PL 2 - Aplicação de CNNs a *datasets* novos
COVID-19: Deteção do Uso Incorrecto da Máscara Cirúrgica

7 de dezembro, 2020

Discentes:
Diogo Silva, 66682
Bruno Silva, 66611

Docente:
António Cunha

1 Caracterização e preparação dos dados em datasets

O *dataset* utilizado foi o MaskedFace-Net [1]. Este *dataset* é composto por 137 016 imagens, geradas por uma *Generative Adversarial Network* (GAN), sendo aproximadamente metade das imagens máscaras colocadas corretamente, e a outra metade máscaras mal colocadas (nariz exposto, boca exposta, queixo exposto). Assim isto é um problema de classificação binária, com duas classes: uso correto (0), e uso incorreto (1).

Para este trabalho, utilizamos apenas 5142 imagens. Este número bastante mais reduzido irá permitir explorar a eficácia de técnicas de augmentação de dados, e reduzir significativamente o poder computacional necessário para a realização das várias experiências realizadas.

Para a utilização dos dados em CNN's, decidimos dividir os dados em três partes: treino (2608 images); validação (1904 images); teste (630 images). Em cada uma destas partes mantemos o número de classes equilibrado.

1.1 Pré-processamento de Dados

Tal como referido anteriormente, iremos explorar o uso de técnicas de augmentação de dados. Em Keras existem *layers* (camadas) especializadas para este efeito. Neste trabalho iremos utilizar uma *layer RandomFlip* (para transformações horizontais) e a *layer RandomRotation* (para rotações aleatórias de cada imagem). A implementação é descrita na Figura 1.

É também importante fazer o reescalonamento dos dados para tornar o problema mais fácil de resolver, além de ser uma prática comum. A implementação é descrita na Figura 2. Todas as *layers* previamente descritas irão ser aplicadas a todos os modelos construídos.

```
from tensorflow.keras import layers
data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal"), # "espelho" horizontal
  layers.experimental.preprocessing.RandomRotation(0.2), # [-20% * 2pi, 20% * 2pi]
])
```

Figura 1: *Layers* de augmentação de dados em Keras

```python
from tensorflow.keras import layers
data_scaling = tf.keras.Sequential([
  layers.experimental.preprocessing.Rescaling(scale=1./127.5, offset=-1), # entre [-1,1]
])
```

Figura 2: *Layer* de reescalonamento de dados em Keras

2 Experiências iniciais

2.1 Funções de ativação

Testamos as seguintes funções: tangente hiperbólica; sigmoide; ReLU; LeakyReLU. Os resultados são apresentados na Figura 3, onde podemos observar que tanto a LeakyReLU apresenta um desempenho superior em relação às restantes. Para as próximas experiências utilizaremos então a função LeakyReLU.

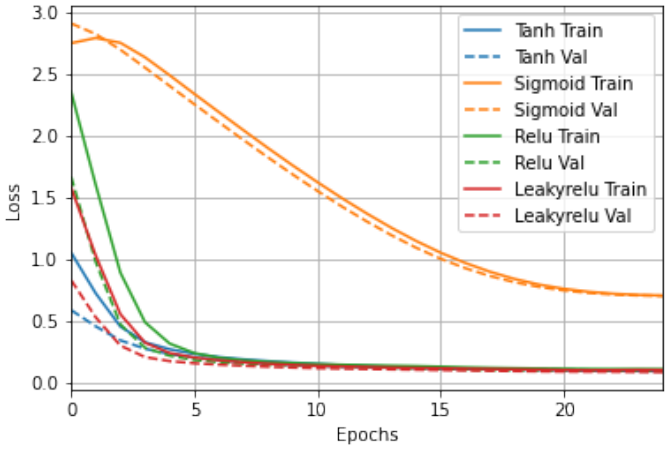

Figura 3: Desempenho de várias funções de ativação na mesma CNN.

2.2 Otimizadores

Testamos os seguintes otimizadores: SDG; Adam; RMSprop. Os resultados são apresentados na Figura 4, onde podemos observar que o otimizador Adam tem um desempenho ligeiramente melhor que os restantes, no entanto podemos afirmar que a escolha de diferentes otimizadores não parece ter um grande impacto no desempenho de um modelo CNN.

2.3 *Learning rates*

Testamos os seguintes valores: 0.0001; 0.0006; 0.001; 0.006; 0.01. Os resultados são apresentados na Figura 5, onde podemos observar que um *learning rate* inicial de 0.001 revela-se mais vantajoso, pois é o que inicia o treino com o valor de *loss* mais pequeno.

