Visão por Computador
e
Processamento de Imagem

MEI - 1º ano - 2º semestre
Universidade do Minho

Deep Learning

Trabalho Prático 2023

Grupo 11

Trabalho realizado por: *Número*

Duarte Moreira PG50349
Pedro Tavares PG50698
Tiago Costa PG50777

Braga, 15 de junho de 2023

Índice

1. Introdução

O presente relatório serve para descrever as decisões tomadas pelo grupo ao longo da realização deste trabalho e para a demonstração dos respetivos resultados. Este foi desenvolvido no âmbito da disciplina de Visão por Computador e Processamento de Imagem e tem como objetivo explorar modelos de *Deep Learning* aplicados ao *dataset* alemão de sinais de trânsito GTSRB. O foco é obter o melhor resultado de *accuracy* possível no *dataset* de teste, sendo que o recorde publicado até à data é de 99.81%.

Assim sendo, numa primeira fase serão apresentados os diferentes tipos de modelos construídos. Posteriormente, o foco será em todos os métodos de processamento de imagem aplicados, desde alterações na cor a transformações geométricas, para a realização de *data augmentation*. Depois destas etapas será avaliado o impacto que estas alterações tiveram ou não no desempenho final da rede.

Já numa segunda fase, serão explorados os *ensembles* de redes com o objetivo de "agrupar"os modelos obtidos anteriormente e obter resultados mais promissores. Os tipos de *ensembles* utilizados são: *concatnate*, *average* e *weighted average*.

Para finalizar será realizada uma discussão e conclusão do trabalho realizado e dos resultados obtidos.

2. Modelos

Como já mencionado, este capítulo apresenta os diferentes tipos de modelos utilizados ao longo da execução do trabalho. Inicialmente, trabalhamos apenas com os dois primeiros modelos, com os quais já havíamos trabalhado nas aulas práticas, para testar as transformações mais simples como iremos ver. No entanto, após alguns testes decidimos desenvolver um modelo com uma rede mais complexa de forma a aproximá-la da complexidade do *dataset*.

2.1 Modelo 1

Como podemos observar pelo código em baixo, o nosso primeiro modelo constitui uma rede neuronal convolucional, que tem início numa camada de *input* que recebe imagens de tamanho *imgSize*. Para além disso contém três *layers* convolucionais com parâmetros distintos, cada uma delas seguida de uma *batch normalization* e uma ativação do tipo *LeakyReLU*. O modelo é compilado usando o *Adam optimizer* e com um *learning rate* inicial de 0.0001. A função de *loss* utilizada é a *categorical cross-entropy* uma vez que é a mais adequada para problemas de classificação com várias classes. Para avaliar o desempenho do modelo durante a fase de treino é usada a métrica de *accuracy*.

```
def model_I(classCount, imgSize, channels):
    model = Sequential()
    # Input Layer
    model.add(Input(shape=(imgSize, imgSize, channels)))
    # Convolutional Layer 1
    model.add(Conv2D(64, (5, 5)))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.01))
    # Convolutional Layer 2
    model.add(Conv2D(64, (5, 5)))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.01))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    # Convolutional Layer 3
    model.add(Conv2D(64, (5, 5)))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.01))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    # Flatten to a 1D vector
    model.add(Flatten())
    model.add(Dense(128))
    model.add(LeakyReLU(alpha=0.01))
    model.add(Dropout(0.2))
    # Output Layer
    model.add(Dense(classCount, activation='softmax'))
    # Compilation
    opt = Adam(learning_rate=0.0001)
    model.compile(opt, 'categorical_crossentropy', metrics=['accuracy'])
    return model
```

