Child Seat Localization

Author: Fernando Kelvin da Silva Soares

Notebook Versio: V0.0

Introduction

In this notebook we will explore the child seat localization problem. The goal here, is to say in wich seat a child/infant seat is locates in the back part of a veicle. It was used as reference the SVIRO sunthetic dataset and some papers [1][2] wrote by the svriro team, available in [the SVIRO website](), as weel as, other cientific papers and websites to implement a image classifier for each back seat position.

Difÿfferently from the implementations presented by the SVIRO team on their papers, wich are focused on train the networks in one vehicle and see how they perform in unknown vhicles. In this work was decided to use more then one veichle for trainin and check the accuracy in unknown vehicles, to explore if there is a significant reseult compared to the single vehicle training.

The porpouse of this project is to give a flexible and fast framework to explore the child seat problem understanding the limitations and possibilities involved.

Copyright The SVIRO Authors

[1] Steve Dias Da Cruz, Oliver Wasenmüller, Hans-Peter Beise, Thomas Stifter, & Didier Stricker (2020). SVIRO: Synthetic Vehicle Interior Rear Seat Occupancy Dataset and Benchmark. In IEEE Winter Conference on Applications of Computer Vision (WACV).

[2] Steve Dias Da Cruz, Bertram Taetz, Oliver Wasenmüller, Thomas Stifter, & Didier Stricker (2021). Autoencoder Based Inter-Vehicle Generalization for In-Cabin Occupant Classification. In IEEE Intelligent Vehicles Symposium (IV).

Attribution-NonCommercial-ShareAlike 4.0 International

Copyright 2019 The TensorFlow Authors.

Licensed under the Apache License, Version 2.0 (the "License");

MIT License

Imports

```
import matplotlib.pyplot as plt
import numpy as np
import random
import pathlib
```

```
import pathlib
import seaborn as sns
from os import listdir
import os.path as path
import os
import PIL
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.layers.experimental.preprocessing import RandomRotation
from tensorflow.keras.layers.experimental.preprocessing import RandomFlip
from tensorflow.keras.layers.experimental.preprocessing import RandomZoom
from tensorflow.keras.layers.experimental.preprocessing import RandomContrast

#Check the tensorflow version, the recomended for this notebook is the 2.5.0.
print(tf.__version__)
```

```
2.5.0
```

If you are using Google Colab, is recomended to use the GPU for fast training. To acctivate the GPU, go to *"Runtime->Change runtime type"* and select GPU at in the *"Hardware Accelerator"* dropdown menu.

You can check the alocated GPU model runnning the following command.

```
!nvidia-smi
```

```
Wed Jun 16 03:00:30 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 465.27       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   34C    P0    26W / 250W |      0MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Conecto to your Drive repository

If you want to use you Google Drive repository to load the train dataset, run the following session, log-in into your accont, copy the verification code and paste in the output entry.

```
from google.colab import drive

drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive
```

Global Definitions

Here we have some constants that are used in all the sections of the notebook.

Set up the MAIN PATH to you repository main folder containing, the models and datasets.

The IMAGE_SHAPE will depend on the network you are using. In this case we will train a [EfficientNet stadard implementation](#) network that accepts different input sizes, so can use the single seat image size of 250x550.

In this notebook we can explore not only the SVIRO classes but also a summarized version of the existent clsses, where all the child/infant seat with or without child are cosidered as the same class.

```
MAIN_PATH  = '/content/gdrive/MyDrive/Colab_Notebooks/Child_Seat_Localization/classifier'

IMAGE_HIGHT = 258
IMAGE_WIDTH = 258
IMAGE_SHAPE = (IMAGE_HIGHT, IMAGE_WIDTH, 3) #The image shape may vary according to you network model

# SVIRO Classes
CLASSES = {0: "Empty seat",
           1: "Infant in infant seat",
           2: "Child on child seat",
           3: "Adult passenger",
           4: "Everyday object",
           5: "Infant seat without baby",
           6: "Child seat without child"}

```

Preparing the Dataset

As we are implementing a sigle seat image cassifier the training dataset was created using the single seat grayscale images fom the SVIRO dataset.

In the original dataset we have 10 different vehicles, that we can arrange in 3 classes: Small Vhicle (two door), Regular Vehicle and Big vehicle (SUV and Truck). So, to create the train/validation dataset one vehicle from each one of this classes was chosen, considereng a good variability of interior styles. Are they: Renault **Zoe**, Toyota **Hilux** and Tesla **Model3**.

From the original train folder of each one, 1800 images ware taken for each of the seven classes, totalizing 3150 images on the final train/validation dataset.

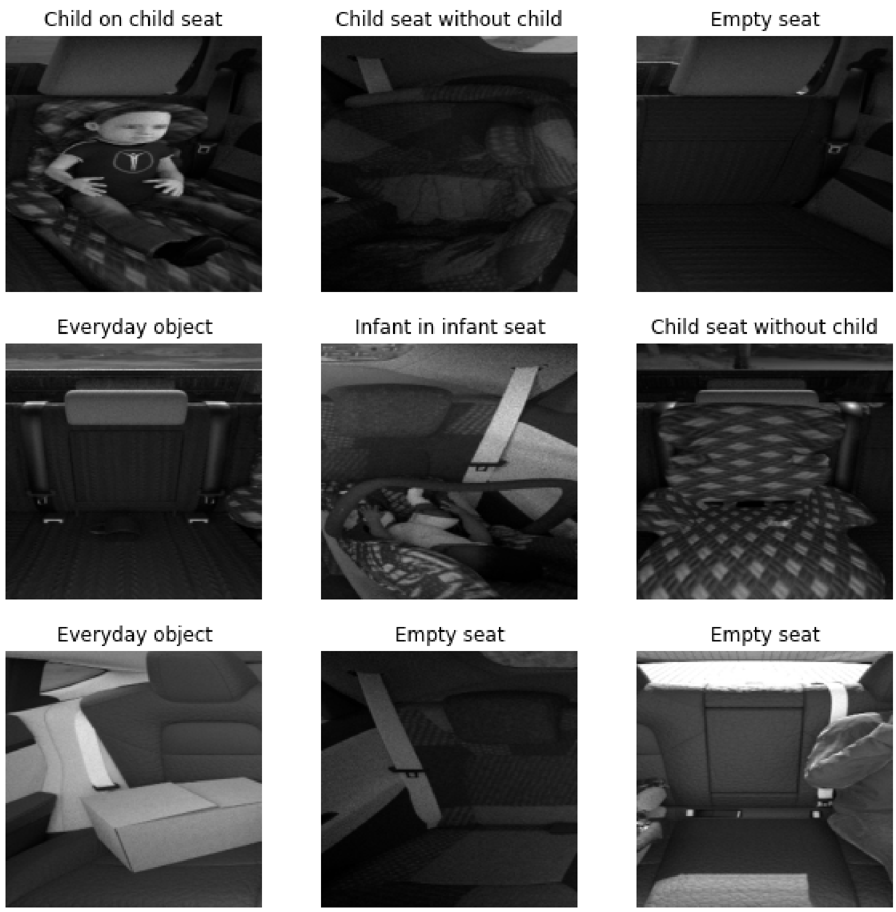
It was used a 80:20 split for training and validation data.

