# CNN: Aplicando Técnicas de Green Deep Learning

**Grupo:** Caio Guimarães, Eduardo Loivos

## Problema

Nós escolhemos resolver o problema de classificação de cores de veículos a partir de imagens utilizando ***CNN***.

O dataset escolhido para o problema foi o ***Vehicle-Rear*** da UFPR, que foi utilizado no artigo “Vehicle-Rear: A New Dataset to Explore Feature Fusion for Vehicle Identification Using Convolutional Neural Networks”.

Esse dataset  é um conjunto de dados que contém mais de três horas de vídeos em alta resolução, com informações precisas sobre marca, modelo, cor e ano de quase 3.000 veículos, além da posição e identificação de suas placas.

As sequências temporais mostram exemplos de (a) motocicletas; (b) carros e ônibus; (c) caminhões; (a) e (c) em condições climáticas normais; (b) quadros escuros causados pelo movimento de veículos de grande porte; e (d) condições severas de iluminação

As arquiteturas, os modelos treinados e o conjunto de dados estão disponíveis no repositório https://github.com/icarofua/vehicle-rear

## Implementação

### Dataset

Inicialmente, nós criamos um script ***data_org.py*** com o objetivo de extrair imagens de carros disponíveis no dataset e organizar em pastas de acordo com a cor do carro, que é extraída de um arquivo XML. Nesse processo, cada imagem é renomeada com a placa do veículo e somente uma imagem por placa é movida para a pasta.

Dessa forma, nós reduzimos drasticamente o tamanho do dataset e fizemos a divisão do dataset em dados de treino na pasta ***data*** e dados de testes na pasta ***data_test***.

A seguir, fizemos um processo manual de remoção de imagens de veículos em que sua cor não condizia com a pasta na qual ele foi inserido. Isso ocorre, devido ao registro da cor do veículo no XML ser uma, mas na imagem ser outra.

## Treinamento

Para o treinamento do modelo de classificação de cores de um veículo, nós escolhemos utilizar a rede neural ***ResNet50*** pré-treinada com pesos do ImageNet. O treinamento foi feito no arquivo ***color_classifier.py***.

Primeiramente, nós separamos o dataset em 80% dos dados para treino e 20% dos dados para validação.

Cada pasta do conjunto representa uma classe, ou seja, uma das possíveis cores pro veículo (blue, red, white, …).

```
# Carrega o dataset de treino e validação
train_ds = image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset='training',
    seed=seed,
    image_size=img_size,
    batch_size=batch_size
)

val_ds = image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset='validation',
    seed=seed,
    image_size=img_size,
    batch_size=batch_size
)
```

Depois, nós aplicamos o pré-processamento específico da ***ResNet50***, ele é responsável por normalizar os valores dos pixels.

```
# Aplica o preprocessamento da ResNet50
train_ds = train_ds.map(lambda x, y: (preprocess_input(x), y))
val_ds = val_ds.map(lambda x, y: (preprocess_input(x), y))
```

O modelo base então é criado sem a camada final, e ele tem os seus pesos congelados, ou seja, eles não serão atualizados durante o treinamento. Assim, esse modelo base será utilizado somente para extração das características

```
# Modelo base: ResNet50 sem o topo (congelada)
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=img_size + (3,))
base_model.trainable = False
```

Nós, então, criamos as camadas finais. A camada ***GlobalAveragePooling2D*** reduz o mapa de ativação da ResNet a um vetor, a camada *fully-connected* ***Dense(128, relu)*** realiza a aplicação da função de ativação ***ReLU***. Por fim, a camada de saída ***Dense(num_classes, softmax)*** utiliza a função ***softmax*** para transformar o modelo em probabilidades de classes, onde cada neurônio representa uma classe (cor), assim ela calcula a probabilidade de uma entrada ser de uma determinada cor.

```
# Adiciona camadas finais
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
outputs = Dense(num_classes, activation='softmax')(x)
```

A compilação do modelo é feita utilizando o otimizador ***Adam***, uma variação do ***SGD***, com ***learning_rate*** de 0,0001. A função de perda utilizada é a ***sparse categorical crossentropy*** e a métrica de avaliação é a acurácia.

