Estudo de modelos de deep learning para classificação de espécies de plantas por imagem

Mateus Fonseca Piris

São Carlos

2023

Estudo de modelos de deep learning para classificação de espécies de plantas por imagem

Mateus Fonseca Piris

Orientador: Prof. Dr. Fernando Santos Osório

Co-orientador: Prof. Dr. Diego Renan Bruno

Monografia referente ao projeto de conclusão de curso dentro do escopo da disciplina SSC0670 – Projeto de Formatura I do Departamento de Sistemas de Computação do Instituto de Ciências Matemáticas e de Computação – ICMC-USP para obtenção do título de Engenheiro de Computação.

São Carlos

2023

Agradecimentos

Quero expressar minha gratidão a todos que tornaram esse trabalho possível. Primeiramente aos meus pais, Denise Angélica Fonseca e Eduardo Lopes Piris, agradeço por todo o amor e suporte dado ao longo da minha vida, que sem eles esse trabalho não teria acontecido. Gratidão aos meus irmãos, Guilherme Fonseca Piris e Lavínia de Lima Piris, pelo amor fraterno. Gratidão à Caroline Santos Corrêia pelo amor e companheirismo, e também pela paciência por me ouvir falar tanto desta monografia nas semanas finais de escrita. Também expresso minha gratidão aos professores Fernando Santos Osório e Diego Renan Bruno por toparem esse trabalho, me orientando e auxiliando.

Resumo

O principal objetivo deste trabalho é realizar um estudo abrangente e comparativo entre diferentes modelos de redes neurais artificiais para a tarefa específica de classificação de imagens, utilizando fotografias de plantas como aplicação. O diferencial desta pesquisa reside no seu enfoque em uma questão atualmente em destaque: a comparação entre modelos convolucionais e modelos baseados em transformers. Por mais de uma década, os modelos convolucionais têm sido a base da visão computacional, mas os transformers estão ganhando crescente reconhecimento e espaço nesse campo. Nesse contexto, este estudo visa analisar e explorar as vantagens e desafios de ambos os tipos de modelos, com o intuito de contribuir para o avanço da visão computacional e sua aplicação na área específica de classificação de imagens de plantas.

Contents

List of Figures vii

List of Tables ix

Abbreviations x

1 Introdução 1
- 1.1 Motivação . . . 1
- 1.2 Objetivos . . . 2

2 Revisão Bibliográfica 3
- 2.1 Redes Neurais Artificiais . . . 3
  - 2.1.1 Neurônio . . . 4
  - 2.1.2 Camada de neurônios . . . 5
  - 2.1.3 *Multi Layer Perceptron* . . . 6
  - 2.1.4 Conexões Residuais . . . 7
  - 2.1.5 Redes Neurais Convolucionais . . . 8
  - 2.1.6 *Vision Tranformer* . . . 12
- 2.2 Treinamento de Redes Neurais . . . 18
  - 2.2.1 Transfer learning . . . 19
  - 2.2.2 Função de perda . . . 19
  - 2.2.3 Otimizadores . . . 20
  - 2.2.4 Overfitting . . . 21
- 2.3 Técnicas de avaliação . . . 24
  - 2.3.1 Acurácia . . . 24
  - 2.3.2 Recall, Precisão e F1-Score . . . 24
  - 2.3.3 Matriz de confusão . . . 25

3 Desenvolvimento do trabalho 26
- 3.1 Arquiteturas utilizadas . . . 26
- 3.2 Bases de dados . . . 29
- 3.3 Data augmentation . . . 30
- 3.4 Ambiente de execução . . . 31
- 3.5 Experimentos e Resultados . . . 32

3.5.1 Dataset Autoral 32
3.5.2 Dataset Oxford 102 40

4 Resultados e discussão **44**
4.1 Dataset Autoral 44
4.2 Dataset Oxford102 45
4.3 Discussão 47

5 Conclusão e trabalhos futuros **49**

Bibliography **51**

List of Figures

2.1 Funções de ativação mais usadas 4
2.2 Representação gráfica de um neurônio 5
2.3 Camada de neurônios 6
2.4 Representação gráfica uma rede densamente conectada 7
2.5 Representação gráfica um bloco residual 8
2.6 Representação da VGG16. Fonte: Research Gate[1] 9
2.7 Representação gráfica da operação de convolução 9
2.8 Representação gráfica da operação de max pooling 10
2.9 Modulo Inception 11
2.10 Representação gráfica do modelo Transformer 12
2.11 Representação gráfica do modelo ViT 13
2.12 Representação gráfica do transformer encoder 14
2.13 Representações do resultado da operação de atenção com as palavras ”it” e ”because” em uma frase 15
2.14 Mapa de atenção 15
2.15 Representação gráfica do Multi Head Attention 16
2.16 Representação gráfica da saída softmax 20
2.17 Representação do dropout 22
2.18 Early Stopping 23
2.19 Matriz de Confusão 25

3.1 Representação gráfica do modelo InceptionResNetV2 27
3.2 Exemplo de imagens do dataset autoral 29
3.3 Exemplo de imagens do dataset Oxford 102 30
3.4 Random Flip 31
3.5 Random Zoom 31
3.6 Evolução das curvas de loss e acurácia do modelo VGG16 34
3.7 Evolução das curvas de loss e acurácia do modelo InceptionResNetV2 . . . 34
3.8 Evolução das curvas de loss e acurácia do modelo EfficientNetB0 35
3.9 Evolução das curvas de loss e acurácia do modelo Vision Tranformer 35
3.10 Evolução das curvas de loss e acurácia do modelo Swin Tranformer 36
3.11 Matrizes de confusão do modelo VGG16 para os conjuntos de dados de validação e teste 36
3.12 Matrizes de confusão do modelo InceptionResNetV2 para os conjuntos de dados de validação e teste 37

3.13 Matrizes de confusão do modelo EfficientNetB0 para os conjuntos de dados de validação e teste . . . 37
3.14 Matrizes de confusão do modelo Vision Tranformer para os conjuntos de dados de validação e teste . . . 38
3.15 Matrizes de confusão do modelo Swin Tranformer para os conjuntos de dados de validação e teste . . . 38
3.16 Evolução das curvas de loss e acurácia do modelo VGG16 . . . 41
3.17 Evolução das curvas de loss e acurácia do modelo InceptionResNetV2 . . . 41
3.18 Evolução das curvas de loss e acurácia do modelo EfficientNetB4 . . . 42
3.19 Evolução das curvas de loss e acurácia do modelo ViT-B16 . . . 42

4.1 Aplicação de filtros passa alta e passa baixa em uma imagem . . . 47
4.2 Representação gráfica do modelo CoCa . . . 48
4.3 Representação do modelo CoAtNet . . . 48

List of Tables

1.1 Os melhores modelos na tarefa de classificação do dataset ImageNet 2

3.1 Tamanho dos modelos utilizados nos experimentos com dataset autoral . . . 33
3.2 Resultado obtidos dos modelos treinados com o conjunto de validação . . . 39
3.3 Resultado obtidos dos modelos treinados com o conjunto de teste 39
3.4 Tamanho dos modelos utilizados nos experimentos com o dataset Oxford102 40
3.5 Resultado obtidos dos modelos 43

4.1 Época em que cada modelo atinge 80% de acurácia com dataset de validação 45
4.2 Ranqueamento dos modelos de acordo com a acurácia 45
4.3 Épocas em que cada modelo atinge 30%, 60% e 80% de acurácia com dataset de validação 46

Abbreviations

USP	Universidade de São Paulo
CNN	Rede Neural Convolucional
MLP	Multi Layer Perceptron
RNA	Redes Neurais Artificiais
ViT	Vision Transformer

Chapter 1

Introdução

1.1 Motivação

A motivação deste trabalho parte do incrível sucesso dos modelos de redes neurais artificiais (RNA) no campo de visão computacional. Tais modelos vêm obtendo ótimos resultados em diversas tarefas, como classificação de imagens, detecção de objetos e segmentação semântica, em aplicações na indústria, no campo, no varejo, em veículos autônomos e em produtos da web, como pesquisa por imagem. Este trabalho por sua vez foca na classificação de imagens de plantas por espécie.

A tarefa de classificação de espécies de plantas possui relevância no em alguns contextos, como:

- Conservação da biodiversidade: A classificação de imagens de plantas pode ser usada para identificar e catalogar espécies vegetais em áreas de conservação, o que é importante para entender a diversidade da flora e monitorar mudanças ao longo do tempo.
- Botânica: A classificação de imagens de plantas é uma ferramenta importante para os botânicos, que podem usar as imagens para identificar espécies desconhecidas ou raras, bem como para estudar a evolução e a genética das plantas.
- Agricultura: A classificação de imagens de plantas pode ser utilizada na agricultura para identificar pragas, doenças e anomalias nas plantas, permitindo que os agricultores tomem medidas preventivas e corretivas de maneira mais eficiente e efetiva.

Outra motivação deste trabalho consiste no debate que está em alta a respeito da comparação entre modelos baseados em convolução e modelos baseados em transformers. Para demostrar brevemente este debate segue abaixo a tabela 1.1 retirada do site Papers With Code [2], em que temos alguns modelos ranqueados de acordo com a sua acurácia na tarefa de classificação do dataset ImageNet. As três primeiras linhas mostram os 3 modelos com a maios acurácia hoje, da 4ª linha em diante temos os modelos que estiveram como Top 1 nesta tarefa, podemos notar que até 2021 os modelos que obtinham melhor performance eram modelos puramente convolucionais, e a partir de 2021 modelos puramente transformer ou combinação de transformer com convolução passaram as ser os melhor ranqueados.

TABLE 1.1: Os melhores modelos na tarefa de classificação do dataset ImageNet

Rank	Modelo	Acurácia	Ano	Tipo
1	BASIC-L	91.1%	2023	Conv+Tansformer
2	CoCa	91.0%	2022	Tansformer
3	Model Soups	90.98%	2022	Conv+Tansformer
10	ViT-G/14	90.45%	2021	Tansformer
12	Meta Pseudo Labels	90.2%	2020	Conv
45	FixEfficientNet-L2	88.5%	2020	Conv
49	NoisyStudent	88.4%	2020	Conv
77	BiT-L	87.54%	2019	Conv
586	Inception ResNet V2	80.1%	2016	Conv
679	ResNet-152	78.57%	2016	Conv
813	VGG16	74.4%	2014	Conv
869	AlexNet	63.3%	2012	Conv

1.2 Objetivos

Este trabalho tem como objetivos fazer uma revisão bibliográfica a respeito de modelos de classificação de imagens de plantas, implementar diversos modelos e realizar o estudo comparativo entre estes. Outro objetivo deste trabalho é aprofundar o estudo no debate que está em alta a respeito de redes convolucionais versus transformers.

Chapter 2

Revisão Bibliográfica

Classificação de imagens é um tema básico no campo de visão computacional, então há muita pesquisa feita tanto na academia quanto na indústria referente a este tema. Como é um tema básico os modelos de classificação são muitas vezes usados em outros tipos de aplicações, por exemplo, na tarefa de detecção de objetos os modelos YOLO[3] e Faster R-CNN[4] podem usar como base a arquitetura VGG16[5] ou ResNet50[6], já na tarefa de segmentação semântica o modelo U-Net[7] também pode usar como base as arquiteturas VGG16 e ResNet50.

Neste capítulo apresenta-se uma breve introdução teórica sobre deep learning, diferentes tipos de arquiteturas, seus blocos constituintes, técnicas de treinamento e de avaliação de modelos.

2.1 Redes Neurais Artificiais

As RNA são modelos matemáticos de aprendizagem de máquina, que basicamente recebem uma entrada e retornam uma saída, por exemplo, podemos alimentar uma RNA com dados do histórico financeiro de uma pessoa e este modelo pode retornar se esta pessoa pode receber crédito financeiro e quanto crédito pode receber; para este modelo conseguir aprender como dar crédito de maneira adequada ele deve ser treinado, de maneira que são apresentados a ele exemplos de dados de pessoas e quanto de crédito financeiro essas pessoas podem receber.

A inspiração para as RNAs está relacionado ao funcionamento biológico do cérebro humano e seus mecanismos de aprendizado. A modelagem matemática claramente difere

do funcionamento biológico, porém muitos resultados relevantes já foram obtidos, como reconhecimento de imagens, reconhecimento e sintetização de voz, interpretação de texto (processamento de linguagem natural), navegação, controle de movimento, criação de arte a partir de texto etc.

2.1.1 Neurônio

O neurônio é a unidade básica de uma RNA. Cada neurônio aplica uma função de ativação (g) ao somatório (Σ) ponderado de diversas entradas (peso w e entrada x) acrescidos de um valor denominado viés ou bias (b). A função de ativação é uma função matemática, sendo as mais usadas a sigmóide, tangente hiperbólica, ReLU e LeakyReLU. A figura 2.1 ilustra essas funções, essa imagem foi retirada do site Research Gate[8].

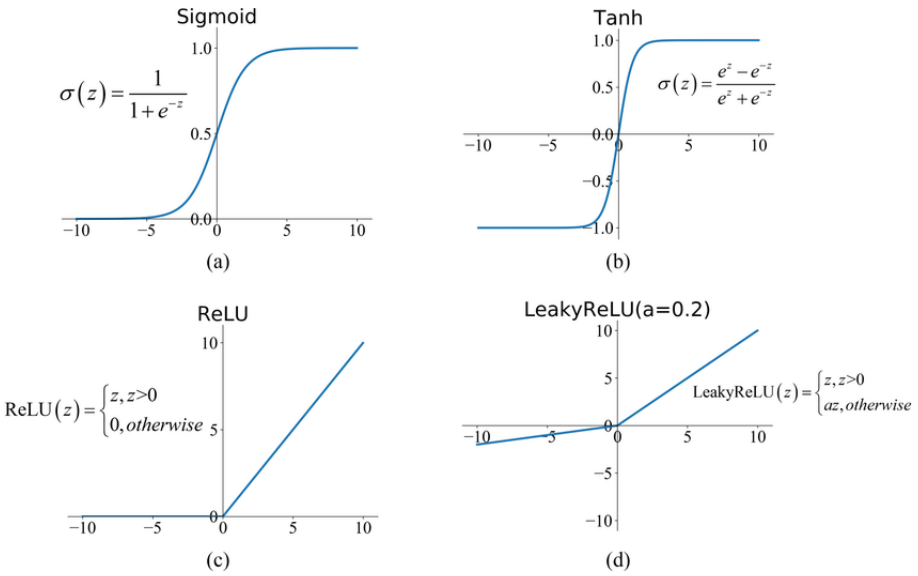


FIGURE 2.1: Funções de ativação mais usadas

Esta é a equação que modela cada neurônio: $a = g(w^T x+b)$. Cabe observar que w e x estão na forma matricial (por isso o símbolo T de matriz transposta) e a é o resultado numérico da equação, ou seja, a é o output do neurônio. Na imagem abaixo há uma representação gráfica de um neurônio, cuja saída é igual ao resultado da função de ativação que recebe o somatório das entradas ponderadas com os pesos de cada sinapse somadas com o bias do neurônio, há três entrada e uma saída.

