Projeto Integrado de Matemática e Computação

Classificação de Viaturas em Imagens

Duarte Silva (PG46719)

Bruno Alves (PG46798)

Alexandre Leite (PG42538)

25 de Junho de 2022

Resumo

Ao longo deste projeto foram estudados e comparados dois algoritmos de visão computacional *state-of-the-art*, o ConvNext e o Swin Transformer aplicados a um problema de classificação (marca, modelo e ano) de imagens de veículos. Em ambos os casos, foi utilizada a técnica de *Transfer Learning*, ou seja, ambos os modelos foram pré-treinados com o *dataset* IMAGENET e posteriormente afinados (*finetuning*) com o *dataset* Stanford Cars. Por fim, após a otimização de alguns hiperparâmetros e com a realização de treinos longos (500 épocas), foram obtidos resultados que apresentavam uma acurácia bastante próxima da considerada atualmente *state-of-the-art*.

Agradecimentos

Queremos agradecer aos orientadores envolvidos neste projeto pois, sem eles, não conseguiríamos obter o aproveitamento que tivemos. Num tom mais particular, queremos agradecer à professora Fernanda Costa, pelo seu apoio constante, sempre disposta a fazer sugestões, a inteirar-se sobre o estado do projeto. Queremos também agradecer ao investigador Emanuel Gouveia, orientador do DTx, que desde o dia um sempre nos facultou todas as ferramentas que nós precisávamos, gerando um ambiente propício para realizar este projeto de forma exemplar. Por fim, deixamos um agradecimento à direção do Mestrado em Matemática e Computação da Universidade do Minho e ao DTx por providenciarem uma nova experiência num contexto muito prático, próximo das necessidades do mercado de trabalho.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Projeto Integrado em Matemática e Computação

Este projeto é uma parceria entre a DTx (Digital Transformation CoLab) e o Mestrado em Matemática e Computação, da Universidade do Minho. A Associação Laboratório Colaborativo em Transformação Digital – DTx, é uma associação privada sem fins lucrativos, que desenvolve a sua atividade efetuando investigação aplicada em diferentes áreas associadas à transformação digital. O DTx trabalha na interseção dos domínios físico, digital e cibernético, com o objetivo de criar a próxima geração de sistemas ciber-físicos evoluídos, que seja capaz de esbater a fronteira entre o mundo real e o mundo virtual, encontrando-se sediado no campus de Azurém. O tema deste projeto é "Deteção e/ou classificação de viaturas em imagens RGB (em colaboração com o laboratório colaborativo DTx".

1.2 Contexto

Atualmente, a *Deep Learning* (DL) está a tentar encontrar soluções para problemas em muitas áreas. Uma delas é a segurança. Este tipo de soluções prendem-se com o controlo dos veículos no tráfego rodoviário, recorrendo a itens diversos como deteção de matrículas, de velocidade, da marca, do modelo ou controlo de faixas de rodagem, entre outros. Uma das funções necessárias no trânsito é a deteção de marcas e modelos de veículos. Há várias razões para este controlo. Uma das razões e a mais importante é a segurança no tráfego rodoviário. Alguns dos diferentes tipos de veículos em trânsito podem ser proibidos de utilizar alguns pontos e estradas. Por exemplo: é criado um ponto de controlo. As propriedades dos veículos que atravessam o ponto de controlo são detetadas. De acordo com as características detetadas, a passagem do veículo pode ser

impedida com antecedência. Uma destas razões é a validade dos veículos. Por outras palavras, a marca e o modelo do veículo são detetados através da matrícula, e a marca e o modelo reais podem não corresponder. Da mesma forma, ainda mais comum pode ser a discrepância entre a cor real do veículo e a cor registada do veículo. Especialmente para veículos roubados ou situações ilegais, torna-se mais fácil encontrar tais eventos com mais frequência. Muitos dos problemas podem ocorrer e variar de acordo com as exigências e necessidades das instituições e organizações. A fim de solucionar alguns destes problemas, o objetivo deste estudo é detetar a marca, modelo dos veículos através de métodos de visão computacional (VC) *state of the art*.

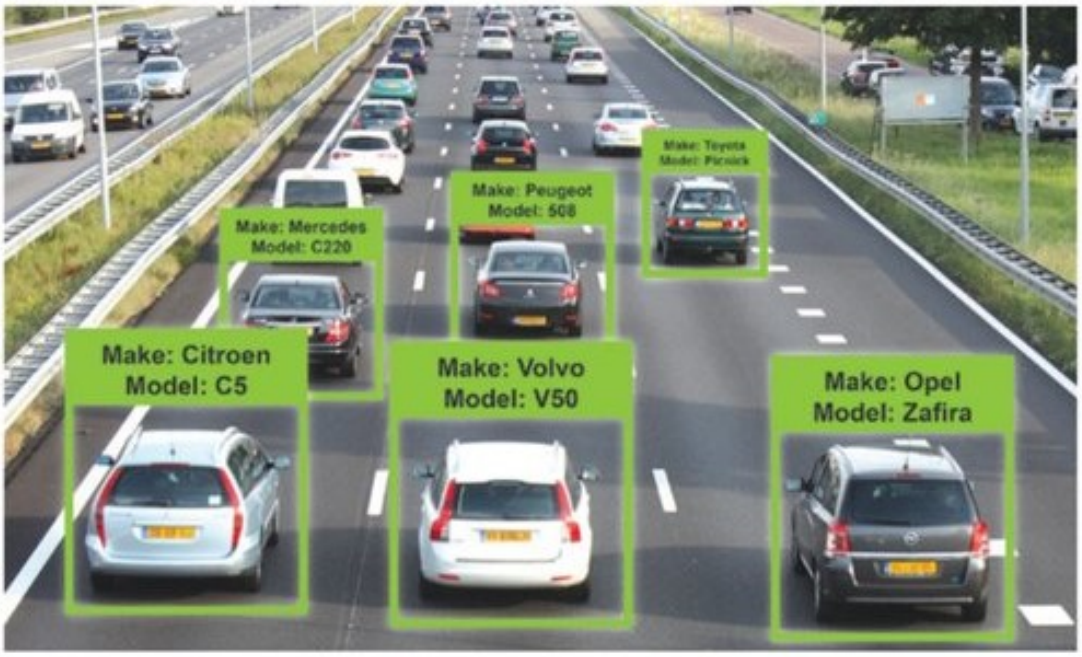


Figura 1.1: Exemplo da aplicabilidade da VC na deteção da marca de carros e modelo.

1.3 Objetivo

Os objetivos deste trabalho prendem-se em:

- Conhecer o estado da arte sobre métodos de Machine Learning aplicados ao problema de classificação de marcas e modelos de veículos automóveis, com recurso a imagens;
- Desenvolvimento e implementação de dois métodos para a classificação de marcas de veículos automóveis: um baseado em CNN e um baseado em atenção, ViT.
- Avaliação e validação dos resultados obtidos pelos dois algoritmos usados.

Capítulo 2

Metodologia

A metodologia desempenha um papel fundamental em qualquer trabalho científico. O DL é considerado uma prática científica, portanto deve incorporar uma metodologia que deve incluir, nomeadamente: a escolha da base de dados a usar; a escolha do(s) algoritmo(s) de otimização a aplicar durante o treino do modelo; o rácio da base de dados (em inglês, *dataset*) que irá ser usado para treino bem como os rácios do *dataset* usados para validação e teste, respetivamente; e as medidas para avaliação do desempenho algoritmos de DL; entre outras.

Esta escolha desempenha um papel fulcral na precisão e fiabilidade dos resultados obtidos, pois estes são influenciados pela abordagem/método de investigação utilizado. No início deste projeto, para além da apresentação dos objetivos do trabalho, foi-nos também disponibilizado o *dataset* que deveria ser utilizado assim como os dois modelos que deveriam ser utilizados para a realização do trabalho para a classificação de marcas de veículos automóveis. A metodologia utilizada para a realização do projeto foi a seguinte:

1. **Estudo dos Modelos**: Para o grupo de trabalho se familiarizar com os modelos sugeridos e a melhor maneira de os utilizar, foi realizado uma pesquisa sobre cada um.

2. **Análise dos dados**. Analisou-se o *dataset* de forma a avaliar a sua qualidade (por exemplo, perceber se existe a presença de dados corrompidos) e descobrir informações úteis, para servir de apoio à tomada de decisões (por exemplo, que transformações serão aplicadas aos dados).

3. **Preparação dos dados**: É necessário certificar de que o *dataset* está num formato correto exigido pelos algoritmos de DL sugeridos. Nesta etapa, há a possível necessidade de fazer alguma formatação específica do *dataset*. Alguns modelos precisam de *features* num formato especial, necessitando assim o *dataset* de algum pré-processamento.

4. **Escolha de métricas**: As métricas são uma componente essencial pois são utilizadas para monitorizar e medir o desempenho de um modelo (durante o treino e testes).

5. **Escolha de hiperparâmetros**: esta escolha influencia a qualidade do modelo final obtido. Assim, nesta etapa é importante, por exemplo, escolher o melhor algoritmo de otimização bem como o melhor valor da *learning rate* a usar; entre outros.

6. **Treinar o algoritmo**. É aqui que o DL tem lugar."Alimenta-se"o modelo com bons dados, limpos, provenientes da etapa 3 tendo em conta a extração de informações das etapas anteriores.

7. **Testar o algoritmo**. É aqui que a informação aprendida na etapa anterior é posta em prática. Analisa-se o desempenho do modelo obtido através do *dataset* de teste.

Capítulo 3

Modelos de Deep Learning (DL)

3.1 Redes Neuronais Profundas aplicadas a Visão por Computador

Uma rede neuronal profunda é uma rede constituída por várias camadas intermédias. Este método de organização dos neurónios por uma série de camadas tem vantagens, pois permite analisar os dados hierarquicamente. As primeiras camadas processam os dados de input *raw*, e cada camada subsequente é capaz de utilizar a informação dos neurónios das camadas anteriores para processar porções cada vez maiores de informação. Por exemplo, se considerarmos uma fotografia de uma zebra como *input*, a primeira camada poderia olhar para os pixeis individualmente, a segunda processa grupos de pixeis, a próxima processa grupos desses grupos de pixeis, e assim sucessivamente. Neste caso, as primeiras camadas poderiam aperceber-se que uns pixeis são mais escuros que outros, camadas intermédias conseguiriam identificar algo que parece-se como um olho num determinado grupo de pixeis, enquanto que camadas ainda mais à frente já conseguiriam identificar a coleção de formas que revelam que a imagem contém uma zebra.

3.1.1 Rede Neuronal Convolucional (CNN)

Neste capítulo será abordado uma técnica de DL designada por convolução. Trata-se de um dos métodos mais populares para classificar, manipular e gerar imagens. Uma rede neuronal em que as camadas de convolução constituem um papel central denomina-se por CNN. A CNN é um modelo aprendizagem profundo fortemente estudado nos últimos anos. Este modelo usa um mecanismo de perceção visual dos seres vivos. O primeiro modelo a ser utilizado foi a LeNet-5, proposta por Yann LeCun et al. em 1989, inspirado na descoberta feita por Kunihiko Fukushima que os neurónios no córtex visual são responsáveis por detetar a luz em campos recetivos.

Existem inúmeras variações de CNN, nas quais a maioria possui geralmente três tipos de camadas altamente conectadas entre si. A camada de convolução é feita de vários *kernels* de convolução que aprendem as representações das características de entradas que geram *features maps* (mapa de características). Cada *feature map* é o resultado da primeira convolução da entrada com um kernel aprendido e seu teste, aplicando nos resultados convolucionais uma função de não linear de elemento a elemento. Em termos matemáticos, o valor da característica, $z^l_{i,j,k}$, em um determinado local (i, j) no $k-ésimo$ feature map da camada $l^{ésima}$, é:

$$z^l_{i,j,k} = w^l_k x^l_{i,j} + b^l_k \tag{3.1}$$

onde w^l_k e b^l_k são o vetor de pesos e o termo *bias*, respetivamente, na $k^{ésima}$ e $l^{ésima}$ camada. Por exemplo, w^l_k é compartilhado entre os $z^l_{:,:,k}$, features maps, que é diferentes das Redes Neuronais Artificiais. Isto teve

vantagem de reduzir a complexidade do modelo e facilitar o treino. Se a função de ativação não linear é chamada a(·), então o valor $a^l_{i,j,k}$ da característica convolucional $z^l_{i,j,k}$ é:

$$a^l_{i,j,k} = a(z^l_{i,j,k}) \tag{3.2}$$

A vantagem da função de activação é que ela introduz não linearidade na CNN, que são boas para detectar características não lineares. Uma das funções de ativação mais usada é a ReLU. A camada de *pooling* (agrupamento) que está geralmente entre duas camadas de convolução, tem o papel de obter invariância ao deslocamento, minimizando as resoluções das *feature maps*. Se a função pooling for dada por *pool()*, então para cada feature map:

$$y^l_{i,j,k} = pool\left(a^l_{m,n,k}\right), \forall (m,n) \in R_{ij} \tag{3.3}$$

com R_{ij} uma vizinhança local em torno da localidade (i, j). As operações de *pooling* podem ser *pooling* médio e *pooling* máximo. Normalmente, os *kernels* nas primeiras camadas detectam recursos de baixo nível, como curvas ou arestas, e os *kernels* nas camadas superiores são ensinados a descodificar recursos mais abstratos. A camada totalmente conectada no final da rede tem o papel de pegar em todos os neurónios da camada anterior e conectá-los a cada neurónio na camada atual, de modo a gerar informações semânticas globais. A camada de saída é a ultima camada de uma rede neuronal convolucional, e geralmente é usado um operador *softmax* ou um *support vector machine*. A função de perda, minimizada durante o treino é dada por:

$$L = \frac{1}{N}\sum_{n=1}^{N}\left[l\left(\theta; x^{(n)}, y^{(n)}, o^{(n)}\right)\right] \tag{3.4}$$

onde N é o número de elementos de entrada-saída $\left(x^{(n)}, y^{(n)}\right), n \in [1, \ldots, N]$ com $x^{(n)}$ o n-ésimo dado de entrada e, $y^{(n)}$ a sua *label*, e $o^{(n)}$ é a saída da CNN. A função de perda l é parametrizada por θ, e mede o erro para cada elemento do dataset, a qual pode ser definida, por exemplo, pela função *cross-entropy*.

3.1.2 Transformer

O Transformer é baseado unicamente em mecanismos de atenção, dispensando a recorrência e as convoluções por completo [Vaswani et al. em *Attention is all you need*, 2017]. Várias experiências mostram que este é superior em qualidade face a RNN (redes neuronais recorrentes) ou CNN, sendo simultaneamente mais paralelizável e exigindo significativamente menos tempo para treinar.

O Transformer segue a arquitetura *encoder-decoder* utilizando a *self-attention* empilhada e camadas ligadas tanto para o codificador como para o descodificador, mostradas nas metades esquerda e direita da figura abaixo, respetivamente. Surgiu aplicado a modelos de linguagem, e depois foi adaptado a imagem no ViT (*vision transformer*), que trata a imagem como uma sequência de *patches*, equivalentes a palavras no modelo de linguagem.

Nos últimos anos, tem-se demonstrado bastante apto para uma variedade de tarefas de visão por computador, com melhorias revolucionárias de desempenho ["an image is worth 16x16 words"de Dosovitskiy, A et al.].

3.2 Swin Transformer

O Swin Transformer (abreviatura de *shifted window transformer*) surge como resposta às falibilidades dos ViT (*vision transformer*) [Liu Zhuang et al. *Swin Transformer: Hierarchical Vision Transformer using*

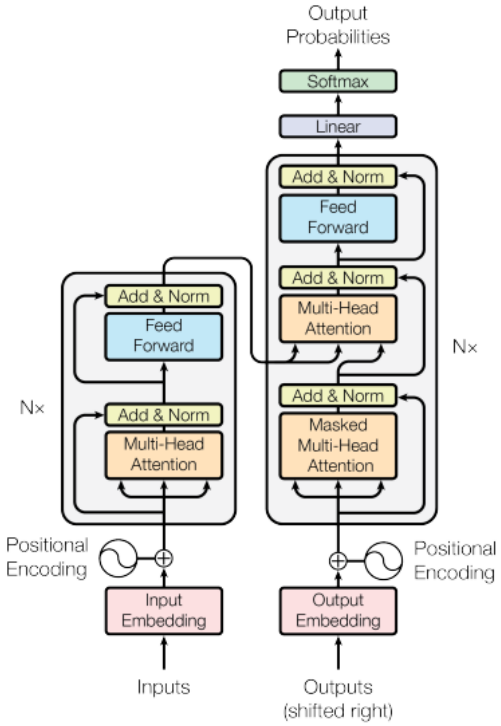


Figura 3.1: Aquitetura de um Transformer

Shifted Windows, 2021].