O modelo foi treinado por 10 épocas e depois salvo no formato ***SavedModel*** do ***TensorFlow***, o que permite ele ser carregado posteriormente ou convertido para ***TFLite***.

```
# Compilação do modelo
model = Model(inputs=base_model.input, outputs=outputs)
model.compile(optimizer=Adam(learning_rate=1e-4),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Treinamento
model.fit(train_ds, validation_data=val_ds, epochs=10)
```

```
C:\Users\kai_o\Downloads\Trabalho03>python color_classifier.py
Found 1178 files belonging to 8 classes.
Using 943 files for training.
2025-06-24 23:51:46.951519: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neura
s:  AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
Found 1178 files belonging to 8 classes.
Using 235 files for validation.
Classes: ['black', 'blue', 'green', 'grey', 'orange', 'red', 'silver', 'white']
WARNING:tensorflow:From C:\Users\kai_o\AppData\Local\Programs\Python\Python37\lib\site-packages\tensorflow\python\autograph\pyct\static_ana
liveness) is deprecated and will be removed after 2023-09-23.
Instructions for updating:
Lambda fuctions will be no more assumed to be used in the statement where they are used, or at least in the same block. https://github.com/
Epoch 1/10
30/30 [==============================] - 76s 2s/step - loss: 1.7426 - accuracy: 0.3266 - val_loss: 1.3063 - val_accuracy: 0.5362
Epoch 2/10
30/30 [==============================] - 70s 2s/step - loss: 1.1452 - accuracy: 0.6543 - val_loss: 0.9740 - val_accuracy: 0.7149
Epoch 3/10
30/30 [==============================] - 66s 2s/step - loss: 0.8663 - accuracy: 0.7540 - val_loss: 0.8244 - val_accuracy: 0.7702
Epoch 4/10
30/30 [==============================] - 66s 2s/step - loss: 0.7175 - accuracy: 0.7985 - val_loss: 0.7464 - val_accuracy: 0.7872
Epoch 5/10
30/30 [==============================] - 65s 2s/step - loss: 0.6229 - accuracy: 0.8240 - val_loss: 0.6923 - val_accuracy: 0.8170
Epoch 6/10
30/30 [==============================] - 66s 2s/step - loss: 0.5515 - accuracy: 0.8494 - val_loss: 0.6538 - val_accuracy: 0.8255
Epoch 7/10
30/30 [==============================] - 66s 2s/step - loss: 0.4853 - accuracy: 0.8802 - val_loss: 0.6272 - val_accuracy: 0.8298
Epoch 8/10
30/30 [==============================] - 61s 2s/step - loss: 0.4390 - accuracy: 0.8897 - val_loss: 0.6079 - val_accuracy: 0.8213
Epoch 9/10
30/30 [==============================] - 60s 2s/step - loss: 0.4017 - accuracy: 0.9109 - val_loss: 0.5939 - val_accuracy: 0.8298
Epoch 10/10
30/30 [==============================] - 60s 2s/step - loss: 0.3674 - accuracy: 0.9099 - val_loss: 0.5849 - val_accuracy: 0.8298
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _ji
tions will not be directly callable after loading.
 Modelo salvo como vehicle_color_classifier_resnet50_savedmodel

C:\Users\kai_o\Downloads\Trabalho03>
```

## Resultados

A avaliação do modelo foi feita no arquivo ***model_evaluation.py***.

O primeiro passo é carregar o modelo salvo no treinamento e carregar todas as imagens do conjunto de teste. Cada imagem, é redimensionada para  224x224 e depois convertida parra um array ***NumPy***. Por fim, é aplicado o mesmo pré-processamento do ***ResNet50*** e então é feita a predição.