2.4 Tamanho e complexidade da CNN

2.4.1 CNN extramemente pequena

Em primeiro lugar construimos uma CNN com um tamanho bastante limitado (Figura 6). Apesar de este modelo ter uma capacidade de aprendizagem bastante limitada, podemos verificar na Figura 6, este aprensenta uma elevada capacidade de aprendizagem

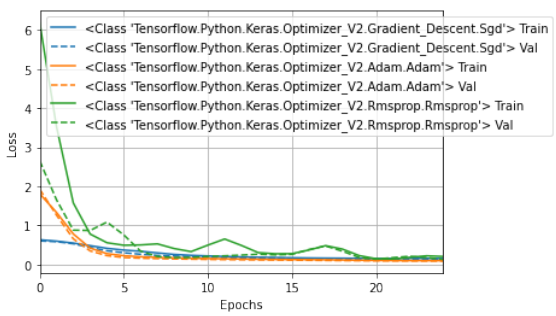

Figura 4: Desempenho de vários otimizadores na mesma CNN.

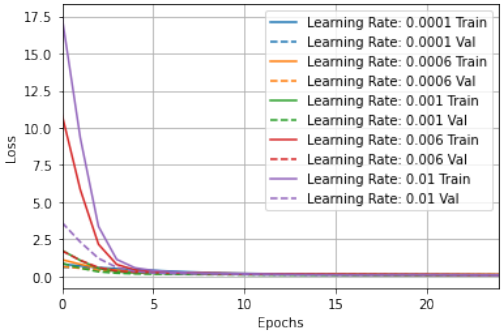

Figura 5: Desempenho de vários *learning rates* na mesma CNN.

tanto para os dados de treino como validação (Figura 7). Nos dados de teste este modelo apresenta uma loss de ≈ 0.17, e uma *binary accuracy* de ≈ 0.94.

```
model = Sequential([
        # Input
        layers.InputLayer(input_shape=input_shape),

        # Augmentação de dados
        data_augmentation,

        # Rescale dos dados
        data_scaling,

        # Camadas convolucionais
        layers.Conv2D(1, 3),
        layers.LeakyReLU(alpha=0.1),
        layers.MaxPooling2D(),

        # Classificação
        layers.Flatten(),
        layers.Dense(5),
        layers.LeakyReLU(alpha=0.1),
        layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')] )
```

Figura 6: Definição do modelo.

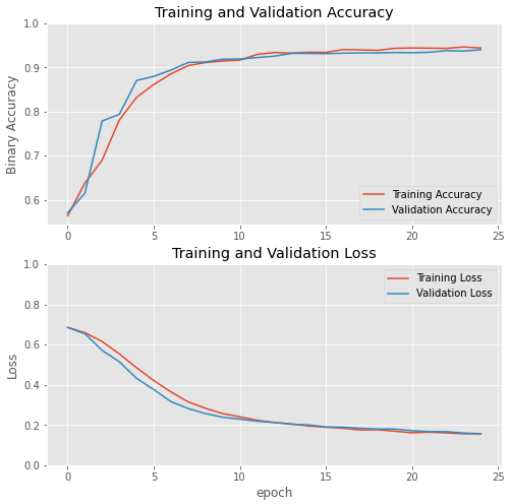

Figura 7: *Learning curves* da CNN de tamanho bastante reduzido.

2.4.2 CNN Complexa

De seguida construímos uma CNN mais complexa (Figura 8). Este modelo tem uma complexidade consideravelmente maior em relação ao anterior, por isso foi importante aplicar técnicas de regularização (L2, *Dropout*) para evitar o *overfitting*. Nos dados de teste este modelo apresenta uma *loss* de ≈ 0.08, e uma *binary accuracy* de ≈ 0.99.

