize=(8, 6))
121   plt.plot(history.history['loss'], label='Training Loss')
122   plt.plot(history.history['val_loss'], label='Validation Loss')
123   plt.xlabel('Epoch')
124   plt.ylabel('Loss')
125   plt.legend()
126   plt.title('Training and Validation Loss')
127   plt.show()
```

Figura 46 -Código para gerar gráficos para a comparação *accuracy*/*loss* de treino com a de validação)

4.3 Comparação de Resultados

4.3.1 Conv2D

Inicialmente, ao desenvolver o modelo Conv2D optamos por investigar que efeitos os tamanhos das imagens carregadas poderiam ter aquando do treino. Para tal o mesmo modelo foi treinado 3 vezes distintas. Primeiramente as dimensões das imagens foram divididas por 2 (320x240), em seguida por 2.5 (256x192) e finalmente por 3 (213x160). Consoante os resultados obtidos foi possível identificar qual a quantidade de *resizing* que iriamos aplicar nos restantes modelos e treinos.

Nas subsecções seguintes serão mostradas as métricas de cada um dos testes assim como as suas matrizes de confusão e gráficos de *accuracy* e *loss*.

4.3.1.1 *Resize* /2 (320x240)

Abaixo é possível observar os resultados do modelo Conv2D com imagens de dimensões 320x240 (Tabela 1), a sua matriz de confusão (Figura 48) e os gráficos que comparam os valores de treino e validação da *accuracy* e da *loss* (Figura 49 e Figura 50, respetivamente).

Tabela 1 - Resultados do modelo Conv2D com *Resize* /2

Resultados				
***Accuracy* Média (%)**	***Loss* Média**	***Precision* (%)**	***Recall* (%)**	**F1 (%)**
96.85%	0.1685	96.30%	96.06%	96.05%

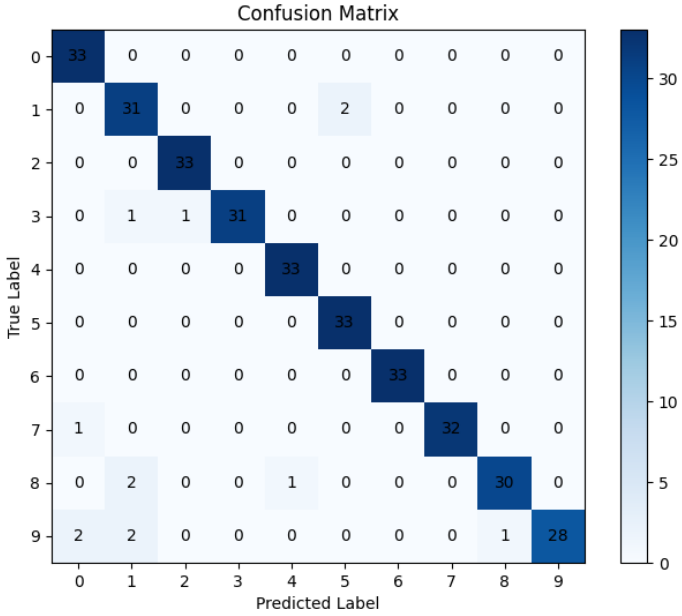


Figura 48 - Matriz de confusão do modelo Conv2D com *resize* de /2

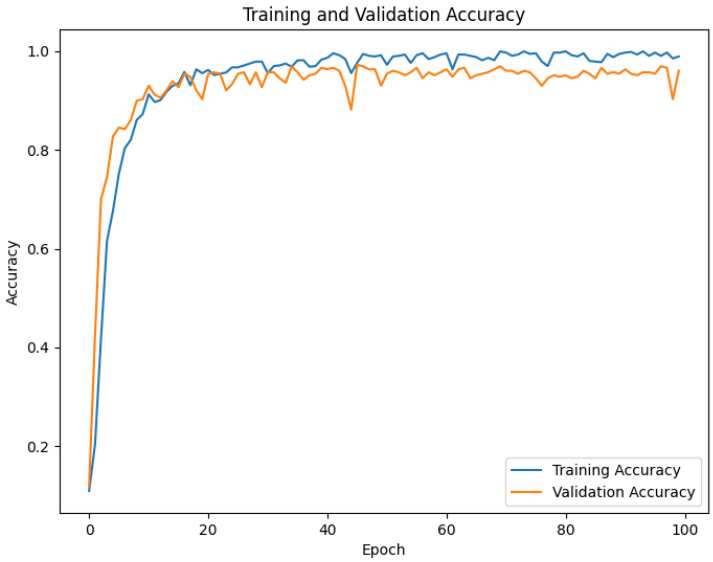


Figura 49 - Comparação da *accuracy* de treino e validação para o modelo Conv2D com *resize* de /2

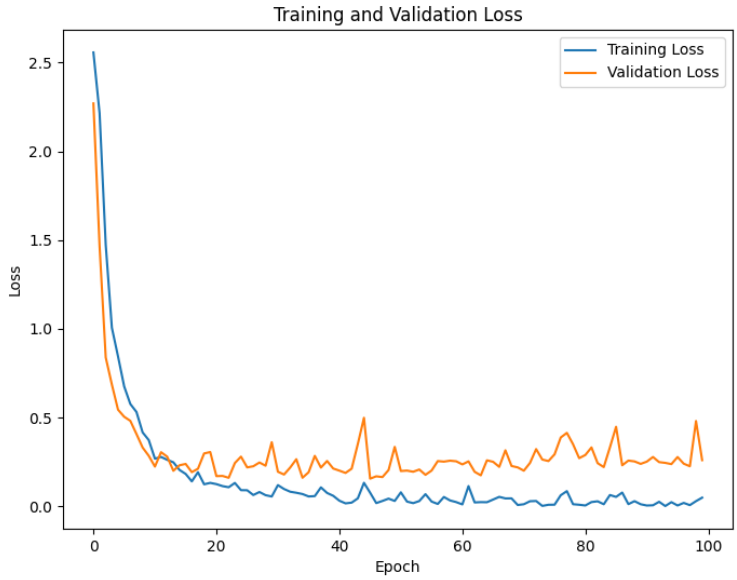


Figura 50 - Comparação da *loss* de treino e validação para o modelo Conv2D com *resize* de /2

Este modelo apresenta um valor de *loss* um pouco acima do desejável enquanto as restantes métricas encontram-se a rondar os 96% A matriz de confusão deste modelo apresenta a maioria das amostras na sua diagonal principal. Os gráficos apresentados para este modelo apresentam vários pontos de interseção entre os valores de treino e de validação, entre a *epoch* 15 e 20, para a *accuracy* verificando-se o mesmo no caso da *loss*. O tempo de execução de cada *epoch* rondou os 45 segundos.

4.3.1.2 *Resize* /2.5 (256x192)

Abaixo é possível observar os resultados do modelo Conv2D com imagens de dimensões 256x192 (Tabela 2), a sua matriz de confusão (Figura 51) e os gráficos que comparam os valores de treino e validação da *accuracy* e da *loss* (Figura 52 e Figura 53, respetivamente).

Tabela 2 - Resultados do modelo Conv2D com Resize /2.5

Resultados				
***Accuracy* Média (%)**	***Loss* Média**	***Precision* (%)**	***Recall* (%)**	**F1 (%)**
96.36%	0.2012	95.41%	95.41%	94.86%

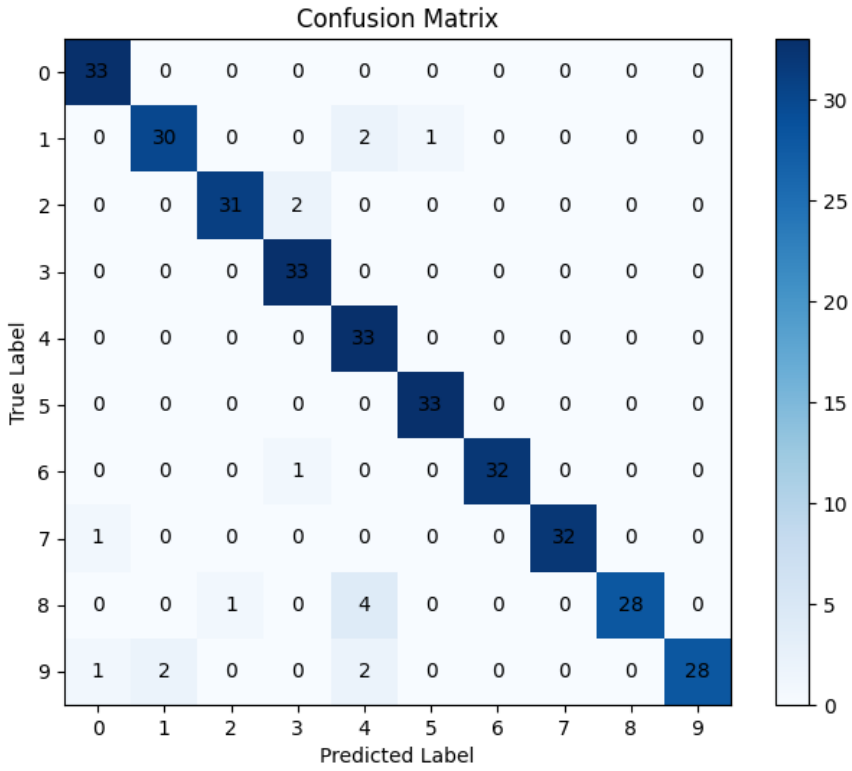


Figura 51 - Matriz de confusão do modelo Conv2D com *resize* de /2.5

Figura 52 - Comparação da *accuracy* de treino e validação para o modelo Conv2D com *resize* de /2.5

Figura 53 - Comparação da *loss* de treino e validação para o modelo Conv2D com *resize* de /2.5

Este modelo apresenta um acréscimo na métrica de *loss* e as restantes métricas com valores ligeiramente abaixo do modelo anterior. Quanto à matriz de confusão, é possível verificar que os valores se encontram, na sua maioria, na diagonal principal. Os gráficos apresentados para este modelo apresentam vários pontos de interseção entre os valores de treino e de validação, entre as *epochs* 15 e 20. O tempo de execução de cada *epoch* foi cerca de 30 segundos.

4.3.1.3 *Resize* /3 (213x160)

Abaixo é possível observar os resultados do modelo Conv2D com imagens de dimensões 213x160 (Tabela 3), a sua matriz de confusão (Figura 54) e os gráficos que comparam os valores de treino e validação da *accuracy* e da *loss* (Figura 55 e Figura 56, respetivamente).

Tabela 3 - Resultados do modelo Conv2D com Resize /3

Resultados				
***Accuracy* Média (%)**	***Loss* Média**	***Precision* (%)**	***Recall* (%)**	**F1 (%)**
96.82%	0.1568	97.03%	96.67%	96.68%

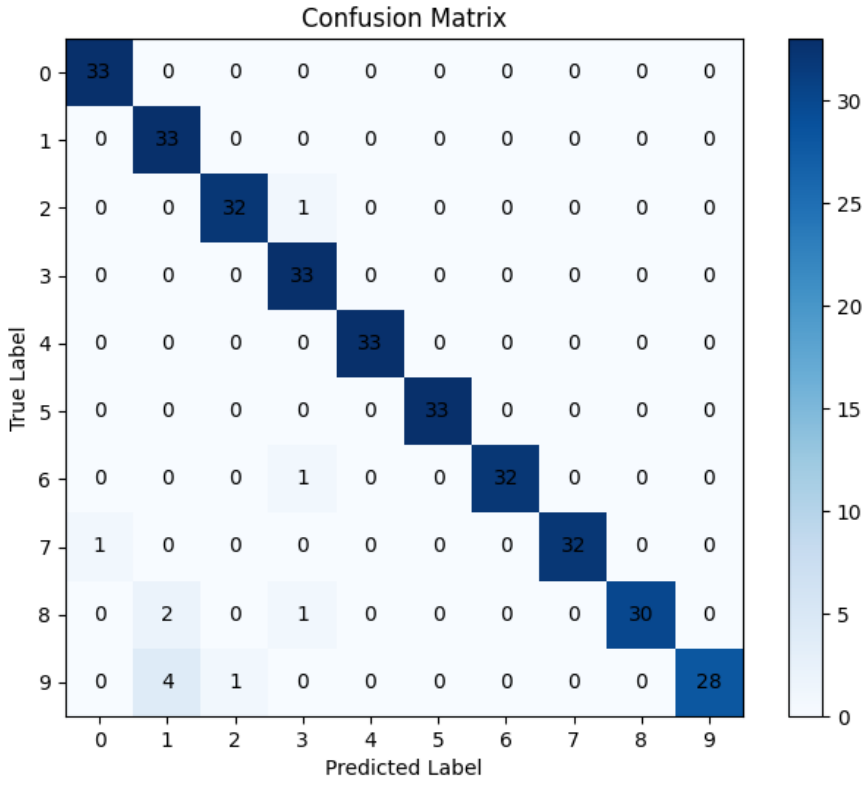


Figura 54 - Matriz de confusão do modelo Conv2D com *resize* de /3

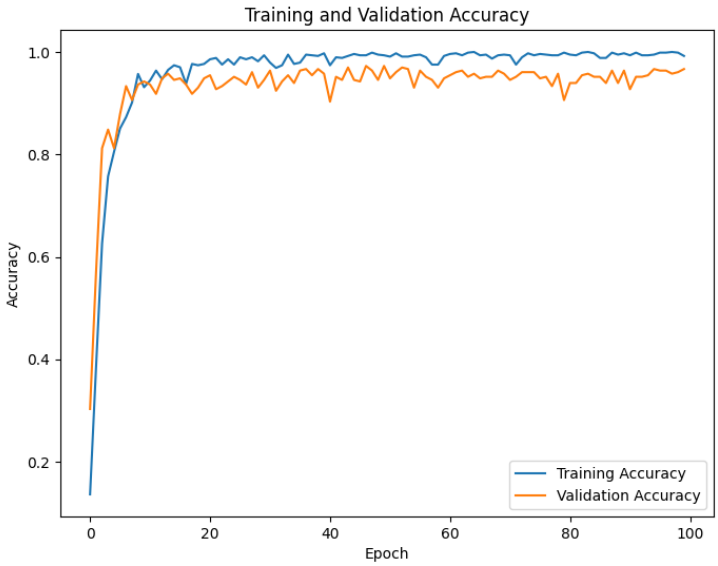


Figura 55 - Comparação da *accuracy* de treino e validação para o modelo Conv2D com *resize* de /3

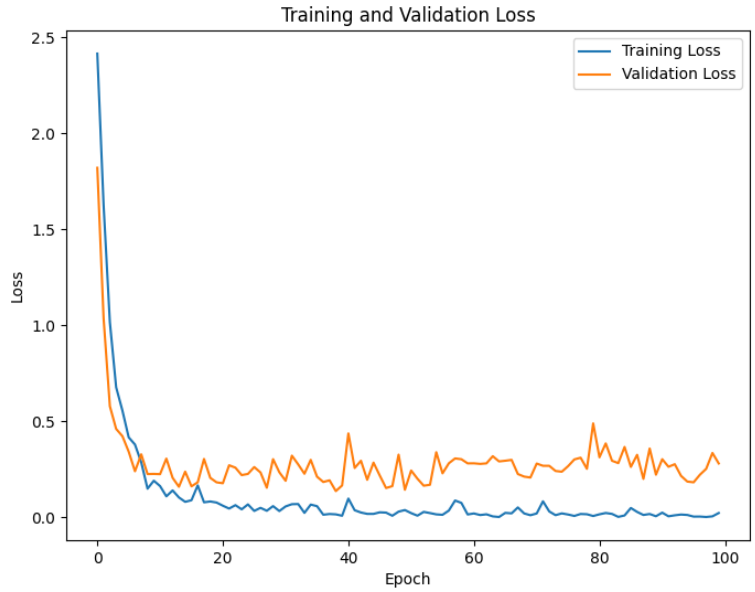


Figura 56 - Comparação da *loss* de treino e validação para o modelo Conv2D com *resize* de /3

Este modelo apesar de possuir uma *accuracy* média ligeiramente menor do que o primeiro, apresenta também uma *loss* menor e os melhores valores nas restantes métricas. Quanto à matriz de confusão, é possível verificar que os valores também se encontram, na sua maioria, na diagonal principal. Os gráficos apresentados para este modelo apresentam vários pontos de interseção entre os valores de treino e de validação, entre as *epochs* 15 e 20. O tempo de execução de cada *epoch* foi cerca de 15 segundos. Dado as métricas de *Precision*, *Recall* e F1 serem as melhores dos 3 testes corridos e o tempo de execução ser significativamente menor, optamos por usar um *resizing* de /3 nos restantes modelos.

4.3.2 Conv2D com *Augmentation*

Visto o total de testes com todas as combinações possíveis entre os 4 parâmetros de *augmentation* ser numeroso, foi optado por apenas incluir as matrizes de confusão e os gráficos de *Accuracy* de treino/teste e *Loss* de treino/teste apenas dos modelos que tiveram resultados mais impactantes, quer pela positiva, quer pela negativa. Os resultados de todos os testes relativos a esta secção podem ser encontrados na Tabela 1 e as respetivas matrizes de confusão e gráficos irão ser incluídos como anexos deste trabalho.

4.3.2.1 Conv2D com Augmentation (Tabela 1, linha 3)

Abaixo é possível observar os resultados do modelo Conv2D com imagens de dimensões 213x160 e *Augmentation* usando o parâmetro *samplewise_center*. A sua matriz de confusão (Figura 57) e os gráficos que comparam os valores de treino e validação da *accuracy* e da *loss* (Figura 58 e Figura 59, respetivamente).

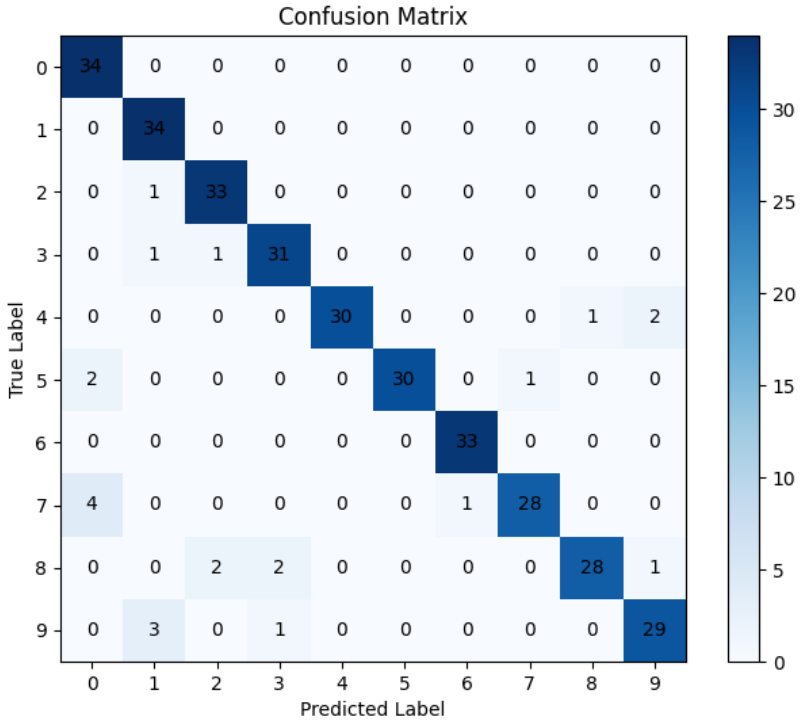


Figura 57 - Matriz de confusão do modelo Conv2D com *Augmentation* (*samplewise_center*)

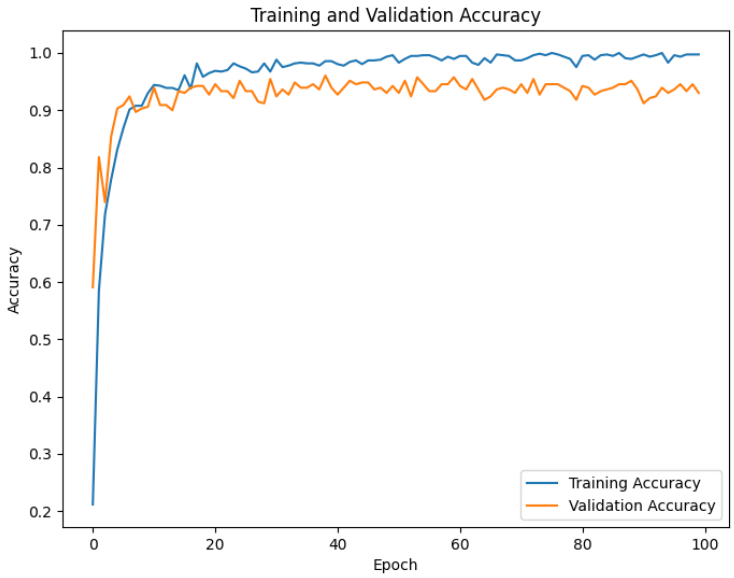


Figura 58 - Comparação da *accuracy* de treino e validação para o modelo Conv2D com *Augmentation* (*samplewise_center*)

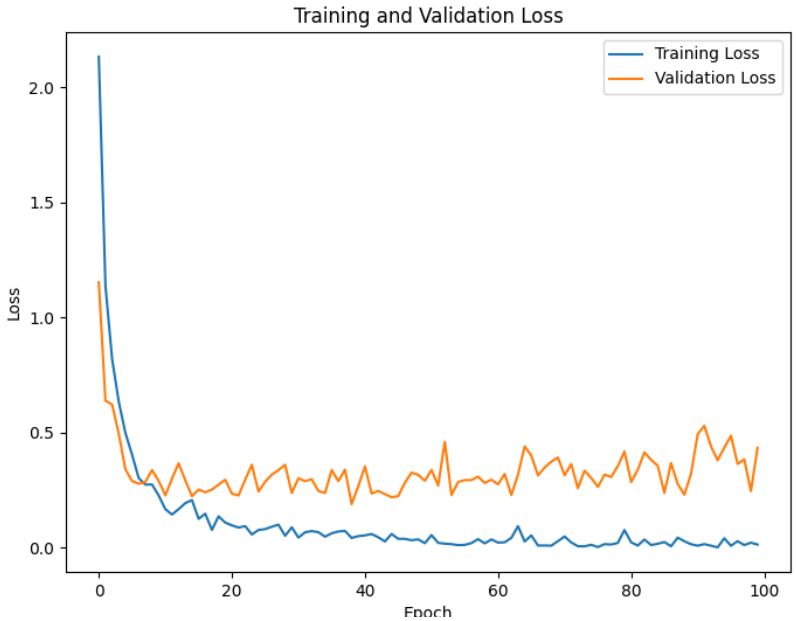


Figura 59 - Comparação da *loss* de treino e validação para o modelo Conv2D com *Augmentation* (*samplewise_center*)

Este modelo apresenta o valor da *loss* acima do desejado apesar das restantes métricas terem obtido valores aceitáveis, acima dos 90%. Ainda assim, observou-se que usando apenas o parâmetro *samplewise_center* as mesmas diminuíram o que não era de todo o resultado esperado, tendo este modelo apresentado os piores resultados em todas as métricas

analisadas. Relativamente à matriz de confusão, é imperativo que os valores mais altos se encontrem na sua diagonal principal, algo que é possível verificar na matriz deste modelo. Podemos observar que os valores de treino vão melhorando até a *epoch* 60, havendo muito poucas melhorias nas *epochs* seguintes.

4.3.2.2 Conv2D com *Augmentation* (Tabela 1, linha 12)

Abaixo é possível observar os resultados do modelo Conv2D com imagens de dimensões 213x160 e *Augmentation* usando os parâmetros *height_shift_range*, *samplewise_center*, *samplewise_std_normalization*. A sua matriz de confusão (Figura 60) e os gráficos que comparam os valores de treino e validação da *accuracy* e da *loss* (Figura 61 e Figura 62, respetivamente).

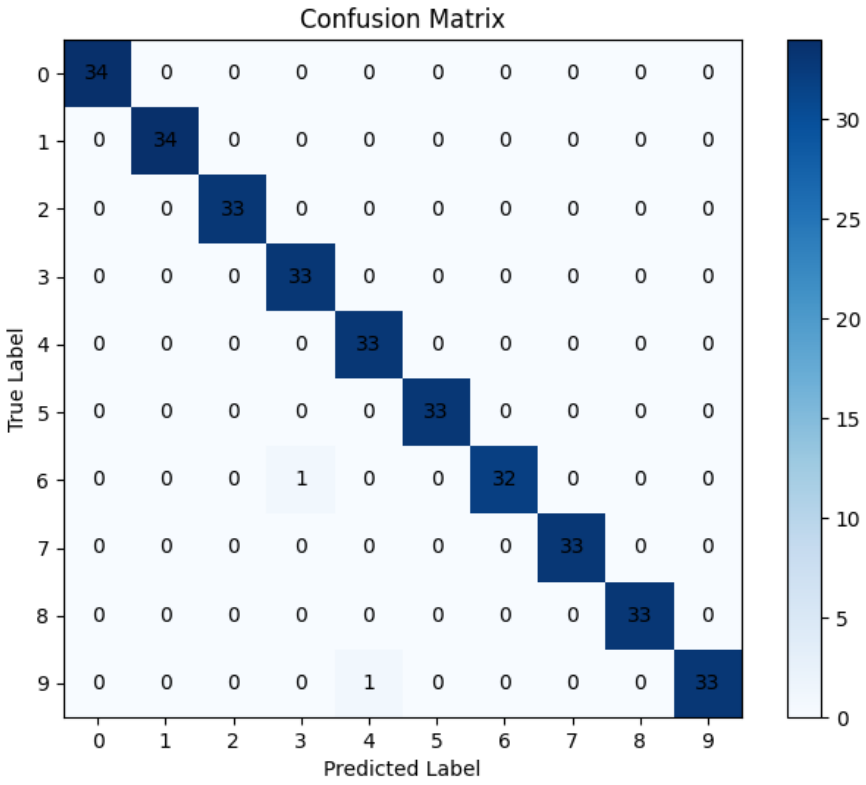


Figura 60 - Matriz de confusão do modelo Conv2D com *Augmentation* (*height_shift_range, samplewise_center, samplewise_std_normalization*)

Figura 62 - Comparação da *accuracy* de treino e validação para o modelo Conv2D com *Augmentation* (*height_shift_range, samplewise_center, samplewise_std_normalization*)

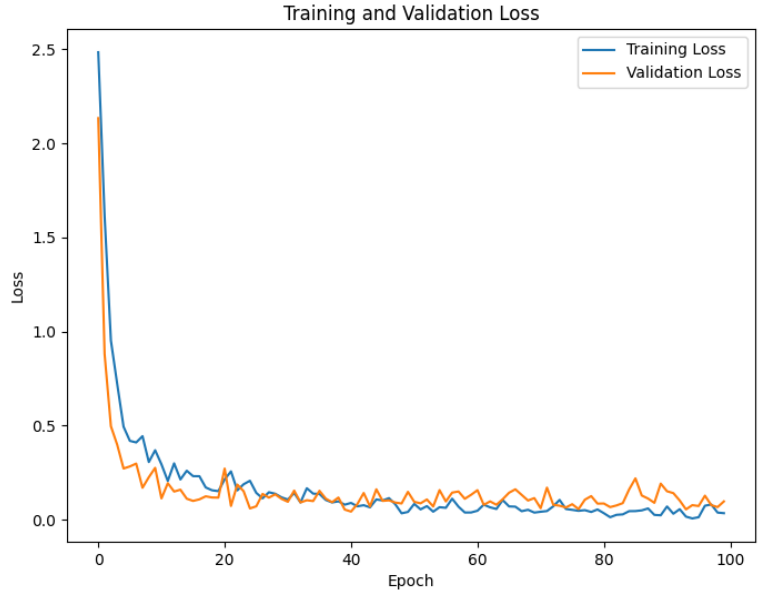


Figura 61 - Comparação da *loss* de treino e validação para o modelo Conv2D com *Augmentation* (*height_shift_range, samplewise_center, samplewise_std_normalization*)

Este modelo apresenta valores médios de *accuracy* e *loss* muito bons e que, apesar de não serem os melhores entre todos os testes efetuados observou-se que usando os parâmetros *height_shift_range*, *samplewise_center*, *samplewise_std_normalization* o modelo apresentou os melhores resultados nas restantes métricas como *Precision*, *Recall* e F1 atingindo a marca dos 99%. Relativamente à matriz de confusão os valores mantêm-se

predominantes na diagonal principal. Podemos observar que os valores de treino vão melhorando até a *epoch* 60, havendo muito poucas melhorias nas *epochs* seguintes.

4.3.2.3 Conv2D com *Augmentation* (Tabela 1, linha 16)

Abaixo é possível observar os resultados do modelo Conv2D com imagens de dimensões 213x160 e *Augmentation* usando os parâmetros h*eight_shift_range*, *samplewise_center*, *samplewise_std_normalization* e *rotation_range*. A sua matriz de confusão (Figura 63) e os gráficos que comparam os valores de treino e validação da *accuracy* e da *loss* (Figura 64 e Figura 65, respetivamente).

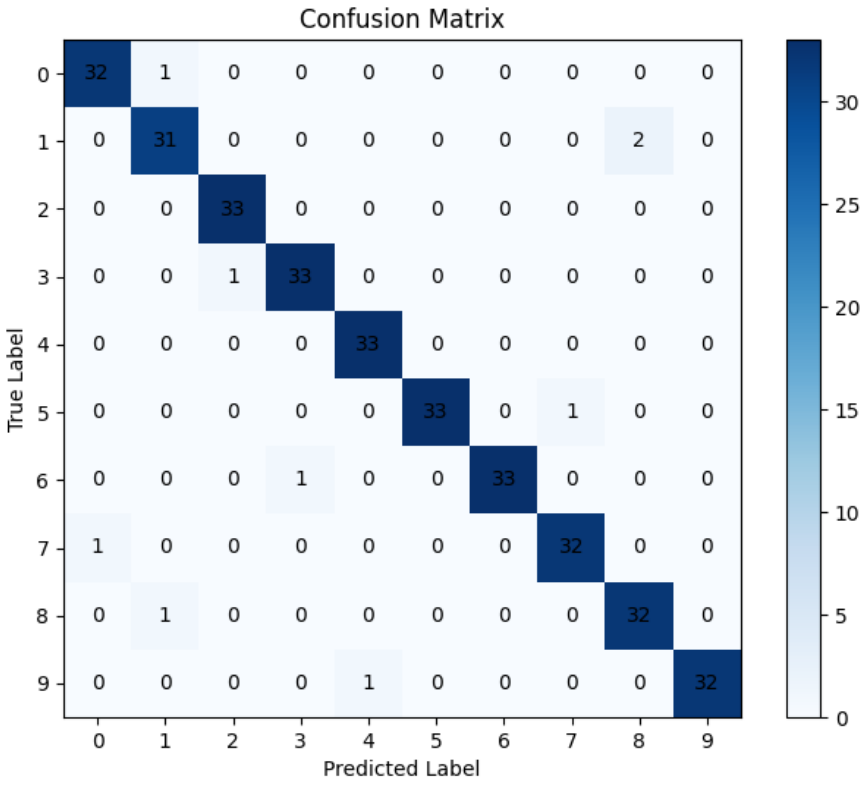


Figura 63 - Matriz de confusão do modelo Conv2D com *Augmentation* (todos os 4 parâmetros)

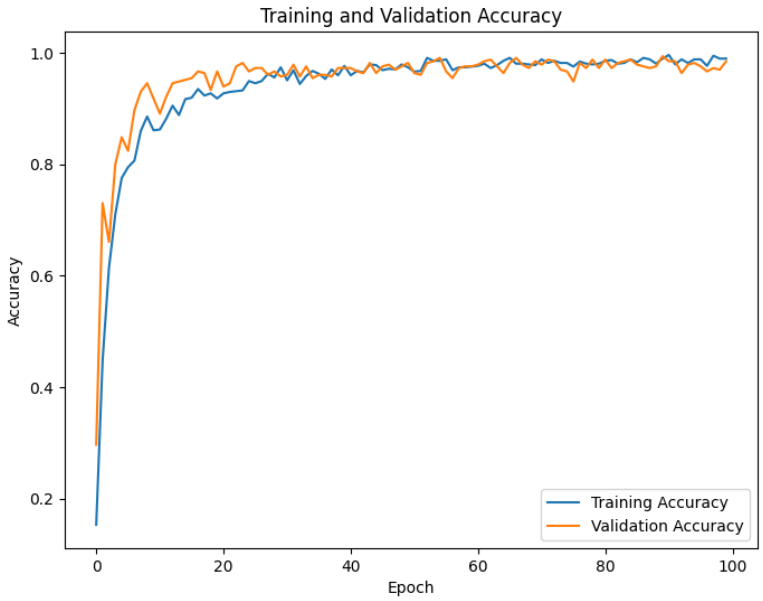


Figura 64 - Comparação da *accuracy* de treino e validação para o modelo Conv2D com *Augmentation* (todos os 4 parâmetros)

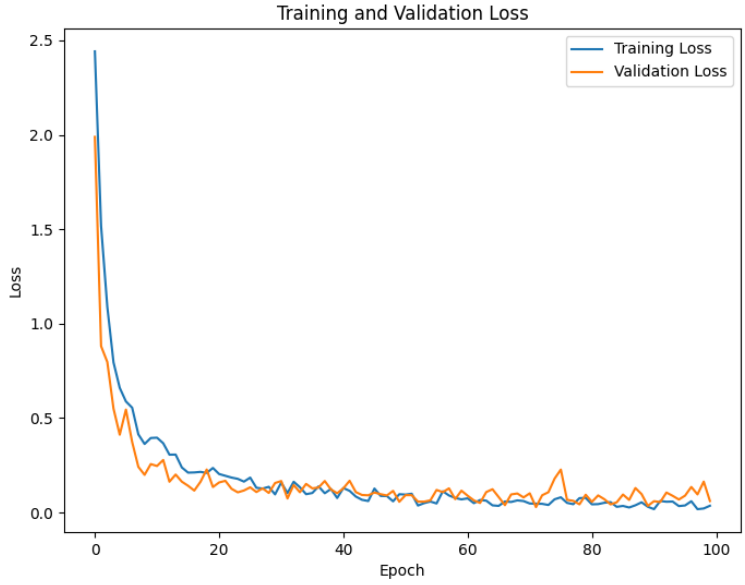


Figura 65 - Comparação da *loss* de treino e validação para o modelo Conv2D com *Augmentation* (todos os 4 parâmetros)

Por fim este modelo que utiliza os 4 parâmetros em simultâneo, apresenta os melhores valores médios de *accuracy* e *loss*, ficando ligeiramente atrás dos melhores valores ao nível das outras métricas. Relativamente à matriz de confusão os valores mantêm-se predominantes na diagonal principal. Os gráficos são semelhantes aos modelos anteriores e podemos observar que os valores de treino vão melhorando até a *epoch* 60, havendo muito poucas melhorias nas *epochs* seguintes. Visto os resultados detalhados anteriormente,

optamos por usar os parâmetros do modelo da linha nº13 da Tabela abaixo como parâmetros a incluir nos modelos de EfficientNet e ResNet50.

Tabela 4 - Resultados obtidos com *Augmentation* usando diferentes parâmetros

		Parâmetros				Resultados				