2.2 Modelo 2

Já em relação ao segundo modelo utilizado, este é quase semelhante ao primeiro. Aqui a *input layer* é especificada na primeira *convolutional layer* e é utilizado um maior número de filtros ao longos das camadas. O código para esta rede apresenta-se a seguir.

```
def model_II(classCount, imgSize, channels):
    model = Sequential()
    # Input Layer/Convolutional Layer 1
    model.add(Conv2D(128, (5, 5), input_shape=(imgSize, imgSize, channels)))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.01))
    # Convolutional Layer 2
    model.add(Conv2D(128, (5, 5)))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.01))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    # Convolutional Layer 3
    model.add(Conv2D(256, (5, 5)))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.01))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    # Flatten to a 1D vector
    model.add(Flatten())
    model.add(Dense(128))
    model.add(LeakyReLU(alpha=0.01))
    model.add(Dropout(0.2))
    # Output Layer
    model.add(Dense(classCount, activation='softmax'))
    # Compilation
    opt = Adam(learning_rate=0.0001)
    model.compile(opt, 'categorical_crossentropy', metrics=['accuracy'])
    return model
```

2.3 Modelo 3

O último modelo desenvolvido, que foi o que apresentou melhores resultados como iremos demonstrar, difere dos restantes no sentido em que são alterados o número de filtros nas respetivas camadas para que este obtenha uma maior capacidade de aprender *features* complexas e são adicionadas camadas de *dropout* com uma frequência maior para evitar *overfitting*. O modelo 3 é definido da seguinte maneira.

```
def model_III(classCount, imgSize, channels):
    model = Sequential()
    # Input Layer/Convolutional Layer 1
    model.add(Conv2D(100, (5, 5), input_shape=(imgSize, imgSize, channels)))
    model.add(LeakyReLU(alpha=0.01))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    # Convolutional Layer 2
    model.add(Conv2D(150, (5, 5) ))
    model.add(LeakyReLU(alpha=0.01))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    # Convolutional Layer 3
    model.add(Conv2D(250, (5, 5) ))
    model.add(LeakyReLU(alpha=0.01))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
```

```
    # Flatten to a 1D vector
    model.add(Flatten())
    model.add(Dense(350))
    model.add(LeakyReLU(alpha=0.01))
    model.add(Dropout(0.2))
    # Output Layer
    model.add(Dense(classCount, activation='softmax'))
    # Compilation
    opt = Adam(learning_rate=0.0001)
    model.compile(opt, 'categorical_crossentropy', metrics=['accuracy'])
    return model
```

2.4 Comparação de modelos

Na seguinte tabela estão apresentados os valores de *accuracy* para cada um dos diferentes modelos aplicados ao *dataset* incial, ou seja, sem modificação alguma. É de notar a diferença de resultados entre o modelo 3 e os restantes, que se deve muito provavelmente, à complexidade do mesmo. Assim os testes iniciais são realizados com os modelos 1 e 2, e numa fase mais avançada é utilizado o modelo 3.

Modelo	I	II	III
Accuracy	97.41%	98.29%	94.64%

3. Pré-processamento

Na fase de pré-processamento das imagens do *dataset* são realizadas duas operações de extrema importância, *crop* e alteração do contraste.

3.1 Crop

A primeira alteração feita é então o *crop* de todas as imagens dos *train, validation* e *test sets* pela respetiva região de interesse (ROI). Depois deste processo as imagens são *resized* para 32x32 *pixels*. Numa primeira fase o grupo tentou realizar este *resize* para 48x48 que mostrava atingir resultados promissores, no entanto, a falta de recursos computacionais não nos permitiu explorar demasiado esta abordagem, devido principalmente ao uso de *concatenates* que aumentava de forma significativa o tamanho do *dataset*.

Com o código a seguir apresentado, demonstramos como o grupo trata do processo de *crop* em todas as imagens e, para além disso, como fazemos a conversão das imagens de *ppm* para *png*.

```
def ppm2png(path):
    df = None
    type = path.split("\\")

    if (type[-1] == "test"):
        df = pd.read_csv(path + "/GT-final_test.test.csv", ";")
        df.set_index("Filename", inplace=True)

    for fpdir in glob.iglob(path +"/*/"):
        if(type[-1] == "train"):
            df = pd.read_csv(fpdir + "GT-" + fpdir.split("\\")[-2] + ".csv",";")
            df.set_index("Filename", inplace=True)

        for filepath in glob.iglob(fpdir + "/*.ppm"):
            im = Image.open(filepath)
            row = df.loc[filepath.split("\\")[-1]]

            if not row.empty:
                im = im.crop((row[2],row[3],row[4],row[5]))

            filepath2 = os.path.splitext(filepath)[0] + ".png"
            im.save(filepath2)
            os.remove(filepath)
```

