Instituto Superior de Gestão e Administração de Santarém

Pós-Graduação em Data Science

Aprendizagem Profunda – Arquiteturas CNN

Carlos Silva, a22208573,
Rodrigo Antunes, a22203942,
Tiago Flor, a22210477

Docente:

Professor Doutor Ricardo Vardasca

Unidade Curricular: Sistemas Inteligentes

Santarém

Ano Letivo 2022-2023

Resumo

Este trabalho compreende a aplicação e análise de três modelos de Redes Neurais Convolucionais (CNN), LeNet, AlexNet e DenseNet de um conjunto e dados que contém imagens de abelhas. A base de dados contém imagens de abelhas com polén nas patas, abelhas a bater as asas para arrefecimento da colmeia, abelhas que contenham varroa – uma bactéria e vespas.

O trabalho começa com uma pequena apresentação dos modelos estudas. LeNet, criado por Yann LeCun em 1998, foi o pioneiro neste campo e serviu como modelo base do desenvolvimento dos restantes. AlexNet, criado por Alex Krizhevsky em 2012, revolucionou a visão computacional com uma arquitetura profunda e por fim, a DenseNet proposto por Huang em 2017, incorporando conexões densas melhorando a propagação de características entre camadas.

Posteriormente foi feito a implementação dos modelos em python e fez-se a comparação das suas performances na classificação de cada uma das imagens da base de dados em termos de 'loss', 'accuracy' e o tempo requerido para treinar os modelos.

Nos modelos criados foram usados diferentes parâmetros para explorar a sua influência nos resultados, tais como, o número de epochs, a função de activação, o 'batch size', o 'kernel' e 'dropout'.

Os resultados mostraram diferentes características de cada modelo. LeNet, com a sua arquitetura simples, foi o mais rápido dos modelos e obteve uma boa performance em termos de 'accuracy' , AlexNet, com as suas camadas profundas, com função de ativação e com a regularização do 'dropout' consegue melhor desempenho tanto no 'test loss' como no 'test accuracy' mas requerendo mais tempo de processamento que o modelo LeNet, e por fim, o modelo DenseNet, com as suas conexões densas, requer um poder de processamento muito superior aos outros modelos e consequentemente é o modelo que leva mais tempo a ser executado, contudo, consegue um bom balanceamento entre o 'accuracy' e a complexidade do modelo.

Palavras chave:alexnet;cnn;densenet;lenet;tensorflow.

Abstract

This work comprises the application and analysis of three models of Convolutional Neural Networks (CNN), LeNet, AlexNet and DenseNet of a set of data that contains images of bees. The database contains images of bees with pollen on their feet, bees flapping their wings to cool the hive, bees containing varroa – a bacteria and wasps.
The work begins with a small presentation of the studied models. LeNet, created by Yann LeCun in 1998, was a pioneer in this field and served as a base model for the development of others. AlexNet, created by Alex Krizhevsky in 2012, revolutionized computer vision with a deep architecture and finally, the DenseNet proposed by Huang in 2017, incorporating dense connections improving the propagation of features between layers.
Subsequently, the models were implemented in python and their performance was compared in classifying each of the images in the database in terms of 'loss', 'accuracy' and the time required to train the models.
In the models created, different parameters were used to explore their influence on the results, such as the number of epochs, the activation function, the 'batch size', the 'kernel' and 'dropout'.
The results showed different characteristics of each model. LeNet, with its simple architecture, was the fastest of the models and obtained a good performance in terms of 'accuracy', AlexNet, with its deep layers, with activation function and with the 'dropout' regularization, achieves better performance both in the 'test loss' as in the 'test accuracy' but requiring more processing time than the LeNet model, and finally, the DenseNet model, with its dense connections, requires a much higher processing power than the other models and consequently is the model that takes longer to run, however, achieves a good balance between 'accuracy' and model complexity.

Keywords:alexnet;cnn;densenet;lenet;tensorflow.

Índice

Índice de Figuras

Índice de Tabelas

INTRODUÇÃO

O tema do projeto apresentado no presente documento foi introduzido no âmbito da unidade curricular de Sistemas Inteligentes, tendo como objetivo principal a aplicação de um, ou vários, modelos de arquitetura CNN num conjunto de dados de imagens, fornecido pelo professor docente e proveniente da biblioteca *Tensorflow*.

O presente documento encontra-se dividido em 3 (três) partes, inicialmente apresentando-se uma descrição sumária do processo de importação e tratamento do *dataset*, seguido de uma breve explicação teórica, com recurso a bibliografia, dos modelos optados por construir e, por fim, respetivas conclusões com base na comparação entre os referidos modelos.

A base de dados selecionada apresenta um registo fotográfico de abelhas consoante a sua entrada/saída de uma colmeia, este expondo um conjunto de características sobre as mesmas, estas consistindo em abelhas que se encontram em estado de arrefecimento da colmeia por via de agitação de asas, abelhas portadoras de pólen e abelhas afetadas pelo ácaro *varroa*. Adicionalmente, o *dataset* fornece fotos de vespas, no sentido de providenciar uma distinção entre vespas e abelhas (“Bee_Dataset,” n.d.).

DESENVOLVIMENTO DO TEMA

Caracterização do conjunto de dados

O conjunto de dados em análise é composto por fotografias de abelhas e vespas e é fornecido pela plataforma do TensorFlow. Este conjunto de dados contém imagens e rótulos que identificam certas características, como infeções, transporte de pólen ou abelhas a bater as suas asas. Para poder distinguir entre abelhas e vespas, foram adicionadas imagens desta última.

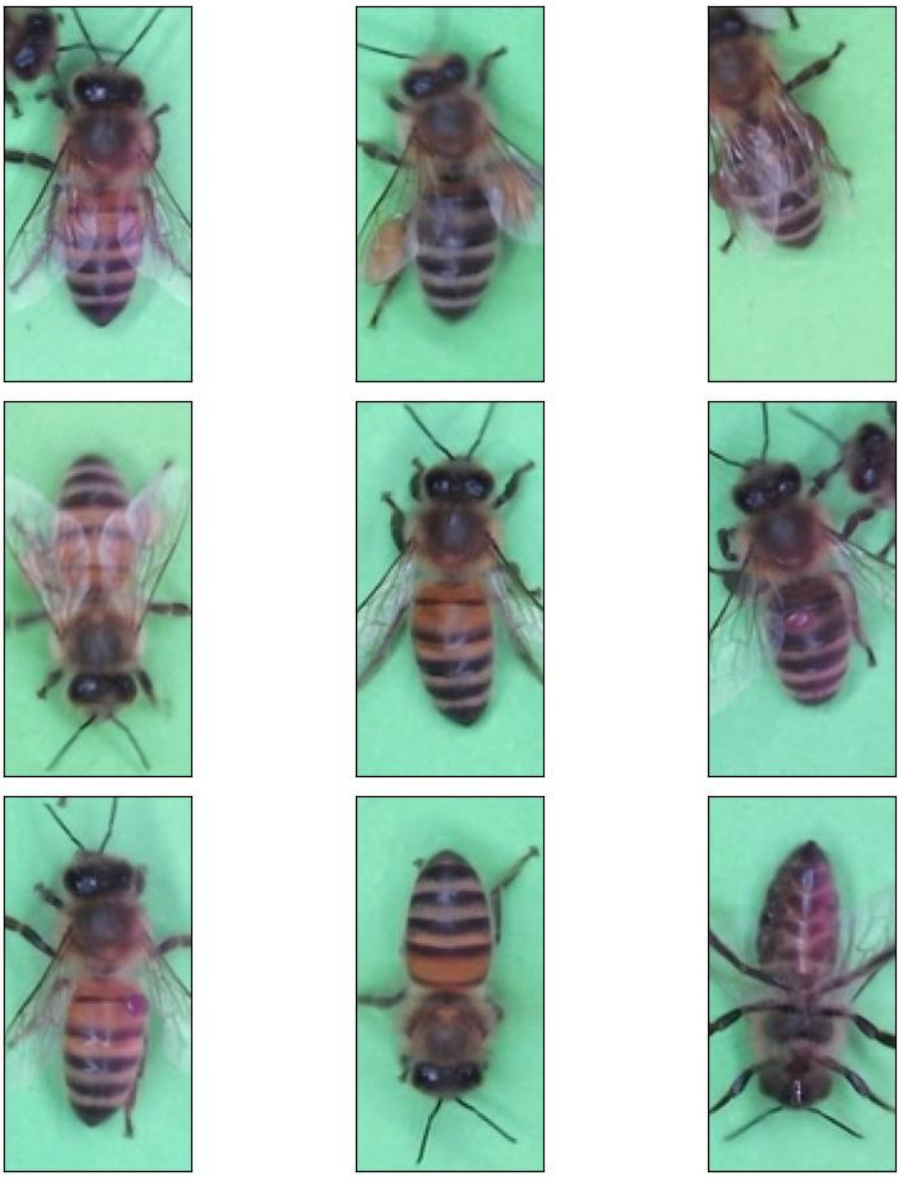

Figura 1 – Exemplos de imagens do dataset

As imagens das abelhas são tiradas de cima e rodadas. A imagem da abelha é vertical e sua cabeça ou tronco está no topo. Todas as imagens foram tiradas com fundo verde e a distância até as abelhas foi sempre a mesma, de forma que todas possam ficar com o mesmo tamanho.

Os rótulos do dataset são compostos por quatro classes distintas:

- cooling_output – Determina se a abelha está a bater as asas,
- pollen_output – Determina se a abelha está a carregar pólen,
- varroa_output – Determina de a abelha está infetada,
- wasps_output – Determina se a abelha é na verdade uma vespa.

Quando a classe é 1, significa que essa característica está presente na imagem, caso contrário, não está presente.

Cada uma das imagens deste dataset pode ter mais que uma característica associada. Ou seja, uma abelha pode ao mesmo tempo ter infeções e estar a bater as assas. Da mesma forma, também há imagens em que a abelha não tem nenhuma das características assinaladas.

Este conjunto de dados é composto por 5790 imagens, e todas elas têm uma resolução de 150px de altura, por 75px de comprimento. Todas as imagens são a cores. Desta forma, a camada de input dos modelos gerados, tem a forma de (150,75,3).