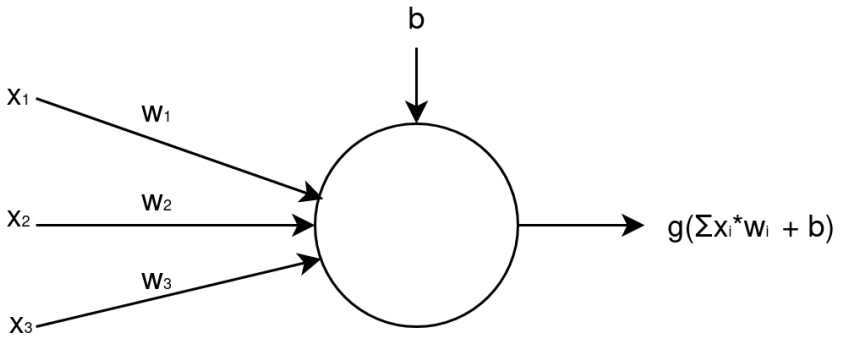


FIGURE 2.2: Representação gráfica de um neurônio

2.1.2 Camada de neurônios

A unidade básica de uma RNA, o neurônio, pode ser agrupada em camadas com outros nerônios. Então diversos neurônios recebem a mesma entrada, e cada neurônio camada gera resultado diferente para a mesma entrada (isso é possível por que cada neurônio possui um conjunto próprio de pesos w e bias b).

Quanto mais camadas um modelo possui maior é o nível de complexidade dos problemas que este modelo consegue resolver, pois as camadas mais profundas aprendem características de mais alto nível do problema. Uma rede neural de uma única camada pode aprender funções lineares, enquanto uma rede neural de duas camadas, com uma camada oculta, pode aprender funções mais complexas, como funções quadráticas. A camada oculta adiciona uma etapa de processamento não linear, permitindo que a rede aprenda relações mais sofisticadas entre as variáveis de entrada e a saída desejada. Redes neurais com mais camadas ocultas podem aprender funções ainda mais complexas.

Na imagem abaixo há uma representação gráfica de uma camada de neurônios, nota-se que todos os neurônios são alimentados com o mesmo conjunto de dados ($x_1, x_2 e x_3$), cada neurônio possui o um conjunto de pesos e bias($w_{x1}, w_{x2}, w_{x3}, b_x$). Há três entradas e três saída.

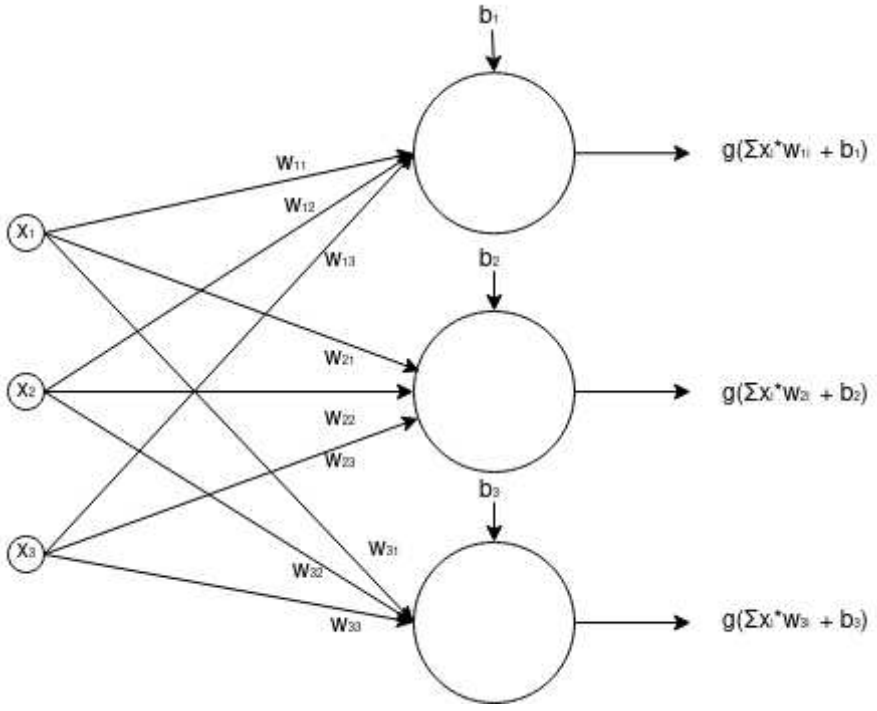


FIGURE 2.3: Camada de neurônios

2.1.3 *Multi Layer Perceptron*

O Multi-Layer Perceptron[9] (MLP) é um tipo de RNA feedforward, composta por múltiplas camadas de neurônios, cada uma conectada a uma camada subsequente através de pesos sinápticos ajustáveis. Essa arquitetura de rede é capaz de mapear relações complexas entre entradas e saídas, tornando-se uma ferramenta valiosa em diversas áreas da ciência e da tecnologia.

O MLP é formado por camadas de neurônios, incluindo uma camada de entrada, uma ou mais camadas intermediárias, também conhecidas como camadas ocultas, e uma camada de saída. Os neurônios em cada camada intermediária aplicam uma função de ativação não-linear para produzir uma saída, que é passada para a próxima camada através dos pesos sinápticos.

Os pesos sinápticos são ajustados durante o treinamento da rede para minimizar a diferença entre as saídas esperadas e as saídas reais da rede. Isso é feito utilizando um algoritmo de aprendizado supervisionado, como o *backpropagation*[10], que atualiza os pesos sinápticos

em cada ciclo de treinamento. Esse algoritmo usa a técnica de otimização gradiente descendente para ajustar os pesos sinápticos de forma a minimizar o erro da rede em relação aos dados de treinamento

A figura abaixo 2.4 representa uma rede MPL, que nada mais é do que uma série de camadas de neurônios empilhadas. Na imagem em questão há três camadas densas, com três neurônios cada uma, e um neurônio de saída.

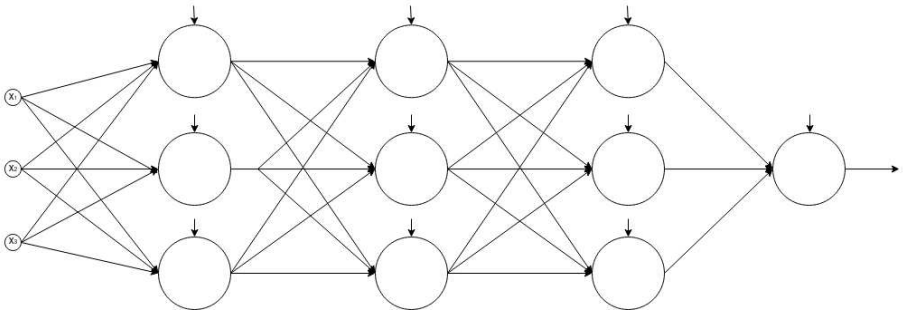


FIGURE 2.4: Representação gráfica uma rede densamente conectada

2.1.4 Conexões Residuais

Um conceito importante em deep learning é o de conexões residuais, pois permite que RNA sejam mais profundas. Temos como exemplo que durante um tempo o modelo mais performático era as VGG19, com 19 camadas, após o desenvolvimento das conexões residuais o modelo ResNet152, com 152 camadas, superou a VGG19.

O problema que redes sem conexões residuais enfrentam ao tentar aprofundar o número de camadas é o "*vanishing gradient*", que é um problema em que as derivadas ficam muito pequenas ao se propagarem das camadas mais profundas para as mais superficiais. Isso afeta a atualização dos pesos das camadas profundas, tornando o treinamento lento e menos eficaz. Para resolver esse problema, no artigo "Deep Residual Learning for Image Recognition"[11] é apresentada a técnica de redes residuais, que resolvem esse problema introduzindo conexões de atalho, permitindo que as informações originais fluam diretamente para as camadas mais profundas. Essas conexões facilitam a propagação do gradiente e possibilitam o treinamento eficiente de redes profundas.

Na figura 2.5 temos a representação de um bloco residual, nele temos uma entrada com um única variável, duas camadas totalmente conectadas e por fim o somatório do resultado da 2ª camada conectada com o valor da entrada.

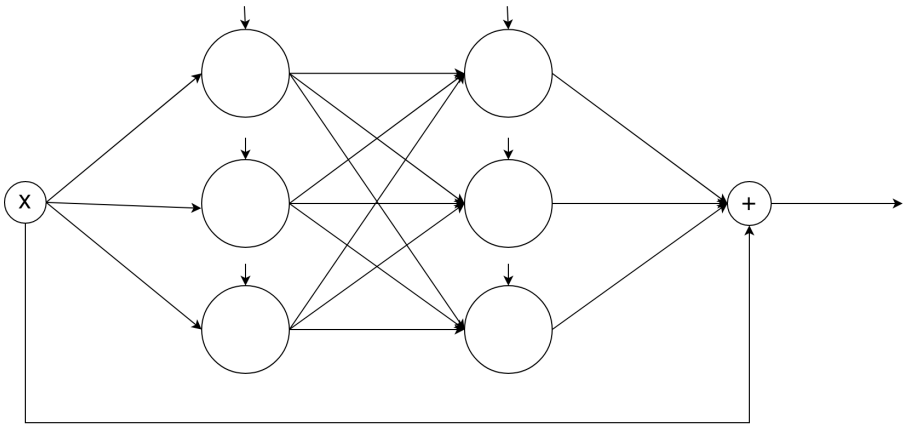


FIGURE 2.5: Representação gráfica um bloco residual

2.1.5 Redes Neurais Convolucionais

As redes neurais convolucionais (CNNs) são uma arquitetura poderosa para o processamento de imagens e visão computacional em geral. Com sua capacidade de extrair automaticamente características importantes das imagens, as CNNs são essenciais para muitas aplicações em áreas como reconhecimento facial, veículos autônomos e diagnóstico médico.

Uma CNN é composta por várias camadas, incluindo camadas de convolução, pooling e camadas totalmente conectadas. A camada de convolução é a camada central da CNN e é responsável por extrair características da imagem de entrada, pela sua capacidade de capturar informações espaciais e padrões locais em imagens, como detecção de cor, textura, formas geométricas etc.

A título de exemplo usaremos o modelo VGG16[5] para tratarmos de modelos convolucionais, visto que é simples e consegue atingir boa acurácia de 74.4% no dataset ImageNet[12].

Convolução é a operação central da CNN e é responsável por possibilitar a extração de características da imagem de entrada. Essa camada utiliza filtros (kernel) que percorrem a imagem, aplicando operações de convolução em cada região. O resultado da convolução é uma representação reduzida da imagem que destaca os recursos mais importantes, criando um mapa de características (feature map).

A VGG16 possui 13 camadas convolucionais, cada uma seguida por uma camada de ativação ReLU. Cada camada convolucional aplica um conjunto de filtros de convolução 3*3 a uma imagem de entrada para extrair recursos cada vez mais complexos. As camadas

convolucionais são agrupadas em blocos, com 2 a 3 camadas em cada bloco. Cada uma das duas primeiras camadas convolucionais possuem 64 filtros, as duas seguintes possuem 128 filtros, as três seguintes 256 e as seis seguintes 512. Podemos ver essa representação na figura2.6.

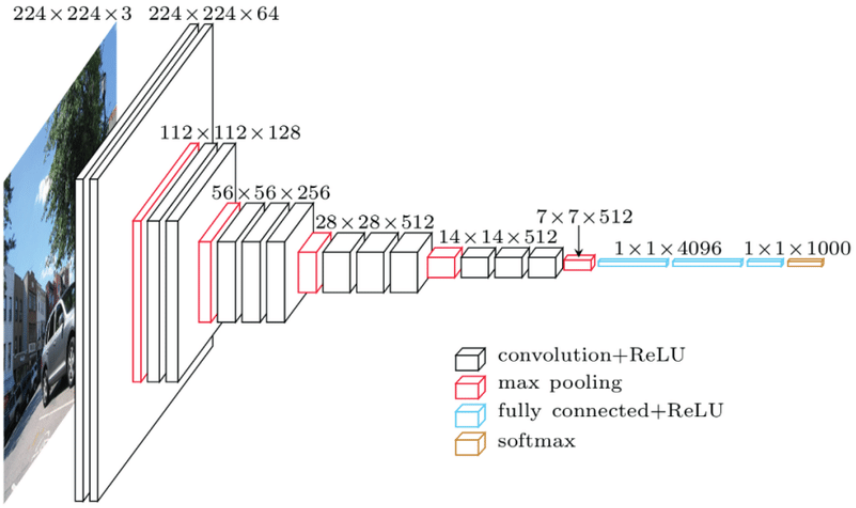


FIGURE 2.6: Representação da VGG16. Fonte: Research Gate[1]

A figura abaixo 2.7 é uma ilustração de como funciona a operação de convolução, à esquerda há a imagem de entrada, no meio o kernel 3*3 (que detecta linhas verticais) e à direita o resultado da convolução. O canto superior à esquerda, pintado de verde passa pelo processo de convolução com o kernel, em que cada elemento do quadrado verde é multiplicado pelo elemento respectivo do kernel, e o resultado das 9 multiplicações são somados, obtendo o resultado em verde na matriz à direita. Como a matriz resultante possui 25 pixels podemos afirmar que o kernel foi aplicado 25 vezes à imagem de entrada.

0	0	1	2	1	0	0
0	0	1	2	1	0	0
0	0	1	2	1	0	0
0	0	1	2	1	0	0
0	0	1	2	1	0	0
0	0	1	2	1	0	0
0	0	1	2	1	0	0

*

1	0	-1
1	0	-1
1	0	-1

=

-1	-2	0	2	1
-1	-2	0	2	1
-1	-2	0	2	1
-1	-2	0	2	1
-1	-2	0	2	1

FIGURE 2.7: Representação gráfica da operação de convolução

Cada filtro na camada de convolução é capaz de detectar um determinado padrão ou característica na imagem de entrada, como bordas, linhas, texturas ou formas específicas.

À medida que a camada de convolução avança, os filtros se tornam mais complexos e são capazes de detectar características mais abstratas, como contorno específico de uma flor ou a textura de uma pétala.

Pooling ou agrupamento é uma operação usada para reduzir o tamanho do mapa de características e diminuir a complexidade computacional, mantendo as informações mais importantes. O processo de pooling consiste em dividir a imagem em regiões não sobrepostas e, em seguida, selecionar o valor mais relevante de cada região para formar o novo mapa de características reduzido, condensando os dados calculados previamente. Na figura 2.8 temos o exemplo de max-pooling, que seleciona o valor máximo de um agrupamento.

