Universidade da Beira Interior
Departamento de Informática

Nº Trabalho Prático 3: *Convolutional Neural Networks (CNNs)*

Elaborado por:

João Miguel Baltazar Martins

Orientador:

Professor Doutor Hugo Pedro Proença

25 de novembro de 2023

Acrónimos

CNN Convolutional Neural Network

VGG Visual Geometry Group

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Excertos de Código

Capítulo 1

Introdução

1.1 Enquadramento

Este projeto tem como foco a aplicação das Convolutional Neural Network (CNN) na análise de imagens faciais, utilizando o conjunto de dados "AR.zip", disponibilizado pelo professor na sua página.

Este conjunto de dados é composto por 3.315 imagens em RGB e contém 136 indivíduos em diferentes expressões faciais e condições, identificados por género e acessórios.

As imagens estão categorizadas por género, expressão facial e atributos como óculos e iluminação. O formato de nomenclatura, distinguindo o sexo masculino ("m-xx-yy.raw.jpg") do sexo feminino ("w-xxx-yy.raw.jpg"), permite identificar exclusivamente cada sujeito ('xx') e as diferentes expressões e condições ('yy').

1.2 Objetivo

Este projeto tem como objetivo principal a construção de modelos de CNN robustos para prever quatro atributos fundamentais:

a. **Identificação (ID):** Prever o identificador único de um indivíduo numa imagem.

b. **Expressão Facial:** Classificar a expressão facial presente na imagem, abrangendo diversas emoções e condições.

c. **Género:** Determinar o género do sujeito na imagem.

d. **Óculos:** Identificar a presença ou ausência de óculos nas imagens.

De notar que deve-se implementar a função `train_on_batch()` e os *data generators*.

1.3 Abordagem ao Problema

Partindo do script `scr_deep_learning_keras.py` (disponibilizado pelo professor), é necessário adaptá-lo para lidar com grandes conjuntos de dados, utilizando a função `train_in_batch()` e os geradores de dados. Além disso, irá ser explorado arquiteturas de CNN, e estruturas reconhecidas como a Visual Geometry Group (VGG)

1.4 Organização do Documento

1. O primeiro capítulo – **Introdução** – apresenta o projeto, o enquadramento para o mesmo, o objetivo do projeto, a abordagem a seguir e a respetiva organização do documento.

2. O segundo capítulo – **Desenvolvimento** – aborda a implementação do *script* geral que depois se adequa a cada tarefa de previsão, nomeadamente, da Identificação (ID), Expressão Facial, Género e Óculos. São enfatizas noções teóricas de CNN e justificadas as escolhas críticas de hiperparâmetros, como o tamanho do lote (*batch size*) e o número de épocas, que influenciam o desempenho do modelo. Para cada *script* especifico é exibido um gráfico que ilustra a relação entre o conjunto de treino e o conjunto de valição nas temáticas da *loss* e *accuracy*.

3. O terceiro capítulo – **Exposição e Análise dos Resultados** – serão expostos e analisados os resultados obtidos para cada tarefa de previsão.

4. O quarto capítulo – **Conclusão** – refere as conclusões principais a tirar deste trabalho, bem como uma reflexão sobre um trabalho futuro.

Capítulo 2

Desenvolvimento

2.1 Introdução

Neste capítulo, apresentaremos o *script* geral que serve como modelo para a tarefa de previsão de cada elemento previamente discriminado em1.2.

Posteriormente, abordaremos a complementarização do *script* para cada elemento e a respetiva análise aos resultados e a análise em secções dedicadas e os respetivos resultados.

2.2 *Script* Molde

Hiperparametros

No código ilustrado na figura 2.1, foi definido:

- Dimensões das imagens: todas as imagens serão redimensionadas para o tamanho (256x256) antes de serem usadas pelo modelo. É uma escolha comum para dimensões de imagem em muitas tarefas de visão computacional. É um equilíbrio entre ter uma resolução alta o suficiente para capturar detalhes importantes e não ter uma resolução tão alta que os requisitos computacionais se tornem proibitivos.

- Tamanho do Lote: o tamanho do lote será oito. Este é o número de amostras que serão passadas pelo modelo de uma vez durante o treino. Os pesos do modelo são atualizados após cada lote. Um tamanho de lote menor significa que o modelo será atualizado com mais frequência, o que pode levar a uma convergência mais rápida, mas também pode resultar em mais ruído nas atualizações dos pesos.

- Épocas: serão estabelecidas 10 épocas. O número ideal de épocas pode depender da tarefa específica, da arquitetura do modelo e do tamanho do conjunto de dados. Muitas épocas podem levar ao *overfitting*, onde o modelo aprende a ter um bom desempenho nos dados de treino, mas não em dados novos e invisíveis.

Os valores escolhidos estão diretamente associados com a performance. Para valores superiores de renderização das imagens e/ou de *batch* o programa falhou.

```
# Define the image dimensions
img_rows, img_cols = 256, 256

# Define the batch size and number of epochs
batch_size = 8
epochs = 10
```

Excerto de Código 2.1: Valores dos Hiperparâmetro

Divisão do Conjunto de Dados

A divisão do conjunto de dados foi feita em três subconjuntos: um conjunto de treino, um conjunto de validação e um conjunto de teste. Na figura 2.2, é ilustrado como foi feita esta divisão.

1. `X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0,2, random_state=SEED_VALUE)` - Esta linha divide o conjunto de dados original (X e y) num conjunto temporário (`X_temp` e `y_temp`) e um conjunto de teste (`X_test` e `y_test`). O conjunto de teste é 20% do conjunto de dados original, conforme especificado por `test_size=0,2`. O parâmetro `random_state` é usado para garantir que a divisão seja reproduzível.