Pré-processamento

De forma a ser possível a utilização dos dados nos diversos algoritmos, foi necessário fazer um tratamento destes. Para este dataset, como as classes estavam dispersas por quatro colunas, foi necessário uni-las numa só.

Tabela 1 – Exemplo distribuição por classe

Classe	output/ cooling_output	output/ pollen_output	output/ varroa_output	output/ wasps_output
Valor	0.0	0.0	0.0	1.0
	0.0	0.0	1.0	0.0

Para isso, em cada linha, foram concatenadas as várias classes com valores a 1 e nos casos em que todas as classes estavam a 0, a imagem foi classificada como "sem classe". Por fim, ambas as imagens e as classes, foram convertidas num array. Desta forma, foram geradas nove classes únicas.

Tabela 2 – Classes geradas no pré-processamento

Classes Geradas
'cooling_output'
'wasps_output'
'cooling_output pollen_output'
'cooling_output pollen_output wasps_output'
'cooling_output varroa_output'
'pollen_output'
'pollen_output varroa_output'
'sem classe'
'varroa_output'

O último passo nesta fase, passou pela conversão do texto das classes, em valores numéricos, através de um LabelEncoder, o qual gera um valor numérico único por cada uma das classes encontradas.

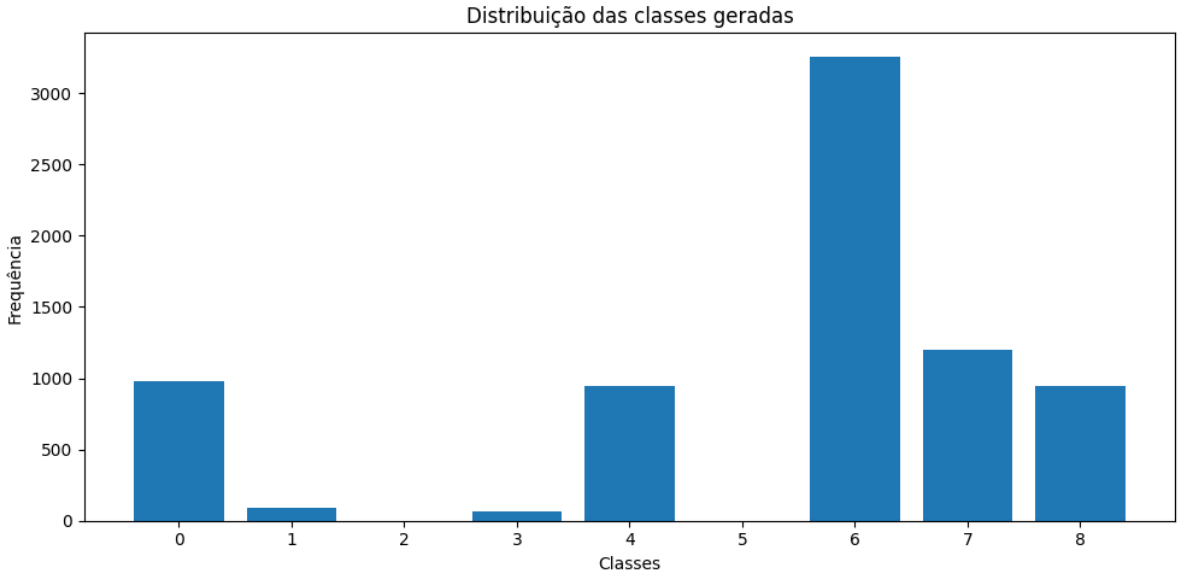

Figura 2 – Distribuição das imagens pelas novas classes

Por fim, quer as imagens, quer as classes, foram, convertidos em uma Binary Class Matrix, o que irá permitir que se use a classe CategoricalCrossentropy, para determinar a perda dos modelos gerados.

Execução dos modelos

LeNet

O modelo LeNet-5 foi desenvolvido por Yann Lecun e os seus colegas na década de 1990, foi um dos modelos pioneiros no desenvolvimento das redes neurais convolucional (CNN) para classificar imagens. Foi primeiramente pensado para processar dígitos escritos à mão.

O modelo LeNet segue os seguintes princípios:

- Camadas Convolucionais (Convolutional Layer)
    - O modelo começa com uma ou mais camadas convolucionais às imagens de entrada.
    - As camadas convolucionais consistem na passagem de um filtro (kernel) na imagem de entrada e na extração de características locais numa imagem, gerando posteriormente um mapa de características.
    - O filtro aprende a detetar padrões ou características da imagem através da aplicação de funções de ativação não lineares à imagem de entrada.
- Camada de agrupamento (Pooling Layer)
    - Após cada camada convolucional é sempre usada uma camada de agrupamento para reduzir a dimensão do mapa de características.
    - Agrupamento (maior parte das vezes agrupamento máximo) envolve em dividir o mapa de características em pequenas regiões e posteriormente selecionar o valor máximo dessa região.
    - O agrupamento ajuda a diminuir o mapa de características retendo as características mais proeminentes de cada região, ajudando a diminuir a quantidade de processamento requerido ao computador.
- Camada achatamento (Flattening Layer)
    - A saída da última camada de agrupamento é ‘achatada’ num vetor de uma dimensão.
    - Esta transformação é feita para que se possa fazer o processamento nas camadas totalmente ligadas.
- Camadas totalmente ligadas (Fully connected Layers)

- O vetor achatado (vector 1 dimensão) ~e conectado a uma ou mais camadas totalmente ligadas.
- Cada neurónio de uma camada totalmente ligada vai se encontrar conectada a cada neurónio da camada anterior.
- Funções de ativação são aplicadas às saídas destas camadas para introduzir uma não-linearidade.

- Camada de saída (Output Layer)
  - A última camada totalmente ligada é a camada onde se obtém as previsões.
  - Em classificações, normalmente, é usada a função de ativação 'softmax', que prevê a probabilidade de cada classe.

O modelo LeNet faz uso das camadas convulacionais para extrair características locais, camadas de agrupamento para reduzir os mapas de características e uma camada totalmente ligada para combinar características para fazer previsões. Esta abordagem hierárquica ajuda a que este modelo consiga classificar objetos ou padrões complexos.

Modelo Gerado

O modelo LeNet criado para a classificação da base de dados 'bee_dataset' foi composto pelo seguinte,

1. Camada de entrada:
   - Camada de entrada com imagens de 150x75 e 3 canais de cor (RGB).
2. Camadas Convolucionais:
   - A primeira camada convolucional tem 6 filtros com o tamanho do kernel de 5x5 e é aplicada uma função de ativação entre 'relu', 'sigmoid' e 'tanh'.
   - A segunda camada convolutional tem 16 filtros com um kernal de 5x5 e a mesma função de ativação que a primeira camada convolucional.
   - A terceira camada convolutional tem 16 filtros com um kernal de 5x5 e a mesma função de ativação que a primeira camada convolucional.
3. Camadas de agrupamento:

- Após cada camada convolucional é aplicada uma camada de agrupamento máximo de tamanho 2x2 para reduzir as suas dimensões e extrair as características predominantes.

4. Camada de achatamento:

    - A seguir a última camada de agrupamento é usada a camada de achatamento para fazer a transformação num vetor de 1 dimensão usado para as camadas seguintes, as camadas totalmente ligadas.

5. Camadas totalmente ligadas:

    - A camada de achatamento é ligada a uma camada totalmente ligada de 120 unidades e utilizada uma função de ativação.
    - Seguidamente de outra camada totalmente ligada de 84 unidades e com a mesma função de ativação.
    - A camada final tem 10 unidades (correspondendo às 10 classes da nossa base de dados) e usando a função de ativação 'softmax' calcula a probabilidade de cada classe.

No modelo criado foi implementado um loop que percorre diferentes funções de ativação, sendo elas a 'relu', 'sigmoid' e 'tanh'. Cada função de ativação foi aplicada nas camadas convolucionais e nas camadas totalmente ligadas. É igualmente usado na compilação do modelo a função 'cross-entropy loss', o optimizador 'Adam' e a métrica 'accuracy'. (Figura 3)

O treino do modelo é feito aos dados de treino usado um "batch size" de 32, 128 e 512, assim como epochs de 1, 5, 10 e 15. Já a avaliação do modelo é feita no 'testing dataset' através do cálculo do 'test loss' e da 'accuracy'.

Depois do modelo treinado e testado foi usado uma imagem modificada da base de dados para fazer a previsão da classe ('Classe da Previsão') e o resultado da probabilidade dessa previsão ('Probabilidade da Previsão').

Por fim, é guardado num dicionário o 'test loss', 'test accuracy', o tempo total de treino, o batch size utilizado, os epochs e a função de ativação utilizada, assim como também os resultados da previsão efetuada. Este dicionário é ainda convertido para um dataframe para posterior utilização na elaboração de gráficos para análise.

