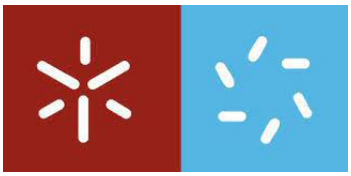

Universidade do Minho
Matemática e Computação

Classificadores e Sistemas Conexionistas

2021/2022

Realizado por:

Luís Araújo	PG46753
Ricardo Teixeira	PG45271

26 de maio de 2022

Resumo

As redes neuronais têm chamado à atenção pelos resultados brilhantes que têm mostrado nas mais diversas áreas, contudo, dado um problema, construir um modelo que se adeque da melhor forma possível ao mesmo e mostre resultados satisfatórios, nem sempre é uma tarefa trivial, considerando que existem certos tipos de restrições que limitam a forma como o possamos desenhar. Muitas vezes, o tempo de treino ou a velocidade de previsão desejada criam a necessidade de restrições na modelação da arquitetura de uma rede, que talvez não a tornem na que consegue os melhores resultados, mas as caraterísticas do problema assim o exigem. Neste trabalho, treinamos, avaliamos e comparamos vários modelos e diferentes tipos de parametrização para 4 problemas de Visão por Computadores distintos: dois problemas de classificação em imagens (*Fashion-MNIST* e *CIFAR100*), um problema de geração de descrição para imagens de posições de xadrez (*Chess-Positions*) e um problema de classificação em vídeos (*UCF101*). Para o processo de treino utilizamos *Cross-validation* em todos os modelos testados.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Modelos Utilizados

1.1 Multilayer Perceptron - *MLP*

A arquitetura geral das redes de tipo *MLP* usadas neste trabalho é composta por uma camada de *input*, *N Hidden Layers*, todas com o mesmo número de neurónios e função de ativação, e por fim, uma camada de *output* (Fig. 1.1).

1.2 Convolutional Neural Network - *CNN*

Para a arquitetura das redes de tipo *CNN*, temos *N* "blocos"convolucionais (compostos por duas camadas *Conv2D* ou *Conv3D* com o mesmo número de *kernels* de mesma dimensão e uma camada MaxPooling) que são encadeados a uma camada *Flatten* e uma (ou duas dependendo do *dataset*) camada *Dense* para classificação, seguida da camada de *output* (Fig. 1.2).

1.3 *CNN-GRU* Hybrid Model

Este modelo foi concebido com o último *dataset* em mente, para tratar as dependências temporais entre os *frames* de um vídeo. Sendo *NF* o número de *frames* de cada vídeo, o modelo é composto por *NF* ramos de redes *CNN* mencionadas em 1.2 (criados usando o *wrapper TimeDistributed*) agrupados numa camada *GRU*, seguida por fim de uma rede *MLP* (semelhante ao modelo *CNN*) para efetuar a classificação final (Fig. 1.3).