3.2 Contraste

De forma a dar mais ênfase aos detalhes de cada imagem e melhorar a aparência geral da mesma, aplicamos uma variante do método de equalização de histograma denominada **CLAHE** (Contrast Limited Adaptive Histogram Equalization), que trabalha certas regiões da imagem. O código para esta transformação apresenta-se a seguir.

```python
def call_change_contrast(img):
    with tf.device("/gpu:0"):
      lab= cv2.cvtColor(img.numpy(), cv2.COLOR_RGB2LAB)
      l_channel, a, b = cv2.split(lab)

      # Applying CLAHE to L-channel
      # feel free to try different values for the limit and grid size:
      clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
      cl = clahe.apply(l_channel)

      # merge the CLAHE enhanced L-channel with the a and b channel
      limg = cv2.merge((cl,a,b))

      # Converting image from LAB Color model to BGR color spcae
      enhanced_img = cv2.cvtColor(limg, cv2.COLOR_LAB2RGB)
      enhanced_img = tf.convert_to_tensor(enhanced_img, dtype=tf.uint8)
    return enhanced_img
```

Em relação aos resultados provocados por esta alteração são os que se apresentam na tabela a seguir. Como podemos observar os resultados sofrem um ligeiro aumento mas nada significativo.

Modelo	I	II	III
Sem contraste	97.41%	98.29%	94.64%
Com contraste	97.75%	98.31%	94.17%

4. Data Augmentation

De forma a aumentar a diversidade e robustez do *dataset* é realizado um processo de *data augmentation*, tanto de forma dinâmica ou não. Este processo torna-se de dificuldade acrescida devido ao tamanho do *dataset* e aos limitados recursos computacionais a que o grupo tinha acesso. O resto deste capítulo apresentará as diferentes transformações aplicadas às imagens.

4.1 Color Transformations

Uma técnica de *data augmentation* muitas vez utilizada nestes problemas é a *color jittering* que permite a variação das componentes de uma imagem como: *hue*, *saturation* e *brightness*. Para além do *contrast* que é aplicado numa fase de pré-processamento, estas três são aplicadas dinamicamente, ou seja, a cada *epoch* realizada na fase de treino estas transformações são aplicadas ao *dataset*. Os resultados obtidos com estas transformações apresentam-se na tabela seguinte. Como podemos observar as diferenças nos valores de *accuracy* são mínimas exceto para o modelo III que parece ter conseguido aprender significativamente com estas transformações, apresentando um aumento de cerca de 3%.

Modelo	I	II	III
Sem *color transf.*	97.75%	98.31%	94.17%
Com *color transf.*	97.55%	97.62%	97.43%

4.1.1 Hue

Para a variação da componente *hue* é dado um intervalo entre [0,0.3], da qual é escolhido um valor aleatório. A seguir apresenta-se alguns exemplos desta transformação.

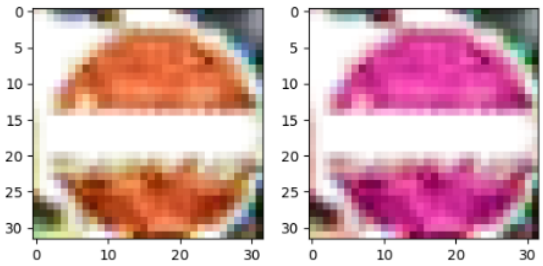

Figura 4.1: Transformação *hue* 1.

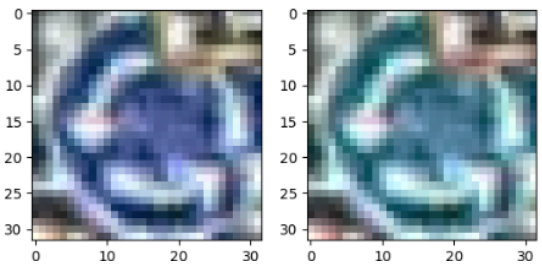

Figura 4.2: Transformação *hue* 2.