2. `X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0,25, random_state=SEED_VALUE)` - Esta linha divide o conjunto temporário num conjunto de treino (X_train e y_train) e um conjunto de validação (X_val e y_val). O conjunto de validação é 25% do conjunto temporário, o que significa que é 20% do conjunto de dados original.

```
# Split the dataset into training/validation and test subsets
# 20% - Test || 20% - Validation || 60% - Train
```

```
X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2,
    random_state=SEED_VALUE)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp,
    test_size=0.25, random_state=SEED_VALUE)  # 0.25 x 0.8 = 0.2
```

Excerto de Código 2.2: Dividir o Conjunto de Dados

Implementação CNN utilizando o *Keras*

O CNN utilizando o *Keras* foi implementado, na figura 2.3, da seguinte forma:

1. O modelo é inicializado como um modelo Sequencial, que é uma pilha linear de camadas.

2. A primeira camada é uma camada convolucional 2D com 32 filtros, um tamanho de kernel de 3x3, padding 'same' (o que significa que a saída tem a mesma largura e altura que a entrada) e uma função de ativação ReLU. Esta camada também especifica a forma de entrada das imagens.

3. A segunda camada é outra camada convolucional 2D com os mesmos parâmetros da primeira camada, mas sem a necessidade de especificar a forma de entrada.

4. A terceira camada é uma camada de ***pooling*** máximo com um tamanho de pool de 2x2, que reduz as dimensões espaciais (largura, altura) da entrada, tomando o valor máximo sobre a janela definida por pool_size para cada dimensão ao longo do eixo das características.

5. Os passos de 2 a 4 são repetidos mais duas vezes, mas com 64 filtros nas camadas convolucionais.

6. Após as camadas convolucionais e de pooling, o modelo possui uma camada de ***flatten***, que converte a matriz 2D (altura, largura) para uma matriz 1D. Isso é necessário porque as camadas densas seguintes esperam a entrada nesse formato.

7. A próxima camada é uma camada densa (totalmente conectada) com 512 unidades e uma função de ativação ReLU.

8. A camada final é outra camada densa com um número de unidades igual ao número de classes na saída (num_classes), e uma função de ativação ***softmax***. A função softmax garante que os valores de saída sejam probabilidades que somam 1, tornando-a adequada para tarefas de classificação **multiclasse**.

Esta arquitetura de modelo é um padrão comum em CNN para classificação de imagens, onde há uma série de camadas convolucionais e de pooling para extrair características das imagens, seguidas por camadas densas para realizar a classificação com base nessas características.

```
def cnn_model(input_shape=(img_rows, img_cols, 3), num_classes=
    num_classes):
    model = Sequential()
    model.add(Conv2D(filters=32, kernel_size=3, padding='same',
        activation='relu', input_shape=input_shape))
    model.add(Conv2D(filters=32, kernel_size=3, padding='same',
        activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(filters=64, kernel_size=3, padding='same',
        activation='relu'))
    model.add(Conv2D(filters=64, kernel_size=3, padding='same',
        activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(filters=64, kernel_size=3, padding='same',
        activation='relu'))
    model.add(Conv2D(filters=64, kernel_size=3, padding='same',
        activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    return model


cnn_model = cnn_model(input_shape=(img_rows, img_cols, 3), num_classes=
    num_classes)
cnn_model.compile(optimizer=RMSprop(),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

Excerto de Código 2.3: Implementação CNN Model

Implementação VGG

No enunciado do trabalho pede também para ser implementado diferentes arquiteturas de rede como, por exemplo, VGG. Na figura 2.4 está ilustrada a sua implementação.

- `vgg_model.compile(optimizer=RMSprop(), loss='categorical_crossentropy', metrics=['accuracy'])` - esta linha compila o modelo com o otimizador RMSprop, a função de perda de entropia cruzada categórica e a precisão como métrica. O otimizador RMSprop é uma escolha popular para redes neurais, e a função de perda de entropia cruzada categórica é adequada para tarefas de classificação multi-classe.

Esta arquitetura de modelo é um padrão comum ao utilizar a transferência de aprendizado para tarefas de classificação de imagens. A ideia é usar um modelo pré-treinado para extrair características das imagens e, em seguida, adicionar algumas camadas no topo para realizar a classificação com base nessas características.

```
def vgg_model(input_shape=(img_rows, img_cols, 3), num_classes=2):
    base_model = VGG16(weights='imagenet', include_top=False,
        input_shape=input_shape)
    base_model.trainable = False  # Freeze the base model

    model = Sequential()
    model.add(base_model)
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    return model

# Create the VGG16 model
vgg_model = vgg_model(input_shape=(img_rows, img_cols, 3), num_classes
    =2)
vgg_model.compile(optimizer=RMSprop(),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

Excerto de Código 2.4: Implementação VGG Model

Data Generators

O código ilustrado na figura 2.5 é responsável por criar geradores de dados para os conjuntos de dados de treino e validação.

- `ImageDataGenerator()` - esta é uma utilidade do Keras que gera lotes de dados de imagem tensor com aumento de dados em tempo real. Neste caso, nenhum aumento de dados é especificado, então ela simplesmente normaliza as imagens.

- `train_datagen.flow(X_train, y_train, batch_size=batch_size)` - cria um gerador Python que itera sobre os dados de treino (`X_train` e

`y_train`) em lotes de tamanho `batch_size`. Isso é útil ao lidar com conjuntos de dados grandes que não cabem na memória, pois permite que o modelo seja treinado em pequenos lotes de dados de cada vez. `val_datagen.flow(X_val, y_val, batch_size=batch_size)` - faz o mesmo para os dados de validação (`X_val` e `y_val`).

O modelo será treinado em lotes de dados gerados pelo `train_generator`, e o seu desempenho será avaliado em lotes de dados gerados pelo `val_generator`.

```
# Create data generators
train_datagen = ImageDataGenerator()
val_datagen = ImageDataGenerator()

train_generator = train_datagen.flow(X_train, y_train, batch_size=
    batch_size)
val_generator = val_datagen.flow(X_val, y_val, batch_size=batch_size)
```

Excerto de Código 2.5: Data Generators

Training and Testing Loop for CNN model

Esta secção do código, ilustrada na figura 2.6, implementa um *loop* de treino e teste personalizado para um modelo CNN.

De forma detalhada:

1. Quatro listas são inicializadas para acompanhar as *losses* e a *accuracies* de treino e validação para cada época.

2. O loop externo é executado por um número de iterações igual ao número de épocas. Cada iteração desse loop representa uma passagem completa pelo conjunto de dados.