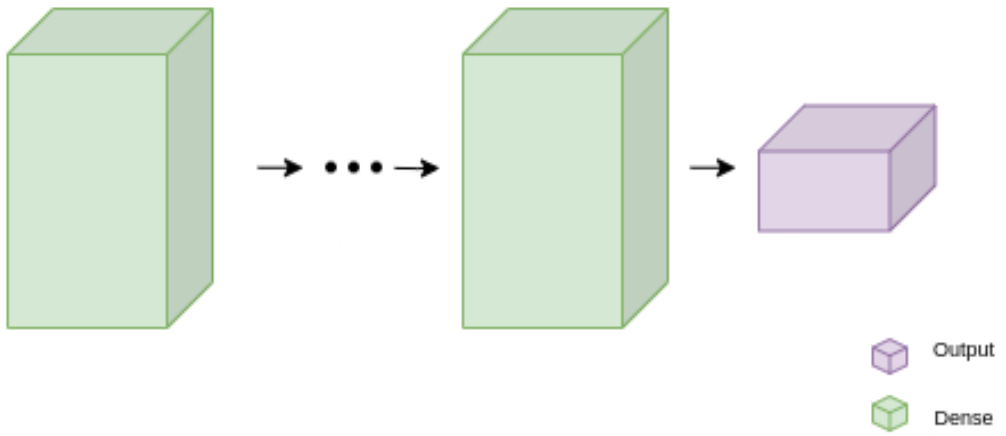

Figura 1.1: Modelo geral de uma rede *MLP*

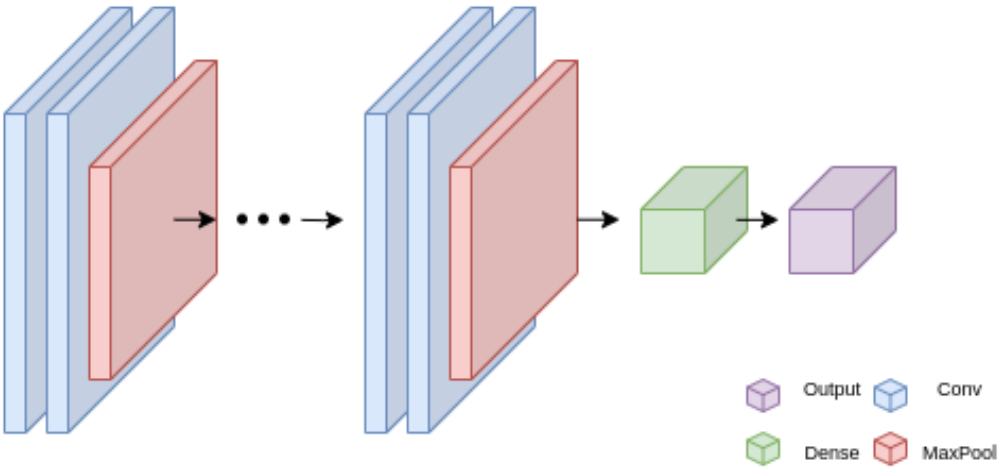

Figura 1.2: Modelo geral de uma rede *CNN*

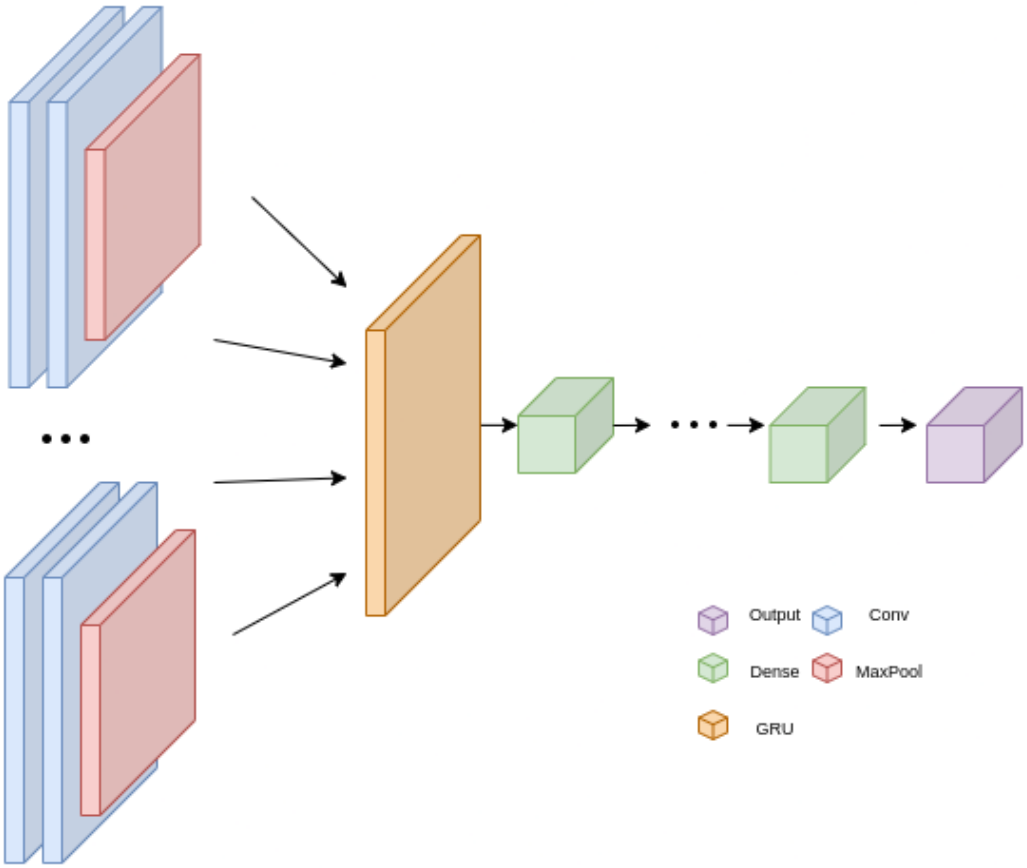

Figura 1.3: Modelo geral da rede híbrida *CNN-GRU*

Capítulo 2

Fashion-MNIST

2.1 Descrição do *dataset* e objetivo dos modelos

O *Fashion-MNIST* é um *dataset* com sessenta mil (60000) imagens de treino e dez mil (10000) imagens de teste, todas em *grayscale* e 28×28 *pixels*, cada uma identificada por uma de 10 *labels* (*T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt ,Sneaker, Bag, Ankle boot*) correspondente ao objeto representado. O objetivo dos modelos para este *dataset* é a classificação das imagens de acordo com as *labels* associadas.