Nº linha	**Nº Anexo**	***height_shift_range***	***samplewise_center***	***samplewise_std_normalization***	***rotation_range***	***accuracy* media**	***loss* media**	***precision***	***recall***	**F1**
1	**Anexo 7**	----	----	----	----	97.48%	0.1134	95.61%	95.20%	95.18%
2	**Anexo 8**	**X**	----	----	----	98.42%	0.0786	97.44%	97.29%	97.29%
3	**Anexo 9**	----	**X**	----	----	**95.45%**	**0.2238**	**93.58%**	**93.04%**	**93.07%**
4	**Anexo 10**	----	----	**X**	----	97.09%	0.1566	95.77%	95.53%	95.52%
5	**Anexo 11**	----	----	----	**X**	96.82%	0.1226	94.62%	94.32%	94.28%
6	**Anexo 12**	**X**	**X**	----	----	98.15%	0.0579	96.41%	95.76%	95.79%
7	**Anexo 13**	----	**X**	**X**	----	96.82%	0.1747	95.49%	95.20%	95.16%
8	**Anexo 14**	----	----	**X**	**X**	96.30%	0.1833	96.06%	95.77%	95.72%
9	**Anexo 15**	**X**	----	**X**	----	97.70%	0.0941	97.73%	97.58%	97.57%
10	**Anexo 16**	**X**	----	----	**X**	98.52%	0.0621	98.25%	98.21%	98.20%
11	**Anexo 17**	----	**X**	----	**X**	96.85%	0.1540	96.27%	95.81%	95.85%
12	**Anexo 18**	**X**	**X**	**X**	----	98.67%	0.0599	**99.41%**	**99.40%**	**99.40%**
13	**Anexo 19**	----	**X**	**X**	**X**	95.88%	0.2012	96.36%	96.08%	96.09%
14	**Anexo 20**	**X**	----	**X**	**X**	98.79%	0.0562	98.53%	98.49%	98.50%
15	**Anexo 21**	**X**	**X**	----	**X**	98.58%	0.0554	98.53%	98.48%	98.49%
16	**Anexo 22**	**X**	**X**	**X**	**X**	**98.85%**	**0.0462**	97.32%	97.30%	97.30%

4.3.3 EfficientNet

Abaixo é possível observar os resultados do modelo EfficientNet com imagens de dimensões 213x160 e *Augmentation* com 3 parâmetros (*height_shift_range*, *samplewise_center*, *samplewise_std_normalization*) (Tabela 5), a sua matriz de confusão (Figura 66) e os gráficos que comparam os valores de treino e validação da *accuracy* e da *loss* (Figura 67 e Figura 68, respetivamente). Como é possível observar, este modelo foi o que ficou em segundo lugar a nível de todas as métricas.

Tabela 5 - Resultados do modelo EfficientNet com *Augmentation*

Resultados				
***Accuracy* Média (%)**	***Loss* Média**	***Precision* (%)**	***Recall* (%)**	**F1 (%)**
99.88%	0.0026	99.71%	99.70%	99.70%

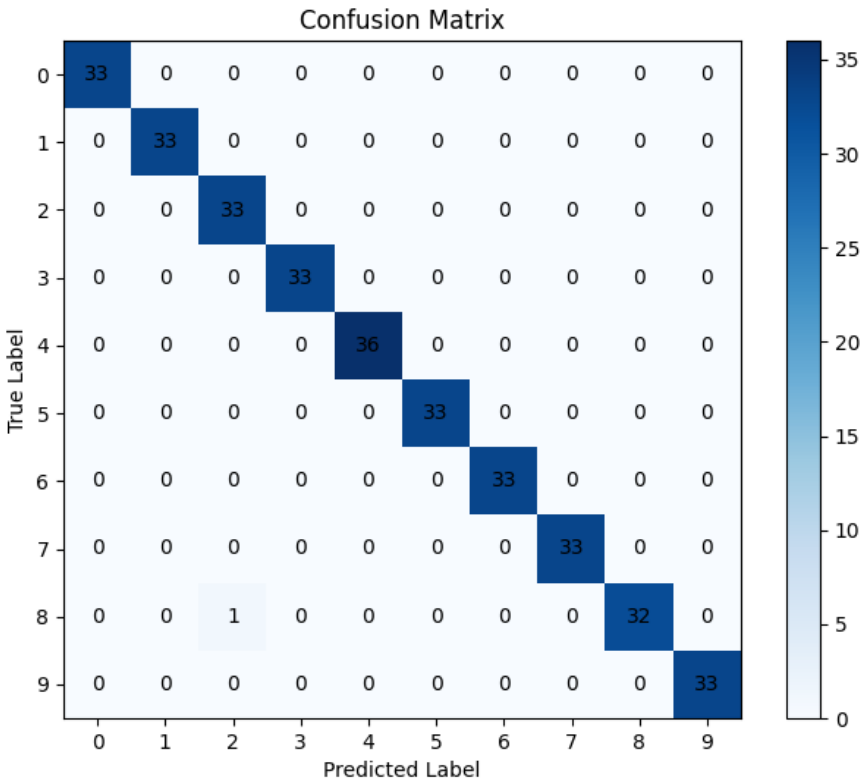


Figura 66 - Matriz de confusão do modelo EfficientNet com *Augmentation*

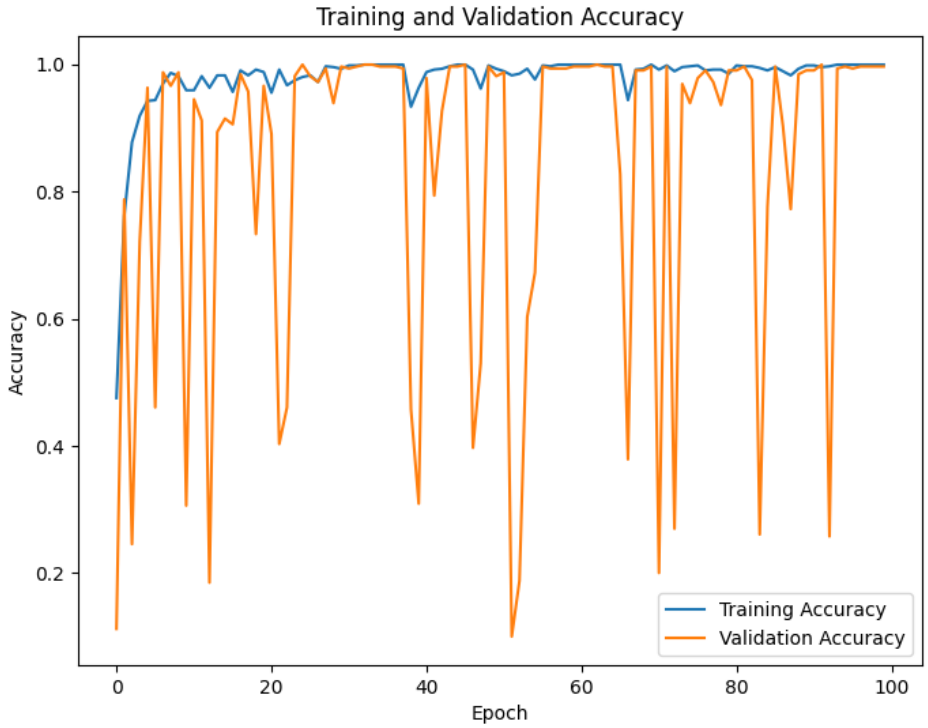


Figura 67 - Comparação da *accuracy* de treino e validação para o modelo EfficientNet com *Augmentation*

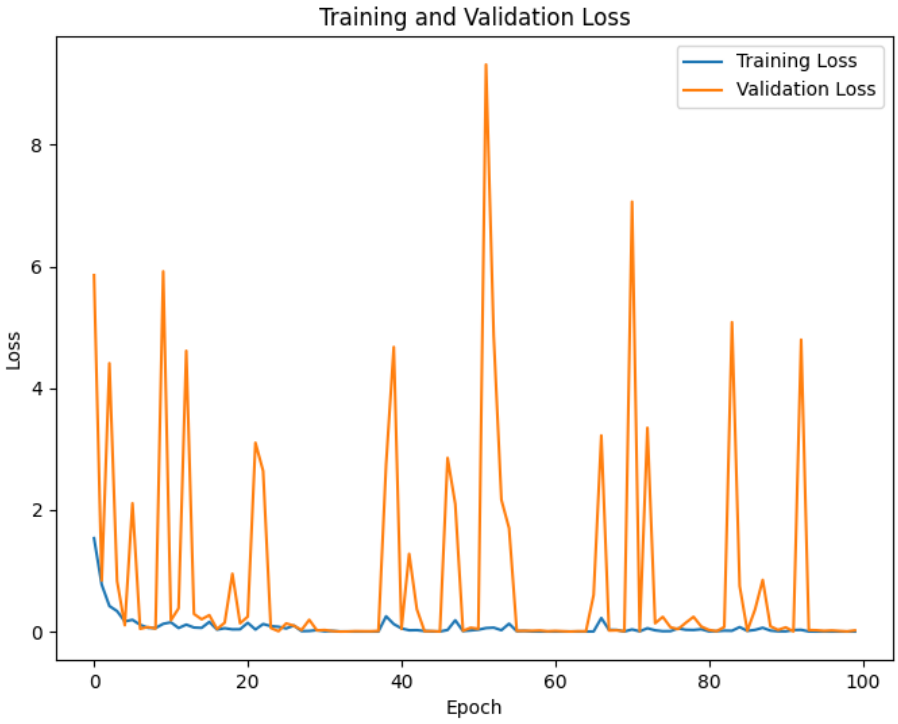


Figura 68 - Comparação da *loss* de treino e validação para o modelo EfficientNet com *Augmentation*

4.3.4 ResNet50

Abaixo é possível observar os resultados do modelo ResNet50 com imagens de dimensões 213x160 e *Augmentation* com 3 parâmetros (*height_shift_range, samplewise_center, samplewise_std_normalization*) (Tabela 6), a sua matriz de confusão (Figura 69) e os gráficos que comparam os valores de treino e validação da *accuracy* e da *loss* (Figura 70 e Figura 71, respetivamente). Como é possível observar, este modelo obteve os melhores resultados possíveis entre todos os outros modelos, tendo uma *loss* virtualmente igual a zero e 100% em todas as outras métricas.

Tabela 6 - Resultados do modelo ResNet50 com *Augmentation*

Resultados				
***Accuracy* Média (%)**	***Loss* Média**	***Precision* (%)**	***Recall* (%)**	**F1 (%)**
100.00%	0.0008	100.00%	100.00%	100.00%

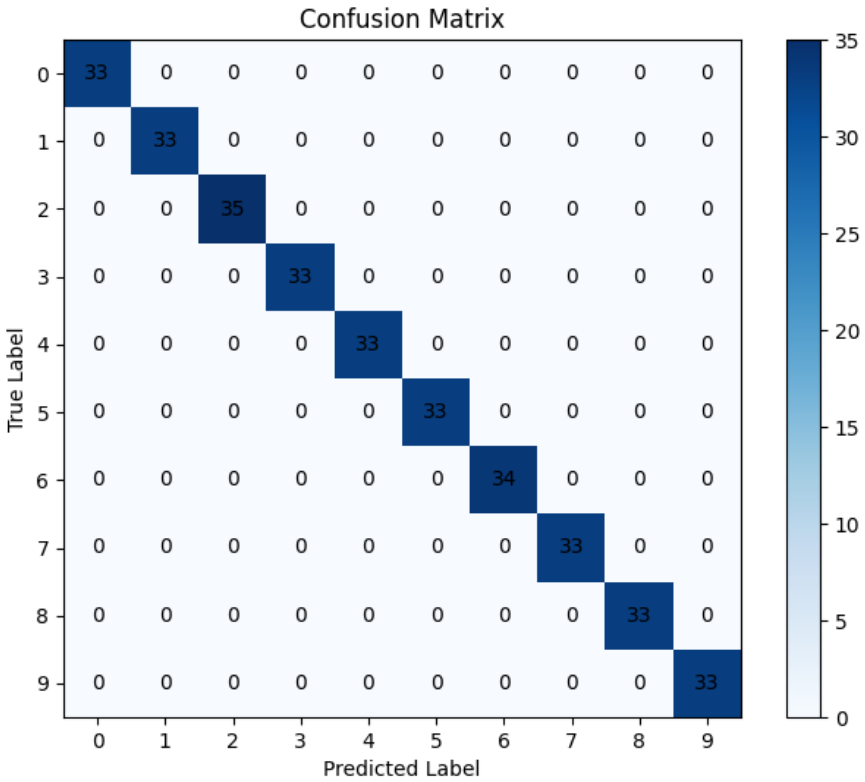


Figura 69 - Matriz de confusão do modelo ResNet50 com *Augmentation*

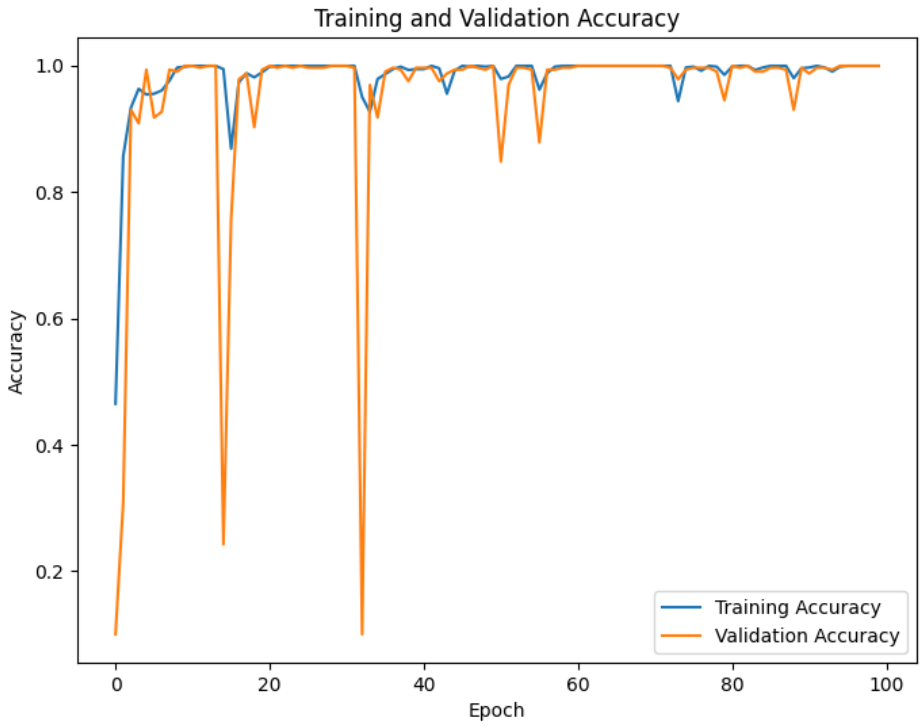


Figura 70 - Comparação da *accuracy* de treino e validação para o modelo ResNet50 com *Augmentation*

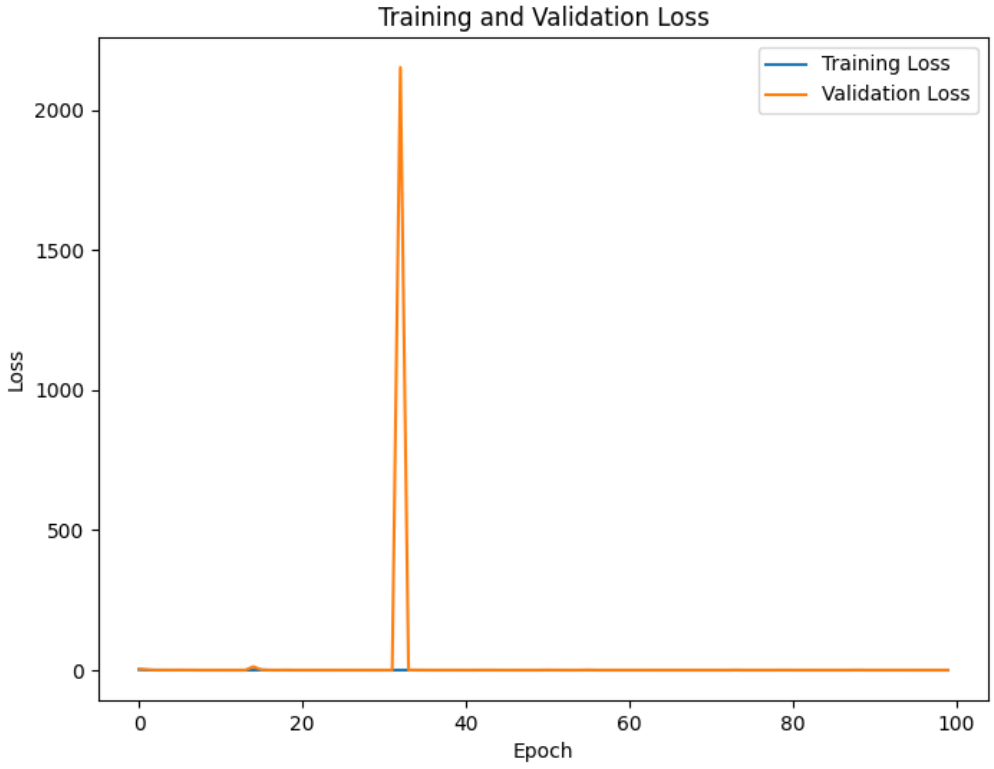


Figura 71 - Comparação da *loss* de treino e validação para o modelo ResNet50 com *Augmentation*

5 Discussão de Resultados

Este capítulo tem como objetivo apresentar os resultados indicados em artigos relevantes, além dos artigos apresentados na secção 2.4, que também tiveram bastante relevância e importância no desenvolvimento deste projeto, e realizar uma comparação com os resultados apresentados no capítulo anterior. Para tal iremos abordar 2 artigos que artigos foram selecionados por realizarem a classificação de áudio com base em espectrogramas. Um onde os autores utilizam diversos áudios divididos em diversas categorias e os transformam em espectrogramas de *Mel* para posterior treino dos modelos e outro onde é utilizado um *Dataset* de comandos de voz em inglês, também posteriormente transformados em espectrogramas para efeitos de treino de diversos modelos.

O primeiro artigo, desenvolvido por Boyang Zhang, Jared Leitner e Sam Thornton, apresenta a seguinte tabela de resultados. As métricas utilizadas nesta tabela são o *False Positive Rate* (FPR), o F1 *Score* e a *Accuracy*.

É necessário ter em conta que os áudios utilizados pelos autores são provenientes do *dataset Freesound Dataset* (FSD), que contem 297,144 áudios. Entre esses, posteriormente foram selecionados e extraídos 4,970 como parte de um *subset* usado para uma competição por parte do popular website de *machine learning* e *data science* denominado Kaggle. Esse mesmo *subset* contem áudios de 80 categorias diferentes. Após a conversão dessas amostras para Espectrogramas de *Mel*, foram utilizados 3 modelos diferentes onde 80% das amostras serviram para treino e os restantes 20% serviram para teste. As métricas utilizadas para avaliação foram o *label-weighted label ranking average precision* (LWLRAP) que consiste na média ponderada das precisões médias e as 5 classes com melhor *accuracy*. Os resultados podem ser observados na tabela abaixo (Tabela 7).

Tabela 7 - Classificação dos resultados dos modelos

Models	*Epochs*	LWLRAP	*Top 5 Categorical Accuracy*
Deep CNN	400	0.813	88.9%
Transfer Learning	100	0.797	88.5%
VGG19 (*no weight*)	400	0.737	82.9%

Como é possível observar, o modelo desenvolvido pelos autores (Deep CNN) provou possuir um LWLRAP mais elevado assim como uma melhor *accuracy* do top 5 das diversas classes, reforçando assim o bom desempenho que as CNN tem vindo a demonstrar na classificação de áudio e reforçando também que o uso de espectrogramas de *Mel* é uma abordagem totalmente viável e que merece maior atenção e investigação.

Já no segundo artigo, escrito por Aljenan Soliman, Salah Mohamed e Iman Abuelmaaly Abdelrahman, os autores propõem-se a apresentar um estudo sobre o uso de CNNs para o reconhecimento de palavras isoladas, o que vai mais de encontro com o tema central e o objetivo desta tese.

Durante o trabalho desenvolvido pelos autores, os mesmo utilizaram um *dataset*, também proveniente do Kaggle, chamado *Speech Command Dataset* v0.01. O mesmo é constituído por diversos áudios com a duração de 1 segundo, cada um deles contendo uma palavra em inglês. Dentro dos 64,727 áudios que constituem o *dataset*, os mesmos encontram-se divididos em 30 palavras diferentes sendo que o foco dos autores é apenas reconhecer 10 dessas palavras sendo as restantes 20 classificadas como “desconhecida”.

Os resultados são apresentados em duas tabelas distintas. A primeira (Tabela 8), onde é mostrada a *Accuracy* e *Loss* de Teste de cada modelo. E a segunda (Tabela 9) onde é apresentada a *Precision*, *Recall*, F1 *Support* e *Accuracy* para cada uma das *labels* do modelo com melhor *Accuracy*. Essa segunda tabela foi necessária para uma melhor avaliação de desempenho do modelo, visto o caso de estudo apresentado ser um problema de classificação de múltiplas *labels* e existir um desequilíbrio entre as classes (mais amostras existentes para determinadas classes), o que não se verificou no caso do nosso projeto. Através dessa tabela conseguimos ter uma visão mais detalhada do desempenho do modelo.

Tabela 8 - Valores de Accuracy e *Loss* de Teste dos modelos

Model	***Test accuracy***	***Test loss***
MFCC-8000 Hz	93.20%	0.2531
MFCC-16000Hz	96.19%	0.1840
Spectrogram-8000 Hz	93.21%	0.2622
Spectrogram-16000 Hz	94.86%	0.2368
1-D convnet	91.03%	0.3602

Tabela 9 - Relatório de classificação do modelo MFCC

Label	***Precision***	***Recall***	**F-1 *score***	***Support***	***Accuracy***
Yes	0.99	0.97	0.98	474	96.20%
No	0.97	0.97	0.97	445	96.17%
Up	0.88	0.98	0.93	478	96.44%
Down	0.98	0.94	0.96	499	96.19%
Left	0.97	0.97	0.97	443	96.16%
Right	0.97	0.97	0.97	440	96.81%
On	0.97	0.97	0.97	496	96.57%
Off	0.96	0.94	0.95	462	90.25%
Stop	0.98	0.97	0.97	514	97.27%
Go	0.94	0.94	0.94	445	90.11%
Unknown	0.94	0.93	0.93	831	92.29%
Silence	1.00	1.00	1.00	785	100%

Através do artigo descrito acima e da análise das tabelas e resultados, podemos aferir que, apesar de nossos testes com o modelo ResNet50 termos obtido resultados excelentes de 100%, esse valores provavelmente iriam diminuir. Nesse caso seria interessante implementar elementos do estudo efetuado no artigo, testando o treino do modelo com *Mel-frequency cepstral coefficients* (MFCCs), que são outra forma de representar áudio, em vez de espetrogramas de Mel, podendo assim potenciar uma melhoria de resultados conforme é possível observar nos resultados.

6 Aplicação

6.1 Introdução à aplicação

Com o intuito de utilizar os modelos anteriormente treinados criou-se uma simples aplicação em Python, sem interface gráfica que funciona em parte como a aplicação que gera os espectrogramas.

A mesma fica a escuta de sinais sonoros e quando esses mesmos sinais ficam abaixo de um *threshold* definido a mesma considera esse estado como "silencio". Assim que se encontra nesse estado por um determinado período, a captação de áudio para o espectrograma é gerado e fornecido ao modelo para ser classificado. De acordo com a classificação obtida, é depois executada uma das ações treinadas como mover o rato para cima, baixo, direita e esquerda, efetuar o *click* ou duplo *click*, aumentar ou diminuir a velocidade de movimento ou, por fim, parar o movimento do rato. Para tal efetuou se o treino do modelo com espectrogramas das palavras "cima", "baixo", "direita", "esquerda", "enter", "duplo", "mais", "menos" e "stop".

De forma a simplificar a sua utilização e integração não só com a aplicação médica, mas também com outras aplicações, optamos por desenvolver a aplicação em separado, permitindo à mesma estar a ser executada em plano de fundo e controlar o rato em todas as outras aplicações que estejam a correr.

6.2 Desenvolvimento

Para elaborar esta aplicação, recorremos às bibliotecas de Python, pyautogui e multiprocessing, as quais iremos abordar mais detalhadamente no decorrer deste capítulo.

Como falado anteriormente, começamos por reutilizar o código da aplicação responsável por gerar os espectrogramas, de forma a que tal como na outra aplicação, fosse possível escutar sinais sonoros, gerar um espectrograma, gravá-lo e posteriormente classifica lo com base no modelo treinado e carregado para que, com base nessa classificação seja executada uma determinada ação (Figura 72 e 73).

