Sistema de Deteção e Classificação de Máscaras Faciais

Bernardo Saraiva[pg50259], José Gonçalves[pg50519], and Rui Moreira[pg50736]

Grupo 3
Universidade do Minho, Departamento de Informática

Resumo O presente relatório aborda os detalhes e resultados do projeto desenvolvido no âmbito da unidade curricular Aprendizagem Profunda, que envolve um sistema de deteção e classificação de máscaras faciais. O objetivo principal do projeto passa por criar um modelo de *machine learning* capaz de identificar corretamente se uma pessoa está a utilizar uma máscara facial e se está a utilizá-la adequadamente, como medida de proteção contra a disseminação do COVID-19.

Keywords: Deep Learning · Redes Neuronais · Processamento de imagens · Covid-19 · Deteção de Objetos · SSD · YOLO · Transfer learning

1 Introdução

No inicio de 2019, teve início a pandemia COVID-19, que apresentou enormes desafios para a saúde a nível global. Desde então, medidas preventivas, como o uso de máscaras faciais, foram fortemente recomendadas pelas autoridades de saúde para reduzir a propagação do vírus até aos dias de hoje. No entanto, a verificação manual do uso correto de máscaras em locais públicos, com muita afluência de pessoas, é um processo demorado e que implica a presença de alguma autoridade a vigiar.

Nesse contexto, o presente projeto procura desenvolver um sistema automático de deteção e classificação de máscaras faciais, utilizando técnicas de *Machine Learning* e Aprendizagem Profunda. Este sistema poderia ser facilmente adotado por estabelecimentos comerciais, instituições públicas de saúde e locais com muita afluência, a fim de reforçar a importância do uso adequado de máscaras faciais como medida de proteção/prevenção contra o COVID-19 ou até outras doenças.

O sistema desenvolvido foi realizado por meio da divisão em duas fases, uma para deteção de faces (boundings boxes) e classificação de máscaras faciais (máscaras corretamente utilizadas, máscaras incorretamente utilizadas e sem máscara).

Na fase de deteção de faces em imagens, a utilização de técnicas de *Transfer Learning* surge como uma abordagem promissora, sendo que esta técnica consiste em aproveitar modelos pré-treinados em grandes conjuntos de dados para tarefas similares ou relacionadas. Esta abordagem permite obter benefícios como a economia de recursos computacionais e a aceleração do processo de treino.

No decorrer do presente relatório, serão apresentados todos os detalhes do software desenvolvido, incluindo a metodologia utilizada, os conjuntos de imagens utilizados, a arquitetura do modelo desenvolvido, bem como os resultados obtidos. Serão discutidos também os desafios enfrentados durante o projeto, bem como possíveis melhorias e evoluções do sistema.

2 Metodologia e descrição do software desenvolvido

A metodologia utilizada na análise de dados e desenvolvimento de software é o *CRISP-DM* (*Cross-Industry Standard Process for Data Mining*), que é um processo bastante frequente em tarefas deste género. O *CRISP-DM* consiste em seis fases: compreensão do problema, compreensão dos dados, preparação dos dados, modelação, avaliação e implementação. O uso desta metodologia permite uma abordagem sistemática e estruturada para a resolução do problema em questão.

Relativamente à estrutura do projeto, todo o software foi desenvolvido com recurso à biblioteca *PyTorch* e pode ser encontrado no repositório do github[1].

[1] https://github.com/DevSaraiva/AP-Aprendizagem-Profunda

Ao longo do projeto foi criado o ficheiro *face-mask-detection-pytorch.ipynb* que conta com todo o código relacionado com o processo de leitura e tratamento de dados, assim como definição, treino e teste do modelo desenvolvido pelo grupo e utilização das técnicas de deteção de objetos, sendo tudo isto referido e explicado com maior detalhe nas secções que se seguem.

Por forma a obter conjuntos de dados específicos e com um propósito definido, foram criados alguns ficheiros auxiliares. São utilizadas três pastas:

- *FaceDataMaker* - para criar um *dataset* com imagens da classe *'face_without_mask'*
- *MaskIncorrectDataMaker* - para criar outro *dataset* com imagens da classe *'face_with_mask_incorrect'*
- *testSetMaker* - para criar um dataset com as imagens do conjunto de dados de teste criado para testar com outro modelo (isto foi necessário para obter as imagens de teste sem pré-tratamento aplicado pelo grupo, e assim aplicar apenas o tratamento necesssário a esse modelo).