A primeira camada é qualitativamente a mesma que no ViT - a imagem original é cortada em *patches* e projetada como uma camada linear. A única diferença é que no Swin, na primeira camada, os *patches* têm dimensão 4×4, o que permite um contexto mais pequeno. Depois há várias camadas de *Patch Merging* e *Swin Transformer Block*. O *Patch Merging* concatena *features* de *tokens* e faz *downsize* vizinhos (numa janela 2×2), obtendo uma vista de nível superior. Assim, após cada etapa, são formados "mapas"de *features*, contendo informação a diferentes escalas espaciais, o que apenas permite obter uma representação hierárquica da imagem, que é útil para uma melhor segmentação/deteção de objetos/ etc.

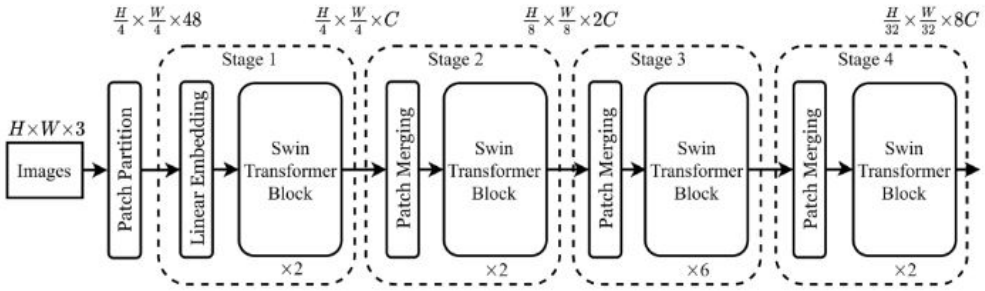


Figura 3.2: Arquitetura Swin Transformer

Isto torna a Swin Transformer uma arquitetura versátil para uma variedade de tarefas de visão computacional.

3.3 ConvNext

Após a modernização de um ResNet padrão, [He, K et al., *Deep Residual Learning for Image Recognition, 2015*, em direção ao projeto de um Transformer, foram descobertas várias componentes-chave que contribuem para a diferença de desempenho ao longo deste caminho. O resultado desta exploração é uma família de modelos puros de ConvNet apelidados de ConvNeXt. Construído inteiramente a partir de módulos ConvNet padrão, os ConvNeXts competem favoravelmente com os Transformers em termos de precisão e escalabilidade, alcançando 87,8% de precisão top-1 do ImageNet e superando os Swin Transformers na deteção COCO e segmentação ADE20K, mantendo a simplicidade e eficiência dos ConvNets padrão.

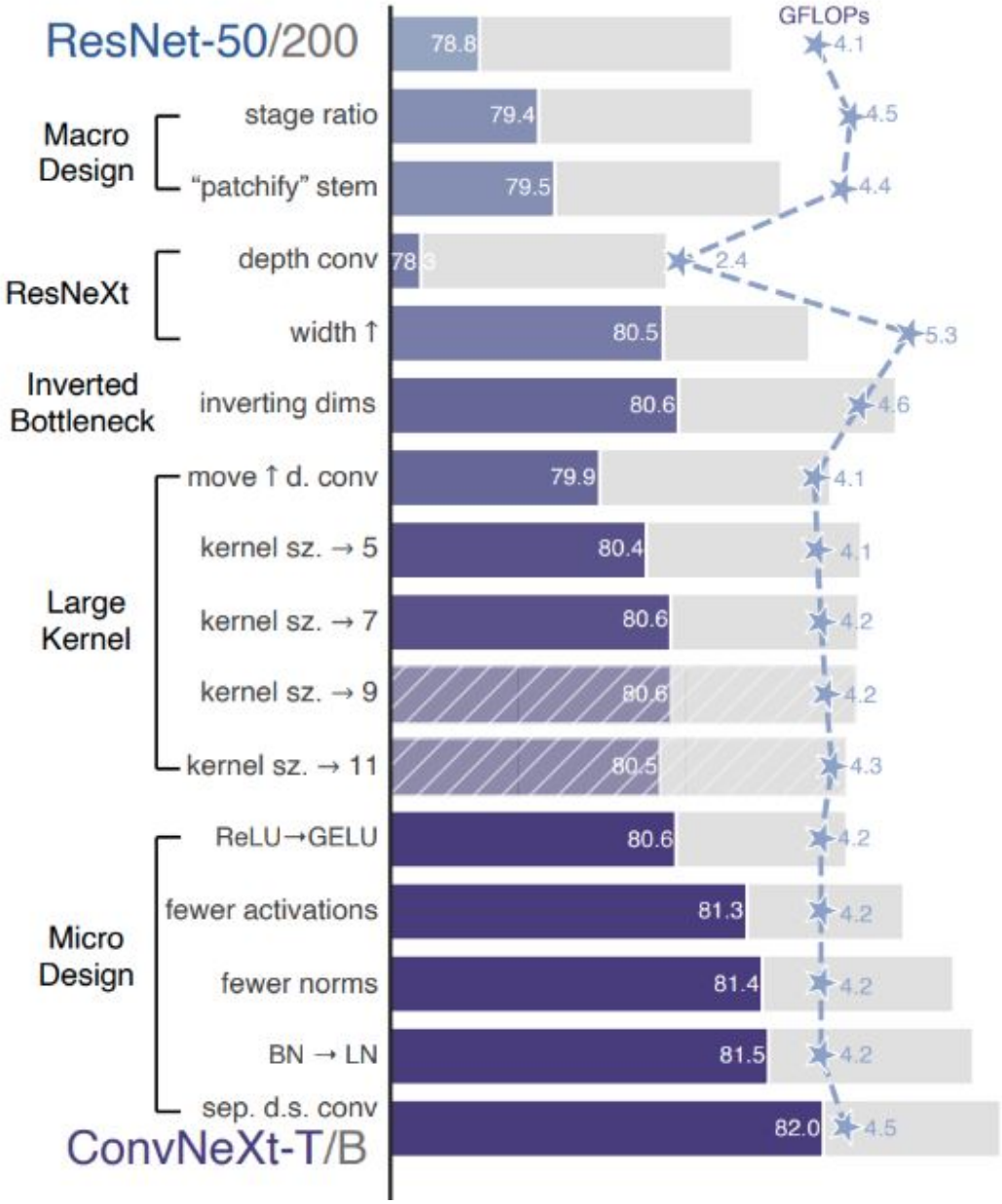


Figura 3.3: Arquitetura ConvNext

Capítulo 4

Stanford Cars Dataset

No âmbito deste projeto foi utilizado o Stanford Cars *dataset* (`http://ai.stanford.edu/~jkrause/cars/car_dataset.html`). Tal como o própio nome indica, este *dataset* é composto por 16185 imagens de carros sendo que se encontra dividido por classes. Cada classe é especificada por marca, modelo e ano do carro (por exemplo, 2012 BMW M3 coupe) sendo que, existem ao todo 196 classes de carros.

Deste *dataset* foram utilizadas 14 567 imagens para treinar os modelos, ficando as restantes 1618 para a realização de testes de forma a que no final do projeto se avalie o desempenho dos modelos obtidos.

4.1 Análise Exploratória dos Dados

De forma a averiguar a qualidade do *dataset*, começou-se inicialmente por analisar se as imagens do *dataset* apresentavam os veículos em posições diversas e com diferentes ângulos de câmara. Para esta análise, foi retirada uma amostra de 30 imagens, sobre a qual se observou uma grande diversidade no que toca às posições dos carros e aos ângulos de câmara das imagens.

Figura 4.1: Alguns carros do Stanford Cars *dataset*.

Após este processo, foi igualmente necessário realizar uma análise quantitativa à distribuição das classes.

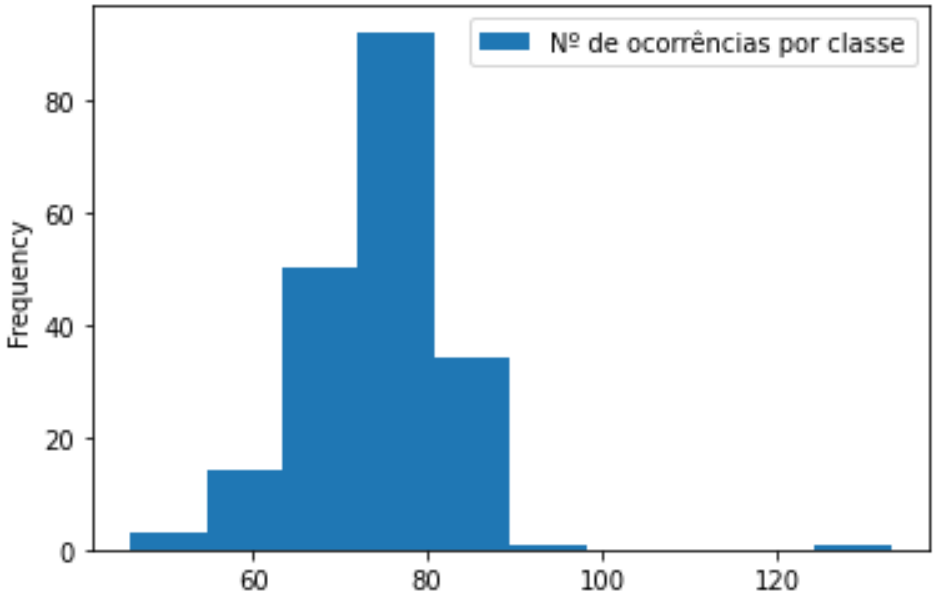


Figura 4.2: Histograma do Nº de Imagens por Classe

Tabela 4.1: Sumário estatístico do número de imagens por classes

Estatística	Valor
Média	74.349
Desvio-padrão	8.479
Coeficiente de variação	0.114
Mínimo	46
Primeiro quartil	70.5
Mediana	76
Terceiro quartil	79
Máximo	133

Pela tabela, observa-se que o valor máximo de imagens por classe é 133, com um mínimo de 46 imagens. Em média, existem cerca de 74 imagens por classe com uma mediana de 76. Ou seja, a mediana é relativamente próxima da média. Observa-se, também um coeficiente de variação reduzido (0.114) sendo indicador de pequena variabilidade no número de imagens por classe, i.e., de equilíbrio em relação à distribuição da quantidade de imagens por classe. A figura ajuda a visualizar esta ideia, mostrando uma clara concentração em torno de 80.

4.2 Tratamento dos Dados

4.2.1 Tratamento de Imagem

De forma a preparar as imagens para o treino com as redes neuronais, foi necessário redimensionar as imagens para 224 por 224 píxeis. Para evitar que ocorra distorção de imagem aquando o redimensionamento das imagens que não sejam quadradas, foi necessário transformar as imagens retangulares em imagens quadradas recorrendo a uma técnica de *padding*. Durante esta transformação, partimos de uma imagem retangular de $n \times m$ píxeis (n $\neq$ m). Seja k igual ao maior entre n e m. Cria-se uma nova imagem de tamanho $k \times k$ píxeis com a imagem original centrada nesta nova imagem. O restante espaço é preenchido com píxeis pretos, ou seja, preenche-se o lado direito e esquerda da imagem quando a imagem apresenta uma altura superior á largura e preenche-se o lado de cima e de baixo da imagem quando a imagem apresenta uma largura superior à altura.

Figura 4.3: 1º Exemplo de Imagem sem *padding*

Figura 4.4: 1º Exemplo de Imagem com *padding*

Figura 4.5: 2º Exemplo de Imagem sem *padding*

Figura 4.6: 2º Exemplo de Imagem com *padding*

4.2.2 Estrutura dos Dados

Ambos os modelos utilizados neste projeto tinham como requisito que os dados apresentassem uma determinada estrutura. Mais em concreto, as imagens precisam de estar divididas por pastas tal como aparece no exemplo da próxima imagem. As pastas *val* e *train* deverão conter respetivamente os dados para validação e para treino. Cada uma dessas pastas contém ainda um total de 196 pastas em que cada pasta corresponde a uma classe contendo dentro dela imagens da respetiva classe.

```
$ tree data
imagenet
├── train
│   ├── class1
│   │   ├── img1.jpeg
│   │   ├── img2.jpeg
│   │   └── ...
│   ├── class2
│   │   ├── img3.jpeg
│   │   └── ...
│   └── ...
└── val
    ├── class1
    │   ├── img4.jpeg
    │   ├── img5.jpeg
    │   └── ...
    ├── class2
    │   ├── img6.jpeg
    │   └── ...
    └── ...
```

Figura 4.7: Estrutura do *dataset*

Do *dataset* disponibilizado, este foi dividido de forma a haver um *dataset* de validação e um *dataset* de treino. Tendo em conta as conclusões retiradas anteriormente sobre a distribuição das classes ser equilibrada tomou-se a decisão que o *dataset* de validação seria formado com 8 imagens de cada classe de forma a que as classes tivessem todas elas o mesmo peso ao se avaliar o desempenho do modelo. Deste modo, o *dataset* de validação ficou constítuido por 1568 (196*8) imagens.

Capítulo 5

Métricas

Precisão, *recall* e acurácia são três métricas para avaliar o desempenho de modelos de classificação com 2 ou mais classes. Essas métricas são baseadas na matriz de confusão. Para classificação binária, a matriz de confusão é uma matriz quadrada de ordem 2 composta por 4 valores:

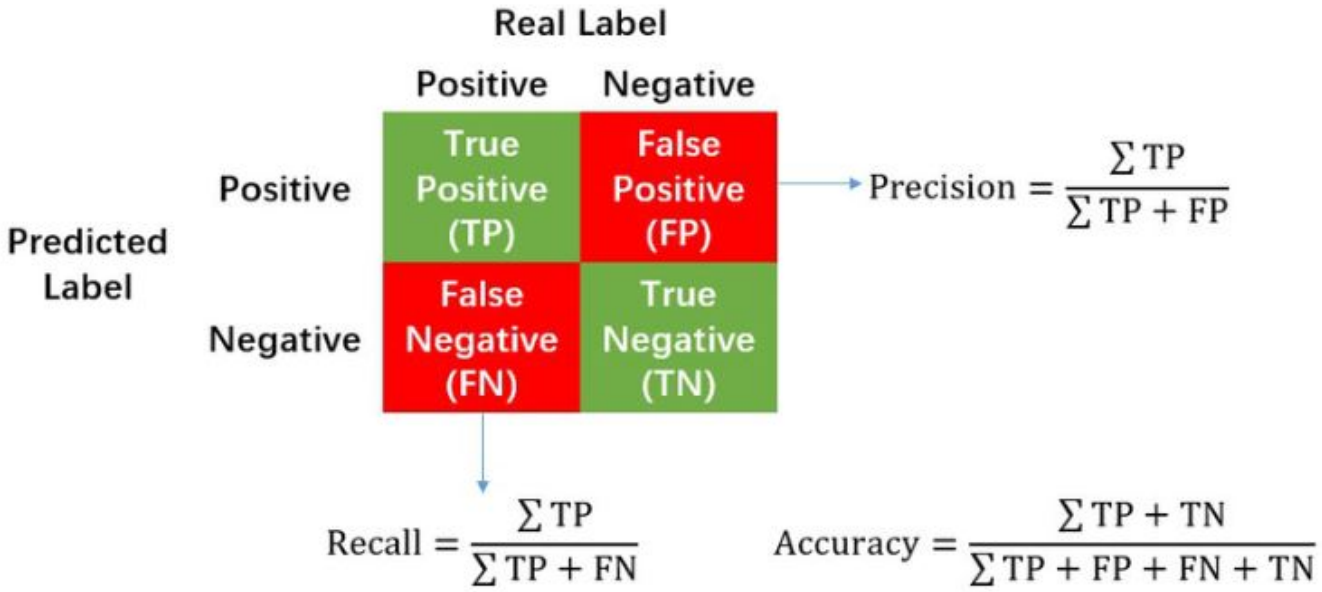


Figura 5.1: Matriz de Confusão

A precisão também é chamada de Valor Preditivo Positivo, correspondendo à taxa de previsões corretas entre as previsões positivas:

$$\frac{TP}{TP + FP} \tag{5.1}$$

Mede a capacidade do modelo de não cometer erros durante uma previsão positiva. O *recall* também é chamado de sensibilidade, taxa de verdadeiro positivo ou taxa de acerto. Corresponde à taxa de indivíduos positivos detetados pelo modelo:

$$\frac{TP}{TP + FN} \tag{5.2}$$

Ele mede a capacidade do modelo de detetar todos os indivíduos positivos.

A acurácia, indica uma performance geral do modelo. Dentre todas as classificações, quantas o modelo classificou corretamente:

$$\frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN} + \mathrm{TN}} \tag{5.3}$$

- TP (Verdadeiros Positivos): classificação correta da classe Positivo;
- FN (Falsos Negativos (Erro Tipo II)): erro em que o modelo previu a classe Negativo quando o valor real era classe Positivo;
- FP (Falsos Positivos (Erro Tipo I)): erro em que o modelo previu a classe Positivo quando o valor real era classe Negativo;
- TN (Verdadeiros Negativos): classificação correta da classe Negativo.