4.1.2 Saturation

Para a variação da componente *saturation* é dado um intervalo entre [0.4,2], da qual é escolhido um valor aleatório. A seguir apresenta-se alguns exemplos desta transformação.

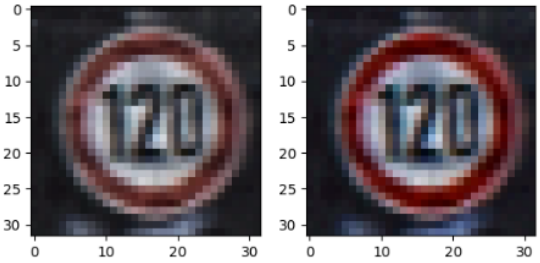

Figura 4.3: Transformação *saturation* 1.

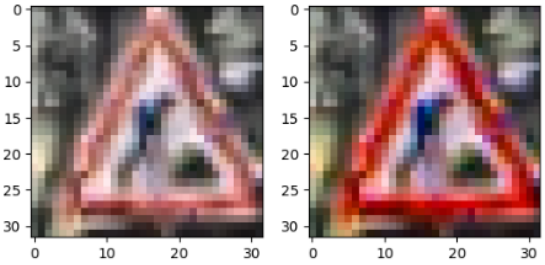

Figura 4.4: Transformação *saturation* 2.

4.1.3 Brightness

Para a variação da componente *brightness* é dado um intervalo entre [-0.8,0.8], da qual é escolhido um valor aleatório. A seguir apresenta-se alguns exemplos desta transformação.

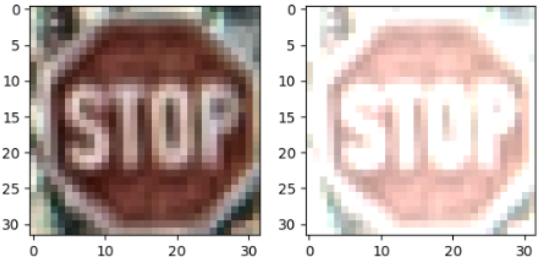

Figura 4.5: Transformação *brightness* 1.

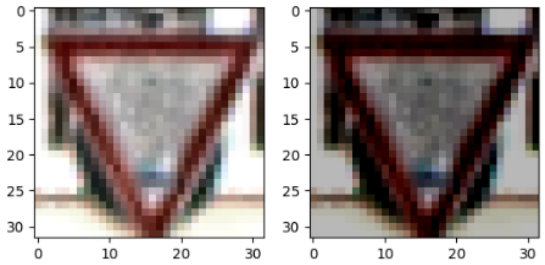

Figura 4.6: Transformação *brightness* 2.

4.2 Geometric Transformations

Sinais de trânsito no mundo real e à semelhança dos que são encontrados aqui neste *dataset* podem não apresentar uma forma bem definida. Como por exemplo, devido a atos de vandalismo, condições atmosféricas ou até mesmo a ângulo com que a imagem foi recolhida podem fazer com que o sinal apareça deslocado, torto, etc...Para combater estes casos e fazer com que o modelo os reconheça, são realizadas algumas transformações geométricas para os simular. Algumas das transformações são realizadas dinamicamente e as restantes são incrementadas ao *dataset* através da função *concatenate* do *tensorflow*. Como podemos observar pelo quadro abaixo apresentado, os resultados atingidos são satisfatórios, com realce para o modelo III.

Modelo	I	II	III
Sem *geometric transf.*	97.55%	97.62%	97.43%
Com *geometric transf.*	98.61%	98.50%	98.97%

4.2.1 Perspective

Para a realização da transformação *perspective* é utilizada a função *warpPerspective* da biblioteca *OpenCV*. Esta permite "alterar"o ângulo em que a foto é tirada. A seguir apresentam-se alguns exemplos desta transformação.

4.2.2 Translate

Para a realização da transformação *translate* é utilizada a função *RandomTranslation* do *tensorflow* e dado como argumento o intervalo [-0.3,0.3] para realizar a translação, tanto verticalmente como horizontalmente. O objetivo desta transformação passa por simular a captura incompleta de sinais de trânsito. A seguir apresenta-se alguns exemplos desta transformação.