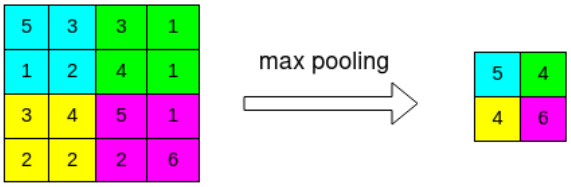


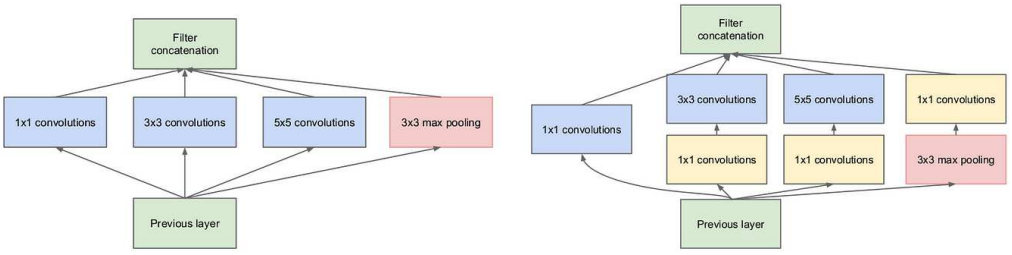
FIGURE 2.8: Representação gráfica da operação de max pooling

A VGG16 utiliza camadas de agrupamento max-pooling após cada bloco de camadas convolucionais. Essas camadas reduzem a dimensão da saída da camada convolucional, mantendo os recursos mais importantes. Por exemplo, aplicando o max-pooling a uma imagem de dimensões (224,224,3) podemos obter uma imagem de (112,112,3) muito similar à imagem original.

Após a última camada de pooling da VGG16, as características extraídas alimentam um bloco MLP, que têm a tarefa de realizar a classificação, mapeando as características extraídas para as classes de saída desejadas. A VGG16 possui 3 camadas totalmente conectadas após as camadas convolucionais e de agrupamento, que são responsáveis pela classificação final da imagem. Cada camada totalmente conectada possui 4096 neurônios, seguidos por uma camada de ativação ReLU, exceto pela última camada, que utiliza a ativação softmax para gerar as probabilidades de classificação.

Inception é um modelo de CNN introduzido em 2014 pelo Google Research no artigo "Going deeper with convolutions"[13]. Este modelo tem como base módulo Inception, representado na figura 2.9, que consiste em várias operações de convolução em paralelo com diferentes tamanhos de filtro (kernels com tamanho de 1x1, 3x3 e 5x5), cujos resultados são concatenados no final. Essas convoluções em paralelo permitem que o modelo extraia informações de diferentes escalas e resoluções, capturando tanto detalhes finos quanto

características globais. Além disso, o modelo Inception utiliza convoluções 1x1 para reduzir a carga computacional, permitindo que a rede seja mais eficiente em termos de memória e operações.



(a) Inception module, naïve version (b) Inception module with dimension reductions

FIGURE 2.9: Modulo Inception

2.1.6 *Vision Tranformer*

A arquitetura ViT (Vision Transformer) é uma abordagem recente em visão computacional que utiliza Transformers para extrair informações de imagens. A arquitetura de Transformers foi inicialmente desenvolvida para processamento de linguagem natural (NLP), para a tarefa de tradução. Hoje ela é utilizada em diversas aplicações, como o ChatGPT e Google Tradutor, sendo a base dos modelos GPT e BERT. Essa arquitetura foi apresentada no artigo "Attention is All You Need"[14], sendo muito inovador ao propor uma nova arquitetura, baseada em mecanismos de atenção, diferentemente das aborgadens baseadas em redes recorrentes e convolucionais. Essa arquitetura se mostrou extremamente eficaz em capturar relações de dependência entre palavras em textos longos, que é uma limitação em modelos do tipo LSTM (Long Short-Term Memory)[15] que eram até então os modelos principais no campo de NLP. A figura 2.10 ilustra a arquitetura do modelo transformer (com encoder, bloco Nx à esquerda, e decoder, bloco Nx à direita), ela foi retirada do artigo acima citado[14].

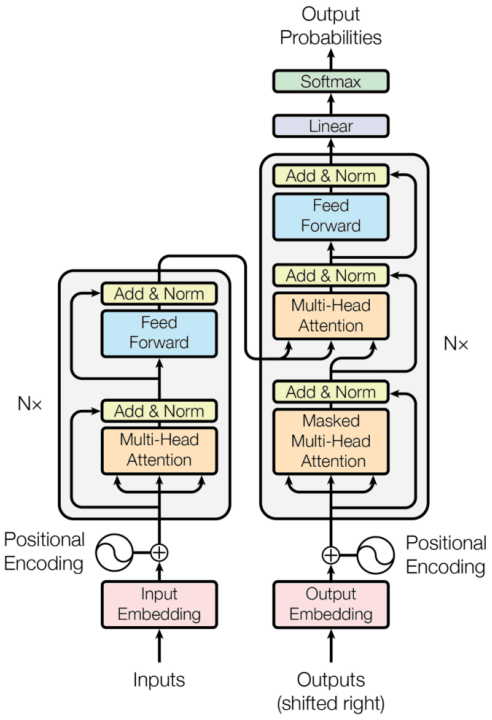


FIGURE 2.10: Representação gráfica do modelo Transformer

A arquitetura de Transformer é muito boa em extrair significados de frases longas, esses

significados residem na relação entre as palavras, que por sua natureza são relações complexas, visto que a linguagem humana é complexa. Assim, foi natural o movimento de cientistas tentarem aplicar modelos baseados em Transformer em problemas de visão computacional. Em 2019 Cordonnier[16] propõe a arquitetura Vision Transformer (ViT), e em 2020, o conhecido artigo "An image is worth 16x16 words"[17] apresenta estudos mais extensos e profundos sobre o modelo. A figura 2.11 ilustra esse modelo, e foi extraída do artigo acima citado[17].

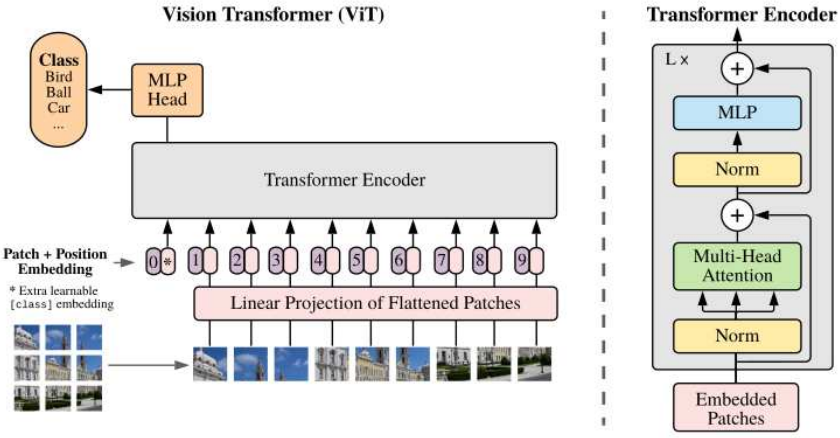


FIGURE 2.11: Representação gráfica do modelo ViT

A ideia central da arquitetura ViT é tratar a imagem como se fosse uma frase, dividi-la em subamostras e tratar cada amostra como se fossem palavras. A camada codificadora (transformer encoder) é construída usando o mecanismo de atenção (self attention) e extrai, para cada subamostra, a relação entre essa subamostra e todas as demais subamostras; um modelo de ViT pode possuir diversas camadas de Tranformer Encoder. Após a última camada de codificação é usado um bloco Multi Layer Perceptron que, a partir dos dados extraídos pela fase de codificação, classifica a imagem de entrada.

Desde sua primeira publicação, a arquitetura ViT vem sendo explorada em outras tarefas de visão computacional, como detecção de objetos e segmentação semântica. Por exemplo, a DETR é uma arquitetura que utiliza transformers para a tarefa de detecção de objetos em imagens, enquanto a DeiT é uma arquitetura que utiliza transformers para a tarefa de classificação de imagens com aumento de dados (data augmentation). Embora ainda seja uma abordagem relativamente nova em visão computacional, a arquitetura ViT tem

mostrado um grande potencial e pode ser uma alternativa promissora às redes convolucionais tradicionais. A seguir aprofundo mais acerca dos blocos constituintes do modelo ViT.

Patch é o termo empregado ao se referir às subamostras já citadas. A imagem é dividida em pequenos patches que são linearizados em uma sequência de vetores. Um patch de 16x16x3 é linearizado em um vetor de 768 posições, como a posição de cada patch na imagem é uma informação importante (por conter as informações espaciais e padrões locais) utiliza-se uma técnica chamada Position Embedding, que adiciona informações de posição ao vetor de cada patch. Existem diversos jeitos de implementar o Position Embedding, mas o jeito mais simples e bem utilizado é somar a cada vetor a posição unidimensional do patch na imagem. Esses vetores são então utilizados como entrada para o transformer encoder.

Transformer Encoder é responsável por receber uma sequência de patchs de entrada e produzir uma representação latente dessa sequência, extraindo a relação entre os patches. O transformer encoder é composto por múltiplas camadas de blocos de atenção paralelos (que calculam a importância de cada token em relação a todos os outros tokens na sequência de entrada) e camadas totalmente conectadas (que ajudam a transformar as representações intermediárias dos tokens, adicionando não-linearidades e expandindo a capacidade expressiva do modelo), além de normalização e conexões residuais. Ele permite que o modelo capture informações contextuais de uma sequência de entrada, permitindo um bom entendimento do contexto global. A figura 2.12, extraída do artigo "An image is worth 16x16 words"[17], ilustra os componentes do encoder.

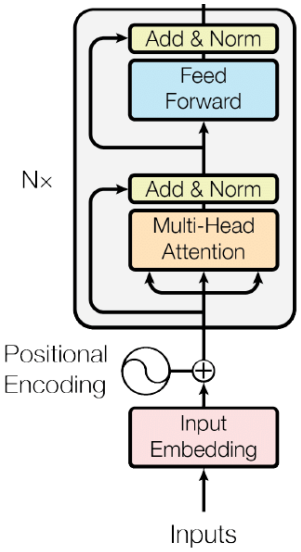


FIGURE 2.12: Representação gráfica do transformer encoder

Para exemplificar o mecanismo de atenção a figura 2.15 mostra o resultado da operação para duas palavras (”it” e ”because”) em duas heads attention diferentes (o modelo em questão possui 8 heads), o resultado pode ser lido como a intensidade da cor da linha que liga a palavra com as demais. Essa figura foi retirada de um notebook do Google Colab, criado e mantido por pesquisadores de um time do Google Brain para disseminar conhecimento sobre o tópico [18].Temos aqui algumas oportunidades de paralelização, em que as 8 heads podem estar processando em paralelo, e todas as palavras podem ser processadas em paralelo, que é um ganho de eficiência comparando com o LSTM (que faz processamento linear da frase de entrada, palavra por palavra).

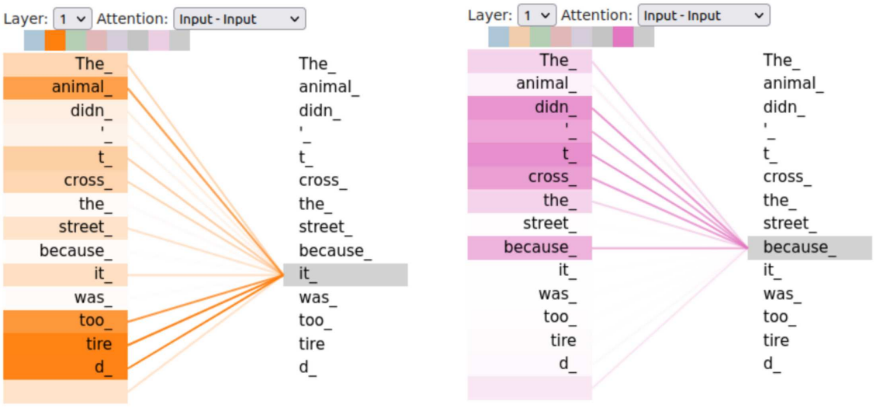


FIGURE 2.13: Representações do resultado da operação de atenção com as palavras ”it” e ”because” em uma frase

Para o caso de processamento de imagens, podemos exemplificar o mecanismo de atenção com a figura 2.14, em que a figura à esquerda é a imagem de entrada e a figura à direita é a mesma imagem mas o pixels em destaque possuem forte correlação, no fim das contas esses pixels contribuem para o modelo predizer que na imagem há um objeto da classe cachorro. Essa figura foi retirada de um repositório do GitHub[19].

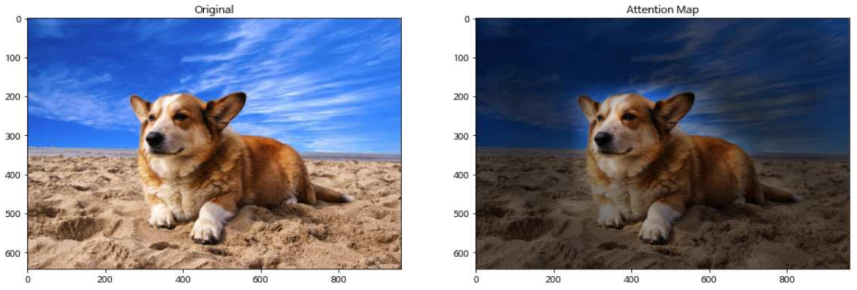


FIGURE 2.14: Mapa de atenção

Mecanismo de atenção é implementado no bloco Multi-Head Attention. A figura 2.15, extraída do artigo "An image is worth 16x16 words"[17], mostra como o Head Attention é constituído. Consideremos a entrada sendo uma matriz M, em que cada linha X_i é um vetor que representa o patch i da imagem, este vetor é transformado nos vetores Q_i (query), K_i (key) e V_i (value) a partir da multiplicação do vetor original com as matrizes W^Q, W^K e W^V, que são obtidas através do treinamento do modelo. Cada um desses vetores alimenta o bloco Multi-Head Attention nas posições Q, K e V mostradas na figura 2.15.

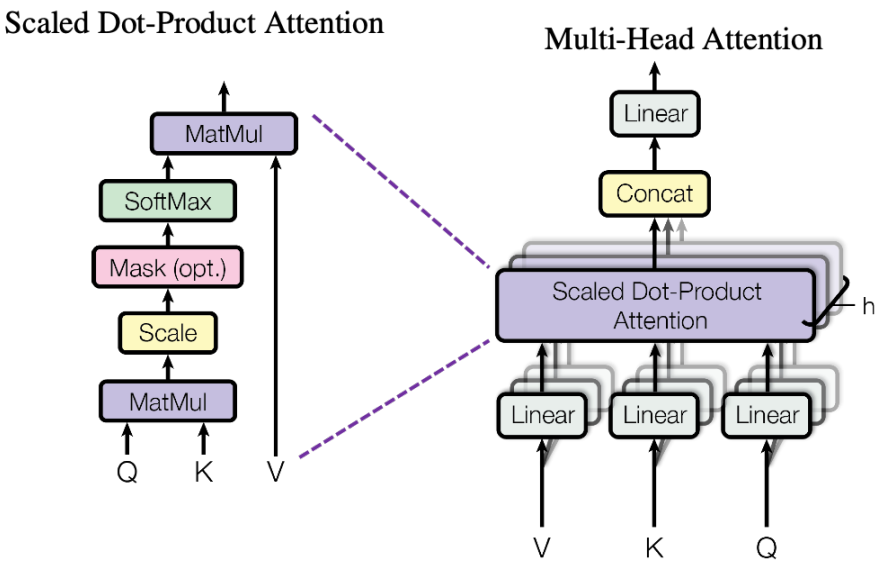


FIGURE 2.15: Representação gráfica do Multi Head Attention

Para calcular a similaridade entre os vetores podemos fazer algo similar à similaridade por cosseno, em que se calcula o produto interno dos dois vetores e normaliza o resultado, para isso multiplica-se Q_i (vetor query de um patch) por K (matriz key que agrega K_i de todos os patches) e normaliza-se o resultado dividindo por $\sqrt{d_k}$; a operação softmax é usada no resultado para transformar os valores de atenção em pesos normalizados, representando a importância relativa de cada elemento. O resultado da operação de softmax apenas extrai a atenção que deve-se dar entre os patches, já o último passo consiste em multiplicar o resultado prévio com o vetor V_i, produzindo uma nova representação que leva em consideração a importância relativa de cada elemento, enfatizando informações importantes de V_i.

