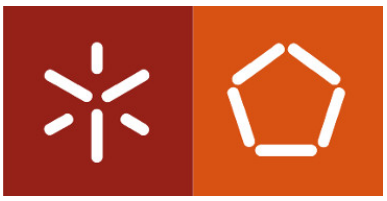
Universidade do Minho

Escola de Engenharia



Processamento de Imagem e Visão por Computador

Mestrado em Engenharia Informática

Trabalho de Grupo

Diogo Vieira - [PG50518]
Laura Rodrigues - [PG50542]
Mariana Marques - [PG50633]

Novembro, 2023

Conteúdo

1. Introdução

Para a realização deste projeto, foi proposta a exploração de modelos de *Deep Learning* aplicados a um conjunto de dados, composto por imagens de sinais de trânsito, com o intuito de obter o melhor resultado possível em termos de *accuracy*.

Por sua vez, o projeto está dividido em duas fases:

- **Data Augmentation** - Nesta primeira fase, é realizado o treino de um conjunto de dados através da estratégia *Data Augmentation*. Desta forma, é possível explorar os filtros e métodos de processamento de imagem escolhidos e avaliar seu impacto no desempenho final da rede.
- **Redes Ensemble** - Nesta segunda fase, são utilizadas as redes resultantes da fase anterior para construir a rede *Ensemble*.

2. Preparação do Conjunto de Dados

De facto, a preparação do conjunto de dados em questão é essencial e envolve a execução de diversas tarefas, tais como:

- **Carregamento dos Dados** - Para ser possível ter acesso aos dados, foi necessário realizar o carregamento dos mesmos. Como o trabalho foi realizado com o auxílio da ferramenta Google Collab. O carregamento dos dados é feito da seguinte forma:

Listing 2.1: Carregamento de Dados

```
from google.colab import drive drive.mount('/content/drive')

path = "/content/drive/MyDrive/data.zip"

def unzip_file(path):
  from zipfile import ZipFile
  file_name = path

  with ZipFile(file_name, 'r') as zip:
    zip.extractall()
    print('Done')

unzip_file(path)
```

- **Balanceamento dos Dados** - Ao analisar o número de elementos que cada classe tinha, verificamos uma grande diferença neste número. Para tentar combater este desbalanceamento, optamos por balancear os dados. A maneira como este balanceamento foi feito passa por, calcular a média dos elementos das classes. Para as classes que tiverem menos elementos do que essa média, gera-se imagens através das existentes com uma ligeira data augmentation. Deste modo o dataset fica mais balanceado, de forma a melhorar o desempenho do modelo e a evitar *overfitting*.

Listing 2.2: Balanceamento de Dados

```
classes = os.listdir(f'{data_path}/train_balanced_{IMAGES_PER_CLASS}')
list_img = []
for cla in classes:
    list_img =
        os.listdir(f'{data_path}/train_balanced_{IMAGES_PER_CLASS}/{cla}')
    random.shuffle(list_img)
    for k in range(len(list_img), IMAGES_PER_CLASS):
        filename =
            f'{data_path}/train_balanced_{IMAGES_PER_CLASS}/{cla}/{list_img[(k
            - len(list_img)) % len(list_img)]}'
        if not filename.endswith('png'):
```

```
        continue
    im = Image.open(filename)
    im = some_data_augmentation(im)
    im.save(f'{data_path}/train_balanced_{IMAGES_PER_CLASS}/{cla}/_{k}.png')
```

- **Funções Auxiliares** - Foram recolhidas as funções auxiliares disponibilizadas, devido à sua utilidade, tais como:

  1. get_label(file_path)
  2. decode_img(img)
  3. get_bytes_and_label(file_path)
  4. show_data(s1,l1, s2,l2, labels, min)
  5. show_batch(image_batch, label_batch)
  6. show_history(history)
  7. show_accuracies(labels, test, val)
  8. show_misclassified(predictions, ground_truth, images, num_rows = 5, num_cols=3)
  9. plot_image(i, predictions_array, true_label, img)
  10. plot_value_array(i, predictions_array, true_label)
  11. show_confusion_matrix(mat, classes)
  12. def split_train_val(train_offset,dataset):

- **Callbacks** - Foi recolhida a função *prepare_callbacks(file_path)* disponibilizada, visto que é responsável por interromper ou reduzir a taxa de aprendizagem quando um modelo não demonstra melhoria.