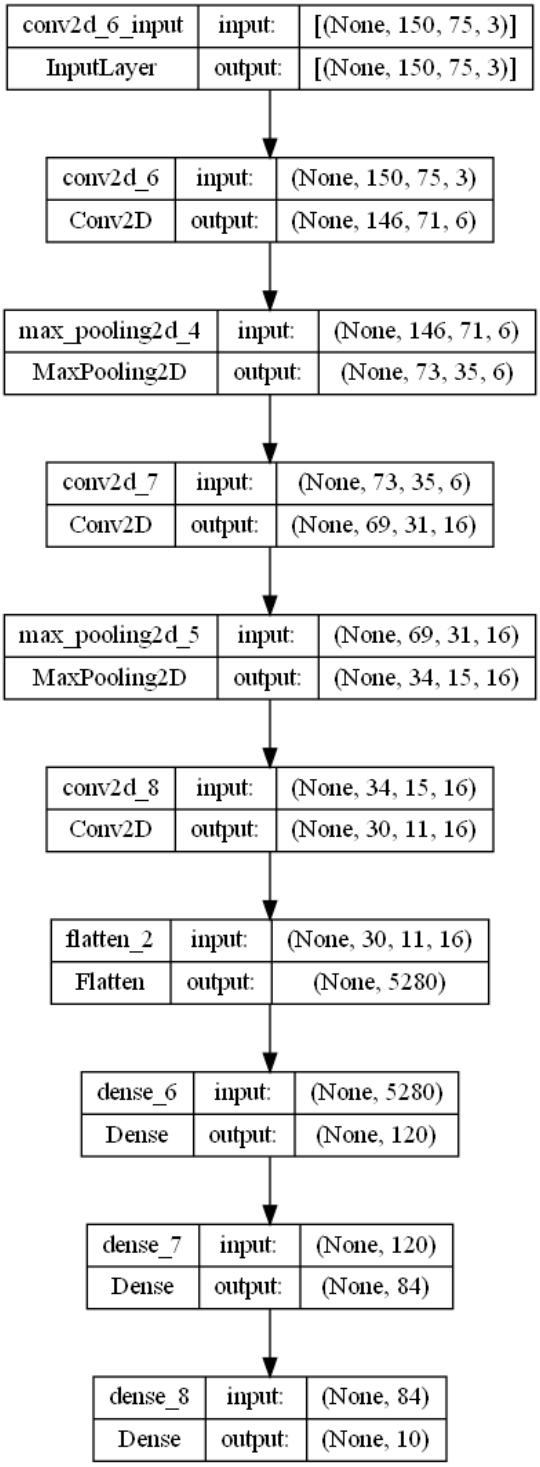

Figura 3 – LeNet Modelo

Resultados

Na tabela seguinte podemos ver os resultados conseguidos na execução deste modelo à base de dados bee_dataset.

Tabela 3 - Resultados Testes LeNet

Activation	Batch Size	Epochs	Perda	Precisão	Tempo (s)	Previsão	Previsão (%)
relu	32	1	0,590	0,773	18,139	sem classe	98,24
relu	32	5	0,558	0,838	80,841	pollen_output	69,86
relu	32	10	0,913	0,859	158,521	pollen_output	99,97
relu	32	15	0,959	0,865	245,966	sem classe	92,8
relu	128	1	1,358	0,872	15,880	sem classe	79,19
relu	128	5	1,299	0,871	69,434	pollen_output	98,06
relu	128	10	1,313	0,861	136,518	pollen_output	100
relu	128	15	1,359	0,893	204,872	pollen_output	93,34
relu	512	1	1,392	0,883	14,052	pollen_output	99,95
relu	512	5	1,793	0,880	66,966	pollen_output	99,95
relu	512	10	1,828	0,894	132,165	pollen_output	61,41
relu	512	15	1,753	0,888	197,748	pollen_output	99,9
sigmoid	32	1	1,551	0,434	16,288	sem classe	44,71
sigmoid	32	5	1,548	0,434	79,961	sem classe	44,82
sigmoid	32	10	1,557	0,434	158,107	sem classe	48,62
sigmoid	32	15	1,557	0,434	281,775	sem classe	39,56
sigmoid	128	1	1,554	0,434	35,984	sem classe	45,88
sigmoid	128	5	1,551	0,434	153,965	sem classe	43,46
sigmoid	128	10	1,551	0,434	292,429	sem classe	42,62

sigmoid	128	15	1,551	0,434	402,321	sem classe	41,89
sigmoid	512	1	1,551	0,434	27,347	sem classe	43,56
sigmoid	512	5	1,551	0,434	113,194	sem classe	44,44
sigmoid	512	10	1,551	0,434	234,187	sem classe	43,46
sigmoid	512	15	1,551	0,434	349,316	sem classe	43,94
tanh	32	1	0,727	0,685	28,321	cooling_output pollen_output wasps_output	35,79
tanh	32	5	0,561	0,768	137,151	wasps_output	99,46
tanh	32	10	1,044	0,636	560,336	wasps_output	40,04
tanh	32	15	0,465	0,831	808,332	wasps_output	83,83
tanh	128	1	0,506	0,819	48,627	wasps_output	98,27
tanh	128	5	0,505	0,834	221,421	wasps_output	67,61
tanh	128	10	0,460	0,863	418,808	wasps_output	94,09
tanh	128	15	0,620	0,843	580,298	wasps_output	98,26
tanh	512	1	0,544	0,866	40,653	wasps_output	97,59
tanh	512	5	0,579	0,851	142,065	wasps_output	99,85
tanh	512	10	0,629	0,866	241,414	wasps_output	99,23
tanh	512	15	0,685	0,863	243,441	wasps_output	99,51

Na figura Figura 4 podemos observar que tivemos uma melhor precisão (accuracy) quando usando um bach size maior e com epochs igual a 15, o que também podemos observar na Figura 5 onde a função com melhor desempenho na precisão foi a função de ativação ‘relu’.

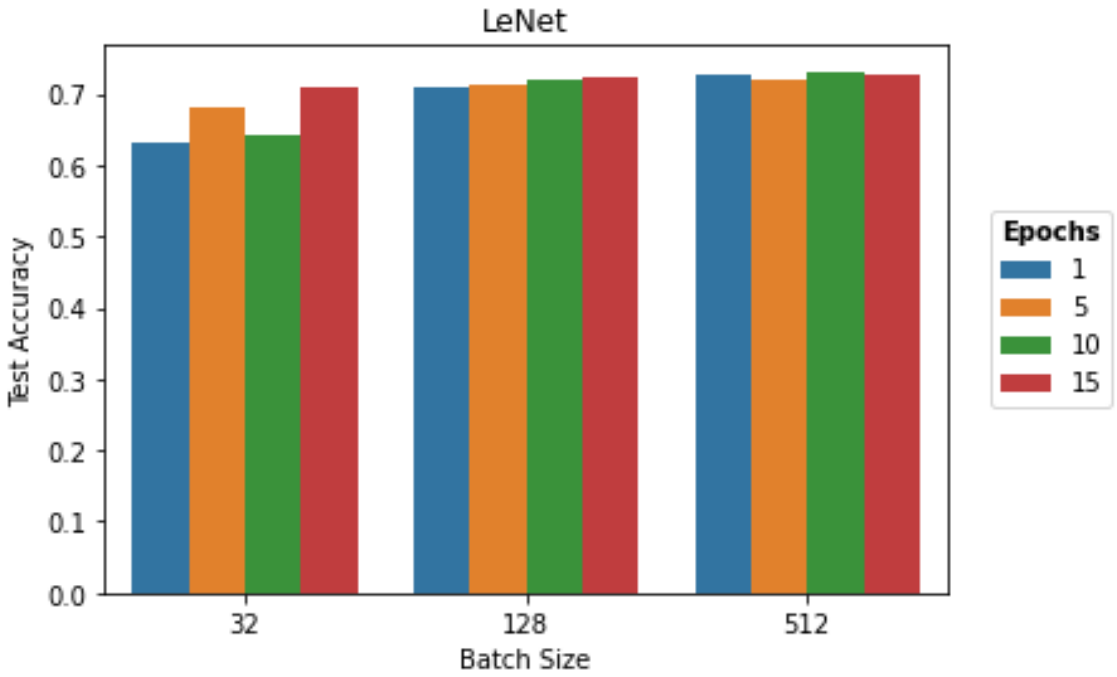

Figura 4 – Resultados Testes batch_size/epochs

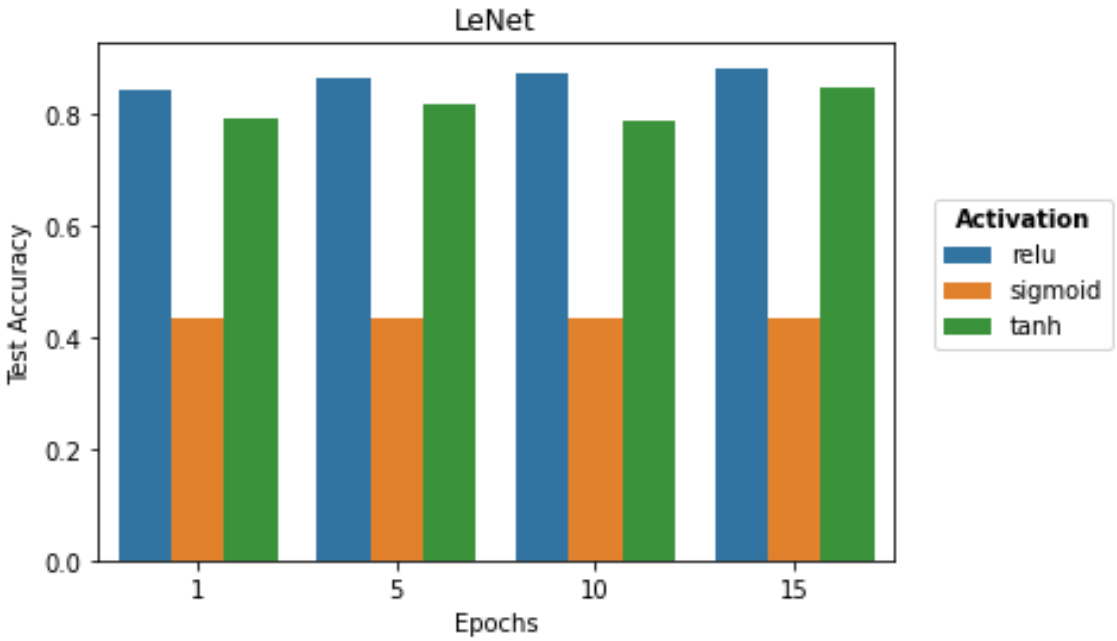

Figura 5 – Resultados Testes ativação/epochs

Relativamente ao test loss a função de ativação que teve um pior desempenho foi a função 'sigmoid', já a função de ativação 'tanh' obteve melhores resultados em todas as variações de epochs, como podemos observar na Figura 6.