3. Dentro do loop externo, mais quatro listas são inicializadas para acompanhar a *loss* e a *accuracy* de treino e validação para cada lote na época atual.

4. O próximo loop é executado por um número de iterações igual ao número de lotes nos dados de treino. Em cada iteração, um lote de dados é recuperado do gerador de dados de treino, e o modelo é treinado neste lote usando o método `train_on_batch`. A *loss* e a *accuracy* para este lote são então adicionadas às listas para a época atual.

5. O próximo loop faz a mesma coisa para os dados de validação, mas usando o método `test_on_batch` para avaliar o modelo nos dados de validação sem atualizar os pesos do modelo.

6. Depois de todos os lotes serem processados, a média da *loss* e da *accuracy* de treino e validação para a época atual é calculada e mostrada.

7. Finalmente, a média da *loss* e da *accuracy* para a época atual são adicionadas às listas para todas as épocas.

8. Após o loop de treino, um gerador de dados de teste é criado da mesma maneira que os geradores de dados de treino e validação.

9. O loop de teste é executado por um número de iterações igual ao número de lotes nos dados de teste. Em cada iteração, um lote de dados é recuperado do gerador de dados de teste, e o modelo é avaliado neste lote usando o método `test_on_batch`. A *loss* e da *accuracy* para este lote são então adicionadas às listas para os dados de teste.

10. Depois que todos os lotes de teste serem processados, a média da *loss* e da *accuracy* de teste é calculada e mostrada.

Este ciclo de treino e teste personalizado oferece mais controlo sobre o processo de treino e teste do que os métodos padrão como o `fit` , e permite modificações no processo de treino no meio de uma época.

```
# Training loop using train_on_batch
train_loss = []
train_acc = []
val_loss = []
val_acc = []

for epoch in range(epochs):
    print(f"Epoch {epoch + 1}/{epochs}")

    train_loss_epoch = []
    train_acc_epoch = []
    val_loss_epoch = []
    val_acc_epoch = []

    for step in range(len(X_train) // batch_size):
        X_batch, y_batch = next(train_generator)
        loss, acc = cnn_model.train_on_batch(X_batch, y_batch) # Change
            Network Architectures (CNN or VGG)
        train_loss_epoch.append(loss)
        train_acc_epoch.append(acc)

    for step in range(len(X_val) // batch_size):
        X_batch, y_batch = next(val_generator)
        loss, acc = cnn_model.test_on_batch(X_batch, y_batch) # Change
            Network Architectures (CNN or VGG)
```

```
        val_loss_epoch.append(loss)
        val_acc_epoch.append(acc)

    avg_train_loss = np.mean(train_loss_epoch)
    avg_train_acc = np.mean(train_acc_epoch)
    avg_val_loss = np.mean(val_loss_epoch)
    avg_val_acc = np.mean(val_acc_epoch)

    print(f"Train - Loss: {avg_train_loss}, Accuracy: {avg_train_acc}")
    print(f"Validation - Loss: {avg_val_loss}, Accuracy: {avg_val_acc}")

    train_loss.append(avg_train_loss)
    train_acc.append(avg_train_acc)
    val_loss.append(avg_val_loss)
    val_acc.append(avg_val_acc)


# Create a test data generator
test_datagen = ImageDataGenerator()

test_generator = test_datagen.flow(X_test, y_test, batch_size=batch_size
    )

# Test loop
test_loss = []
test_acc = []

print("Testing the model...")

for step in range(len(X_test) // batch_size):
    X_batch, y_batch = next(test_generator)
    loss, acc = cnn_model.test_on_batch(X_batch, y_batch)
    test_loss.append(loss)
    test_acc.append(acc)

avg_test_loss = np.mean(test_loss)
avg_test_acc = np.mean(test_acc)

print(f"Test - Loss: {avg_test_loss}, Accuracy: {avg_test_acc}")
```

Excerto de Código 2.6: Training and Testing Loop

2.3 Ajuste do *Script* para a Previsão de IDs

O trecho de código ilustrado em 2.7 indica que

1. Primeiramente extraem-se as etiquetas dos nomes dos arquivos. Em se-

guida, converte as etiquetas, que são inicialmente strings, em IDs de inteiros únicos para cada etiqueta distinta. Finalmente, transforma esses IDs de inteiros numa representação de codificação *one-hot*, de forma a ter um formato adequado para treinar a rede neuronal.

```
# Load the dataset
X = []
y = []

# Load the images and labels into X and y respectively
for filename in os.listdir(dataset_path):
    if filename.endswith(".jpg"):
        img = load_img(os.path.join(dataset_path, filename), target_size
            =(img_rows, img_cols))
        X.append(img_to_array(img))
        id = filename.split('-')[1]  # Assuming the ID is the first part
             of the filename, separated by '-'
        y.append(int(id))

# Convert y to integer IDs
unique_ids = list(set(y))
id_to_int = {id: i for i, id in enumerate(unique_ids)}
y = [id_to_int[id] for id in y]

# Convert y to one-hot encoding
num_classes = len(unique_ids)
y = to_categorical(y, num_classes)
```

Excerto de Código 2.7: Parte que pre-processa os dados para prever IDs

Ajuste do *Script* Para a Previsão de Expressões Faciais

O trecho de código ilustrado em 2.8 mostra que:

1. Para cada arquivo, extrai-se o identificador de características do nome do arquivo e mapeia-se esse identificador para uma expressão facial. As imagens são carregadas e redimensionadas, e apenas as imagens correspondentes às expressões faciais definidas no mapeamento são consideradas. As expressões faciais correspondentes são então adicionadas à lista de etiquetas y.