As duas primeiras servirão para balancear os dados, processo que será explicado em detalhe posteriormente. Cada uma das pastas contém todas as imagens usadas para criar o *dataset* e um notebook *makingDataset.ipynb*, onde é utilizado, por vezes, um algoritmo de deteção para gerar as *bounding boxes* e, por fim, gerado o ficheiro *".csv"* que contém essas informações juntamente com a *label* e nome da imagem correspondente.

3 Descrição e exploração de dados

Com vista a estabelecer um ponto de partida que permitisse obter conhecimento e material suficiente para desenvolver o presente projeto prático, começou-se por efetuar uma pesquisa que permitisse obter recursos suficientes para o desenvolvimento do mesmo. Deste modo, foi encontrado um conjunto de dados no Kaggle [2] que contém imagens de pessoas com variações de acessórios faciais.

3.1 Descrição do Dataset

Ao todo, este *dataset* é composto por 6024 imagens de diferentes pessoas em diferentes cenários e ângulos, a fim de representar uma variedade de situações reais em que a deteção e classificação de máscaras faciais seriam aplicadas. É de notar que este já contém uma divisão em *train* e *test* (denominado *submission*), em que o primeiro contém 4326 imagens com anotações e o segundo contém 1698 imagens, mas estas não contêm anotações. No entanto, o conjunto de dados *submission* não se encontra balanceado, pelo que não se usou este conjunto de imagens de forma a avaliar o nosso modelo.

Como referido acima, o conjunto de dados de treino é composto por um conjunto de ficheiros com anotações para cada imagem (no formato *.json*), estando descrito o nome da imagem e um conjunto de anotações para cada objeto detetado, tendo sido utilizadas apenas duas, que correspondem, entre outras, as seguintes informações:

- **BoundingBox:** As coordenadas da *bounding box* que envolve a região anotada da imagem. É uma lista de 4 itens, representando as coordenadas x e y do ponto inicial da caixa (canto superior esquerdo) e as coordenadas x e y do ponto final da caixa (canto inferior direito).
- **classname:** A classe ou categoria da anotação, havendo 20 possibilidades diferentes e que serão explicadas em seguida.

Como referido anteriormente, cada objeto detetado é classificado com uma das classes de um conjunto bem definido, uma vez que são representados no *dataset* vários casos relativos a diversos acessórios que possam ser utilizados na cabeça (óculos, chapéus, entre outros).

Uma vez que o modelo que se pretende desenvolver apenas pretende trabalhar com máscaras, optou-se por reduzir a apenas 3 classes: **face_with_mask**, **face_no_mask** e **face_with_mask_incorrect**. Deste modo, todas as outras *labels* foram descartadas.

[2] https://www.kaggle.com/datasets/wobotintelligence/face-mask-detection-dataset

3.2 Exploração e Tratamento dos Dados

Leitura das anotações e imagens

Uma vez que cada imagem dos dados de treino contém a informação das *bounding boxes* de cada face e respetiva classe, começa-se pela leitura e extração da informação presente nos ficheiros de anotações. Assim, como é possível verificar no excerto de código abaixo, para cada uma das imagens são percorridas as suas anotações e efetuada a leitura da imagem centrada nos valores da *bounding Box* correspondente, onde é efetuado um *resize* da imagem para o tamanho 124x124 (sendo estas dimensões fixadas para todas as imagens), assim como a sua *label*, ficando estes dados armazenados na variável *data*.

```
data = []
img_size = 124
mask = ['face_with_mask']
non_mask = ["face_no_mask"]
labels = { "face_with_mask":0, "face_no_mask":1 , "
    face_with_mask_incorrect":2 }

for i in df["name"].unique():
    f = i+".json"
    for j in getJSON(os.path.join(directory,f)).get("Annotations"):
            if(j['classname'] in labels):
                x,y,w,h = j["BoundingBox"]
                img = cv2.imread(os.path.join(image_directory,i),1)
                img = img[y:h,x:w]
                img = cv2.resize(img,(img_size,img_size))
                data.append([img,labels[j["classname"]]])
random.shuffle(data)
```