Na prática ao invés de trabalhar com Q_i, K_i e V_i para cada patch de forma separada, eles são agrupados em matrizes Q, K e V, visto que processadores atuais possuem grande

capacidade de realizar operações matriciais e a matemática que é computada continua a mesma. A operação descrita anteriormente pode ser modelada pela seguinte equação:

$$Attention(Q, K, V) = softmax(\frac{Q \cdot K^T}{\sqrt{d_k}}) \cdot V$$

O resultado de todos os blocos head attention é concatenado e enviado para o próximo bloco.

2.2 Treinamento de Redes Neurais

O treinamento de redes neurais é um processo fundamental para permitir que uma rede neural aprenda a partir dos dados e realize tarefas específicas. Em resumo, o treinamento de uma rede neural envolve os seguintes passos:

1. Inicialização: Os pesos da rede neural são inicializados com valores aleatórios ou utilizando técnicas específicas, como a inicialização de Xavier. A escolha adequada dos pesos iniciais é importante para facilitar o treinamento. Não é adequado utilizar peso igual para todos os parâmetros, já que se todos os pesos tiverem o mesmo valor, as atualizações durante o treinamento seriam simétricas em relação a cada peso. Isso pode levar a problemas como falta de diversidade nas respostas dos neurônios e dificuldade em capturar relações complexas nos dados.

2. Forward Propagation: Os dados de entrada são alimentados para a rede neural, que realiza uma propagação direta (forward propagation) através das camadas. Cada camada aplica uma transformação aos dados de entrada e produz uma saída ativada.

3. Cálculo da Função de Perda: A saída da rede neural é comparada com os rótulos ou valores esperados usando uma função de perda (loss function). A função de perda quantifica o quão bem o modelo está performando na tarefa em questão.

4. Backpropagation: O algoritmo de backpropagation calcula o gradiente da função de perda em relação aos pesos da rede neural. Esse gradiente indica a direção e a magnitude da mudança necessária nos pesos para minimizar a função de perda.

5. Atualização dos Pesos: Com base no gradiente calculado durante o backpropagation, os pesos da rede neural são atualizados usando um otimizador, como Adam, RMSProp ou SGD (Stochastic Gradient Descent). O otimizador ajusta os pesos de forma a minimizar a função de perda.

6. Iteração: Os passos 2 a 5 são repetidos para vários exemplos de treinamento, formando uma época. O processo é repetido por várias épocas, permitindo que a rede neural refine gradualmente seus pesos e melhore seu desempenho.

7. Avaliação: Periodicamente, a rede neural é avaliada em um conjunto de dados de validação separado para verificar seu desempenho em dados não vistos durante o treinamento. Isso ajuda a monitorar a capacidade de generalização do modelo e evitar o overfitting.

8. Teste: Após o treinamento, a rede neural é avaliada em um conjunto de dados de teste independente para obter uma estimativa final de seu desempenho. Esse conjunto de dados fornece uma avaliação imparcial do modelo em dados não vistos.

A seguir aprofundo em algumas técnicas e bases matemáticas referentes aos passos acima citados.

2.2.1 Transfer learning

A técnica de transferência de aprendizado está relacionada com a etapa de inicialização dos pesos de uma RNA. O objetivo desta técnica é de usar pesos de um modelo pré treinado para, ao treinar o mesmo para um novo dataset, não começar do zero. Essa abordagem possui algumas vantagens, como poupar recursos computacionais e tempo ao treinar e/ou refinar um modelo.

Na parte experimental deste trabalho, com os modelos convolucionais, foram baixados, de modelos pré treinados com o dataset ImageNet, os dados dos filtros (kernels) das camadas convolucionais e foram acoplados aos novos modelos que foram criados. Dessa maneira as camadas convolucionais herdam os filtros de um modelo público e as camadas densas são inicializadas com pesos aleatórios. Já com o modelo transformer pré treinado (ViT-B/16) que foi utilizado o modelo que foi baixado foi integralmente utilizado e treinado, com a exceção da última camada que originalmente classifica entre 1000 classes e neste trabalho é adaptada para classificar entre 102 classes.

2.2.2 Função de perda

Como já citado a função perda, em inglês loss function, serve para metrificar como o modelo está performando em dada tarefa, ela o faz representando a diferença entre a previsão do modelo e os valores reais dos dados. Existem diferentes funções de perda para diferentes tipos de problema, por exemplo, quando o problema é de regressão recomenda-se usar a função Erro Quadrático Médio, quando o problema é de classificação multiclasse (em que cada exemplo de entrada relaciona-se com apenas uma classe) recomenda-se usar Entropia Cruzada Categórica (em inglês, Categorical Cross Entropy), que é o método usado neste trabalho.

Softmax Para explicar a função Categorical Cross Entropy é necessário detalhar a respeito da saída do modelo. O problema que este trabalho foca é o de classificação multiclasse,

assim, a saída do modelo deve retornar a probabilidade que uma dada entrada possui de ser cada uma das classes do dataset. A figura 2.16 exemplifica a saída de um modelo que faz a classificação entre 3 classes, a penúltima camada é de neurônios que tem uma função de ativação não limitada, que pode ir de zero a $+\infty$, como a ReLU ou GELU. Para a saída ser a probabilidade das classes, de forma percentual, podemos usar uma camada softmax, que converte uma lista de N números em uma distribuição probabilística de N resultados possíveis, de tal maneira cujo somatório da saída é igual a 1.

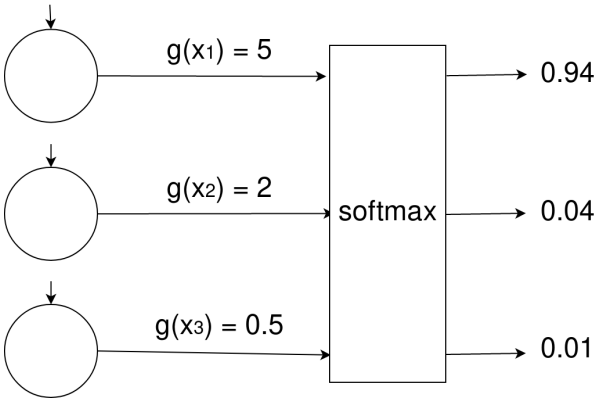


FIGURE 2.16: Representação gráfica da saída softmax

Categorical Cross Entropy é a função de perda para problemas multiclasse usada nesse trabalho. $-\sum_{i=1}^{C} y_i log(p_i)$ é a equação que a descreve, em que :

- C é o número de classes possíveis,
- y_i é o rótulo (ou valor verdadeiro) binário da classe i (0 ou 1),
- p_i é a probabilidade prevista pelo modelo para a classe i.

A título de exemplo peguemos a saída representada na figura 2.16 [0.94,0.04,0.01] sendo o vetor p, e [1,0,0] sendo o vetor y. O valor do erro para esse cenário é de 0.05 aproximadamente, já para o caso em que y é igual a [0,1,0] o valor do erro é de 3.2 aproximadamente. O ajuste dos pesos do modelo é feito de maneira que a saída do modelo se aproxime da saída desejada.

2.2.3 Otimizadores

Como já citado no começo desta seção, uma etapa fundamental no treinamento de um modelo de RNA é o ajuste dos pesos deste modelo, e esta é uma tarefa de otimização

matemática. Segundo a Wikipédia, "*a otimização matemática é a seleção de um melhor elemento, com relação a algum critério, de algum conjunto de alternativas disponíveis. [...] No caso mais simples, um problema de otimização consiste em maximizar ou minimizar uma função real escolhendo sistematicamente valores de entrada de um conjunto permitido e computando o valor da função.*" No contexto de treinamento de RNA o problema de otimização que busca-se resolver é o de minimizar a função loss, ajustando os pesos do modelo de modo a reduzir esse loss. Para isso, os otimizadores utilizam informações sobre a direção e a magnitude dos gradientes para ajustar os pesos de forma iterativa e realizar as atualizações necessárias para minimizar a função de perda. Existem diferentes tipos de otimizadores, cada um com suas próprias características e algoritmos de atualização dos pesos. Alguns exemplos populares são:

Gradient Descent (Descida do Gradiente) é o otimizador básico que realiza atualizações dos pesos na direção oposta ao gradiente da função de perda. O Gradient Descent tem várias variantes, incluindo o Batch Gradient Descent, que atualiza os pesos considerando todos os exemplos de treinamento de uma vez, e o Stochastic Gradient Descent (SGD), que atualiza os pesos para cada exemplo de treinamento individualmente.

Adam (Adaptive Moment Estimation) é um otimizador que combina elementos do Momentum e do RMSProp. Ele calcula adaptativamente taxas de aprendizado individuais para cada parâmetro com base em estimativas do primeiro e segundo momentos dos gradientes. O Adam é amplamente utilizado devido ao seu desempenho eficaz e capacidade de se adaptar a diferentes tipos de problemas.

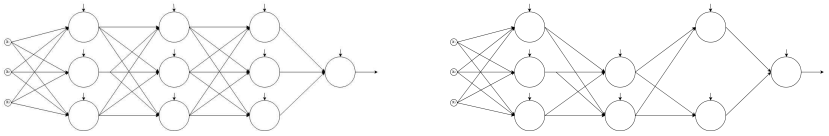
RMSProp (Root Mean Square Propagation) é um otimizador que ajusta as taxas de aprendizado para cada parâmetro com base na média dos gradientes recentes. Ele utiliza uma média móvel do quadrado dos gradientes para controlar a taxa de aprendizado de cada parâmetro de forma adaptativa.

2.2.4 Overfitting

O sobreajuste (overfitting) quando ocorre em um modelo de rede neural treinado é um problema. Pode-se dizer que ocorre overfitting quando um modelo estatístico se ajusta demais com a base de dados de treino, mas não tem a capacidade de generalizar o conhecimento, obtendo baixo desempenho com bases de dados que não são a base de treino. Para resolver esse problema existem algumas técnicas que podem ser utilizadas na etapa de treinamento do modelo, chamadas de técnicas de regularização. Por exemplo: dropout,

Regularização L1 e L2, data augmentation, batch normalization e early stopping. Abaixo discorro brevemente sobre algumas dessas técnicas de regularização.

Dropout é uma técnica que envolve a desativação aleatória de um conjunto de neurônios durante o treinamento, na implementação é definida a taxa porcentual de desativação dos neurônios por camada. Isso força a rede neural a aprender representações mais robustas, pois não pode depender excessivamente de unidades individuais. O Dropout melhora a generalização do modelo. A figura 2.17 ilustra como os neurônios são desativados.



(a) MLP (b) MLP com Dropout

FIGURE 2.17: Representação do dropout

Data Augmentation é uma técnica em que o conjunto de dados de treinamento é aumentado artificialmente com transformações nos dados existentes. Isso pode incluir rotações, reflexões, redimensionamentos, deslocamentos e outras manipulações dos dados de entrada. Essa técnica aumenta a quantidade de dados disponíveis para o treinamento e ajuda a melhorar a capacidade de generalização da rede neural.

Early Stopping (parada precoce) é uma técnica que consiste em monitorar a evolução do loss ou acurácia do conjunto de validação e parar o treino quando esses valores estabilizaram ou começaram a crescer. A figura 2.18 representa como são as curvas de loss em um cenário de overfitting, em que o modelo começa a se especializar com o dataset de treino e deteriora sua precisão com dataset de validação, e a técnica de early stopping entra nesse contexto para impedir que o modelo se especialize e fique com a melhor precisão possível.

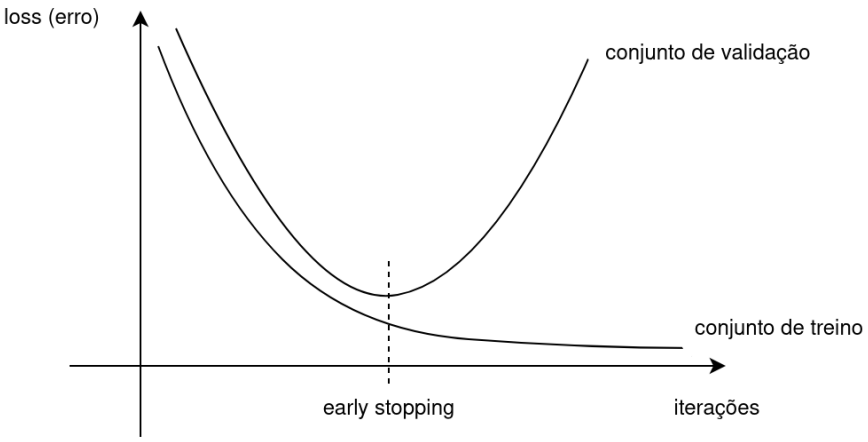


FIGURE 2.18: Early Stopping

2.3 Técnicas de avaliação

Como já citado na seção sobre treinamento de redes neurais a etapa de teste (ou avaliação) ocorro após o treino, em que ele é avaliado usando uma partição do dataset diferente da usada no treinamento, para constatar de modo imparcial precisão do modelo. Existem alguns métodos para a avaliação de desempenho de um modelo e nessa sessão discorro brevemente sobre os métodos mais importantes e amplamente utilizados.

2.3.1 Acurácia

O modo mais simples de medir a precisão de um modelo é pela acurácia, que é basicamente a razão entre o número acertos do modelo pelo número de casos de teste. Digamos que ao medir a precisão de um modelo com 100 casos ele acerta 95 casos dizemos que a acurácia deste modelo é de 95%.

Apesar de ser uma métrica muito conveniente pode haver cenários em que ela seja inconveniente, como no caso em que se está trabalhando com um dataset desbalanceado. Por exemplo, considerando um dataset com 3 classes, a 1ª tem 90 exemplares, a 2ª tem 5 exemplares e a 3ª tem 5 exemplares, se um modelo de classificação classificar todos os exemplares como sendo da 1ª classe ele terá uma acurácia de 90%, pode parecer uma performance boa, mas neste cenário é complemente distorcido.

2.3.2 Recall, Precisão e F1-Score

Para contornar o problema de avaliação de datasets desbalanceados é interessante o uso de métricas que considerem a taxa de verdadeiros positivos (TPR), também conhecida como recall, e a taxa de falsos positivos (FPR).