Listing 2.3: CallBack prepare_callbacks(file_path)

```
def prepare_callbacks(file_path):

checkpointer = ModelCheckpoint(filepath= file_path,
                           monitor = 'val_accuracy',
                           verbose=1,
                           save_weights_only=True,
                           save_best_only=True)


earlyStopper = EarlyStopping(monitor='val_loss', min_delta = 0.0001,
    patience = 10, verbose = 1)

reduceLR = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
    patience=5, min_lr=0.000000001, verbose = 1)

return [checkpointer, earlyStopper, reduceLR]
```

- **Tamanho da Batch** - O tamanho da *batch* refere-se ao número de amostras que são executadas na rede em apenas uma iteração. Desta forma, define-se, por omissão, 32 o tamanho de cada *batch*.

Listing 2.4: Tamanho da Batch

```
BATCH_SIZE = 32
IMAGE_SIZE = 32
```

- **Carregamento das Imagens** - Por fim, é feito o carregamento das imagens, começando por obter o nome das classes, normalizando o conjunto de dados de treino e, por opção, visualização da informação da imagem e do conjunto de dados. Estas imagens foram convertidas do formato ppm para png de modo a facilitar o seu carregamento.

3. Processo de Treino de Modelos

Para ser possível treinar um modelo que requer a utilização de uma rede neural, é necessário executar as seguintes etapas:

1. **Construção de uma Rede Neuronal** - É desenvolvida a estrutura da rede neuronal, optando-se pela arquitetura CNN (Convolutional Neural Network), amplamente utilizada para processamento de imagens e reconhecimento visual. A implementação física da rede ocorre através da função *model_III* que possui várias camadas convolucionais responsáveis pela extração das características relevantes das imagens - *Conv2D*, cada uma seguida pela normalização em lote para estabilizar o processo de treino *BatchNormalization* e uma função de ativação *LeakyReLU* para introduzir não-linearidade.

   Camadas de *pooling* responsáveis pela redução da dimensão dessas características - *MaxPooling2D*. As camadas seguintes continuam a aprofundar a compreensão da rede incorporando camadas convolucionais adicionais com tamanhos de filtro crescentes. A saída é então achatada,*Flatten*, e passada por uma cada densa de 128 neurónios, *Dense*, assim como uma camada de *Dropout* para evitar overfitting. A camada final utiliza uma função de ativação *softmax* para produzir probabilidades de classe para classificação de multi classes. O modelo é treinado usando o otimizador Adam com uma taxa de aprendizado de 0,0001, otimizando a perda de entropia cruzada categórica.

   A avaliação é realizada com base na precisão, e toda a arquitetura é projetada para melhorar a aprendizagem de características e o desempenho de classificação em tarefas de análise de imagens.

Listing 3.1: Contrução de uma rede CNN

```
def model_III(classCount, imgSize, channels):

model = Sequential()

model.add(Conv2D(128, (5, 5),
                 input_shape=(imgSize, imgSize, channels)
                 ))
model.add(BatchNormalization())
model.add(LeakyReLU(alpha=0.01))

model.add(Conv2D(128, (5, 5) ))
model.add(BatchNormalization())
model.add(LeakyReLU(alpha=0.01))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (5, 5) ) )
model.add(BatchNormalization())
model.add(LeakyReLU(alpha=0.01))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
```

```
model.add(Dense(128))
model.add(LeakyReLU(alpha=0.01))
model.add(Dropout(0.2))

model.add(Dense(classCount, activation='softmax'))


opt = Adam(lr=0.0001)
model.compile(optimizer = opt, loss='categorical_crossentropy',
    metrics=[ 'accuracy'])
return model
```

2. **Treino do Modelo** - De seguida, é necessário treinar o modelo na rede construída, utilizando o conjunto de dados de treino. Optámos por passar 50 epochs, sendo que o modelo poderá sempre parar antes através de *early stopping*. É também utilizado um validation set, este validation set é resultante da divisão de 20% do dataset de treino original, antes de algum filtro dinâmico ser aplicado.