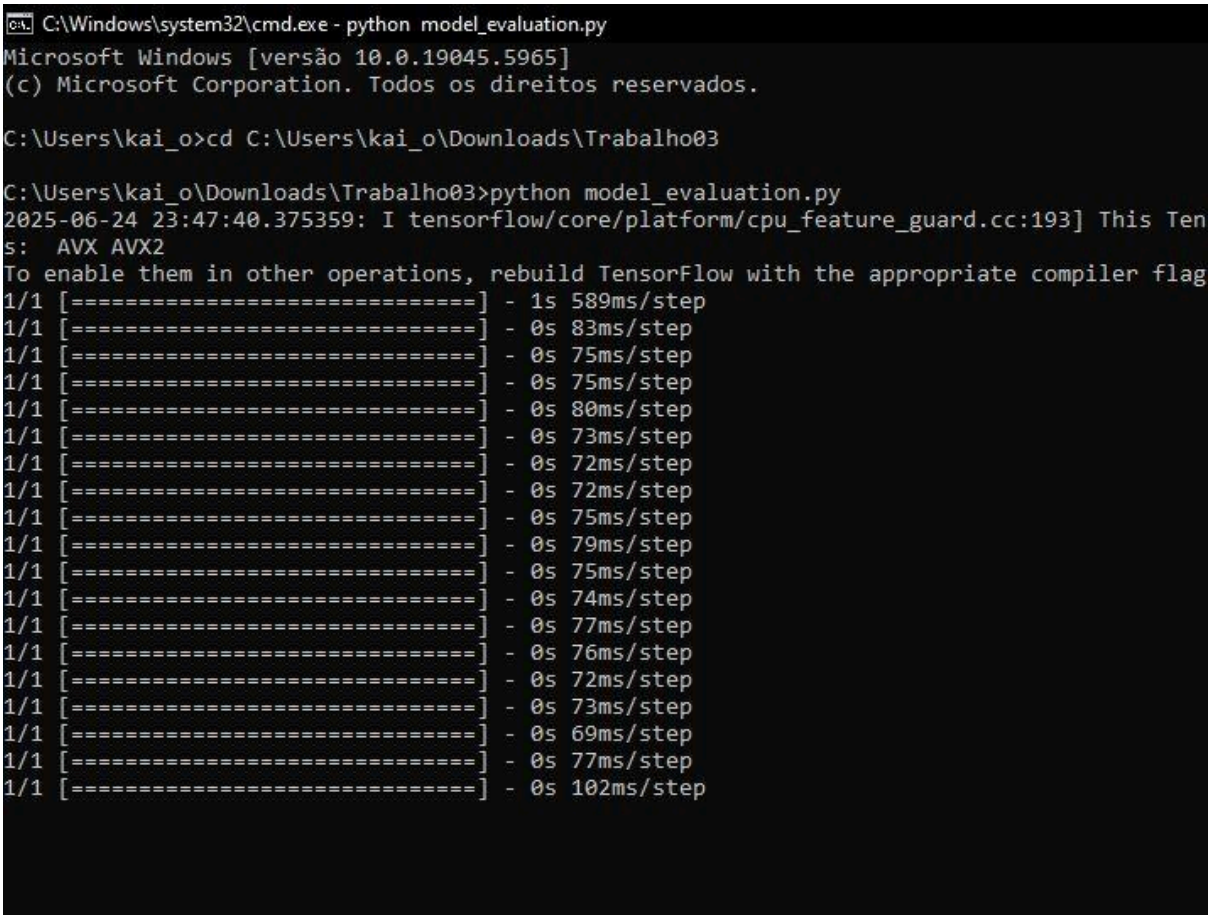
```python
# Percorre cada classe (pasta de cor)
for label in class_names:
    class_path = os.path.join(test_dir, label)
    for fname in os.listdir(class_path):
        if not fname.lower().endswith(('.jpg', '.jpeg', '.png')):
            continue

        img_path = os.path.join(class_path, fname)

        # Carrega imagem e pré-processa
        img = image.load_img(img_path, target_size=(224, 224))  # ajuste ao tamanho do seu modelo
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array = preprocess_input(img_array)

        # Predição
        prediction = model.predict(img_array)
        predicted_class = class_names[np.argmax(prediction)]

        # Registra resultado
        y_true.append(label)
        y_pred.append(predicted_class)
```