2.2 Pré-processamento e Loading dos dados

Sendo o *MNIST* um *dataset* relativamente leve, os dados foram todos carregados em memória usando a função *tf.keras.datasets.fashion_mnist.load_data()*. Quanto ao pré-processamento dos dados, apenas foi necessário fazer a normalização dos pixels e, para o caso do modelo *MLP*, a vetorização das imagens (Fig. 2.1), o que não foi necessário para o modelo *CNN* visto que este consegue trabalhar com imagens nativamente.

2.3 Técnicas de *Benchmark*

Para o modelo *MLP*, devido à facilidade de implementação de um modelo parametrizado, foi usada a técnica de *random search* manual, iterando por listas de possíveis parâmetros e testando todas as combinações. Os hyper-parâmetros otimizados foram:

- função de ativação - [*ReLU*,*tanh*]
- *batch size* - [16,32,64,128,256]
- número de neurónios - [16,32,64,128,256]

```python
def prepare_data_mlp():
    (x_train, y_train), (x_test, y_test), classes = load_data()
    # normalizing/scaling pixel values to [0, 1]
    x_train = x_train.astype('float32')/255.
    x_test = x_test.astype('float32')/255.
    x_train = x_train.reshape(
        x_train.shape[0], x_train.shape[1]*x_train.shape[2])
    x_test = x_test.reshape(x_test.shape[0], x_test.shape[1]*x_test.shape[2])
    return x_train, y_train, x_test, y_test, classes
```

Figura 2.1: Modelo geral de uma rede *CNN*

- número de *hidden layers* - [2,3,4,5,6]

No que toca ao modelo *CNN*, o *tunning* foi realizado manualmente, usando a intuição como guia. Os hyper-parâmetros optimizados foram:

- função de ativação
- *batch size*
- número de *kernels*
- número de blocos

O número de épocas é determinado em ambos os modelos pela análise das curvas de aprendizagem. Um bom método heurístico para aproximar o valor ótimo do número de épocas é verificar o ponto de interseção das curvas (Fig. 2.2).

2.4 Análise e comparação de resultados

Métricas	Batch Size	Epochs	Hidden Layers	Neurons	K-folds	Activations	Test Accuracy
MLP1	128	2	3	256	5	ReLU	88.1%
MLP2	128	2	2	128	5	ReLU	88.0%
MLP3	128	2	4	256	5	ReLU	87.9%
MLP4	128	2	5	128	5	ReLU	87.7%
MLP5	128	2	5	256	5	ReLU	87.7%

Tabela 2.1: Comparação de resultados das *MLP* para o Fashion_MNIST.

Analisando os valores das tabelas 2.2 e 2.1 podemos concluir que ambos os modelos produzem resultados satisfatórios em termos de *accuracy* para este *dataset* mais simples, tendo o modelo *CNN* atingido valores ligeiramente superiores