Listing 3.2: Treino do Modelo

```
callbacks = prepare_callbacks(file_path)

history = model.fit(dataSolo,
epochs = 50,
validation_data = valset,
callbacks = callbacks)
```

3. **Validação do Modelo** - Para a validação do modelo, é utilizado o conjunto de dados de teste.

Listing 3.3: Validação do Modelo

```
model.load_weights(file_path)

eval = model.evaluate(testset, verbose=2)
val = model.evaluate(valset, verbose=2)
train = model.evaluate(dataSolo, verbose=2)
```

4. Apresentação dos Modelos

Neste projeto, a criação dos modelos seguiu um processo sequencial, em busca de aprimoramento contínuo.

Inicialmente, foi desenvolvido um modelo responsável por treinar os dados de uma forma simples, visando uma primeira análise do desempenho da rede neuronal. Este modelo treina a rede neuronal com o conjunto de dados de treino e avalia com o respetivo conjunto de dados de teste.

Após uma minuciosa análise, identificou-se margem para aprimorar os resultados alcançados durante o treino do *Modelo sem filtros*. Desta forma, foi criado um segundo modelo com um treino de dados que recorre à estratégia *Data Augmentation* e à aplicação de filtros de forma a melhorar os resultados obtidos.

4.1 Modelo sem filtros

Este modelo não exibe nada de extraordinário, visto que apenas implementa as etapas definidas no capítulo 3.

4.2 Modelo com filtros

De modo a tentar tornar o dataset de treino mais representativo, iremos aplicar vários filtros de data augmentation às imagens existentes, para assim criar modelos mais preparados para as previsões.

4.2.1 Data Augmentation

Filtros Aplicados

Em relação aos filtros utilizados para o processamento de imagens, foram aplicados um conjunto de filtros com aplicações em áreas distintas, nomeadamente:

- **Cor** - Filtros responsáveis por modificar aspetos da cor, ou seja:
    1. Brigthness
    2. Contrast
    3. Hue
    4. Saturation

- **Transformações Geométricas** - Filtros responsáveis por modificar a posição da imagem, ou seja:
    1. Rotate
    2. Shear
    3. Translate
    4. Crop
- **Contornos** - Filtro responsável por salientar os contornos da imagem, ou seja:
    1. Edges

Listing 4.1: Filtro Arestas

```
def process_edges(image, label):
    expanded_image = tf.expand_dims(image, axis=0)
    sobel = tf.image.sobel_edges(expanded_image)
    gradient_magnitude = tf.sqrt(tf.square(sobel[..., 0]) +
        tf.square(sobel[..., 1]))
    clipped_img = tf.clip_by_value(gradient_magnitude / 8, 0, 1)
    img = tf.squeeze(clipped_img, axis=0)
    return img, label
```

Figura 4.1: Filtro num sinal da classe 1

Houve também várias tentativas de implementar os filtros Fourier, Gray Scale e Perlin Noise, no entanto, sem sucesso.

4.2.2 Modelos

Foram treinados vários modelos utilizando diferentes técnicas de *data augmentation*. Inicialmente, um modelo foi treinado para cada tipo de filtro. Além disso, foi criado um que agrupa os filtros em categorias, como cores e posições.

Esses modelos foram treinados tanto com os dados de treino originais como com os dados de treino normalizados.

Os resultados dos testes e análise correspondente serão apresentados posteriormente.

5. Ensemble

Um ensemble, refere-se à combinação de múltiplos modelos com o objetivo de melhorar a precisão e o desempenho geral do sistema. A ideia por trás do ensemble é que a combinação de várias previsões independentes pode produzir resultados mais confiáveis e robustos do que um único modelo.