A acurácia é uma boa indicação geral do desempenho do modelo. Porém, pode haver situações em que ela é enganosa. Por exemplo, na criação de um modelo de identificação de fraudes em cartões de crédito, o número de casos considerados como fraude pode ser bem pequeno em relação ao número de casos considerados legais. Para colocar em números, numa situação hipotética de 280 000 casos legais e 2000 casos fraudulentos, um modelo simples que simplesmente classifica tudo como legal obteria uma acurácia de 99,3%. Ou seja, estar-se-ia validando como ótimo um modelo que falha em detetar fraudes.

A precisão pode ser usada em uma situação em que os Falsos Positivos são considerados mais prejudiciais que os Falsos Negativos. Por exemplo, ao classificar uma ação como um bom investimento, é necessário que o modelo esteja correto, mesmo que acabe classificando bons investimentos como maus investimentos (situação de Falso Negativo) no processo. Ou seja, o modelo deve ser preciso nas suas classificações, pois a partir do momento que se considera um investimento bom quando na verdade ele não é, neste caso, uma grande perda de dinheiro pode acontecer.

O *recall* pode ser usado numa situação em que os Falsos Negativos são considerados mais prejudiciais que os Falsos Positivos. Por exemplo, o modelo deve de qualquer maneira encontrar todos os pacientes doentes, mesmo que classifique alguns saudáveis como doentes (situação de Falso Positivo) no processo. Ou seja, o modelo deve ter alto *recall*, pois classificar pacientes doentes como saudáveis pode ser uma tragédia.

De forma a que pudesse ser realizada uma comparação entre diferentes modelos, foi necessário primeiro definir uma métrica. Após algum estudo sobre ambos os modelos utilizados, e as métricas disponíveis, chegou-se à conclusão de que a acurácia seria a melhor métrica para comparar os modelos. Isto porque, se trata de um problema multi-classe em que é indiferente errar na classificação do modelo i como j ou vice-versa, o que torna a acurácia a melhor opção. Contudo, esta necessita de especial cuidado pois pode produzir resultados indevidos, mediante o problema em estudo, como é o caso deste trabalho. A tabela 5.1 esboça a problemática:

Tabela 5.1: Enquadramento de balanceamento e ponderações.

Pesos \ Dataset	Balanceado	Não Balanceado
Sim	Balanceado	Balanceado (***Dataset* de teste**)
Não	Balanceado (***Dataset* de validação**)	Não Balanceado

Dado que neste trabalho houve a decisão do *dataset* de teste não ser balanceado, i.e., a distribuição de elementos de cada classe nesse conjunto não ser representativa mas, por causa disso, a acurácia teria em conta este desequílibrio, compensando-o por via de pesos de tal forma que todas as classes têm contribuição idêntica para o cálculo da acurácia.

Já no conjunto de validação, a decisão foi no sentido contrário, ou seja, assegurou-se que o peso relativo de cada classe no *dataset* total fosse o mesmo no de validação. Por isso, não existe necessidade de alterar a acurácia.
É de referir que, como o problema a resolver é multi-classe, a fórmula para o cálculo da acurácia é dada por:

$$Acc = \frac{\sum_{i=1}^{196} M_{ii}}{N} \tag{5.4}$$

onde M_{ii} são os elementos da diagonal da matriz de confusão $M_{196\times196}$ e N é a cardinalidade do *dataset* de treino.

Capítulo 6

Otimização de Hiperparâmetros

Após a métrica estar definida e considerando que ambos os modelos foram pré-treinados com o *dataset* IMAGENET-22K, contendo 14 000 000 imagens e 21841 classes, que inclui objetos, animais, pessoas, etc. É então necessário otimizar os modelos de forma a que possam obter os melhores resultados possíveis. Uma vez que o treino destes modelos é bastante demorado devido à complexidade do modelo e ao tamanho do *dataset* foi necessário escolher quais os hiperparâmetros que seriam otimizados de forma a reduzir ao máximo o número de treinos a serem realizados para a escolha dos hiperparâmetros.

Após bastante ponderação sobre quais seriam os melhores hiperparâmetros a otimizar, foi decidido que a otimização de hiperparâmetros se iria focar na escolha do *Algoritmo de Otimização* e nos valores do *Learning Rate*.

6.1 Otimizadores

Utiliza-se algoritmos de otimização para treinar modelos de DL. Eles são as ferramentas que permitem obter os melhores valores para os parâmetros do modelo que minimizam o valor da função de perda, durante o processo de treino.

Assim, os algoritmos de otimização são importantes para o DL. Por um lado, a formação de um modelo complexo de DL pode levar horas, dias, ou mesmo semanas. O desempenho do algoritmo de otimização afeta diretamente a eficácia do modelo. Por outro lado, a compreensão dos princípios dos diferentes algoritmos de otimização, e o papel dos seus hiperparâmetros, permitirá afinar os hiperparâmetros de uma forma orientada, para melhorar o desempenho destes. Nesta secção, explora-se com alguma profundidade algoritmos usados neste trabalho. Nesta subsecção, apresenta-se uma breve descrição de alguns dos algoritmos de otimização que foram usados neste trabalho.

6.1.1 Adam

A *Adaptive Moment Estimation* (Adam) é um algoritmo para a técnica de optimização da descida do gradiente. O método é realmente eficiente quando se trabalha com grandes problemas, que envolvem muitos dados ou parâmetros, ao requerer menos memória. Intuitivamente, é uma combinação do algoritmo de descida de gradiente com momento (impulso) e o algoritmo RMSProp. O Adam envolve uma combinação de duas metodologias de descida do gradiente:

1. Momento:

   Este algoritmo é utilizado para acelerar o algoritmo de descida de gradientes tendo em consideração a

"média exponencialmente ponderada"dos gradientes. A utilização de médias faz o algoritmo convergir para o mínimo a um ritmo mais rápido.

$$w_{t+1} = w_t - \alpha m_t \tag{6.1}$$

$$m_t = \beta m_{t-1} + (1-\beta)\left[\frac{\partial L}{\partial w_t}\right] \tag{6.2}$$

onde

- m_t : gradientes agregados no tempo t (m_t inicializado a 0);
- m_{t-1} : gradientes agregados em t-1;
- w_t : pesos em t;
- w_{t+1} = pesos em t+1;
- α_t = taxa de aprendizagem em t;
- $\dfrac{\partial L}{\partial w_t}$ = derivada da *loss* em função a w_t;
- β = parâmetro da média móvel (maior parte das vezes é 0.9).

2. **RMSProp**

RMSprop é um algoritmo de aprendizagem adaptativa que tenta melhorar o AdaGrad, ver [*Adagrad stepsizes: Sharp convergence over nonconvex landscapes*, PMLR, 2019]. Em vez de tomar a soma acumulada de gradientes quadráticos como no AdaGrad, toma a "média móvel exponencial".

$$w_{t+1} = w_t - \frac{\alpha_t}{(v_t+\varepsilon)^{1/2}}\left[\frac{\partial L}{\partial w_t}\right] \tag{6.3}$$

$$v_t = \beta v_{t-1} + (1-\beta)\left[\frac{\partial L}{\partial w_t}\right]^2 \tag{6.4}$$

onde (os restantes "ingredientes do algoritmo" já se encontram definidos anteriormente):

- v_t: soma do quadrado dos gradientes anteriores, i.e., soma de $\left[\dfrac{\partial L}{\partial w_{t-1}}\right]^2$ (inicializado a 0);
- ϵ: constante positiva (usualmente 10^{-8}).

Uma vez que tanto o m_t como o v_t se inicializaram como 0 (com base nos métodos acima referidos), observa-se que ambos são enviesados para 0. Logo, para corrigir este problema, são sugeridas as seguintes fórmulas:

$$w_{t+1} = w_t - \widehat{m}_t\left(\frac{\alpha}{\sqrt{\hat{v}_t}+\varepsilon}\right) \tag{6.5}$$

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{6.6}$$

$$\widehat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{6.7}$$

onde

- β_1 e β_2: taxas de decaimento das médias de gradientes nos dois métodos acima referidos, respetivamente.

6.1.2 NAdam

Nadam é um acrónimo para Nesterov Adam. A componente Nesterov, no entanto, é uma modificação mais eficiente do que sua implementação original. O método Adam também pode ser escrito como:

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \left(\beta_1 \hat{m}_{t-1} + \frac{1 - \beta_1}{1 - \beta_1^t} \cdot \frac{\partial L}{\partial w_t} \right) \tag{6.8}$$

O NAdam usa Nesterov para atualizar o gradiente um passo à frente, substituindo o $\hat{m}_{t-1}$ anterior na equação acima pelo $\hat{m}_t$ atual:

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \left(\beta_1 \hat{m}_t + \frac{1 - \beta_1}{1 - \beta_1^t} \cdot \frac{\partial L}{\partial w_t} \right) \tag{6.9}$$

onde

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{6.10}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{6.11}$$

e

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \frac{\partial L}{\partial w_t} \tag{6.12}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left[\frac{\partial L}{\partial w_t} \right]^2 \tag{6.13}$$

com m_t e v_t inicializados em 0.

6.1.3 RAdam

O método RAdam (Adam retificado) proporciona um ajustamento automático e dinâmico à taxa de aprendizagem adaptativa com base no seu estudo detalhado dos efeitos da variância e do impulso durante o treino. Introduz um retificador, r_t, que por meio de p_t (variância em t) vai retificar a variância.

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \cdot r_t \hat{m}_t \tag{6.14}$$

onde

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{6.15}$$

$$\hat{v}_t = \begin{cases} \frac{v_t}{1-\beta_2^2} & se \quad p_t > 4 \\ 1 & caso\ contrário \end{cases} \tag{6.16}$$

$$r_t = \begin{cases} \sqrt{\frac{(p_t-4)(p_t-2)p_\infty}{(p_\infty-4)(p_\infty-2)p_t}} & se \quad p_t > 4 \\ 1 & caso\ contrário \end{cases} \tag{6.17}$$

e

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \frac{\partial L}{\partial w_t} \tag{6.18}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left[\frac{\partial L}{\partial w_t} \right]^2 \tag{6.19}$$

$$p_t = p_\infty - \frac{\beta_2^t}{1-\beta_2^t} \cdot 2t \tag{6.20}$$

$$p_\infty = \frac{2}{1-\beta_2} - 1 \tag{6.21}$$

6.1.4 AdamW

Esta versão do Adam, com *decoupled weight decay* (decaimento dos pesos desassociados, em tradução livre). Tem este nome pois além de calcular o passo de atualização dos pesos ele adicionalmente decai o peso, recorrendo a λ. Note-se que isto é diferente de adicionar a regularização L2 nas variáveis da *loss*: regulariza variáveis com grandes gradientes mais do que a regularização L2, mostrando produzir uma melhor *loss* de treino.

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t - \lambda w_t \tag{6.22}$$

onde

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{6.23}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{6.24}$$

e

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w_t} \tag{6.25}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\partial L}{\partial w_t}\right]^2 \tag{6.26}$$

6.1.5 AdamP

Esta versão do Adam baseia-se na filosofia do SGDP. O procedimento é o seguinte:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w_t} \tag{6.27}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\partial L}{\partial w_t}\right]^2 \tag{6.28}$$

com m_t e v_t inicializados em 0.
Define-se um p_t:

$$p_t = \frac{m_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{6.29}$$

A atualização dos pesos é feita da seguinte forma:

$$w_{t+1} = \begin{cases} w_t - \Pi_{w_t}(p_t) & se \quad w_t \frac{\partial L}{\partial w_t} < \delta \\ w_t - \alpha p_t & caso\ contrário \end{cases} \tag{6.30}$$

onde ϵ, δ, α são constantes positivas, em que α é a *learning rate*. $\Pi_{w_t}(p_t)$ denota a projeção de p_t no espaço dos w_t.

6.1.6 Adadelta

O Adadelta é uma melhoria do método AdaGrad (não abordado no trabalho), com foco na componente de taxa de aprendizagem. Adadelta é provavelmente a abreviação de "delta adaptativo", onde delta aqui se refere à diferença entre o peso atual e o peso recém-atualizado, i.e., $w_t - w_{t-1}$. Ao contrário do RMSprop, o Adadelta remove completamente o uso do parâmetro de taxa de aprendizagem, substituindo-o por D, a média móvel exponencial dos deltas quadrados.

$$w_{t+1} = w_t - \frac{\sqrt{D_{t-1} + \epsilon}}{\sqrt{v_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t} \tag{6.31}$$

onde

$$D_t = \beta D_{t-1} + (1 - \beta) \left[\Delta w_t\right]^2 v_t = \beta v_{t-1} + (1 - \beta) \left[\frac{\partial L}{\partial w_t}\right]^2 \tag{6.32}$$

com D e v inicializado em 0 , e

$$\Delta w_t = w_t - w_{t-1} \tag{6.33}$$

6.1.7 Adafactor

O AdaFactor é uma resposta à pouca eficiência, em termos de memória, que métodos como o AdaDelta, Adam ou RMSProp apresentam:

$$\alpha_t = \max\left(\epsilon_2, RMS\left(w_{t-1}\right)\right) \rho_t \tag{6.34}$$

$$g_t = \frac{\partial L}{\partial w_{t-1}} \tag{6.35}$$

$$\hat{v}_t = \hat{\beta}_2^t \hat{v}_{t-1} + \left(1 - \hat{\beta}_2^t\right) \left(g_t^2 + \epsilon_1 1_n\right) \tag{6.36}$$

$$u_t = \frac{g_t}{\sqrt{\hat{v}_t}} \tag{6.37}$$

$$\hat{u}_t = \frac{u_t}{\max\left(1, \frac{RMS(u_t)}{d}\right)} \tag{6.38}$$

$$w_t = w_{t-1} - \alpha_t \hat{u}_t \tag{6.39}$$

onde, usualmente, $\rho_t = \min\left(10^{-2}, \frac{1}{t}\right)$, $\epsilon_1 = 10^{-30}$, $\epsilon_1 = 10^{-3}$, $d = 1$ e $\hat{\beta}_2^t = 1 - t^{-0.8}$. RMS denota o valor quadrático médio, que no caso $RMS(u_t) = \sqrt{m\left(\frac{g_t^2}{\hat{v}_t}\right)}$, onde $m(\cdot)$ denota média. α_t, tal como tem sido usual, é o *learning rate*.

6.1.8 NovoGrad

O NovoGrad é um algoritmo estocástico adaptativo de descida do gradiente, tal como as várias variantes Adam, muito semelhante ao AdamW. A grande diferença centra-se no facto de operar em termos de camadas. (Note-se que l indica que a atualização do peso é feita por camada.)

$$w_{t+1}^{(l)} = w_t^{(l)} - \alpha m_t^{(l)} \tag{6.40}$$

onde

$$m_t^{(l)} = \beta_1 m_{t-1}^{(l)} + (1-\beta_1)\left(\frac{1}{\sqrt{v_t^{(l)}}+\epsilon}\cdot\frac{\partial L}{\partial w_t}^{(l)} + \lambda w_t^{(l)}\right) \tag{6.41}$$

e

$$v_t^{(l)} = \beta_2 v_{t-1}^{(l)} + (1-\beta_2)\left[\frac{\partial L}{\partial w_t}^{(l)}\right]^2 \tag{6.42}$$

6.1.9 SGD

Embora o método do gradiente estocástico de descida (SGD) esteja presente na comunidade de aprendizagem de máquina há muito tempo, recebeu recentemente uma atenção considerável no contexto do DL. A atualização dos pesos é dada por:

$$w_{t+1} = w_t - \alpha\frac{\partial L}{\partial w_t} \tag{6.43}$$

onde α é a *learning rate*.

6.1.10 SGDP

Esta versão do SGDP acrescenta ao SGD uma componente de projeção, que regulariza o crescimento normal induzido pelo momento e melhora os desempenhos do modelo. Esta ideia surgiu pois notou-se que, quando os pesos são invariantes quanto à escala (maior parte das situações em DL), nota-se um crescimento excessivo das normas deles durante o treino.

$$w_{t+1} = w_t - \alpha p_t \tag{6.44}$$

com

$$p_t = \beta p_{t-1} + \frac{\partial L}{\partial w_t} \tag{6.45}$$

e com a atualização dos pesos:

$$w_{t+1} = \begin{cases} w_t - \Pi_{w_t}(p_t) & se \quad w_t\frac{\partial L}{\partial w_t} < \delta \\ w_t - \alpha p_t & caso\ contrário \end{cases} \tag{6.46}$$

onde δ, β, α são constantes positivas, em que α é a *learning rate* e β é o momento. $\Pi_{w_t}(p_t)$ denota a projeção de p_t no espaço dos pesos w_t.