2. Após carregar todas as imagens e expressões faciais, o código converte as expressões faciais, que são strings, em IDs de inteiros únicos para cada expressão facial distinta. Para isso, cria-se uma lista de todas as expressões faciais únicas e imprime-se essas classes. Em seguida, cria-se um mapeamento de cada expressão facial para um inteiro único e

substituem-se as expressões faciais em y pelos seus IDs de inteiros correspondentes.

```
# Define the mapping from 'yy' to facial expression

feature_id_to_expression = {
    '1': 'neutral',
    '2': 'smile',
    '3': 'anger',
    '4': 'scream',
    # Add other mappings
}

for filename in os.listdir(dataset_path):
    if filename.endswith(".jpg"):
        feature_id = filename.split('-')[2].split('.')[0]  # Extract 'yy
            ' from the filename
        expression = feature_id_to_expression.get(feature_id)  # Map 'yy
            ' to facial expression
        if expression is not None:  # Only consider images corresponding
             to the four classes
            img = load_img(os.path.join(dataset_path, filename),
                target_size=(img_rows, img_cols))
            X.append(img_to_array(img))
            y.append(expression)

# Convert y to integer expressions
unique_expressions = list(set(y))
print("Detected classes:", unique_expressions)


expression_to_int = {expression: i for i, expression in enumerate(
    unique_expressions)}
y = [expression_to_int[expression] for expression in y]
# Convert y to one-hot encoding
num_classes = len(unique_expressions)
print("Classes Number: ", num_classes)
```

Excerto de Código 2.8: Parte que pre-processa os dados para prever a expressão facial

Ajuste do *Script* Para a Previsão de Género

No trecho de código ilustrado em 2.9 é possível observar que:

1. Para cada arquivo, extrai-se o identificador de género do nome do arquivo e mapeia-se esse identificador para um rótulo de género. As ima-

gens são carregadas e redimensionadas, e apenas as imagens correspondentes aos rótulos de género definidos no mapeamento são consideradas. Os rótulos de género correspondentes são então adicionados à lista de etiquetas y.

2. Após carregar todas as imagens e rótulos de género, o código converte os rótulos de género, que são strings, em IDs de inteiros únicos para cada rótulo de género distinto. Para isso, cria-se uma lista de todos os rótulos de género únicos e imprime-se essas classes. Em seguida, cria-se um mapeamento de cada rótulo de género para um inteiro único e substituem-se os rótulos de género em y pelos seus IDs de inteiros correspondentes.

```
feature_id_to_gender = {
    'm': 'male',
    'w': 'women',
    # Add other mappings if necessary
}

for filename in os.listdir(dataset_path):
    if filename.endswith(".jpg"):
        gender_id = filename[0]  # Extract the gender identifier from
            the filename
        gender = feature_id_to_gender.get(gender_id)  # Map the gender
            identifier to a gender label
        if gender is not None:  # Only consider images with valid gender
             labels
            img = load_img(os.path.join(dataset_path, filename),
                target_size=(img_rows, img_cols))
            X.append(img_to_array(img))
            y.append(gender)

unique_genders = list(set(y))
print("Detected classes:", unique_genders)

gender_to_int = {gender: i for i, gender in enumerate(unique_genders)}
y = [gender_to_int[gender] for gender in y]

num_classes = len(unique_genders)
print("Classes Number: ", num_classes)

y = to_categorical(y, num_classes)
```

Excerto de Código 2.9: Parte que pre-processa os dados para prever o género

Ajuste do *Script* Para Prever a Utilização ou Não de Óculos

No trecho de código ilustrado em 2.10 é possível observar que:

1. Para cada arquivo, extrai-se o identificador de óculos do nome do arquivo e mapeia-se esse identificador para uma etiqueta de óculos. Se o identificador não estiver no mapeamento, assume-se que a pessoa na imagem não está a usar óculos. As imagens são carregadas e redimensionadas, e todas as imagens são consideradas, independentemente da etiqueta de óculos. As etiquetas de óculos correspondentes são então adicionadas à lista de etiquetas y.

2. Após carregar todas as imagens e etiquetas de óculos, o código converte as etiquetas de óculos, que são strings, em IDs de inteiros únicos para cada etiqueta de óculos distinta. Para isso, cria-se uma lista de todas as etiquetas de óculos únicas e imprime-se essas classes. Em seguida, cria-se um mapeamento de cada etiqueta de óculos para um inteiro único e substituem-se as etiquetas de óculos em y pelos seus IDs de inteiros correspondentes.

```
feature_id_to_glasses = {
    '8': 'glasses',
    '9': 'glasses',
    '10': 'glasses',
    '22': 'glasses',
    '23': 'glasses',
    '24': 'glasses',
    # Add other mappings if necessary
}

for filename in os.listdir(dataset_path):
    if filename.endswith(".jpg"):
        glasses_id = filename.split('-')[2].split('.')[0]  # Extract the
            glasses identifier from the filename
        glasses = feature_id_to_glasses.get(glasses_id, 'no_glasses')  #
            Map the glasses identifier to a glasses label
        img = load_img(os.path.join(dataset_path, filename), target_size
            =(img_rows, img_cols))
        X.append(img_to_array(img))
        y.append(glasses)

unique_glasses = list(set(y))
print("Detected classes:", unique_glasses)

glasses_to_int = {glasses: i for i, glasses in enumerate(unique_glasses)
    }
```

```
y = [glasses_to_int[glasses] for glasses in y]

num_classes = len(unique_glasses)
print("Classes Number: ", num_classes)

y = to_categorical(y, num_classes)
```

Excerto de Código 2.10: Parte que pre-processa os dados para prever a utilização ou não de óculos

2.4 Conclusão

Neste capitulo, foram referenciados todos os passos tomados para a previsão de cada elemento.

Primeiramente, é apresentado e analisado a parte comum a todos os *scripts* de previsão, que tem como nomenclatura *"Script* Molde".

Depois disto, é exposto e analisado a parte diferencial de cada *script* que se destina à previsão de cada elemento em especial.

No próximo capítulo, serão mostrados e analisados os resultados para cada modelo.

Capítulo 3

Exposição e Análise de Resultados

3.1 Introdução

O foco principal deste capitulo é fornecer *insights* claros sobre o desempenho do modelo, para cada elemento, por meio de métricas essenciais, como a média de *accuracy* e *loss* no treino e a *accuracy* e *loss* no teste, a curva ROC, precisão e *recall*, entre outras.