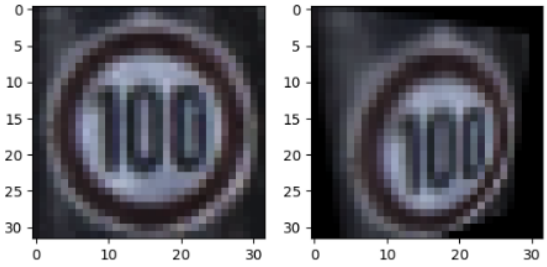

Figura 4.7: Transformação *perspective* 1.

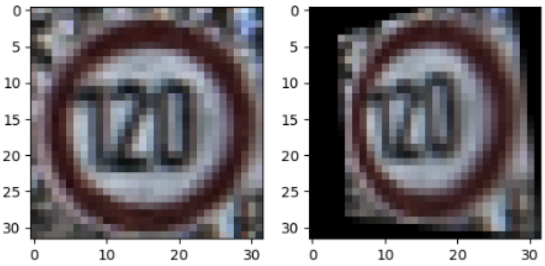

Figura 4.8: Transformação *perspective* 2.

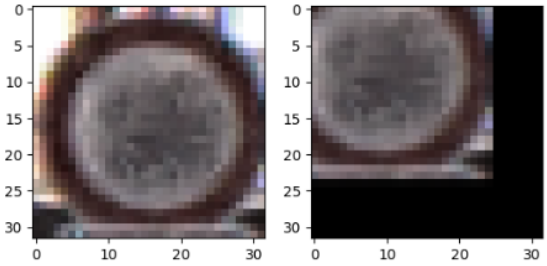

Figura 4.9: Transformação *translate* 1.

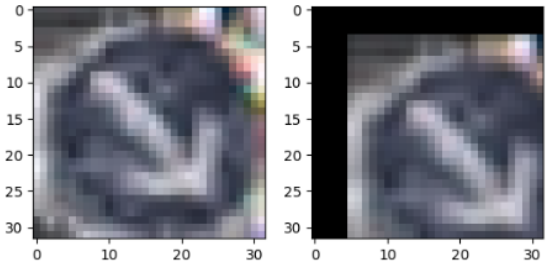

Figura 4.10: Transformação *translate* 2.

4.2.3 Rotation

Para a realização da transformação *rotation* é utilizada a função *rotate* do *tensorflow* e dado como argumento o intervalo [-0.5,0.5] para a realização das rotações. O objetivo desta transformação passa por simular a captura de sinais de trânsito feita, por exemplo, na oblíqua ou então sinais que tenham sido entortados. A seguir apresenta-se alguns exemplos desta transformação.

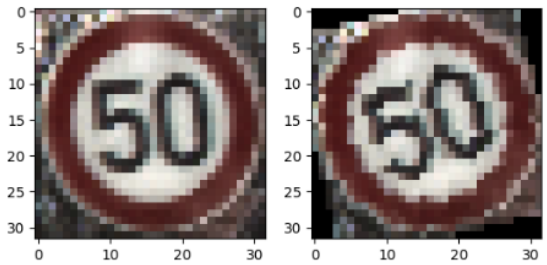

Figura 4.11: Transformação *rotation* 1.

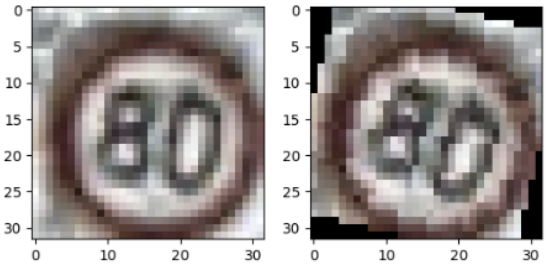

Figura 4.12: Transformação *rotation* 2.

4.2.4 Shear

Para a realização da transformação *shear* são encadiadas as funções *rotate* e *transform* do *tensorflow*. Esta transformação visa corrigir distorções encontradas em algumas das imagens do *dataset*. A seguir apresenta-se alguns exemplos desta transformação.