Existem diferentes técnicas de ensemble, algumas delas são voting ensemble, averaging ensemble e o stacking ensemble.

Primeiramente, optou-se por fazer *Voting Ensemble*.

A ideia fundamental por trás do *voting ensemble* é utilizar diversos modelos, cada um treinado com conjuntos de dados distintos. Cada modelo gera uma previsão para uma entrada específica e essas previsões são combinadas para alcançar uma decisão final. Esta abordagem permite obter previsões de múltiplos modelos, em vez de depender exclusivamente de um único. Dessa forma, podemos aproveitar a diversidade e complementaridade dos modelos para melhorar a precisão e o desempenho geral do modelo.

Após realizar os testes anteriores a esta parte percebeu-se que, quando o dataset era balanceado, se obtia melhores resultados. Optámos, assim, por testar a implementação do ensemble apenas para este caso.

Para a escolha dos modelos no ensemble, testámos dois conjuntos:

- Conjunto com os filtros combinados (3 modelos)
- Conjunto com os filtros sozinhos (9 modelos)

Do primeiro conjunto, fazem parte os modelos Colors_Concat, que agrupa todos os filtros relativos à cor, Geo_Concat, que agrupa todos os filtros relativos a geometria e Sobel, que agrupa as imagens geradas pelo filtro sobel com as imagens de treino sem filtro.

Do segundo fazem parte os modelos gerados com imagens de apenas um filtro.

Os resultados destes testes e a sua análise serão apresentados posteriormente.

Numa tentativa de explorar outras técnicas de ensemble, tentámos fazer stacking ensemble, mas sem sucesso.

6. Testes Realizados

6.1 1ª Parte

Nas tabelas abaixo, Colors_Concat e Geo_Concat representam, respetivamente, a junção de todos os filtros de transformações de côr e a junção de todos os filtros de transformações geométricas.

Relativamente à performance dos vários modelos usando o dataset não balanceado, obtivemos os seguintes valores:

Modelo	loss	accuracy
No filters	0.1648	0.9839
Crop	0.1796	0.9739
Contrast	0.1376	0.9864
Brightness	0.2021	0.9804
Hue	0.1949	0.9763
Translate	0.1059	0.9813
Rotate	0.1411	0.9793
Saturation	0.1666	0.9804
Shear	0.0947	0.9850
Sobel	0.2133	0.9780
Colors_Concat	0.2700	0.9836
Geo_Concat	0.1893	0.9726
All	0.0755	0.9908

Tabela 6.1: Dataset original e data augmentation

Quanto aos testes após proceder ao balanceamento do dataset, apresenta-se:

Modelo	loss	accuracy
No filters	0.0931	0.9836
Crop	0.0990	0.9807
Contrast	0.0925	0.9868
Brightness	0.1632	0.9820
Hue	0.0781	0.9874
Translate	0.1064	0.9838
Rotate	0.0799	0.9875
Saturation	0.0802	0.9859
Shear	0.0727	0.9866
Sobel	0.0979	0.9884
Colors_Concat	0.1181	0.9876
Geo_Concat	0.0822	0.9909
All	0.1138	0.9892

Tabela 6.2: Dataset balanceado e data augmentation

6.1.1 Análise de Resultados

Como podemos observar, no geral o modelo balanceado deu melhores resultados do que o não balanceado.

No modelo 1, destaca-se como melhor resultado a combinação de todos os filtros.

No modelo 2, os que apresentaram melhores resultados foram os 4 últimos testes, correspondentes à combinação de mais do que um filtro. Destaca-se como melhor resultado a combinação das transformações geométricas.

Em ambos os modelos destaca-se que quando aplicado apenas o filtro Shear o erro torna-se bastante reduzido.

6.2 2ª Parte

Como já explicado anteriormente, optámos por, nesta fase, utilizar apenas os modelos do dataset balanceado.

Para o ensemble foram feitos testes com dois conjuntos.

Para o primeiro conjunto, composto por 3 modelos:

- Geo_Concat
- Colors_Concat
- Sobel

A accuracy média deste modelos é de **98.894**. Sendo a pior, **98.76**, e a melhor **99.09**.