6.2 Learning Rate

Até agora, houve um foco principalmente nos algoritmos de otimização, de como calcular a direção de procura e atualizar os vetores de peso, a qual envolve a taxa de aprendizagem. O cálculo da taxa de aprendizagem apropriado é importante pois permitirá que o algoritmo possa convergir mais rapidamente. Assim, para o cálculo da taxa de aprendizagem há série de aspetos a ter em consideração:

- a **magnitude** da taxa de aprendizagem é importante. Se for demasiado grande, a otimização diverge, se for demasiado pequena, leva demasiado tempo a treinar e pode não convergir para uma solução ótima.
- Nos caos em que se usa uma **taxa de decaímento** esta é igualmente importante. Pois se a taxa de aprendizagem continuar a ser grande a otimização diverge, e se for demasiado pequena, leva muito tempo a treinar e pode não convergir para uma solução ótima.
- Outro aspeto igualmente importante é a sua **inicialização**. Isto diz respeito tanto à forma como os parâmetros são definidos inicialmente como também à sua evolução inicial. Esta questão é monitorizada pelo *warmup* (aquecimento), ou seja, a rapidez com que se começa a avançar para a solução inicialmente. Grandes passos no início podem não ser benéficos, em particular porque o conjunto inicial de parâmetros é aleatório. As instruções iniciais de atualização podem também não ter qualquer significado;
- Por último, é de referir que alguns métodos de otimização efetuam um ajustamento cíclico da taxa de aprendizagem .

Dado o facto de haver muitos detalhes necessários para gerir as taxas de aprendizagem, a maior parte do *software* tem ferramentas para lidar com isto automaticamente.

Por motivos de poupança de poder computacional, foi decidido que o processo de otimização do *Learning Rate* passaria por se escolher o valor ótimo para o seu expoente do multiplicador de base 10 da sua magnitude. Devido a possuírem bases diferentes por predefinição, no caso do ConvNext este valor é multiplicado por 5 enquanto que no Swin Transformer é multiplicado por 2.

Após alguma pesquisa, foi decidido que os modelos seriam testados com *Learning Rate* igual a 10^{-4}, 10^{-5} e 10^{-6} uma vez que, seria nessa gama de valores que se provavelmente encontra o valor ótimo para o *learning rate*. De acrescentar que ambos os modelos têm um decaímento de acordo com um *cosine schedule*. Quanto à inicialização, como se partiu de modelos pré-treinados, não iniciámos com parâmetros aleatórios, pelo que o *warmup* pode ser dispensado e a magnitude do *learning rate* será menor.

6.3 Análise de Resultados

Ao se realizar a otimização dos hiperparâmetros para o ConvNext foram testados 13 algoritmos de otimização (SGD, SGDP, RMSProp, RMSPropTf, AdaFactor, AdaDelta, Adam, Nadam, RAdam, AdamW, AdamP, NovoGrad e NvNovoGrad). Tal como mencionado anteriormente, para cada um dos algoritmos de otimização foram testados 3 valores diferentes de *learning rate*. Ao todo foram testadas 39 combinações de hiperparâmetros sendo que para cada combinação foi realizado um treino de 100 épocas, onde época denota N avaliações consecutivas, em que N a cardinalidade do *dataset* usado.

Devido ao peso computacional que este processo de otimização requer ser bastante elevado, foi necessário correr os testes remotamente via SSH numa máquina com elevado poder computacional cujo acesso foi disponibilizado pelo DTx. Apesar da utilização deste *hardware*, e após a escolha de um tamanho de *batch* de 32, o máximo permitido pela memória do sistema, cada época demorava em média 90 segundos, ou seja, o tempo total de treino para cada configuração de hiperparâmetros era cerca de 2 horas e meia. Por consequência, o tempo total de treino para o ciclo de otimização proposto era aproximadamente 100 horas. Deste modo, foi necessário construir *scripts* de forma a automatizar os treinos dos modelos. Estas scripts ficavam depois a correr em *"background"* graças ao comando nohup.

Os seguintes gráficos mostram a evolução da acurácia no *dataset* de validação, em percentagem, ao longo das épocas para o modelo ConvNeXt:

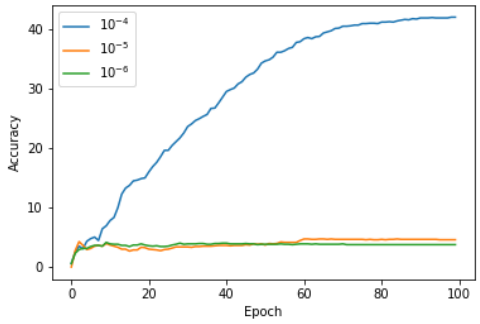


Figura 6.1: SGDP ConvNext

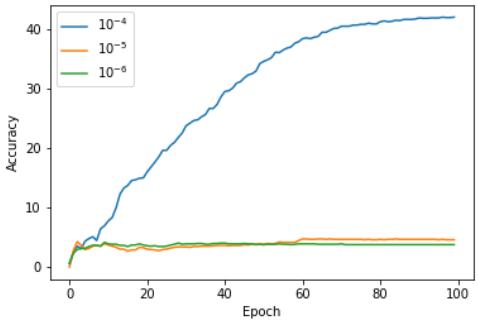


Figura 6.2: SGD ConvNext

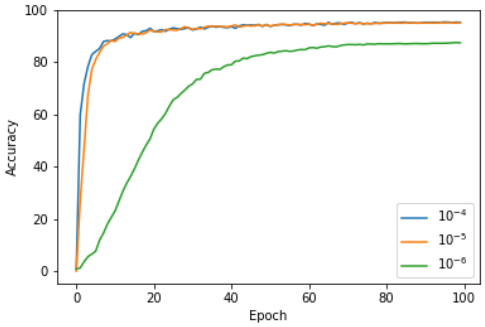


Figura 6.3: RMSPropTf ConvNext

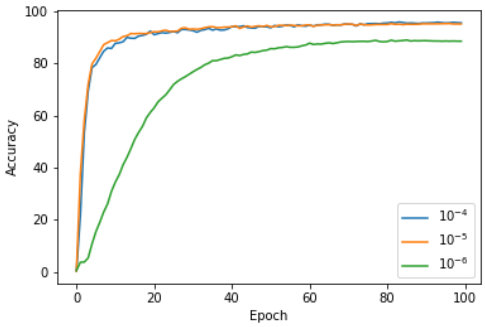


Figura 6.4: RMSProp ConvNext

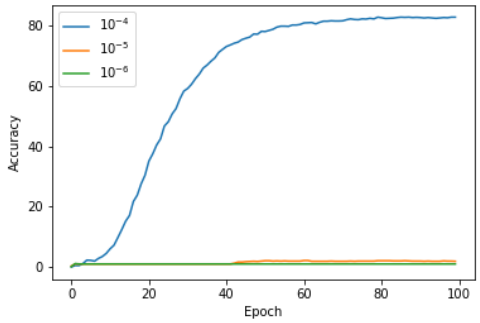


Figura 6.5: NovoGrad ConvNext

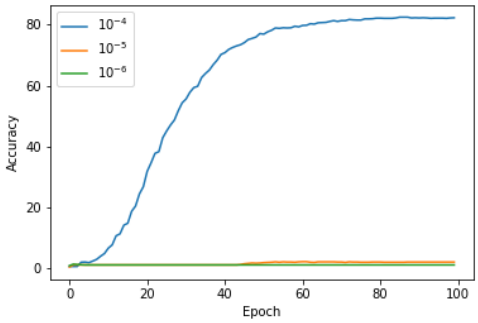


Figura 6.6: NvNovoGrad ConvNext

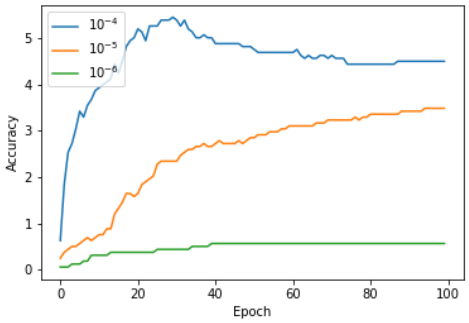


Figura 6.7: AdaDelta ConvNext

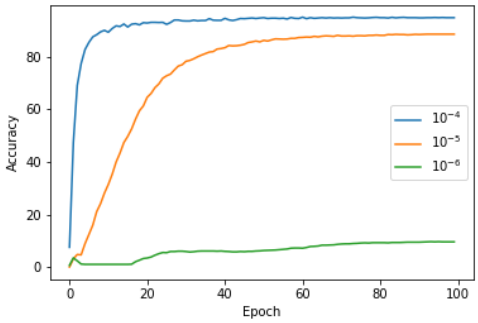


Figura 6.8: AdaFactor ConvNext

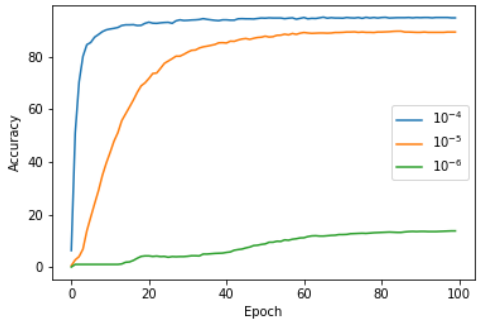


Figura 6.9: Adam ConvNext

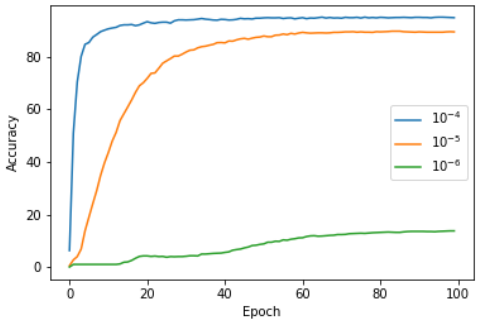


Figura 6.10: AdamW ConvNext

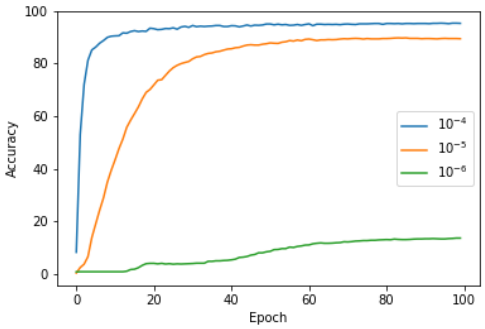


Figura 6.11: AdamP ConvNext

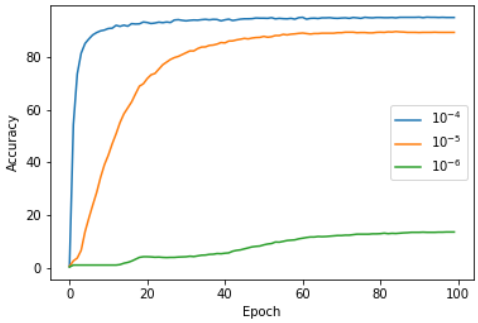


Figura 6.12: NAdam ConvNext

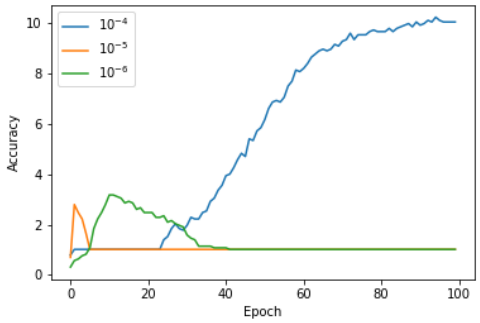


Figura 6.13: RAdam ConvNext

No apêndice B, foram também produzidos gráficos que mostram a variação do valor da *Loss* ao longo dos treinos de 100 épocas.

Após a visualização destes gráficos e após consultar a tabela de resultados (Anexo A), foi possível concluir que relativamente ao *learning rate* que para todos os otimizadores utilizados, os melhores resultados foram obtidos para a ordem de grandeza de 10^{-4} enquanto que 10^{-6} apresentou os piores resultados.

Relativamente à escolha do algoritmo de otimização, apenas com a consulta dos gráficos acima é possível descartar algumas opções mas não é possível dizer qual o melhor. Deste modo, foi criado um novo gráfico com as 5 melhores configurações de hiperparâmetros. Ainda assim, o comportamento da acurácia em função das épocas era bastante similar entre as diversas combinações de hiperparâmetros. Tendo isto em conta, produziu-se um novo gráfico olhando apenas para valores altos de acurácia com uma escala bastante reduzida.

Estes novos gráficos com a comparação das cinco melhores combinações de hiperparâmetros podem ser consultados na página seguinte.

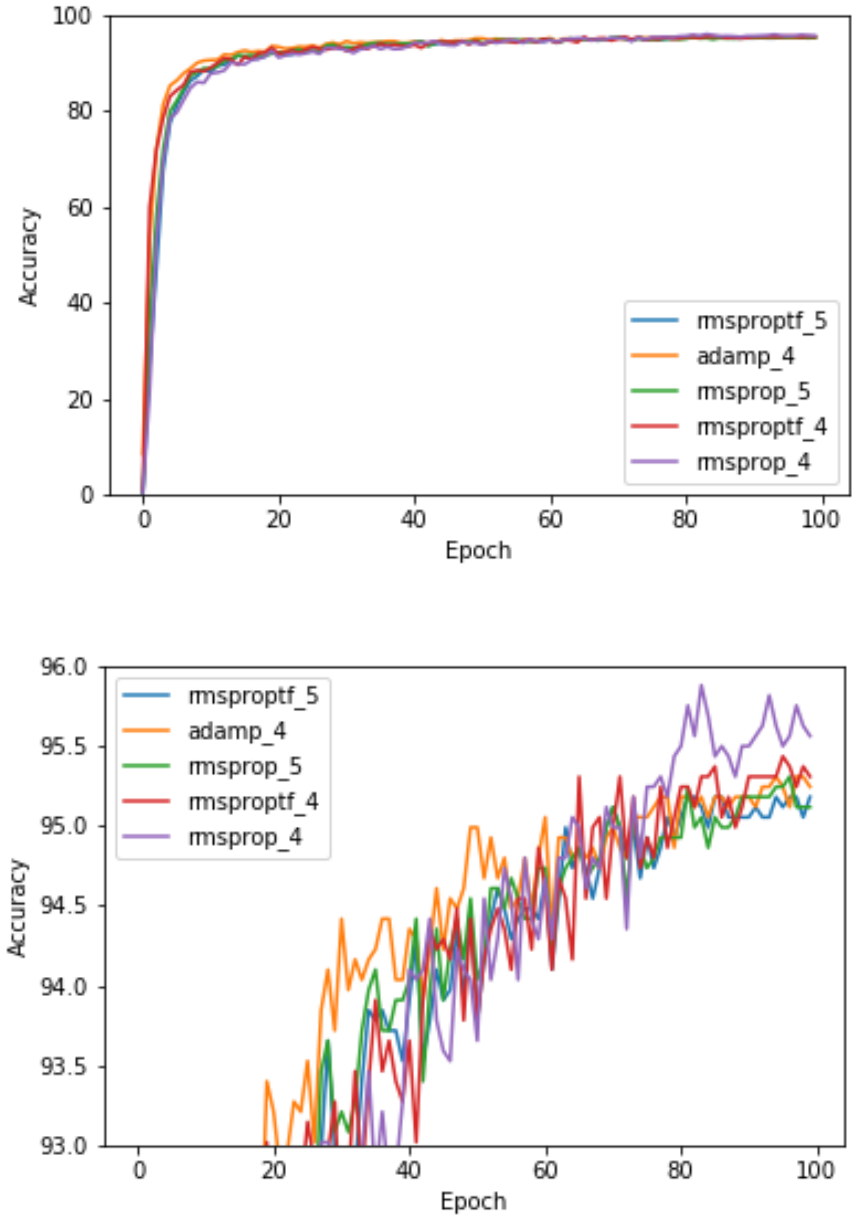


Figura 6.14: Top 5 Otimizadores para ConvNext

Apesar de terem um comportamento bastante similar, foi decidido que se escolheria a combinação dos hiperparâmetros que apresentasse a acurácia mais alta. Como tal, para o modelo do ConvNext será utilizado o algoritmo de otimização RMSProp e uma ordem de grandeza do *learning rate* de -4.

Após a escolha de hiperparâmetros para o ConvNext, foi necessário repetir todo o processo para se escolher os hiperparâmetros para o Swin Transformer. Com a experiência adquirida na otimização de hiperparâmetros do ConvNext relativamente ao *learning rate*, foi tomada a decisão de não serem realizados testes para o valor de *learning rate* na ordem de grandeza de -6. Foi realizado um teste com ConvNext a utilizar o RMSProp e uma ordem de grandeza de -3 para o *learning rate* para se averiguar se valeria a pena realizar testes com esse valor. Devido ao resultados poucos motivadores apresentados no fim das 100 épocas e tendo em

mente a diminuição do custo computacional do processo de otimização de hiperparâmetro foi decidido não se considerar -3 como um valor possível para a ordem de grandeza do *learning rate*.