```
if pred_labels[0] == 0:
    #print(is_moving.value)
    print('UP')
    if not bool(is_moving.value):
        is_moving.value = True
        movement_direction.value = 0
    else:
        is_moving.value = False
        movement_direction.value = 0
        is_moving.value = True

elif pred_labels[0] == 1:
    print('DOWN')
    if not bool(is_moving.value):
        is_moving.value = True
        movement_direction.value = 1
    else:
        is_moving.value = False
        movement_direction.value = 1
        is_moving.value = True

elif pred_labels[0] == 2:
    print('RIGHT')
    if not bool(is_moving.value):
        is_moving.value = True
        movement_direction.value = 2
    else:
        is_moving.value = False
        movement_direction.value = 2
        is_moving.value = True

elif pred_labels[0] == 3:
    print('LEFT')
    if not bool(is_moving.value):
        is_moving.value = True
        movement_direction.value = 3
    else:
        is_moving.value = False
        movement_direction.value = 3
        is_moving.value = True
```

Figura 73 - Código de controlo do rato (parte 1)

```
elif pred_labels[0] == 4:
    print('STOP')
    is_moving.value = False

elif pred_labels[0] == 5:
    print('CLICK')
    is_moving.value = False
    pyautogui.click()

elif pred_labels[0] == 6:
    print('DECREASE SPEED')
    if speed.value >= 4:
        speed.value = speed.value - 3

elif pred_labels[0] == 7:
    print('INCREASE SPEED')
    if speed.value <= 7:
        speed.value = speed.value + 3
else:
    print('DOUBLE CLICK')
    is_moving.value = False
    pyautogui.click(clicks=2, interval=0.25)
```

Figura 72 - Código de controlo do rato (parte 2)

Consoante a *label* atribuída ao espectrograma gerado é então atribuída uma direção de movimento ao atributo "*value*" da variável *movement_direction* ou então um valor de velocidade ao atributo "*value*" da variável *speed*.

No próximo passo de execução é onde a parte do multiprocessamento entra em vigor visto que no caso desta aplicação vamos ter duas funções distintas a correr em *loop*. Uma a captar áudio continuamente e outra a executar o movimento do rato.

A integração das duas funções em simultâneo foi possível usando *Threading* inicialmente, mas o seu desempenho ficou à quem das expectativas tornando a aplicação praticamente inútil visto o tempo entre a classificação do espectrograma e a execução da respetiva ação possuir um *delay* considerável.

Como solução, recorremos ao uso das capacidades do Multiprocessamento tendo obtido melhorias significativas no desempenho da aplicação ainda que a sua implementação possa ser mais complexa visto ser necessário criar variáveis de acesso partilhado que vão permitir o acesso por parte das duas funções e que vão sendo atualizadas durante a execução do *script* modificando o atributo *value* das diferentes variáveis.

```
if __name__ == '__main__':
    is_moving = Value('i', False)
    movement_direction = Value('i', 0)
    speed = Value('i', 10)
```

Figura 74 – variáveis do tipo *Value* para acesso partilhado

Essas mesmas variáveis são depois utilizadas pela função *move_mouse* para efetuar o movimento do rato e definir a velocidade de movimento.

```
def move_mouse(self, is_moving, movement_direction, speed):
    if bool(is_moving.value):
        if movement_direction.value == 0:
            pyautogui.move(0, -speed.value)

        elif movement_direction.value == 1:
            pyautogui.move(0, speed.value)

        elif movement_direction.value == 2:
            pyautogui.move(speed.value, 0)

        elif movement_direction.value == 3:
            pyautogui.move(-speed.value, 0)
```

Figura 75 – código para movimentar o rato conforme valores recebidos

Por fim, apenas foi necessário criar os respetivos processos para cada função, passar-lhes as variáveis necessárias e inicializá-los conforme a Figura 76.

```
def audio_thread(is_moving, movement_direction, speed):
    a.listen(is_moving, movement_direction, speed)

def mouse_thread(is_moving, movement_direction, speed):
    while True:
        mouse_controller.move_mouse(is_moving, movement_direction, speed)

if __name__ == '__main__':
    is_moving = Value('i', False)
    movement_direction = Value('i', 0)
    speed = Value('i', 10)

    audio_process = multiprocessing.Process(target=audio_thread, args=(is_moving,
                                                                       movement_direction,
                                                                       speed))
    mouse_process = multiprocessing.Process(target=mouse_thread, args=(is_moving,
                                                                       movement_direction,
                                                                       speed))

    audio_process.start()
    mouse_process.start()

    audio_process.join()
    mouse_process.join()
```

Figura 76 – Inicialização dos processos responsáveis pelo tratamento do áudio e controlo do rato

6.3 Funcionamento

A aplicação é de fácil utilização, no entanto, é necessário um sistema com o Python instalado (recomenda-se o uso da versão 3.9.10) e as respetivas bibliotecas.

Para iniciar a aplicação é necessário, iniciar a linha de comandos na pasta onde se encontra o ficheiro controlMouseMultiprocessing.py e executar o comando “python controlMouseMultiprocessing.py”, como é possível observar na Figura 77.

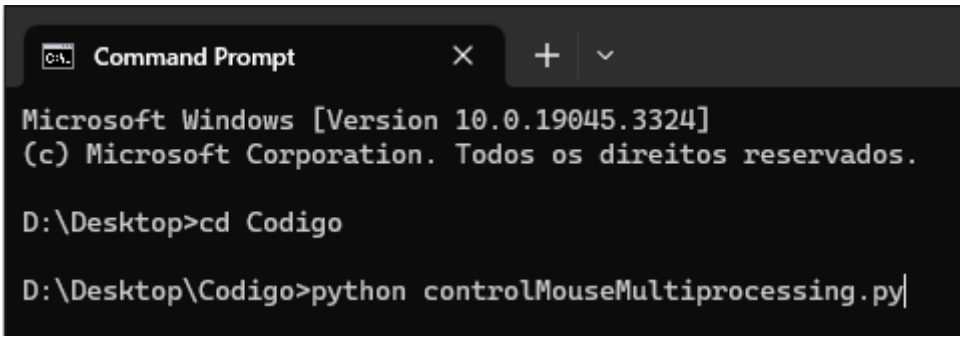


Figura 77 – comando para iniciar a aplicação

Após a aplicação iniciar todas as suas funcionalidades são mostradas na janela todas as etapas de execução do programa como quando é iniciada a captação do áudio, quando é detetado áudio e é iniciada a sua gravação, o tamanho da imagem após redimensionamento, o tempo decorrido para classificação da imagem, o resultado da classificação, a palavra a que corresponde, a velocidade atual do movimento do ponteiro do rato e por fim quando voltou à captação do áudio (Figura 78).

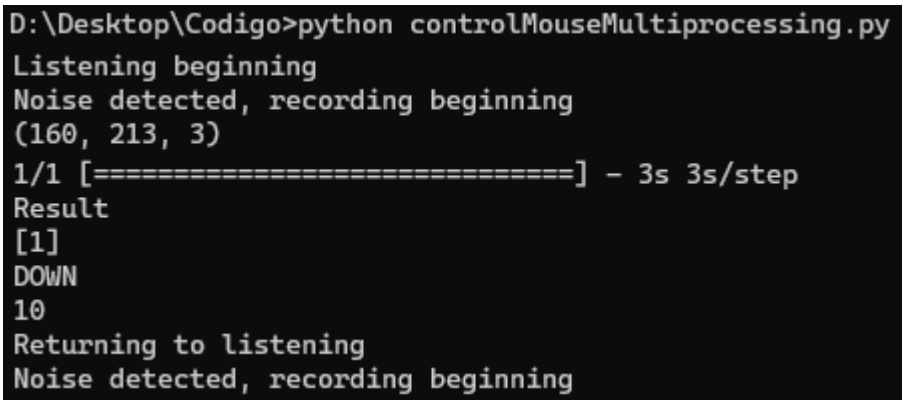


Figura 78 – Informação mostrada durante a execução

7 Trabalhos Futuros

Na fase final da elaboração de um projeto, é normal que surjam ideias que permitam melhorar o que foi desenvolvido ou que permitam dar continuidade ao projeto. Este capítulo tem como objetivo apresentar e explorar algumas dessas ideias. Algumas destas ideias poderão entrar em conflito caso sejam implementas em simultâneo, pelo que devem ser aprofundadas antes da sua aplicação.

Uma das ideias mais relevantes, refere-se à utilização de um *dataset* diferente, mais completo e robusto. Apesar de o *dataset* utilizado incluir 7 oradores diferentes, essa amostra não é suficiente para representar bem a variedade de frequência vocais, entoações e sotaques existentes na língua portuguesa.

Para resolver esta questão sugere-se adicionar ao *dataset* mais amostras, de vários oradores no geral, mas principalmente de oradoras femininas, visto o nosso *dataset* apenas incluir uma.

Essa alteração irá com certeza impactar as diferentes métricas podendo existir a necessidade de investigar novas formas de melhorar os resultados obtidos como por exemplo o uso de MFCC's ao invés de espetrogramas de *Mel* conforme visto no capítulo 5.

Alguns dos modelos desenvolvidos à exceção dos pré treinados apresentam valores de *validation loss* maiores que o desejado e alguns sinais de *overfitting*. Estes modelos deveriam ser melhorados de forma a tentar reduzir estes valores e, idealmente, aumentar as restantes métricas em simultâneo.

A aplicação desenvolvida contém as funcionalidades essenciais, contudo podem ser incorporadas novas funcionalidades de acordo com novas necessidades que surjam por parte de utilizadores durante o seu uso. Para além disso, existe sempre a possibilidade de melhorar o seu tempo de execução e resposta, tornando o seu tempo de resposta mais *real-time*.

8 Conclusão

A elaboração deste projeto, cujo objetivo principal consistiu em desenvolver um modelo de *Deep Learning* para classificação de palavras, permitiu observar a natureza iterativa deste tipo de problemas.

No contexto deste projeto, esta iteratividade centrou-se nas componentes de preparação de dados e desenvolvimento de modelos. Frequentemente, durante o processo de desenvolvimento de modelos é possível observar que os resultados obtidos podem estar a ser influenciados negativamente devido a uma preparação de dados desenquadrada do problema, ou simplesmente um erro presente nessa preparação.

A relevância dos recursos disponíveis demonstrou-se ser um fator predominante na realização bem-sucedida deste tipo de projetos. A qualidade e o volume de dados de um *dataset* são a fundação para o desenvolvimento de bons modelos. Estes devem ser analisados para determinar se as classes que contêm se encontram equilibradas, caso não seja possível equilibrar o *dataset* os passos seguintes devem ter este facto em consideração.

O hardware onde se pretende realizar este tipo de projetos é de igual forma importante. Um computador com boas especificações técnicas permite realizar os processos de treino mais rapidamente, o que permite adicionar mais iterações no processo se necessário de forma a atingir melhores resultados.

Os modelos desenvolvidos neste projeto apresentam, para as métricas de avaliação selecionadas, resultados muito animadores compreendidos entre 95% e 100% o que mostra a total viabilidade do uso de espectrogramas de *Mel* como forma de reconhecimento de palavras e treino de modelos.

Além disso, apesar de alguns valores da *validation loss* serem superiores a 0.2, que em geral, é um resultado mau [31], houve também modelos que conseguiram atingir valores inferiores a esse, tendo um deles se aproximado do valor zero com 3 casas decimais.

Na fase final de elaboração deste projeto, não foi possível realizar uma avaliação da aplicação desenvolvida recorrendo a amostras de outro *dataset*. Esta avaliação seria relevante para se obter uma perspetiva independente do desempenho dos modelos.

Referências

[1] IBM, “What is machine learning?” [Online]. Available: https://www.ibm.com/topics/machine-learning. [Acedido em 4 Agosto 2023].

[2] S. Bhatt, “Reinforcement Learning 101,” 19 Março 2018. [Online]. Available: https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292. [Acedido em 5 Agosto 2023].

[3] F. L. La, “Inteligência Artificial - Análise da Cesta de Compras,” Dezembro 2018. [Online]. Available: https://learn.microsoft.com/pt-br/archive/msdnmagazine/2018/december/artificially-intelligent-market-basket-analysis. [Acedido em 4 Agosto 2023].

[4] AI Wiki, “Weights and Biases,” 7 Março 2021. [Online]. Available: https://machine-learning.paperspace.com/wiki/weights-and-biases. [Acedido em 4 Agosto 2023].

[5] Patterson, Cameron. (2012). “Managing a real-time massively-parallel neural architecture.” [Online]. Available: https://www.researchgate.net/publication/262493920_Managing_a_real-time_massively-parallel_neural_architecture [Acedido em 1 Setembro 2023]

[6] Krenker, A., Bešter, J., & Kos, A. (2011). “Introduction to the artificial neural networks. Artificial Neural Networks: Methodological Advances and Biomedical Applications.” [Online]. Available: https://www.researchgate.net/profile/Kenji-Suzuki-2/publication/319316102_Artificial_Neural_Networks_-_Methodological_Advances_and_Biomedical_Applications/links/59a42f16aca272a6461bb35e/Artificial-Neural-Networks-Methodological-Advances-and-Biomedical-Applications.pdf. [Acedido em 1 Setembro 2023]

[7] Raschka, Sebastian. “Machine Learning FAQ”. [Online]. Available: https://sebastianraschka.com/faq/docs/relu-derivative.html [Acedido em 1 Setembro 2023]

[8] Hvidberrrg@GitHub “The sigmoid function (a.k.a. the logistic function) and its derivative” [Online]. Available:

https://hvidberrrg.github.io/deep_learning/activation_functions/sigmoid_function_and_derivative.html [Acedido em 1 Setembro 2023]

[9] S. SHARMA, "Activation Functions in Neural Networks," 6 Setembro 2017.

[Online]. Available: https://towardsdatascience.com/activation-functions-neuralnetworks-1cbd9f8d91d6. [Acedido em 13 Agosto 2023].

[10] Kampakis, Dr. S. (Stelios). (2023, Agosto 28). "What Deep Learning is and isn't". [Online]. Available: https://thedatascientist.com/what-deep-learning-is-and-isnt/. [Acedido em 15 Agosto 2023].

[11] J. Brownlee, "Difference Between Backpropagation and Stochastic Gradient Descent," 1 Fevereiro 2021. [Online]. Available: https://machinelearningmastery.com/difference-between-backpropagation-andstochastic-gradient-descent/. [Acedido em 13 Agosto 2023].

[12] M. Nielsen, "Why are deep neural networks hard to train?," Dezembro 2019. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap5.html. [Acedido em 12 Setembro 2023].

[13] Mesuga, Reymond & Bayanay, Brian. (2021). "A Deep Transfer Learning Approach to Identifying Glitch Wave-form in Gravitational Wave Data." [Online]. Available: https://www.researchgate.net/publication/353049916_A_Deep_Transfer_Learning_Approach_to_Identifying_Glitch_Wave-form_in_Gravitational_Wave_Data. [Acedido em 12 Setembro 2023].

[14] Dertat, A. (2018, Junho 19). "Applied Deep Learning - Part 4: Convolutional Neural Networks." [Online]. Available: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2. [Acedido em 24 Agosto 2023].

[15] M. Montero, "Resampling Methods for Machine Learning modeling," 7 Abril 2021. [Online]. Available: https://medium.com/geekculture/resampling-methodsfor-machine-learning-modeling-d2cdc1d3640f. [Acedido em 24 Agosto 2023].

[16] S. Chauhan, “Model Selection for Machine Learning,” 1 Junho 2023. [Online]. Available: https://www.scholarhat.com/tutorial/machinelearning/modelselection-for-machine-learning. [Acedido em 29 Julho 2023].

[17] Turing, “Different Types of Cross-Validations in Machine Learning and Their Explanations,” [Online]. Available: https://www.turing.com/kb/different-typesof-cross-validations-in-machine-learning-and-their-explanations. [Acedido em 29 Julho 2023].

[18] A. F. Gad, “Evaluating Deep Learning Models: The Confusion Matrix, Accuracy, Precision, and Recall,” 2020. [Online]. Available: https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/. [Acedido em 29 Julho 2023].

[19] M. Sokolova e G. Lapalme, “Information Processing and Management,” A systematic analysis of performance measures for classification tasks, 2009.

[20] A. Géron, Hands-On Machine Learning with Scikit Learn and TensorFlow, United States of America: O’Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2019.

[21] M. Banoula, “What is Tensorflow? Deep Learning Libraries & Program Elements,” 16 Fevereiro 2023. [Online]. Available: https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-tensorflow. [Acedido em 3 Julho 2023].

[22] TensorFlow, “TensorFlow,” 29 Junho 2023. [Online]. Available: https://www.tensorflow.org/install/pip. [Acedido em 3 Julho 2023].

[23] Deeplizard, “TensorFlow and Keras GPU Support - CUDA GPU Setup,” 21 Maio 2020. [Online]. Available: https://www.youtube.com/watch?v=IubEtS2JAiY&t=367s. [Acedido em 3 Julho 2023].

[24] Keras-Team. “TypeError: Unable to serialize [2.0896919 2.1128857 .1081853] to JSON.” [Online]. Available: https://github.com/keras-team/keras/issues/18218. [Acedido em 12 Setembro 2023].

[25] Scikit-learn, “scikit-learn: machine learning in Python — scikit-learn 1.3.0,” [Online]. Available: https://scikit-learn.org/stable/. [Acedido em 12 Setembro 2023].

[26] J. Bronwlee, “Gentle Introduction to the Adam Optimization Algorithm for Deep Learning,” 3 Julho 2017. [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deeplearning/. [Acedido em 16 Julho 2023].

[27] K. E. Koech, “Cross-Entropy Loss Function,” 2 Outubro 2020. [Online]. Available: https://towardsdatascience.com/cross-entropy-loss-functionf38c4ec8643e. [Acedido em 16 Julho 2023].

[28] N. Lang, “Breaking down Convolutional Neural Networks: Understanding the Magic behind Image Recognition,” 4 Dezembro 2021. [Online]. Available: https://towardsdatascience.com/using-convolutional-neural-network-for-imageclassification-5997bfd0ede4. [Acedido em 20 Julho 2023].

[29] Baheti, Bhakti & Innani, Shubham & Gajre, Suhas & Talbar, Sanjay. (2020).”Eff-UNet: A Novel Architecture for Semantic Segmentation in Unstructured Environment.” [Online]. Available: https://www.researchgate.net/publication/343271189_Eff-UNet_A_Novel_Architecture_for_Semantic_Segmentation_in_Unstructured_Environment. [Acedido em 9 Agosto 2023]

[30] S. Mukherjee, “The Annotated ResNet-50,” 18 Agosto 2022. [Online]. Available: https://towardsdatascience.com/the-annotated-resnet-50- a6c536034758. [Acedido em 9 Agosto 2023].

[31] B. Wang, “Loss Functions in Machine Learning,” 13 Janeiro 2021. [Online]. Available: https://medium.com/swlh/cross-entropy-loss-in-pytorchc010faf97bab. [Acedido em 3 Setembro 2023].

[32] Juang, B. & Rabiner, Lawrence. (2005). Automatic Speech Recognition - A Brief History of the Technology Development.

[33] KAMATH, U.; LIU, J.; WHITAKER, J. Automatic speech recognition. In: DEEP Learning for NLP and Speech Recognition. Gewerbestrasse 11, 6330 Cham, Switzerland: Springer Nature Switzerland, 2019. p. 370–377. ISBN 978-3-030-14595- 8. Available: https://doi.org/10.1007/978-3-030-14596-5.

[34] KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, v. 25, p. 1097–1105, 2012.

[35] Hinton GE, Osindero S, Teh YW. A fast learning algorithm for deep belief nets. Neural Comput. 2006 Jul;18(7):1527-54. doi: 10.1162/neco.2006.18.7.1527. PMID: 16764513.

[36] Shewalkar, Apeksha & Nyavanandi, Deepika & Ludwig, Simone. (2019). Performance Evaluation of Deep neural networks Applied to Speech Recognition: Rnn, LSTM and GRU. Journal of Artificial Intelligence and Soft Computing Research. 9. 235-245. 10.2478/jaiscr-2019-0006.

[37] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In Proceedings of the 23rd international conference on Machine learning (ICML '06). Association for Computing Machinery, New York, NY, USA, 369–376. https://doi.org/10.1145/1143844.1143891

[38] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang and Y. Bengio, "Batch normalized recurrent neural networks," 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2016, pp. 2657-2661, doi: 10.1109/ICASSP.2016.7472159.

[39] Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.

[40] GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. Deep learning. MIT press, 2016.

[41] GERÓN, A. Hands-On Machine Learning with Scikit-Learn, Keras and Tensorflow: concepts, tools and techniques to build intelligent systems. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly, 2019

[42] COOK, A.; SHEAHEN, D.; GEARHART, C. AIND-VUI-Capstone. [S. l.]: GitHub, 2018. Disponível em: https://github.com/udacity/AIND-VUI-Capstone.

[43] Sarkar, A. (2021, 8 de Maio). Understanding EfficientNet — The most powerful CNN architecture. Medium. https://medium.com/mlearning-ai/understanding-efficientnet-themost-powerful-cnn-architecture-eaeb40386fad

[44] Tan, M., & Le, Q. v. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. https://doi.org/10.48550/arXiv.1905.11946

[45] Speech Emotion Recognition using Time Distributed CNN and LSTM Beenaa Salian, Omkar Narvade, Rujuta Tambewagh and Smita Bharne, ITM Web Conf., 40 (2021) 03006, DOI: https://doi.org/10.1051/itmconf/20214003006

[46] Qidong Lu, Yingying Li, Zhiliang Qin, Xiaowei Liu, and Yun Xie. 2020. Speech Recognition using EfficientNet. In Proceedings of the 2020 5th International Conference on Multimedia Systems and Signal Processing (ICMSSP '20). Association for Computing Machinery, New York, NY, USA, 64–68. https://doi.org/10.1145/3404716.3404717

[47] IBM, "O que é reconhecimento de voz?" [Online]. Available: https://www.ibm.com/br-pt/cloud/learn/speech-recognition#toc-o-que--rec-SMJbcsvz. [Acedido em 4 Agosto 2023].

[48] Ramesaliyev. (2022). "MNIST Digit Recognizer with CNN (Tutorial)". *Kaggle*. [Online]. Available: https://www.kaggle.com/code/ramesaliyev/mnist-digit-recognizer-with-cnn-tutorial/notebook. [Acedido em 13 Setembro 2023].

[49] Chen, L., & Li, S. (2018). "Improvement Research and Application of Text Recognition Algorithm Based on CRNN". International Conference on Signal Processing and Machine Learning.

[50] Gao, Miao & Shi, Guoyou & Li, Shuang. (2018). Online Prediction of Ship Behavior with Automatic Identification System Sensor Data Using Bidirectional Long Short-Term Memory Recurrent Neural Network. Sensors. 18. 4211. 10.3390/s18124211.

[51] Abdulwahab, Saddam & Jabreel, Mohammed & Moreno, Dr. (2017). Deep Learning Models for Paraphrases Identification. 10.13140/RG.2.2.15743.46240.

[52] Dey, Polash & Hossain, Emam & Hossain, Md & Chowdhury, Mohammed & Alam, Md & Hossain, Mohammad & Andersson, Karl. (2021). Comparative Analysis of Recurrent Neural Networks in Stock Price Prediction for Different Frequency Domains. Algorithms. 14. 251. 10.3390/a14080251.

Anexo 1 – Código GravaEspectrograma.py

