Great learning – Capstone project

Car Detection - Automotive Surveillance.

Design a DL based car identification model with Bounding box

Done by,

Varun Sachar

Kanaghappriya karunanithi

Pankaj

Shreekant Singh

Batch – 2022 to 2023

Contents

Problem Statement

DOMAIN - Automotive Surveillance.

PROJECT OBJECTIVE: Design a DL based car identification model.

We are designing a deep learning model which can identify the Model Name of the car in the given picture along with its bounding box using CNN algorithm.

USE CASE OF THE PROJECT:

Computer vision can be used to automate supervision and generate action appropriate action trigger if the event is predicted from the image of interest.

For example : a car moving on the road can be easily identified by a camera as make of the car, type, colour, number plates etc.

DATA DESCRIPTION:

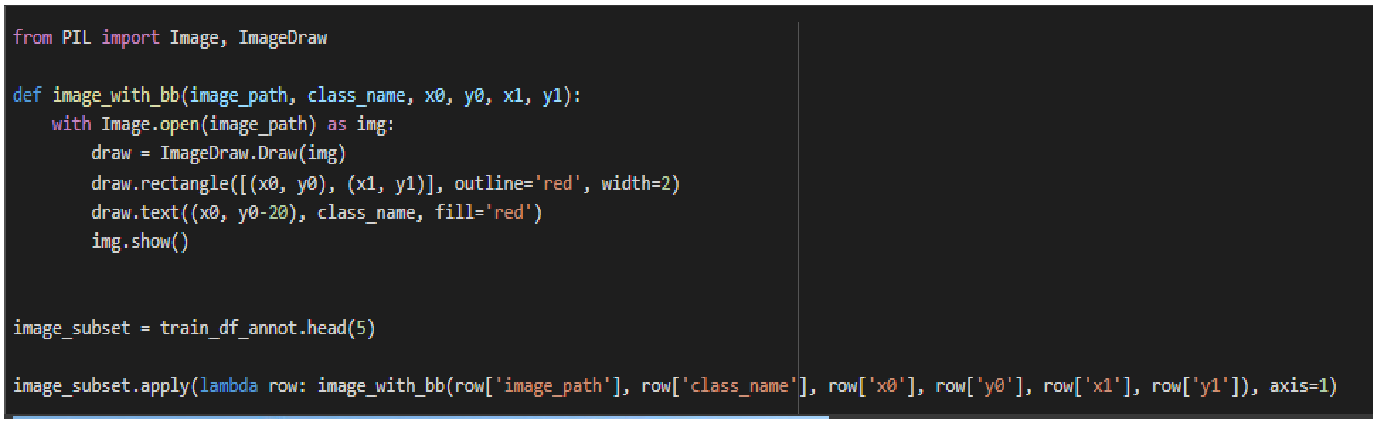
The Cars dataset contains 16,185 images of 196 classes of cars. The data is split into 8,144 training images and 8,041 testing images, where each class has been split roughly in a 50-50 split. Classes are typically at the level of Make, Model, Year, Example: 2012 Tesla Model S or 2012 BMW M3 coupe.

- ‣ Train Images: Consists of real images of cars as per the make and year of the car.
- ‣ Test Images: Consists of real images of cars as per the make and year of the car.
- ‣ Train Annotation: Consists of bounding box region for training images.
- ‣ Test Annotation: Consists of bounding box region for testing images.

Some of the car label names that we have in our dataset:

- Toyota Sequoia SUV 2012
- Volkswagen Golf Hatchback 1991
- Volkswagen Golf Hatchback 2012
- Suzuki SX4 Sedan 2012
- Volvo 240 Sedan 1993
- Land Rover Range Rover SUV 2012
- Spyker C8 Convertible 2009
- Isuzu Ascender SUV 2008

The below code displays the Samples images with bounding box and its Label:

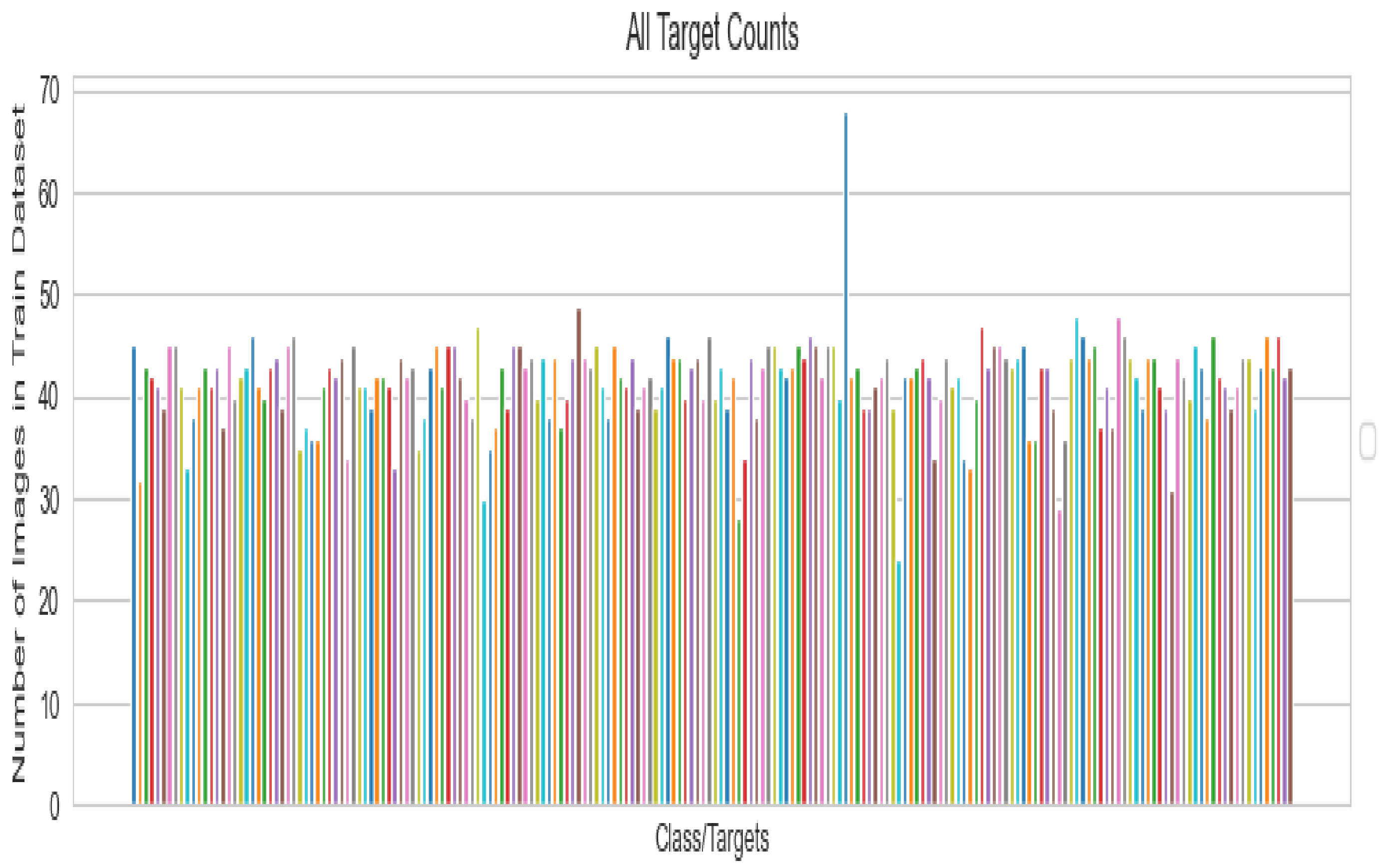


```
from PIL import Image, ImageDraw

def image_with_bb(image_path, class_name, x0, y0, x1, y1):
    with Image.open(image_path) as img:
        draw = ImageDraw.Draw(img)
        draw.rectangle([(x0, y0), (x1, y1)], outline='red', width=2)
        draw.text((x0, y0-20), class_name, fill='red')
        img.show()


image_subset = train_df_annot.head(5)

image_subset.apply(lambda row: image_with_bb(row['image_path'], row['class_name'], row['x0'], row['y0'], row['x1'], row['y1']), axis=1)
```