Listing 1.1: Código referente ao tratamento dos dados

Em seguida decidiu-se verificar como estavam distribuídas as classes dos dados lidos, pelo que se apresenta na figura 1 os resultados obtidos:

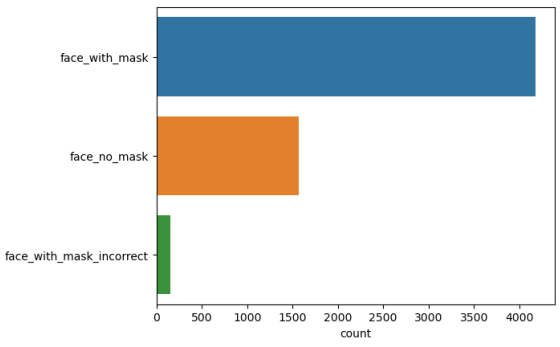


Figura 1: Distribuição numérica por cada *label*

Pela análise da figura, é possível verificar que há uma grande predominância de imagens cujos sujeitos estão com a máscara corretamente colocada, verificando mais de 4000 casos. Já a classe *face_no_mask* apresenta uma quantidade de casos consideravelmente menor, a rondar os 1500 casos, mas onde efetivamente se revela a lacuna dos dados de treino é na classe *face_with_mask_incorrect*, que contém uma quantidade extremamente baixa em relação às outras e não permite ao modelo adquirir capacidade suficiente de se especializar nestes casos.

Balanceamento dos dados de treino

De modo a contornar esta lacuna, decidiu-se adotar mais conjuntos de dados de maneira a balancear os mesmos, fornecendo assim uma maior viabilidade ao modelo.

Após uma pesquisa pelo Kaggle, de modo a acrescentar mais dados do tipo 'face_no_mask', o grupo encontrou um conjunto de dados [3] que contém uma enorme quantidade de imagens de pessoas sem máscara, pelo que não se utilizou todas as imagens. Para este conjuntos de dados, foi criado um *notebook makingDataset.ipynb*, de modo a criar um ficheiro '*.csv*' (*datasets*) com os nomes das imagens juntamente com as respetivas *bounding boxes* (calculadas com o algoritmo *YOLO*) e *classname('face_no_mask')*.

É importante referir que foram comparados dois algoritmos de deteção de faces em imagens (*SSD* e *YOLO*) na secção 5.1, onde são apresentados os resultados dessa comparação e é explicado a escolha pelo algoritmo *YOLO*.

Já para balancear os dados do tipo 'face_with_mask_incorrect', encontrou-se um conjunto de dados[4], que contém 3 tipos de imagens (correspondentes às 3 classes pretendidas para o modelo). Neste contexto, incorporou-se apenas o conjunto de todas as imagens de faces com máscara colocada incorretamente, que se encontram na pasta *mask_weared_incorrect* disponível no *kaggle*.

Para este conjunto de dados foi também criado um *notebook* de forma a criar um *dataset* com nome das imagens, *bounding boxes* e *classnames*. Uma vez que neste conjunto de dados, as imagens já estão recortadas pela face, não é necessário aplicar o algoritmo *YOLO* para calcular as *bounding boxes*, pelo que se passou todos os campos da *bounding boxes* para 0.

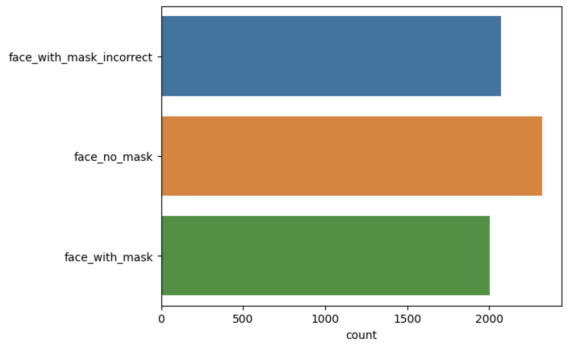


Figura 2: Balanceamento final dos dados de treino

Assim, no fim de todo este processo, é possível afirmar que o conjunto de dados se encontra balanceado.