```
from os.path import dirname, join as pjoin
from matplotlib import pyplot as plt
import tensorflow_io as tfio
from scipy.io import wavfile
from sys import byteorder
from scipy import signal
import librosa
import librosa.display
import tensorflow as tf
from array import array
import numpy as np
import scipy.io
import pyaudio
import os
import sys
from numpy import savetxt
from keras.models import load_model
import math
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas

np.set_printoptions(threshold=sys.maxsize)
sampleDir = r'D:\Desktop\samples'
datasetDir = r'D:\Desktop'
trainingDir = r'D:\Desktop\\Training'
validationDir = r'D:\Desktop\\Validation'
trainingSamples = 7
validationSamples = 3
print('Inserir numero a gravar')
palavra = input()

class AudioRecorder:

  def __init__(self):
    self.rate = 44100
    self.threshold = 0.014 # silence threshold
    self.chunk_size = 1024
    self.format = pyaudio.paFloat32
    self._pyaudio = pyaudio.PyAudio()

  def isSilent(self, data):
    #Returns true if below the silence threshold
    return max(data) < self.threshold
```

```
    def trim(self, data):
      # Trim the blanks at the start and end
     def _trim(data):
       started = False
       r = array('f')
       for i in data:
         if not started and abs(i) > self.threshold: #self.threshold
           started = True
           r.append(i)

         elif started:
           r.append(i)

       return r
     #first trim the left side
     data = _trim(data)
     data.reverse()

     # #then trim the right side
     data = _trim(data)
     data.reverse()

     return data

   def addSilence(self, data):
     #adds silence to the start and end of 0.1 seconds
     r = array('f', [0 for i in range(int(0.1 * self.rate))])
     r.extend(data)
     r.extend([0 for i in range(int(0.1 * self.rate))])

     return r

   def record(self):
     stream = self._pyaudio.open(format=self.format, channels=1, rate=self.rate, input=True,
output=True, frames_per_buffer=self.chunk_size)
     numSilent = 0
     started = False
     r = array('f')

     while 1:
       #has to be little endian and signed short
       data = array('f', stream.read(self.chunk_size))

       if byteorder == 'big':
         data.byteswap()

       r.extend(data)
       silent = self.isSilent(data) #check silent to add blanks
```

```
      if silent and started:
        numSilent += 1

      elif not silent and not started:
        started = True
        print('Noise detected, recording beginning')
      if started and numSilent > 20: #if there are 30 silences, break. doesnt handle long
sentences.
        break

    width = self._pyaudio.get_sample_size(self.format)
    stream.stop_stream()
    stream.close()

    sampleDir_n_files = len(os.listdir(sampleDir))
    r = self.trim(r)
    r = self.addSilence(r)

################################################################

    r = np.array(r, dtype=np.float32)
    scipy.io.wavfile.write(os.path.join(sampleDir,'{}_{}.wav'.format(palavra,
sampleDir_n_files)),44100,r)
    print ('Written to file')

    fig, ax = plt.subplots()

    melspec = librosa.feature.melspectrogram(y=r, sr=self.rate,
                                             n_mels=256,
                                             fmax=22000,
                                             hop_length=8)

    norm_melspec = librosa.core.power_to_db(melspec, ref=np.max)

    img = librosa.display.specshow(norm_melspec,
                                   x_axis='time',
                                   y_axis='mel',
                                   sr=self.rate,
                                   fmax=22000,
                                   ax=ax)
    #plt.show()
    print(np.shape(norm_melspec))

    ############# TRATAMENTO TAMANHOS DO AUDIO ####################

    comprimentoAudio = np.shape(norm_melspec)[1]
    comprimentoMaximo = 64

    if comprimentoAudio < comprimentoMaximo:
```

```
     filler1 = math.floor((comprimentoMaximo-comprimentoAudio)/2)
     filler2 = math.ceil((comprimentoMaximo-comprimentoAudio)/2)
      fillerInicial = np.zeros((256,filler1))-80
      fillerFinal =np.zeros((256,filler2))-80
      norm_melspec = np.append(fillerInicial, norm_melspec, axis=1)
      norm_melspec = np.append(norm_melspec, fillerFinal, axis=1)
    else:
      norm_melspec = np.delete(norm_melspec, np.s_[comprimentoMaximo+1:comprimentoAudio+1],
axis=1)

    print(np.shape(norm_melspec))

    ############################################################
    ##################### MOSTRAR ESPECTOGRAMA ###############

    #img = librosa.display.specshow(norm_melspec, x_axis='time', y_axis='mel', sr=self.rate,
fmax=22000, ax=ax)

    #img = librosa.display.specshow(norm_melspec, x_axis='time', y_axis='mel', sr=self.rate,
fmax=22000, ax=ax)
    #librosa.display.specshow(norm_melspec, fmax=16000)
    #plt.colorbar(format='%+2.0f dB')
    #plt.title('Mel spectrogram')
    # plt.margins(0,0)
    # plt.gca().xaxis.set_major_locator(plt.NullLocator())
    # plt.gca().yaxis.set_major_locator(plt.NullLocator())

    if not os.path.exists(os.path.join(trainingDir, palavra)):
      os.makedirs(os.path.join(trainingDir, palavra))
    else:
      pass

    if not os.path.exists(os.path.join(validationDir, palavra)):
      os.makedirs(os.path.join(validationDir, palavra))
    else:
      pass

    trainingDir_n_files = len(os.listdir(os.path.join(trainingDir, palavra)))
    validationDir_n_files = len(os.listdir(os.path.join(validationDir, palavra)))

    print(validationDir_n_files)

    if trainingDir_n_files < trainingSamples:
      plt.savefig(os.path.join(os.path.join(trainingDir, palavra),
                               '{}_{}.png'.format(palavra,
                                                  trainingDir_n_files)),
                                                  transparent=True)
    elif validationDir_n_files < validationSamples:
      plt.savefig(os.path.join(os.path.join(validationDir, palavra),
```

```
                        '{}_{}.png'.format(palavra,
                                           trainingDir_n_files+validationDir_n_files)),
                                           transparent=True)

    plt.show()

    print('Returning to listening')

    if silent:
        self.record()

  def listen(self):
          print('Listening beginning')
          self.record()

a = AudioRecorder()

a.listen()
```

Anexo 2 – Código Conv2D.py

```
import os
import numpy as np
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, Dropout
import numpy as np
import tensorflow as tf
from keras.callbacks import ModelCheckpoint
from matplotlib import pyplot as plt
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix
from tensorflow import keras

# Define the directory where the dataset is located
training_dir = r'D:\Desktop\Dataset\\Training'
validation_dir = r'D:\Desktop\Dataset\\Validation'

img_height = int(480/2) #432 #28
img_width = int(640/2)

checkpoint_path = 'ModeloNumerosNormal.h5'
checkpoint_callback = ModelCheckpoint(checkpoint_path, monitor='val_accuracy',
save_best_only=True, mode='max', verbose=1)

model = Sequential()
model.add(Conv2D(4, (3,3), activation='relu', input_shape=(img_height, img_width,3)))
model.add(tf.keras.layers.LayerNormalization(axis=-1 , center=True , scale=True))
model.add(Flatten())
model.add(Dense(80, activation='relu'))
model.add(Dropout(0.3))
model.add(tf.keras.layers.LayerNormalization(axis=-1 , center=True , scale=True))
model.add(Dense(80, activation='relu'))
model.add(tf.keras.layers.LayerNormalization(axis=-1 , center=True , scale=True))
model.add(Dense(10, activation='softmax'))

model.summary()

model.compile(
    optimizer='adam',
    loss=tf.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy'])

          classes = sorted(os.listdir(training_dir))

          imagesTreino = tf.keras.utils.image_dataset_from_directory(
              training_dir,
```

```
    image_size=(img_height, img_width),
    batch_size=3)

imagesTest = tf.keras.utils.image_dataset_from_directory(
    validation_dir,
    image_size=(img_height, img_width),
    batch_size=3)

history = model.fit(imagesTreino, epochs=100, validation_data=imagesTest,
callbacks=[checkpoint_callback])

validation_losses = history.history['val_loss']
validation_accuracies = history.history['val_accuracy']

validation_losses = np.array(validation_losses)
validation_accuracies = np.array(validation_accuracies)

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]

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
y_val_true = []
y_val_pred_classes = []
for x_val_batch, y_val_batch in imagesTest:
    y_val_true.extend(y_val_batch.numpy())  # Directly extend the true class labels
    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(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%}")

# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_val_true, y_val_pred_classes)

# Plot the confusion matrix as a heatmap with numbers inside cells
```

```
plt.figure(figsize=(8, 6))
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)

for i in range(len(conf_matrix)):
    for j in range(len(conf_matrix[i])):
        plt.text(j, i, str(conf_matrix[i, j]), ha='center', va='center', color='black')

plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(conf_matrix))
plt.xticks(tick_marks, tick_marks)
plt.yticks(tick_marks, tick_marks)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

# Plot the training and validation accuracy graph
plt.figure(figsize=(8, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()

# Plot the training and validation loss graph
plt.figure(figsize=(8, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()
```

Anexo 3 – Código Conv2DAugmented.py

```
import numpy as np
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, Dropout
import numpy as np
import tensorflow as tf
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 the directory where the dataset is located
training_dir = r'D:\Desktop\Dataset\\Training'
validation_dir = r'D:\Desktop\Dataset\\Validation'

img_height = int(480/3) #432 #28
img_width = int(640/3)

checkpoint_path = 'ModeloNumerosAugmented.h5'
checkpoint_callback = ModelCheckpoint(checkpoint_path, monitor='val_accuracy',
save_best_only=True, mode='max', verbose=1)

model = Sequential()
model.add(Conv2D(4, (3,3), activation='relu', input_shape=(img_height, img_width,3)))
model.add(tf.keras.layers.LayerNormalization(axis=-1 , center=True , scale=True))
model.add(Flatten())
model.add(Dense(80, activation='relu'))
model.add(Dropout(0.3))
model.add(tf.keras.layers.LayerNormalization(axis=-1 , center=True , scale=True))
model.add(Dense(80, activation='relu'))
model.add(tf.keras.layers.LayerNormalization(axis=-1 , center=True , scale=True))
model.add(Dense(10, activation='softmax'))

model.summary()

model.compile(
    optimizer='adam',
    loss=tf.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy'])

train_datagen = ImageDataGenerator(rotation_range=2,
                                   height_shift_range=15,
                                   samplewise_center=True,
                                   samplewise_std_normalization=True)
```

```
validation_datagen = ImageDataGenerator(rotation_range=2,
                                        height_shift_range=15,
                                        samplewise_center=True,
                                        samplewise_std_normalization=True)

train_generator = train_datagen.flow_from_directory(
    training_dir,
    target_size=(img_height, img_width),
    batch_size=3,
    class_mode='sparse',
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(img_height, img_width),
    batch_size=3,
    class_mode='sparse',
)


history = model.fit(train_generator, epochs=100, validation_data=validation_generator,
callbacks=[checkpoint_callback])

validation_losses = history.history['val_loss']
validation_accuracies = history.history['val_accuracy']

validation_losses = np.array(validation_losses)
validation_accuracies = np.array(validation_accuracies)

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]

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 = []
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(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%}")

# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_val_true, y_val_pred_classes)

# Plot the confusion matrix as a heatmap with numbers inside cells
plt.figure(figsize=(8, 6))
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)

for i in range(len(conf_matrix)):
    for j in range(len(conf_matrix[i])):
        plt.text(j, i, str(conf_matrix[i, j]), ha='center', va='center', color='black')

plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(conf_matrix))
plt.xticks(tick_marks, tick_marks)
plt.yticks(tick_marks, tick_marks)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

# Plot the training and validation accuracy graph
plt.figure(figsize=(8, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()

# Plot the training and validation loss graph
plt.figure(figsize=(8, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
```

```
plt.legend()
plt.title('Training and Validation Loss')
plt.show()
```

Anexo 4 – Código EfficientNetAugmented.py

```
import numpy as np
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
import tensorflow as tf
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 the directory where the dataset is located
training_dir = r'D:\Desktop\Dataset\\Training'
validation_dir = r'D:\Desktop\Dataset\\Validation'

img_height = int(480/3) #432 #28
img_width = int(640/3)

checkpoint_path = 'ModeloNumerosEfficientNet.h5'
checkpoint_callback = ModelCheckpoint(checkpoint_path, monitor='val_accuracy',
save_best_only=True, mode='max', verbose=1)

pretrained_model=tf.keras.applications.EfficientNetB7(
    include_top=False,
    weights='imagenet',
    input_tensor=None,
    input_shape=(img_height, img_width,3),
    pooling='avg',
    classes=10)

model = Sequential()
model.add(pretrained_model)
model.add(Dense(10, activation='softmax'))


model.summary()

model.compile(
    optimizer='adam',
    loss=tf.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy'])

train_datagen = ImageDataGenerator(height_shift_range=15, samplewise_center=True,
samplewise_std_normalization=True)
```

```
validation_datagen = ImageDataGenerator(height_shift_range=15, samplewise_center=True,
samplewise_std_normalization=True)

train_generator = train_datagen.flow_from_directory(
    training_dir,
    target_size=(img_height, img_width),
    batch_size=3,
    class_mode='sparse',
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(img_height, img_width),
    batch_size=3,
    class_mode='sparse',
)

history = model.fit(train_generator, epochs=100, validation_data=validation_generator,
callbacks=[checkpoint_callback])

validation_losses = history.history['val_loss']
validation_accuracies = history.history['val_accuracy']

validation_losses = np.array(validation_losses)
validation_accuracies = np.array(validation_accuracies)

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]

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 = []
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(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%}")

# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_val_true, y_val_pred_classes)

# Plot the confusion matrix as a heatmap with numbers inside cells
plt.figure(figsize=(8, 6))
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)

for i in range(len(conf_matrix)):
    for j in range(len(conf_matrix[i])):
        plt.text(j, i, str(conf_matrix[i, j]), ha='center', va='center', color='black')

plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(conf_matrix))
plt.xticks(tick_marks, tick_marks)
plt.yticks(tick_marks, tick_marks)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

# Plot the training and validation accuracy graph
plt.figure(figsize=(8, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()

# Plot the training and validation loss graph
plt.figure(figsize=(8, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()
```

Anexo 5 – Código ResNet50Augmented.py

```
import numpy as np
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
import tensorflow as tf
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 the directory where the dataset is located
training_dir = r'D:\Desktop\Dataset\\Training'
validation_dir = r'D:\Desktop\Dataset\\Validation'

img_height = int(480/3) #432 #28
img_width = int(640/3)

checkpoint_path = 'FinalResNet50.h5'
checkpoint_callback = ModelCheckpoint(checkpoint_path, monitor='val_accuracy',
save_best_only=True, mode='max', verbose=1)

pretrained_model=tf.keras.applications.ResNet50(
    include_top=False,
    weights='imagenet',
    input_tensor=None,
    input_shape=(img_height, img_width,3),
    pooling='avg',
    classes=10)

model = Sequential()
model.add(pretrained_model)
model.add(Dense(10, activation='softmax'))


model.summary()

model.compile(
    optimizer='adam',
    loss=tf.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy'])

train_datagen = ImageDataGenerator(height_shift_range=15, samplewise_center=True,
samplewise_std_normalization=True)
```

```
validation_datagen = ImageDataGenerator(height_shift_range=15, samplewise_center=True,
samplewise_std_normalization=True)

train_generator = train_datagen.flow_from_directory(
    training_dir,
    target_size=(img_height, img_width),
    batch_size=3,
    class_mode='sparse',
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(img_height, img_width),
    batch_size=3,
    class_mode='sparse',
)

history = model.fit(train_generator, epochs=100, validation_data=validation_generator,
callbacks=[checkpoint_callback])

validation_losses = history.history['val_loss']
validation_accuracies = history.history['val_accuracy']

validation_losses = np.array(validation_losses)
validation_accuracies = np.array(validation_accuracies)

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]

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 = []
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(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%}")

# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_val_true, y_val_pred_classes)

# Plot the confusion matrix as a heatmap with numbers inside cells
plt.figure(figsize=(8, 6))
plt.imshow(conf_matrix, interpolation='nearest', cmap=plt.cm.Blues)

for i in range(len(conf_matrix)):
    for j in range(len(conf_matrix[i])):
        plt.text(j, i, str(conf_matrix[i, j]), ha='center', va='center', color='black')

plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(len(conf_matrix))
plt.xticks(tick_marks, tick_marks)
plt.yticks(tick_marks, tick_marks)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

# Plot the training and validation accuracy graph
plt.figure(figsize=(8, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')
plt.show()

# Plot the training and validation loss graph
plt.figure(figsize=(8, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()
```

Anexo 6 – Código controlMouseMultiprocessing.py (Aplicação)

```
import pyautogui
from tensorflow import keras
import numpy as np
from matplotlib import pyplot as plt
from sys import byteorder
import librosa
import librosa.display
from array import array
import pyaudio
from keras.models import load_model
from PIL import Image
import matplotlib
matplotlib.use('Agg')
import multiprocessing
from multiprocessing import Value

model = keras.models.load_model(r'ModeloNormalFinal.h5')

img_height = int(480/3) #432 #28
img_width = int(640/3)

class AudioRecorder:
    def __init__(self):
        self.rate = 44100
        self.threshold = 0.014 # silence threshold {Need to experiment with it}
        self.chunk_size = 1024
        self.format = pyaudio.paFloat32
        self._pyaudio = pyaudio.PyAudio()

    def isSilent(self, data):
        # Returns true if below the silence threshold
        return max(data) < self.threshold

    def trim(self, data):
        # Trim the blanks at the start and end
        def _trim(data):
            started = False
            r = array('f')
            for i in data:
                if not started and abs(i) > self.threshold:
                    started = True
                    r.append(i)
                elif started:
                    r.append(i)
```