```
train_dataset_path = '{}/datasets/train/sviro_classes'.format(MAIN_PATH)

# From the paper of efficient det they use a 258x258 resolution for the b7 model
BATCH_SIZE = 64
VALIDATION_SPLIT = 0.2

# crate a train and validation dataset with a 80:20 split
train_dataset = image_dataset_from_directory(train_dataset_path,
                                             validation_split=VALIDATION_SPLIT,
                                             subset="training",
                                             seed=123,
                                             image_size=(IMAGE_HIGHT, IMAGE_WIDTH),
                                             color_mode="rgb",
                                             batch_size=BATCH_SIZE,
                                             smart_resize=False)

valid_dataset = image_dataset_from_directory(train_dataset_path,
                                             validation_split=VALIDATION_SPLIT,
                                             subset="validation",
                                             seed=123,
                                             image_size=(IMAGE_HIGHT, IMAGE_WIDTH),
                                             color_mode="rgb",
                                             batch_size=BATCH_SIZE,
                                             smart_resize=False)
```

```
Found 3150 files belonging to 7 classes.
Using 2520 files for training.
Found 3150 files belonging to 7 classes.
Using 630 files for validation.
```

Looking at some examples from the train dataset.

```
plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(CLASSES[int(labels[i])])
        plt.axis("off")
```



Here we use AUTOTUNE prefetch to optimise the traing performance.

```
AUTOTUNE = tf.data.experimental.AUTOTUNE

train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = valid_dataset.prefetch(buffer_size=AUTOTUNE)
```

Training the Model

Create the model based on a pre-treined model

For a faster implemantation and training, it was chosen to use the pre-trained models, on the imagenet dataset, from [Keras Aplication](#) module.

As said before, for this implemantation the EfficientNet B7 was chosen, due it's good performance on the imagenet dataset and faster inference time compared to the best existent neural networks, as we can see in the [paperswithcode.com Image Classification on ImageNet benchmark].

A addinal data augmentation layer was placed ind the model during training to avoid overfiting.

```
number_of_classes = len(CLASSES)

data_augmentation = keras.Sequential([
    RandomRotation(0.2,input_shape=IMAGE_SHAPE),
    RandomFlip(mode="horizontal"),
    RandomContrast(factor=0.2)
  ]
)
pretrained_model = tf.keras.applications.EfficientNetB7(input_shape=IMAGE_SHAPE,
                                                        include_top=False,
                                                        weights="imagenet")
# pretrained_model.summary()

#Freeze the original convolutional wheights from the pre treined model
pretrained_model.trainable=False

model = tf.keras.Sequential([
    data_augmentation,
    pretrained_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(number_of_classes,activation="softmax")
])

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential (Sequential)      (None, 258, 258, 3)       0
_________________________________________________________________
efficientnetb7 (Functional)  (None, 9, 9, 2560)        64097687
_________________________________________________________________
global_average_pooling2d (Gl (None, 2560)              0
_________________________________________________________________
dense (Dense)                (None, 7)                 17927
=================================================================
Total params: 64,115,614
Trainable params: 17,927
Non-trainable params: 64,097,687
_________________________________________________________________
```

Compiling the model

For optimize the mothel the Adam otmizer was used, for beeing easier to use and have a good performance on most problems.

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['accuracy'])
```