Em relação aos algoritmos de otimização, ao contrário do ConvNext o Swin Transformer apenas disponibiliza como opção o SGD e o Adam. Como tal, foi necessário alterar diretamente o código do Swin Transformer de forma a que se pudesse escolher outros otimizadores. Assim, foram então acrescentados os algoritmos RMSProp, AdamW, NAdam e RAdam uma vez que tinham apresentado bons resultados com o ConvNext.

Os seguintes gráficos mostram a evolução da acurácia ao longo das épocas para o modelo Swin Transformer:

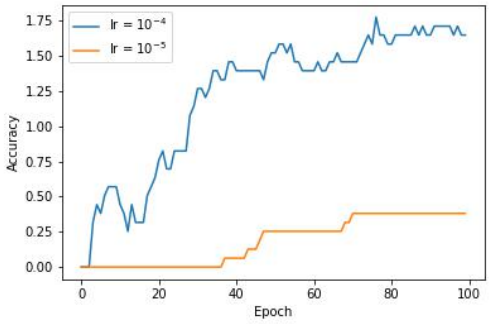


Figura 6.15: SGD Swin Transformer

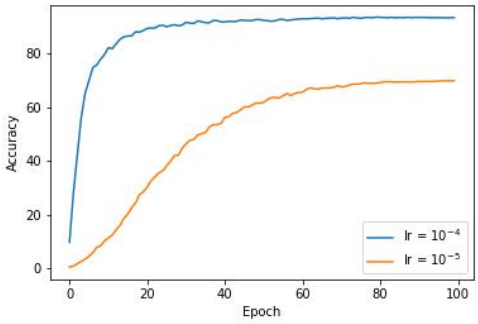


Figura 6.16: RMSProp Swin Transformer

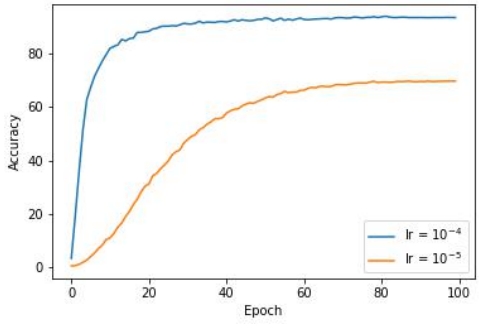


Figura 6.17: Adam Swin Transformer

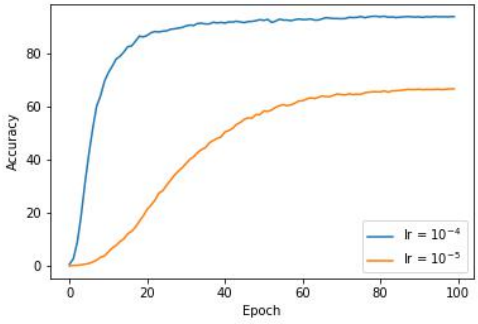


Figura 6.18: RAdam Swin Transformer

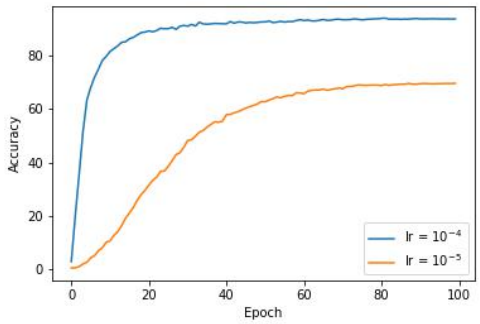


Figura 6.19: AdamW Swin Transformer

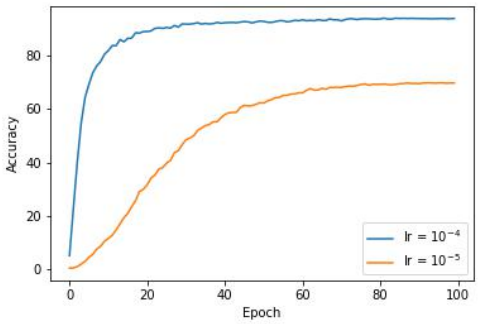


Figura 6.20: NAdam Swin Transformer

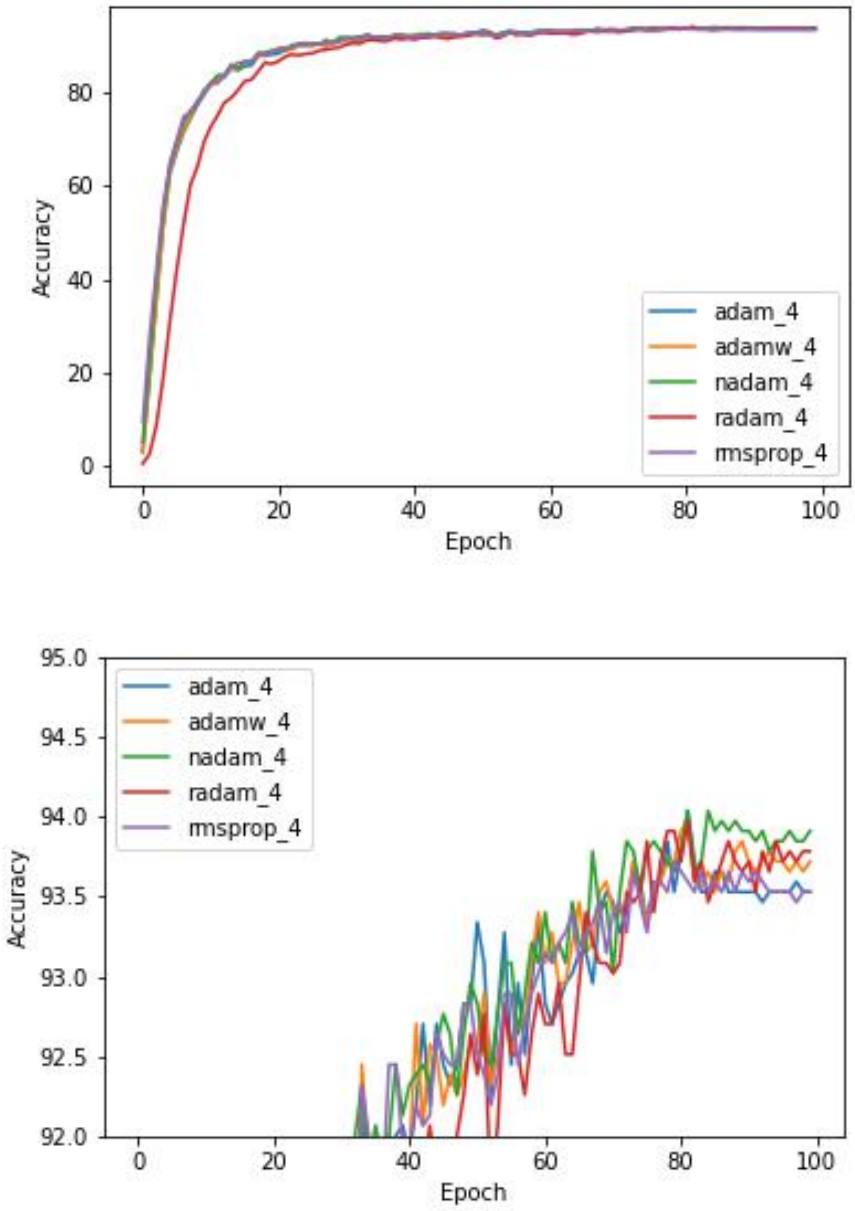


Figura 6.21: Top 5 Otimizadores para Swin Transformer

Após se repetir todo o processo de otimização de hiperparâmetros e pelas mesmas razões da escolha dos hiperparâmetros do ConvNext, foi escolhido para o Swin Transformer o NAdam como algoritmo de otimização e a ordem de grandeza de -4 para o *learning rate*.

Capítulo 7

Resultados e Comparação dos Modelos

Com os hiperparâmetros escolhidos para ambos os modelos, foi então realizado um treino de 500 épocas com cada um. Os resultados obtidos são visíveis nas imagens abaixo.

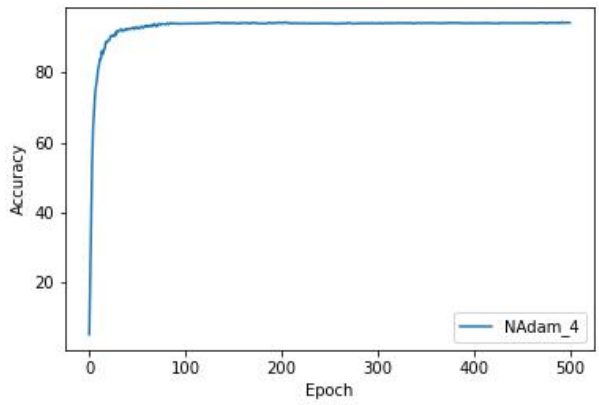


Figura 7.1: Swin Transformer 500 Épocas

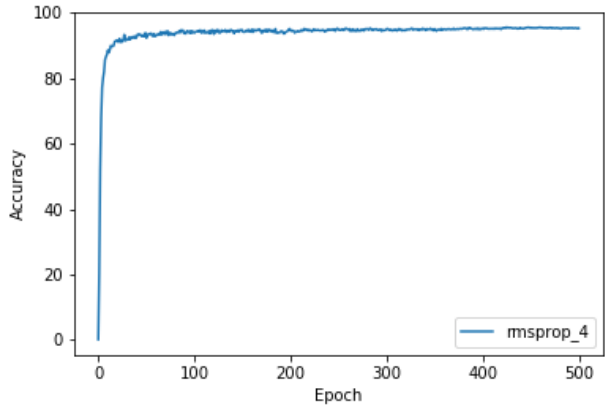
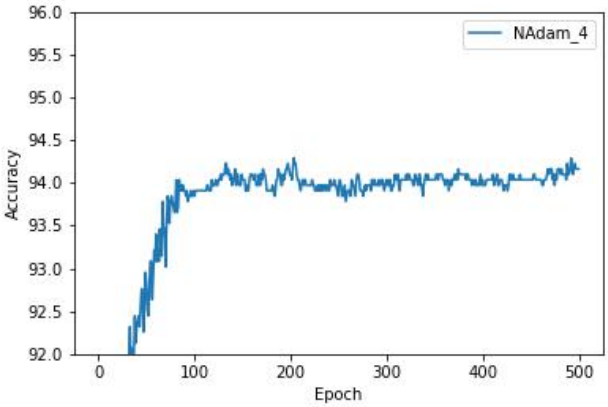


Figura 7.2: ConvNext 500 Épocas



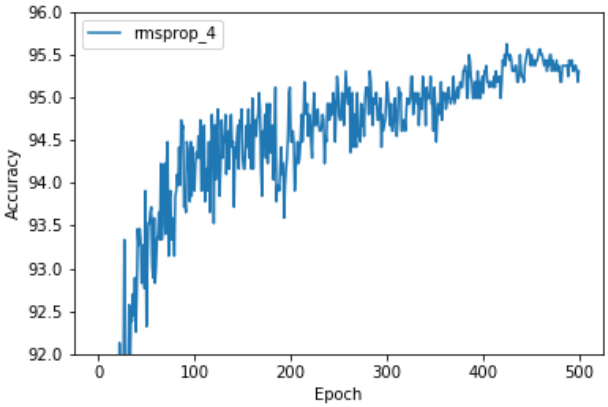


Figura 7.3: Resultados dos Treinos de 500 Épocas

Pela figura 7.3, é percetível a diferença no comportamento da acurácia entre os dois modelos. Enquanto que

no ConvNext a acurácia aparenta ainda possuir margem de crescimento, no Swin Transformer, sensivelmente a partir das 150 épocas, existe uma certa estagnação dos valores.

Após a realização dos treinos longos de 500 épocas, ambos os modelos foram avaliados utilizando o *dataset* de teste. Para o treino e validação da solução foram utilizadas 14567 imagens, ou seja 90% do *dataset* (respetivamente 80% e 10%) e 1618 para o teste que equivale a 10% do *dataset*.

Nas Figuras 7.1 e 7.2, observam-se as matrizes de confusão normalizadas que resultaram da validação do modelo com o *dataset* de teste, que permitem uma leitura mais fácil dos resultados.