```
            return r

        # First trim the left side
        data = _trim(data)
        data.reverse()

        # Then trim the right side
        data = _trim(data)
        data.reverse()

        return data

    def addSilence(self, data):
        # Adds silence to the start and end of 0.1 seconds
        r = array('f', [0 for i in range(int(0.1 * self.rate))])
        r.extend(data)
        r.extend([0 for i in range(int(0.1 * self.rate))])

        return r

    def record(self, is_moving, movement_direction, speed):
        stream = self._pyaudio.open(format=self.format, channels=1, rate=self.rate,
input=True, output=True, frames_per_buffer=self.chunk_size)
        numSilent = 0
        started = False
        r = array('f')

        while True:
            # Has to be little endian and signed short
            data = array('f', stream.read(self.chunk_size))

            if byteorder == 'big':
                data.byteswap()

            r.extend(data)
            silent = self.isSilent(data)

            if silent and started:
                numSilent += 1
            elif not silent and not started:
                started = True
                print('Noise detected, recording beginning')
            if started and numSilent > 20:
                break

        stream.stop_stream()
        stream.close()

        r = self.trim(r)
```

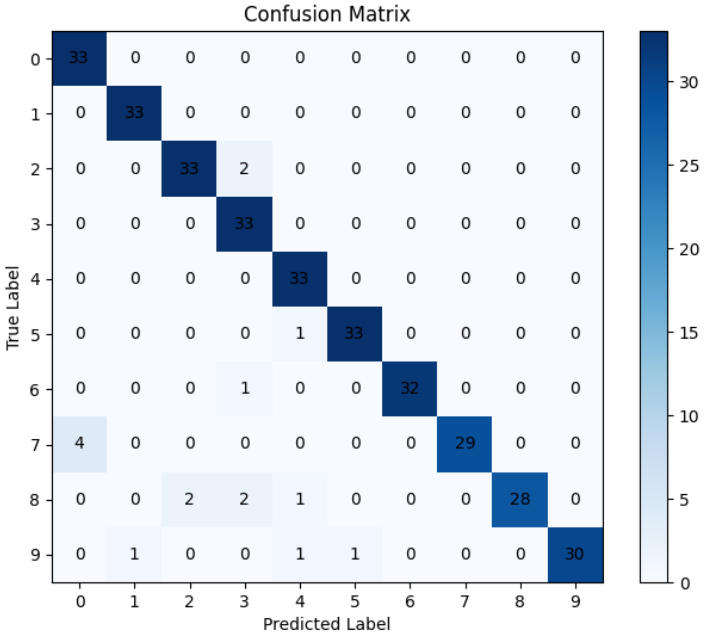
```
        r = self.addSilence(r)

        r = np.array(r, dtype=np.float32)

        fig, ax = plt.subplots()

        melspec = librosa.feature.melspectrogram(y=r, sr=self.rate, n_mels=256, fmax=22000,
hop_length=8)
        norm_melspec = librosa.core.power_to_db(melspec, ref=np.max)
        img = librosa.display.specshow(norm_melspec, x_axis='time', y_axis='mel',
sr=self.rate, fmax=22000, ax=ax)
        plt.savefig(('spokenWord.png'), transparent=True)
        plt.close()

        imgToPredict = Image.open('spokenWord.png').convert('RGB')
        imgToPredict = imgToPredict.resize((img_width, img_height))
        im_np = np.asarray(imgToPredict)
        print(np.shape(im_np))
        numpydata = im_np.reshape(1, img_height, img_width, 3)
        predictions = model.predict(numpydata)
        pred_labels = np.argmax(predictions, axis=1)
        print("Result")
        print(pred_labels)

        if pred_labels[0] == 0:

            #print(is_moving.value)
            print('UP')
            if not bool(is_moving.value):
                is_moving.value = True
                movement_direction.value = 0
            else:
                is_moving.value = False
                movement_direction.value = 0
                is_moving.value = True

        elif pred_labels[0] == 1:
            print('DOWN')
            if not bool(is_moving.value):
                is_moving.value = True
                movement_direction.value = 1
            else:
                is_moving.value = False
                movement_direction.value = 1
                is_moving.value = True

        elif pred_labels[0] == 2:
            print('RIGHT')
            if not bool(is_moving.value):
```

```
            is_moving.value = True
            movement_direction.value = 2
        else:
            is_moving.value = False
            movement_direction.value = 2
            is_moving.value = True

    elif pred_labels[0] == 3:
        print('LEFT')
        if not bool(is_moving.value):
            is_moving.value = True
            movement_direction.value = 3
        else:
            is_moving.value = False
            movement_direction.value = 3
            is_moving.value = True

    elif pred_labels[0] == 4:
        print('STOP')
        is_moving.value = False

    elif pred_labels[0] == 5:
        print('CLICK')
        is_moving.value = False
        pyautogui.click()

    elif pred_labels[0] == 6:
        print('DECREASE SPEED')
        if speed.value >= 4:
            speed.value = speed.value - 3

    elif pred_labels[0] == 7:
        print('INCREASE SPEED')
        if speed.value <= 7:
            speed.value = speed.value + 3
    else:
        print('DOUBLE CLICK')
        is_moving.value = False
        pyautogui.click(clicks=2, interval=0.25)

    print(speed.value)
    print('Returning to listening')

    if silent:
        self.record(is_moving, movement_direction, speed)

def listen(self, is_moving, movement_direction, speed):
    print('Listening beginning')
    self.record(is_moving, movement_direction, speed)
```

```
class MouseController:
    def move_mouse(self, is_moving, movement_direction, speed):
        if bool(is_moving.value):
            if movement_direction.value == 0:
                pyautogui.move(0, -speed.value)

            elif movement_direction.value == 1:
                pyautogui.move(0, speed.value)

            elif movement_direction.value == 2:
                pyautogui.move(speed.value, 0)

            elif movement_direction.value == 3:
                pyautogui.move(-speed.value, 0)

a = AudioRecorder()
mouse_controller = MouseController()

def audio_thread(is_moving, movement_direction, speed):
    a.listen(is_moving, movement_direction, speed)

def mouse_thread(is_moving, movement_direction, speed):
    while True:
        mouse_controller.move_mouse(is_moving, movement_direction, speed)

if __name__ == '__main__':
    is_moving = Value('i', False)
    movement_direction = Value('i', 0)
    speed = Value('i', 10)

    audio_process = multiprocessing.Process(target=audio_thread, args=(is_moving,
                                                                       movement_direction,
                                                                       speed))
    mouse_process = multiprocessing.Process(target=mouse_thread, args=(is_moving,
                                                                       movement_direction,
                                                                       speed))

    audio_process.start()
    mouse_process.start()

