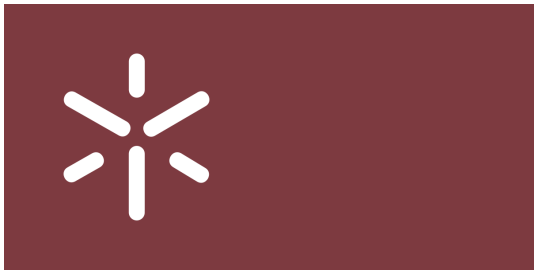

Universidade do Minho

Grupo 1
José Malheiro ($PG50509$), **David Duarte** ($PG50315$)

Mestrado em Engenharia Informática

Trabalho Prático 1 - Dataset GTSRB

Junho 2023

Visão por Computador e Processamento de Imagem

Conteúdo

Capítulo 1

Introdução

No âmbito da Unidade Curricular (UC) de Visão por Computador e Processamento de Imagem (VCPI), foi desenvolvido, num primeiro trabalho de grupo, o treino de modelos aplicando *data augmentation*, tanto em pré-processamento como dinâmico, bem como um estudo do potencial de utilizar *ensembles* de redes. Deste modo, numa primeira fase, vão ser explorados os filtros e métodos de processamento de imagem , no âmbito de avaliar o impacto dos mesmos no desempenho final da rede. Deste modo, serão treinados vários modelos, com *data augmentation* e, no seguimento de tal, serão realizadas as respetivas análises dos resultados obtidos.

Por fim, numa segunda parte, as redes treinadas anteriormente, farão parte de uma rede ensemble, de modo a explorar o uso de várias redes para tentar obter melhores resultados.

O trabalho, como especificado no enunciado, será realizado em *notebooks*, do *Python*, sendo utilizada a biblioteca *Tensorflow* para as operações de treino, teste dos datasets, entre outras.

Num prefácio adicional, decidiu-se tratar imagens num formato *.PNG*, pelo que teve-se, num pré-processamento, as converter do formato *.PPM* inicial. Convencionou-se que as imagens terão um tamanho (32 x 32) e que as *batches* de imagens serão de tamanho **64**.

Por fim, independentemente da *Data augmentation* que será aplicada no final, todos os *datasets*, de treino e teste, serão passados por um processo de **normalização**.

Capítulo 2

Modelos criados

Começamos por definir um primeiro modelo com apenas *Conv Layers*, que foi adaptado de um dos exemplos vistos em aula. É um modelo simples com o intuito de entender o estado do treino/validação do modelo num contexto básico. Inicialmente foi criado um modelo **sequencial**, que permite construir uma *stack* linear de camadas, de seguida é adicionada uma camada de *input* ao modelo, representando o tipo de dados que irá receber (**largura, altura** e **número de canais**). Foi também adicionada uma camada de **convolução 2D** ao modelo com 64 filtros e uma janela de convolução de tamanho *(5,5)*. Esta camada extrai recursos das imagens através da convolução. São criadas duas destas *hidden layers*, sendo que para cada uma delas é adicionada uma função de ativação *LeakyReLU()* (variação da função *ReLU()* que permite valores negativos. A camada *Flatten()* converte o *output* das camadas convolucionais num vetor unidimensional, preparando os dados para serem alimentados nas camadas densamente conectadas. É Adicionada uma camada densamente conectada com 128 unidades (*Dense()*); A própria rede densamente conectada receberá uma função de ativação (*LeakyReLU()*). Adicionamos a camada de saída ao modelo com *classCount* unidades, correspondendo ao número de classes do problema. A ativação **'softmax'** é usada para obter probabilidades de cada classe. Criamos um otimizador *Adam* com uma taxa de aprendizagem de 0.001 (número testado arbitrariamente). Por fim, compilamos o modelo, definindo o otimizador, a função de perda (*categorical_crossentropy* para classificação multiclasse) e as métricas (neste caso, apenas a *accuracy*).

Este modelo é uma **CNN** composta por várias camadas convolucionais seguidas por camadas densamente conectadas. É usada a função de ativação *LeakyReLU* para introduzir não-linearidade e uma camada de saída com ativação softmax para classificação multiclasse. A

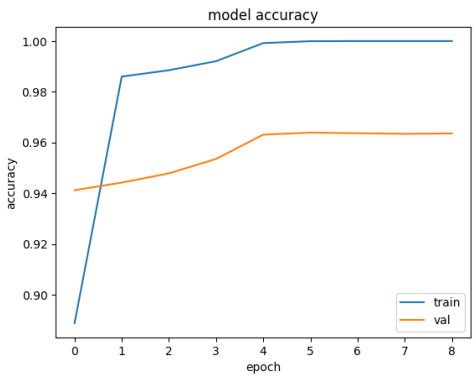

Figura 2.1: Modelo 1 - *Accuracy*

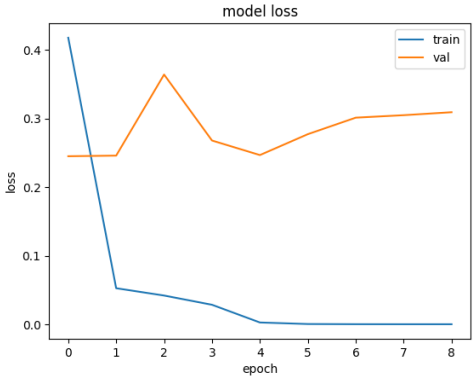

Figura 2.2: Modelo 1 - *Loss*

função de perda é a *categorical_crossentropy* e a métrica de avaliação é a accuracy.