Create chececk points and train the model

As only the last layers (classification layes) will be trained, the model is fitted for only 25 epochs. And a check point will be saved each time we get a better accuracy result on the validation dataset.

```
checkpoint_path = "{}/models/EfficientNet_b7_258_258_RFC/".format(MAIN_PATH)

# Create a callback that saves the model's weights
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                 save_weights_only=True,
                                                 verbose=1,
                                                 monitor='val_accuracy',
                                                 mode='max',
                                                 save_best_only=True)


# If you want to train from the last chachpoint uncomment the section below
# latest_checkpoint = tf.train.latest_checkpoint(checkpoint_path)
# print(latest_checkpoint)

# if not latest==None:
#   model.load_weights(latest)

# Train the model with the new callback
epochs = 25
history = model.fit(train_dataset,
                    validation_data=valid_dataset,
                    epochs=epochs,
                    callbacks=[cp_callback])  # Pass callback to training
```

```
Epoch 1/25
40/40 [==============================] - 54s 900ms/step - loss: 1.1647 - accuracy: 0.6865 - val_loss: 0.5

Epoch 00001: val_accuracy improved from -inf to 0.89365, saving model to /content/gdrive/MyDrive/Colab_No
Epoch 2/25
40/40 [==============================] - 32s 772ms/step - loss: 0.5463 - accuracy: 0.8829 - val_loss: 0.3

Epoch 00002: val_accuracy improved from 0.89365 to 0.93016, saving model to /content/gdrive/MyDrive/Colab
Epoch 3/25
40/40 [==============================] - 32s 771ms/step - loss: 0.3933 - accuracy: 0.9063 - val_loss: 0.2

Epoch 00003: val_accuracy improved from 0.93016 to 0.95238, saving model to /content/gdrive/MyDrive/Colab
Epoch 4/25
40/40 [==============================] - 32s 772ms/step - loss: 0.3288 - accuracy: 0.9274 - val_loss: 0.2

Epoch 00004: val_accuracy improved from 0.95238 to 0.96032, saving model to /content/gdrive/MyDrive/Colab
Epoch 5/25
40/40 [==============================] - 33s 773ms/step - loss: 0.2793 - accuracy: 0.9333 - val_loss: 0.2

Epoch 00005: val_accuracy did not improve from 0.96032
Epoch 6/25
40/40 [==============================] - 32s 772ms/step - loss: 0.2618 - accuracy: 0.9317 - val_loss: 0.1

Epoch 00006: val_accuracy improved from 0.96032 to 0.96349, saving model to /content/gdrive/MyDrive/Colab
Epoch 7/25
40/40 [==============================] - 32s 774ms/step - loss: 0.2168 - accuracy: 0.9492 - val_loss: 0.1

Epoch 00007: val_accuracy did not improve from 0.96349
Epoch 8/25
40/40 [==============================] - 32s 772ms/step - loss: 0.2139 - accuracy: 0.9433 - val_loss: 0.1

Epoch 00008: val_accuracy improved from 0.96349 to 0.96984, saving model to /content/gdrive/MyDrive/Colab
Epoch 9/25
40/40 [==============================] - 32s 773ms/step - loss: 0.1911 - accuracy: 0.9520 - val_loss: 0.1

Epoch 00009: val_accuracy improved from 0.96984 to 0.97143, saving model to /content/gdrive/MyDrive/Colab
Epoch 10/25
40/40 [==============================] - 32s 773ms/step - loss: 0.1853 - accuracy: 0.9548 - val_loss: 0.1

Epoch 00010: val_accuracy did not improve from 0.97143
Epoch 11/25
40/40 [==============================] - 32s 773ms/step - loss: 0.1712 - accuracy: 0.9579 - val_loss: 0.1

Epoch 00011: val_accuracy did not improve from 0.97143
Epoch 12/25
40/40 [==============================] - 32s 772ms/step - loss: 0.1662 - accuracy: 0.9544 - val_loss: 0.1

Epoch 00012: val_accuracy improved from 0.97143 to 0.97302, saving model to /content/gdrive/MyDrive/Colab
```

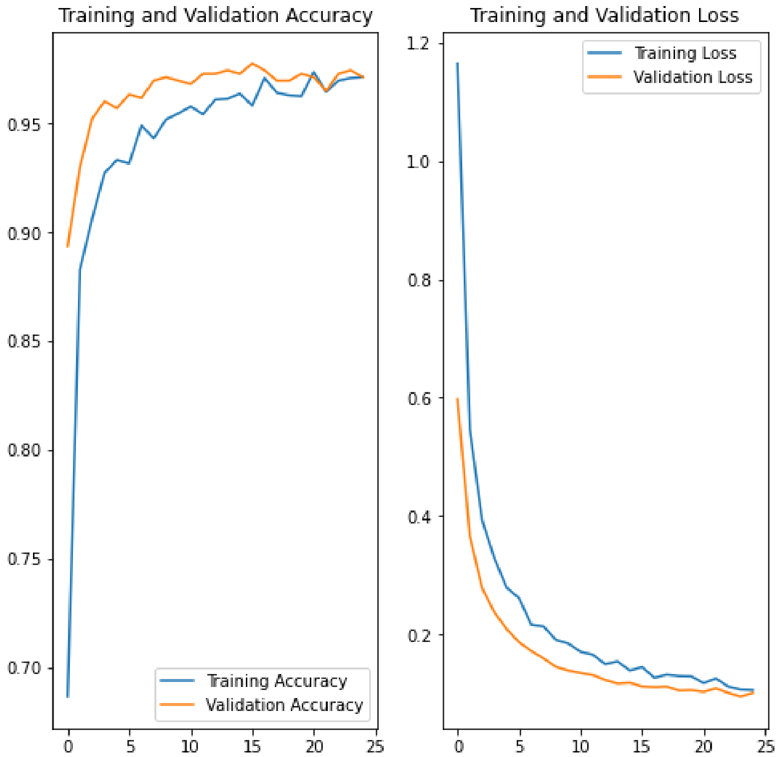
```
Epoch 13/25
40/40 [==============================] - 32s 773ms/step - loss: 0.1503 - accuracy: 0.9611 - val_loss: 0.1

Epoch 00013: val_accuracy did not improve from 0.97302
Epoch 14/25
40/40 [==============================] - 32s 772ms/step - loss: 0.1549 - accuracy: 0.9615 - val_loss: 0.1

Epoch 00014: val_accuracy improved from 0.97302 to 0.97460, saving model to /content/gdrive/MyDrive/Colab
Epoch 15/25
40/40 [==============================] - 32s 772ms/step - loss: 0.1395 - accuracy: 0.9639 - val_loss: 0.1
```

Visualize training results

Create plots of loss and accuracy on the training and validation sets.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
```

```
Text(0.5, 1.0, 'Training and Validation Loss')
```



Saving the model

We can save the entire model for use in future applications.