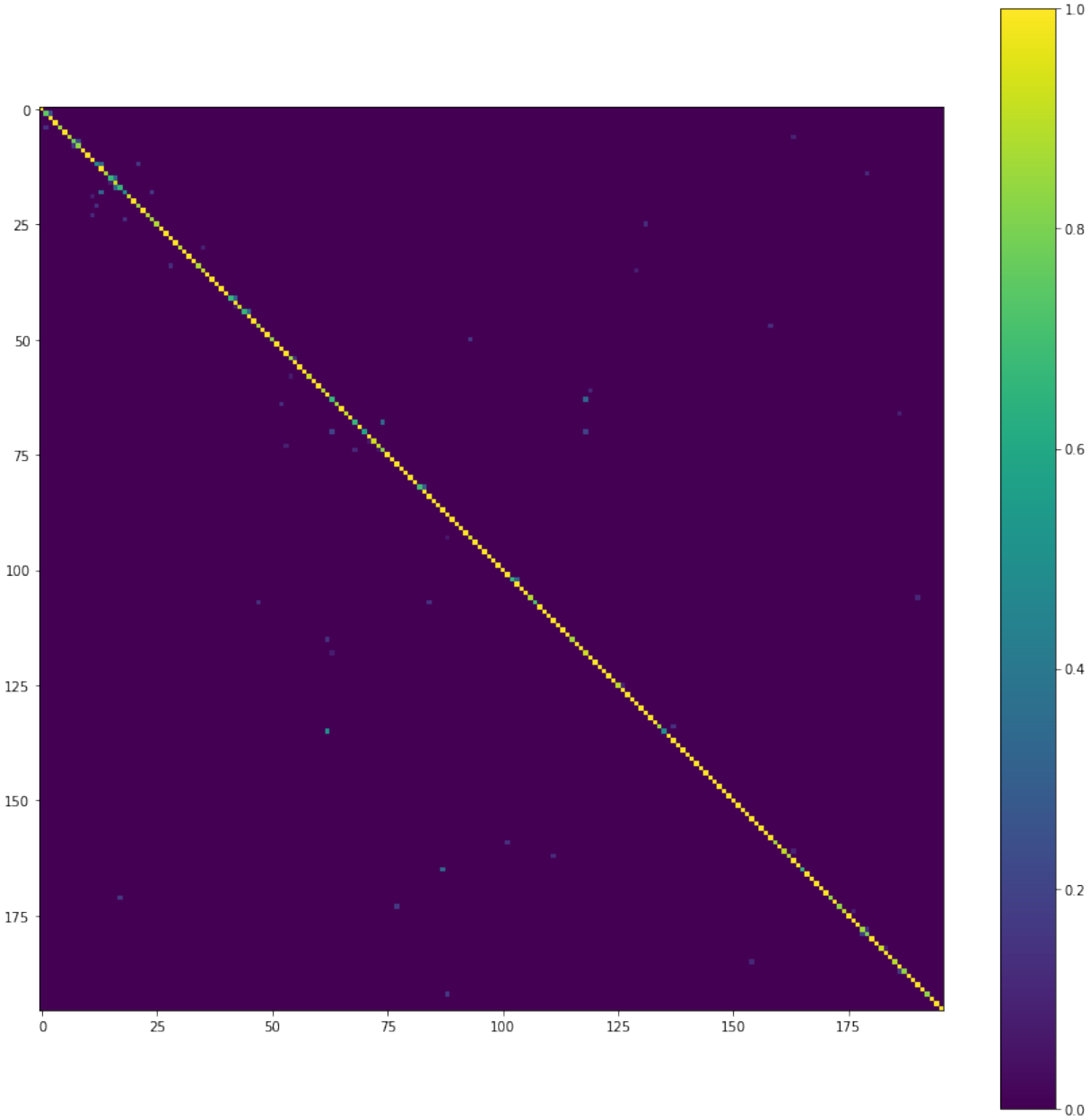


Figura 7.4: Matriz de Confusão normalizada ConvNext

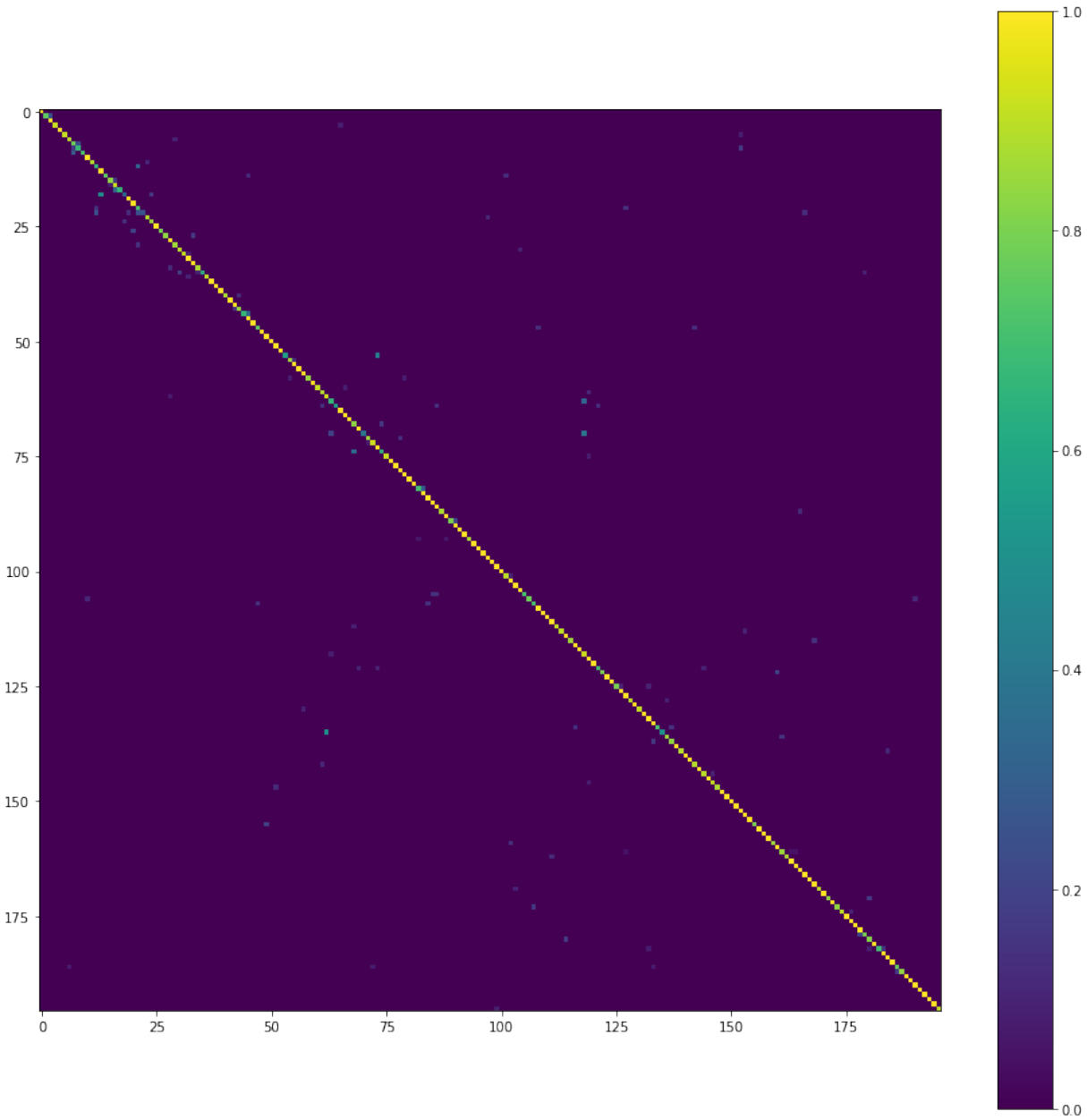


Figura 7.5: Matriz de Confusão normalizada Swin Transformers

Utilizando o ConvNext com o otimizador RMSprop e uma *learning rate* de 10^{-4} ao longo de 500 épocas, obteve-se resultados de 94.53% de acurácia, já no Swin-transformers com o otmizador NAdam e uma *learning rate* de 10^{-4} ao longo de 500 épocas, obteve-se um resultado de 91.429%. Ambos os resultados foram positivos, principalmente o ConvNext cujo resultados se aproximam de valores *state-of-the-art* (recorde atual: 96.41% segundo `https://paperswithcode.com/sota/fine-grained-image-classification-on-stanford`). A diferença de acurácias dos modelos é visível na matriz de confusão, visto que no caso do Swin Transformer há mais pontos fora da diagonal, indicando assim uma menor acurácia. Quanto às diagonais em si, numa análise visual apenas, não aparentam ter diferenças significativas.

Para ilustrar melhor esses erros foram representados graficamente algumas imagens de veículos bem e mal classificados nos dois modelos.

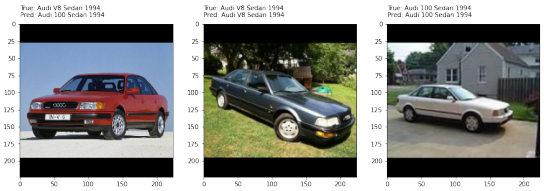


Figura 7.6: Classificações certas e erradas de veículos da AUDI com o modelo ConvNext

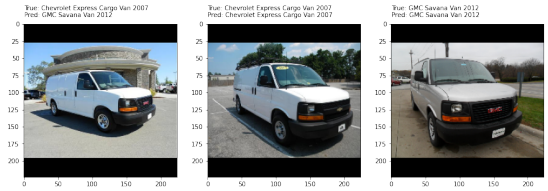


Figura 7.7: Classificações certas e erradas de veículos da Chevrolet e GMC com o modelo ConvNext

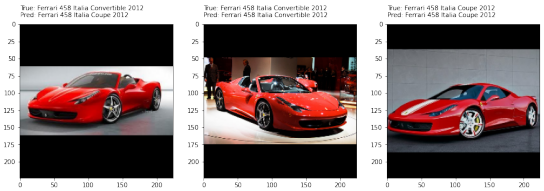


Figura 7.8: Classificações certas e erradas de veículos da Ferrari com o modelo ConvNext

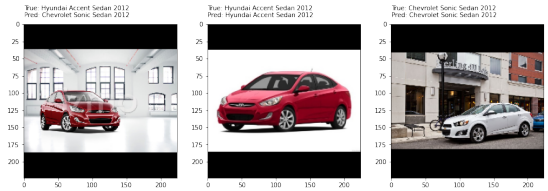


Figura 7.9: Classificações certas e erradas de veículos da Hyundai e Chevrolet com o modelo ConvNext

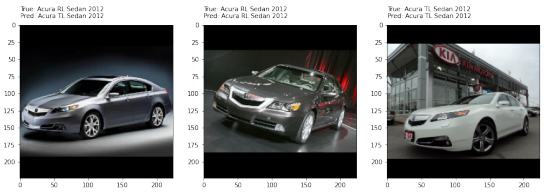


Figura 7.10: Classificações certas e erradas de veículos da Acura com o modelo Swin Transformer

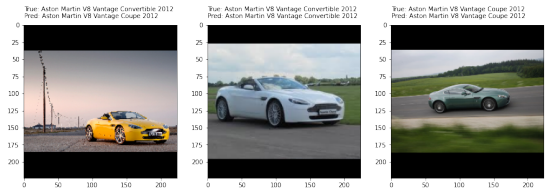


Figura 7.11: Classificações certas e erradas de veículos da Aston Martin com o modelo Swin Transformer

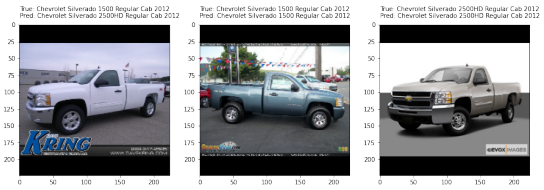


Figura 7.12: Classificações certas e erradas de veículos da Chevrolet com o modelo Swin Transformer

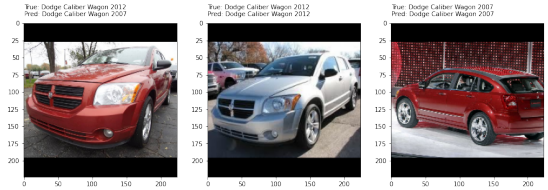


Figura 7.13: Classificações certas e erradas de veículos da Dodge com o modelo Swin Transformer

Observando as Figuras 7.3 até a 7.10, verifica-se que mesmo com o uma classificação errada os modelos mostraram que na maioria das vezes as suas previsões estão muito próximas da realidade, errando apenas o modelo ou ano do carro mas acertando nas marcas dos veículos.

Capítulo 8

Conclusão

Ao longo deste trabalho foram explorados dois algoritmos de visão artificial *state-of-the-art*, o ConvNext e o Swin Transformer. Comparando os resultados obtidos ao longo do projeto, observou-se que o modelo ConvNeXt teve um melhor desempenho na métrica avaliada, acurácia, quer no treino longo (500 épocas) quer no treino mais curto (100 épocas). Estes resultados estão em consonância com o que os autores desta arquitetura escrevem no artigo onde enunciam o ConvNeXt.

Relativamente aos hiperparâmetros, constatou-se que o algoritmo de otimização mais indicado varia consoante o algoritmo de visão artificial utilizado enquanto que, em praticamente todos os testes realizados, o expoente do multiplicador de base 10 da magnitude do *learning rate* que obteve os melhores resultados foi 10^{-4}.

Face aos resultados obtidos ao longo do projeto, foi possível verificar o poder e importância do *Transfer Learning*. Esta técnica que consiste na existência de modelos pré-treinados num dataset permitiu-nos obter performances excelentes que se aproximam de resultados *state-of-the-art* num novo dataset e num problema diferente. Vale a pena salientar que estes resultados foram obtidos com treinos que duraram apenas algumas horas, em *hardware* que apesar de tudo não é especializado.

Ainda assim, como trabalho futuro, apesar de os resultados obtidos estarem perto, seria possível igualar ou até mesmo ultrapassar os valores *state-of-the-art* com a otimização de mais hiperparâmetros e com o estudo e análise dos casos de erro de previsão dos modelos aliados com outras estratégias que não foram exploradas neste projeto como por exemplo, a utilização de *data augmentation*.

Por fim, em relação ao problema da deteção de marca,modelo e ano de veículos, verifica-se que em casos de uso reais, nem sempre existem *datasets* com *labels* pois a quantidade de dados sem *label* é muito superior á quantidade de dados com *label*. Com isto em mente, no futuro seria benéfico explorar quer a utilização de dados sintético gerados por renderização de modelos de carro 3D, quer uma abordagem a este problema com aprendizagem não-supervisionada.

Apêndice A

Tabelas de Resultados

Tabela A.1: Acurácias Máximas ao fim de 500 Épocas

Modelo	Otimizador	*Learning Rate*	Acurácia Máxima(%)
Swin Transformer	NAdam	10^{-4}	91.429
ConvNext	RMSProp	10^{-4}	94.525

Tabela A.2: Swin Transformer: Acurácias Máximas ao fim de 100 Épocas

Otimizador	*Learning Rate*	Acurácia Máxima(%)
SGD	10^{-4}	1.777
SGD	10^{-5}	0.381
RMSProp	10^{-4}	93.718
RMSProp	10^{-5}	8.249
Adam	10^{-4}	94.036
Adam	10^{-5}	8.376
AdamW	10^{-4}	93.972
AdamW	10^{-5}	8.122
NAdam	10^{-4}	94.036
NAdam	10^{-5}	8.756
RAdam	10^{-4}	93.972
RAdam	10^{-5}	9.201

Tabela A.3: ConvNext: Acurácias Máximas ao fim de 100 Épocas

Otimizador	*Learning Rate*	Acurácia Máxima(%)
SGD	10^{-4}	42.069
SGD	10^{-5}	4.759
SGD	10^{-6}	4.187
SGDP	10^{-4}	42.068
SGDP	10^{-5}	4.758
SGDP	10^{-6}	4.187
Adam	10^{-4}	95.114
Adam	10^{-5}	89.784
Adam	10^{-6}	13.769
AdamW	10^{-4}	95.051
AdamW	10^{-5}	89.721
AdamW	10^{-6}	13.769
NAdam	10^{-4}	95.178
NAdam	10^{-5}	89.721
NAdam	10^{-6}	13.579
RAdam	10^{-4}	10.216
RAdam	10^{-5}	2.792
RAdam	10^{-6}	3.173
AdamP	10^{-4}	95.305
AdamP	10^{-5}	89.721
AdamP	10^{-6}	13.769
AdaDelta	10^{-4}	5.467
AdaDelta	10^{-5}	3.490
AdaDelta	10^{-6}	0.571
AdaFactor	10^{-4}	94.987
AdaFactor	10^{-5}	88.515
AdaFactor	10^{-6}	9.645
RMSProp	10^{-4}	95.876
RMSProp	10^{-5}	95.305
RMSProp	10^{-6}	88.959
RMSPropTf	10^{-4}	95.431
RMSPropTf	10^{-5}	95.178
RMSPropTf	10^{-6}	87.563
NovoGrad	10^{-4}	82.741
NovoGrad	10^{-5}	2.157
NovoGrad	10^{-6}	1.142
NvNovoGrad	10^{-4}	82.487
NvNovoGrad	10^{-5}	2.093
NvNovoGrad	10^{-6}	1.269