Em relação à preparação e treino deste modelo, considerou-se 20 *epochs* um número bom para o modelo, considerando os *early stoppings* obtidos em *fits* sucessivos.

A partir da quarta *epoch*, tanto a *loss* quanto a *accuracy* de validação não melhoraram significativamente o que nos indica que o modelo pode ter atingido um limite no seu desempenho e não está a melhorar mais com o treino adicional. A partir da quarta *epoch*, também podemos observar a redução da taxa de aprendizagem (*learning rate*) devido ao uso do *callback ReduceLROnPlateau*. Com isto, sabemos que o modelo se está a ajustar gradualmente à taxa de aprendizagem em busca de uma convergência mais precisa. O treino foi interrompido na nona *epoch* devido ao uso do *callback EarlyStopping*. Tal ocorreu porque a perda de validação não melhorou por um período de 8 *epochs* consecutivas. Com base nessas observações, podemos concluir que o modelo obteve um desempenho geral muito bom, alcançando alta *accuracy* tanto nos dados de treino quanto nos dados de validação.

Definição do 2º Modelo Partindo do 1º modelo criado, verificamos que estava a acontecer um caso de *overfitting*, pelo que seria o nosso trabalho melhorar a componenente de generalização do modelo. A partir disto, adicionamos três novas componentes:

BatchNormalization(): Adiciona-se camadas de normalização por *Batch* após cada camada convolucional. A *Batch normalization* ajuda a acelerar o treino, tornando os gradientes mais estáveis durante o processo de otimização; levando a uma convergência mais rápida e a um melhor desempenho do modelo. Ao normalizar as ativações, a *Batch Normalization* ajuda a estabilizar o treino da rede neuronal; reduz-se a dependência das atualizações dos pesos em relação às alterações nas ativações anteriores. Deste modo, o treino mais robusto e menos sensível a

hiperparâmetros de inicialização. Para além disso, atua como uma forma de regularização, ao introduzir um pequeno ruído durante o treino (reduzir o *overfitting*, especialmente em conjuntos de dados menores). Reduz-se a sensibilidade da rede em relação à inicialização dos pesos (têm menos impacto sobre o desempenho da rede, o que facilita a escolha de valores iniciais dos pesos.) Por fim, permite que a rede neuronal seja treinada mais rapidamente; uma vez que a normalização ajuda a evitar a saturação das funções de ativação e permite uma propagação mais suave dos gradientes durante o processo de retropropagação (por este motivo tem uma camada de normalização após cada camada de convolução). *MaxPooling2D()*: São utilizadas camadas de *pooling* máximo após as camadas convolucionais. Estas ajudam a reduzir a dimensionalidade espacial do output das camadas convolucionais, preservando as características mais importantes. Isso pode melhorar a capacidade de generalização do modelo e reduzir o *overfitting*, ao fornecer uma forma de regularização e prevenção do aumento excessivo de parâmetros na rede neural. (O tamanho da janela foi escolhido de modo a ser compatível com as dimensões das imagens em questão.) *Dropout()*: Inclui-se, finalmente, uma camada de *Dropout* com uma taxa de 0.2 após a camada densa. O *Dropout* é uma técnica de regularização que desativa aleatoriamente uma fração dos neurônios durante o treino; ajudando a prevenir o *overfitting*, forçando o modelo a aprender representações mais robustas e independentes. O nosso *model_II* adiciona *batch normalization*, camadas de *Max pooling 2D* e *Dropout*, que são técnicas comumente usadas para melhorar o desempenho e a capacidade de generalização de modelos de redes nueronais convolucionais (e prevenir o *overfitting*). Essas melhorias podem levar a uma maior precisão e estabilidade durante o treino e teste do modelo.

Fit do modelo atualizado Consegue-se visualizar que o modelo, neste caso, passa por todas as 20 *epochs*, o que indica que existe uma probabilidade de precisar de mais para completamente aprender (com correção).

Contudo, o *fit* atual, com 20 *epochs* já demora bastante tempo, com uma média de 31 segundos por *epoch* (considerando que estamos a correr o modelo sob a GPU).

Avaliação gráfica dos resultados Através da observação dos gráficos é possível reparar que existe uma clara melhoria no aspeto das curvas para *loss* e *accuracy*, no caso do **modelo II**. Contudo, verificam-se vários picos, em ambos os gráficos, o que comprova o que anteriormente tinha sido dito, ou seja, que o programa necessitava de mais *epochs* para treinar mais; pelo que o atual não tem dados representativos do *dataset* de teste que recebeu. Neste caso, procurar-se-á

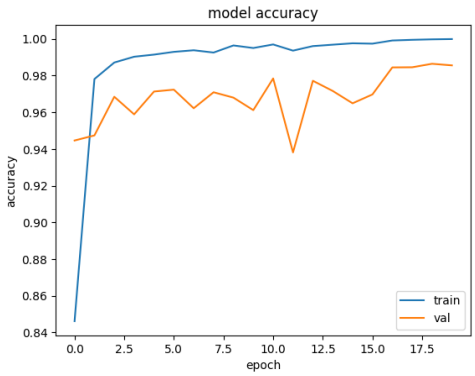

Figura 2.3: Modelo 2 - *Accuracy*

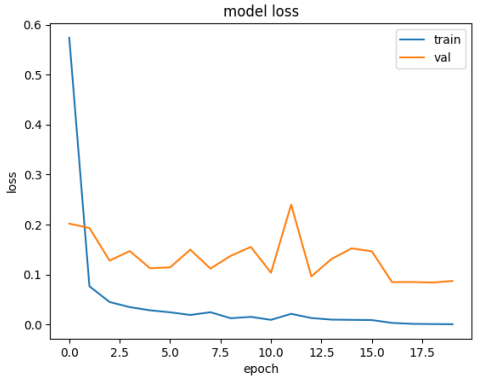

Figura 2.4: Modelo 2 - *Loss*

melhorar este aspeto, tendo em consideração a performance do modelo também.