```
save_model_path = '{}/models/EfficientNet_b7_258_258_RFC/model'.format(MAIN_PATH)
model.save(save_model_path)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/utils/generic_utils.py:497: CustomMaskWarnin
  category=CustomMaskWarning)
INFO:tensorflow:Assets written to: /content/gdrive/MyDrive/Colab_Notebooks/Child_Seat_Localization/classifi
```

Testing the model performance

Load the model

It is possible to load differents models to evalueta the performance. Pay atention on the IMAGE_HIGHT and IMAGE_WIDTH to match the model input size.

```
load_model_path = '{}/models/EfficientNet_b7_258_258_RFC/model'.format(MAIN_PATH)

model = tf.keras.models.load_model(load_model_path)
```

```
WARNING:absl:Importing a function (__inference_block3a_expand_activation_layer_call_and_return_conditiona
WARNING:absl:Importing a function (__inference_block1a_activation_layer_call_and_return_conditional_losse

WARNING:absl:Importing a function (__inference_block4b_activation_layer_call_and_return_conditional_losse
WARNING:absl:Importing a function (__inference_block5j_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block5c_expand_activation_layer_call_and_return_conditiona
WARNING:absl:Importing a function (__inference_block6i_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block3e_activation_layer_call_and_return_conditional_losse
WARNING:absl:Importing a function (__inference_block2a_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block2g_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block2b_activation_layer_call_and_return_conditional_losse
WARNING:absl:Importing a function (__inference_block2f_expand_activation_layer_call_and_return_conditiona
WARNING:absl:Importing a function (__inference_block6j_activation_layer_call_and_return_conditional_losse
WARNING:absl:Importing a function (__inference_block6l_activation_layer_call_and_return_conditional_losse
WARNING:absl:Importing a function (__inference_block3b_activation_layer_call_and_return_conditional_losse
WARNING:absl:Importing a function (__inference_block4a_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block5c_activation_layer_call_and_return_conditional_losse
WARNING:absl:Importing a function (__inference_block6j_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block6b_expand_activation_layer_call_and_return_conditiona
WARNING:absl:Importing a function (__inference_block5f_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block4c_expand_activation_layer_call_and_return_conditiona
WARNING:absl:Importing a function (__inference_block1d_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block5g_expand_activation_layer_call_and_return_conditiona
WARNING:absl:Importing a function (__inference_block1c_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block7d_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block6l_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block5a_expand_activation_layer_call_and_return_conditiona
WARNING:absl:Importing a function (__inference_block6d_expand_activation_layer_call_and_return_conditiona
WARNING:absl:Importing a function (__inference_block6f_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block4a_activation_layer_call_and_return_conditional_losse
WARNING:absl:Importing a function (__inference_block1d_activation_layer_call_and_return_conditional_losse
WARNING:absl:Importing a function (__inference_block4g_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block4i_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block2e_activation_layer_call_and_return_conditional_losse