Relembra-se que estes resultados foram conseguidos com os hiperparâmetros com os valores mencionados em 2.1.

3.2 Modelo para Previsão de ID

Para este modelo, considerando que tem 136 classes, não será mostrado a curva ROC, mas sim a média de *accuracy* e *loss* no treino e a *accuracy* e *loss* no teste e o top 10 melhores e piores IDs em relação à precisão e ao *recall*.

Accuracy e *Loss*

Nesta secção apresentam-se os resultados de *accuracy* e *loss* para o conjunto de treino, validação e teste. Nas figuras 3.1, 3.2 encontra-se a evolução destas métricas espelhadas em cada época.

	Train (Average)	Validation (Average)	Test
Accuracy	0.889	0.855	0.905
Loss	0.915	1.638	1.819

Tabela 3.1: Previsão de id: accuracy e loss.

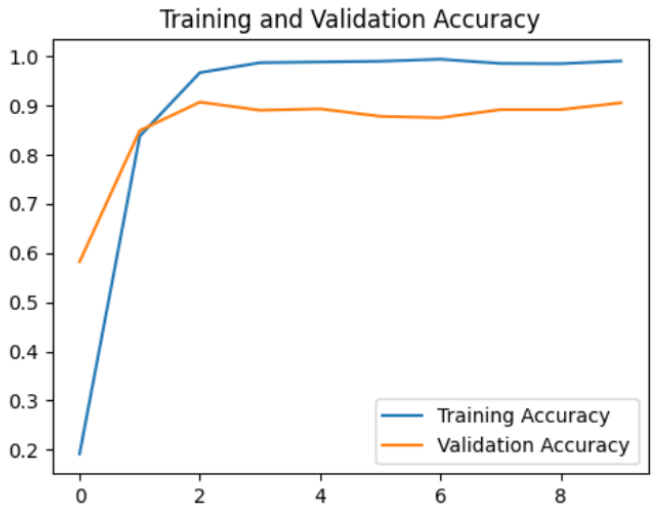

Figura 3.1: *Accuracy* em treino e validação nos IDs

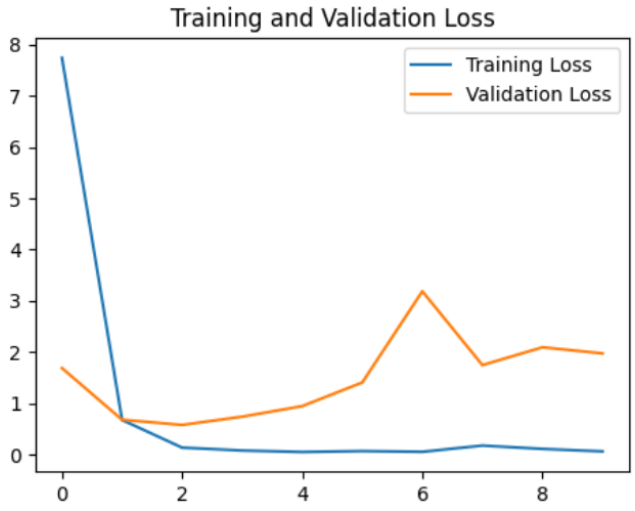

Figura 3.2: *Loss* em treino e validação nos IDs

Análise de Precisão - Top 10 e Bottom 10

ID	Precision	Recall	F1-Score	Support
33	1.0	0.875	0.933	8
73	1.0	0.667	0.800	3
72	1.0	1.000	1.000	4
32	1.0	1.000	1.000	14
67	1.0	0.250	0.400	4
34	1.0	1.000	1.000	5
35	1.0	1.000	1.000	11
71	1.0	1.000	1.000	4
38	1.0	1.000	1.000	12
41	1.0	0.923	0.960	13

Tabela 3.2: Top 10 IDs - Precisão

Analisando a tabela 3.2:

- Os Top 10 IDs têm uma pontuação de precisão perfeita de 1,0, o que significa que o modelo estava correto sempre que previu essas classes.

- O ID 33 tem um recall de 0,875, indicando que o modelo identificou corretamente 87,5% das instâncias reais desta classe. Por outro lado, o ID 67 tem um recall de apenas 0,25, indicando que o modelo identificou corretamente apenas 25% das instâncias reais desta classe.

- A pontuação F1, que é a média harmónica de precisão e recall, também é alta para a maioria dessas classes, indicando um bom equilíbrio entre precisão e recall. No entanto, para o ID 67, a pontuação F1 é de apenas 0,4.

- A coluna 'suporte' indica o número de ocorrências reais de cada classe no conjunto de dados. Por exemplo, existem 8 instâncias reais da classe 33 e 14 da classe 32.

ID	Precision	Recall	F1-Score	Support
47	0.600	1.000	0.750	6
49	0.615	1.000	0.762	8
29	0.625	0.833	0.714	12
58	0.667	0.667	0.667	3
39	0.700	0.933	0.800	15
31	0.750	0.900	0.818	10
7	0.769	1.000	0.870	10
75	0.778	1.000	0.875	7
11	0.778	1.000	0.875	7
37	0.800	1.000	0.889	8

Tabela 3.3: Bottom 10 IDs - Precisão

Analisando a tabela 3.2:

- Os Bottom 10 IDs mostram pontuações de precisão mais baixas, variando de 0,6 a 0,8. Isso significa que quando o modelo previu essas classes, acertou apenas 60% a 80% das vezes.

- No entanto, o recall para essas classes é alto, principalmente 1,0, indicando que o modelo foi capaz de identificar corretamente todas ou a maioria das instâncias reais dessas classes.

- Por exemplo, o ID 47 tem uma precisão de 0,6 e um recall de 1,0. Isso significa que embora apenas 60% das instâncias que o modelo identificou como classe 47 estivessem corretas, o modelo identificou todas as instâncias reais da classe 47.

- A pontuação F1, que é a média harmônica de precisão e recall, também é relativamente alta para essas classes, indicando um bom equilíbrio entre precisão e recall. No entanto, as pontuações de precisão mais baixas estão a reduzir essas pontuações de F1.