No entanto, não obstante este aspetos, o modelo continua a ter bons valores finais para todas métricas; ainda melhor que o primeiro (sendo este um ponto positivo de que encontramos-nos num bom caminho). A *val_accuracy* acima de tudo, de 98.56% encontra-se num valor bastante alto e a *accuracy* encontra-se bastante próxima de 1, sendo isto muito bom também.

Definição do 3º Modelo Ao aumentar o número de filtros em cada camada convolucional, a rede é capaz de aprender e extrair características mais complexas e abstratas das imagens de *input*. Cada filtro uma camada convolucional é responsável por detetar um conjunto específico de características de uma imagem, como bordas, texturas, padrões, entre outros. Quanto mais filtros uma camada tiver, mais características diferentes ela será capaz de capturar.

Na prática, uma abordagem comum é aumentar gradualmente o número de filtros à medida que a rede se aprofunda. Isso permite que as camadas iniciais aprendam características mais simples e gerais, enquanto as camadas mais profundas aprendem características mais específicas e complexas. Essa progressão gradual de filtros pode ajudar a melhorar a capacidade da rede de representar e distinguir características detalhadas nas imagens.

No nosso **model_III**, onde a última camada de convolução possui 256 filtros e a anterior possui 128 filtros, espera-se que a camada com mais filtros seja capaz de aprender características mais complexas e detalhadas do que a camada anterior; tal pode ser benéfico para a tarefa específica em questão, pois permite que a rede capture mais informações discriminativas das imagens durante o processo de aprendizagem.

Sendo assim, para o terceiro e último modelo tem-se:

Aumento do número de filtros convolucionais: O modelo III possui um número maior de filtros convolucionais em comparação ao modelo II, no âmbito de ajudar o modelo a capturar mais características relevantes nas imagens e extrair representações mais ricas dos dados.

Diminuir o *kernel* das camadas convolucionais: Diminuiu-se para ser (3,3), o que implica uma redução da área local da imagem que é levada em consideração durante a convolução.

Menor dimensionalidade: Ao usar um *kernel* menor, a dimensão da saída da camada convolucional também será reduzida; útil quando se deseja reduzir a dimensionalidade espacial da imagem, como em camadas de *pooling*. Maior foco local: Com um *kernel* menor, a camada convolucional se concentra numa área menor da imagem de *input* o que pode permitir que a rede neuronal aprenda características mais localizadas e específicas, capturando detalhes finos ou padrões mais subtis. Menor quantidade de parâmetros: Reduzir o tamanho do *kernel* numa camada convolucional também resulta numa redução no número de parâmetros treináveis na rede neural, o que vai ser benéfico em termos de eficiência computacional e capacidade de generalização, especialmente em conjuntos de dados menores. De resto, consegue-se ver que o modelo assemelha-se nos restantes aspetos ao modelo II.

Resultados do fitting do modelo Em termos das métricas a serem avaliadas, estas são tão boas ou melhores ao do modelo II, se considerarmos os resultados finais, contudo, se analisarmos no todo, temos que os valores são muito mais favoráveis e representam um modelo que está claramente a aprender; e que demonstra varições saudáveis para o *loss* e a *accuracy*.

Não obstante isso, o modelo também tem uma performance semelhante ao modelo II, se avaliarmos o tempo para cada *epoch*, apesar de ser uma rede mais complexa; para além disso, obteve-se um *early stopping* na *epoch* 18, o que significa que 20 *epochs* parece ser satisfatório para o modelo aprender corretamente (segundo o que foi definido na função de *callbacks*).

Curvas de loss e accuracy Aqui repara-se as maiores diferenças com os modelos anteriores, uma vez que as variações das curvas no modelo III representa algo próximo de um bom *fit*.

Continua a não ser perfeito, uma vez que as linhas tem uma clara distância entre elas mas, não obstante, não existem picos a significar *overfitting*/*underfitting*, ou que o modelo não teve dados representativos.

É claro que, com *data augmentation*, se pretende melhor os resultados finais.