Separação das features das labels e normalização dos dados

Em seguida, procedeu-se à separação entre as *features* e as *labels* no conjunto X e Y, respetivamente. É importante referir que ao obter o *shape* de cada *feature* (que corresponde a uma imagem) se verifica que é (124,124,3), ou seja, 124 pixeis de altura, 124 pixeis de largura e com 3 canais de cor (rgb).

De modo a proceder à normalização dos valores de X, este é convertido num *array* através da função *np.array()*, normalizando os valores dividindo-os por 255. Este processo de divisão por 255 é uma forma comum de normalização para imagens, uma vez que os valores originais dos pixeis estão no intervalo de 0 a 255 (correspondentes aos valores de intensidade RGB). De notar que, o Y também convertido num *array* através da função *np.array()*.

[3] https://www.kaggle.com/datasets/ashwingupta3012/human-faces?select=Humans
[4] https://www.kaggle.com/datasets/vijaykumar1799/face-mask-detection

Na figura 3, é apresentado uma imagem exemplificativa de cada uma das diferentes *labels* adotadas.

Figura 3: Exemplos de imagem para cada *label*

Divisão dos dados e preparação dos DataLoaders

Por fim, de forma a preparar os dados para serem introduzidos no modelo foi necessário realizar uma divisão dos dados em conjuntos de treino, teste e validação, com vista a avaliar o desempenho do modelo. O conjunto de dados foi inicialmente dividido em treino e teste, sendo 20% para teste e 80% para treino. Destes 80%, 10% foram para validação e os restantes 90% foram efetivamente para o treino do modelo. Além disso, foi utilizada uma técnica de *stratified sampling* para garantir uma distribuição equilibrada das classes nas amostras. Como resultado, foi obtida a seguinte distribuição de dados pelo treino, teste e validação.

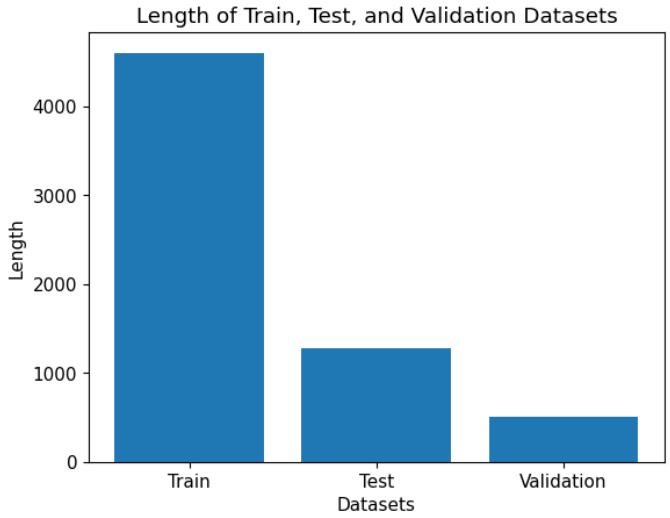


Figura 4: Quantidade de dados de treino, teste e validação

Foi ainda necessário converter estes dados em tensores, sendo aplicado um *reshape* para ajustar a dimensão dos tensores de entrada para terem formato (-1, 3, 124, 124), onde basicamente se inverte a ordem do número de canais de cores com as dimensões das imagens, sendo o valor -1 para manter o mesmo número de imagens de entrada.

Para facilitar o processo de alimentação dos dados utilizando *batches* durante o treino e teste do modelo, foram criados os *DataLoaders*. Estes são objetos do *PyTorch* que permitem carregar os dados de forma eficiente, processando os vários *batches* em paralelo. Foi definido um *batch size* de 32, o que significa que a cada iteração do processo de treino, serão processadas 32 imagens de

uma vez, proporcionando assim uma otimização do tempo de treino, aproveitando, de forma mais eficiente, a capacidade de paralelização da *GPU*.

4 Descrição do Modelo e suas características

No presente capítulo descrever-se-á o modelo utilizado no projeto, que adota uma arquitetura de rede neural convolucional (CNN). A CNN é conhecida por a sua eficácia no processamento de imagens, tornando-a uma escolha adequada para a tarefa de deteção e classificação de máscaras faciais.