Apêndice B

Gráficos da Loss ao longo de 100 Épocas

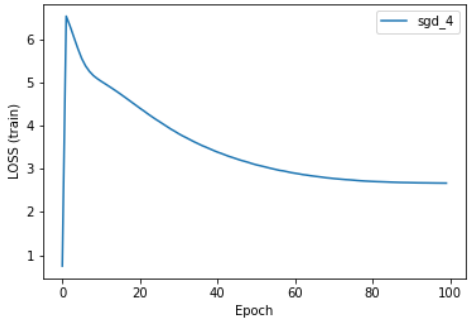


Figura B.1: SGD com lr = 10^{-4}, ConvNext

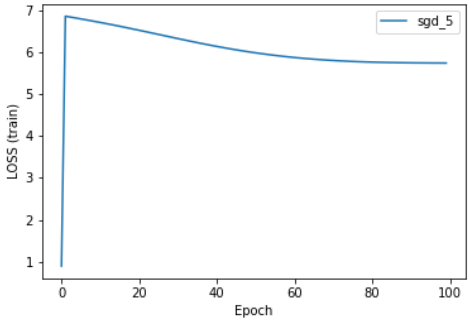


Figura B.2: SGD com lr = 10^{-5}, ConvNext

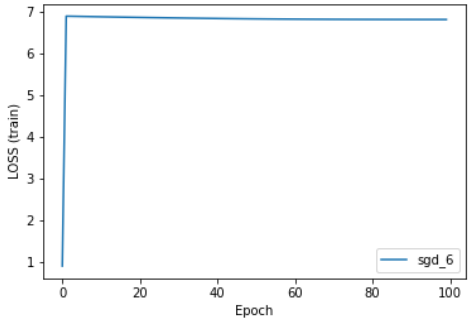


Figura B.3: SGD com lr = 10^{-6}, ConvNext

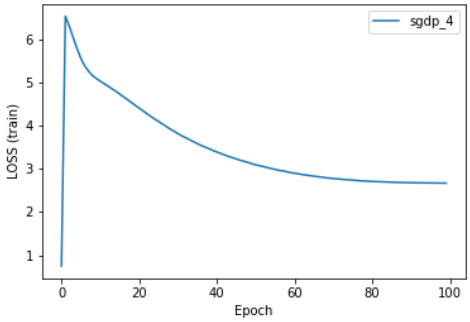


Figura B.4: SGDP com lr = 10^{-4}, ConvNext

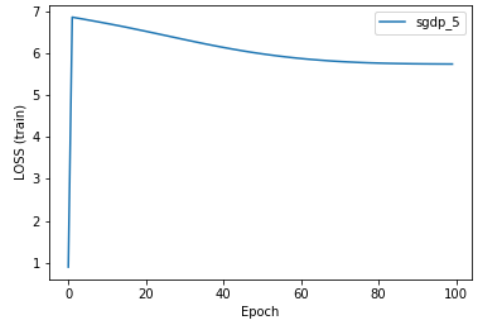


Figura B.5: SGDP com lr = 10^{-5}, ConvNext

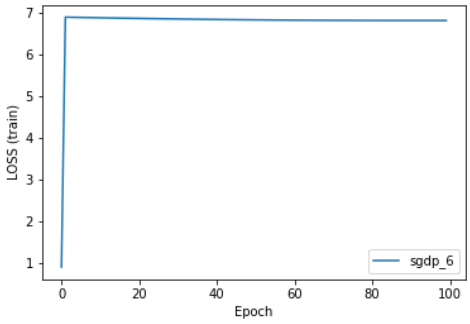


Figura B.6: SGDP com lr = 10^{-6}, ConvNext

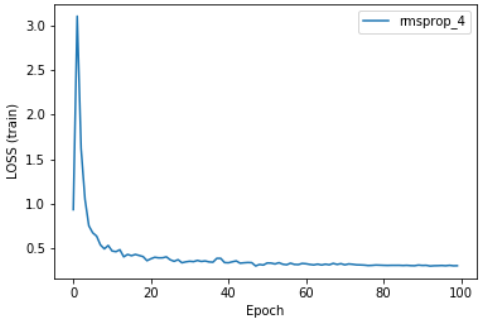


Figura B.7: RMSProp com lr = 10^{-4}, ConvNext

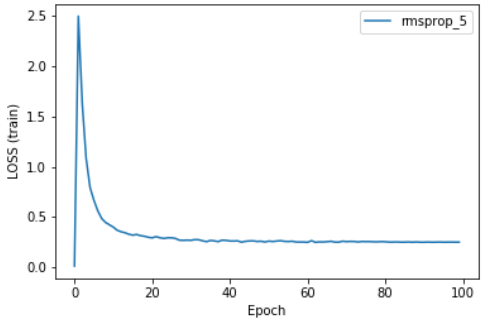


Figura B.8: RMSProp com lr = 10^{-5}, ConvNext

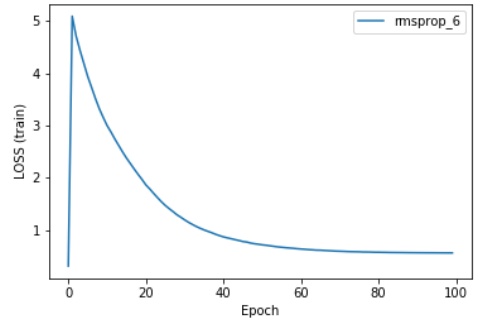


Figura B.9: RMSProp com lr = 10^{-6}, ConvNext

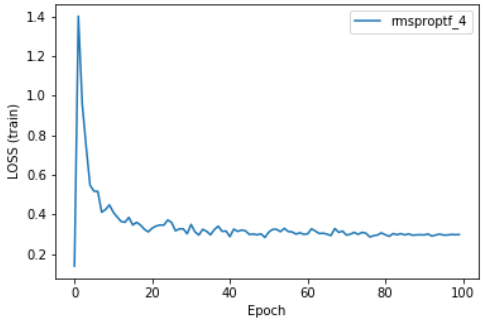


Figura B.10: RMSPropTf com lr = 10^{-4}, ConvNext

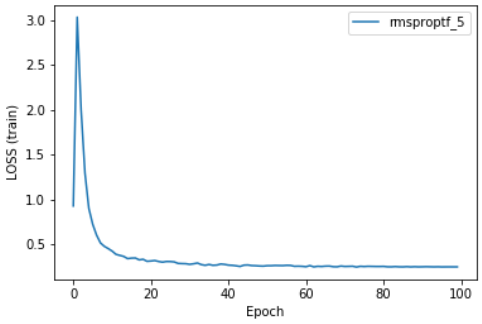


Figura B.11: RMSPropTf com lr = 10^{-5}, ConvNext

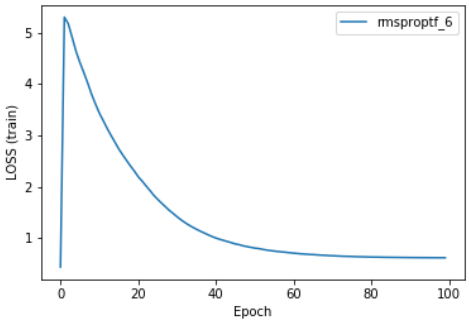


Figura B.12: RMSPropTf com lr = 10^{-6}, ConvNext

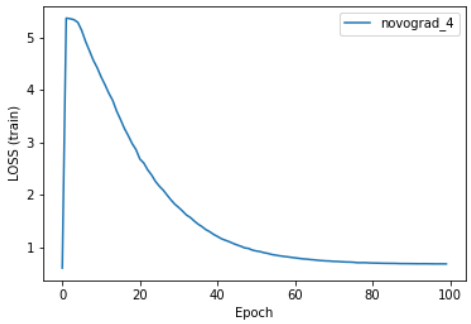


Figura B.13: NovoGrad com lr = 10^{-4}, ConvNext

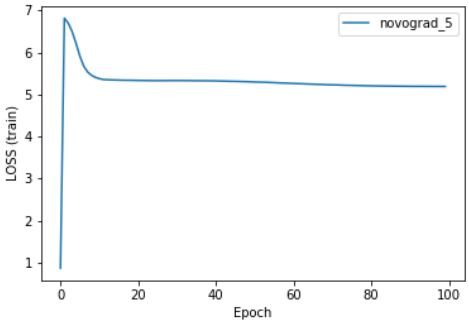


Figura B.14: NovoGrad com lr = 10^{-5}, ConvNext

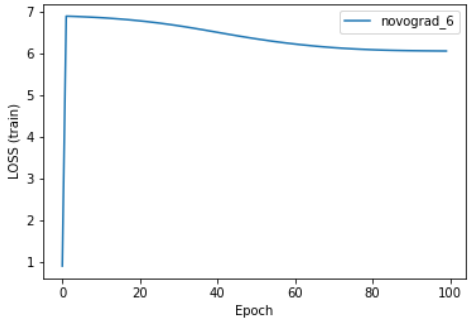


Figura B.15: NovoGrad com lr = 10^{-6}, ConvNext

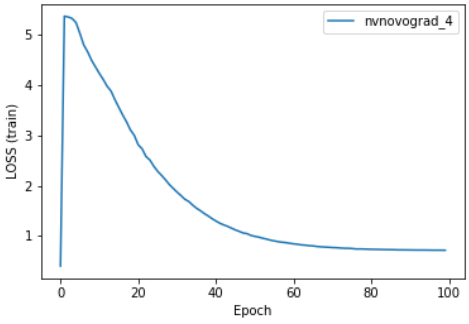


Figura B.16: NvNovoGrad com lr = 10^{-4}, ConvNext

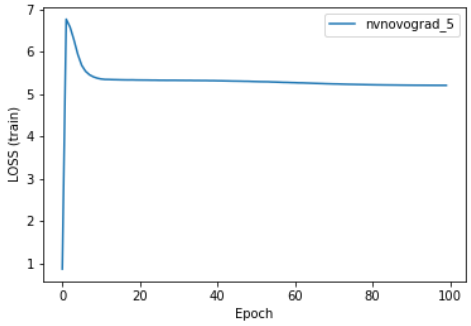


Figura B.17: NvNovoGrad com lr = 10^{-5}, ConvNext

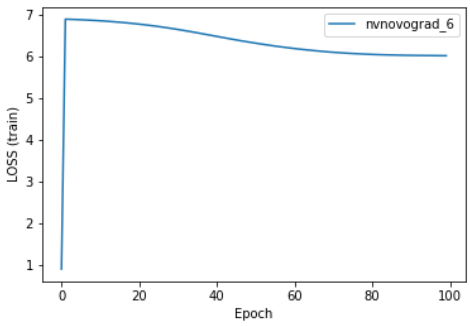


Figura B.18: NvNovoGrad com lr = 10^{-6}, ConvNext

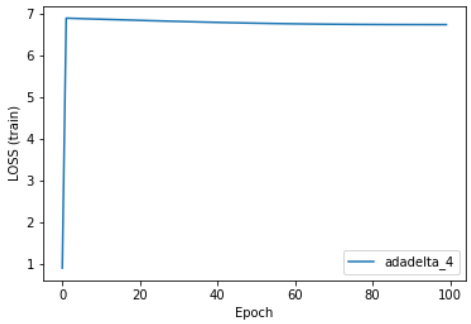


Figura B.19: AdaDelta com lr = 10^{-4}, ConvNext

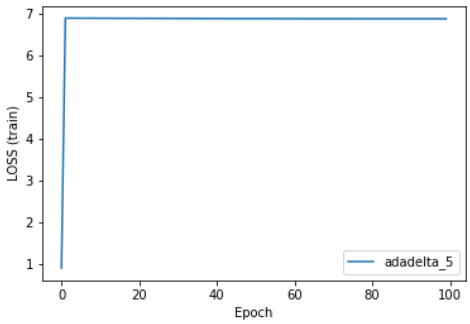


Figura B.20: AdaDelta com lr = 10^{-5}, ConvNext

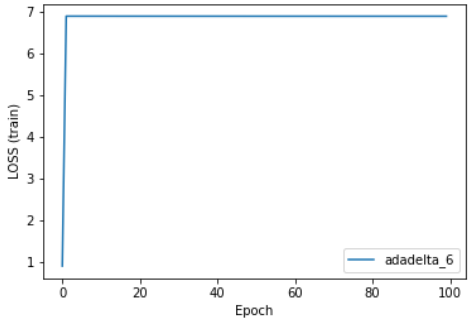


Figura B.21: AdaDelta com lr = 10^{-6}, ConvNext

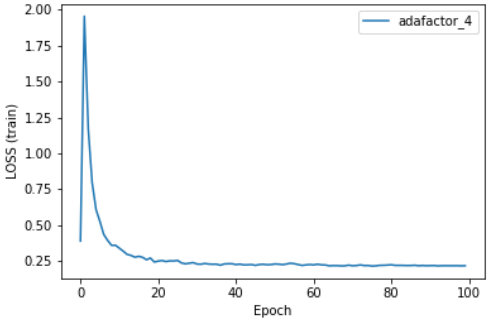


Figura B.22: AdaFactor com lr = 10^{-4}, ConvNext

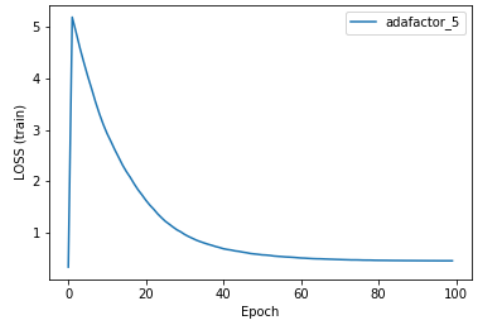


Figura B.23: AdaFactor com lr = 10^{-5}, ConvNext

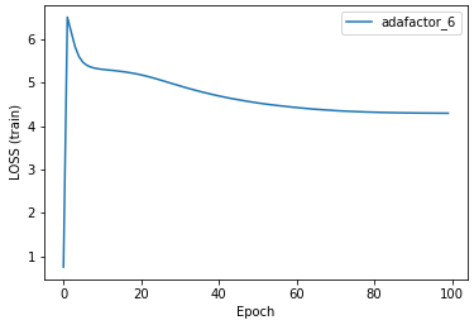


Figura B.24: AdaFactor com lr = 10^{-6}, ConvNext

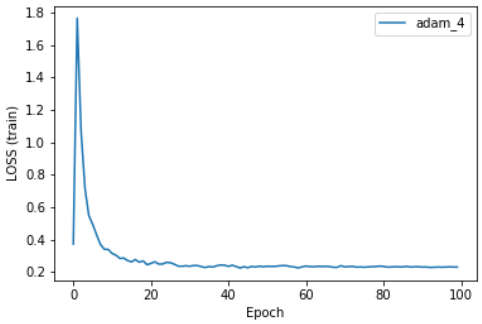


Figura B.25: Adam com lr = 10^{-4}, ConvNext

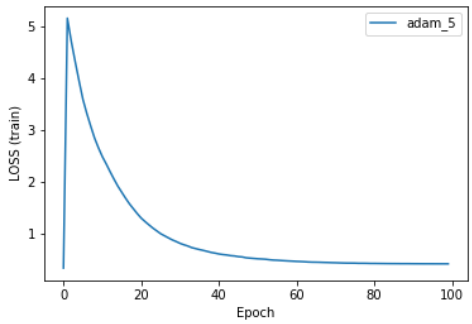


Figura B.26: Adam com lr = 10^{-5}, ConvNext

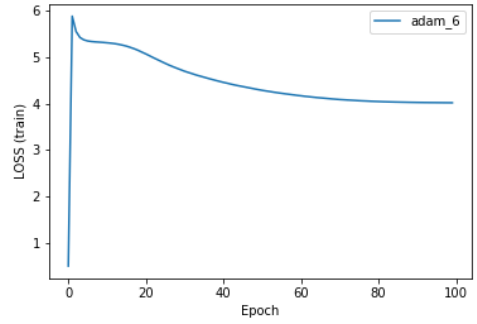


Figura B.27: Adam com lr = 10^{-6}, ConvNext

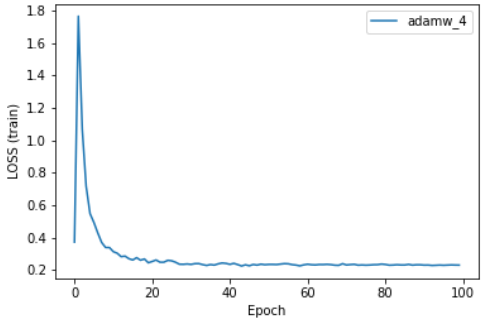


Figura B.28: AdamW com lr = 10^{-4}, ConvNext

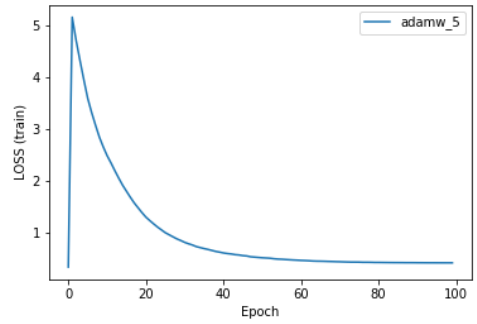


Figura B.29: AdamW com lr = 10^{-5}, ConvNext

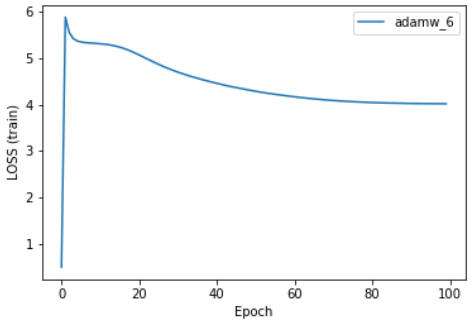


Figura B.30: AdamW com lr = 10^{-6}, ConvNext

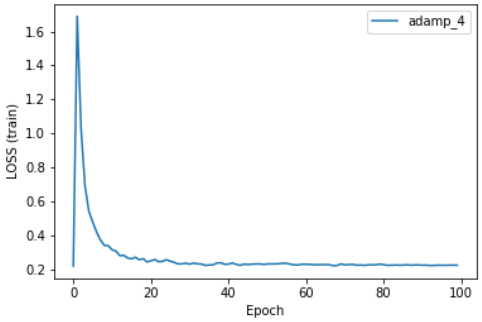


Figura B.31: AdamP com lr = 10^{-4}, ConvNext

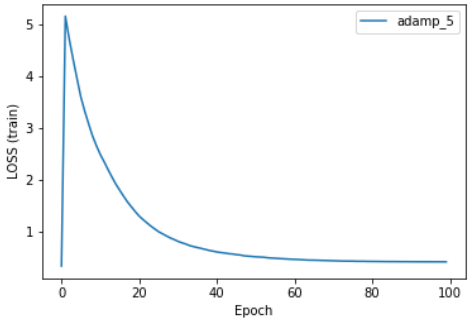


Figura B.32: AdamP com lr = 10^{-5}, ConvNext

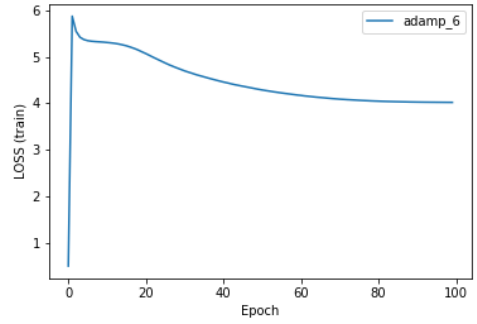


Figura B.33: AdamP com lr = 10^{-6}, ConvNext

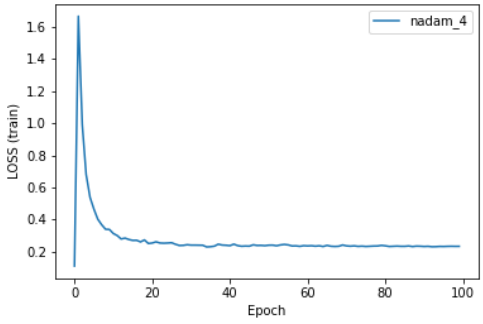


Figura B.34: NAdam com lr = 10^{-4}, ConvNext

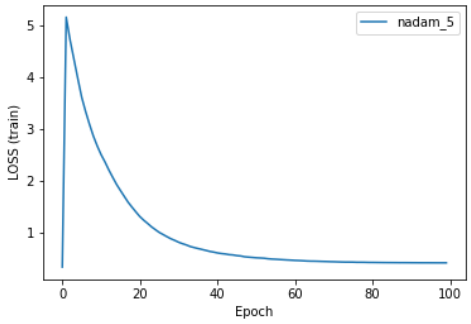


Figura B.35: NAdam com lr = 10^{-5}, ConvNext

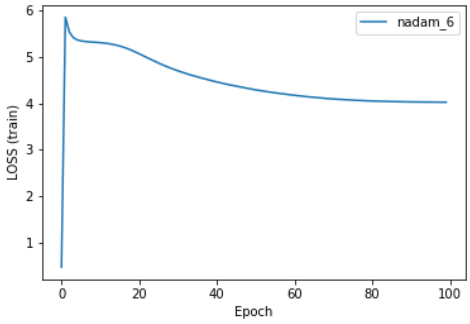


Figura B.36: NAdam com lr = 10^{-6}, ConvNext

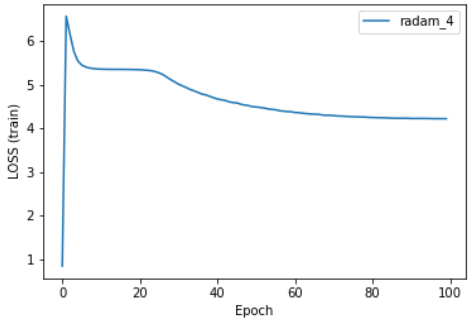


Figura B.37: RAdam com lr = 10^{-4}, ConvNext

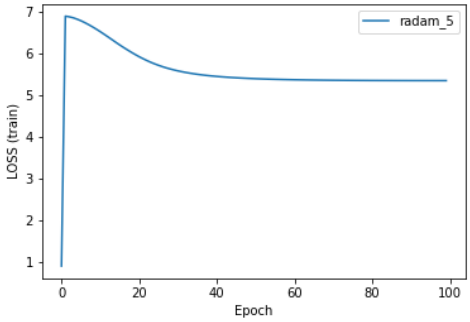


Figura B.38: RAdam com lr = 10^{-5}, ConvNext

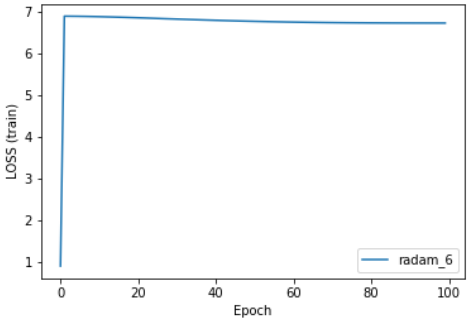


Figura B.39: RAdam com lr = 10^{-6}, ConvNext

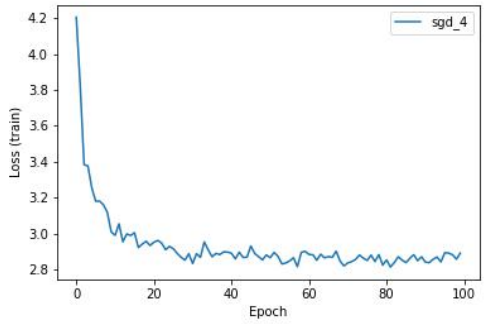


Figura B.40: SGD com lr = 10^{-4}, Swin Transformer

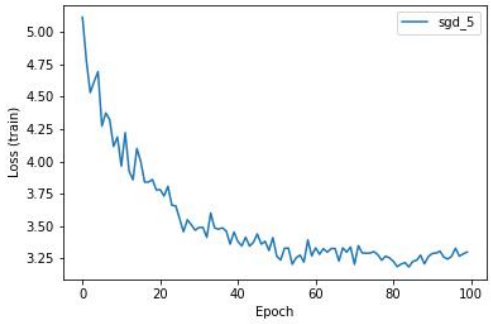


Figura B.41: SGD com lr = 10^{-5}, Swin Transformer

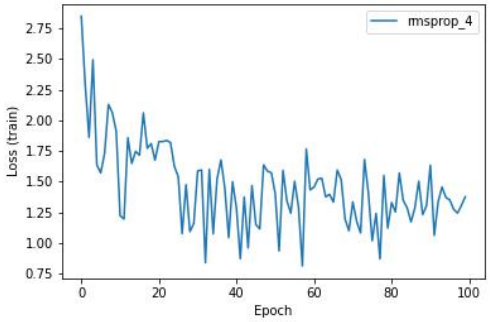


Figura B.42: RMSProp com lr = 10^{-4}, Swin Transformer

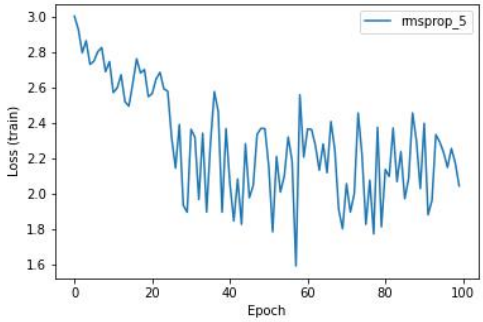


Figura B.43: RMSProp com lr = 10^{-5}, Swin Transformer

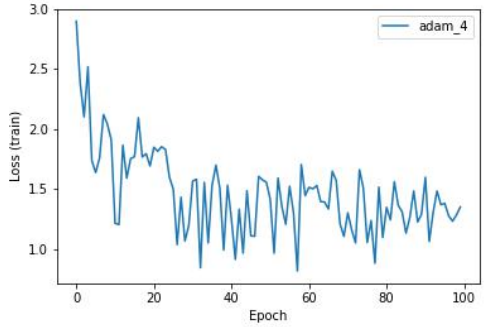


Figura B.44: Adam com lr = 10^{-4}, Swin Transformer

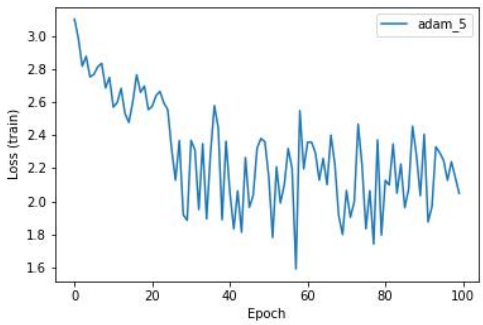


Figura B.45: Adam com lr = 10^{-5}, Swin Transformer

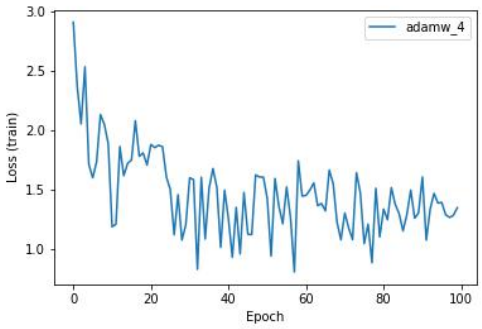


Figura B.46: AdamW com lr = 10^{-4}, Swin Transformer

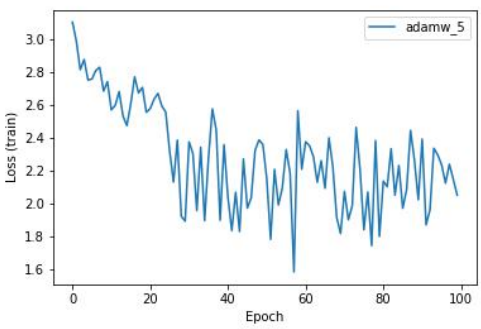


Figura B.47: AdamW com lr = 10^{-5}, Swin Transformer

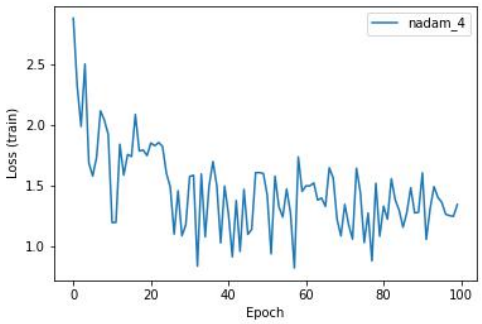


Figura B.48: NAdam com lr = 10^{-4}, Swin Transformer

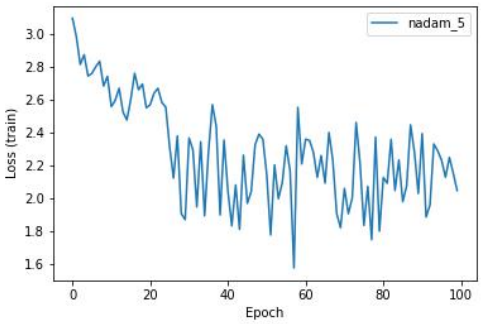


Figura B.49: NAdam com lr = 10^{-5}, Swin Transformer

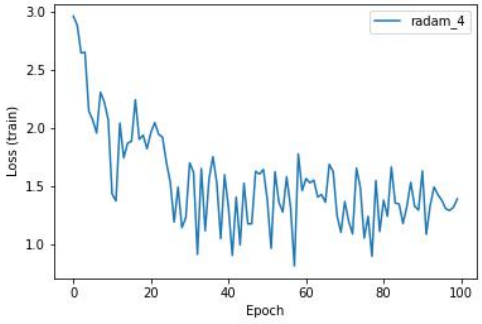