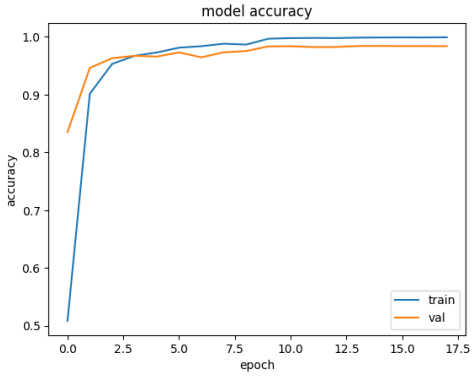

Figura 2.5: Modelo 3 - *Accuracy*

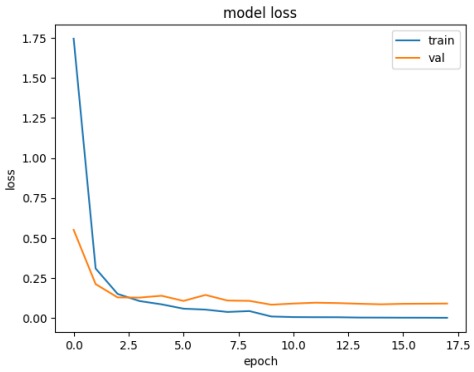

Figura 2.6: Modelo 3 - *Loss*

Comparação entre o modelo I e o modelo II Por razões demonstrativas, apresenta-se um *side by side* dos dois primeiros modelos (I e II). Pelo que se repara na melhoria das métricas entre os modelos.

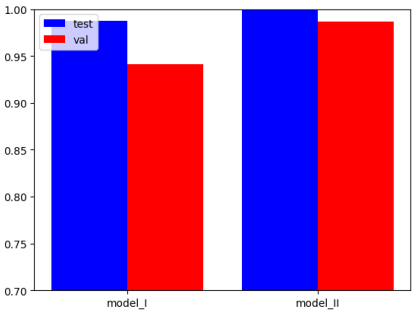

Figura 2.7: Modelo I vs Modelo II - *Loss*

Comparação entre o modelo II e o modelo III Testando o II modelo com o III atinge-se uma incógnita - ambos tem valores equitativos, neste caso, o II até possuí melhores resultados que o III.

Contudo, este gráfico apenas simboliza os valores de accuracy e loss finais e não são simbólicos da correção dos modelos em sí; nem representam uma boa métrica de comparação.

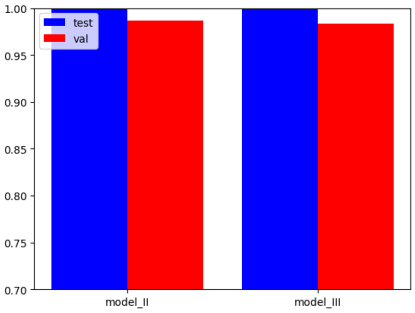

Figura 2.8: Modelo II vs Modelo III - *Loss*

Capítulo 3

Callbacks

Foram preparados quatro callbacks, sendo esses *ModelCheckPoint*, *EarlyStopping*, *TensorBoard* e *ReduceLROnPlateau*. Em relação ao *ModelCheckPoint*, este *callback* é responsável por guardar os pesos do modelo num arquivo de *checkpoint*. Este recebe o caminho onde o arquivo será guardada a métrica a ser monitorizada para determinar se o modelo melhorou na aprendizagem (*monitor*), o nível de detalhe das mensagens (*verbose*); Os dois finais indicam que serão guardados apenas os pesos do modelo, bem cmom os que tiveram melhores valores.

O *callback EarlyStopping* é utilizado para interromper o treino antecipadamente caso não exista uma melhoria na métrica a ser monitorizada, durante um determinado número de *epochs* consecutivas.

Por sua vez o *Tensorboard* utiliza-se para gerar registos que serão utilizados para visualização no *TensorBoard*, que é uma ferramenta de visualização fornecida pelo *TensorFlow*.

Por fim, o *callback ReduceLROnPlateau* é utilizado para reduzir a taxa de aprendizagem (*learning rate*) do *optimizer* quando a melhoria na métrica a ser monitorizada diminui. Para este caso, temos o fator de redução do *learning rate* (new_learning_rate = prev_learning_rate * factor)m o número de *epochs* consecutivas sem melhorias que levará à redução do *learning rate* (*patience*), e finalmente o limite inferior para o *learning rate* (*min_lr*)

Capítulo 4

Data Augmentation

Com os três modelos criados (I, II e III), seguiu-se para o ponto fulcral da primeira parte do trabalho, a decisão sobre as transformações a aplicar sobre os modelos, de modo a exaltar os resultados previamente óbtidos; a *data augmentation*.

Para esta parte do trabalho, a estratégia partiu no teste de várias transformações diferentes com o melhor modelo (III), fornecidas pelo *Keras*, na forma da classe *ImageDataGenerator*, para descobrir quais são as mais satisfatórias; e as que melhoram o nosso modelo.

Deste modo, realiza-se o processo sobre o modelo III primeiro, dado a ser o que possui melhores resultados, contudo, iremos, também, aplicar nos restantes dois modelos o mesmo processo de transformações, para visualizar as melhorias dadas; e visualizar se diferentes transformações são melhores num modelo relativamente a outro.

Assim, permite-nos comparar os resultados óbtidos, desde os valores das métricas de avaliação, curvas de *accuracy*
loss à *performance* final dos modelos; após a aplicação da "melhor" *data augmentation* encontrada.

As transformações a aplicar sobre o modelo III serão, neste sentido, as seguintes:

1. **Rotação**;
2. **Zoom**;
3. **Espelhamento**;
4. **Alteração de brilho e contraste**;
5. **Adição de ruído**.

No contexto deste relatório, serão apenas descritos os detalhes relativos às transformações que acabaram por ser utilizadas no final: **rotação**, **zoom** e **alteração de brilho**.