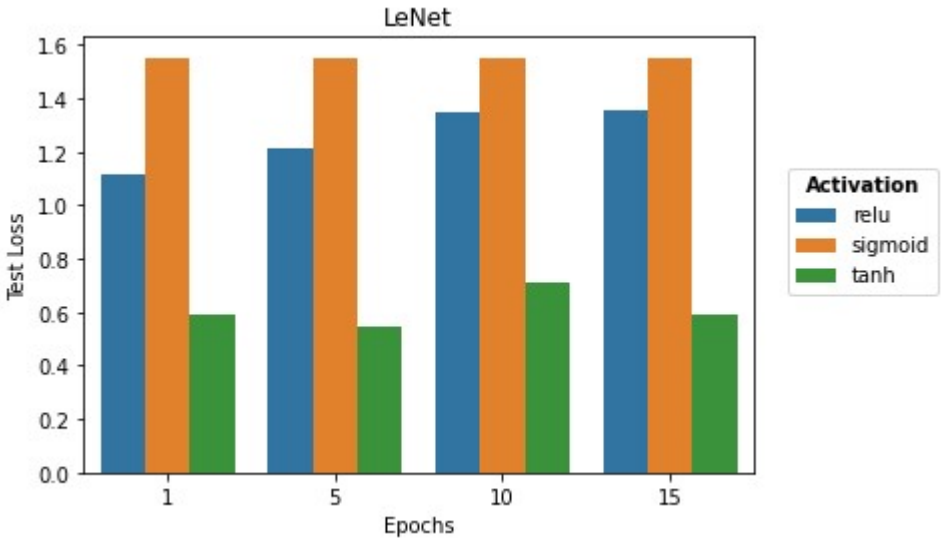

Figura 6 – Resultados Testes loss/ativação

No tempo de execução, a função de ativação 'tanh' demora mais tempo como podemos observar no gráfico da Figura 7, como era de esperar, usando mais epochs também aumenta o tempo de execução do modelo.

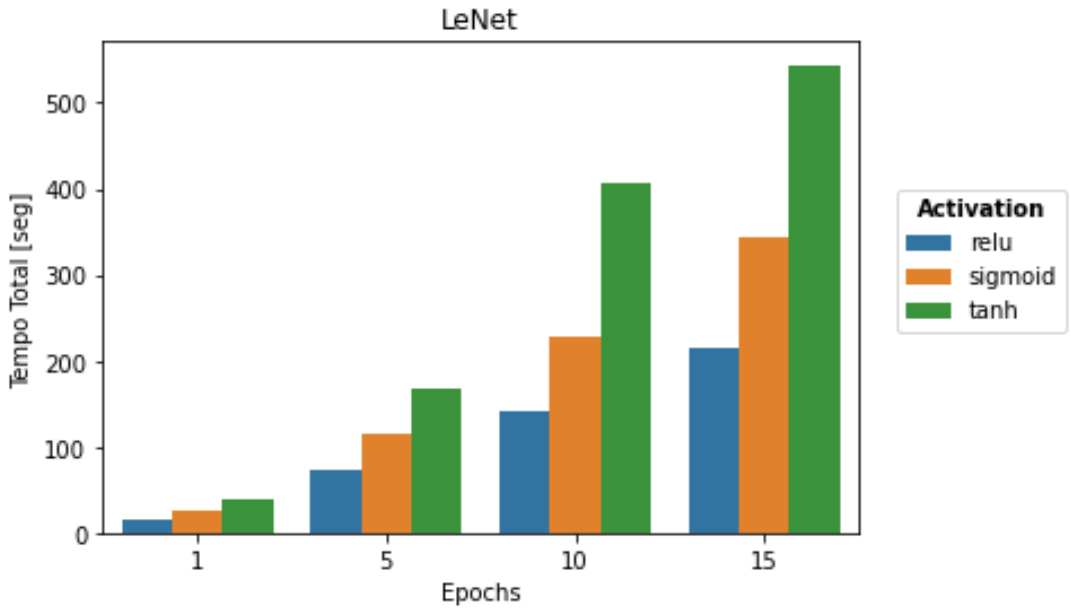

Figura 7 – Resultados Testes tempo/ativação

Na previsão efetuada em cada variação do modelo a função de ativação ‘relu’ foi a que obteve sempre uma previsão com probabilidade maior. Por sua vez, a função ‘sigmoid’ foi a que teve sempre o pior resultado, como mostrado na Figura 8.

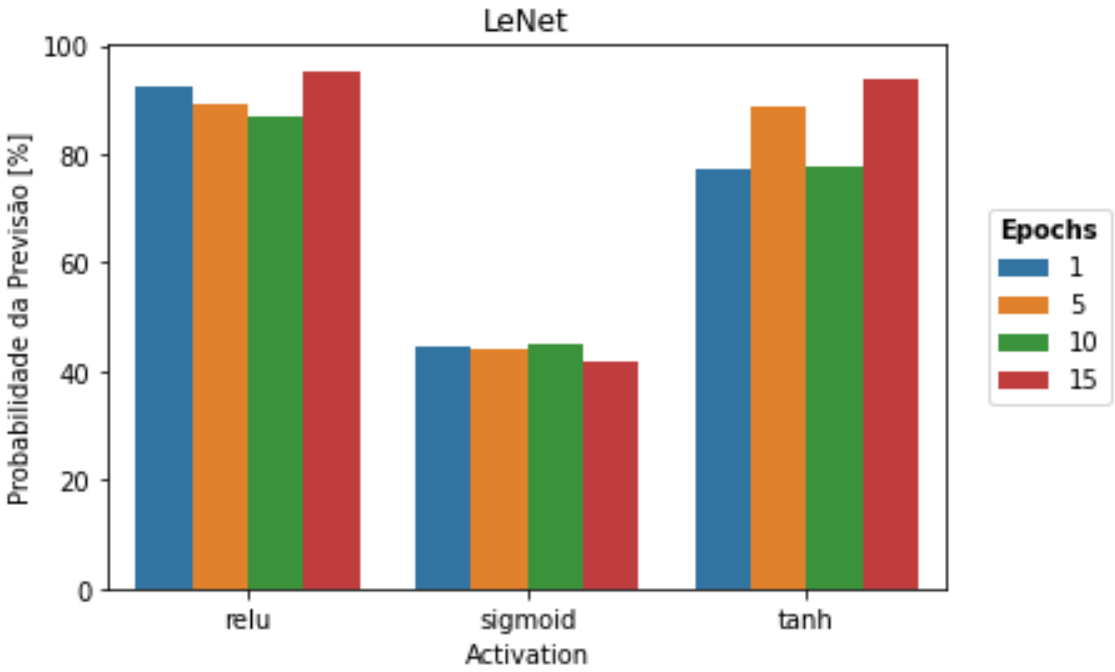

Figura 8 – Resultados Testes previsão/ativação

AlexNet

Trata-se de uma arquitetura CNN projetada por Alex Krizhevsky, Ilya Sutskever e Geoffrey Hinton e representa um avanço significativo no ramo da aprendizagem profunda. É constituído por oito camadas, das quais cinco camadas convolucionais e três camadas FC. O desempenho significativo deste modelo é alcançado através da utilização de técnicas como unidades lineares retificadas (ReLU), *pooling* sobreposto e regularização por *dropout* (CS231n Convolutional Neural Networks for Visual Recognition, n.d.).

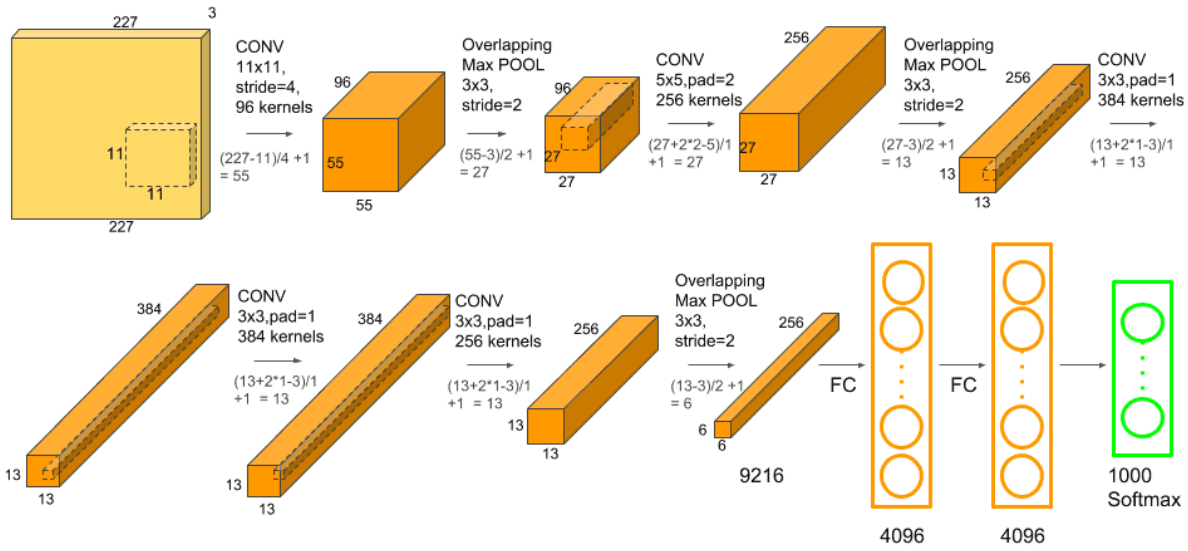

Figura 9 – Diagrama estrutural rede AlexNet (Nayak & Nayak, 2021).

No sentido de realização de testes de desempenho desta arquitetura, foi utilizado o modelo supra apresentado como modelo base de análise, sendo que futuras comparações em termos de performance (precisão versus tempo de execução) de teste de parâmetros assumem esta base como denominador comum.

Funções de ativação

Relativamente às funções de ativação em vigor nas camadas ocultas do modelo, foram consideradas as Funções ReLu e Leaky ReLu, em detrimento de *Step*, por não se adequar a modelos de *deep learning*, assim como *Tanh*, devido ao reduzido uso corrente. Nas camadas de output optou-se pela utilização da função Softmax, uma vez que, ao contrário da função *Sigmoid* que melhor se adapta a classificação binária, por ser utilizada na classificação multi classe (Sharma, 2022).