4.2.5 Crop

Para a realização da transformação *crop* é utilizada a função *random_crop* do *tensorflow*. Esta transformação consiste em pegar num *patch* aleatório da imagem e dar *resize* à mesma. A seguir apresenta-se alguns exemplos desta transformação.

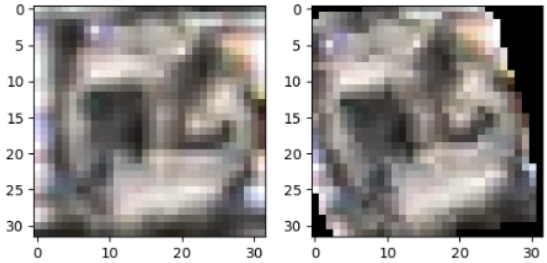

Figura 4.13: Transformação *shear* 1.

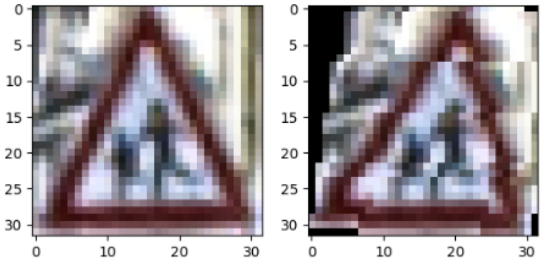

Figura 4.14: Transformação *shear* 2.

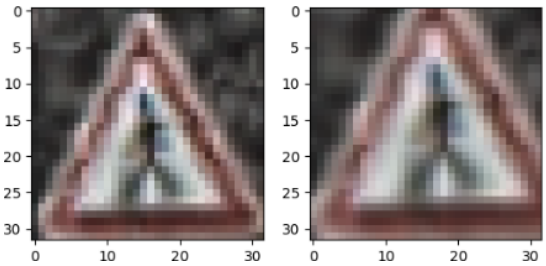

Figura 4.15: Transformação *crop* 1.

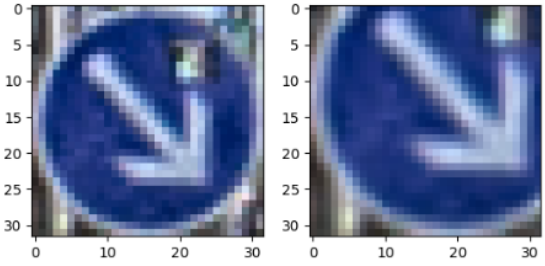

Figura 4.16: Transformação *crop* 2.

4.3 Disturbance Transformations

Nesta secção são apresentadas duas das transformações utilizadas para tentar simular vários casos observados no *dataset* onde a imagem do sinal de trânsito apresentava uma qualidade de imagem muito fraca ou então desfocada. Depois de aplicadas estas transformações ao *dataset* através de *concatenates* os resultados obtidos foram os que se apresentam na tabela em baixo. Visto que o modelo III mostrava ser o mais promissor, uma vez que a complexidade do *dataset* aumentava cada vez mais, decidiu-se optar apenas por testar esse. No entanto, como podemos observar as diferenças são mínimas, o que poderá indicar que as transformações deveriam ser feitas misturando umas com as outras, ou seja, umas dinamicamente e outras através de *concatenate*.

Modelo	III
Sem *disturbance transf.*	98.97%
Com *disturbance transf.*	99.13%

4.3.1 Perlin noise

Para a realização da transformação que adiciona *perlin noise* às imagens é utilizada a função *generate_perlin_noise_2d* da biblioteca *perlin_numpy*. Esta transformação visa simular as imagens de pouca qualidade e imagens com manchas escuras como sombras. A seguir apresenta-se alguns exemplos desta transformação.

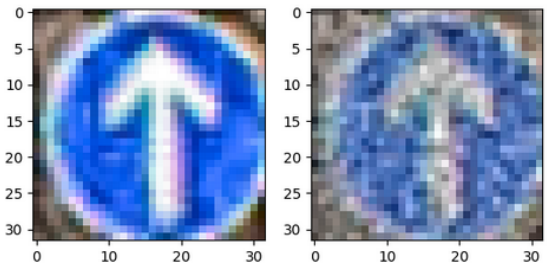

Figura 4.17: Transformação *perlin noise* 1.