4.1 Rotação

A **transformação de rotação** permite que as imagens sejam giradas num ângulo específico; o que torna-se útil para aumentar a variedade dos dados de treino e tornar o modelo mais robusto a diferentes orientações.

Tal como as restantes transformações, a transformação de rotação será implementada segundo a classe do *Keras* - *ImageDataGenerator*, na qual pode-se especificar o intervalo de ângulos de rotação a serem aplicados às imagens (*rotation_range*). Durante o treino, a cada *epoch*, as imagens são giradas aleatoriamente dentro do intervalo especificado.

Visualizando as imagens numa das *batches* (4.1), parece que as imagens geralmente têm a mesma orientação, pelo que o *range* da rotação escolhido foi razoavelmente pequeno. Neste sentido, o *range* máxima escolhido, em graus, foi de **20**.

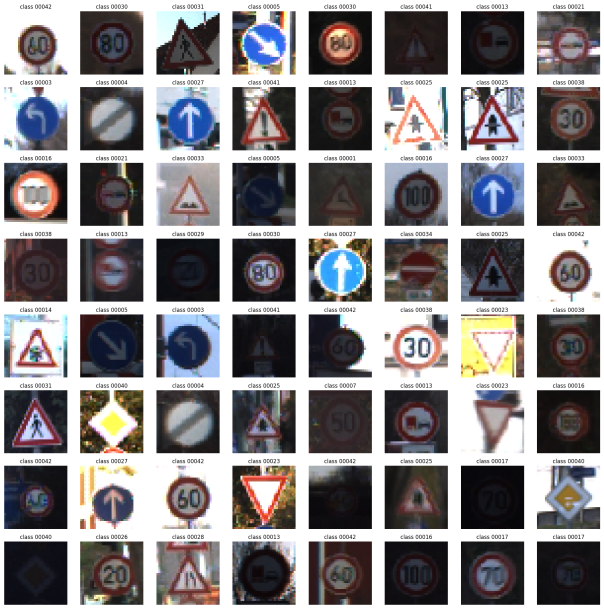

Figura 4.1: Visualização de uma *batch* de imagens.

Comparando os resultados do *fitting* (utilizando a transformação de rotação) com as óbtidas anteriormente (modelo III com apenas normalização), temos que:

- Os valores relativos ao *dataset* de treino (*accuracy* e *loss*) encontram-se um pouco a baixo dos óbtidos no *fitting* inicial; mas a diferença é insignificante, pelo que continuam muito bons.
- Por outro lado, os valores relativos ao *dataset* de teste obtiveram melhorias, tanto na *val_accuracy*, como na *val_loss*; agora são, respetivamente, 98.65% e 0.0702.

4.1.1 Resultados

Contudo, é também necessário sublinhar que a *performance*, no geral, diminui, pelo que o tempo por *epoch* passou a ser, em média, **50 segundos**. Estes valores provavelmente conseguiriam usufruir de mais *epochs*, considerando que não houve nenhum *early stopping* e, próximo do fim, ainda haviam melhorias no *loss* do *dataset* de validação.

Não obstante, consiste numa operação que pode ser usado no final, em termos de teste.

Visualizando as curvas de *accuracy* e *loss* (4.2 e 4.3), tem-se uma representação de um **bom fit**, em que os picos são quase suaves.

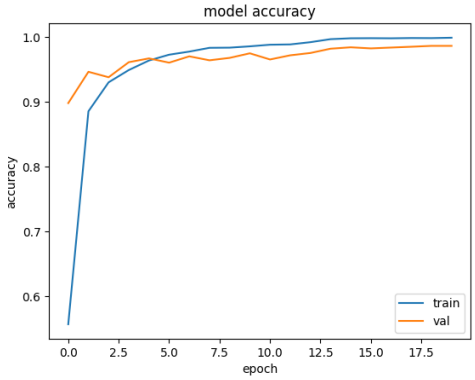

Figura 4.2: Rotação - Curva *Accuracy*

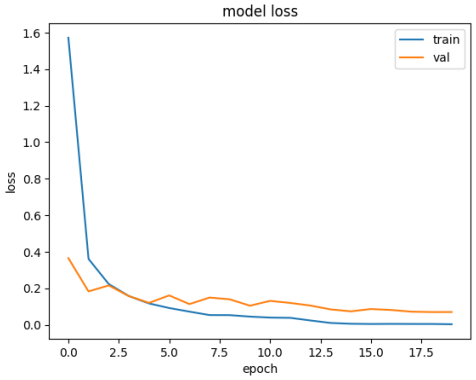

Figura 4.3: Rotação - Curva *Loss*

4.2 Zoom

A transformação de *zoom*, assim como a transformação de rotação, também é uma das transformações disponíveis na classe *ImageDataGenerator* do *Keras*. A transformação de *zoom* permite ampliar ou reduzir a escala das imagens; sendo esta escola escolhida arbitrariamente. Durante o treino, a cada *epoch*, as imagens são ampliadas ou reduzidas aleatoriamente dentro do intervalo especificado.

Neste caso, podemos definir, tal como na rotação, um *range* para os níveis de *zoom* que se pode efetuar nas imagens. O valor dado foi de **0.2**; o que implica que as imagens podem ser *zoomed in or out* por um fator até **20%**. Deste modo, consegue-se pequenas variações na escala dos objetos nas imagens, de modo a generalizar e lidar com variações de um modo mais eficiente.