WARNING:absl:Importing a function (__inference_block6l_expand_activation_layer_call_and_return_conditiona
WARNING:absl:Importing a function (__inference_block4c_expand_activation_layer_call_and_return_conditiona
WARNING:absl:Importing a function (__inference_block5h_activation_layer_call_and_return_conditional_losse
WARNING:absl:Importing a function (__inference_block6j_expand_activation_layer_call_and_return_conditiona
WARNING:absl:Importing a function (__inference_block6a_expand_activation_layer_call_and_return_conditiona
WARNING:absl:Importing a function (__inference_block5g_activation_layer_call_and_return_conditional_losse
WARNING:absl:Importing a function (__inference_block5f_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block5d_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block4j_expand_activation_layer_call_and_return_conditiona
WARNING:absl:Importing a function (__inference_block3g_expand_activation_layer_call_and_return_conditiona
WARNING:absl:Importing a function (__inference_block6j_expand_activation_layer_call_and_return_conditiona
WARNING:absl:Importing a function (__inference_block4g_expand_activation_layer_call_and_return_conditiona
WARNING:absl:Importing a function (__inference_block7b_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block1c_activation_layer_call_and_return_conditional_losse
WARNING:absl:Importing a function (__inference_block5i_se_reduce_layer_call_and_return_conditional_losses
```

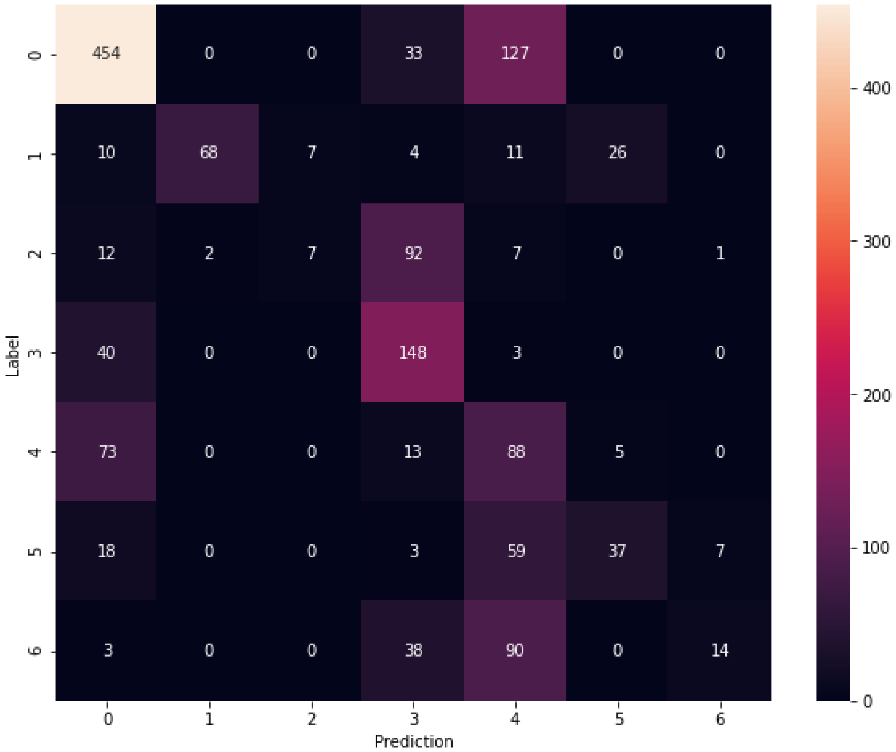
```
WARNING:absl:Importing a function (__inference_block5i_activation_layer_call_and_return_conditional_losse
WARNING:absl:Importing a function (__inference_block5g_expand_activation_layer_call_and_return_conditiona
WARNING:absl:Importing a function (__inference_block6k_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block2b_expand_activation_layer_call_and_return_conditiona
WARNING:absl:Importing a function (__inference_block6g_activation_layer_call_and_return_conditional_losse
WARNING:absl:Importing a function (__inference_block3c_activation_layer_call_and_return_conditional_losse
WARNING:absl:Importing a function (__inference_block4h_se_reduce_layer_call_and_return_conditional_losses
WARNING:absl:Importing a function (__inference_block6i_activation_layer_call_and_return_conditional_losse
WARNING:absl:Importing a function (__inference_block2a_expand_activation_layer_call_and_return_conditiona
```

Evaluate the model performance and plot the confusuin matrix

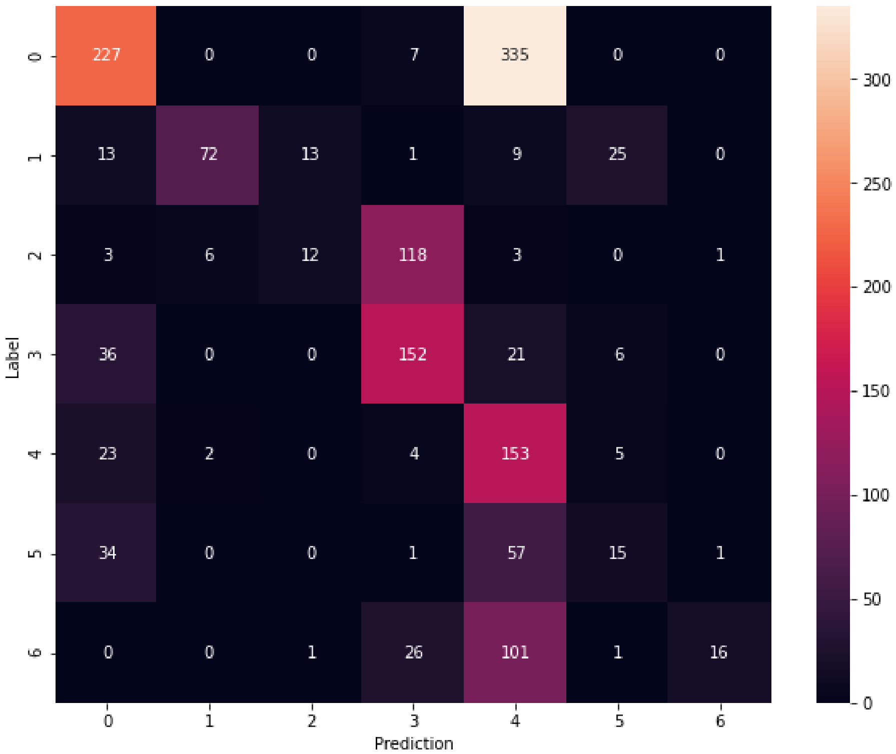
The model is evaluated on the single seat grayscale test images from each unknown vehicle (aclass, escape, gsf, i3, tiguan, tucson and x5) and plot the confusion matrix.

```python
# path to the riginal SVIRO datase containing the grayscale single seat images
# of each vehicle
original_dataset_path = '{}/datasets/test/grayscale_single_seat'.format(MAIN_PATH)

vehicles_to_test = ['aclass','escape', 'gsf', 'i3','tiguan', 'tucson','x5']

cars_accuracy = {}
for vehicle in vehicles_to_test:

    test_dataset_path = original_dataset_path + \
                        "/{}/test_with_labels/grayscale".format(vehicle)

    test_dataset = image_dataset_from_directory(test_dataset_path,
                                                validation_split=None,
                                                subset=None,
                                                image_size=(IMAGE_HIGHT, IMAGE_WIDTH),
                                                color_mode="rgb",
                                                batch_size=1)