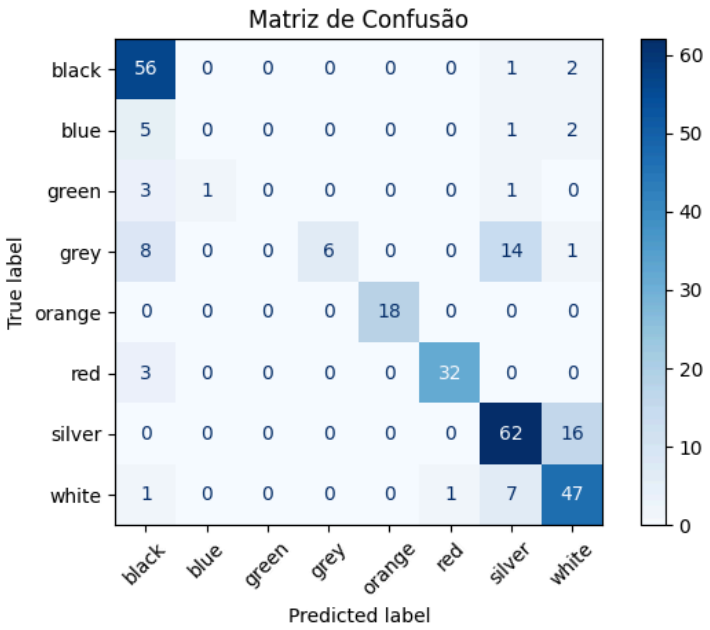



A partir da predição, podemos gerar o relatório de métricas e realizar sua análise.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| black | 0.74 | 0.95 | 0.83 | 59 |
| blue | 0.00 | 0.00 | 0.00 | 8 |
| green | 0.00 | 0.00 | 0.00 | 5 |
| grey | 1.00 | 0.21 | 0.34 | 29 |
| orange | 1.00 | 1.00 | 1.00 | 18 |
| red | 0.97 | 0.91 | 0.94 | 35 |
| silver | 0.72 | 0.79 | 0.76 | 78 |
| white | 0.69 | 0.84 | 0.76 | 56 |
| accuracy | | | 0.77 | 288 |
| macro avg | 0.64 | 0.59 | 0.58 | 288 |
| weighted avg | 0.76 | 0.77 | 0.73 | 288 |

Podemos perceber que o modelo apresenta uma acurácia de 77% e que ele não conseguiu aprender a prever as cores azul e verde, um dos motivos pode ser a baixa quantidade de exemplos disponíveis.

Por último, é gerada uma matriz de confusão para análise dos resultados.




A partir dela é possível analisar que o modelo apresenta um bom desempenho para a maioria das classes. As cores azul e verde foram confundidas com a cor preta, e a cor cinza foi confundida com a cor preta e prata, o que pode ser explicado pela proximidade das cores e pela variação de luz das imagens.

# Técnicas de Green AI

Com a intenção de reduzir o tamanho do modelo, foi utilizada duas técnicas de Green AI

## Pruning

É uma técnica de compressão que zera pesos de baixa importância na rede neural, criando esparsidade. Ela foi aplicada através do arquivo ***generate_pruning.py***.

Para utilizar esta técnica vamos utilizar a biblioteca ***tfmot*** (TensorFlow Model Optimization Toolkit). A função responsável por aplicar o pruning é a ***prune_low_magnitude***, zerando os pesos menos relevantes, o parâmetro ***final_sparsity=0.5*** indica que a poda termina com 50% dos pesos zerados. O parâmetro **end_step=1000** indica o numéro máximo de iterações a serem feitas.

```
35  # Define parâmetros de pruning
36  pruning_params = {
37      'pruning_schedule': tfmot.sparsity.keras.PolynomialDecay(
38          initial_sparsity=0.0,
39          final_sparsity=0.5,
40          begin_step=0,
41          end_step=1000  # ajuste com base no número total de steps
42      )
43  }
44
```