Figura B.50: RAdam com lr = 10^{-4}, Swin Transformer

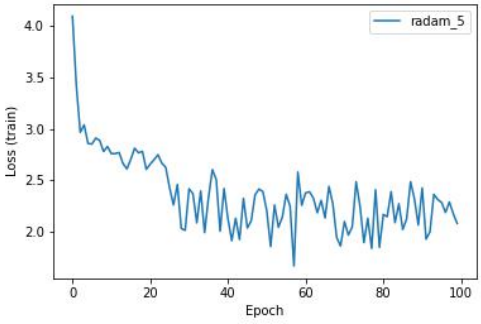


Figura B.51: RAdam com lr = 10^{-5}, Swin Transformer

Apêndice C

Gráficos da Loss ao longo de 500 Épocas

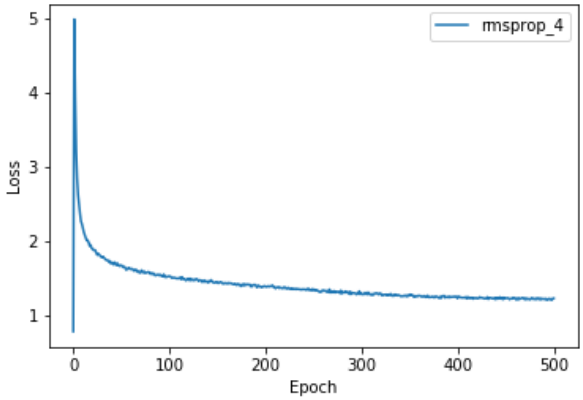


Figura C.1: RMSProp com lr = 10^{-4}, ConvNext

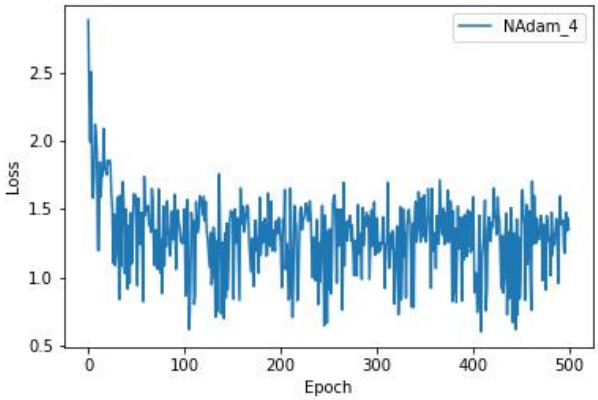


Figura C.2: NAdam com lr = 10^{-4}, Swin Transformer

Apêndice D

Exemplo de configuração para treino com Swin Transformers

```
MODEL:
  TYPE: swin
  NAME: adam_4
  DROP_PATH_RATE: 0.2
  SWIN:
    EMBED_DIM: 128
    DEPTHS: [2, 2, 18, 2]
    NUM_HEADS: [4, 8, 16, 32]
    WINDOW_SIZE: 7
TRAIN:
  EPOCHS: 1
  WARMUP_EPOCHS: 0
  WEIGHT_DECAY: 1e-8
  BASE_LR: 2e-04
  WARMUP_LR: 2e-08
  MIN_LR: 2e-06
  OPTIMIZER:
    NAME: adam
```

Apêndice E

Script *setup* de treino para Swin Transformer

Listing E.1: ␣ deve ser entendido como espaço branco; — deve ser entendido como −−

```
import␣json
import␣os

#Escolha␣de␣Otimizador␣com␣Otimizacao␣da␣Ordem␣de␣Grandeza␣do␣Learning␣Rate

fich␣=␣input('Nome␣Ficheiro␣de␣Output:')

with␣open(fich,"w")␣as␣o:
␣␣␣␣o.write('#!␣/bin/bash␣\n')

␣␣␣␣n␣=␣int(input('N␣␣de␣Epocas:'))
␣␣␣␣otimList␣=␣['sgd',␣'adam',␣'adamw',␣'nadam',␣'radam',␣'adadelta','rmsprop']
␣␣␣␣lrexpList␣=␣[4,5]

␣␣␣␣for␣otim␣in␣otimList:
␣␣␣␣␣␣␣␣for␣lrexp␣in␣lrexpList:
␣␣␣␣␣␣␣␣␣␣␣␣o.write("mkdir␣{}_{}\n".format(otim,lrexp))
␣␣␣␣␣␣␣␣␣␣␣␣comd␣=␣"python␣−m␣torch.distributed.launch␣−−nproc_per_node␣1␣−−
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣master_port␣12345␣main.py␣−−cfg␣configs/{}_{}.yaml␣−−pretrained␣../
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Checkpoints/swin_base_patch4_window7_224_22k.pth␣−−data−path␣../
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Datasets/padded␣−−batch−size␣32␣−−accumulation−steps␣2\n\n".format(
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣otim,lrexp)
␣␣␣␣␣␣␣␣␣␣␣␣o.write(comd)
```

Apêndice F

Script de *padding*

```
import torch
from PIL import Image
import torchvision.transforms as T
import json
import os

n = 14567  #numero de imagens

for x in range(n):
    #paths de origem e destino das img
    orig = '/home/br99/Desktop/imgs/orig/{:06}.jpg'.format(x)
    dest = '/home/br99/Desktop/imgs/pad/{:06}.jpg'.format(x)

    #imagem a transformar
    orig_img = Image.open(orig)
    w, h = orig_img.size
    dif = 0

    if(w>h):
        dif = w-h
        if (dif%2==0):
            wh = [0,dif//2]
        else:
            wh = [0,(dif//2)+1,0,dif//2]
        new_img = T.Pad(wh)(orig_img)

    if(h>w):
        dif = h-w
        if (dif%2==0):
            wh = [dif//2,0]
        else:
            wh = [(dif//2)+1,0,(dif//2),0]
        new_img = T.Pad(wh)(orig_img)

    if(h==w): new_img = orig_img

    #guardar imagem padded
    new_img.save(dest)
```

Bibliografia

3D Object Representations for Fine-Grained Categorization Krause Jonathan et al. *4th IEEE Workshop on 3D Representation and Recognition*, ICCV 2013, 2013.

Vaswani et al. *Attention is all you need*, 2017.

Izmailov, P. et al. *Averaging weights leads to wider optima and better generalization*, 2018.

Jiayuan Gu et al. *Learning Region Features for Object Detection*, ECCV 2018, 2018.

Shazeer Noam, Stern Mitchell. *Adafactor: Adaptive Learning Rates with Sublinear Memory Cost*, 2018.

M. F. Kunduracı, H. Kahramanli Örnek. *Vehicle Brand Detection Using Deep Learning Algorithms*, International Journal of Applied Mathematics Electronics and Computers, vol. 7, no. 3, pp. 70-74, 2019.

Heo Byeongho et al. *AdamP: Slowing Down the Slowdown for Momentum Optimizers on Scale-invariant Weights*, 2020.

Liu Zhuang et al. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*, 2021.

Kamiri, Jackson Mariga, Geoffrey. *Research Methods in Machine Learning: A Content Analysis*. International Journal of Computer and Information Technology, 2021

R. Llugsi, S. E. Yacoubi, A. Fontaine e P. Lupera. *Comparison between Adam, AdaMax and Adam W optimizers to implement a Weather Forecast based on Neural Networks for the Andean city of Quito* 2021 IEEE Fifth Ecuador Technical Chapters Meeting (ETCM), pp. 1-6, 2021.

Liu Zhuang et al. *A ConvNet for the 2020s*, 2022.

`https://prog.world/swin-transformer-architecture-overview/`

`https://remykarem.github.io/blog/gradient-descent-optimisers.html`

Ward, Rachel, Xiaoxia Wu, e Leon Bottou. *Adagrad stepsizes: Sharp convergence over nonconvex landscapes*"International Conference on Machine Learning, PMLR, 2019.

Suvrit Sra, Sebastian Nowozin Stephen J. Wright. *Optimization for Machine Learning*, 2012.

Zhang Aston, Lipton Zachary, Li Mu Smola Alexander. *Dive into Deep Learning*, 2022.