    # get the true values and the predicted values in all the images from the test dataset
    y_pred = []
    y_true = []
    for image, label in test_dataset:
        y_pred.append(int(np.argmax(model.predict(image), axis=1)))
        y_true.append(int(label))

    sum = 0
    for i in range(len(y_pred)):
        if y_pred[i] == y_true[i]:
            sum +=1

    test_accuracy = sum / len(y_true)

    cars_accuracy.update({vehicle: test_accuracy})

    print('Test set accuracy for {}: {}\n'.format(vehicle, test_accuracy))
    print(CLASSES.values)

    # plot the confusion matrix
    confusion_mtx = tf.math.confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(10, 8))
    sns.heatmap(confusion_mtx, xticklabels=CLASSES, yticklabels=CLASSES,
                annot=True, fmt='g')
    plt.xlabel('Prediction')
    plt.ylabel('Label')
    plt.show()

mean_accuracy = np.mean(list(cars_accuracy.values()))
print(cars_accuracy)
print("mean accuracy: {}".format(mean_accuracy))
```

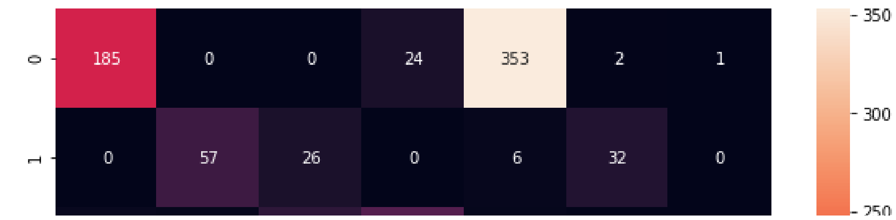
```
Found 1500 files belonging to 7 classes.
Test set accuracy for aclass: 0.544

<built-in method values of dict object at 0x7f4414d10870>
```

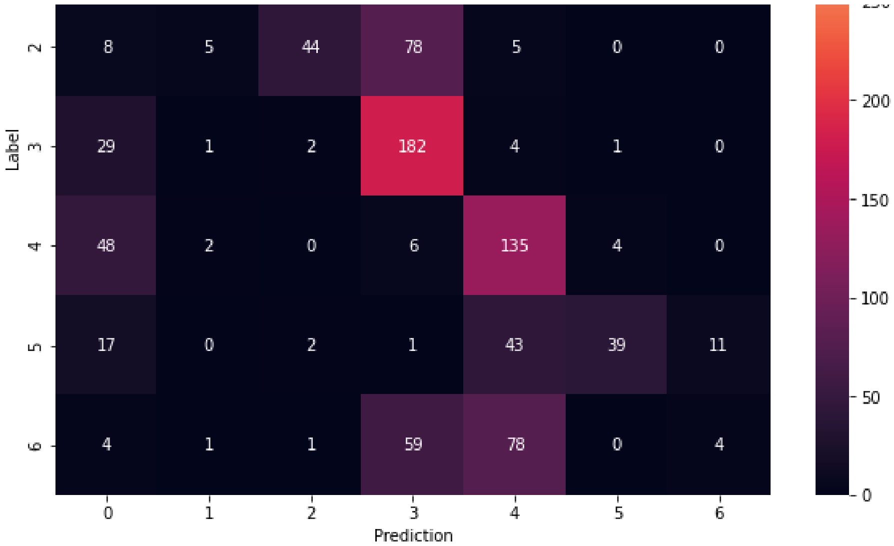



```
Found 1500 files belonging to 7 classes.
Test set accuracy for escape: 0.43133333333333335

<built-in method values of dict object at 0x7f4414d10870>
```

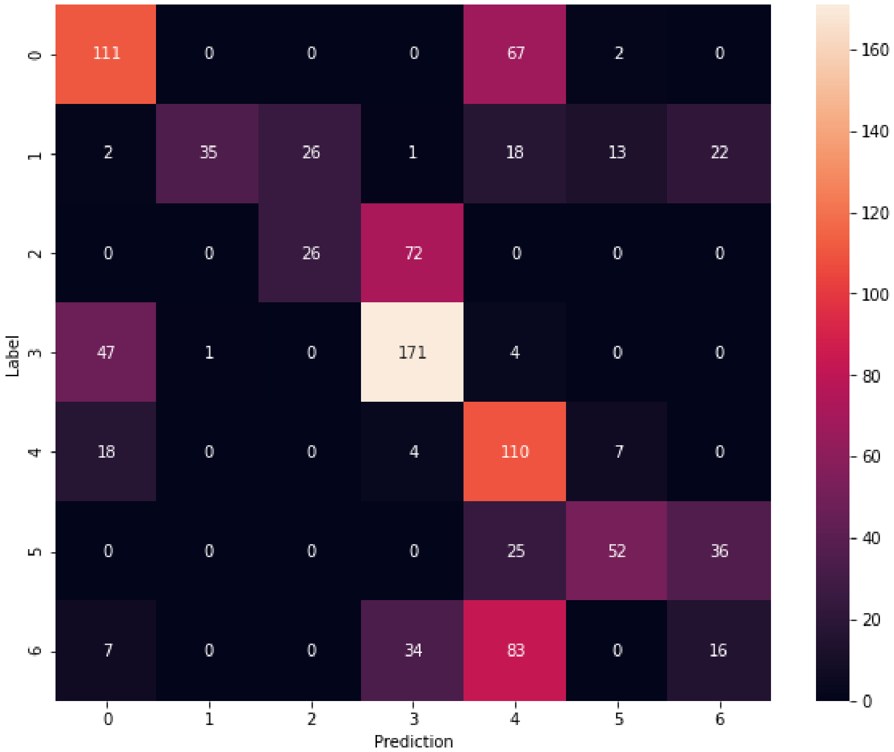