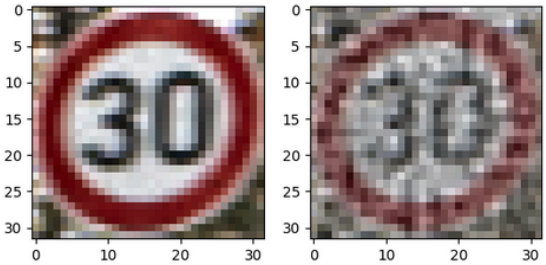

Figura 4.18: Transformação *perlin noise* 2.

4.3.2 Motion blur

Para a realização da transformação que adiciona *motion blur* às imagens é utilizada a função *filter2D* da biblioteca *OpenCV*. Esta transformação visa simular as imagens desfocadas, muito provavelmente obtidas por uma camera em movimento. A seguir apresenta-se alguns exemplos desta transformação.

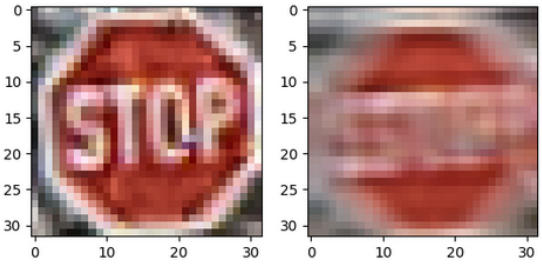

Figura 4.19: Transformação *motion blur* 1.

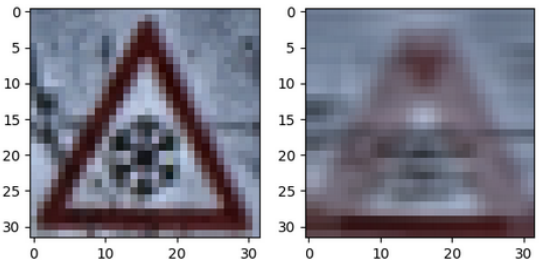

Figura 4.20: Transformação *motion blur* 2.

4.4 Oclusion Transformations

Nesta secção são tratadas as transformações relativas a oclusões provocadas na imagem. Depois de várias imagens encontradas com, por exemplo, postes, árvores ou até sombras a interferir com a visibilidade do sinal o grupo decidiu utilizar uma técnica denominada *random erasing*. Após aplicação da mesma, os resultados obtidos foram os seguintes.

Modelo	III
Sem *oclusion transf.*	99.13%
Com *oclusion transf.*	99.22%

4.4.1 Random Erasing

Para a realização da transformação *erasing* é criada uma função que escolhe aleatoriamente um pedaço da imagem e a retira da mesma dando *overwrite* á cor dos pixeis. Esta transformação visa simular imagens vandalizadas com manchas que não pertencem ao sinal ou imagens tiradas com objetos à frente por exemplo árvores ou postes. A seguir apresenta-se alguns exemplos desta transformação.

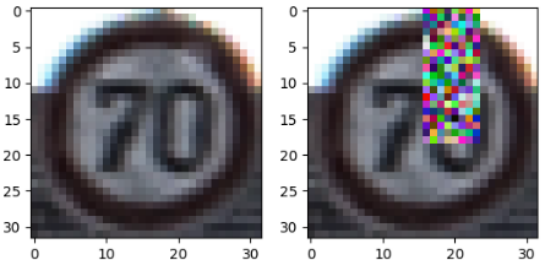

Figura 4.21: Transformação *erasing* 1.

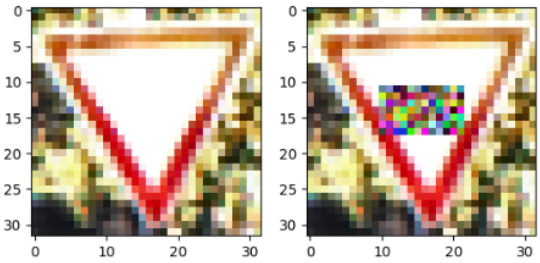

Figura 4.22: Transformação *erasing* 2.