4.2.1 Resultados

No que toca ao *zoom*, pode-se verificar que ocorre algo semelhante às rotações, n a medida em que os valores finais óbtidos têm claras melhorias relativamente ao modelo sem *data augmentation*, no que toca aos valores das métricas do *dataset* de validação.

O *val_accuracy* e *val_loss* tem valores muito satisfatórios e são um avanço; *98.89%* e *0.0606*, respetivamente.

A *performance* é outro aspeto que se assemelha à rotação, existindo um aumento de tempo, para **50 segundos**, em média. Contudo, no zoom ocorre um *early stopping*, que é visível nas curvas a baixo, para *accuracy* e *loss* (4.4 e 4.5), que não são as mais presentáveis; sendo que isto é devido ao facto do modelo não ter tido muito treino.

Não obstante isto, o zoom apresenta-se como uma transformação a adicionar no modelo final.

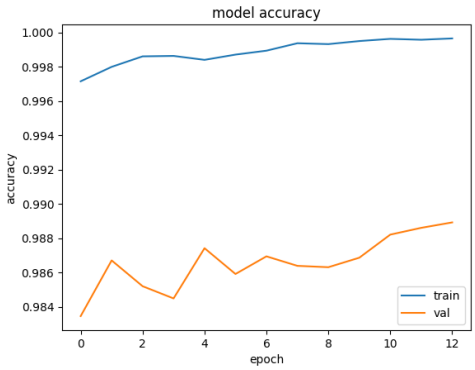

Figura 4.4: Zoom - Curva *Accuracy*

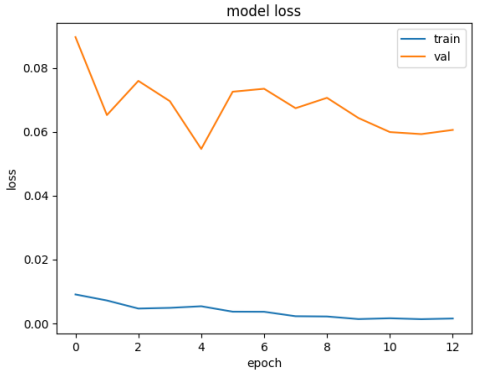

Figura 4.5: Zoom - Curva *Loss*

4.3 Alteração do brilho

Por fim, o "*brightness_range*"é uma técnica de *data augmentation* que ajusta o brilho das imagens durante o treino, permitindo aumentar ou diminuir o brilho das imagens; e adicionar variações ao conjunto de dados.

Este parâmetro é um tuplo que define a faixa de variação de brilho que será aplicada às imagens. Por exemplo, se for definido tal como *"brightness_range=(0.5, 1.5)"*; significa que o brilho das imagens pode variar entre 50% e 150% do brilho original.

Neste caso, tal como os exemplos anteriores foi-se com uns valores *default*, genéricos, para testar se o modelo teria ou não aversão a estas transformações.

Para simular variações de brilho que ocorram naturalmente nas imagens é importante escolher uma faixa de valores que seja realista. Observando o intervalo de brilho presente nas imagens do nosso *dataset*, tem-se que serão necessárias mudanças de contraste mais subtis; daí o range permanecer pequeno.

4.3.1 Resultados

Neste caso, tal como nas rotações e no *zoom*, encontram-se melhorias aqui, com as alterações do contraste das imagens.

Mais uma vez, tem-se:

- **Métricas do *dataset* de treino**: Permanecem-se equivalentes, ou até melhores, do que as do *fit* inicial, o que indica que consegue quase perfeitamente identificar as imagens do *dataset* de treino.

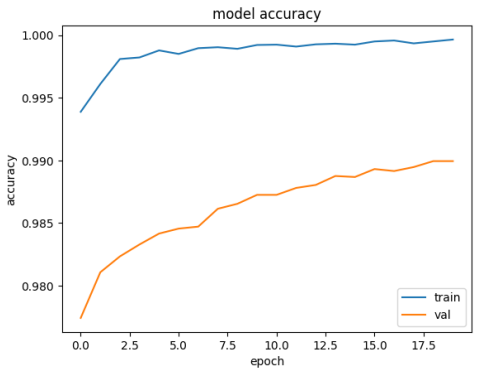

Figura 4.6: Alteração de brilho - Curva *Accuracy*

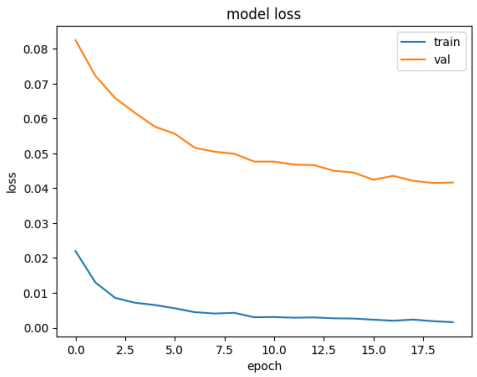

Figura 4.7: Alteração de brilho - Curva *Loss*

- **Métricas do *dataset* de teste**: Existem melhorias aqui, de certo modo significativas. Os valores para *val_accuracy* e *val_loss* encontram-se agora, respetivamente, como $98,99\%$ e 0.0416.