Modelo Gerado

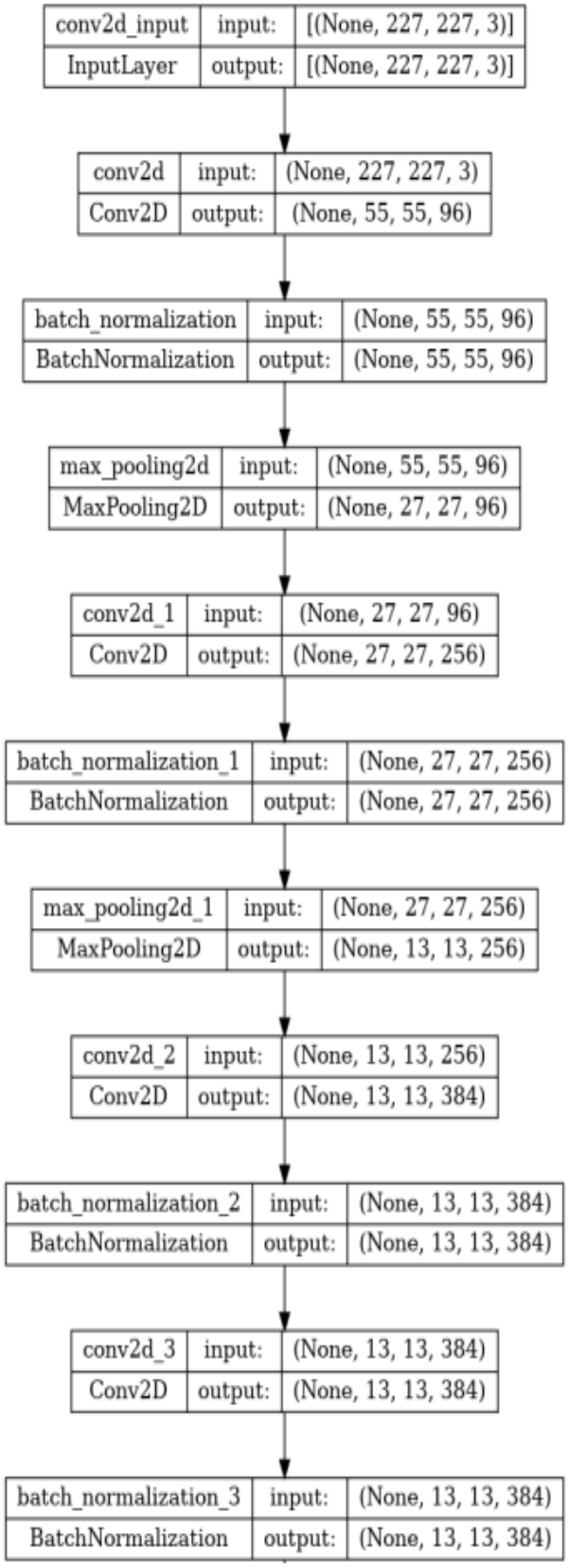

Figura 10 - AlexNet Modelo

Camada de Convolução 1: Aplica 96 filtros de tamanho 11x11 com stride de 4, extraindo características de baixo nível como arestas e gradientes (Krizhevsky et al., 2012).

Camada de Convolução 2: Utiliza 256 filtros de tamanho 5x5 com stride de 1 e aplica normalização de resposta local (LRN) para realçar as características e capturar padrões e texturas com maior complexidade (Krizhevsky et al., 2012).

Camada de Convolução 3: Possui 384 filtros de tamanho 3x3 com *stride* de 1, aumentando a capacidade da rede para capturar características intricadas (Krizhevsky et al., 2012).

As duas primeiras camadas de convolução, assim como a quinta, são, cada uma, seguidas de uma camada de *max pooling*, de 3x3 e *stride* de 2 (*8.1. Deep Convolutional Neural Networks (AlexNet) — Dive Into Deep Learning 1.0.0-beta0 Documentation*, n.d.).

Camada de Convolução 4: Com 384 filtros de tamanho 3x3, extrai características e padrões de alto nível (Krizhevsky et al., 2012).

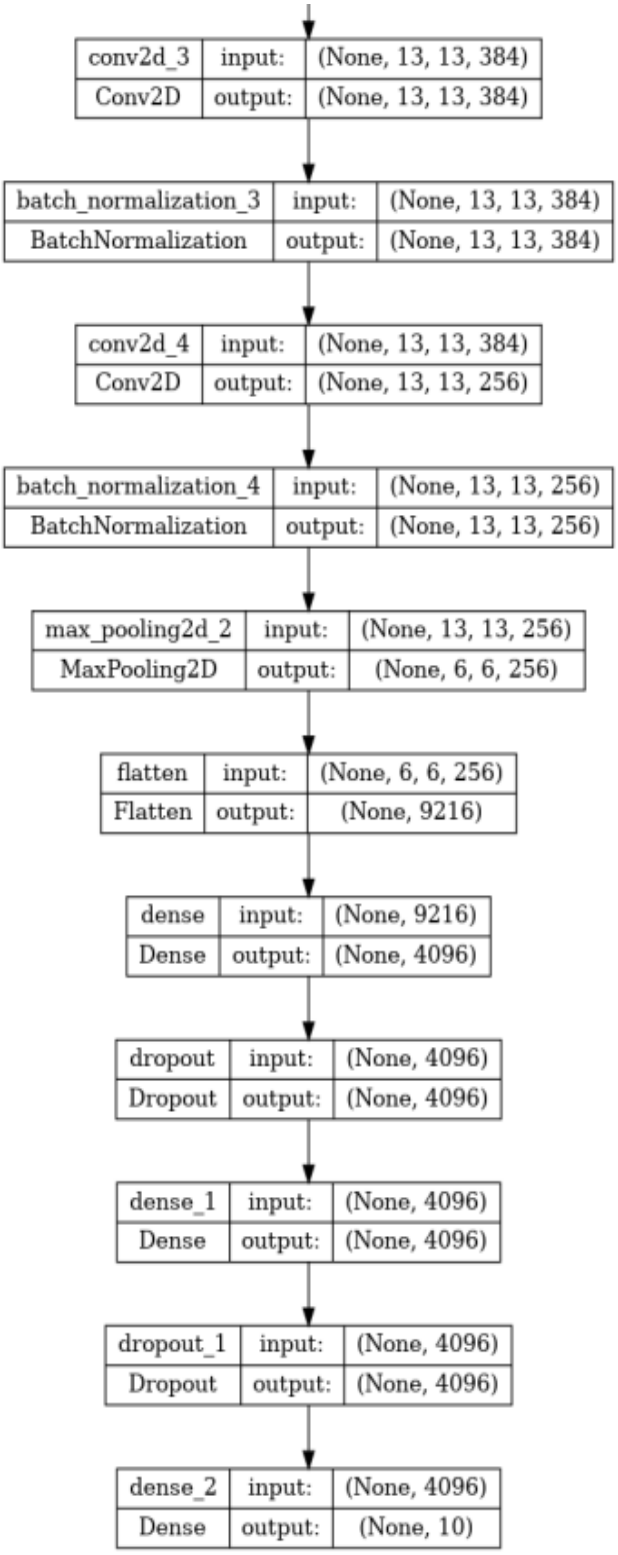

Figura 11 - AlexNet Modelo 2

Camada de Convolução 5: Aplica 256 filtros de tamanho 3x3, capturando características complexas e abstraindo informações de alto nível (Krizhevsky et al., 2012).

Camada FC 1: Consiste em 4096 neurónios, recebendo entrada das camadas anteriores e extraindo características de alto nível (Krizhevsky et al., 2012).

Camada FC 2: Também com 4096 neurónios, refina as características obtidas, preparando-as para a classificação final (Krizhevsky et al., 2012).

Camada de Saída: Última camada totalmente conectada com 1000 neurónios, representando as 1000 classes do conjunto de dados ImageNet, produzindo as probabilidades de saída para determinar a classe prevista da imagem de entrada (Krizhevsky et al., 2012).

Resultados

Como pressupostos de teste do efeito do impacto de diferentes parâmetros na arquitetura do modelo, foram considerados os fatores do número de *epochs*, ou seja, o número de passagens que o modelo executa sobre si mesmo, o fator de perda de dados, a precisão e tempo de execução. Adicionalmente, foi testado o efeito dos rendimentos decrescentes relativamente ao número de *epochs versus* ganho de precisão e acréscimo de tempo de execução.

Tabela 4 – Teste de modelo padrão

Teste	Epochs	Perda	Precisão	Δ (p)	Tempo Execução (s)	Δ (t)	Δ (p) / Δ (t)
1	5	43.59%	81.83%	-	656	-	
	10	16.45%	94.69%	15.72%	1309	99.54%	0.16
	15	11.35%	97.10%	2.55%	1952	49.12%	0.05

Conforme execução da arquitetura padrão do modelo *AlexNet*, verifica-se uma precisão até 97,10% com a utilização de quinze *epochs*. A duplicação de cinco para dez epochs produz um rendimento de 15,72% em termos de precisão com aproximadamente o dobro do tempo de execução, enquanto a seleção de quinze *epochs* incrementa a precisão em 2,55% adicionais. O rácio de variação de precisão sobre variação de tempo de execução apresenta-se em 0.16 e 0.05 com dez e quinze *epochs*, respetivamente, evidenciando-se que a utilização de *epochs* acima do valor máximo selecionado já evidencia forte efeito de rendimentos decrescentes.

Tabela 5 – Funções de ativação: ReLu vs. Leaky_ReLu

Teste	Epochs	Perda	Precisão	Δ (p)	Tempo Execução (s)	Δ (t)	Δ (p) / Δ (t)
2	5	44.76%	82.81%	-	677	-	
	10	14.61%	95.71%	15.58%	1389	105.17%	0.15
	15	12.94%	97.14%	1.49%	2075	49.39%	0.03

A alteração da função de ativação do modelo base para *Leaky_ReLu* resulta num ganho, ainda que pouco significativo, de precisão, com observado acréscimo no tempo total de execução. A maior variação na precisão regista-se com utilização de dez *epochs*. Uma vez que se obtém maior grau de precisão com acréscimo pouco relevante no tempo de execução, considerou-se a função *Leaky_ReLu* nos restantes testes.