4.5 Failed Transformations

Para além das apresentadas anteriormente, o grupo ainda testou mais alguns transformações diferentes. No entanto estas não mostraram ser eficientes. Entre estas destacam-se:

- Laplacian filter
- Gaussian blur
- Confetti noise
- Sharpness
- Zoom

5. Resultados obtidos

Depois de todas as transformações apresentadas nos capítulos anteriores o grupo decidiu fazer vários testes, alternando o valor de certas variáveis como por exemplo, tornar o *noise* mais concentrado, tornar os pedaços de imagem do *erasing* brancos em vez de terem uma cor aleatória por *pixel*, tornar o *blur* mais intenso, fazer variar os intervalos das transformações geométricas, entre outras. Com isto, reunimos um total de cinco modelos - Model III - todos a partir de um *dataset* sintético distinto. Os valores de *accuracy* atingidos por estes modelos foram os seguintes:

1. 99.44%
2. 99.38%
3. 99.26%
4. 99.22%
5. 99.16%

5.1 Ensembles

Passando para a fase seguinte do projeto, o grupo reuniu os cinco melhores modelos obtidos, que são os apresentados anteriormente, e usou-os para realizar três tipos diferentes de *ensembles*: *concatenate*, *average* e *weighted average*. Para o primeiro, *concatenate ensemble*, que de uma forma resumida reúne todos os parâmetros dos cinco modelos para criar um novo, adicionamos uma camada de *dropout* final para evitar *overfitting*. Já para os dos tipos de *weighted* ensenbles esta *layer* não foi necessária uma vez que o *output* será uma média dos *outputs* de todos os modelos. Para o *weighted average ensemble* atribuiu-se pesos aos modelos de acordo com o seu valor de *accuracy*, ou seja, quanto maior fosse esse valor, maior era a contribuição do modelo para o resultado final. Infelizmente, devido à falta de poder computacional não conseguimos treinar corretamente estes *ensembles* porque antes de os mesmos darem *improve* ao valor de *accuracy* no *valdiation set*, o tempo de execução permitido para utilização da GPU no *Colaboratory* terminava, o que não nos permitia sequer gravar os pesos para o treinarmos novamente. No entanto, deixamos aqui os resultados conseguidos apesar de não termos obtido valores satisfatórios.

Ensemble	*Concatenate*	*Average*	*Weighted Average*
Accuracy	99.13%	99.17%	99.37%

6. Conclusão

O grupo considera que realizou com sucesso os objetivos propostos inicialmente no trabalho, usando técnicas de processamento de imagem e técnicas de *data augmentation* e, por fim, a utilização de ensembles, o que permitiu alargar os conhecimentos de redes convolucionais. A accuracy obtida também é de facto bastante elevada e aproxima-se do recorde a nível global.

Uma conclusão que o grupo retirou deste trabalho é o facto de a exploração dos casos mal classificados pela rede, revelar pontos onde certas *augmentations* devem ser feitas de modo a melhorar a accuracy. Outro ponto que gostaríamos de salientar é o facto de o trabalho ser de extrema dificuldade de realização por grupos que não possuam poder computacional. Isto deve-se ao facto de ter de utilizar plataformas como o Colab, que apenas dão recursos limitados e que não permitiram ao grupo explorar de forma mais alargada as transformações que pretendíamos realizar. Para além disso, após ficarmos sem os recursos disponibilizados, apenas podíamos voltar a testar modelos no dia a seguir o que demonstrou um grande entrave na realização do trabalho. A abordagem também seguida foi realizar menos *concatenates* e apostar em modelos mais variados para treinar um ensemble, porém como já referido acima, o Colab também não "aguentava"com o treino destes modelos, o que limitou imenso as opções exploradas.

Concluindo, o grupo considera que tendo em conta as limitações encontradas, superou os objetivos com sucesso, atingindo uma boa *accuracy* e explorou diferentes técnicas de processamento de imagem.