Adicionalmente, como uma exceção à regra, as alterações no brilho das imagens dos *datasets* não só trouxe uma melhoria clara na aprendizagem do modelo, como possui uma melhor *performance* comparada com os restantes exemplos; em média, **41 segundos** por **epoch** (próximo da realidade sem *data augmentation*).

As respetivas curvas de aprendizagem:

4.4 Resultados no modelo III

As novas alterações, implementadas como *data augmentation* trouxeram, respetivamente:

- *accuracy* - $0.9991, 0.9996, 0.9996$.
- *loss* - $0.0033, 0.00150.0016$.
- *val_accuracy* - $0.9865, 0.9889, 0.9899$.
- *val_loss* - $0.0702, 0.0606, 0.0416$.

Considerando os valores iniciais, temos que existe nos melhores dos casos, melhorias de:

- *accuracy* $+0.0003$.
- *loss* -0.0010.
- *val_accuracy* $+0.0060$.

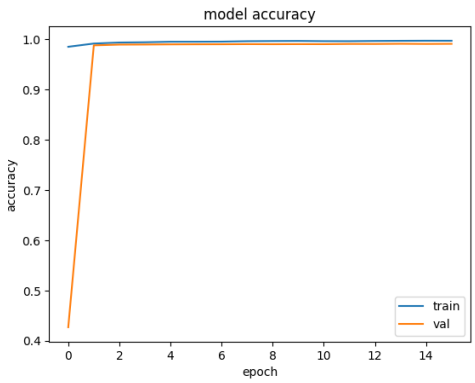

Figura 4.8: Modelo III com data augmentation - Curva *Accuracy*

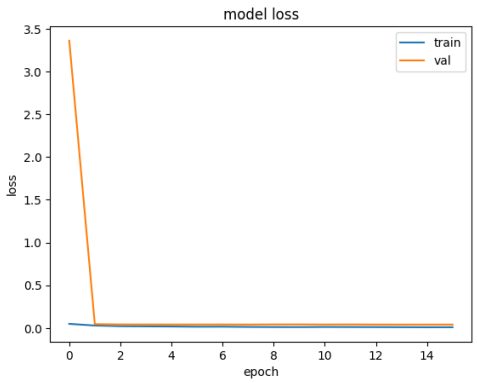

Figura 4.9: Modelo III com data augmentation - Curva *Loss*

- *val_loss* −0.05.

Sendo assim, irá-se utilizar, no final estas três transformações, para o **modelo III**, de modo a aumentar, com *data augmentation*, os valores das métricas desejados.

4.4.1 Resultados

Neste caso, como esperado, houve melhorias significativas com a aplicação das três transformações, em simultâneo, sobre o **modelo III**.

Não obstante os valores finais, que são todos melhorias relativamente aos óbtidos em testes prévios (*accuracy* = 99.71%, *loss* = 0.0096, *val_accuracy* = 99.10% e *val_loss* = 0.0408), as curvas de aprendizagem óbtidas para as métricas de *accuracy* e *loss* (4.8 e 4.9) demonstram que o modelo está a conseguir identificar as imagens de ambos os *datasets* **quase perfeitamente**.

Isto implica que haverá algum *overfitting*, dado que deve sempre haver algumas irregularidades e não linearidades, mas que, no contexto destes *datasets* (e deste trabalho), implicam muitos bons resultados.

4.5 Testes no modelo I e II

4.5.1 Modelo I

Apesar de não ter sido testado individualmente, com as respetivas transformações mencionadas anteriormente, consegue-se, puramente por observação das curvas de aprendizagem 4.10 e 4.11 (tal como dos valores das métricas finais), que o conjunto de transformações que funcionaram para o modelo III, também têm efeitos positivos no modelo I.

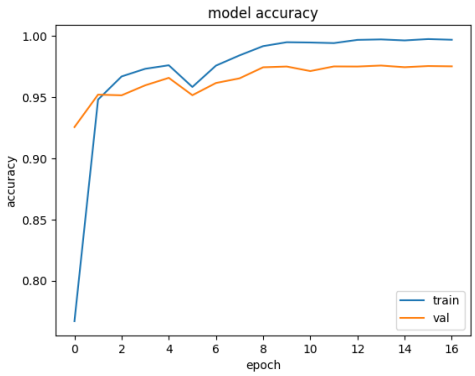

Figura 4.10: Modelo I com data augmentation - Curva *Accuracy*

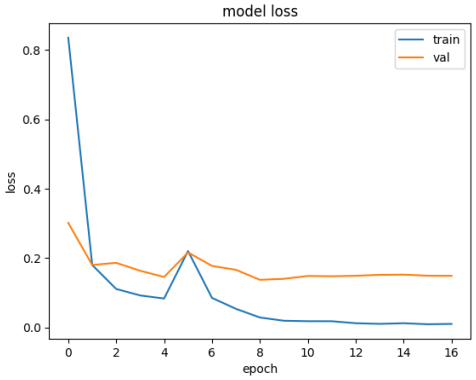

Figura 4.11: Modelo I com data augmentation - Curva *Loss*

Neste caso, o modelo apresenta melhores valores para as suas métricas como, por exemplo, a *accuracy* do *dataset* de teste, com um aumento significativo de aproximadamente 1%.

As curvas apresentam agora um *fit* mais saudável, se bem que continua com alguns picos acentuados, o que sinaliza um término demasiado rápido para a aprendizagem (de facto, ocorreu um *early stopping*).