Tabela 6 – Efeito de redução de filtro das camadas de convolução

Teste	Epochs	Perda	Precisão	Δ (p)	Tempo Execução (s)	Δ (t)	Δ (p) / Δ (t)
3	5	43.87%	81.83%	-	410	-	
	10	20.31%	93.48%	14.24%	812	98.05%	0.15
	15	9.48%	97.90%	4.73%	1222	50.49%	0.09

Conforme apresentado, a redução o tamanho do filtro em um patamar de grandeza nas camadas dois a cinco de convolução apresenta uma redução significante no tempo de execução do modelo, de 2075 segundos registados na Tabela 2, para 1222 segundos. Adicionalmente, verifica-se alguma perda, com reduzida relevância, de precisão com cinco e dez *epochs* e um ganho de precisão a partir de quinze *epochs*, estas apresentando um rácio de rendimentos decrescentes inferior, verificando-se ganhos superiores com utilização de *epochs* adicionais.

Tabela 7 – Aumento do tamanho do kernel

Teste	Epochs	Perda	Precisão	Δ (p)	Tempo Execução (s)	Δ (t)	Δ (p) / Δ (t)
4	5	51.86%	75.31%	-	1565	-	
	10	40.54%	86.79%	15.24%	3131	100.06%	0.15
	15	19.66%	93.93%	8.23%	4780	52.67%	0.16

Uma alteração do tamanho do *kernel* na segunda camada de convolução de 5px para 7px e nas camadas terceira a quinta, de 3px para 5px resulta num aumento do tempo de execução do modelo base superior a 2x. Não obstante, não se observa qualquer ganho em termos de precisão, com valores ligeiramente inferiores. O rácio de rendimentos decrescentes diminui quanto maior o número de *epochs* selecionado, pelo que ainda será possível obter graus superiores de precisão com esta arquitetura, ainda que o tempo de execução seja consideravelmente superior.

Tabela 8 – Redução do tamanho do kernel

Teste	Epochs	Perda	Precisão	Δ (p)	Tempo Execução (s)	Δ (t)	Δ (p) / Δ (t)
5	5	33.28%	87.72%	-	364	-	
	10	16.75%	95.54%	8.91%	727	99.73%	0.09
	15	18.14%	95.54%	0.00%	1105	51.99%	0.00

Inversamente à tabela anterior, a redução do tamanho do *kernel* face ao modelo base resulta num decréscimo no tempo total de execução, assim como o maior valor de precisão observado com dez *epochs*. Não obstante, contrariamente aos restantes, este modelo não apresenta qualquer ganho observável de precisão acima de dez *epochs*.

Tabela 9 – Tamanho kernel reduzido + filtro reduzido camada convolução

Teste	Epochs	Perda	Precisão	Δ (p)	Tempo Execução (s)	Δ (t)	Δ (p) / Δ (t)
6	5	40.31%	83.84%	-	274	-	
	10	14.29%	95.94%	14.43%	548	100.00%	0.14
	15	10.32%	97.14%	1.25%	813	48.36%	0.03

Este quadro representa a combinação dos fatores introduzidos nas tabelas 3 e 5, acima apresentadas, verificando-se que, não obstante não se ter apurado a maior precisão em termos absolutos, ainda assim manteve-se significativamente elevado, acima de 95% e 97% para dez e quinze *epochs*, com o menor tempo de execução observado em todos os modelos.

Tabela 10 – Aumento do valor de dropout

Teste	Epochs	Perda	Precisão	Δ (p)	Tempo Execução (s)	Δ (t)	Δ (p) / Δ (t)
7	5	55.78%	78.53%	-	679	-	
	10	26.95%	90.58%	15.34%	1325	95.14%	0.16
	15	98.00%	80.54%	-11.08%	1986	49.89%	-0.22

O último teste efetuado ao modelo *AlexNet* tem como base o aumento do valor de *dropout* nas primeiras duas camadas FC, de 0.5 para 0.7, registando uma tendência inversa aos restantes modelos, uma vez que, acima de dez *epochs*, verifica-se uma perda de precisão. Adicionalmente, este modelo apresenta elevada perda de dados e um tempo superior de execução face ao modelo padrão.

Tabela 11 – Quadro resumo AlexNet

Teste	Epochs	Perda	Precisão	Tempo Execução (s)
1	5	43.59%	81.83%	656
	10	16.45%	94.69%	1309
	15	11.35%	97.10%	1952
2	5	44.76%	82.81%	677
	10	14.61%	95.71%	1389

	15	12.94%	97.14%	2075
	5	43.87%	81.83%	410
3	10	20.31%	93.48%	812
	15	9.48%	97.90%	1222
	5	51.86%	75.31%	1565
4	10	40.54%	86.79%	3131
	15	19.66%	93.93%	4780
	5	33.28%	87.72%	364
5	10	16.75%	95.54%	727
	15	18.14%	95.54%	1105
	5	40.31%	83.84%	274
6	10	14.29%	95.94%	548
	15	10.32%	97.14%	813
	5	55.78%	78.53%	679
7	10	26.95%	90.58%	1325
	15	98.00%	80.54%	1986

O presente quadro representa um resumo de todos os testes efetuados ao modelo AlexNet.

DenseNet

DenseNet (Densely Connected Convolutional Network) é uma arquitetura de modelo de apredizagem profunda, introduzida por Huang et al. em 2017. É um tipo de rede neural convolucional (CNN) que aborda o problema do gradiente de desaparecimento e promove a reutilização de recursos com a introdução de conexões densas entre as camadas.

Numa rede CNN tradicional, cada camada recebe entrada apenas da camada anterior. No entanto, na DenseNet, cada camada é conectada a todas as outras de maneira feed-forward. Isso significa que a saída de cada camada é concatenada com as entradas de todas as camadas subsequentes. Esse padrão de conectividade densa permite que as informações fluam diretamente das camadas iniciais para as camadas posteriores, facilitando o fluxo de gradiente e promovendo a reutilização de recursos.

O bloco chave de um DenseNet é o bloco denso. Este consiste em várias camadas, cada uma das quais conectada de forma direta a todas as outras. Dentro de um bloco denso, a entrada para cada camada é a concatenação de mapas de recursos produzidos por todas as camadas anteriores. Isso resulta num padrão de conectividade denso, onde cada camada recebe os mapas de recursos de todas as camadas anteriores como entrada.

Para reduzir as dimensões espaciais, DenseNet usa camadas de transição entre blocos densos. As camadas de transição consistem numa camada de normalização, uma camada de convolução 1x1 e uma camada de agrupamento de média 2x2. A camada convolucional 1x1 é usada para reduzir o número de canais de recursos, comprimindo assim a informação antes de passá-la para o próximo bloco denso.

DenseNet tem várias vantagens sobre as arquiteturas CNN tradicionais. Primeiro, o padrão de conectividade densa incentiva a reutilização de recursos, o que reduz o número de parâmetros e melhora a eficiência geral do modelo. Em segundo lugar, alivia o problema de desaparecimento do gradiente, uma vez que o gradiente pode fluir diretamente das camadas posteriores para as camadas anteriores através das conexões densas. Por fim, o DenseNet demonstrou alcançar desempenho elevado em várias tarefas de visão computacional, incluindo classificação de imagens, deteção de objetos e segmentação.

Modelo Gerado

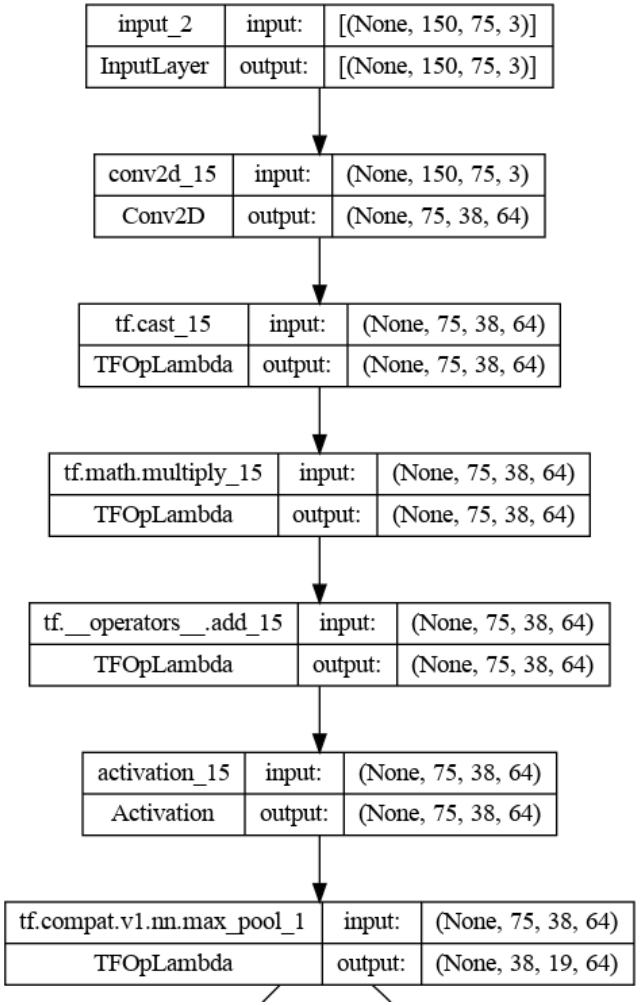

Figura 12 – DenseNet Modelo 1

A primeira parte do modelo gerado é composta pela camada de entrada que aceita o formato das imagens do dataset, (150,75,3).

A camada seguinte é uma convulsão, a qual realiza operações nos dados de entrada de forma a ser possível extrair atributos com recurso a kerneis e/ou filtros (Singh et al., 2020). O número de filtros determina o número dos canais de saída e o tamanho determina o receptive field (tamanho da parte dos dados de entrada que produz o atributo) (Adaloglou, 2020).

De seguida temos uma camada de normalização, a qual, é responsável pela normalização do output da camada anterior através da subtração a média e pela sua divisão pelo desvio padrão (Khandelwal, 2021).

A camada de ativação faz uso de uma função, neste caso ReLU (Rectified Linear Unit), e introduz no modelo a não linearidade que permite que este possa aprender padrões e representações complexas (Jain, 2021).