    audio_process.join()
    mouse_process.join()
```

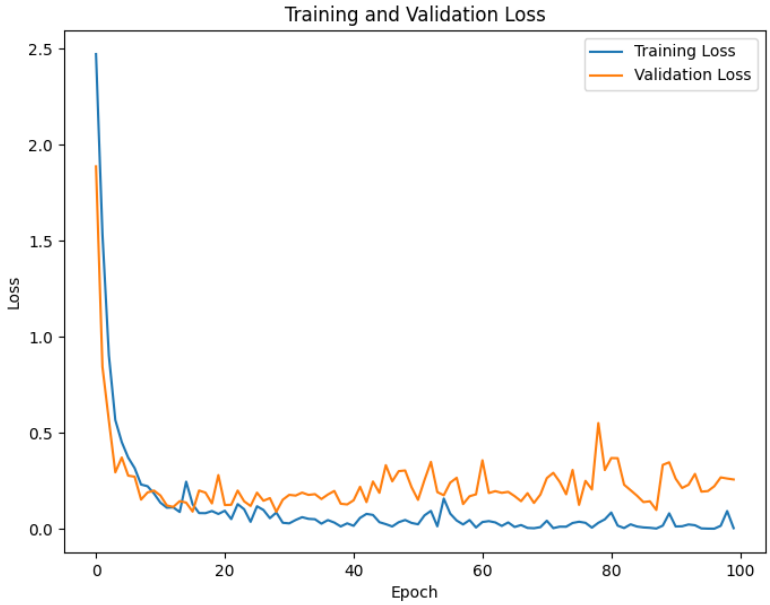
Anexo 7 - Conv2D com *Augmentation* (Tabela 1, linha 1)

- Matriz de Confusão



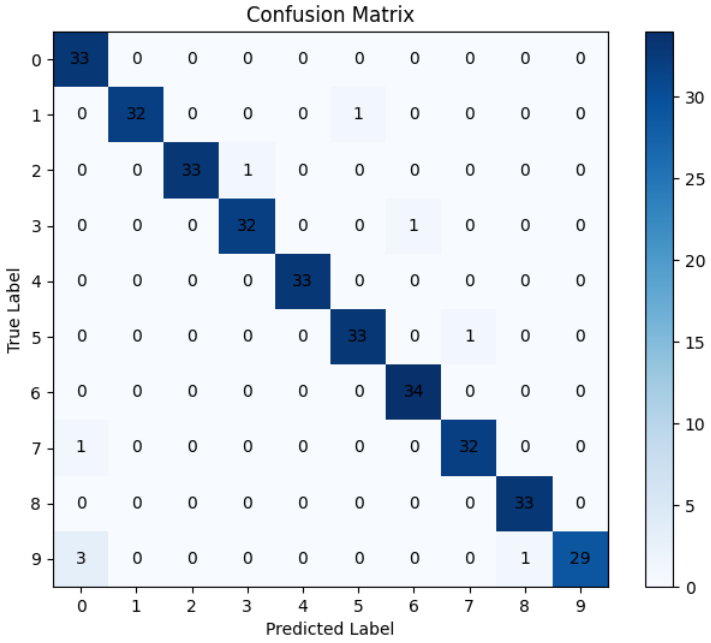
- Gráfico de *Accuracy*

- Gráfico de *Loss*

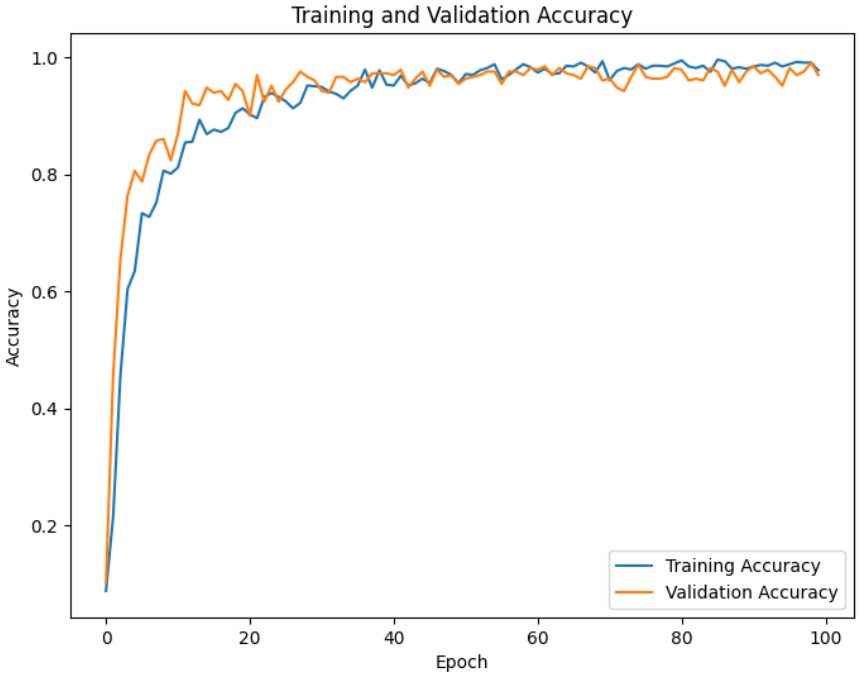


Anexo 8 - Conv2D com *Augmentation* (Tabela 1, linha 2)

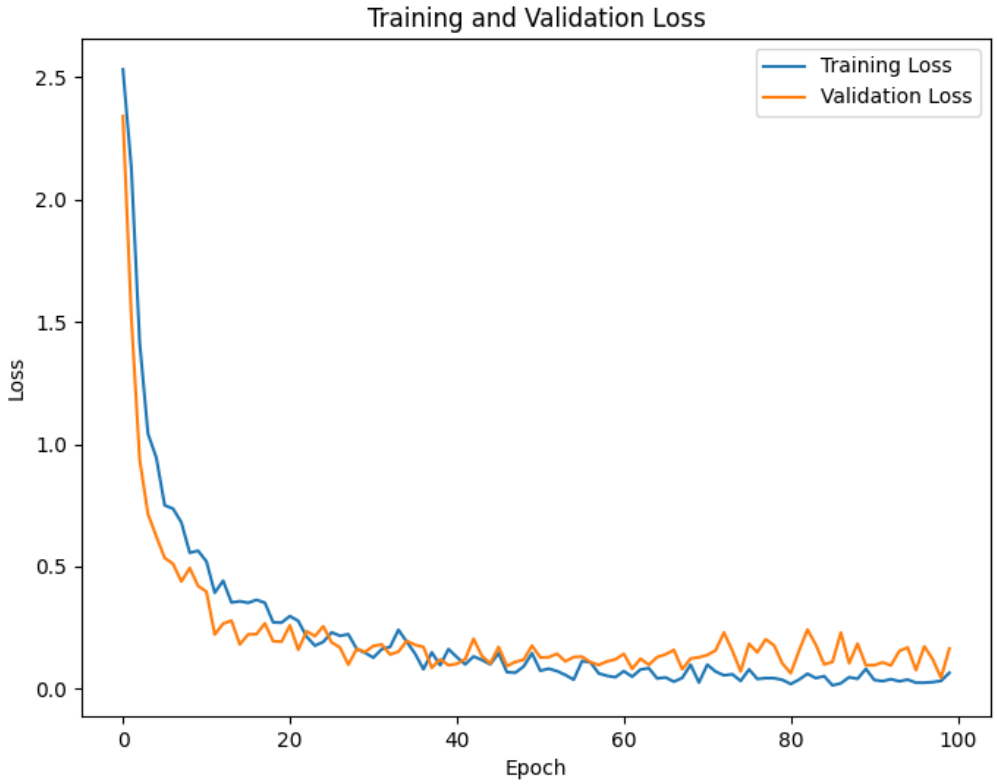
- Matriz de Confusão



- Gráfico de *Accuracy*

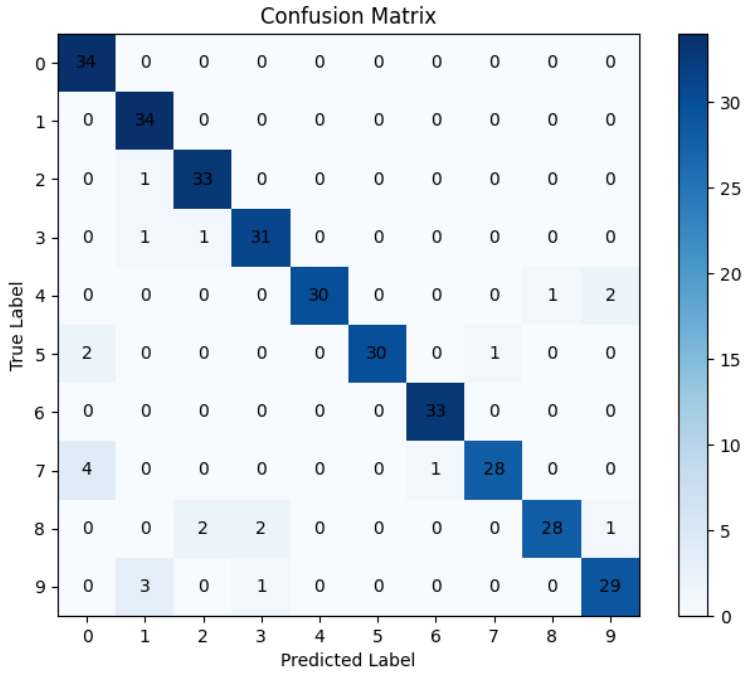


- Gráfico de *Loss*

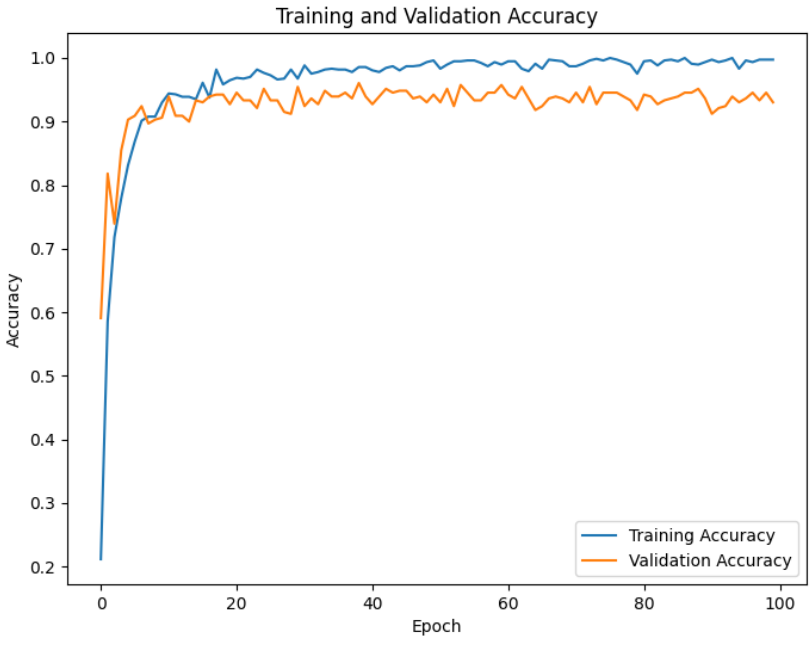


Anexo 9 - Conv2D com *Augmentation* (Tabela 1, linha 3)

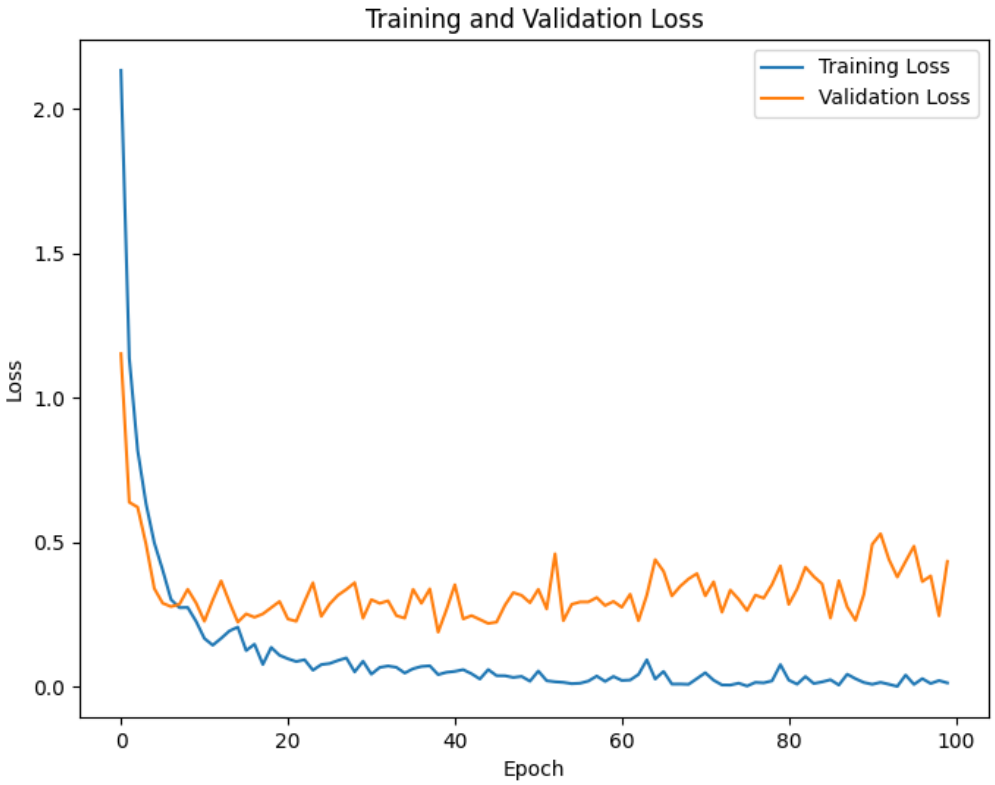
- Matriz de Confusão



- Gráfico de *Accuracy*

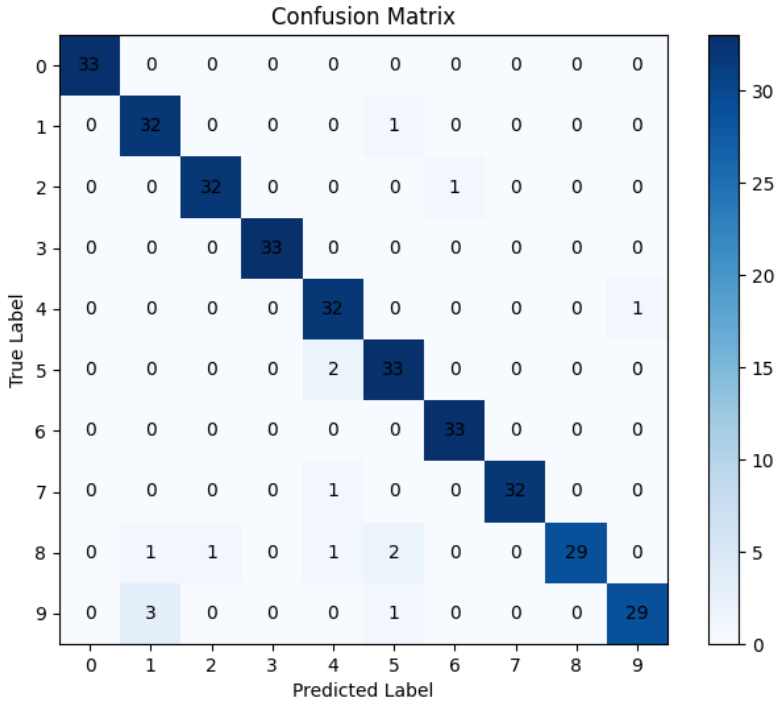


- Gráfico de *Loss*



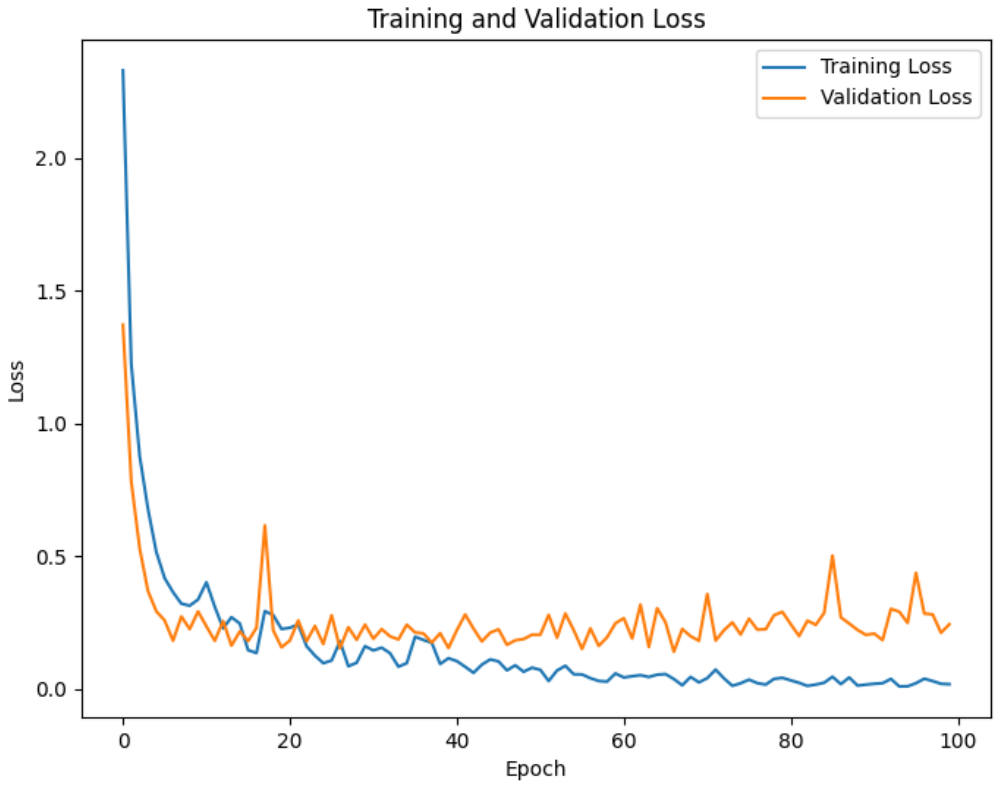
Anexo 10 - Conv2D com *Augmentation* (Tabela 1, linha 4)

- Matriz de Confusão



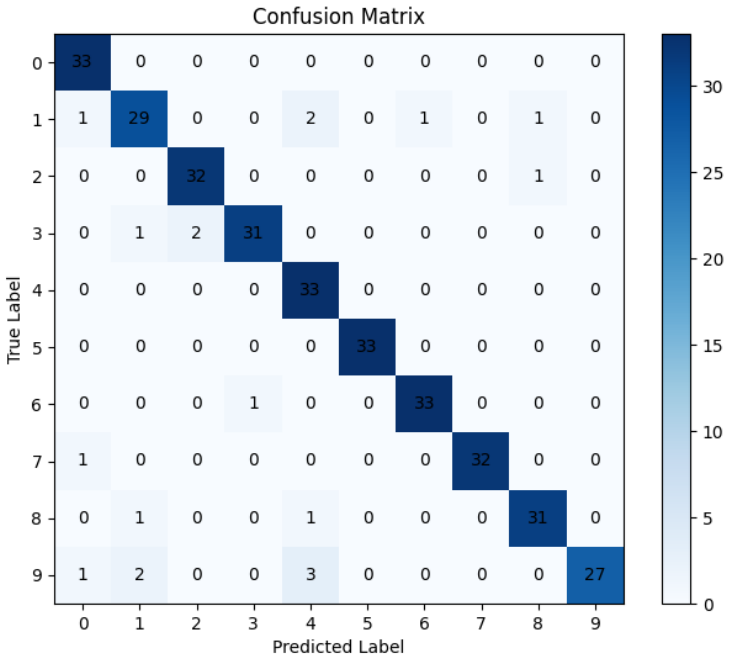
- Gráfico de *Accuracy*

- Gráfico de *Loss*

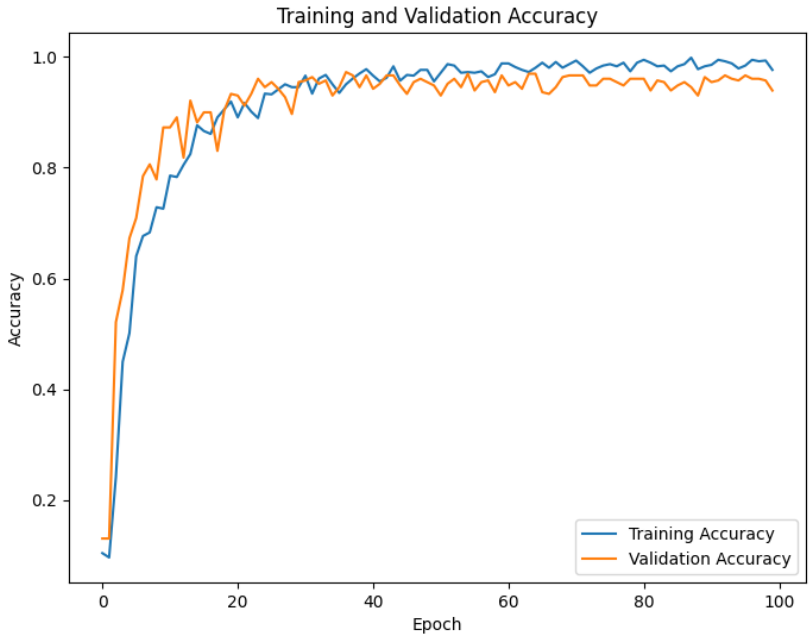


Anexo 11 - Conv2D com *Augmentation* (Tabela 1, linha 5)

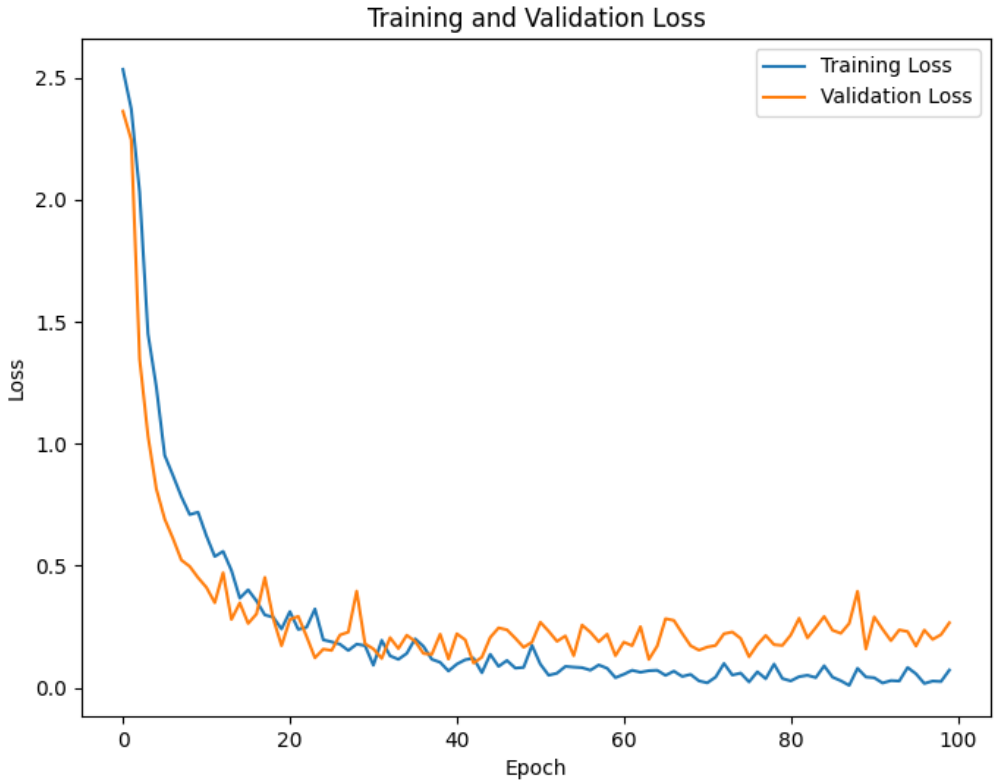
- Matriz de Confusão



- Gráfico de *Accuracy*

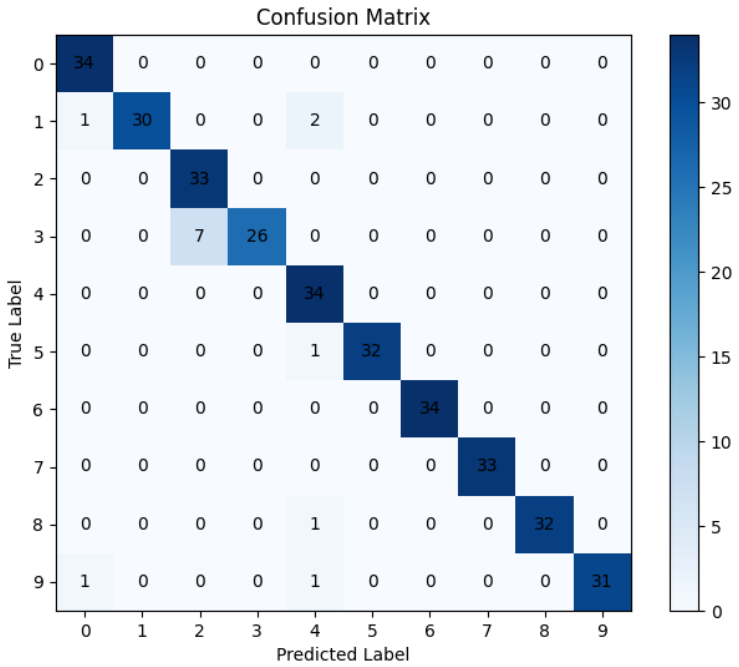


- Gráfico de *Loss*

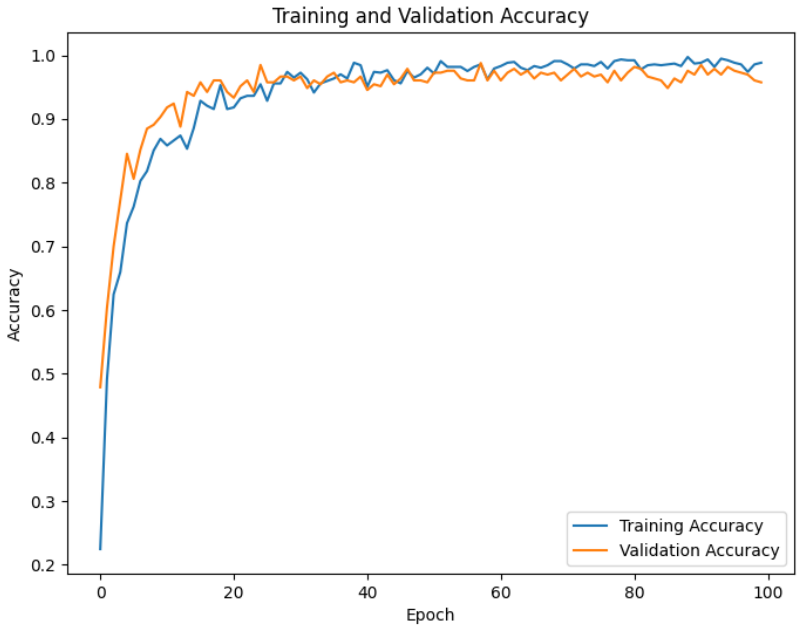


Anexo 12 - Conv2D com *Augmentation* (Tabela 1, linha 6)

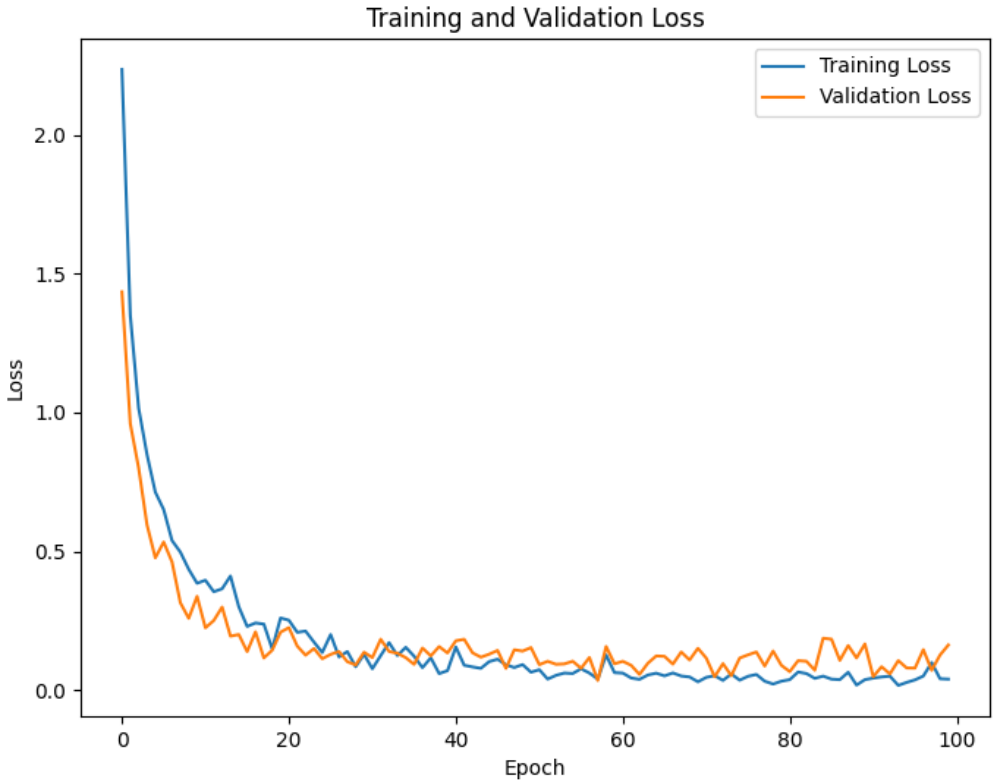
- Matriz de Confusão



- Gráfico de *Accuracy*

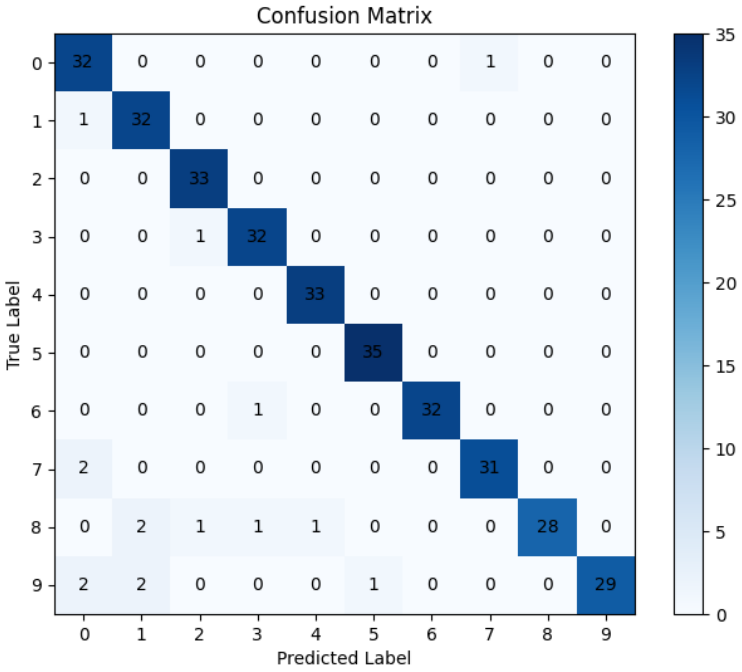


- Gráfico de *Loss*

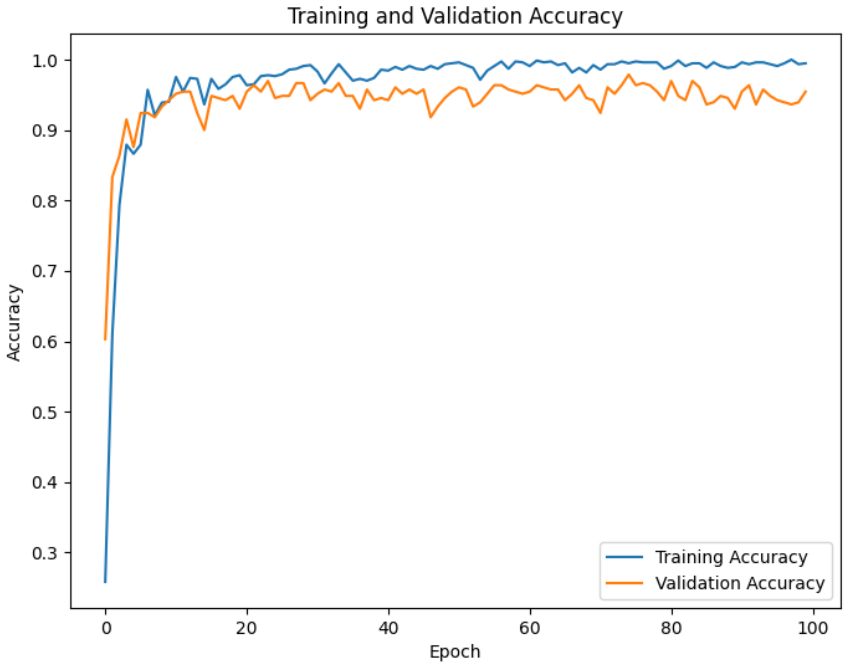


Anexo 13 - Conv2D com *Augmentation* (Tabela 1, linha 7)

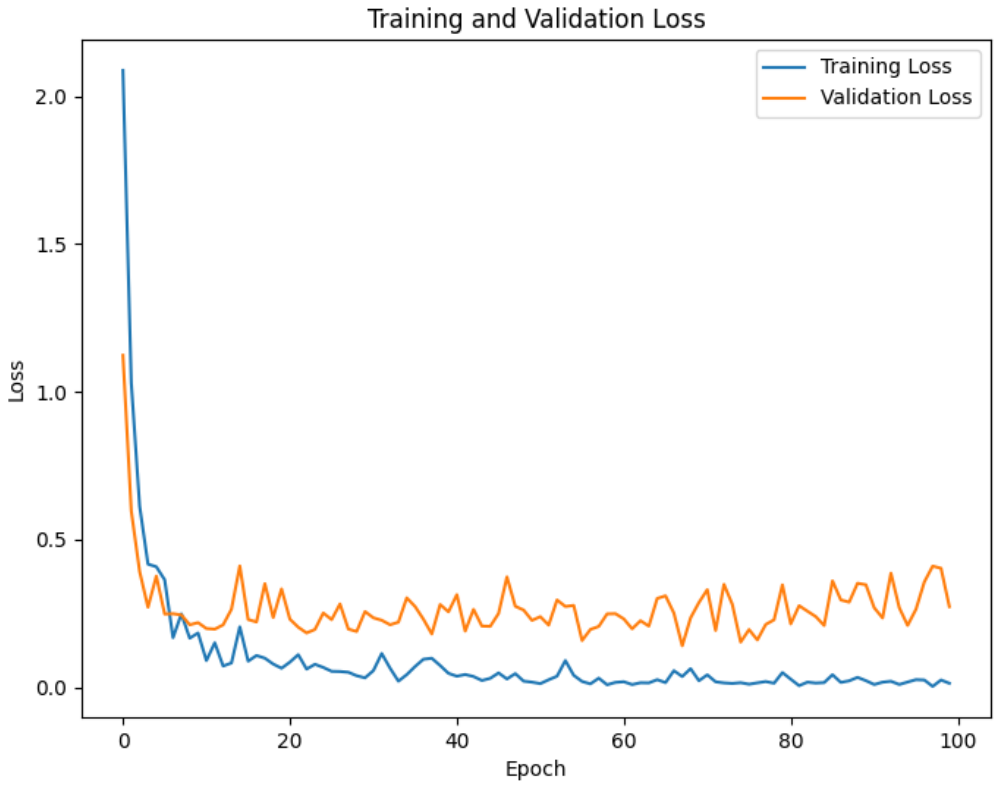
- Matriz de Confusão



- Gráfico de *Accuracy*

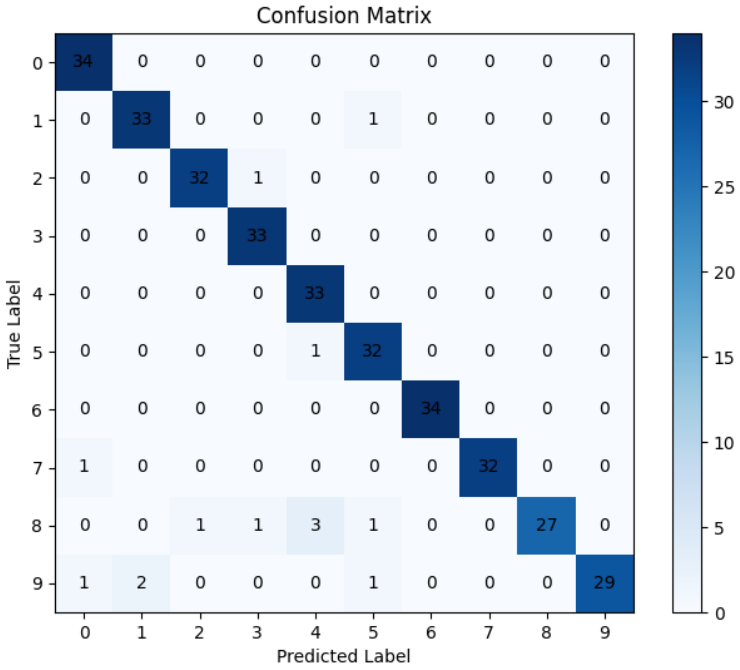


- Gráfico de *Loss*



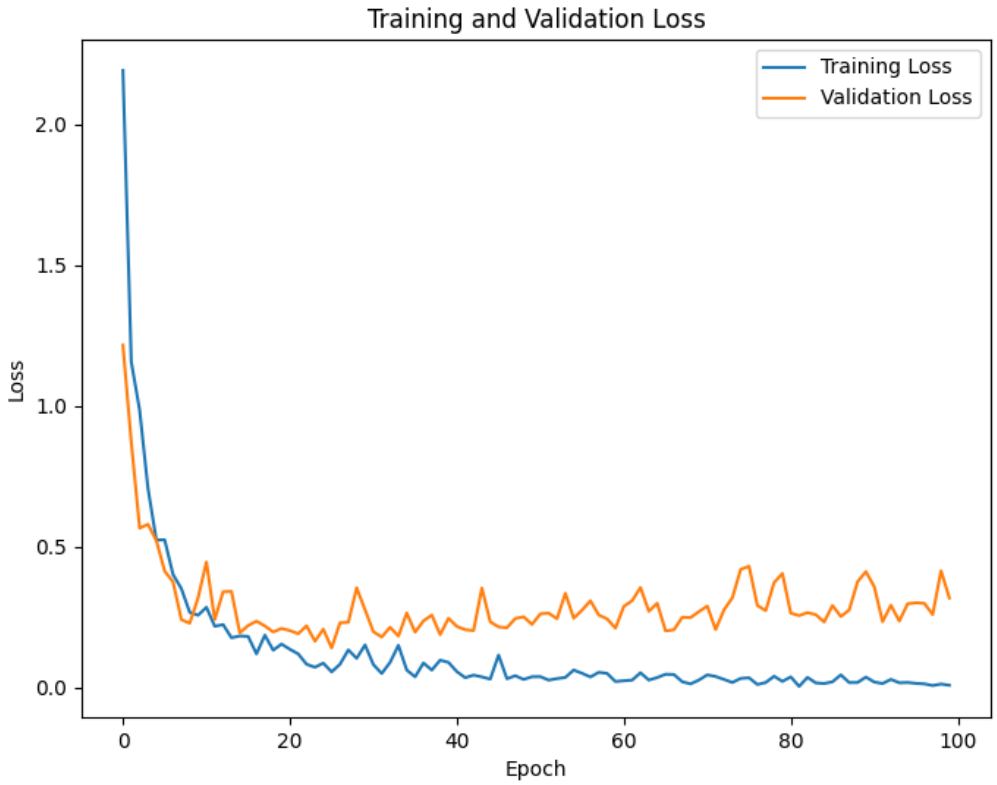
Anexo 14 - Conv2D com *Augmentation* (Tabela 1, linha 8)

- Matriz de Confusão



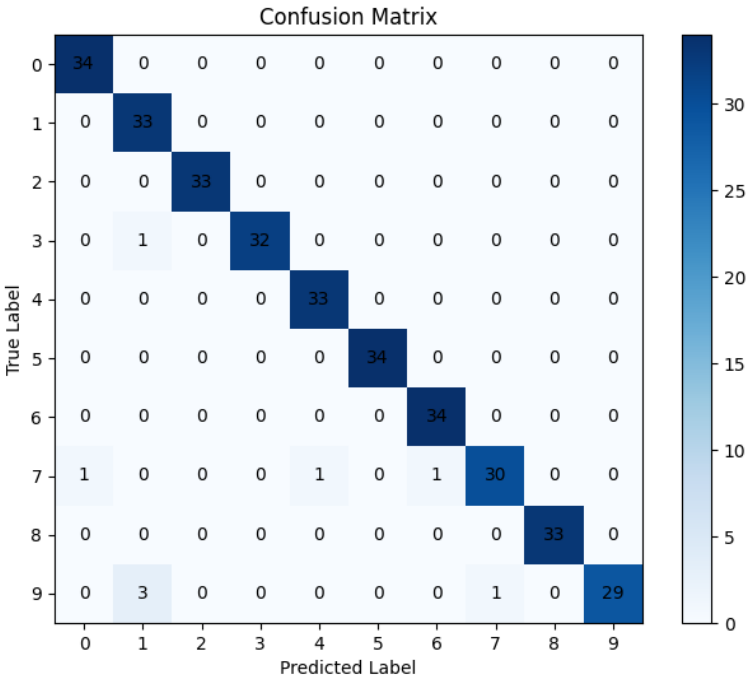
- Gráfico de *Accuracy*

- Gráfico de *Loss*



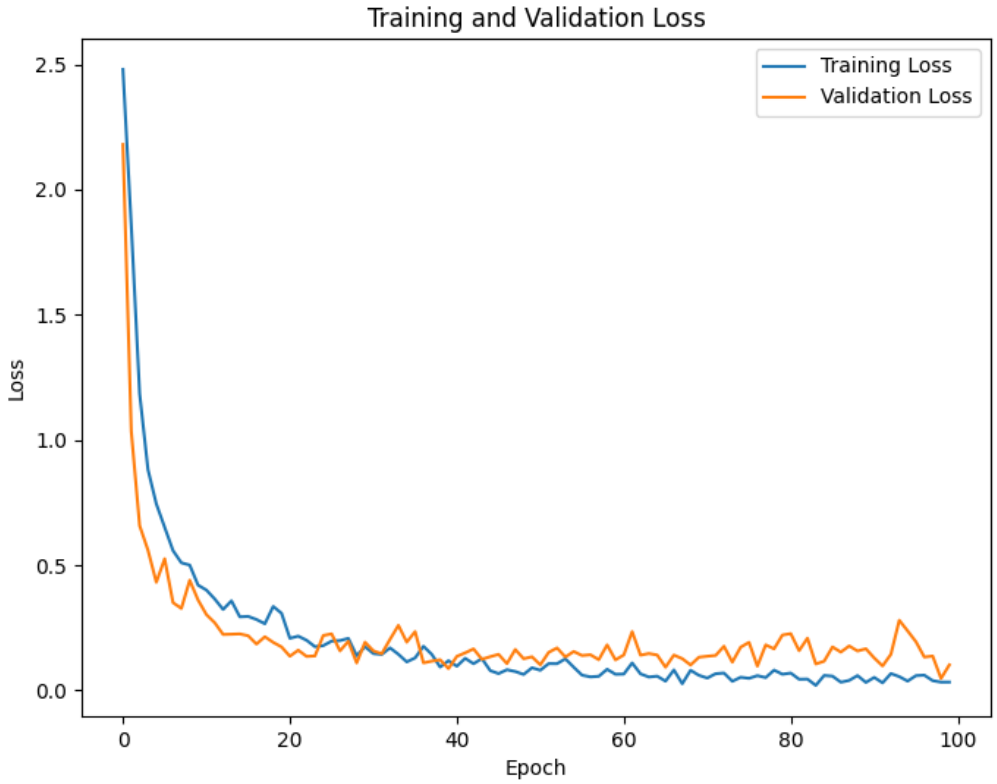
Anexo 15 - Conv2D com *Augmentation* (Tabela 1, linha 9)

- Matriz de Confusão



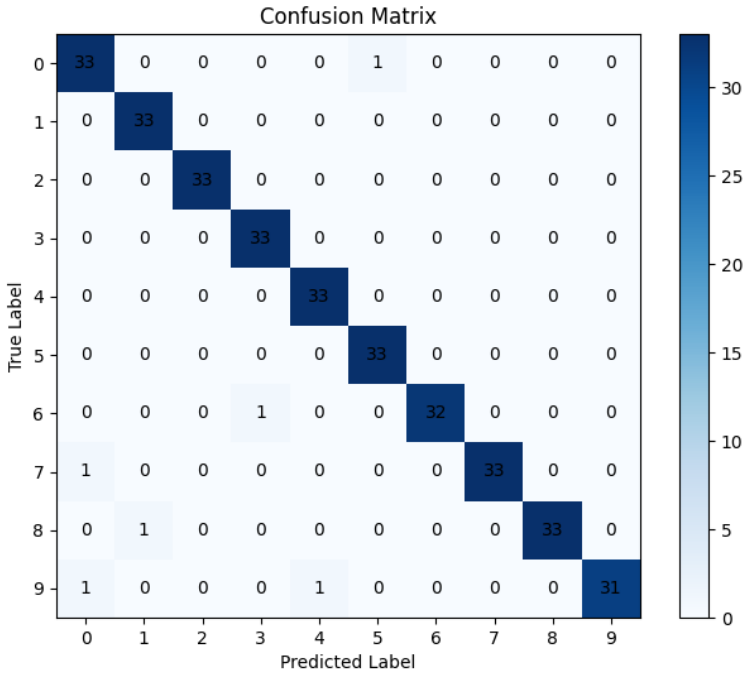
- Gráfico de *Accuracy*

- Gráfico de *Loss*

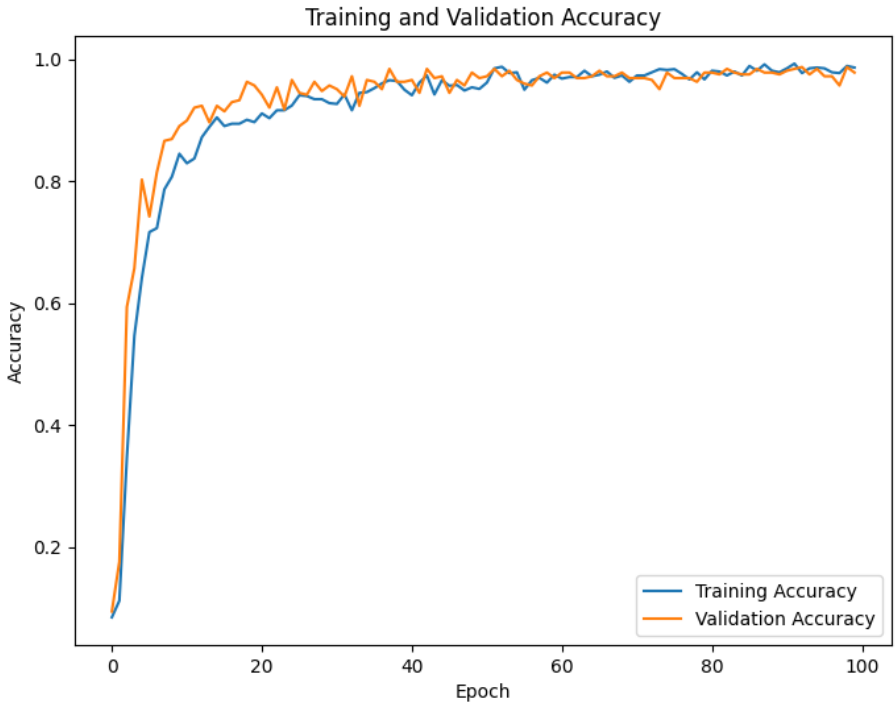


Anexo 16 - Conv2D com *Augmentation* (Tabela 1, linha 10)

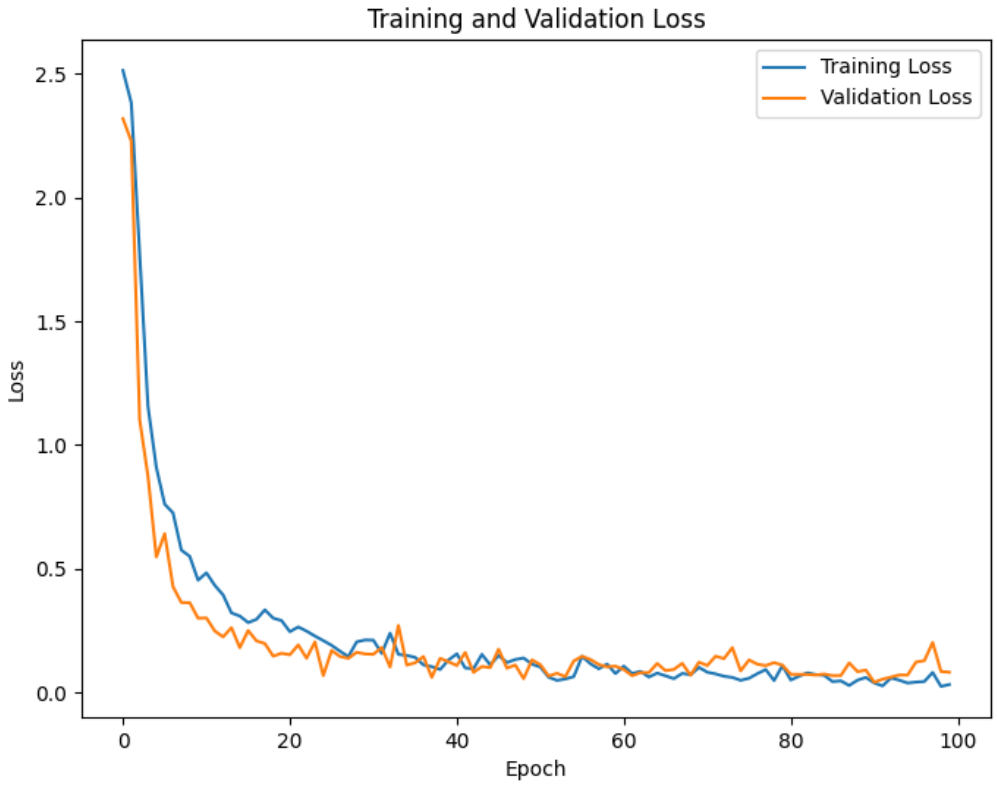
- Matriz de Confusão



- Gráfico de *Accuracy*

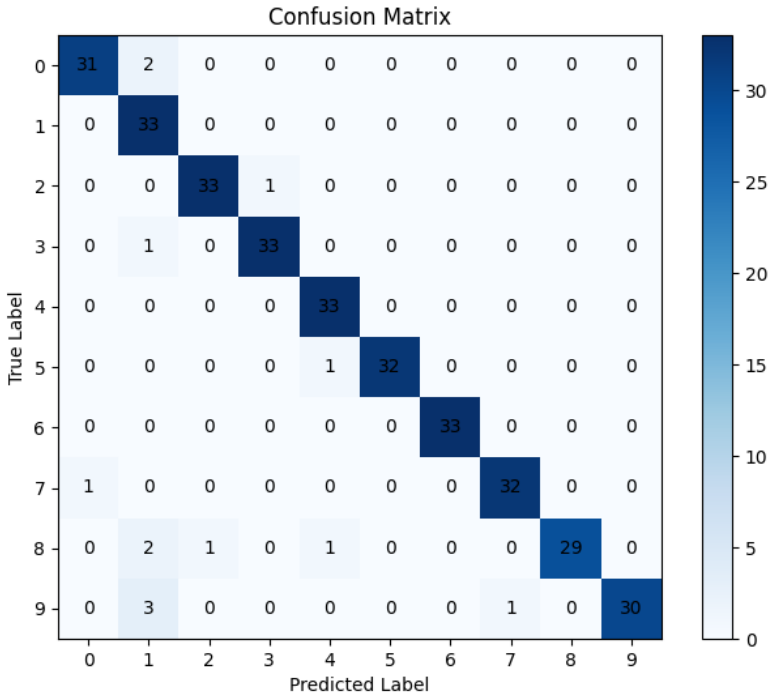


- Gráfico de *Loss*

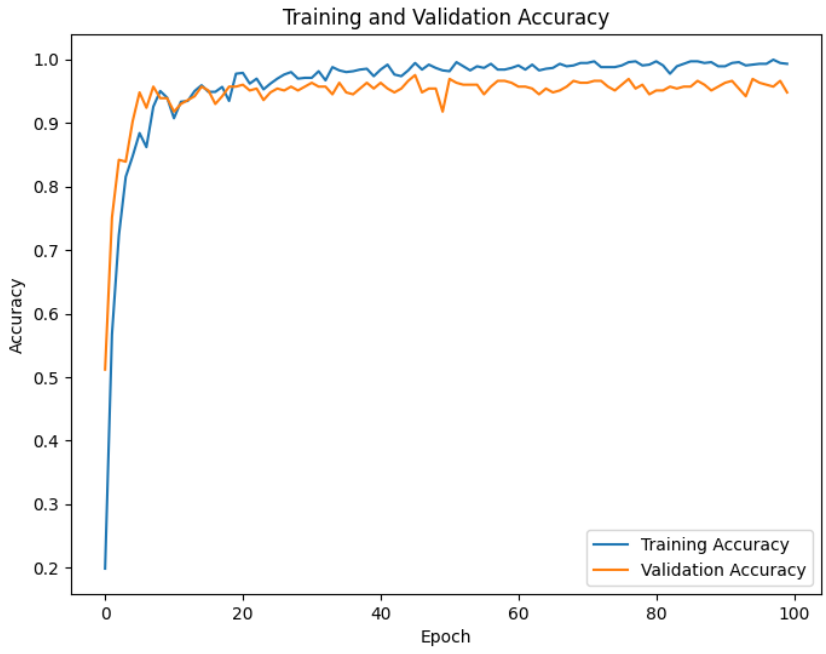


Anexo 17 - Conv2D com *Augmentation* (Tabela 1, linha 11)

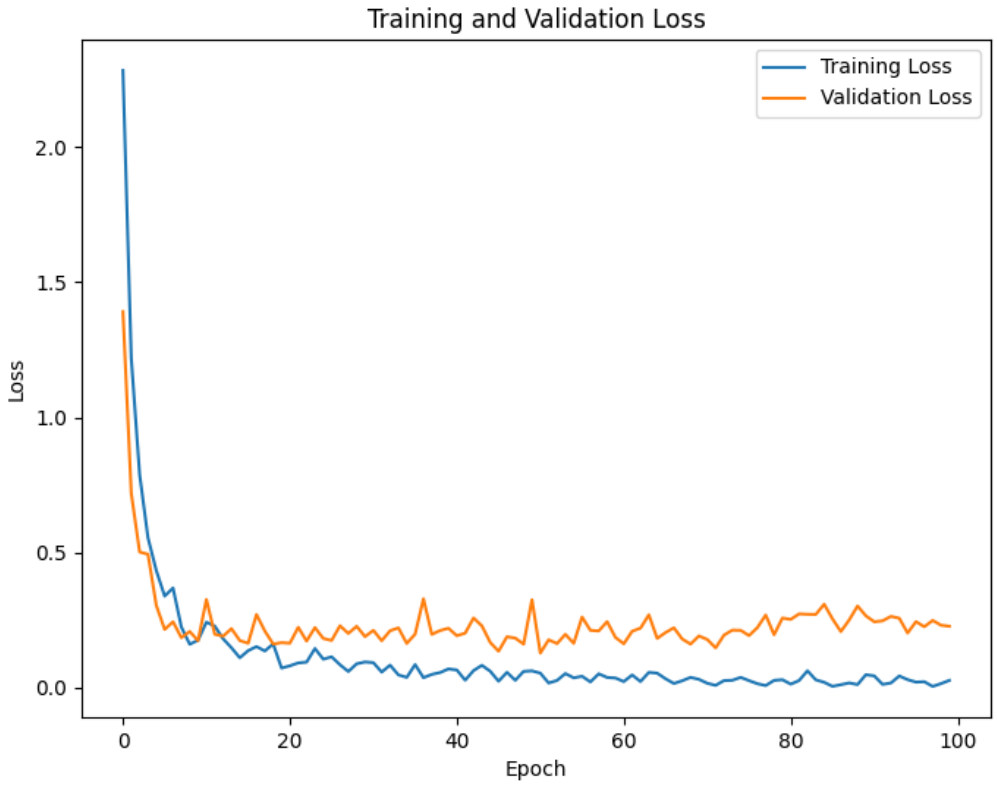
- Matriz de Confusão



- Gráfico de *Accuracy*

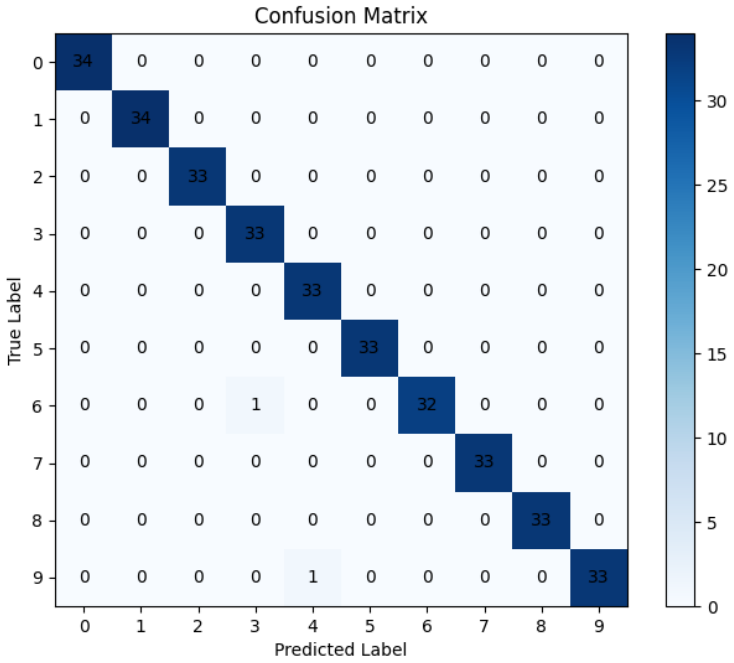


- Gráfico de *Loss*

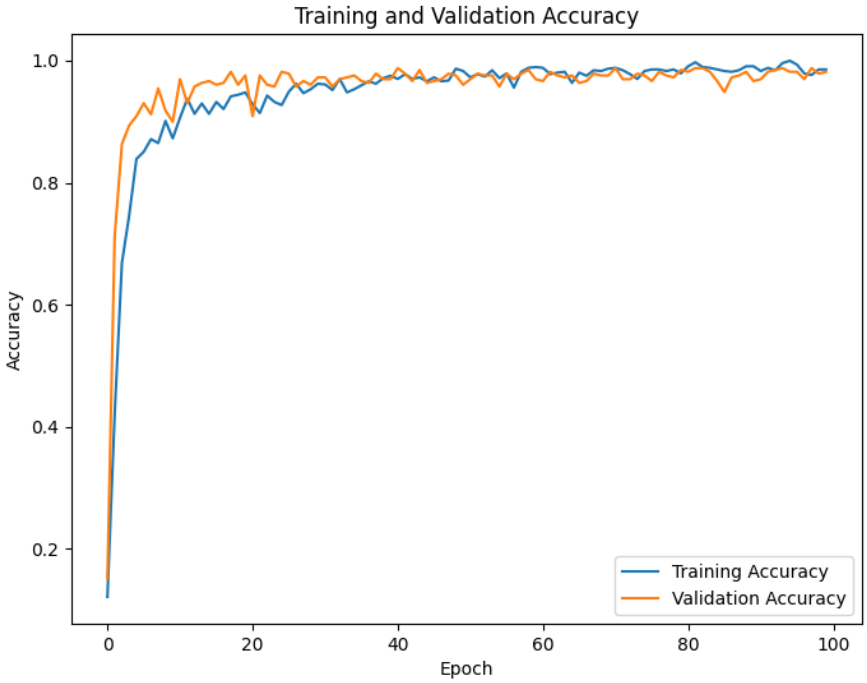


Anexo 18 - Conv2D com *Augmentation* (Tabela 1, linha 12)

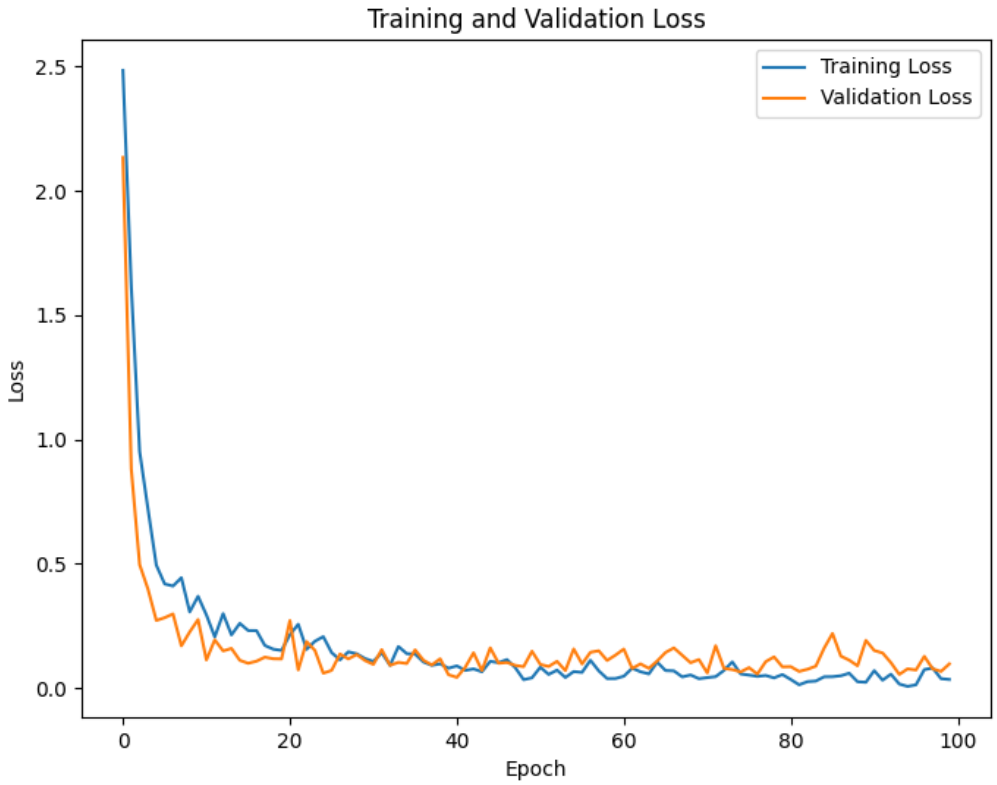
- Matriz de Confusão



- Gráfico de *Accuracy*

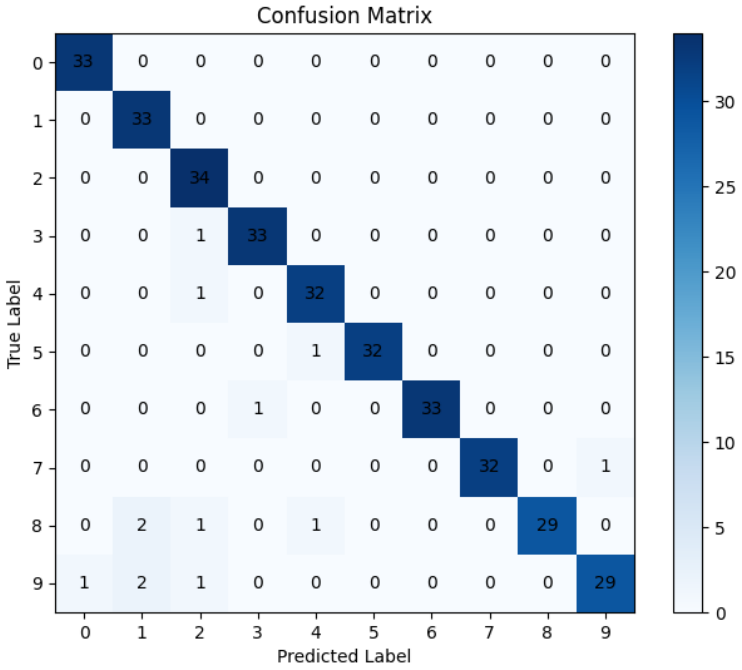


- Gráfico de *Loss*



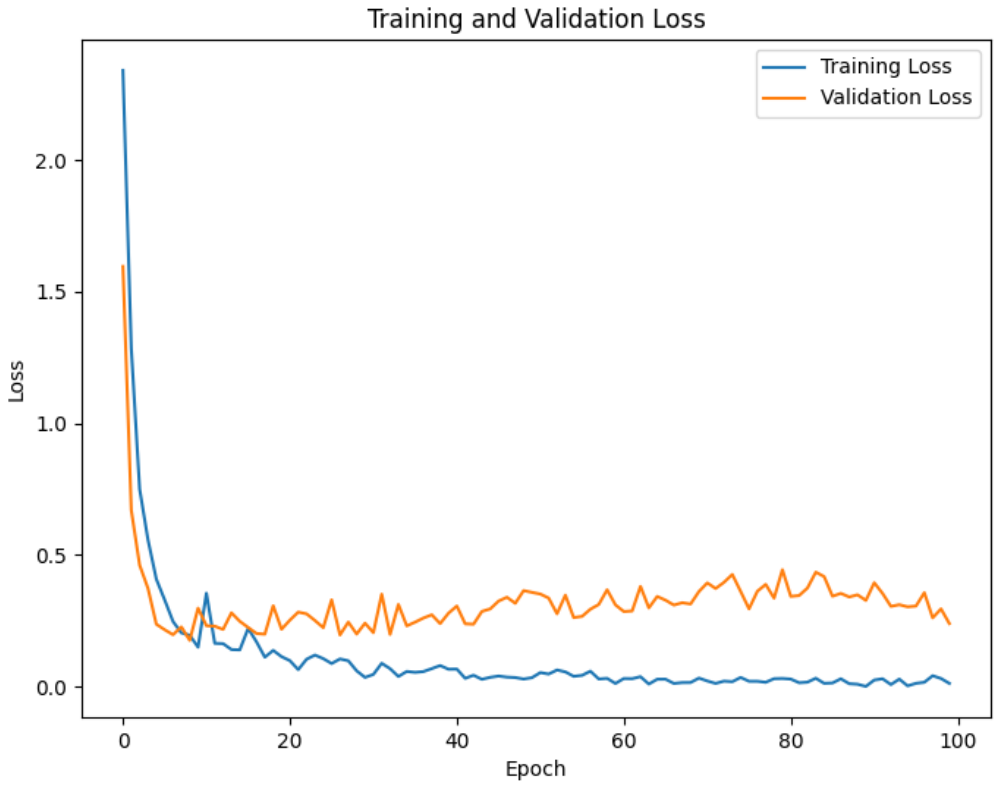
Anexo 19 - Conv2D com *Augmentation* (Tabela 1, linha 13)

- Matriz de Confusão



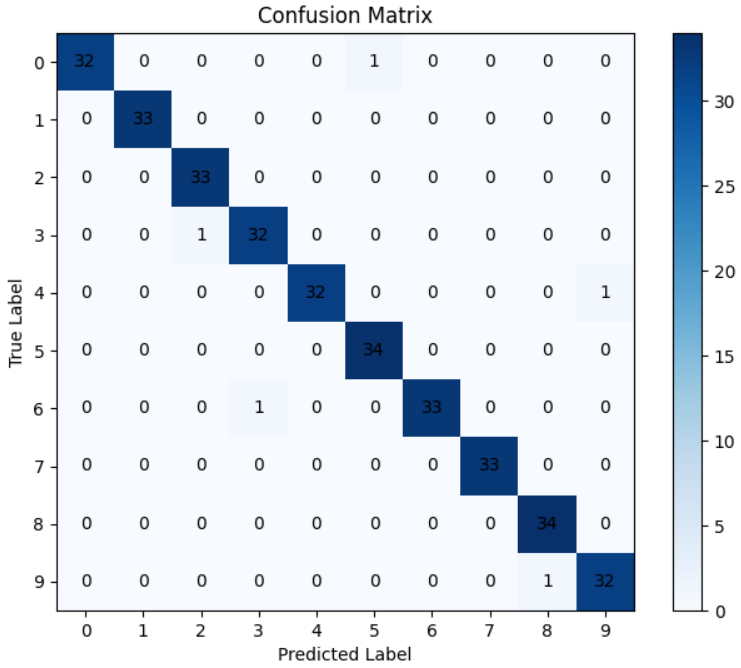
- Gráfico de *Accuracy*

- Gráfico de *Loss*

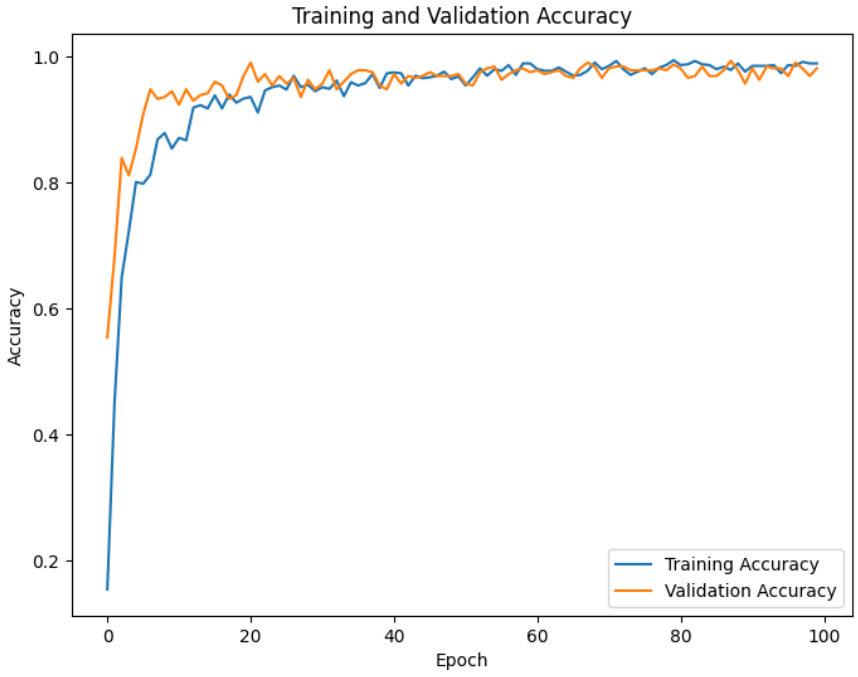


Anexo 20 - Conv2D com *Augmentation* (Tabela 1, linha 14)

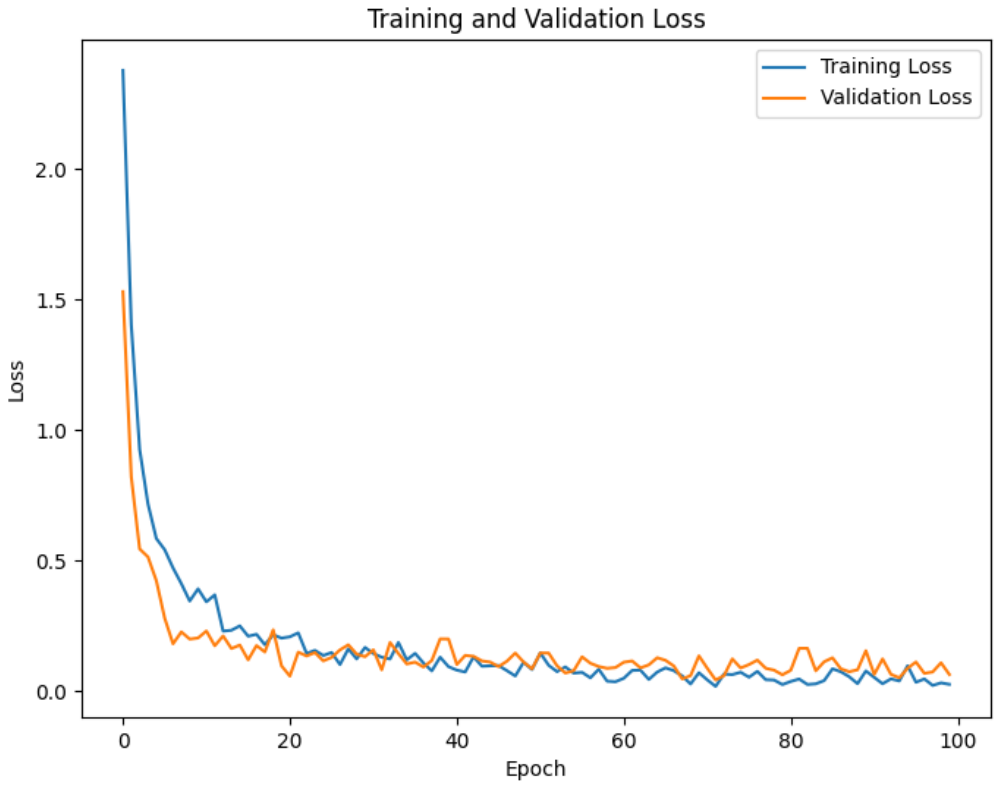
- Matriz de Confusão



- Gráfico de *Accuracy*

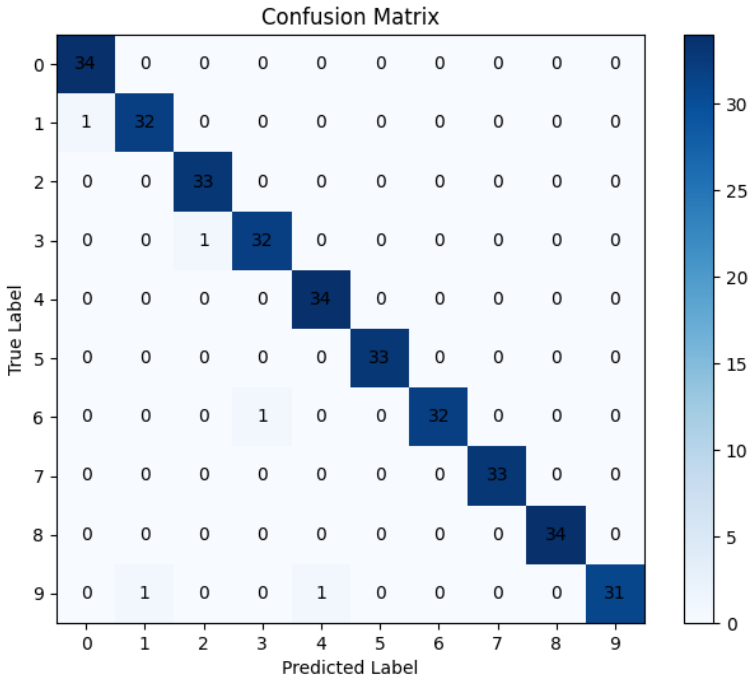


- Gráfico de *Loss*

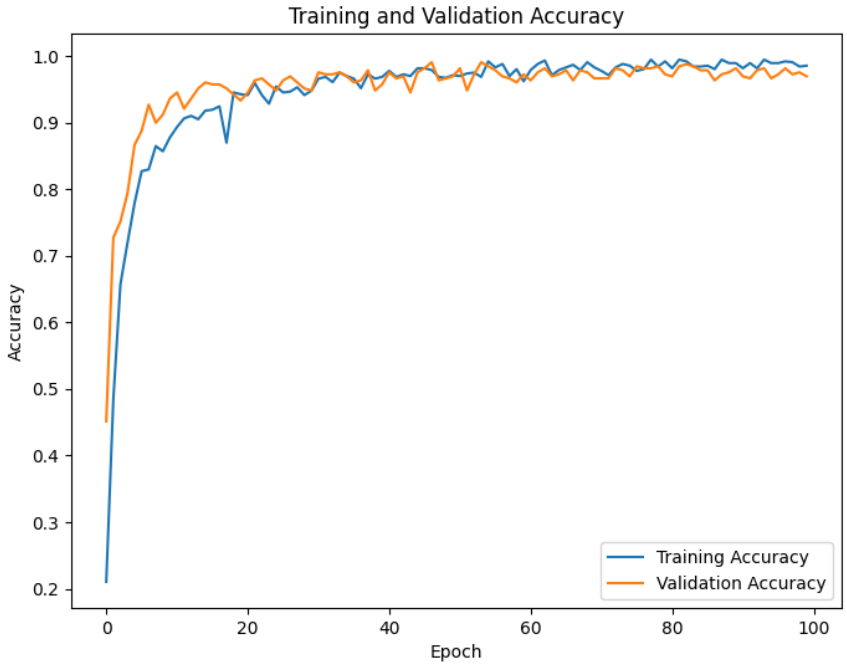


Anexo 21 - Conv2D com *Augmentation* (Tabela 1, linha 15)

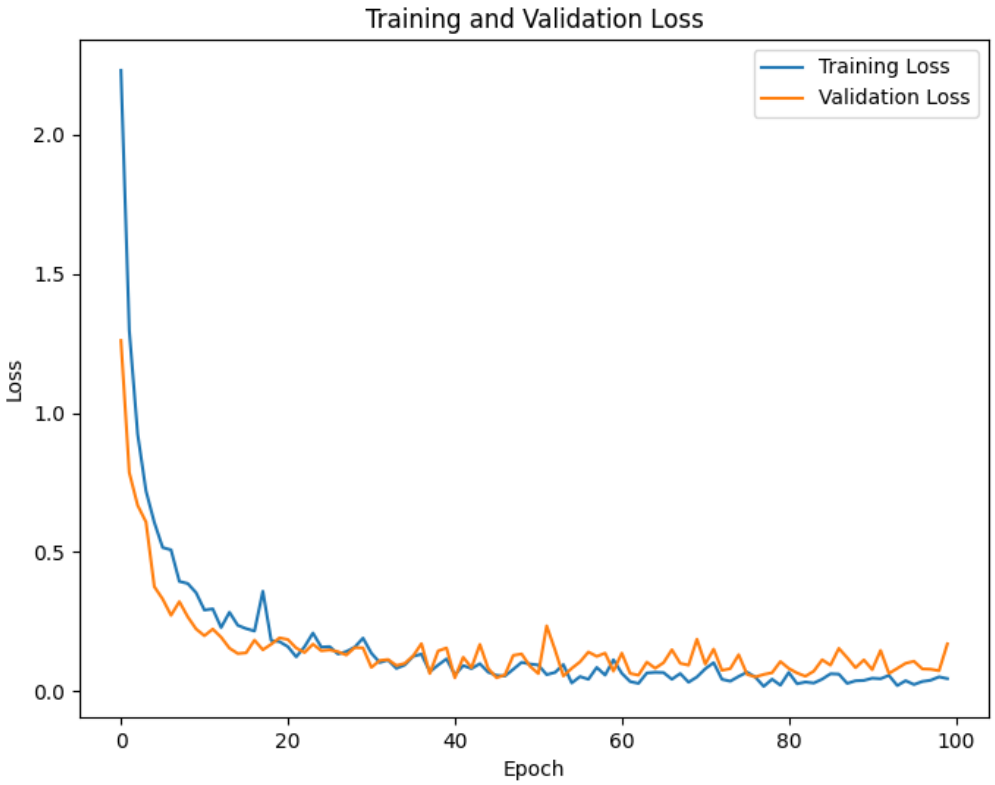
- Matriz de Confusão



- Gráfico de *Accuracy*

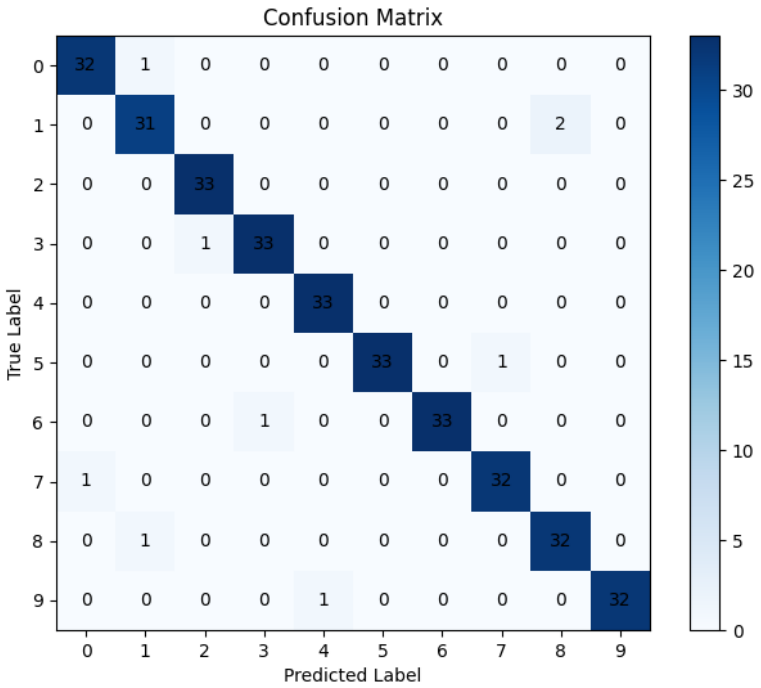


- Gráfico de *Loss*

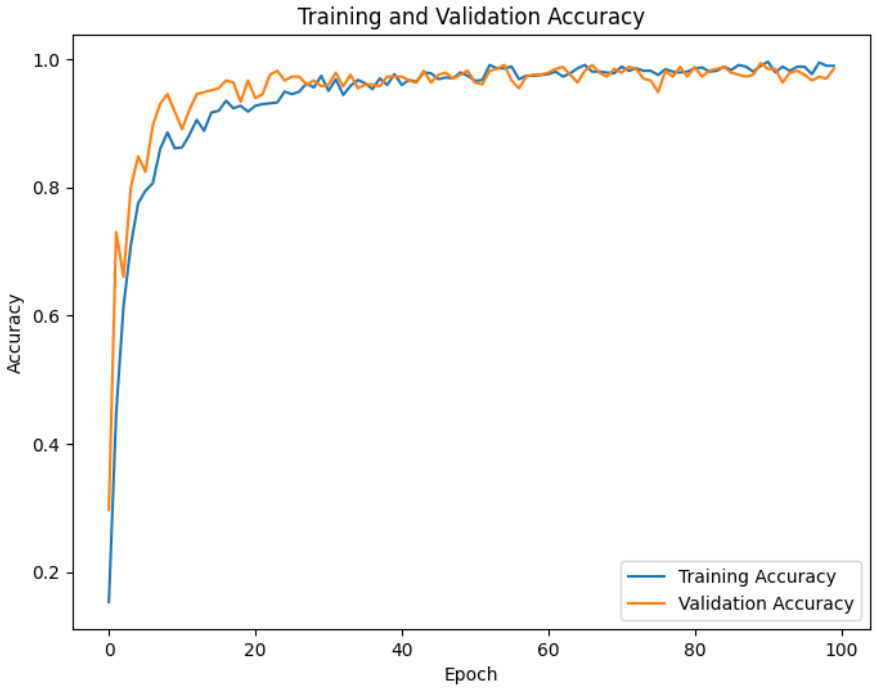


Anexo 22 - Conv2D com *Augmentation* (Tabela 1, linha 16)

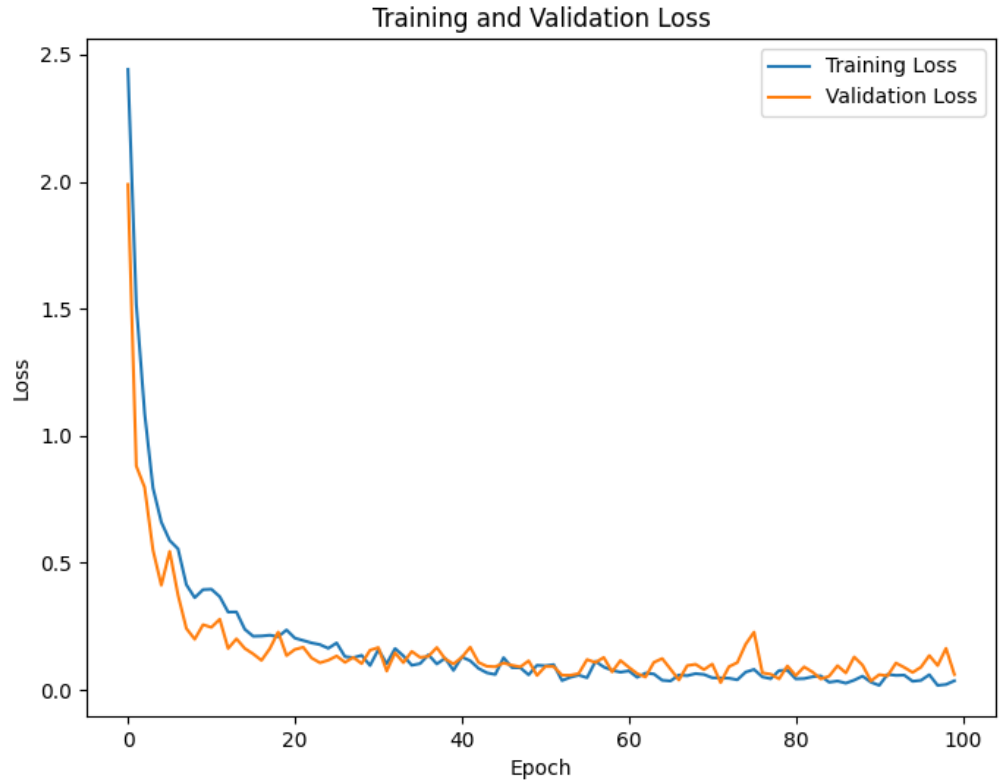
- Matriz de Confusão



- Gráfico de *Accuracy*



- Gráfico de *Loss*



Anexo 23 – Artigo submetido para publicação

Deep Learning for Voice Recognition

Pedro Carreira

Master's in Computer Engineering - Mobile Computing
Escola Superior de Tecnologia e Gestão, Alto do Vieiro,

Morro do Lena, P-2401-951 Leiria,

Apartado 3063,

PORTUGAL

2210491@my.ipleiria.pt

Abstract— **This work aimed to analyze various forms, algorithms and structures of Deep Learning Neural Networks to determine which of the various options would be most viable, accurate and fast to use for Voice Recognition. With that, a voice recognition system was implemented using an innovative method based on Mel spectrograms that are later analyzed by the neural network to train the model to be used by an application. The use of Deep Learning techniques and the innovative approach with spectrograms provided very good results that refer this methodology for further investigation, possibly with datasets that include a larger number of samples from more speakers.**

Keywords— **Deep Learning, CNN, EfficientNet, Resnet50, Spectrograms**

I. INTRODUCTION

In recent decades, we have witnessed remarkable growth and development in the field of voice in the field of speech recognition, an achievement largely due to the Deep Learning revolution. Applications of this technology are increasing in our daily lives, from voice assistants in mobile devices to security and accessibility systems that respond to voice commands. As a result, voice recognition has established itself as a central tool that enables natural and intuitive interactions between humans and machines. In addition to these numerous advantages, development in this field also aims to increasingly blur the impossibility to disabled people perform certain actions in their daily lives, thus enabling more autonomy, whether in leisure activities or in a professional environment.

It is in this context that the need arises to obtain mechanisms for understanding word commands that are accurate and fast, thus reducing the error rate of applications and providing a better user experience.