Após vários testes de configurações diferentes de arquiteturas, considerou-se que a melhor seria a seguinte:

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
Model                                    [32, 3]                   --
├─Conv2d: 1-1                            [32, 32, 124, 124]        896
├─Dropout: 1-2                           [32, 32, 124, 124]        --
├─Conv2d: 1-3                            [32, 64, 122, 122]        18,496
├─Dropout: 1-4                           [32, 64, 122, 122]        --
├─Conv2d: 1-5                            [32, 128, 120, 120]       73,856
├─Dropout: 1-6                           [32, 128, 120, 120]       --
├─MaxPool2d: 1-7                         [32, 128, 60, 60]         --
├─Flatten: 1-8                           [32, 460800]              --
├─Linear: 1-9                            [32, 50]                  23,040,050
├─Dropout: 1-10                          [32, 50]                  --
├─Linear: 1-11                           [32, 3]                   153
==========================================================================================
Total params: 23,133,451
Trainable params: 23,133,451
Non-trainable params: 0
Total mult-adds (G): 44.02
==========================================================================================
Input size (MB): 5.90
Forward/backward pass size (MB): 841.69
Params size (MB): 92.53
Estimated Total Size (MB): 940.13
==========================================================================================
```

Figura 5: Arquitetura do Modelo

Na inicialização do modelo, são definidas as diferentes camadas e suas configurações, pelo que cada uma conta com uma funcionalidade na rede, sendo descritas em seguida:

- **camadas convolucionais(*Conv2d*)** - aplicam filtros convolucionais para capturar informações importantes da imagem em diferentes níveis de abstração.
- **camadas de *Dropout*** - utilizadas para regularização, evitando o *overfitting* durante o treino.
- **camada de *max pooling* (*MaxPool2d*)** - reduz a dimensionalidade da saída das camadas convolucionais
- **camada *Flaten*** - usada para transformar o output das camadas anteriores (tensores tridimensionais) em uma forma linear de forma que possa ser conectada a uma ou mais camadas densamente conectadas (*fully connected layers*)
- **camadas *Linear*** - responsáveis pela classificação final das características extraídas, mapeando-as para as classes de máscaras faciais.

De referir que a última camada tem um output de [32,3], uma vez que se está a prever as 3 classes referidas com um batch size de 32.

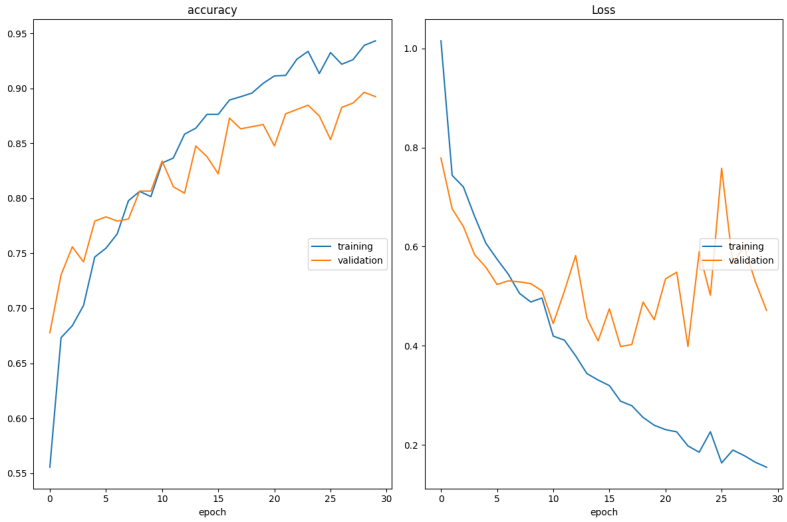


Figura 6: Gráfico do processo de treino e validação do modelo

No processo de treino, foi utilizada a função de *loss* a *CrossEntropyLoss()* e como *optimizer* a função *Adam* e um learning rate de 0.001.

Através da análise da figura 6, constata-se que se obteve um modelo bastante capaz, tendo tido em conta os fenómenos de *overfitting* e *underfitting*. Após análise dos resultados, analisou-se até as 30 *epochs*, uma vez que a partir deste valor, o modelo começa a entrar overfitting. Optou-se pelas 25 *epochs* como modelo final, devido ao súbito pico de *loss* e à consistência da *accuracy* da validação após esta época.