Recall ponderado é calculado levando em consideração o desequilíbrio das classes no conjunto de dados. Ele calcula o recall separadamente para cada classe e, em seguida, faz uma média ponderada pelos pesos relativos às proporções das classes no conjunto de dados. Essa métrica fornece uma avaliação do desempenho de recuperação para cada classe, considerando o desequilíbrio.

Precisão segue uma abordagem semelhante ao recall ponderado. Ela calcula a precisão para cada classe e faz uma média ponderada com base nas proporções das classes. Essa

métrica fornece uma avaliação do desempenho de precisão para cada classe, considerando o desequilíbrio.

F1-Score Ponderado uma média ponderada do F1-score para cada classe, levando em consideração o desequilíbrio das classes. Assim como o recall ponderado e a precisão ponderada, essa métrica fornece uma avaliação balanceada do desempenho em relação a todas as classes, considerando suas proporções no conjunto de dados.

2.3.3 Matriz de confusão

A matriz de confusão é uma representação visual das previsões de um modelo de classificação em relação aos rótulos verdadeiros. É uma boa ferramenta para avaliar o desempenho de um modelo e entender como ele está classificando corretamente e erroneamente as instâncias em diferentes classes. A matriz de confusão é organizada em forma de tabela, em que as linhas representam as classes verdadeiras e as colunas representam as classes previstas pelo modelo. Cada célula da matriz mostra a contagem (ou proporção) de instâncias classificadas em uma determinada combinação de classe verdadeira e classe prevista.

O exemplo da figura 2.19 mostra um caso em que há 3 classes, 30 casos testados (sendo 10 casos por classe). Temos 7 casos de verdadeiro positivo para classificação de cachorro, 8 casos de verdadeiro positivo para classificação de gato e 9 casos de verdadeiro positivo para classificação de cavalo. Para a classe cachorro houveram 2 casos em que o modelo indicou que era gato e 1 caso indicou que era cavalo, para a classe gato houveram 2 casos em que o modelo indicou que era cavalo, e para a classe cavalo houve 1 caso em que o modelo indicou que era gato. Esse modelo de visualização é muito bom para verificar a distribuição de erros e acertos do modelo entre as diversas classes, a desvantagem ocorre quando o conjunto de dados possui muitas classes (na casa das centenas) dificultando a visualização, nesse caso pode ser uma alternativa criar subconjuntos de classes para analisar de forma mais detalhada. Por exemplo, agrupar classes relacionadas ou selecionar um conjunto aleatório de classes para análise. Ou em vez de analisar todas as classes, pode-se concentrar em classes críticas ou de maior importância para o problema específico.

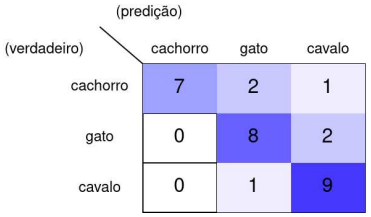


FIGURE 2.19: Matriz de Confusão

Chapter 3

Desenvolvimento do trabalho

Este capítulo descreve como foi a implementação do trabalho, discorrendo acerca das escolhas feitas, técnicas de treinamento e avaliação adotadas, modelos de RNA e bases de dados utilizadas. Em resumo, foram usadas duas abordagens experimentais, em uma abordagem utilizou-se um dataset pequeno com 7 classes, inicialização randômica dos modelos e avaliação com matriz de confusão, na outra abordagem utilizou-se um dataset maior com 102 classes, transfer learning e avaliação com F1-Score ponderado.

3.1 Arquiteturas utilizadas

A pesquisa dos modelos de classificação utilizados teve como foco os modelos mais citados na literatura, que são aqueles que propuseram inovações no momento de sua publicação e obtiveram desempenho superior aos padrões da época. Os modelos utilizados são apresentados a seguir, com uma breve descrição dos mecanismos básicos de funcionamento de cada um deles. Os pesos e vieses utilizados na abordagem de transfer learning advém do treinamento com o dataset Imagenet

VGG16 e VGG19 são modelos clássicos no campo de visão computacional, sendo que o 1º já discutido neste trabalho no capítulo anterior, a diferença entre eles é que a VGG19 possui 19 camadas enquanto a VGG16 possui 16.

InceptionResNetV2 é uma arquitetura de rede neural convolucional (CNN) proposta em 2016 por Christian Szegedy et al.[20] como uma extensão do modelo InceptionV3. Este modelo combina elementos do modelo Inception e do modelo ResNet, a figura 3.1 representa a arquitetura da InceptionResNetV2 e foi extraída do artigo de Masoud Mahdianpari[21].

No ano de 2016 este modelo conseguiu o melhor desempenho até então com o dataset Imagenet, como aponta a tabela 1.1.

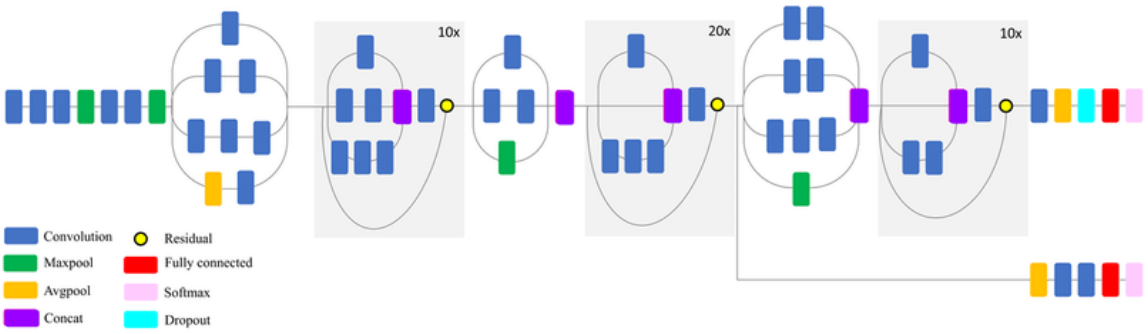


FIGURE 3.1: Representação gráfica do modelo InceptionResNetV2

EfficientNet é uma arquitetura proposta em 2019[22] e se destaca por sua eficiência em termos de uso de recursos computacionais, ao mesmo tempo em que mantém um alto desempenho em tarefas de visão computacional. A arquitetura da EfficientNet implementa uma estratégia matemática para escalar seu poder de processamento, através do aumento do número de filtros convolucionais, aumento da resolução da imagem de entrada e aumento da profundidade do modelo. Essa estratégia consiste em usar o método chamado Compound Scaling, que ajusta simultaneamente a largura, altura e profundidade da rede de forma proporcional, permitindo que ela seja dimensionada de forma eficiente em diferentes tamanhos. A arquitetura EfficientNet é uma família de 8 modelos, que avança do B0 para o B7, de maneira que o tamanho e a complexidade aumentam progressivamente, obtendo maior precisão e requerendo maior poder computacional. O modelo utilizado aqui foi o EfficientNetB0 por conta de trabalharmos com datasets pequenos, e a possuirmos recursos computacionais limitados.

ViT é o primeiro modelo de vision transformer publicado, já discutido no capítulo anterior. Na etapa em que se avaliou o modelo pré-treinado, para transfer learning, foi usado o modelo ViT-B/16, que possui 12 camadas de encoder, e um total de 86M de parâmetros

treináveis, este modelo foi pré-treinado com o dataset ImageNet-21K. O código deste modelo utilizado nos experimentos foi retirado do site de documentação do Keras, disponível na bibliografia [23].

Swin Transformer (Sliding-Window Inference Models) é uma variação da arquitetura Vision Transformer, que foi proposta por no artigo "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows" em 2021[24]. Ela é adaptada para lidar com tarefas que envolvem processamento de imagens de alta resolução. O Swin Transformer adota uma estratégia de janelas deslizantes, dividindo a imagem em pequenas regiões, chamadas de janelas, realiza processamentos independentes em cada uma delas. Essa abordagem permite que o Swin Transformer processe eficientemente imagens de alta resolução, capturando informações contextuais ricas em cada região da imagem. Posteriormente, as informações relevantes são combinadas para gerar uma saída coerente, permitindo que o modelo compreenda e capture efetivamente os detalhes e a estrutura das imagens. O Swin Transformer é particularmente útil em tarefas como segmentação semântica e detecção de objetos em imagens de alta resolução, onde a preservação dos detalhes é crucial para uma representação precisa da cena. O código deste modelo utilizado nos experimentos foi retirado do site de documentação do Keras, disponível na bibliografia [25].

3.2 Bases de dados

Dataset autoral foi a primeira base de dados usada nos experimentos e foi criada pelo autor deste trabalho. Este dataset possui no total 350 imagens, de 7 classes diferentes (Lantana Rasteira, Algodão Bravo, Amor Agarradinho, Ave do Paraíso, Calístemo, Jasmin e Orquídea), sendo 50 imagens por classe. A motivação de usar um dataset original se deu ao fato do autor buscar estar mais próximo do contexto da aplicação deste trabalho.

A resolução original das imagens é de 3088 por 3088 pixels, porém nos experimentos a resolução foi reduzida para 75 por 75 pixels. O equipamento utilizado para efetuar os registros foi a câmera de um aparelho celular Samsung SM-A105M, os registros foram realizados com luz natural durante o dia. A escolha das espécies se deu de maneira aleatória, de acordo com as espécies que foi possível capturar um bom número de fotografias com algum grau de diversidade entre elas. A título de exemplo a figura 3.2 mostra alguns exemplos de imagens do dataset autoral, o dataset pode ser encontrado de forma integral no repositório do projeto por meio deste github.com/Fonsecaaso/ViT-vs-CNN/tree/master/Experiments

FIGURE 3.2: Exemplo de imagens do dataset autoral

Oxford 102[26] foi o dataset utilizado em um segundo momento, contém 102 classes de plantas, em que cada classe possui de 40 a 258 imagens, 6149 destinadas ao treino do modelo e 1020 para avaliação (teste). Na figura abaixo 3.3, retirada do site Research Gate [27], segue uma amostra das imagens do dataset Oxford 102. Ao longo do desenvolvimento do trabalho, com a implementação dos modelos de classificação e avaliação de desempenho

dos mesmos, se mostrou necessário usar uma base de dados maior. Uma vez que ao usar uma base maior é possível inferir com melhor qualidade a capacidade de aprendizagem dos modelos analisados, pois datasets com mais classes e maior diversidade de imagens e contextos traz consigo um desafio de aprendizagem maior para os modelos.

FIGURE 3.3: Exemplo de imagens do dataset Oxford 102

3.3 Data augmentation

Como já descrito na sessão Data augmentation do capítulo anterior, esta técnica é utilizada para evitar o overfitting do modelo, ao aplicar pequenas variações às imagens de entrada na fase de treinamento o modelo não "decora" os padrões do dataset de treina mas sim aprende e se ajusta de modo a generalizar, desenvolvendo a capacidade de classificar outras imagens corretamente que não foram apresentadas a ele durante o treino.

As técnicas aqui utilizadas foram escolhidas por serem amplamente citadas na literatura e pela comunidade de machine learning pelo fato de apresentarem bons resultados. Foram utilizados os métodos **RandomFlip**, **RandomRotation** e **RandomZoom** da biblioteca layers do Keras. Abaixo há as figuras 3.4 e 3.5 que exemplificam a aplicação das técnicas de **RandomFlip** e **RandomZoom**, essas imagens foram retiradas do site tensorflow.org[28]

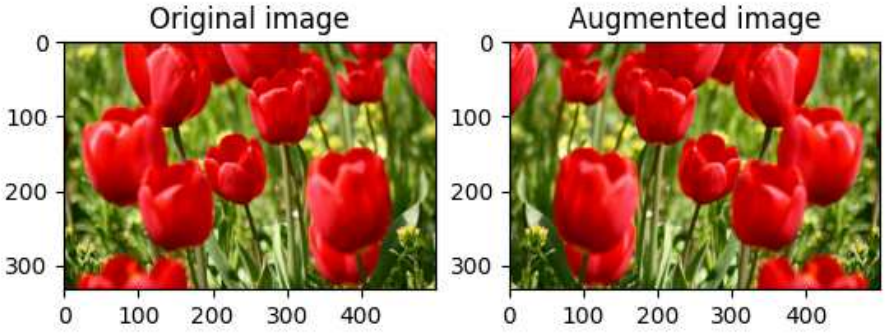


FIGURE 3.4: Random Flip

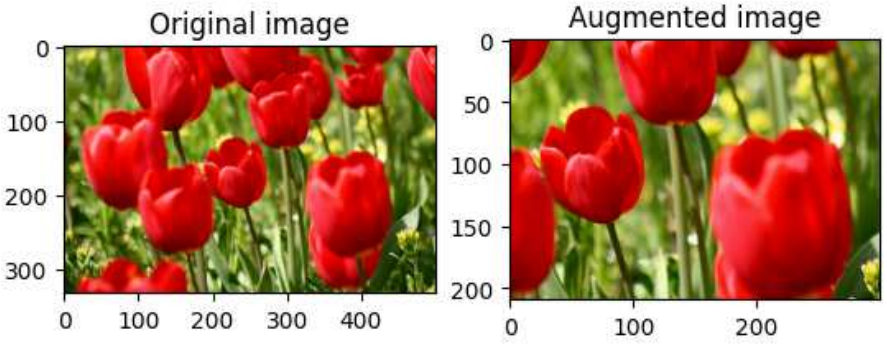


FIGURE 3.5: Random Zoom

3.4 Ambiente de execução

Para treinar modelos de aprendizado de máquina do zero, que são geralmente grandes e possuem muitos parâmetros treináveis, além de receberem imagens de alta resolução como entrada, é necessário contar com recursos computacionais adequados. Nesse contexto, o uso do ambiente Google Colab Pro foi necessário devido aos recursos disponíveis. O Colab Pro oferece limitações de tempo de uso, mas apresenta uma configuração com 25GB de RAM, 166GB de espaço em disco e uma GPU Nvidia T4 com 15GB de memória. A GPU T4 é especialmente projetada para lidar com tarefas de deep learning, como treinamento e inferência de modelos. Vale destacar que o serviço Colab utiliza um processador Intel Xeon, que é uma linha de processadores projetada especificamente para cargas de trabalho de servidores e estações de trabalho de alto desempenho. Ele oferece recursos avançados, como suporte a multi-threading, cache de alto desempenho e uma arquitetura escalável que permite lidar com tarefas computacionalmente intensivas de forma eficiente.

3.5 Experimentos e Resultados

Esta seção discorre acerca de algumas decisões tomadas nos experimentos e exibe os resultados obtidos. Como já dito a parte prática deste trabalho foi realizada com dois experimentos distintos, em um utilizou-se o dataset autoral e no outro utilizou-se o dataset Orford102. A decisão de utilizar-se duas bases de dados possui algumas motivações, como:

- Averiguar a capacidade de aprendizagem dos modelos em diferentes contextos, visto que no experimento com o dataset autoral foram utilizadas 280 imagens de treino para 7 classes e no experimento com o dataset Oxford102 foram utilizadas mais de 6000 imagens de treino para 102 classes;
- Permitir o estudo de ferramentas diversas, por exemplo, o uso de matriz de confusão funciona em um contexto de poucas classes, que é o caso do experimento com dataset autoral, já o uso de F1-Score Ponderado é útil no cenário de datasets desbalanceados, que é o caso do experimento com o dataset Oxford102. Essa motivação é a mesma para o uso de transfer learning em um experimento e o não uso em outro

A lista acima não esgota as discussões acerca de decisões tomadas nos experimentos, sendo esta apenas uma introdução aos assuntos que nas próximas seções serão aprofundados com maior rigor e detalhamento. Cabe pontuar que nessa seção apenas apresenta-se alguns resultados e uma análise parcial, sendo que uma análise e discussão mais aprofundadas ficam a cargo do próximo capítulo desta monografia.