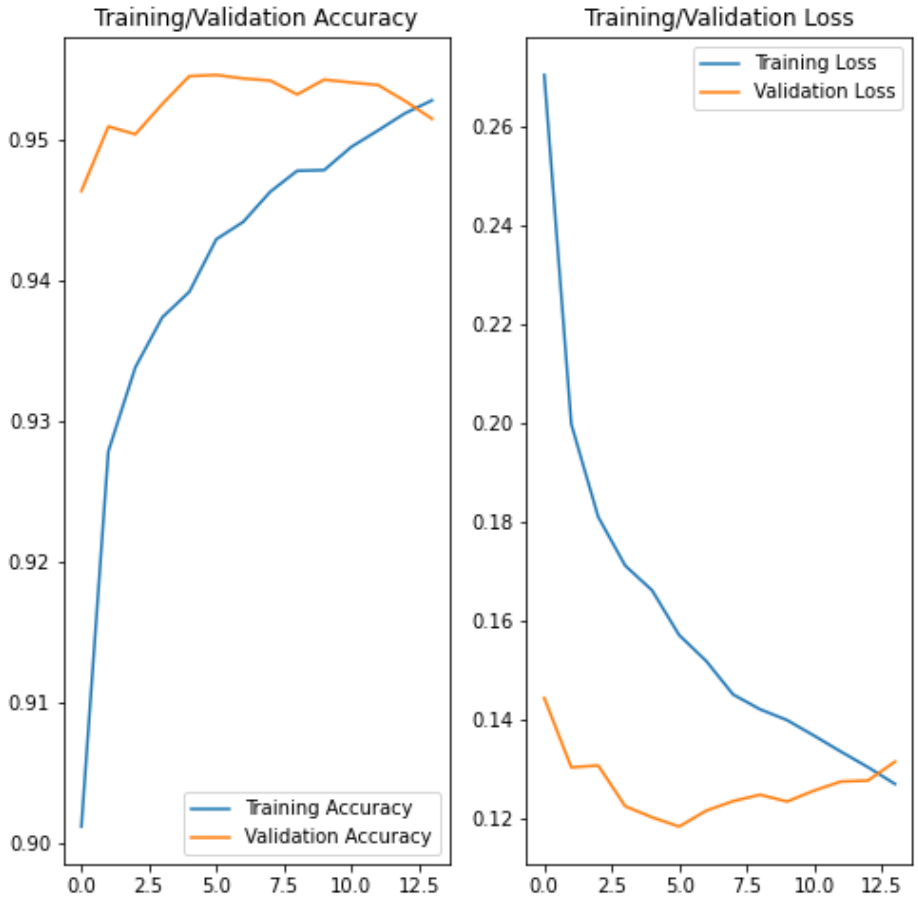

Figura 2.2: O número ótimo de épocas é entre 12 e 13

Métricas	Batch Size	Epochs	Hidden Blocks	Kernels	K-folds	Activations	Test Accuracy
CNN1	128	14	2	(64,128)	5	ReLU	93.7%
CNN2	64	14	3	(64,128,256)	5	ReLU	93.4%
CNN3	128	14	3	(32,64,128)	5	ReLU	93.3%

Tabela 2.2: Comparação de resultados das *CNN* para o Fashion_MNIST.

(entre 5% a 6% de diferença). No entanto é de notar que o aumento de performance não é alto o suficiente para justificar o aumento exponencial do modelo *CNN* em termos de poder de computação necessário e tempo gasto para o treino (o modelo *MLP* demora cerca de um minuto a treinar, dependendo do número de *folds* usados na *cross-validation*, enquanto que o modelo *CNN* demora à volta de minutos).

Capítulo 3

CIFAR100

3.1 Descrição do *dataset* e objetivo dos modelos

O *dataset CIFAR100* é constituído por 100 classes de imagens bastante heterógeneas e 20 superclasses também de naturezas bastantes distintas. As imagens são *RGB* e de dimensão 28×28 pixels, sendo que o *dataset* de treino tem cinquenta mil (50000) exemplares e o de teste dez mil (10000). Os modelos que desenhamos têm como objetivo prever as superclasses de cada imagem. Escolhemos este *dataset* devido ao aumento de complexidade em relação ao *Fashion_MNIST*, pelo facto de ter imagens *RGB* em vez de imagens *GreyScale*.

Superclass	**Classes**
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Figura 3.1: Classes do CIFAR100

3.2 Pré-processamento e Loading dos dados

Embora mais pesado que o *dataset Fashion_ MNIST*, o *CIFAR100* é leve o suficiente para permitir o *loading* do *dataset* completo para a memória. Quanto ao pré-processamento dos dados, para o modelo *MLP*, semelhantemente ao *Fashion_ MNIST*, fizemos a normalização dos pixels e vetorização das imagens, com a pequena diferença de que neste caso as imagens são *RGB*, tornando necessário o colapso de uma terceira dimensão do *input*. No caso do modelo *CNN*, apenas foi necessária a normalização dos pixels, visto que este modelo aceita tanto imagens em *GreyScale* como em *RGB*.

3.3 Técnicas de *Benchmark*

Embora mais complexo, este *dataset* é simples o suficiente em termos de complexidade, o que permitiu o uso das mesmas técnicas de *benchmarking* usadas no *dataset Fashion_ MNIST*.