Com isto, obteve-se uma *accuracy* de 89% na validação, o que se revela um resultado bastante positivo, como podemos verificar na secção seguinte.

5 Resultados e análise crítica

5.1 Algoritmos de deteção de faces em imagens

Tal como referido anteriormente, algumas das imagens utilizadas ao longo do desenvolvimento não contêm ficheiros de anotações com as correspondentes *bounding boxes*, pelo que se torna obrigatório o uso de ferramentas auxiliares para o efeito, como modelos de deteção de faces.

Com vista a melhorar a performance destes modelos de deteção de faces, é importante referir que antes de alimentar os mesmos com as imagens, foi aplicado um filtro de ajuste do Gamma, sendo esta uma técnica frequentemente usada nesta área de processamento de imagens. Basicamente, esta técnica torna as imagens mais claras, permitindo assim ao modelo analisar melhor a imagem, conduzindo deste modo a melhores resultados.

Para tal, recorreu-se a dois algoritmos de deteção de faces que tinham sido previamente abordados no trabalho de investigação: *SSD (Single Shot Detector)* e *YOLO (You Only Look Once)*, implementando o conceito de *transfer learning*. Estes algoritmos são responsáveis por efetuar a deteção de faces, devolvendo as respetivas *bounding boxes*.

Comparação de Algoritmos SSD vs YOLO

O *YOLO* é um método que reconhece objetos em imagens rapidamente através da divisão em *grid cells*, sendo esta a sua principal característica, uma vez que realiza uma só passagem. Já o *SSD* executa uma rede convolucional na imagem de entrada apenas uma vez e calcula um *feature map*.

(a) Resultados SSD (b) Resultados YOLO

Figura 7

De modo a comparar os resultados obtidos com cada algoritmo de deteção de faces (*SSD* vs *YOLO*), apresenta-se acima na Figura 6, 4 exemplos representativos de vários resultados obtidos com o algoritmo *SSD* (Figura 7a) e *YOLO* (Figura 7b).

Começando pelo exemplos da esquerda, no exemplo de cima (1ª imagem) é possível verificar que, com o algoritmo *SSD*, não é detetada a pessoa que apenas tem metade da face na imagem, ao contrário do *YOLO*, em que é detetada essa mesma face. Passando para o exemplo da direita (2ª imagem), apesar do *SSD* só detetar apenas uma das faces na imagem, produziu resultados melhores do que com *YOLO*, uma vez que este último não detetou nenhuma das faces.

Observando os exemplos de baixo, é possível observar que no exemplo da esquerda, o *SSD* deteta o animal na imagem, ao contrário do algoritmo *YOLO*. Visto que se considera que, neste contexto, a deteção de faces de animais é desnecessária, o algoritmo *YOLO* produz o melhor resultado. Por fim, o exemplo de baixo à direita demonstra que o algoritmo *SSD* deteta erradamente duas faces numa imagem que contém apenas uma pessoa.

Assim, uma vez que o algoritmo *YOLO* produz melhores resultados do que o *SSD* na grande maioria dos casos analisados, decidiu-se usar o algoritmo *YOLO*.

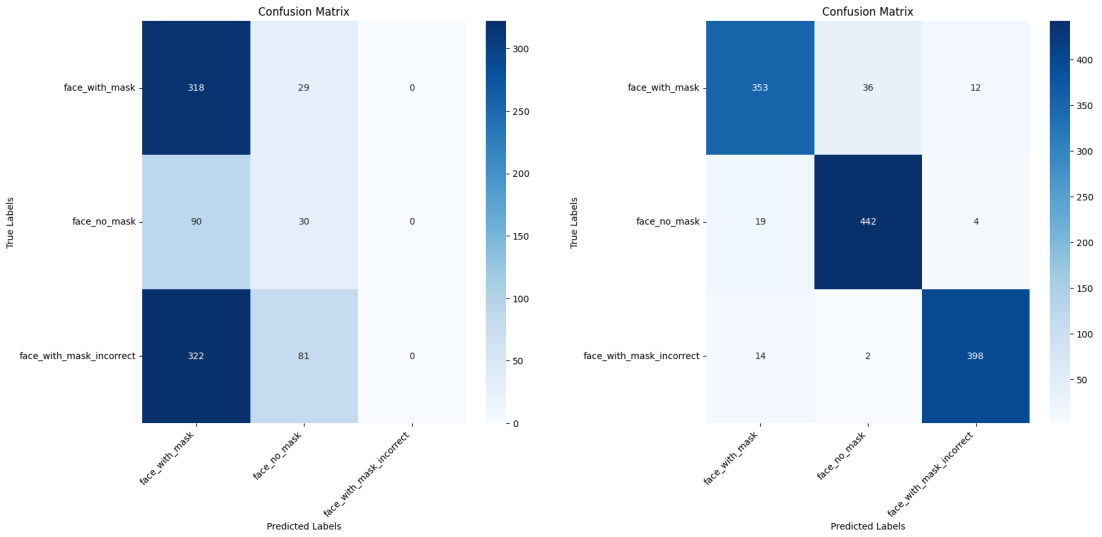