Para aplicar o pruning, é necessário carregar o modelo já treinado que foi salvo anteriormente. O modelo com pruning precisa ser recompilado para ser ajustado com a poda ativa.

```
8
9   model = load_model("vehicle_color_classifier_resnet50_savedmodel")
0   model_for_pruning = prune_low_magnitude(model, **pruning_params)
1
2
3 v model_for_pruning.compile(optimizer=Adam(1e-4),
4                             loss='sparse_categorical_crossentropy',
5                             metrics=['accuracy'])
6
7   # Re-treinar por algumas épocas
8 v model_for_pruning.fit(train_ds, validation_data=val_ds, epochs=5,
9                          callbacks=[tfmot.sparsity.keras.UpdatePruningStep()])
0
1   model = tfmot.sparsity.keras.strip_pruning(model_for_pruning)
2   model.save("pruned_vehicle_color_classifier", save_format="tf")
3
```

```
C:\Users\kai_o\Downloads\Trabalho03>python generate_pruning.py
Found 1178 files belonging to 8 classes.
Using 943 files for training.
2025-06-25 00:27:31.581800: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep N
s:  AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
Found 1178 files belonging to 8 classes.
Using 235 files for validation.
Classes: ['black', 'blue', 'green', 'grey', 'orange', 'red', 'silver', 'white']
WARNING:tensorflow:From C:\Users\kai_o\AppData\Local\Programs\Python\Python37\lib\site-packages\tensorflow\python\autograph\pyct\static
liveness) is deprecated and will be removed after 2023-09-23.
Instructions for updating:
Lambda fuctions will be no more assumed to be used in the statement where they are used, or at least in the same block. https://github.
Epoch 1/5
30/30 [==============================] - 89s 2s/step - loss: 0.4067 - accuracy: 0.8950 - val_loss: 0.3766 - val_accuracy: 0.8979
Epoch 2/5
30/30 [==============================] - 65s 2s/step - loss: 0.3465 - accuracy: 0.9130 - val_loss: 0.3765 - val_accuracy: 0.8979
Epoch 3/5
30/30 [==============================] - 68s 2s/step - loss: 0.3052 - accuracy: 0.9279 - val_loss: 0.3609 - val_accuracy: 0.9064
Epoch 4/5
30/30 [==============================] - 70s 2s/step - loss: 0.2750 - accuracy: 0.9332 - val_loss: 0.3733 - val_accuracy: 0.9106
Epoch 5/5
30/30 [==============================] - 73s 2s/step - loss: 0.2542 - accuracy: 0.9459 - val_loss: 0.3756 - val_accuracy: 0.8894
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty unti
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op,
tions will not be directly callable after loading.

C:\Users\kai_o\Downloads\Trabalho03>
```

## Quantização

É uma técnica que reduz a precisão numérica dos pesos e ativações do modelo, diminuindo o tamanho, uso de memória, e podendo aumentar a velocidade da inferência.

A quantização é feita convertendo o modelo treinado para o formato ***TensorFlow Lite (TFLite)***. Ela foi executada através do arquivo ***generate_quantization.py***.

generate_quantization.py > ...

```python
import tensorflow as tf

model = tf.keras.models.load_model("vehicle_color_classifier_resnet50_savedmodel")

converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]

quantized_model = converter.convert()

# Salva modelo quantizado
with open('model_quantized.tflite', 'wb') as f:
    f.write(quantized_model)

```

```
PS C:\Users\loivo\Documents\Projects\Aula_GAI\Trabalho03> python3 .\eval_TFLite.py
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
Found 288 files belonging to 8 classes.
2025-06-24 23:17:10.235600: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with
 79%
```

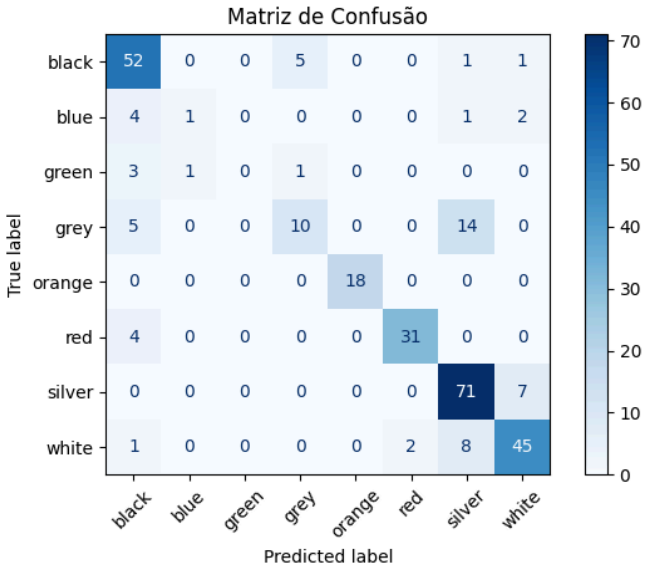
```
to use available CPU instructions in performance-critical operations.
ow with the appropriate compiler flags.
                                                    | 233/288 [35:27<08:33,  9.34s/it]
```