Within this dynamic and diverse scenario of neural networks, the ones that recently have been increasingly more used and studied are the Convolucional Neural Networks, that are excellent at recognizing visual pattern in images [4]. With that we thought of combining those strengths with visual representations of audio (Mel Spectrograms).

This study aims to achieve three fundamental objectives:

- Extracting Relevant Features from Spectrograms, i.e., extracting characteristics and patterns from the intensity and amplitude graphs of the sound frequencies of the words.
- Development of Deep Learning Models capable of classifying, based on the various voice commands/words.

Considering the diversity of architectures that can be obtained with Deep networks, we decided to implement a model that we had structured, analyze its performance and try to improve it using the Augmentation techniques and, finally, compare it with two pre-trained models, one that implements the ResNet50 architecture and the other that implements the another that implements the EfficientNet architecture.

II. COMMANDS MEL SPECTROGRAMS

Since one of the objectives was to train the different models in our native language (Portuguese), and there's a lack of preexisting datasets in that language, to create the dataset used throughout the tests, it was necessary to develop our own way of creating spectrograms from audio, either as a file or inputs directly into the microphone. To do this, we developed a script which allowed us to generate the spectrograms of the spoken commands/words.

When developing the script, we wanted to make it as simple to use as possible. simple as possible. When starting the script, the user is asked to input the number/command/word to be spoken after that the microphone will then listen for sound signals. If they exceed the set threshold, the recording process will begin. Then, as soon as the script identifies silence for a certain period, the recording is terminated. The word is then recorded in audio format for control purposes and the spectrogram is also generated using a library called Librosa.

It's important to note that all audio was captured at a rate of 44100 Hz and with a chuck size of 1024.

The spectrograms were generated using 256 Mel, a max frequency of 22000 Hz and a hop length of 8.

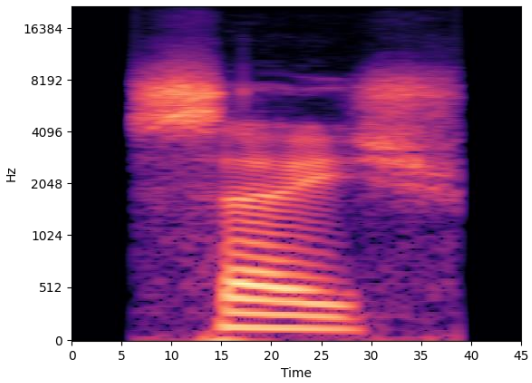


Image 79 - Generated Spectrogram of the word "Seis" (six)

III. Training Methodology

The model was always trained and evaluated using the holdout method, to obtain a general metric of the model's performance. Since the obtained results were almost always above 95% (and reaching 100%) in several metrics, we didn't feel the need to test other methodologies as there was little room for improvement.

As previously mentioned, the focus of this study was to find an innovative solution for speech recognition and to this end we have moved away from Datasets based on languages for which there are already numerous solutions. To this end we decided to train the model with our mother tongue, Portuguese.

Due to the scarcity of Datasets in the same language, it was necessary to create a sample of what would be a complete Dataset, using the script that we talked about in the previous chapter, to record the various spectrograms.

In the end, a set of spectrograms of the numbers 0 to 9, from 7 speakers (6 male and 1 female), containing 1100 images in total.

To implement the holdout evaluation method, 70% of the data was used for training. of the data to carry out training, and the remaining 30% of the data to validate the training process. process. In this case, the division itself was carried out using different directories [5].

IV. Deep neural Networks Used

While developing this work, we gradually improved the complexity of the used neural networks depending on the results obtained in different tests.

Firstly, we created a Deep Neural Network with Conv2D layers that we called Conv2D model, and we tested what were the impacts that resizing the datasets images would have on the performance model.

After choosing the best resizing for our study case, we added data augmentation to our model and used the Conv2D with Augmentation model that we created to run different tests manipulating the dataset images with vertical shifts, rotations, and value centering.

In the end, since our model had shown impressive results, we decided to compare it to already existing pre trained models like ResNet50 and EfficientNet.

- Conv2D

Initially, when developing the Conv2D model, we chose to investigate what effects the size of the images loaded could have during training. To this end, the same model was trained three separate times. First, the dimensions of the images were divided by 2 (320x240), then by 2.5 (256x192) and finally by 3 (213x160). Depending on the results obtained, it was possible to identify the amount of resizing we would apply to the in the remaining models and training sessions.

Table 10 - Results of different image resizing.

	Results				
Resize	mean Accuracy (%)	Mean Loss	Precision (%)	Recall (%)	F1 Score (%)
/2	96.85%	0.1685	96.30%	96.06%	96.05%
/2.5	96.36%	0.2012	95.41%	95.41%	94.86%
/3	96.82%	0.1568	97.03%	96.67%	96.68%

When we divided the image dimensions by 3, the results, despite having a slightly lower average accuracy than the first, also has a lower loss and the best values in the other metrics and the execution time for each epoch was around 15 seconds, lower than the other two resizing.
Given that the metrics of Precision, Recall and F1 were the best of the 3 tests run and the execution time was significantly lower, we opted to use a resizing of /3 in the remaining models.

- Conv2D with Augmentation

Theres a lot of available option when implementing Augmentation to our data but since we are dealing with words and audio, we only used the ones which made sense in the context of this study and proved to help slightly with the performance of the various model.

These parameters include rotation_range = 2, which will perform slight rotations in a range of [-2,2] degrees. We have the parameter height_shift_range = 15, which will