Esta camada é uma MaxPooling, que é responsável pela redução das dimensões dos dados de entrada de forma extrair mais atributos e reduzir o tempo de computação (DeepAI, 2020).

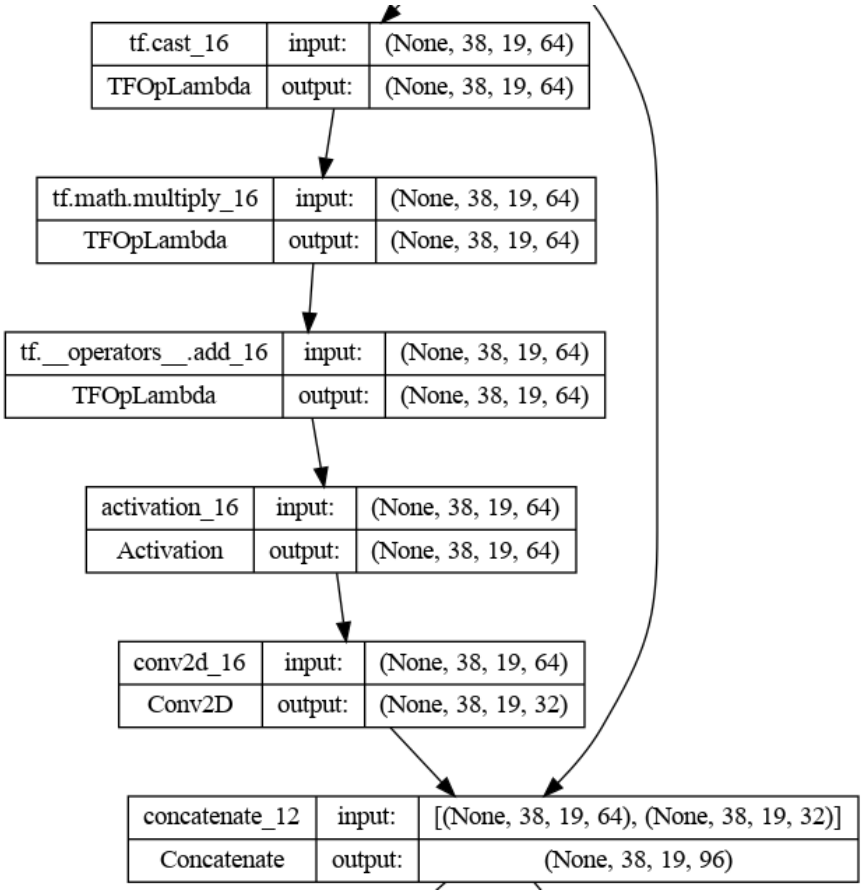

Figura 13 – DenseNet Modelo 2

De seguida, entramos nos blocos densos. Estes blocos são um dos mais importantes componentes do algoritmo. Cada uma das camadas dentro de um bloco denso recebe mapas de recursos de todas as camadas anteriores como entrada e produz mapas de recursos que são concatenados e passados para as camadas subsequentes.

Este bloco é composto por uma camada de normalização, uma camada de ativação com a função ReLU, uma camada de convulsão e por fim, uma camada de concatenação.

Para esta implementação, este bloco é repetido 3x5 vezes, intercalado com camadas de transição, as quais, são descritas em baixo. Ou seja, são três blocos de cinco camadas densas e uma de transição.

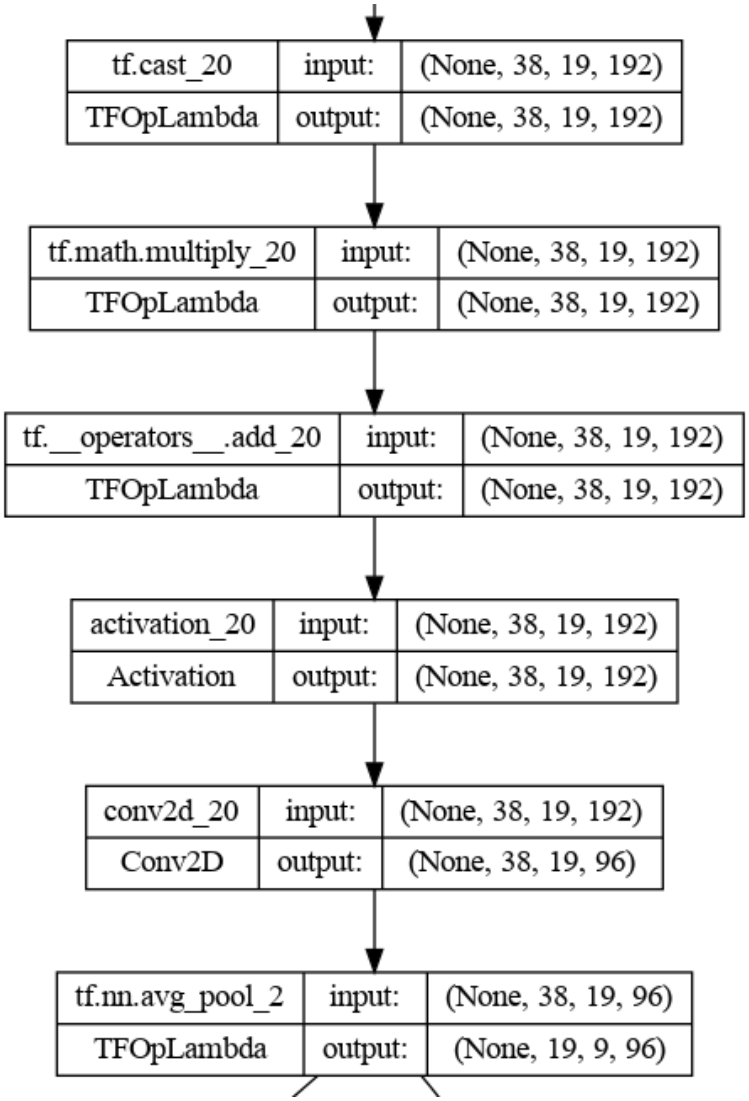

Figura 14 – DenseNet Modelo 3

Depois dos blocos densos, temos um bloco de transição. Este é inserido entre blocos densos para controlar as dimensões espaciais e reduzir o número de mapas de recursos. Uma camada de transição típica geralmente inclui uma camada de normalização em lote, uma camada convolucional 1x1 para compactar os mapas de recursos e uma camada de pooling (neste caso, o maxpooling) para reduzir o tamanho espacial.

O bloco aqui apresentado, é composto por uma camada de normalização, uma camada de ativação, com a função ReLU, uma camada de convulsão e por fim, uma camada de MaxPooling.

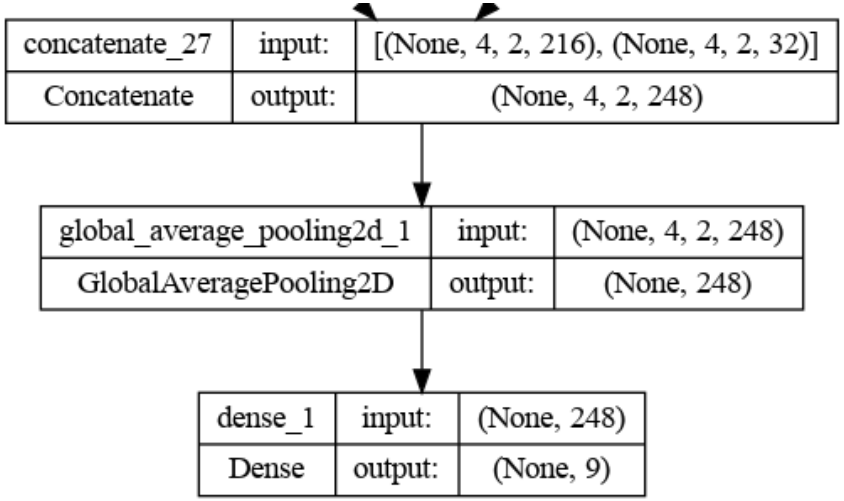

Figura 15 – DenseNet Modelo 4

Por fim, após a concatenação da última camada do último bloco denso, temos uma camada de Pooling, neste caso, AveragePooling e uma camada densa de saída, com a função de ativação a ser neste caso, softmax.

Este modelo, gerou um total de 534.985 parâmetros, todos eles disponíveis para treino.

Resultados

De forma a poder testar o modelo gerado, foram realizados vários testes. Os parâmetros que foram variados são os seguintes:

- num_classes – Este parâmetro determina quantos conjuntos de blocos densos e transição serão gerados,
- num_blocks – Este parâmetro determina quantos conjuntos de camadas densas terá um bloco denso,
- growth_rate – Este parâmetro determina o tamanho dos filtros das camadas de convulsão nos blocos densos,
- compression_factor – Este parâmetro determina o tamanho dos filtros das camadas de convulsão nos blocos de transição,
- batch_size - Este parâmetro determina o número de amostras que serão usadas em cada iteração,
- epochs - Este parâmetro determina o número de passagens para a frente e trás do algoritmo.

A primeira passagem serviu para criar uma base para a posterior avaliação das variações nos parâmetros listados.