2.4.3 *Transfer learning* e *fine-tuning*

Podemos tentar melhorar o modelo através do uso de redes pré-treinadas para a extração de *features*, adicionando as nossas próprias camadas de classificação. De seguida treinamos parte da pré-treinada e as nossas próprias camadas com um learning rate muito baixo, pois este processo pode levar rapidamente a *overfitting*. Iremos utilizar a rede pré-treinada Xception [2], para a construção do novo modelo (Figura 10). Através desta técnica o modelo atingiu uma *loss* de ≈ 0.04, e uma *binary accuracy* de ≈ 1!

```python
model = Sequential([
        # Input
        layers.InputLayer(input_shape=input_shape),

        # Augmentação de dados
        data_augmentation,

        # Rescale dos dados
        data_scaling,

        # Camadas convolucionais
        layers.Conv2D(64, 3),
        layers.LeakyReLU(alpha=0.1),
        layers.MaxPooling2D(),

        # Camadas convolucionais
        layers.Conv2D(64, 3),
        layers.LeakyReLU(alpha=0.1),
        layers.MaxPooling2D(),

        # Classificação
        layers.Flatten(),

        layers.Dense(512, activity_regularizer=tf.keras.regularizers.L2(0.001)),
        layers.LeakyReLU(alpha=0.1),
        layers.Dropout(0.2), # regularizacao

        layers.Dense(1024, activity_regularizer=tf.keras.regularizers.L2(0.001)),
        layers.LeakyReLU(alpha=0.1),
        layers.Dropout(0.2), # regularizacao
        layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')])
```

Figura 8: Definição do modelo.

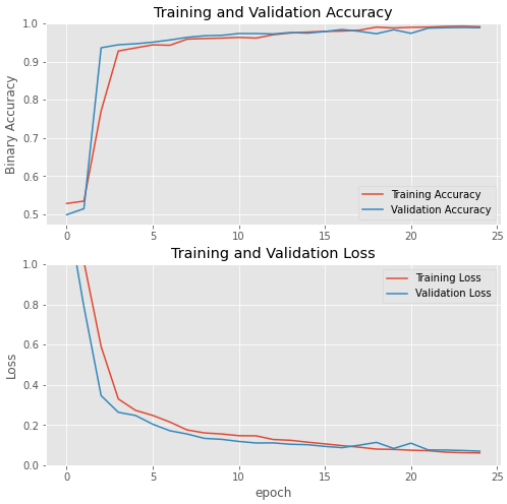

Figura 9: *Learning curves* da CNN complexa

```python
base_model = tf.keras.applications.Xception(
    weights="imagenet",  # Carregar pesos da ImageNet
    input_shape=input_shape,
    include_top=False,
)  # Nao incluir classificador do Xception

# Congelar modelo base
base_model.trainable = False

# Criar novo modelo em cima
inputs = tf.keras.Input(shape=input_shape)

x = data_augmentation(inputs)  # Aplicar augmentacao

# Xception precisa de valores [-1, 1]
x = data_scaling(x)

# Adicionar Xception
x = base_model(x, training=False)
x = layers.GlobalMaxPooling2D()(x)


# Classificação
x = layers.Dense(512)(x)
x = layers.LeakyReLU(alpha=0.1)(x)
x = layers.Dropout(0.3)(x) # Regularização

x = layers.Dense(512)(x)
x = layers.LeakyReLU(alpha=0.1)(x)
x = layers.Dropout(0.3)(x) # Regularização

outputs = layers.Dense(1, activation = 'sigmoid')(x)
model = tf.keras.Model(inputs, outputs)

initial_lr = 0.001
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate = initial_lr),
    loss=tf.keras.losses.BinaryCrossentropy(),
    metrics=[tf.keras.metrics.BinaryAccuracy(name='accuracy')]
)
```

Figura 10: Definição do modelo *transfer learning*.

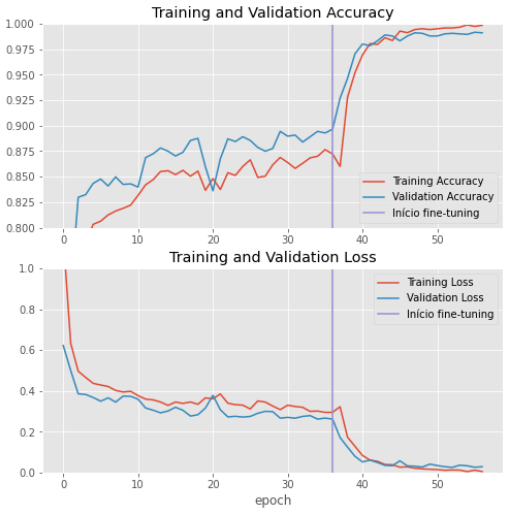

Figura 11: *Learning curves* do modelo de *transfer learning*

Referências

[1] Adnane Cabani, Karim Hammoudi, Halim Benhabiles, and Mahmoud Melkemi. Maskedface-net – a dataset of correctly/incorrectly masked face images in the context of covid-19, 2020.

[2] François Chollet. Xception: Deep learning with depthwise separable convolutions, 2017.