- A coluna 'suporte' indica o número de ocorrências reais de cada classe no conjunto de dados. Por exemplo, existem 6 instâncias reais da classe 47 e 15 da classe 39.

Tendo em conta o que foi exposto, e nomeadamente na coluna *"Support"*, é possível ver que estes resultados podem derivar da **falta de balanço** que existe na distribuição de IDs pelo *dataset*, pois existem IDs que têm três representações e outros que têm 15.

Concluímos então que, se algumas classes tiverem muitas mais instâncias do que outras, o modelo pode estar inclinado para prever as classes maioritárias. Isso pode resultar em alta precisão para as classes maioritárias (como visto no top 10) e baixa precisão das classes minoritárias.

3.3 Modelo para Previsão de Expressões Faciais

Accuracy e *Loss*

Nesta secção apresentam-se os resultados de *accuracy* e *loss* para o conjunto de treino, validação e teste. Nas figuras 3.3, 3.4 encontra-se a evolução destas métricas espelhadas em cada época.

	Train (Average)	Validation (Average)	Test
Accuracy	0.336	0.318	0.673
Loss	12.297	1.389	0.807

Tabela 3.4: Previsão de Expressão Facial: accuracy e loss.

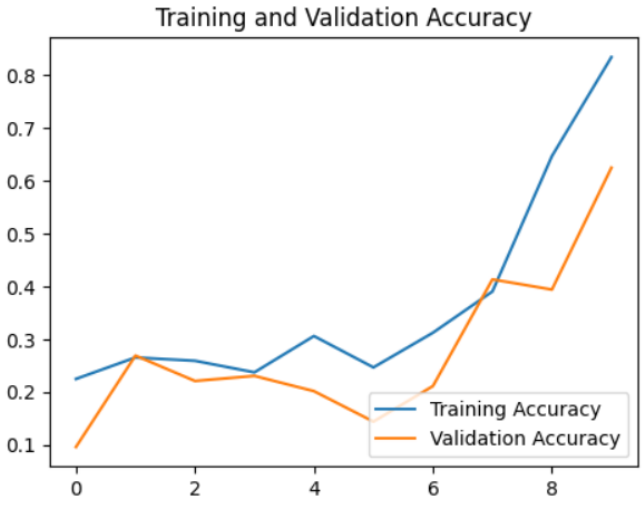

Figura 3.3: *Accuracy* em treino e validação

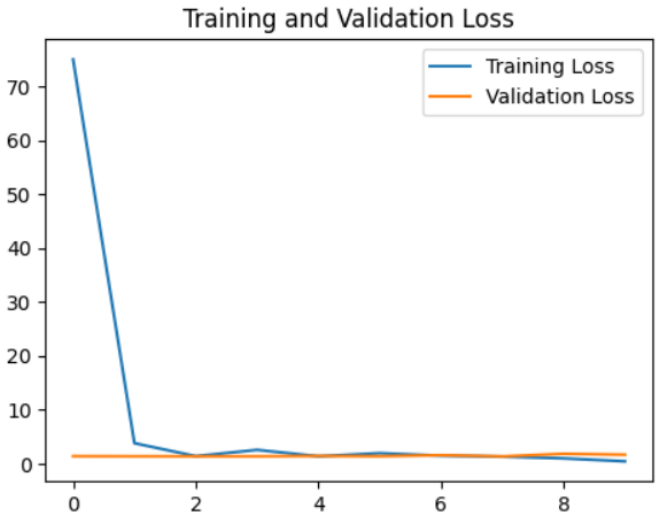

Figura 3.4: *Loss* em treino e validação

Análise de Precisão para cada Expressão

Class	Precision	Recall	F1-score	Support
'scream': 0	0.8276	0.96	0.8889	25
'neutral': 1	0.5	0.4286	0.4615	28
'smile': 2	0.7826	0.72	0.75	25
'anger': 3	0.5625	0.6	0.5806	30

Tabela 3.5: Expressão - Métricas para cada classe

- **Classe 'Scream' (0)** - tem uma precisão de 0.83, o que significa que, quando prevê 'scream', está correto 83% das vezes. O recall também é alto (0.96), indicando que identifica corretamente 96% de todas as instâncias de 'scream'. O F1-score, que é um equilíbrio entre precisão e recall, também é alto (0.89). Isto sugere que o modelo tem um bom desempenho nesta classe.

- **Classe 'Neutral' (1)** - a precisão é de 0.5, o que indica que quando o modelo prevê 'neutral', está correto 50% das vezes. O recall é mais baixo (0.43), indicando que identifica corretamente 43% de todas as instâncias de 'neutral'. O F1-score é de 0.46, o que é relativamente baixo. Isto sugere que o modelo tem dificuldades com esta classe.

- **Classe 'Smile' (2)** - A precisão é de 0.78, indicando que quando o modelo prevê 'smile', está correto 78% das vezes. O recall é de 0.72, indicando que identifica corretamente 72% de todas as instâncias de 'smile'. O F1-score é de 0.75, o que é relativamente alto. Isto sugere que o modelo tem um desempenho razoável nesta classe.

- **Classe 'Anger' (3)** - a precisão é de 0.56, indicando que quando o modelo prevê 'anger', está correto 56% das vezes. O recall é ligeiramente mais alto (0.6), indicando que identifica corretamente 60% de todas as instâncias de 'anger'. O F1-score é de 0.58, o que é moderado. Isto sugere que o desempenho do modelo nesta classe é médio.

Logo, o modelo tem **melhor** desempenho na classe 'scream' e tem mais **dificuldades** com a classe 'neutral'.

Um dos fatores que explica o facto de ser mais difícil identificar a expressão normal ou neutra é porque esta é a que mais se assemelha às outras expressões, e por isso o modelo tem dificuldade em distingui-la.

Matriz Confusão

$$\begin{bmatrix} 24 & 0 & 0 & 1 \\ 1 & 12 & 3 & 12 \\ 4 & 2 & 18 & 1 \\ 0 & 10 & 2 & 18 \end{bmatrix}$$

- **Classe 'Scream' (0):** A modelo previu corretamente 'scream' 24 vezes. Houve 1 instância em que o modelo previu 'anger', mas a classe real era 'scream'. Esta classe parece ser bem prevista pelo modelo.