3.4 Análise e comparação de resultados

Métricas	Batch Size	Epochs	Hidden Layers	Neurons	K-folds	Activations	Test Accuracy
MLP1	128	2	3	128	5	ReLU	35.2%
MLP2	64	2	3	256	5	ReLU	33.5%
MLP3	128	2	3	256	5	ReLU	33.1%
MLP4	64	2	4	256	5	ReLU	33%
MLP5	128	2	4	256	5	ReLU	32.9%

Tabela 3.1: Comparação de resultados das *MLP* para o CIFAR100.

Métricas	Batch Size	Epochs	Hidden Blocks	Kernels	K-folds	Activations	Test Accuracy
CNN1	128	20	3	(32,64,128)	5	ReLU	58.2%
CNN2	128	20	3	(64,128,256)	5	ReLU	55.3%
CNN3	64	20	4	(16,32,64,128)	5	ReLU	47.4%

Tabela 3.2: Comparação de resultados das *CNN* para o CIFAR100.

Olhando para os resultados das tabelas 3.1 e 3.2, podemos concluir que a utilidade de modelos *MLP* para resolução de problemas de visão por computador começa a diminuir em função do aumento da complexidade do problema, sendo

que os modelos *CNN* produzem resultados muito mais satisfatórios (como era de esperar visto ser uma arquitetura pensada especificamente para este tipo de problemas). Neste *dataset* mantêm-se a diferença entre os tempos de treino (valores semelhantes ao *dataset Fashion_MNIST*) pelo que desta vez são justificados.

Capítulo 4

Chess-Positions Dataset

4.1 Descrição do *dataset* e objetivo dos modelos

O *dataset* *Chess-Positions* foi obtido no Kaggle e é composto por cem mil (100000) imagens *RGB* (400×400 pixels) de posições de xadrez geradas aleatoriamente. Os modelos que criamos têm o objetivo de determinar qual o **FEN-Forsyth-Edwards Notation** associado a uma determinada imagem. A notação **FEN** descreve o estado do tabuleiro utilizando apenas uma linha de texto, da seguinte forma:

- cada hífen separa a representação de duas linhas adjacentes do tabuleiro.
- as letras 'P','R','N','B','Q' e 'K' representam respetivamente o Peão, Torre, Cavalo, Bispo, Dama e Rei. Maiúsculas representam peças Brancas e minúsculas peças pretas.
- Os números representam o número de casas vazias consecutivas.
- A notação é estabelecida pela ordem natural de cima para baixo, começando na casa a8 e terminando na casa h1.

Esta tarefa de determinação da notação é, de certa forma, semelhante ao problema de geração de legendas descritivas de uma imagem, contudo, a complexidade é menor, visto que o problema pode ser genericamente decomposto em 64 problemas individuais de previsão do conteúdo de uma certa casa do tabuleiro.

Figura 4.1: Exemplos da notação FEN (chess.com)

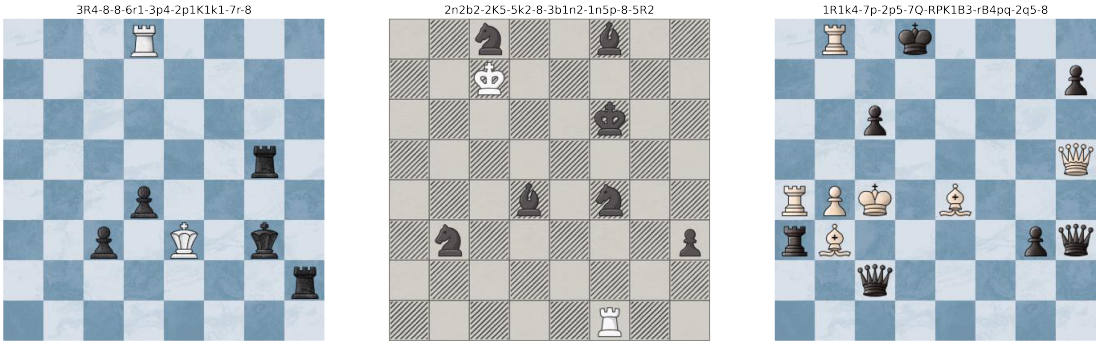

Figura 4.2: Exemplos de figuras do Chess-Positions

4.2 Pré-processamento e Loading dos dados

Uma vez que este *dataset* é substancialmente mais pesado que os anteriores, verificamos rapidamente que seria impossível processar os dados imagem a imagem e simplesmente dar *load* para a memória. Para contornar esta situação, a nossa função de pré-processamento apenas recebe como input uma imagem original (400,400,3) e redimensiona para uma imagem mais pequena (200,200,3). Mesmo com esta redução de dimensão, era computacionalmente inviável pré-processar as imagens todas de uma vez, assim sendo, a estratégia que utilizamos foi recorrer a *Generators*. Aquando do *fit* e *predict* do modelo, em vez de passarmos como argumento um *array* com os dados de input processados, ambos os métodos recebem como argumento uma função *generator*, que se comporta como um *iterator*, ou seja, pode ser consumida parcialmente, o que é particularmente útil porque nos