3.5.1 Dataset Autoral

Como já devidamente apresentado esse dataset possui 7 classes, todas as classes possuem o mesmo número de imagens, a subdivisão de treino possui 280 amostrar, as de validação e teste possuem 35 amostrar cada. Foi tomada a decisão de não utilizar a técnica de transfer learning, uma vez que é um conjunto de dados reduzido entende-se que o aprendizado que é uma tarefa mais simples às arquiteturas estudadas neste experimento em contraste ao experimento com conjunto de dados maior. Outro motivo de optar por não usar transfer learning neste experimento consiste no fato de que assim pode-se permitir estudar com maios detalhe acerca dos ritmos e dinâmicas dos modelos a partir do zero (pesos e bias inicializados randomicamente), permitindo uma comparação mais detalhada entre os modelos.

Experimentos, nessa etapa de fato foram feitas dezenas de tentativas, ou experimentações, para encontrar uma boa implementação de código que permitisse uma análise dos diversos modelos de forma coerente e eficaz. Para tanto foram ajustados parâmetros como tamanho do batch, número de épocas de treinamento, tipo do otimizador, dimensão das imagens de entrada e técnicas de avaliação.

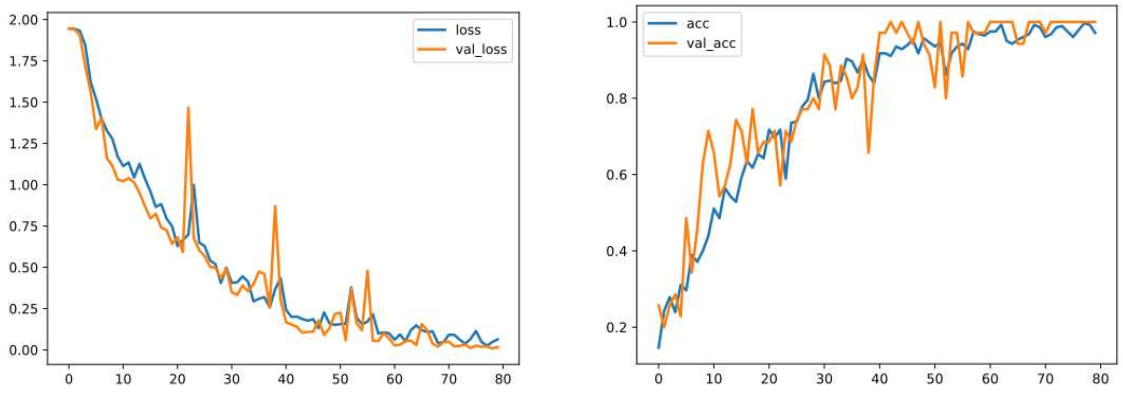
As escolhas e ajustes dos parâmetros citados tiveram como objetivo final uma análise justa e equilibrada entre os diversos modelos, em termos de ajustes podemos citar que para alguns modelos usar um mini-batch de tamanho 1 era suficiente para treiná-lo e obter bom desemprenho, entretanto para outros modelos usar batch de tamanho 1 não permitia treiná-lo, ficando estagnado em patamares baixos de desempenho. O número de épocas de treinamento escolhido foi aquele que permitiu os modelos atigirem seu potencial máximo (acurácia) com esse dataset. Outro exemplo é o caso da resolução da imagem, para o modelo InceptionResNetV2 a largura e altura das imagens não podem ser menores que 75 pixels, já o modelo ViT retirado do site do Keras[23] trabalha com imagens com resolução de 72 pixels (que é um valor ancorado em outros parâmetros, como tamanho dos patches). A respeito do tamanho dos modelos utilizados e outros dados relevantes a planilha 3.1 os compila.

TABLE 3.1: Tamanho dos modelos utilizados nos experimentos com dataset autoral

Modelos	nº parâmetros	nº camadas*	tamanho das imagens
VGG16	15M	16	(72,72,3)
InceptionResNetV2	54M	164	(72,72,3)
EfficientNetB0	4.2M	326	(75,75,3)
Vision Transformer	10M	2	(72,72,3)
Swin Transformer	0.2M	16	(72,72,3)

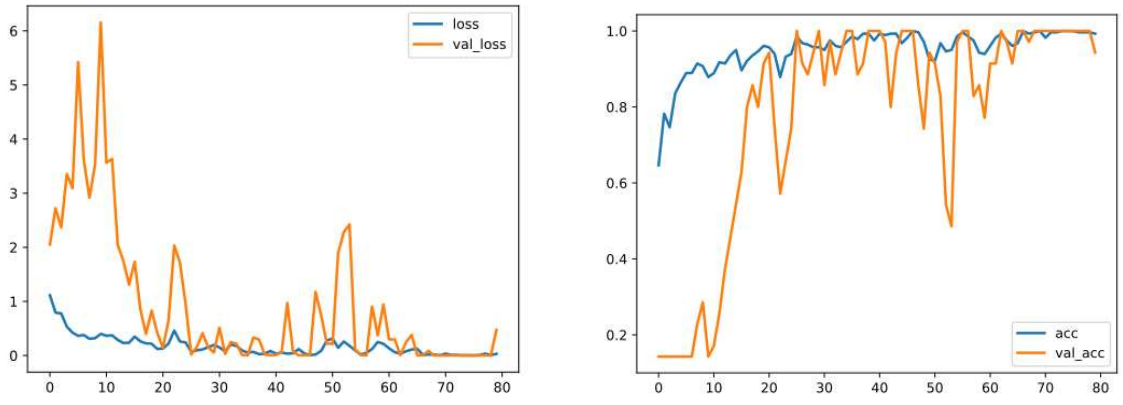
*número de camadas convolucionais e número de blocos encoder

Resultados obtidos a partir dos experimentos são agora apresentados de forma sucinta. As figuras 3.6, 3.7, 3.8, 3.9 e 3.10 mostram as curvas de acurácia e loss obtidas durante o treinamento, dos modelos VGG16, InceptionResNetV2, EfficientNetB0, ViT e Swin, respectivamente.



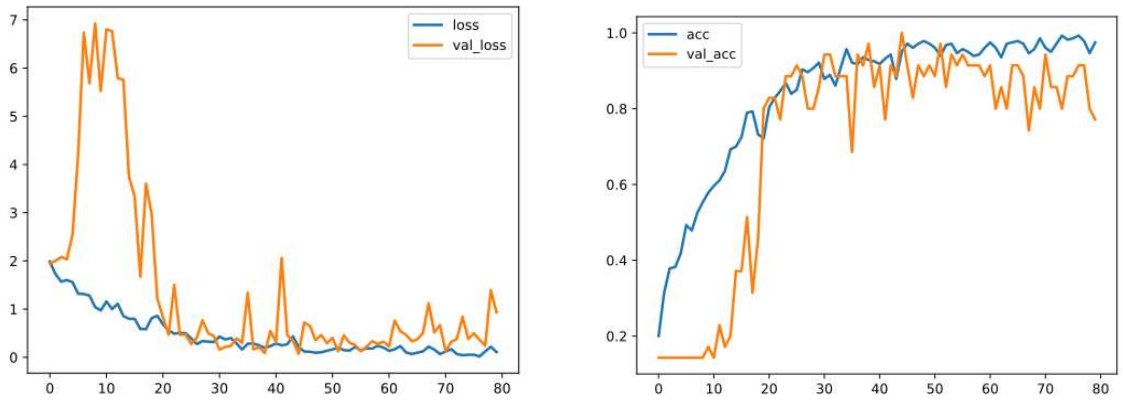
(a) Loss (b) Acurácia

FIGURE 3.6: Evolução das curvas de loss e acurácia do modelo VGG16



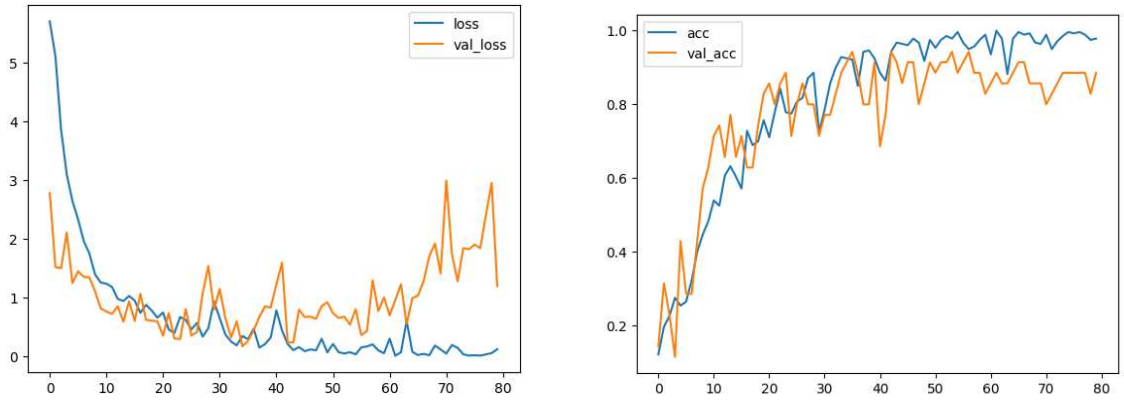
(a) Loss (b) Acurácia

FIGURE 3.7: Evolução das curvas de loss e acurácia do modelo InceptionResNetV2



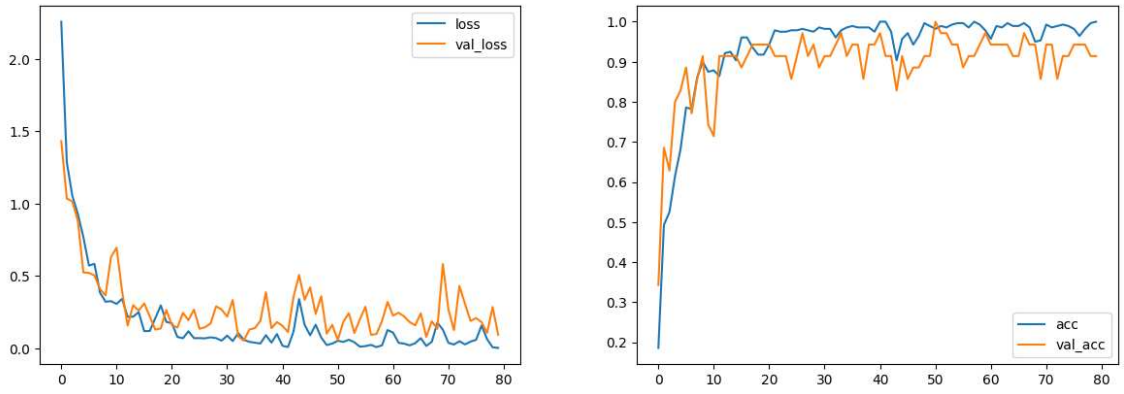
(a) Loss (b) Acurácia

FIGURE 3.8: Evolução das curvas de loss e acurácia do modelo EfficientNetB0



(a) Loss (b) Acurácia

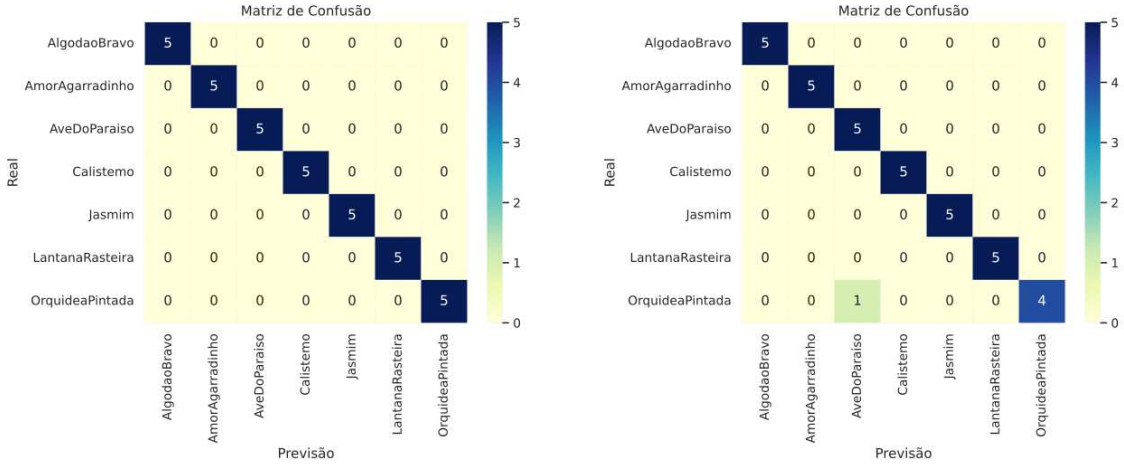
FIGURE 3.9: Evolução das curvas de loss e acurácia do modelo Vision Tranformer



(a) Loss (b) Acurácia

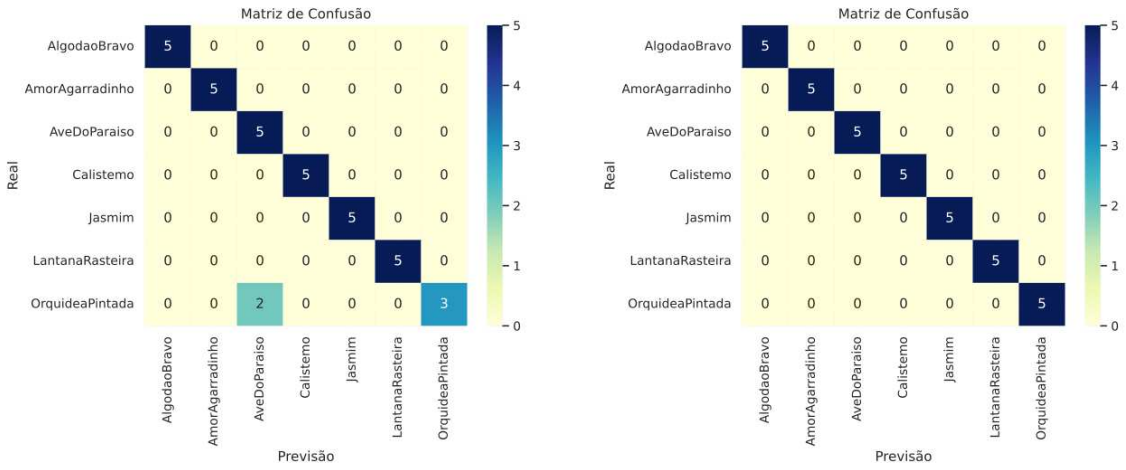
FIGURE 3.10: Evolução das curvas de loss e acurácia do modelo Swin Tranformer

As figuras 3.11, 3.12, 3.13, 3.14 e 3.15 mostram as as matrizes de confusão obtidas testando o modelo treinado com os conjuntos de avaliação e teste, dos modelos VGG16, InceptionResNetV2, EfficientNetB0, ViT e Swin, respectivamente.