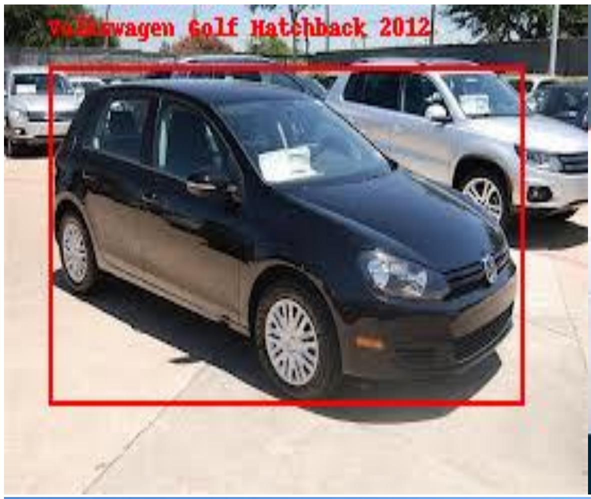
- The function "image_with_bb" in the above code takes in path of the image, label names, and bounding box co-ordinates – X_max, X_min, Y_max, Y_min and displays the car images with its bbox along with the label names.
- Invoking the function on few samples of the train dataset. The train dataset contains all the information required by the function.

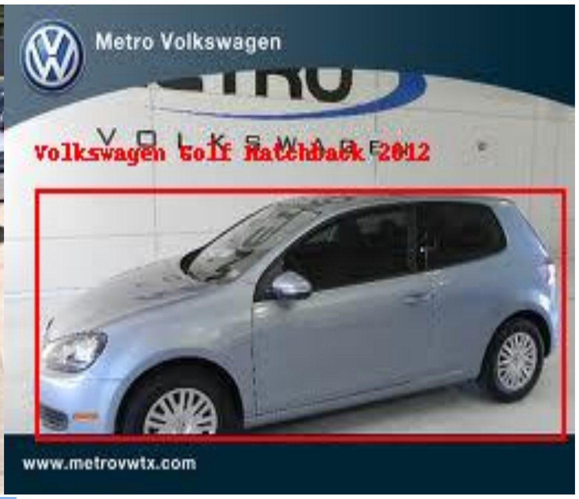
The below figure represents the No of samples in each class in the Training dataset :

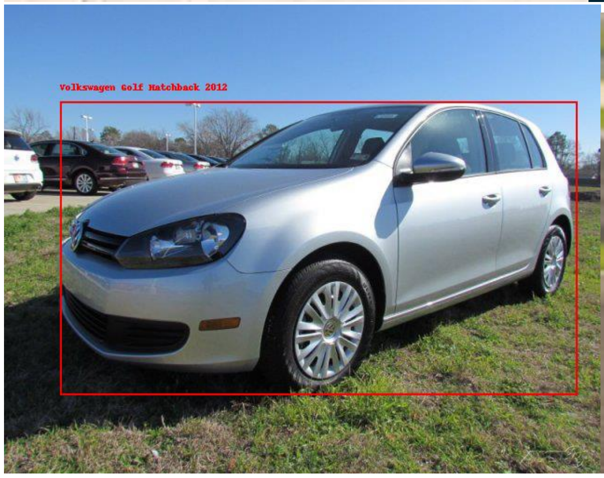
The chart clearly shows that the number of images in each class is almost the same. In this case we can say that the data is pretty evenly balanced.

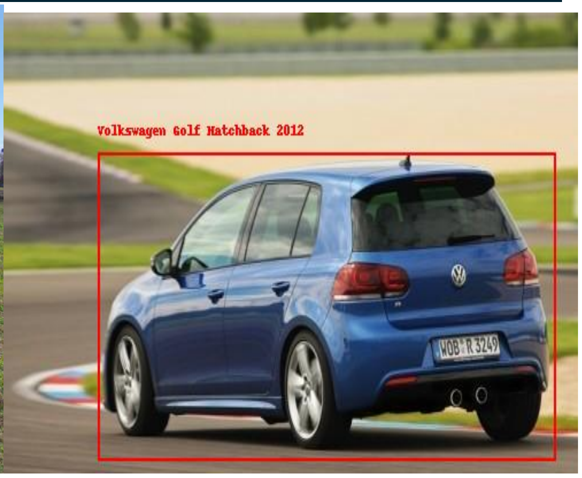

All Target Counts
Number of Images in Train Dataset
Class/Targets
0
10
20
30
40
50
60
70