permite processar uma imagem apenas quando é selecionada para um certo *batch* de treino ou no momento da *prediction* caso pertença aos dados de teste. Esta técnica aumentou a duração do treino da rede, uma vez que o pré-processamento se realiza nesta fase.

```
def pred_gen_mlp(features, batch_size):
    for i, img in enumerate(features):
        res = process_image(img)
        yield res.reshape((64,np.prod(res.shape[1:])))
```

Figura 4.3: Exemplo da função *generator* para os dados de teste (MLP)

4.3 One-Hot Encoding

Para realizar *One-Hot Encoding* das *labels*, dada a complexidade das mesmas e o universo de possibilidades, necessitamos definir uma função que transforma as *labels* (strings) em tensores de dimensão (64,13). Na realidade, queremos que esta função seja bijetiva, de forma a que cada *label* esteja associada a um e um só elemento do contradomínio.

Definição 4.3.1 (Conjunto $\mathbb{E}_{13}$)**.** Seja M a matriz identidade de dimensão 13×13, $\mathbb{E}_{13}$ é o conjunto cujos elementos são as colunas de M, ou seja $\mathbb{E}_{13} = \{e1, e2, ..., e13\}$.

Definição 4.3.2 (λ_{OHE} - função de encoding auxiliar para peças)**.** Seja FEN o espaço dos caracteres permitidos numa label, define-se a função:

$$\lambda_{OHE} : FEN \longrightarrow (\mathbb{E}_{13})$$
$$\lambda_{OHE}(P) = e1$$
$$\lambda_{OHE}(N) = e2$$
$$\lambda_{OHE}(B) = e3$$
$$\lambda_{OHE}(R) = e4$$
$$\lambda_{OHE}(Q) = e5$$
$$\lambda_{OHE}(K) = e6$$
$$\lambda_{OHE}(p) = e7$$
$$\lambda_{OHE}(n) = e8$$
$$\lambda_{OHE}(b) = e9$$
$$\lambda_{OHE}(r) = e10$$
$$\lambda_{OHE}(q) = e11$$
$$\lambda_{OHE}(k) = e12$$
$$\forall n \in \{1, 2, ..., 8\} \quad \lambda_{OHE}(n) = e13$$

Para que o One hot encoding seja possível é necessário que a função implementada seja um ismorfismo entre o espaço das labels e o contradomínio definido que, no nosso caso, é $(\mathbb{E}_{13})^{64}$, ou seja um vetor de $\mathbb{E}_{13}$ por cada casa do tabuleiro. Para este efeito definimos então a função *one_hot_encode()* e a sua inversa, também isomorfismo, *one_hot_decode()*. A função *one_hot_encode()* define-se aplicando sucessivamente λ_{OHE} a uma string em notação **FEN**.

4.4 Técnicas de *Benchmark*

Dado o salto de complexidade do *dataset* anterior para este, e considerando que houve uma aumento considerável do tempo de treino, o *tunning* dos modelos para este problema passou necessariamente a ter uma componente bastante mais intuitiva para a escolha dos melhores parâmetros.

4.4.1 *CNN's*

No caso das redes CNN, uma vez que o primeiro modelo que testamos conseguiu resultados muito satisfatórios, o processo de *tunning* passou por tentar minimizar o tempo de treino mantendo a mesma capacidade de previsão. Verificamos que poderíamos aumentar o tamanho dos filtros e diminuir o número de layers e neurónios.

4.4.2 *MLP's*

Aquando do treino das MLP's, percebemos que a inclusão de mais camada e de mais neurónios não conseguia melhorar o desempenho do modelo. Tentamos então manter um modelo não muito profundo fazendo variar o número apenas entre 16,32 e 64.

4.5 Análise e comparação de Resultados

Neste *dataset* , pudemos perceber rapidamente que as *CNN* se comportavam bastante melhor do que as *MLP*. Durante o treino com *cross-validation*, as MLP demonstraram dificuldades em estabilizar o valor de *loss*, o que revela que a convergência da *loss function*, neste caso, ficou comprometido, algo que não aconteceu no melhor modelo *CNN*, como podemos ver pelas figuras 4.4 e 4.5.