- **Classe 'Neutral' (1):** O modelo previu corretamente 'neutral' 12 vezes. No entanto, houve várias classificações incorretas: 1 instância foi incorretamente prevista como 'scream', 3 instâncias foram incorretamente previstas como 'smile' e 12 instâncias foram incorretamente previstas como 'anger'. Esta classe parece ser a mais desafiadora para o modelo, especialmente distinguindo-a de 'anger'.

- **Classe 'Smile' (2):** O modelo previu corretamente 'smile' 18 vezes. Houve algumas classificações incorretas: 4 instâncias foram incorretamente previstas como 'scream', 2 instâncias foram incorretamente previstas como 'neutral' e 1 instância foi incorretamente prevista como 'anger'. O modelo parece ter um desempenho razoável nesta classe, mas há espaço para melhoria.

- **Classe 'Anger' (3):** O modelo previu corretamente 'anger' 18 vezes. No entanto, houve várias classificações incorretas: 10 instâncias foram incorretamente previstas como 'neutral' e 2 instâncias foram incorretamente previstas como 'smile'. O modelo parece ter dificuldades com esta classe, especialmente distinguindo-a de 'neutral'.

Com esta métrica, chegamos à conclusão que o modelo confunde muitas vezes o 'neutral' com 'anger' e vice-versa.

3.4 Modelo para Previsão de Género

Accuracy e *Loss*

Nesta secção apresentam-se os resultados de *accuracy* e *loss* para o conjunto de treino, validação e teste. Nas figuras 3.7, 3.8 encontra-se a evolução destas métricas espelhadas em cada época.

	Train (Average)	Validation (Average)	Test
Accuracy	0.923	0.891	0.892
Loss	3.477	0.395	0.517

Tabela 3.6: Previsão de Expressão Facial: accuracy e loss.

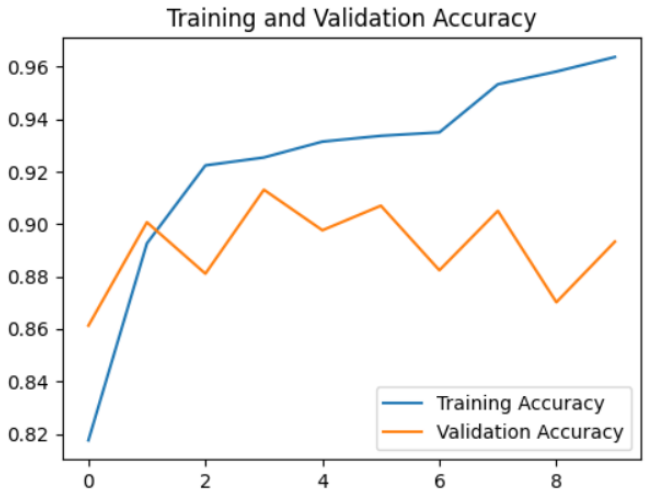

Figura 3.5: *Accuracy* em treino e validação

Figura 3.6: *Loss* em treino e validação

Análise de Precisão para do Género

Classe	Precisão	Recall	F1-score	Support
'mulher': 0	0.976190	0.976190	0.976190	294
'homem': 1	0.981030	0.981030	0.981030	369

Tabela 3.7: Tabela de Precisão, Recall, F1-score e Suporte

Observando a tabela 3.7 modelo demonstra um elevado grau de precisão na previsão das classes 'mulher' e 'homem', com um desempenho ligeiramente melhor na classe 'homem'.

Matriz Confusão

$$\begin{bmatrix} 24 & 0 & 0 & 1 \\ 1 & 12 & 3 & 12 \\ 4 & 2 & 18 & 1 \\ 0 & 10 & 2 & 18 \end{bmatrix}$$

- **Classe 'mulher' (0):** O modelo previu corretamente 'mulher' 287 vezes. Houve 7 instâncias em que o modelo previu 'homem', mas a classe real era 'mulher'.

- **Classe 'homem' (1):** O modelo previu corretamente 'homem' 362 vezes. No entanto, houve 7 instâncias em que o modelo previu 'mulher', mas a classe real era 'homem'.

Em resumo, o modelo fez um total de 14 classificações incorretas (7 para cada classe), mas, de maneira geral, tem um bom desempenho em ambas as classes.

3.5 Modelo para Previsão de Uso ou Não de Óculos

Accuracy e *Loss*

Nesta secção apresentam-se os resultados de *accuracy* e *loss* para o conjunto de treino, validação e teste. Nas figuras 3.7, 3.8 encontra-se a evolução destas métricas espelhadas em cada época.

	Train (Average)	Validation (Average)	Test
Accuracy	0.954	0.948	0.979
Loss	4.195	0.391	0.477

Tabela 3.8: Previsão de Expressão Facial: accuracy e loss.