```
Found 1500 files belonging to 7 classes.
Test set accuracy for gsf: 0.43066666666666664

<built-in method values of dict object at 0x7f4414d10870>
```

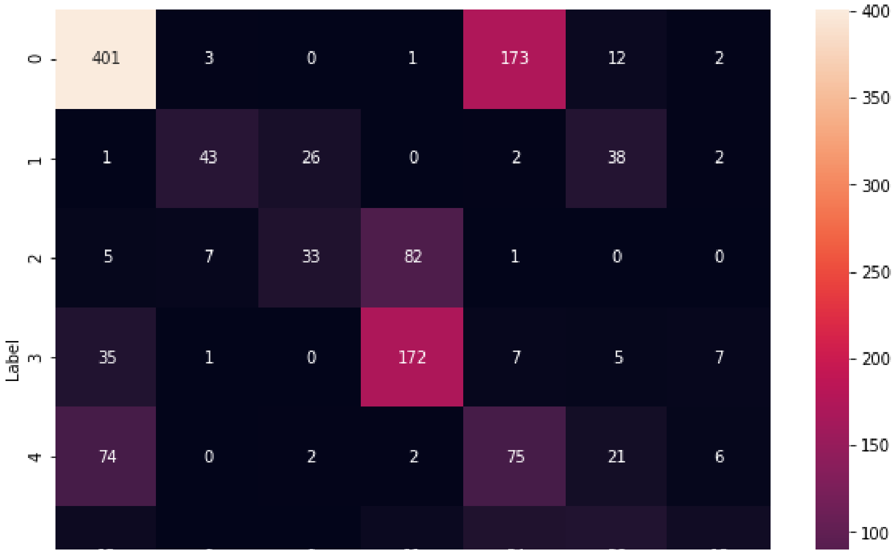


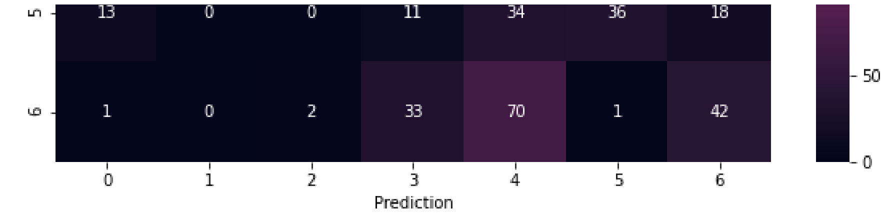



```
Found 1010 files belonging to 7 classes.
Test set accuracy for i3: 0.5158415841584159

&lt;built-in method values of dict object at 0x7f4414d10870&gt;
```



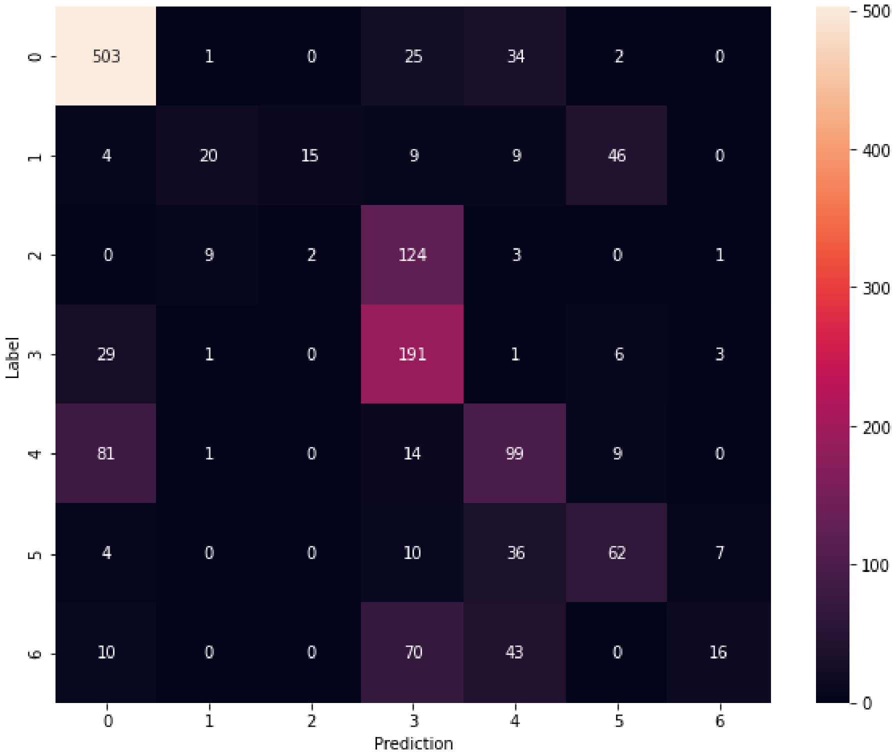

```
Found 1500 files belonging to 7 classes.
Test set accuracy for tiguan: 0.5346666666666666