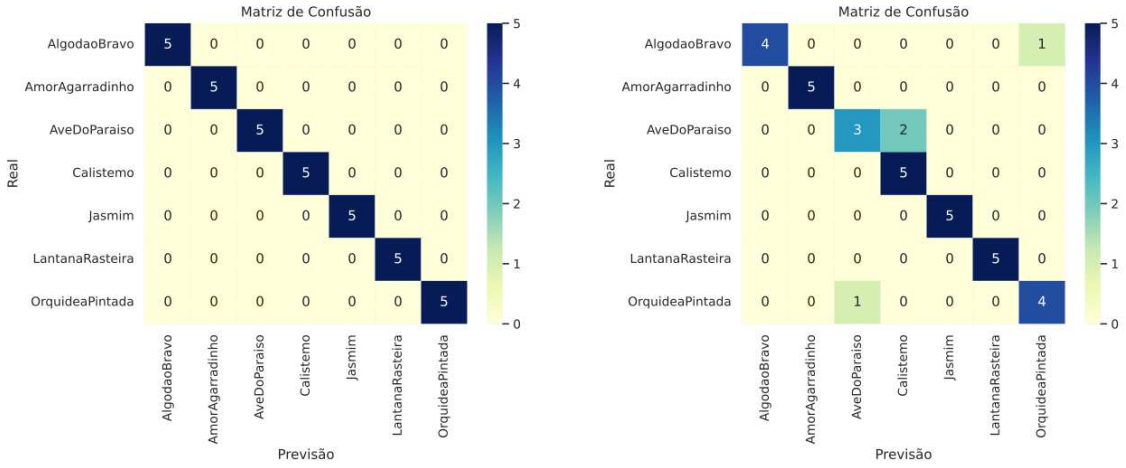
(a) Validação (b) Teste

FIGURE 3.11: Matrizes de confusão do modelo VGG16 para os conjuntos de dados de validação e teste



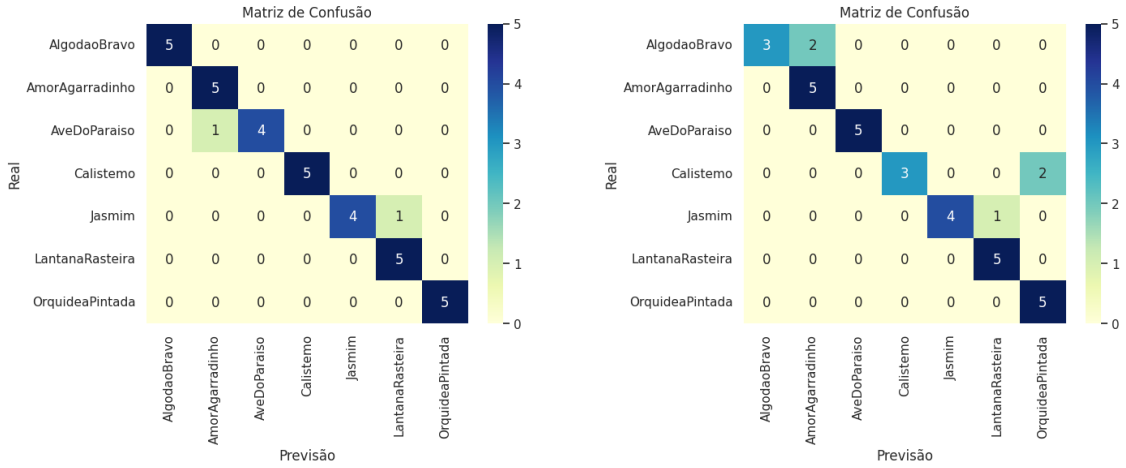
(a) Validação (b) Teste

FIGURE 3.12: Matrizes de confusão do modelo InceptionResNetV2 para os conjuntos de dados de validação e teste



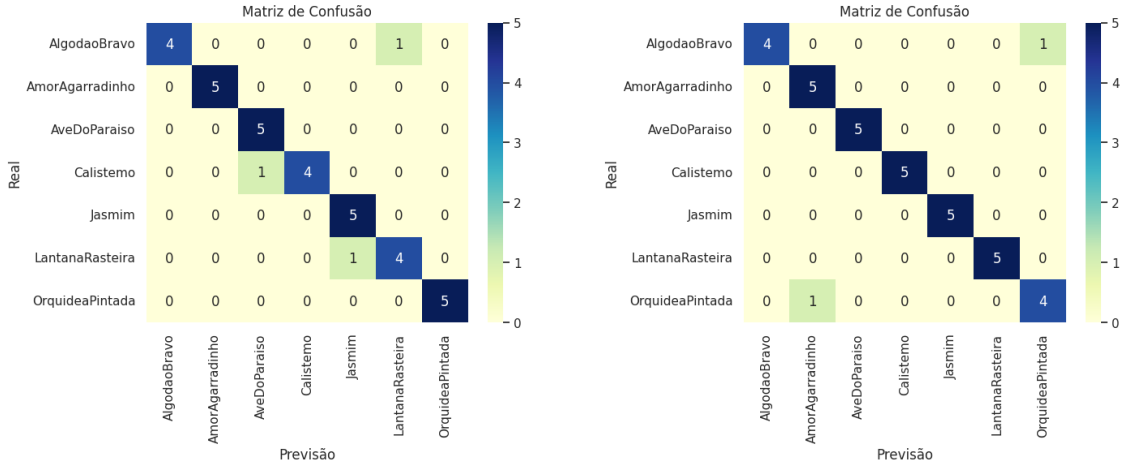
(a) Validação (b) Teste

FIGURE 3.13: Matrizes de confusão do modelo EfficientNetB0 para os conjuntos de dados de validação e teste



(a) Validação (b) Teste

FIGURE 3.14: Matrizes de confusão do modelo Vision Tranformer para os conjuntos de dados de validação e teste



(a) Validação (b) Teste

FIGURE 3.15: Matrizes de confusão do modelo Swin Tranformer para os conjuntos de dados de validação e teste

As tabelas 3.2 e 3.3 agregam os resultados obtidos dos treinamentos dos modelos de classificação. Optou-se por usar dois subconjuntos distintos para analisar-se com maior detalhe a capacidade de aprendizagem dos modelos estudados. Os dados apresentados são acurácia, loss e tempo de predição.

TABLE 3.2: Resultado obtidos dos modelos treinados com o conjunto de validação

Modelos	acurácia	loss	Tempo de predição
VGG16	100%	0	28ms
InceptionResNetV2	94.29%	0.47	49ms
EfficientNetB0	100%	0	15ms
ViT	94.29%	0.26	14ms
Swin Transformer	91.43%	0.11	14ms

TABLE 3.3: Resultado obtidos dos modelos treinados com o conjunto de teste

Modelos	acurácia	loss	Tempo de predição
VGG16	100%	0	32ms
InceptionResNetV2	100%	0	71ms
EfficientNetB0	88.57%	0.48	16ms
ViT	85.71%	1.72	14ms
Swin Transformer	94.29%	0.21	13ms

3.5.2 Dataset Oxford 102

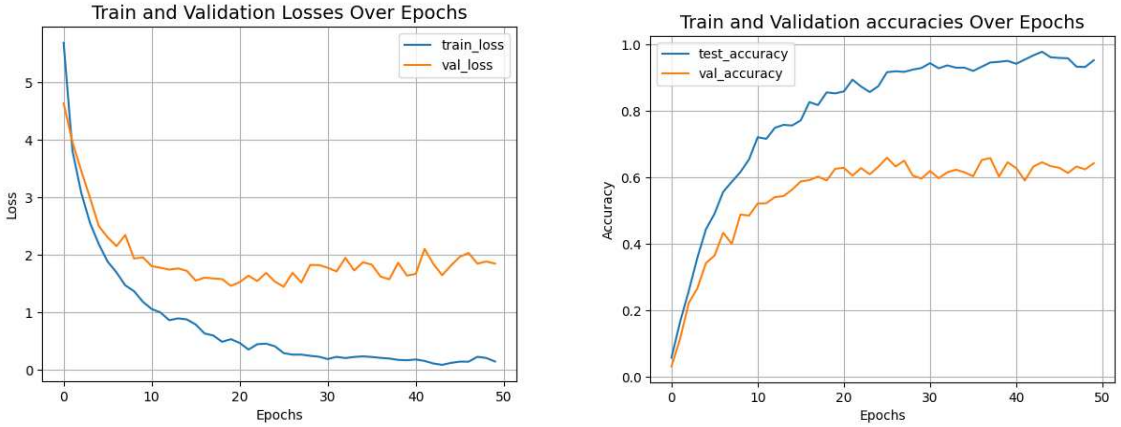
Como já previamente apresentado esse dataset possui 102 classes, em que o número de amostras por classe varia entre 40 e 258, em que 6149 amostras são destinadas ao treinamento do modelo e 1020 para avaliação. Foi tomada a decisão de utilizar a técnica de transfer learning, uma vez que é um conjunto de dados mais complexo e que treinar os modelos do zero daria muito mais trabalho, por conta de ser necessário ajustar diversos hiperparâmetros através de diversos experimentos. Assim, ao usar transfer learning requere-se muito menos poder de processamento computacional tornando este trabalho viável. Os modelos utilizados nessa fase de experimentos são os mesmo utilizados na fase de dataset autoral, com exceção do Swin Transformer que não há modelo pré-treinado e treinável disponível na comunidade.

Experimentos, nessa etapa já havia conhecimento acumulado proveniente dos experimentos com o dataset autoral, de tal maneira que em seu desenvolvimento as alterações feitas foram de maneira a utilizar transfer learning (em que os modelos foram previamente treinados com o dataset ImageNet). Outra mudança relevante foi a decisão de usar uma resolução de 224 pixels, em contras da resolução de 72 pixels usada no experimento anterior, uma vez que há muito mais classes e um desafio maior para os modelos inferirem caracteristicas de cada classe, assim, com maior resolução os modelos conseguem extrair mais características das imagens. A escolha do número de épocas de treinamento foi limitada à capacidade computacional disponível. A respeito do tamanho dos modelos utilizados e outros dados relevantes a planilha 3.4 os compila.

TABLE 3.4: Tamanho dos modelos utilizados nos experimentos com o dataset Oxford102

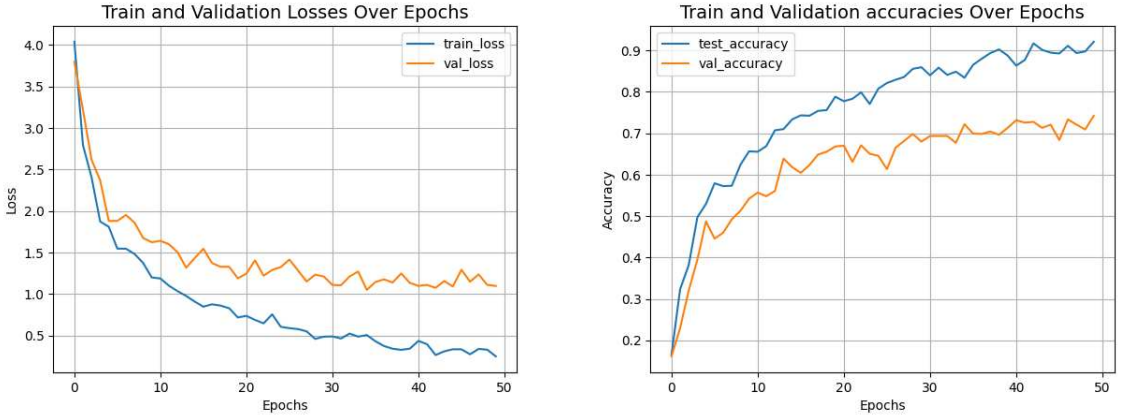
Modelos	nº parâmetros	nº camadas	tamanho das imagens
VGG19	140M	16	(224,224,3)
InceptionResNetV2	77M	164	(224,224,3)
EfficientNetB4	42M	326	(224,224,3)
ViT-B16	85M	12	(224,224,3)

Resultados obtidos a partir dos experimentos são agora apresentados de forma sucinta. As figuras 3.16, 3.17, 3.18 e 3.19 mostram as curvas de acurácia e loss obtidas durante o treinamento, dos modelos VGG19, InceptionResNetV2, EfficientNetB4 e ViT-B16, respectivamente.



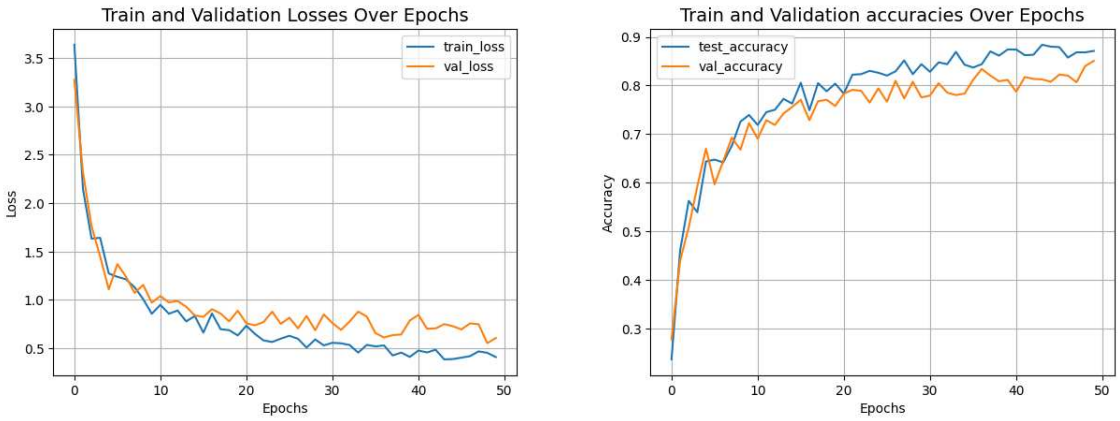
(a) Loss (b) Acurácia

FIGURE 3.16: Evolução das curvas de loss e acurácia do modelo VGG16



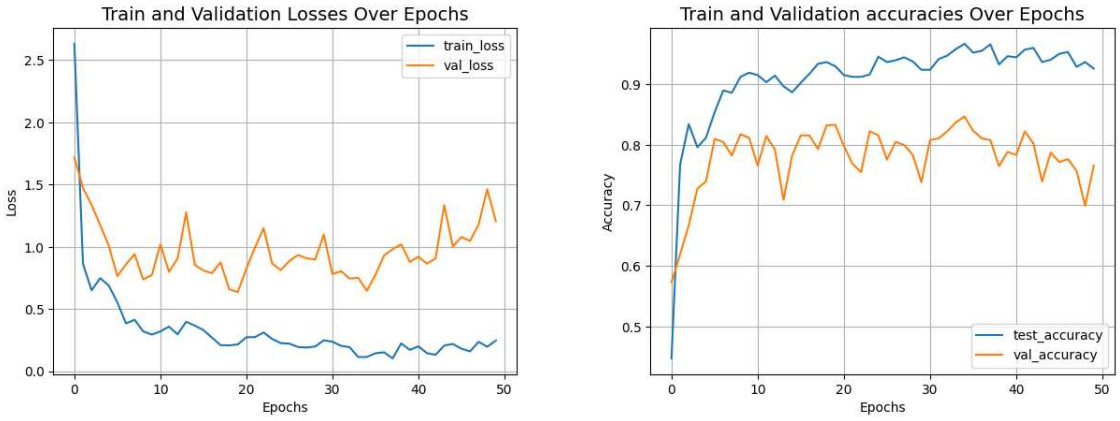
(a) Loss (b) Acurácia

FIGURE 3.17: Evolução das curvas de loss e acurácia do modelo InceptionResNetV2



(a) Loss (b) Acurácia

FIGURE 3.18: Evolução das curvas de loss e acurácia do modelo EfficientNetB4



(a) Loss (b) Acurácia

FIGURE 3.19: Evolução das curvas de loss e acurácia do modelo ViT-B16

A tabela 3.5 agrega os resultados obtidos dos treinamentos dos modelos de classificação. Os dados apresentados são acurácia, loss, F1-Score ponderado e tempo de predição.