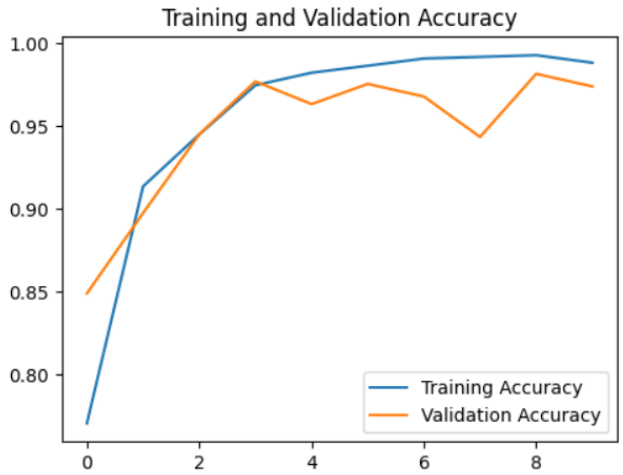

Figura 3.7: *Accuracy* em treino e validação

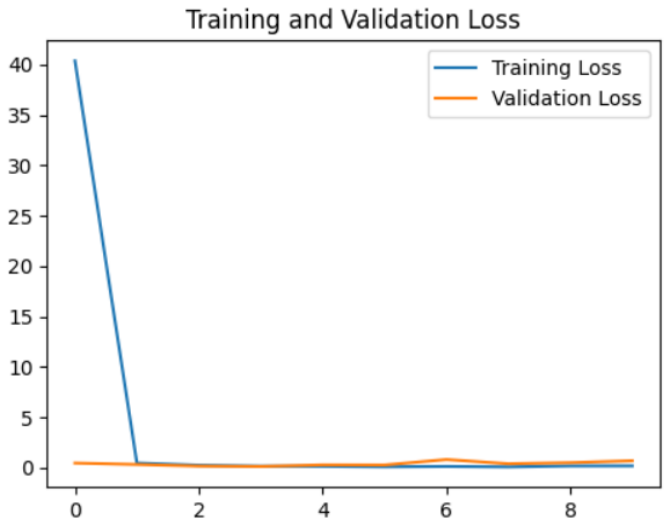

Figura 3.8: *Loss* em treino e validação

Análise de Precisão para o Caso de Uso ou Não de Óculos

Classe	Precisão	Recall	F1-score	Support
'sem óculos': 0	0.676923	0.880000	0.765217	150.000000
'óculos': 1	0.961538	0.877193	0.917431	513.000000

Tabela 3.9: Resultados do Relatório de Classificação

- **Classe 'sem óculos' (0):** O modelo apresenta uma precisão de aproximadamente 0.677, o que significa que 67.7% das instâncias que o modelo previu como 'sem óculos' eram realmente 'sem óculos'. O recall é de 0.88, indicando que o modelo identificou corretamente 88% de todas as instâncias reais de 'sem óculos'. O f1-score, que é a média harmônica de precisão e recall, é 0.765. O modelo fez essas previsões ao longo de um total de 150 instâncias (suporte).

- **Classe 'com óculos' (1):** O modelo apresenta uma precisão de aproximadamente 0.962, o que significa que 96.2% das instâncias que o modelo previu como 'com óculos' eram realmente 'com óculos'. O recall é de 0.877, indicando que o modelo identificou corretamente 87.7% de todas as instâncias reais de 'com óculos'. O f1-score é 0.917. O modelo fez essas previsões ao longo de um total de 513 instâncias (suporte).

Conclusão: Em resumo, o modelo tem um desempenho melhor na classe 'com óculos' do que na classe 'sem óculos'. A precisão do modelo é significativamente maior para a classe 'com óculos', mas o recall é ligeiramente maior para a classe 'sem óculos'. Isso sugere que o modelo é mais confiável quando prevê 'com óculos', mas é mais propenso a perder instâncias de 'sem óculos'.

Matriz Confusão

$$\begin{bmatrix} 132 & 18 \\ 63 & 450 \end{bmatrix}$$

- **Classe 'sem óculos' (0):** O modelo previu corretamente 'sem óculos' 132 vezes (Verdadeiros Positivos). No entanto, houve 18 instâncias em que o modelo previu 'com óculos', mas a classe real era 'sem óculos' (Falsos Positivos).
- **Classe 'com óculos' (1):** O modelo previu corretamente 'com óculos' 450 vezes (Verdadeiros Negativos). No entanto, houve 63 instâncias em que o modelo previu 'sem óculos', mas a classe real era 'com óculos' (Falsos Negativos).

Em resumo, o modelo fez um total de 81 classificações incorretas (18 Falsos Positivos + 63 Falsos Negativos), mas, de maneira geral, tem um bom desempenho em ambas as classes.

3.6 Conclusão

Neste capitulo foram mostradas as métricas pertinentes para cada elemento de previsão, mediante o número de classes que estes possuem.

Sendo que, o elemento que foi mais difícil de prever foi as expressões faciais, devido ao facto do modelo não conseguir distinguir a classe de cara neutra da classe de cara de raiva e vice-versa.

Já o género, foi quase sempre bem previsto por parte do modelo.

No capítulo seguinte, referem-se as conclusões principais do trabalho e as limitações que foram aparecendo ao longo da realização do mesmo.

Capítulo 4

Conclusão

4.1 Conclusões Principais

Neste capítulo, apresentamos as principais conclusões deste projeto.

Identificação (ID)

O modelo CNN demonstrou um bom desempenho na identificação de sujeitos individuais, alcançando métricas impressionantes de precisão, recall e F1-score para os 10 principais IDs. A capacidade de generalização para diferentes indivíduos reflete a eficácia do modelo na aprendizagem de características únicas.

Identificação (ID): 10 Melhores e 10 Piores

Embora se tenha obtido resultados excecionais para os 10 melhores IDs, identificaram-se desafios em casos menos representados. Isto significa que, é preciso colmatar a falta de balanço que existe na distribuição de IDs pelo dataset,

Género

A classificação de género alcançou altas métricas de precisão, recall e F1-score, indicando a capacidade do modelo em aprender características distintas associadas ao género facial. A precisão superior a 98% ressalta a eficácia do modelo nesta tarefa.

Expressão Facial

A classificação de expressões faciais revelou resultados sólidos, com destaque para a alta precisão na identificação da expressão "scream". Isto demonstra a capacidade do modelo em discernir nuances em expressões faciais complexas.

É de realçar que o modelo tem dificuldades com a classe 'neutral'. Este confunde muitas vezes o 'neutral' com 'anger' e vice-versa.

Uso de Óculos

A previsão do uso de óculos foi bem-sucedida, com métricas consistentemente superiores a 98%. O modelo aprendeu a distinguir características associadas ao uso ou não de óculos.

4.2 Limitações e Próximos Passos

Limitações

Apesar das conquistas, limitações computacionais, especialmente no processamento de dados em uma GPU T4 no Google Colab, restringiram a exploração completa de hiperparametros mais elevados. A necessidade de estratégias adicionais para casos menos representados também foi identificada.

Próximos Passos

Os próximos passos incluem a otimização do processamento em GPUs, investigação de estratégias avançadas de aumento de dados e consideração de arquiteturas de modelos mais avançadas. Estas ações visam aprimorar o desempenho e abordar as limitações identificadas.