perform transformations on the yy axis within the range [-15,15] pixels.

Finally, we have the Samplewise_center and Samplewise_std_normalization parameters.

The former is based on calculating the average of the values of each individual sample (image, matrix, etc.) and then subtracting this average from all the values in that sample. This has the effect of centering each sample around the average value of zero, which can be beneficial for improving convergence during model training.

The second involves two steps: the first is to center the sample (as explained above) and the second is to divide each value in the sample by the standard deviation of that specific sample. As a result, each sample has a mean of zero and a standard deviation of unit value.

- EfficientNetB7

EfficientNetB7 is a pre-trained neural network model that is part of the EfficientNet family of models. This set of models (B0-B7) was designed with the aim of achieve a balance between performance and computational efficiency, allowing larger and more accurate neural networks without disproportionately increasing the computational resources required [1].

EfficientNet uses a building block called block called MBConv (Convolutional Mobile Inverted Bottleneck), which combines deep convolution operations with separable convolutions in narrower layers, drastically reducing the number of parameters and operations. As we increase the number of B's, we are increasing the number of blocks that make up the model. Therefore, the model used, B7, is the one with the largest number of blocks and, in turn, greater power to analyze the characteristics of the input data at the cost of requiring more computing power for training [2].

- ResNet50

Another of the various pre-trained neural network models used for comparison was ResNet50, which is a convolutional neural network with 50 layers deep, capable of layers, capable of classifying objects into 1000 different categories and trained with the ImageNet dataset.

Due to its performance, it was decided to train the ResNet50 model with the dataset used in this project to have a comparison metric for the other models developed.

V. Results

Since the total number of tests with all possible combinations of the 4 augmentation parameters and the tests with the pre trained models was numerous, it was decided to only include the results that were the most significant, either positive or negative. The results of all tests in this section can be found in Table 2. The tests were all run with the same Dataset and with the same parameters of Augmentation.

Table 11 - Results of the best and worst models with augmentation in comparison with EfficientNet and ResNet50

	Results				
Model	mean Accuracy (%)	Mean Loss	Precision (%)	Recall (%)	F1 Score (%)
Conv2D w/ Augmentation (1)	95.45%	0.2238	93.58%	93.04%	93.07%
Conv2D w/ Augmentation (2)	98.67%	0.0599	99.41%	99.40%	99.40%
Conv2D w/ Augmentation (3)	98.85%	0.0462	97.32%	97.30%	97.30%
ResNet50	100.00%	0.0008	100.00%	100.00 %	100.00 %
EfficientNet	99.88%	0.0026	99.71%	99.70%	99.70%

(1) – Conv2D with Augmentation (Samplewise_center)
(2) – Conv2D with Augmentation (height_shift_range, samplewise_center, samplewise_std_normalization)
(3) – Conv2D with Augmentation (height_shift_range, samplewise_center, samplewise_std_normalization, rotation_range)

VI. Conclusion

The development of this project, the main objective of which was to develop a model for word classification, has allowed us to observe the iterative nature of this type of problems.

In the context of this project, this iterative process focused on the components of data preparation and model development. Often, during the development process, it is possible to observe that the results obtained may be negatively influenced due to data preparation that is out of step with the problem, or simply an error in that preparation.

The relevance of the resources available proved to be a predominant factor in the successful completion of this type of project. The quality and volume of data in a dataset are the foundation for developing good models. These must be analyzed to determine whether the classes they contain are balanced, and if balancing isn't possible, the next steps should take this into account.

The hardware used to carry out this type of project is equally important. A computer with good technical specifications makes it possible to carry out the training processes faster,

which allows you to add more iterations to the process, if necessary, to achieve better results.

For the evaluation metrics selected, the models developed in this project show very encouraging results between 95% and 100%, which shows the viability of the use of Mel spectrograms as a mean of word recognition and model training.

In addition, although some validation loss values are higher than 0.2, which in general is a bad result, there were also models that managed to achieve values lower than this, with one of them approaching zero to 3 decimal places.

At the final stage of this study, it was not possible to carry out an evaluation of the application using samples from another dataset. This evaluation would be relevant to obtain an independent perspective on the performance of the models.

REFERENCES

[1] Sarkar, A. (2021, 8 de Maio). Understanding EfficientNet — The most powerful CNN architecture. Medium. https://medium.com/mlearning-ai/understanding-efficientnetthemost-powerful-cnn-architecture-eaeb40386fad.

[2] Tan, M., & Le, Q. v. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. https://doi.org/10.48550/arXiv.1905.11946

[3] B. Wang, “Loss Functions in Machine Learning,” 13 Janeiro 2021. [Online]. Available: https://medium.com/swlh/cross-entropy-loss-in-pytorchc010faf97bab.

[4] GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. Deep learning. MIT press, 2016.

[5] S. Chauhan, “Model Selection for Machine Learning,” 1 Junho 2023. [Online]. Available: https://www.scholarhat.com/tutorial/machinelearning/modelselection-formachine-learning.