Foram calculadas várias estatísticas, para além da accuracy final, de modo a perceber melhor o funcionamento do ensemble.

Estas estatísticas são:

- Nr de Imagens (num_images)- número de imagens de teste
- Todas Corretas (all_correct) - número de imagens em que todos os modelos concordam e está correto
- Todos Incorretas (all_incorrect)- número de imagens em que todos os modelos concordam e está incorreto
- Maioria Correta (maj_vote)- número de imagens em que a maioria está correta
- Empate (tie) - Número de imagens em que existem empates relativamente à escolha da classe
- Maioria Incorreta (maj_wrong)- número de imagens em que a maioria está incorreta
- Imagens Corretas (log_ok)- número de imagens corretamente classificadas
- Imagens Incorretas (log_ko) - número de imagens incorretamente classificadas

num_images	all_correct	all_incorrect	maj_vote	tie	maj_wrong	log_ok	log_ko
12630	12330	11	209	41	39	12573	57

Tabela 6.3: Resultados Ensemble 1

A accuracy total é **0.9955**.

Para o segundo conjunto, composto por 9 modelos:

- Brigthness

- Contrast
- Hue
- Saturation
- Rotate
- Shear
- Translate
- Crop
- Sobel

A accuracy média deste modelos é de **98.55**. Sendo a pior, **98.07**, e a melhor **98.84**.

Foram obtidos os seguintes resultados:

num_images	all_correct	all_incorrect	maj_vote	tie	maj_wrong	log_ok	log_ko
12630	12023	4	523	20	60	12561	69

Tabela 6.4: Resultados Ensemble 2

A accuracy total é **0.9945**.

6.2.1 Análise de resultados

Pela accuracy total de ambos os ensembles verificamos que o primeiro conjunto tem uma accuracy superior. No entanto, para além desta estatística, existem também outros dados importantes a salientar: em termos de imagens em que todos os modelos concordam e estão corretas, existe uma diferença de mais de 300 imagens a favor do primeiro conjunto. Por outro lado, o número de imagens em que todos concordam e estão incorretas, é menor no segundo conjunto, o que seria de esperar pois, uma vez que é menor número de modelos presentes neste conjunto, é mais fácil existir pelo menos uma rede que acerte. É também, no entanto, mais complicado estarem todas de acordo.

Se compararmos os resultados do Ensemble com os resultados tanto dos modelos individualmente, como da sua média, verificamos que houve um aumento significativo, este aumento seria de esperar em relação à média. No entanto, também acaba por superar os modelos com melhores resultados do conjunto.

7. Conclusão

Em conclusão, este relatório apresentou uma investigação sobre a eficácia de algumas técnicas de aumento de dados para treinar o conjunto de dados de sinais de trânsito alemão. O *German Traffic Sign Dataset* é um conjunto de dados de referência amplamente utilizado para tarefas de reconhecimento de sinais de trânsito. Aplicando vários métodos de aumento de dados como rotação, translação, ajuste de brilho, etc., procurámos aumentar a diversidade e a robustez do conjunto de dados.

Os nossos resultados demonstraram que o aumento de dados melhorou o desempenho dos modelos de aprendizagem profunda, assim como o Ensemble.

Gostaríamos, no entanto, de ter conseguido concretizar as implementações de Stacking Ensemble e dos filtros de Fourier, Gray Scale e Perlin Noise. Ficam assim estes tópicos para trabalho futuro de modo a tentar obter melhores resultados.

Em conclusão, este artigo ressalta a importância do aumento de dados como uma técnica valiosa para treinar modelos de aprendizagem profunda no conjunto de dados de sinais de trânsito alemão. A utilização de *data augmentation* melhora a capacidade de generalização dos modelos, levando a um melhor desempenho em tarefas de reconhecimento de sinais de trânsito. Os resultados deste tipo de estudo poderão contribuir para avanços no campo de sistemas de transporte inteligentes e abrir caminho para futuras pesquisas nesta área.