TABLE 3.5: Resultado obtidos dos modelos

Modelos	acurácia	loss	F1-Score Ponderado	Tempo de predição
VGG19	63.62%	1.48	62.75%	207ms
InceptionResNetV2	71.17%	1.21	71.09%	174ms
EfficientNetB4	80.58%	0.75	80.11%	157ms
ViT-B16	82.05%	0.80	81.74%	441ms

Chapter 4

Resultados e discussão

Este capítulo tem como objetivo aprofundar a discussão sobre os resultados apresentados no capítulo anterior. Essa análise será dividida por meio dos dois experimentos realizados, analisados separadamente.

4.1 Dataset Autoral

Em primeiro momento podemos afirmar categoricamente que todos os modelos atingiram uma ótima performance, já que durante o treinamento todos obtiveram acurácia com o conjunto de validação de 95%. Podemos ranquear os modelos tendo em vista a acurácia obtida com o dataset de teste: VGG16 (1º), InceptionResNetV2 (2º), Swin Tranformer (3º), EfficientNet (4º) e ViT (5º); mas isso não diz muita coisa, já que o resultado com o dataset de validação nos dá um ranque diferente e a performance dos modelos está muito próxima, dada a simplicidade do dataset.

Podemos analisar os gráficos de evolução da acurácia, pontuando a respeito da velocidade de aprendizagem dos modelos. Ao usar como métrica a época em que cada modelo atinge uma acurácia de 80% com o subconjunto de validação, constrói-se a tabela 4.1, em que o modelo Swin é o 1º colocado disparado, seguido por InceptionResNetV2, Vision Transformer e EfficientNet próximos um do outro, e VGG16 por último longe dos demais modelos. Um dos motivos para essa grande vantagem, no quesito tempo de treinamento, do modelo Swin Tranformer sobre os modelos convolucionais estudados, pode residir no mecanismo de atenção, que permite que cada elemento de entrada se relacione diretamente com todos os outros elementos. Essa estrutura de atenção global permite que o modelo

capture informações de contexto de maneira mais eficiente e eficaz do que as operações locais realizadas pelas convoluções em modelos convolucionais.

TABLE 4.1: Época em que cada modelo atinge 80% de acurácia com dataset de validação

Modelos	Época
Swin Transformer	4
InceptionResNetV2	17
Vision Transformer	20
EfficientNetB0	20
VGG16	29

4.2 Dataset Oxford102

Diferentemente do experimento com o dataset autoral, neste experimento os modelos alcançaram acurácias distintas entre si. Assim, a simples análise da acurácia pode nos revelar a performance de cada modelo, o ranqueamento dos modelos tendo em vista essa métrica está na tabela abaixo 4.2. Cabe pontuar que a métrica de F1-Score, para todos os modelos, esteve muito próxima à acurácia, assim, não sendo relevante para a análise da performance. Este resultado obtido é coerente e esperado, uma vez que ele segue a mesma ordem presente na tabela dos modelos líderes na tarefa de classificação do ImageNet 1.1.

Cabe um ponto de atenção, de que o modelo EfficientNetB4 possui metade da quantidade dos parâmetros do modelo ViT-B16, seria interessante realizar outro experimento com um modelo EfficientNet com tamanho equivalente do ViT, de tal maneira que se compararia com maior honestidade ambos os modelos.

TABLE 4.2: Ranqueamento dos modelos de acordo com a acurácia

Rank	Modelos	Acurácia
1	ViT-B16	82.05%
2	EfficientNetB4	80.58%
3	InceptionResNetV2	71.17%
4	VGG19	63.62%

Podemos analisar os gráficos de evolução da acurácia, pontuando a respeito da velocidade de aprendizagem dos modelos. Ao usar como métrica a época em que cada modelo atinge acurácias de 30%, 60% e 70% com o subconjunto de validação, constrói-se a tabela 4.3. Mais uma vez os modelos estão ranqueados de acordo com esperado, coerentes com a tabela que ranqueia de acordo com as acurácias 4.2.

TABLE 4.3: Épocas em que cada modelo atinge 30%, 60% e 80% de acurácia com dataset de validação

Modelos	Época 30%	Época 60%	Época 70%
ViT-B16	1	2	4
EfficientNetB4	2	5	10
InceptionResNetV2	3	14	35
VGG19	5	18	-

Neste experimento o modelo ViT superou o modelo EfficientNet, diferentemente do primeiro experimento. Podemos atribuir isso ao fato de no primeiro experimento o modelo ViT ter sido implementada "na mão" (a partir de referências de códigos públicos, como o presente no site do Keras [23]), com hiperparâmetros não muito bem escolhidos e com poucos elementos treináveis comparado com o EfficientNet (200 mil parâmetros no modelo ViT e 4.2 milhões no modelo Inception). Modelo EfficientNet este, que não foi implementado "na mão", mas sim baixado da biblioteca pública do Keras.

4.3 Discussão

Os resultados apresentados mostram boa competitividade entre os modelos transformers e modelos convolucionais, mais especificamente o modelo EfficientNet, apresentando uma disputa acirrada quanto à acurácia final. Entretanto um ponto de vantagem dos modelos transformer reside no fato de possuírem um tempo de treinamento acelerado, como pode-se contatar nos gráficos da evolução da acurácia ao longo do treinamento.

Para se escolher entre modelo convolucional e um modelo baseado em self-attention existem fatores além da acurácia e tempo de treinamento. Um possível fator é a existência de modelos pré-treinados, que facilitam muito a vida dos desenvolvedores, poupando horas de treinamento de modelos. Outro possível fator é o caso de uso, que implicará escolher entre um e outro, a seguir seguem diferenças entre as arquiteturas que podem ser melhor utilizadas em contextos específicos.

Uma das diferenças entre as arquiteturas, de acordo com o artigo "How do Vision Transformers Work?"[29], é que a operação de Self-Attention trabalha como filtro passa baixa enquanto a operação de Convolução trabalha como filtro passa alta. A figura 4.1 ilustra como fica uma imagem ao se usar filtros passa alta e passa baixa, também ilustra que tipo de informação que cada uma das operações acima citadas extrai da imagem de entrada, enquanto o filtro passa alta prioriza contornos e formatos o filtro passa baixa prioriza cores e texturas.

FIGURE 4.1: Aplicação de filtros passa alta e passa baixa em uma imagem

Uma característica marcante dos modelos baseados em self-attention é que eles seguem uma arquitetura semelhante aos Transformers usados em tarefas de processamento de linguagem natural, o que facilita a fusão de informações visuais e textuais em um modelo conjunto. Essa capacidade pode ser útil em tarefas que envolvem descrição de imagem, geração de legenda ou qualquer situação em que informações textuais sejam relevantes.

Esse é o caso do modelo CoCa, que tem como objetivo a conversão Imagem-Texto, a figura 4.2 representa esse modelo (estraída do artigo "CoCa: Contrastive Captioners are Image-Text Foundation Models"[30])

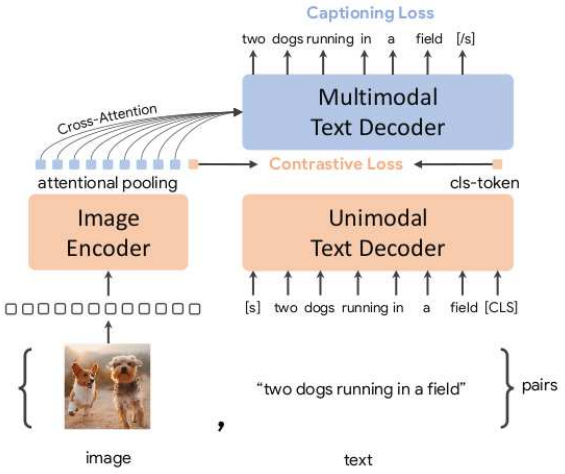


FIGURE 4.2: Representação gráfica do modelo CoCa

Muito se discutiu no âmbito "transformer vs CNN", entretando cabe pontuar que, dentre os melhores modelos ranquados em datasets públicos, há aqueles que usam self-attention e convolução. Um exemplo de modelo híbrido é o CoAtNet, representado na figura 4.3 retirada do artigo "CoAtNet: Marrying Convolution and Attention for All Data Sizes"[31].

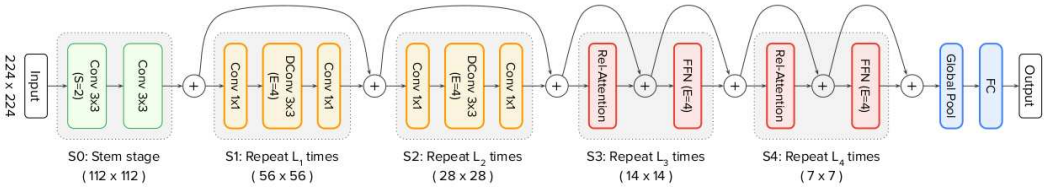


FIGURE 4.3: Representação do modelo CoAtNet

Chapter 5

Conclusão e trabalhos futuros

Neste trabalho, foi realizado um estudo sobre a competitividade entre modelos convolucionais e transformers. No experimento com o dataset Oxford102, os modelos Vision Transformer e EfficientNetB4, sendo um transformer e um convolucional, respectivamente, obtiveram os melhores resultados. No entanto, os resultados do experimento com o dataset autoral não foram conclusivos em relação à capacidade de aprendizagem dos modelos. Porém, foi possível analisar a velocidade de aprendizagem dos modelos, e nesse aspecto, o modelo Swin Transformer foi o mais rápido.

Devido a limitações de poder computacional, não foi viável treinar os modelos com o dataset Oxford102 sem transfer learning, impossibilitando a avaliação do modelo Swin Transformer com esse dataset. Durante a elaboração deste trabalho, não havia um modelo Swin Transformer pré-treinado disponível para uso. No entanto, o autor deste trabalho hipotetiza que se todos os modelos fossem treinados com o dataset Oxford102 sem transfer learning, incluindo o modelo Swin Transformer, ele apresentaria o melhor desempenho tanto em termos de acurácia quanto de tempo de treinamento. No entanto, como esse experimento não foi realizado, não é possível fazer tal afirmação.

Além disso, é importante ressaltar que ao escolher uma arquitetura de classificação de imagens, além da busca pela melhor acurácia, outros aspectos devem ser considerados, como o poder computacional disponível e o domínio da aplicação e do dataset utilizados. Esses fatores desempenham um papel crucial na seleção da arquitetura mais adequada para o problema em questão.

Como possíveis trabalhos futuros, é possível a criação de um aplicativo móvel que sirva de ferramenta para conservação da biodiversidade, tendo como motor um classificador de plantas.

Bibliography

[1] Researchgate.net, "The architecture of vgg16." [Online; accessed May 16, 2023].

[2] paperswithcode.com, "Image classification on imagenet." [Online; accessed May 17, 2023. https://paperswithcode.com/sota/image-classification-on-imagenet].

[3] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[5] A. Z. Karen Simonyan, "Very deep convolutional networks for large-scale image recognition," *ICLR 2015*, 2015.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '16, pp. 770–778, IEEE, June 2016.

[7] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015. cite arxiv:1505.04597Comment: conditionally accepted at MICCAI 2015.

[8] "Reconstruction of porous media from extremely limited information using conditional generative adversarial networks." Online, Available from: https://www.researchgate.net/figure/Commonly-used-activation-functions-a-Sigmoid-b-Tanh-c-ReLU-and-d-LReLU_fig3_335845675 [accessed 25 May, 2023].

[9] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[10] G. E. H. David E. Rumelhart and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, 1986.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (Red Hook, NY, USA), p. 6000–6010, Curran Associates Inc., 2017.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] J. Cordonnier, A. Loukas, and M. Jaggi, "On the relationship between self-attention and convolutional layers," *CoRR*, vol. abs/1911.03584, 2019.

[17] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021.

[18] "*tensor*2*tensornotebook*." Online, Available from: https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb#scrollTo=OJKU36QAfqOC [accessed 25 May, 2023].

[19] "*visualize_attention_map.ipynb*." Online, Available from: *https* : *//github.com/jeonsworld/ViT−pytorch/blob/main/visualize_attention_map.ipynb* [accessed 25 May, 2023].

[20] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *CoRR*, vol. abs/1602.07261, 2016.

[21] M. Mahdianpari, B. Salehi, M. Rezaee, F. Mohammadimanesh, and Y. Zhang, "Very deep convolutional neural networks for complex land cover mapping using multispectral remote sensing imagery," *Remote Sensing*, vol. 10, no. 7, 2018.

[22] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019.

[23] K. Salama, "Image classification with vision transformer." [Online; accessed June 1st, 2023. https://keras.io/examples/vision/image_classification_with_vision_transformer/].

[24] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," *CoRR*, vol. abs/2103.14030, 2021.

[25] R. Dagli, "Image classification with swin transformers." [Online; accessed June 1st, 2023. https://keras.io/examples/vision/swin_transformers/].

[26] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

[27] Researchgate.net, "Some samples in the oxford 102." [Online; accessed May 17, 2023].

[28] TensorFlow, "Data augmentation." [Online; accessed May 23, 2023. tensorflow.org/tutorials/images/data_augmentation].

[29] N. Park and S. Kim, "How do vision transformers work?," in *International Conference on Learning Representations*, 2022.

[30] J. Yu, Z. Wang, V. Vasudevan, L. Yeung, M. Seyedhosseini, and Y. Wu, "Coca: Contrastive captioners are image-text foundation models," 2022.

[31] Z. Dai, H. Liu, Q. V. Le, and M. Tan, "Coatnet: Marrying convolution and attention for all data sizes," *CoRR*, vol. abs/2106.04803, 2021.

[32] F. Chollet *et al.*, "Keras," 2015.

[33] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283, 2016.

[34] Z. Liu, H. Mao, C. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," *CoRR*, vol. abs/2201.03545, 2022.