&lt;built-in method values of dict object at 0x7f4414d10870&gt;
```





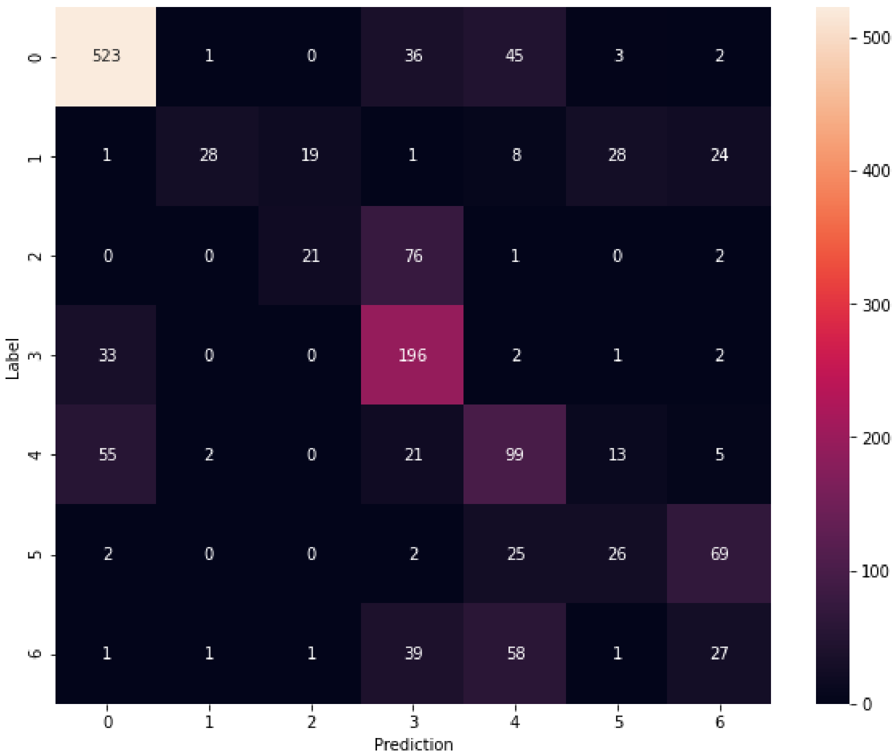
Found 1500 files belonging to 7 classes.
Test set accuracy for tucson: 0.5953333333333334

<built-in method values of dict object at 0x7f4414d10870>



Found 1500 files belonging to 7 classes.
Test set accuracy for x5: 0.6133333333333333

<built-in method values of dict object at 0x7f4414d10870>



{'aclass': 0.544, 'escape': 0.43133333333333335, 'gsf': 0.43066666666666664, 'i3': 0.5158415841584159, 'tigu
mean accuracy: 0.5235964167845356

Testing on back seat images

```
import cv2
import matplotlib.pyplot as plt
import tensorflow as tf

SEAT_HIGHT = 550
SEAT_WIDTH = 250
SEAT_SIZE = (SEAT_HIGHT, SEAT_WIDTH)

Y_START = [70, 70, 70]
X_START = [130, 364, 582]


def crop_seats_from_image (image, y_start, x_start, seat_size):
    croped_seats = []
    seat_hight = seat_size[0]
    seat_width = seat_size[1]
    for x, y in zip(x_start, y_start):
        croped_seats.append(image[y:y+seat_hight, x:x+seat_width])

    return croped_seats


def anotate_classes_on_image (image, class_texts, anotate_points):
    for text, point in zip(class_texts, anotate_points):
        print("")
        cv2.putText(image,text,point, cv2.FONT_HERSHEY_SIMPLEX,0.7,(0,255.0),2)

    return image


def classify_seats (seat_images, model):
    class_texts = []
    class_labels = []

    for seat_image in seat_images:

        seat_image_reshaped = cv2.resize(seat_image,(IMAGE_WIDTH,IMAGE_HIGHT), cv2.INTER_LINEAR)
        seat_image_reshaped = tf.expand_dims(seat_image_reshaped, 0)

        prediction = model.predict(seat_image_reshaped)
        class_label = int(np.argmax(prediction, axis=1))

        class_labels.append(class_label)
        class_texts.append(CLASSES[class_label])

    return class_labels, class_texts


# load images from vehicle

SAMPLES = 1
VEHICLE = "escape"
test_images_path = "{}/datasets/test/grayscale/{}/test_with_labels/grayscale_wholeImage".format(MAIN_PATH, VEH

# load images
image_files = [f for f in listdir(test_images_path) if
                    path.isfile(path.join(test_images_path, f)) and f.endswith(".png")]

image_file_samples = random.sample(image_files, SAMPLES)

# Do the inference and show the seat classification
for image_file in image_file_samples:
    image = cv2.imread(test_images_path + '/' + image_file)
```

```
croped_seat_images = crop_seats_from_image(image, Y_START, X_START, SEAT_SIZE)

seat_labels, seat_class_texts  = classify_seats(croped_seat_images, model)

anotation_points = [(X_START[0],Y_START[0]),
                    (X_START[1]+10,Y_START[1]),
                    (X_START[2]+10,Y_START[2])]

anotetated_image = anotate_classes_on_image(image,seat_class_texts,anotation_points)

plt.figure(figsize=(12,12))
plt.imshow(anotetated_image)
```

Conclusion and Future Work

After training and evaluating the model we can see that the results obaitained with the train dataset with different vihicles, are verry similar to the average accuracy foud by the SVIRO team in their benchmark on classifiers, arround 50% when evaluated on unknown vehicles. One of the reasons why the results are low, for unknown vehicles, is that the model lean very easelly the texture of the objects in the train dataset and overfits. Another problem is that in the croped image the objects apear to be verry big, making the ectraction of the relevant features harder. Besudes that, lokking at the confuion matrixes we can see that there is a lot of missclassifications between "everyday objects" and "Empty Seats", probably because in many pictures, parts of objects and people from de adjacent seats apear on the image.

Nevertheless, this project was a great opportunity to understande the problem and the limits of the image clasification approach. There is other solutions to explore in order to have a better vision of the objects, using the hole image, as object detection and segmantation.

✓ 4s completed at 1:07 AM