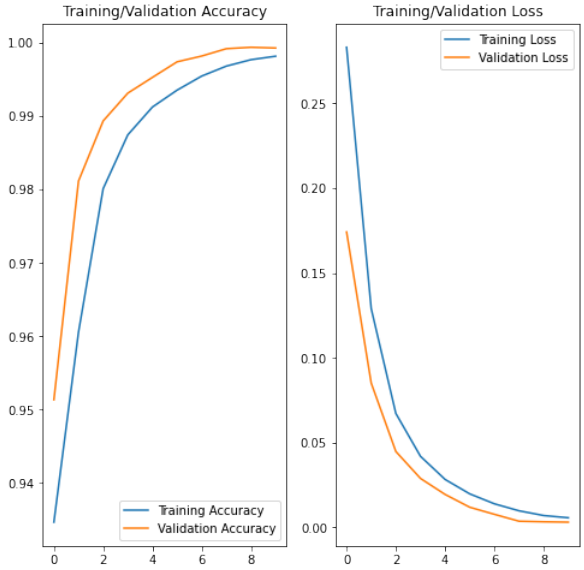

Figura 4.4: Gráficos de treino dos 2 melhores modelos CNN (CHESS)

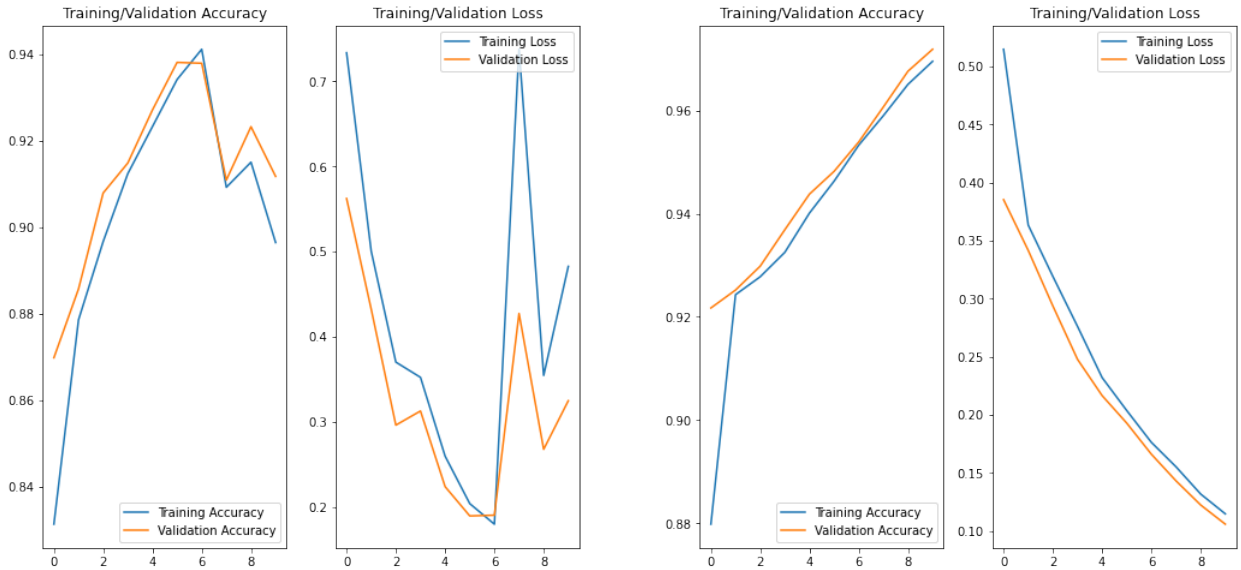

Figura 4.5: Gráficos de treino dos 2 melhores modelos MLP (CHESS)

Métricas	CNN1	CNN2	MLP1	MLP2
Batch Size	64	64	64	64
Epochs	8	8	10	10
Hidden Layers	6	4	5	3
Conv Layers	4	3	-	-
K-folds	3	3	3	3
Activations	relu	relu	relu	relu
Neurónios	128	256	128	(32,64,64)
Train time	30min	25min	30min	30min
Test Accuracy	100%	100%	27%	74.5

Tabela 4.1: Melhores modelos por arquitetura (CHESS-POSITIONS)