Contudo, valores muito satisfatórios para o modelo I, que era muito simples na sua construção e simplicidade; apesar que a sua performance foi atingida (duração do dobro do tempo).

4.5.2 Modelo II

O modelo II, por outro lado, apesar de as métricas (no final), terem valores análogos ao seu treino anterior, ele apresenta curvas de *accuracy* e *loss* vastamente melhores; neste caso, de todos os modelos até então, deve ser o que apresenta o melhor *fit*.

Existe uma clara convergência entre a aprendizagem de ambos os *datasets*, sem haver claros sinais de *overfitting*. Apear de existirem alguns picos "pouco suaves", é uma boa representação para o *dataset*; sendo que os valores finais, apesar de não serem os melhores entre os testados, são bons vistos de um ponto de vista isolado.

Adicionalmente, a performance está análoga aos anteriores, quando aplicada a *data augmentation*.

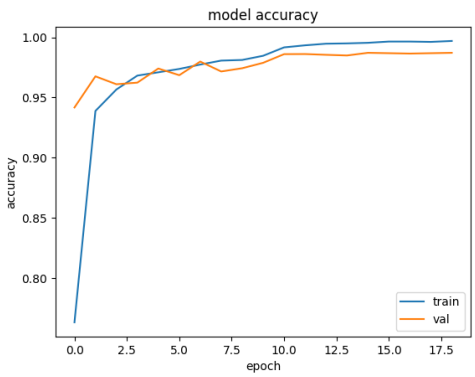

Figura 4.12: Modelo II com data augmentation - Curva *Accuracy*

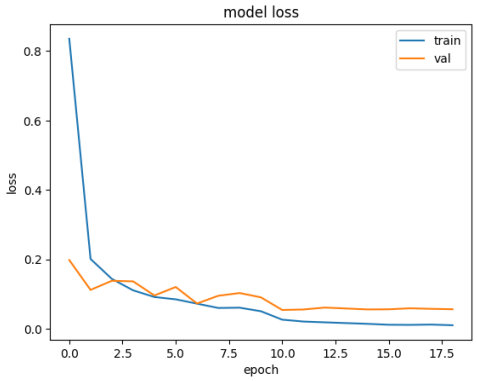

Figura 4.13: Modelo II com data augmentation - Curva *Loss*

Capítulo 5

Ensembles de redes

Os ensembles de modelos são uma técnica amplamente utilizada em aprendizagem para melhorar o desempenho e a robustez dos modelos. A abordagem envolve a combinação de múltiplos modelos individuais para realizar previsões mais precisas e confiáveis. O ensemble criado consiste em um conjunto de três modelos de rede neural convolucional (CNN). Cada modelo é criado usando a função create_model, que define a arquitetura da rede neural. Os modelos compartilham a mesma arquitetura, mas têm pesos inicializados aleatoriamente.

O treino dos modelos é realizado pela função train_models; por 20 épocas, sendo os pesos dos modelos salvos em arquivos para uso posterior.

Obteve-se para os três modelos resultados como:

1. 1º - loss: 0.0711 - accuracy: 0.9801 - 4s/epoch;
2. 2º - loss: 0.0670 - accuracy: 0.9816 - 3s/epoch;
3. 3º - loss: 0.0588 - accuracy: 0.9839 - 3s/epoch.

No final, recorrendo à biblioteca *Collections*, na função get_stats, $sãocalculadasváriasestatísticassobreasprevisõ$

5.0.1 Stacked Ensemble

Ainda tentou-se utilizar o stacked ensemble que, tal como no caso anterior, foi vastamente definido com base no código proporcionado pela equipa docente (e alterados detalhes para coincidir com o problema atual).

Stacked Ensemble é uma técnica de aprendizagem que combina as previsões de vários modelos individuais usando um modelo final chamado meta-modelo.

Novamente, o ensamble é construído a partir de três modelos individuais do tipo ensemble treinados anteriormente. Cada modelo individual fornece um conjunto de logits (saídas antes da função softmax) para cada imagem de teste.

O código coleta os logits de treino e label verdadeiros dos modelos individuais e utiliza-os para o treino; sendo este realizado ao longo de 2o epochs. Durante o treino, também são fornecidos os logits combinados de teste e os labels verdadeiros correspondentes como conjunto de validação.

Durante o treino, também são fornecidos os logits combinados de teste e os labels verdadeiros correspondentes como conjunto de validação.

Após o treino, as previsões são feitas usando o meta-modelo para os logits combinados de teste. Cada imagem de teste é classificada pela previsão com a maior probabilidade. Em seguida, a precisão é calculada comparando-se as previsões com os labels verdadeirqs correspondentes. O número total de previsões corretas e a precisão são exibidos como resultados finais.

No final, infelizmente, os resultados também não foram superiores ao melhor modelo, com o valor de $98,48\%$.

Capítulo 6

Conclusão

Deste modo, num contexto de visão por computadores, conseguimos criar três modelos, de complexidades e arquiteturas diferentes, os três com resultados satisfatórios; sendo uns melhores que os outros. O modelo III, por ventura, mesmo sem qualquer data augmentation possui resultados muitos bons.

A data augmentation, utilizando o lado de processamento de imagem, traz várias transformações, todas com o seu próprio impacto; e que no final do treino aumentaram o performance e detalhes de todos os modelos.

Por fim, ainda tentou se testar alguns exemplos de ensemble, mas sem conclusões benéficas.