Este processo foi importante, uma vez que, alguns conjuntos de dados encontrados para balancear os dados de treino (nomeadamente com imagens de pessoas sem máscara) não possuíam as *bounding boxes* das faces, tornando-se necessária a utilização deste algoritmo.

5.2 Comparação do modelo de classificação de máscaras faciais com outros

De modo a aferir a qualidade do modelo desenvolvido, efetuou-se uma pesquisa por outros projetos no mesmo âmbito, que também trabalhassem com as 3 classes utilizadas: com máscara, sem máscara e com a máscara incorretamente colocada.

Deste modo, utilizou-se um notebook [5] encontrado no Kaggle, pelo qual se testou o modelo com os dados de teste, construído no *split* dos dados, e se comparou com o desenvolvido pelo grupo. É importante referir que foi necessário um *script* para copiar as imagens do conjunto de teste para a pasta *testSetMaker*. Posteriormente, com um *notebook*, criou-se um *dataset* com as imagens de teste juntamente com ass respetivas *bounding boxes*, de forma análoga à criação dos *datasets* para balanceamento dos dados já explicado anteriormente. Assim, teve-se em conta o tratamento de dados aplicado no treino deste modelo externo e aplicou-se o mesmo nestes dados de teste.

[5] https://www.kaggle.com/code/nettasav/face-mask-detection-classsifier-using-cnn/notebookTesting-the-Results



(a) Modelo extraído do Kaggle

(b) Modelo desenvolvido pelo grupo

Figura 8: Matriz de Confusão

Analisando a figura 8a (que corresponde ao modelo do *Kaggle*), é possível verificar que este modelo não consegue prever corretamente nenhuma imagem de *label* 2 (classe 'face_with_mask_incorrect'), maioritariamente prevê a *label* 1 (classe 'face_with_mask'), confundindo assim as 2 *labels*. Isto é explicado por este ter sido treinado com dados muito desbalanceados, problema com que o grupo também se deparou inicialmente. A classe *face_no_mask* foi prevista com muitas falhas, tendo apenas 30 dos 120 casos sido previstos com sucesso e, novamente, os restantes 90 sido considerados como *face_with_mask*. Relativamente à classe *face_with_mask*, é possível verificar que foi o que obteve maior taxa de sucesso, sendo que grande parte das previsões foram assertivas, o que se justifica pelo facto de a grande maioria dos casos presentes nos dados com que este modelo externo foi treinado, serem correspondentes a esta *label*.

Já o modelo desenvolvido pelo grupo, por análise da Figura 8b, é possível verificar maior sucesso nas previsões, tendo sido predominante em todas as classes. Relativamente ao anterior, é possível notar as melhores melhorias na previsão da classe *face_with_mask_incorrect*, uma vez que o outro modelo não consegui prever esta classe e a justificação desta melhoria é o facto de se ter recorrido aos *datasets* referidos acima para complementar estes casos.

De seguida, de maneira a comparar melhor os modelos, apresenta-se na tabela 1 as *accuracies* relativas a cada uma das classes e a global para cada modelo.

Modelos	***face_with_mask***	***face_no_mask***	***face_with_mask_incorrect***	**Global**
Modelo Kaggle	91.6%	25%	0%	40%
Modelo desenvolvido	88%	95%	96%	93%