Os parâmetros usados nesta primeira avaliação foram os seguintes:

- num_blocks = 3
- num_layers = 4
- growth_rate = 32
- compression_factor = 0.5
- batch_size = 32
- epochs = 10

Posteriormente, os dados usados para os vários testes foram os seguintes:

- num_blocks = 4, 5
- num_layers = 5, 6
- growth_rate = 8, 16
- compression_factor = 0.3, 0.7
- batch_size = 8, 16
- epochs = 5, 15

Tabela 12 – Resultados Testes DenseNet

teste	input_shape	num_classes	num_blocks	num_layers	growth_rate	compression_factor	epochs	batch_size	accuracy	loss	tempo
base	(150, 75, 3)	9	3	4	32	0,5	10	32	88,52%	0,3422	674,97
num_blocks_4	(150, 75, 3)	9	4	5	32	0,5	10	32	94,26%	0,2149	684,72
num_blocks_5	(150, 75, 3)	9	5	6	32	0,5	10	32	83,58%	0,4682	687,36
num_layers_5	(150, 75, 3)	9	3	5	32	0,5	10	32	89,50%	0,3041	901,05
num_layers_6	(150, 75, 3)	9	3	6	32	0,5	10	32	80,11%	0,4708	1202,59
growth_rate_8	(150, 75, 3)	9	3	4	8	0,5	10	32	78,24%	0,4899	157,31
growth_rate_16	(150, 75, 3)	9	3	4	16	0,5	10	32	77,57%	0,5149	298,05
compression_ factor_0.3	(150, 75, 3)	9	3	4	32	0,3	10	32	80,77%	0,4876	630,58
compression_ factor_0.7	(150, 75, 3)	9	3	4	32	0,7	10	32	82,73%	0,4602	712,26
batch_size_8	(150, 75, 3)	9	3	4	32	0,5	10	8	69,34%	0,7431	715,79
batch_size_16	(150, 75, 3)	9	3	4	32	0,5	10	16	79,66%	0,5168	685,39
epochs_5	(150, 75, 3)	9	3	4	32	0,5	5	32	65,78%	0,8286	338,09
epochs_15	(150, 75, 3)	9	3	4	32	0,5	15	32	92,12%	0,2566	992,90

Como podemos verificar na tabela em cima, o teste base teve uma precisão de 88,52%, com uma perda de 0,3422. Este modelo demorou 674 segundos a ser gerado.

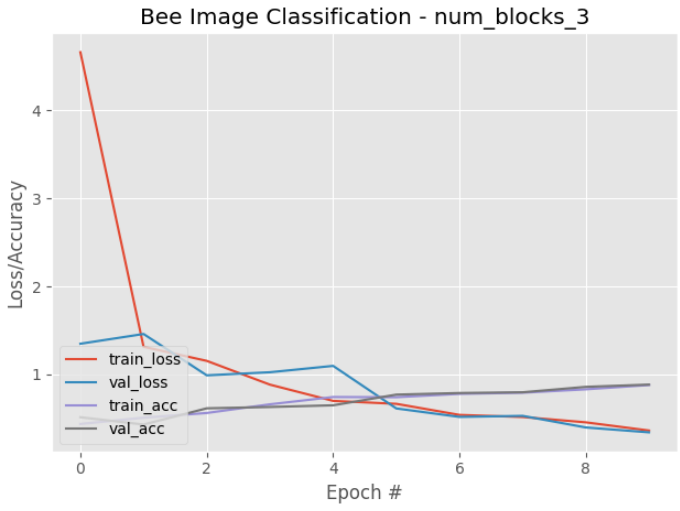

Figura 16 – Precisão e perda, teste base

Em termos de precisão, o resultado pior ocorreu quando foram usados apensas 5 epochs, sendo o único teste com uma precisão menor que 70%. Nesta métrica, o teste que ficou melhor classificado, teve um valor de 94,26%, com uma perda de 21,49% e demorou 684,72 segundos a ser executado.

Figura 17 – Melhor e pior precisão

Em termos de perda, o pior resultado aconteceu quando foram usados 5 epochs (Figura 17), com 0,6578, seguido do teste com um batch size de 8, com um valor de 0,7431. Já o melhor resultado ocorreu no teste com número de blocos 4, com um valor de 0,2149 (Figura 17).

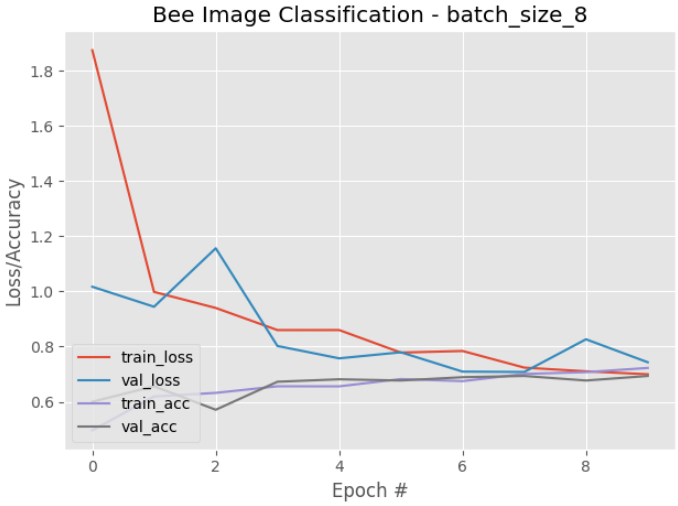

Figura 18 – Teste batch_size 8

Por fim, em termos de tempo demorado, o teste mais eficiente foi o realizado com parâmetro growth_rate igual a 8, com 157 segundos. Já o teste executado mais lentamente, foi o parâmetro num_layers igual a 6, com um tempo de 1202 segundos.

CONCLUSÕES

As redes neurais convolucionais (CNN) são amplamente usadas para tarefas de visão computacional, sendo que ao longo do tempo foram desenvolvidas várias arquiteturas para melhorar a suas performances. Neste trabalho foram estudadas três desses modelos LeNet, Alex Net e DenseNet e a influência dos seus vários parâmetros nos resultados.

O modelo LeNet, sendo uma das primeiras arquiteturas desenvolvidas, consiste numa estrutura simples com um número pequeno de camadas, tornando-o bastante eficiente para base de dados relativamente pequenas. Contudo, por este mesmo motivo poderá ter problemas a resolver padrões mais complexos e/ou base de dados maiores. A escolha da função de ativação neste modelo, como por exemplo 'relu' ou 'tanh' afeta a precisão dos resultados. O número de 'epochs' também influência bastante os resultados, contudo o número alto de epochs aumenta consideravelmente o tempo de treino sendo que o aumento da precisão pode não o justificar.

Por sua vez, o modelo AlexNet é composto por uma arquitetura com mais camadas, juntamente com a introdução da função de ativação 'relu', aumentou consideravelmente a sua habilidade de resolver problemas mais complexos. Os parâmetros como o 'batch size', 'kernel' e 'dropout' também não podem ser postos de parte uma vez que uma boa combinação destes parâmetros ajuda bastante na eficiência e precisão deste modelo.

Mais recentemente foi criado o modelo DenseNet, onde uma estrutura de blocos densamente conectados é a base do modelo. Esta estrutura faz com que cada camada receba informação de todas as camadas, promovendo a reutilização das características entre camadas. Devido à esta sua densa estrutura, este modelo consegue resultados com grandes níveis de precisão comparativamente aos modelos LeNet e AlexNet, mas em contrapartida usa muito poder computacional o que se pode traduzir em custos bastante mais elevados em ambientes onde os recursos são limitados.

A escolha dos parâmetros, como o número de 'epochs' e função de ativação, impacta diretamente a performance de cada modelo. O aumento de epochs que determina o número de vezes que o modelo interage com a basa de dados, pode originar melhores resultados e mais precisão, mas também aumenta o risco de overfitting. A escolha da função de ativação afeta a capacidade de treino do modelo, sendo que a função 'relu' é a mais usada devido à sua simplicidade e capacidade de resolver problemas de 'vanishing gradient'.

A complexidade e as camadas usadas em cada modelo influenciam diretamente o tempo de treino e o tempo de inferência, o modelo LeNet, sendo este o mais simples, requer muito menos

tempo de processamento. O modelo AlexNet, devido à sua estrutura com mais camadas requer mais tempo que LeNet e o modelo DenseNet, com a sua estrutura de conexões densa entre todos os blocos, é o que requer mais poder de processamento e consequentemente, mais tempo também.

BIBLIOGRAFIA

Adaloglou, N. (2020, July 2). Understanding the receptive field of deep convolutional networks | AI Summer. AI Summer. https://theaisummer.com/receptive-field/

bee_dataset. (n.d.). TensorFlow. https://www.tensorflow.org/datasets/catalog/bee_dataset

Convolutional Neural Network: Benefits, Types, and Applications. (2023, May 22). Datagen. https://datagen.tech/guides/computer-vision/cnn-convolutional-neural-network/

CS231n Convolutional Neural Networks for Visual Recognition. (n.d.). https://cs231n.github.io/convolutional-networks/#case-studies

Deep Convolutional Neural Networks (AlexNet) — Dive into Deep Learning 1.0.0-beta0 documentation. (n.d.). https://d2l.ai/chapter_convolutional-modern/alexnet.html

DeepAI. (2020). Max Pooling. DeepAI. https://deepai.org/machine-learning-glossary-and-terms/max-pooling

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. https://doi.org/10.1109/cvpr.2017.243

Jain, V. (2021, December 13). Everything you need to know about "Activation Functions" in Deep learning models. Medium. https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253

Nayak, S., & Nayak, S. (2021). Understanding AlexNet | LearnOpenCV #. LearnOpenCV – Learn OpenCV, PyTorch, Keras, Tensorflow With Examples and Tutorials. https://learnopencv.com/understanding-alexnet/

Khandelwal, S. (2021, December 16). Normalization, its types and Normalization layers - MUACM - Medium. Medium. https://medium.com/muacm/normalization-its-types-and-normalization-layers-c9f1bb40b2dd

Krizhevsky et al. "ImageNet classification with deep convolutional neural networks." Advances in Neural Information Processing Systems, 2012.

Kumar, A. (2023, May 12). Different Types of CNN Architectures Explained: Examples - Data Analytics. Data Analytics. https://vitalflux.com/different-types-of-cnn-architectures-explained-examples/

Patel, S. (2020). A Comprehensive Analysis of Convolutional Neural Network Models. International Journal of Advanced Science and Technology, 29(4), 771–777. https://www.researchgate.net/profile/Sanskruti-Patel-2/publication/344248968_A_Comprehensive_Analysis_of_Convolutional_Neural_Network_Models/links/5f6086ca299bf1d43c04fac5/A-Comprehensive-Analysis-of-Convolutional-Neural-Network-Models.pdf

Singh, S. A., Meitei, T. G., & Majumder, S. (2020). Short PCG classification based on deep learning. In Elsevier eBooks. https://doi.org/10.1016/b978-0-12-819061-6.00006-9

Sharma, S. (2022, November 20). Activation Functions in Neural Networks - Towards Data Science. Medium. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. https://doi.org/10.1109/cvpr.2017.243

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11), 1998