## Resultados

As métricas obtidas pelo modelo após o pruning são as seguintes:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| black | 0.75 | 0.88 | 0.81 | 59 |
| blue | 0.50 | 0.12 | 0.20 | 8 |
| green | 0.00 | 0.00 | 0.00 | 5 |
| grey | 0.62 | 0.34 | 0.44 | 29 |
| orange | 1.00 | 1.00 | 1.00 | 18 |
| red | 0.94 | 0.89 | 0.91 | 35 |
| silver | 0.75 | 0.91 | 0.82 | 78 |
| white | 0.82 | 0.80 | 0.81 | 56 |
| accuracy | | | 0.79 | 288 |
| macro avg | 0.67 | 0.62 | 0.63 | 288 |
| weighted avg | 0.77 | 0.79 | 0.77 | 288 |

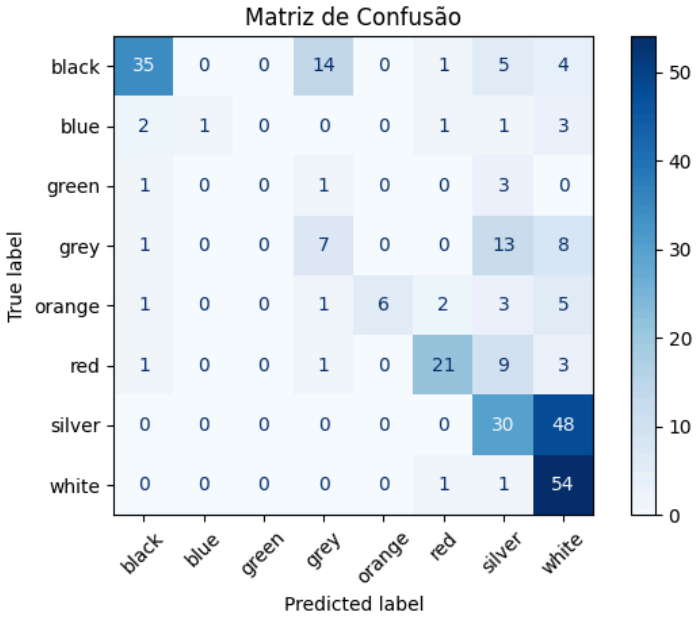
A matriz de confusão gerada pelo modelo após o pruning é a seguinte:




As métricas obtidas pelo modelo após a quantização são as seguintes:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| black | 0.85 | 0.59 | 0.70 | 59 |
| blue | 1.00 | 0.12 | 0.22 | 8 |
| green | 0.00 | 0.00 | 0.00 | 5 |
| grey | 0.29 | 0.24 | 0.26 | 29 |
| orange | 1.00 | 0.33 | 0.50 | 18 |
| red | 0.81 | 0.60 | 0.69 | 35 |
| silver | 0.46 | 0.38 | 0.42 | 78 |
| white | 0.43 | 0.96 | 0.60 | 56 |
| accuracy | | | 0.53 | 288 |
| macro avg | 0.61 | 0.41 | 0.42 | 288 |
| weighted avg | 0.60 | 0.53 | 0.52 | 288 |

A matriz de confusão gerada pela quantização é a seguinte:




Em relação ao tamanho dos modelos, a CNN original possui um tamanho de 93.9 MB, enquanto o modelo após o pruning possui um tamanho de 91.9 MB e o modelo após a quantização ocupa somente 22.8 MB sendo este o ideal para execução em dispositivos com baixa capacidade de armazenamento.

Quanto ao processamento, a técnica de pruning gera uma matriz esparsa reduzindo o tempo de processamento embora preserve a mesma dimensão. O modelo TFLite é otimizado para dispositivos móveis, o teste em CPU precisa ser emulado e portanto consome muito tempo.

## Conclusão

Após analisarmos as métricas dos modelos pode que o modelo original apresenta uma acurácia de 77% e que ele não conseguiu aprender a prever as cores azul e verde, além de ter dificuldades com a cor cinza. Essas dificuldades enfrentadas estão presentes nos demais modelos, no entanto, no modelo com quantização o desempenho cai consideravelmente, apresentando apenas 53% de acurácia, 61% de precisão e 41% de recall.

O modelo com pruning apresentou o melhor desempenho, obtendo uma acurácia de 79%, que é superior ao modelo original, e um tempo de processamento menor do que o original.