Tabela 1: Tabela relativa aos resultados obtidos na matriz de confusão para ambos os modelos

Através da analise à tabela, apesar de o modelo desenvolvido pelo grupo ter uma *accuracy* pior em específico na classe *with_mask*, as *accuracies* das outras 2 classes são claramente superiores, o que torna o modelo mais robusto e generalizável, para além de que é possível verificar que todas as classes foram previstas com uma taxa alta, sendo relativamente equitativa entre todas as classes.

Assim, considera-se bastante mais eficaz o modelo desenvolvido relativamente ao do *kaggle*.

De modo a visualizar ainda alguns exemplos de resultados produzidos pelo modelo desenvolvido pelo grupo, apresenta-se ainda um conjunto de imagens com a respetiva classe prevista.

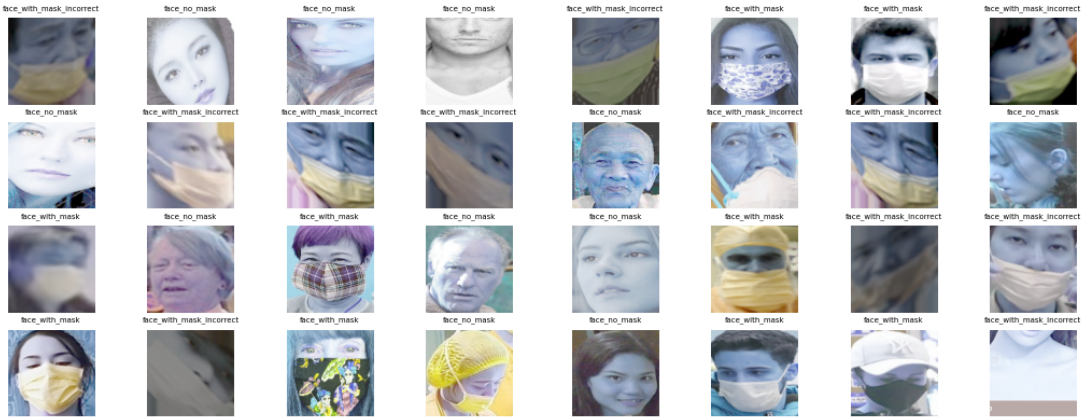


Figura 9: Conjunto de imagem exemplificativas dos resultados produzidos pelo modelo

Analisando a figura 9, é possível verificar que o modelo produz resultados bastante satisfatórios, uma vez que neste conjunto de imagens, erra apenas em 2 casos (o caso da 3ª linha 6ª coluna e o outro caso da última linha última coluna). Neste último caso, é aceitável que o modelo não tenha conseguido prever bem a classe, uma vez que a imagem apresenta apenas uma parte do rosto.

De notar que, uma vez que os dados estão normalizados as imagens encontram-se com pouca variação de cor (mais azuladas).

6 Conclusão

Dado por concluído o projeto, considera-se que se atingiu com sucesso os objetivos propostos, tendo consolidado e aprofundado os conhecimentos abordados ao longo do semestre, nomeadamente na área de aplicação de modelos de Aprendizagem profunda e processamento de imagem, assim como conhecimentos adquiridos pelo trabalho de investigação, principalmente o conceito de *Transfer Learning*.

Considera-se positivo o tratamento dos dados e a análise do conjunto de dados, que permitiram potencializar a rede neuronal e, por sua vez, o modelo preditivo. Para além disso, foi tido especial cuidado com os *finetuning* do modelo, de modo a trabalhar com os melhores parâmetros e, assim, evitar fenómenos de *overfitting* ou *underfitting*.

Ao longo do projeto foram atravessadas algumas dificuldades, como o elevado tempo de treino, que atrasa o desenvolvimento projeto.

Algo que é importante reter é que nem sempre a *accuracy* indica tudo. Como foi possível verificar pelo caso do modelo obtido através do *Kaggle*, em que, apesar da sua *accuracy* ser de 90%, o facto do seu *dataset* de treino ser desbalanceado significa que este modelo não está preparado para prever todas as classes equitativamente. Deste modo, o valor foi meramente ilusório, sendo que quando submetido a um conjunto de dados mais balanceado demonstrou ser ineficiente.

Como trabalho futuro seria interessante expandir o sistema desenvolvido a processamento de vídeo, de modo a permitir a sua utilização em sistemas de gravação em tempo real, como por exemplo câmaras de vídeo-vigilância em locais públicos de forma a detetar automaticamente pessoas com máscara indevidamente colocada ou sem máscara, que aciona um alerta aos órgãos de segurança responsáveis pelo local.