Capítulo 5

UCF101

5.1 Descrição do *dataset* e objetivo dos modelos

O *dataset* **UCF101** é um conjunto de vídeos recolhidos do *youtube*, categorizados por 101 subclasses associadas a ações humanas e 20 superclasses mais abstratas. Os modelos que iremos comparar receberão como input a totalidade deste dataset e deverão prever a que superclasses pertencem cada um dos vídeos de input. Com a escolha deste *dataset* , tínhamos como objetivo a análise da performance dos modelos em análise de vídeos *versus* análise de imagens (*datasets* anteriores). Começamos por tentar prever 10 classes mas rapidamente percebemos que seria computacionalmente muito pesado, e rapidamente passamos para a tarefa de previsão de 5 e, por fim, de 3 classes.

5.2 Pré-processamento e Loading dos dados

Uma vez que este *dataset* tinha cerca de 7GB de dados, utilizamos o google colab e a *API* do Kaggle para facilitar o download e armazenar os dados no google drive. Visto que ambos os modelos conseguiam trabalhar sequências de imagens (uso de camadas *Conv3D* no modelo *CNN* e o uso de vários ramos no modelo *CNN-GRU*), o pré-processamento de dados passou apenas pela seleção e *downscalling* de um número fixo de *frames* em cada vídeo (Fig. 5.1).

5.3 Técnicas de *Benchmark*

Neste *dataset* a o *tunning* de parâmetros intra-arquitetura foi o menos exaustivo de todos os *datasets* por uma questão de custos computacionais. Optamos por tentar ver como se comportavam dois modelos bastantes distintos para o problema de classificação dos vídeos.

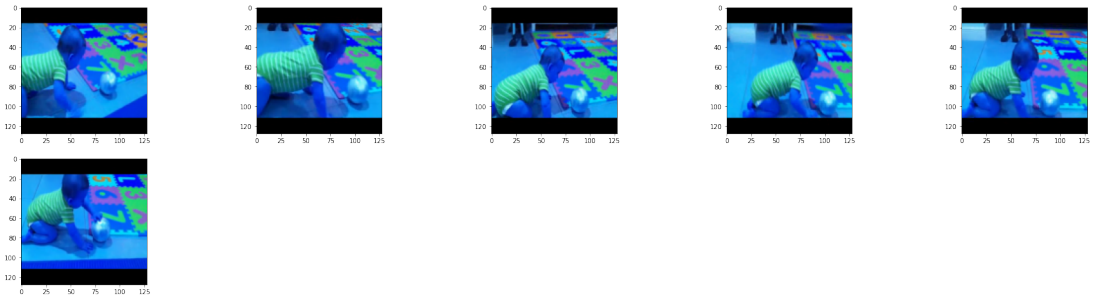

Figura 5.1: Exemplo de seleção de 6 frames

5.4 Análise e comparação de Resultados

Métricas	CNN	CNN-GRU
Batch Size	8	8
Epochs	4	2
Hidden Layers	AL	ALB
K-folds	3	3
Activations	ReLU	ReLU
Train time	4h	3h
Test Accuracy	46.6%	62.5%

Tabela 5.1: Melhores modelos por arquitetura UCF101

Pela tabela 5.1 podemos afirmar que, para condições de estudo semelhantes, o modelo *CNN-GRU* é mais rápido tem melhor performance, embora estes resultados sejam inconclusivos devido à falta de poder computacional para a realização de um *tunning* exaustivo dos modelos.

Capítulo 6

Conclusão

Após lidarmos com problemas de várias complexidades e de índoles bastante distintas, percebemos que os tipos de modelos de Redes Neuronais que utilizamos, se adequam de formas bastante particulares a certos tipos de tarefas, ou, por outras palavras, as diferentes arquiteturas, apesar de conseguirem efetuar previsões para muitos tipos diferentes de *inputs*, servem melhor certos tipos de propósitos. Os modelos de CNN's (ou Modelos híbridos com camadas convolucionais) mostram-se claramente soluções mais apelativas para a resolução de problemas de *Computer Vision*, algo que nos pareceu mais evidente à medida que a complexidade dos *datasets* aumentava.

Ao longo da realização deste trabalho, vimos que certos tipos de otimização de hiperparâmetros, como o *random search* ou algum tipo de pesquisa *exaustiva*, dado um *dataset* complexo ,tornam-se quase impraticáveis para arquiteturas de *Deep Learning* que possuam várias camadas convolucionais, uma vez que o tempo de computação exigido aumenta muito consideravelmente.