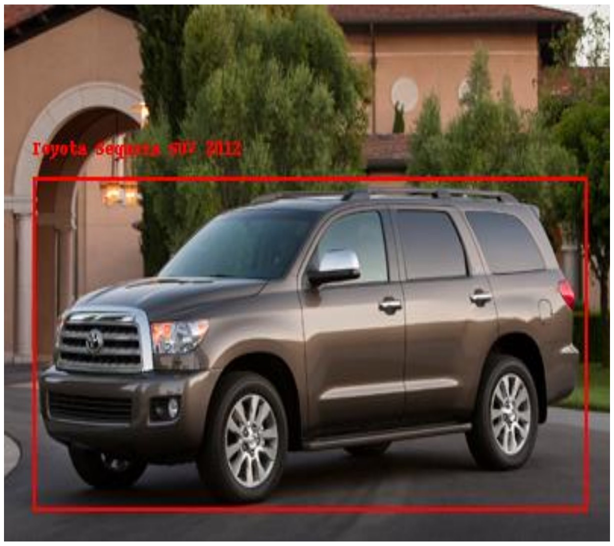
Samples images with Label and bounding box:

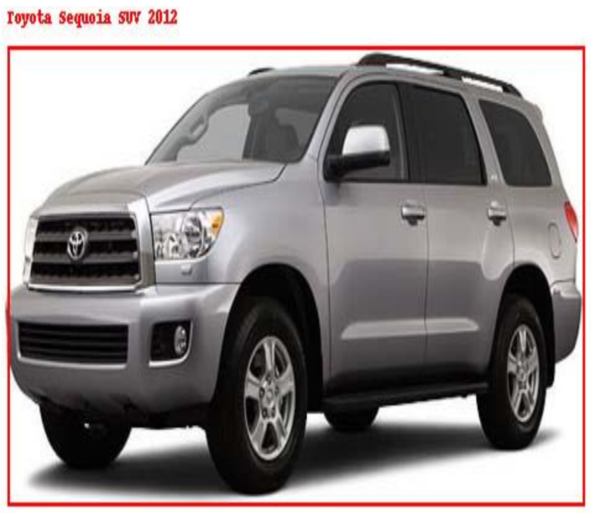






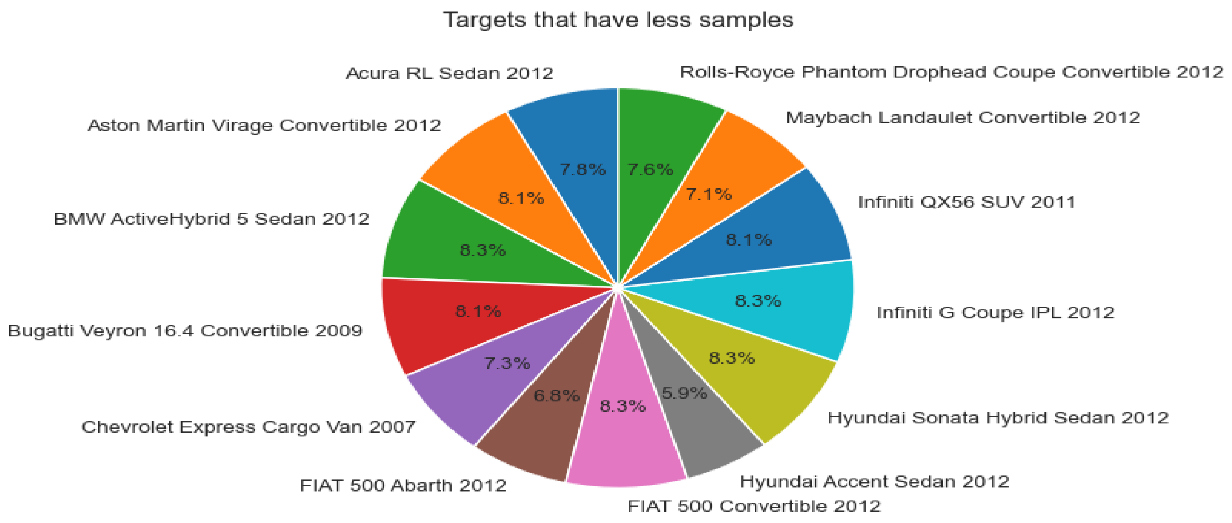






Classes containing least images:

Most of the Target class seem to have atleast 35 or 40 + images for the training part, very few Target class seem to contain less than 35 images. Listing out those images in a pie chat format for our analysis.



Data Augmentation:

- As we have slight imbalance in the data, as well as we have very few samples (average of 40-45 images in each class) , we will have to do data augmentation in order to overcome any underfit.
- More the samples, better the training and accuracy.
- Data augmentation is a technique of artificially increasing the training set by creating modified copies of a dataset using existing data. It includes making minor changes to the dataset, we make geometric and color space transformations (flipping, resizing, cropping, brightness, contrast) to increase the size and diversity of the training set or using deep learning to generate new data points.

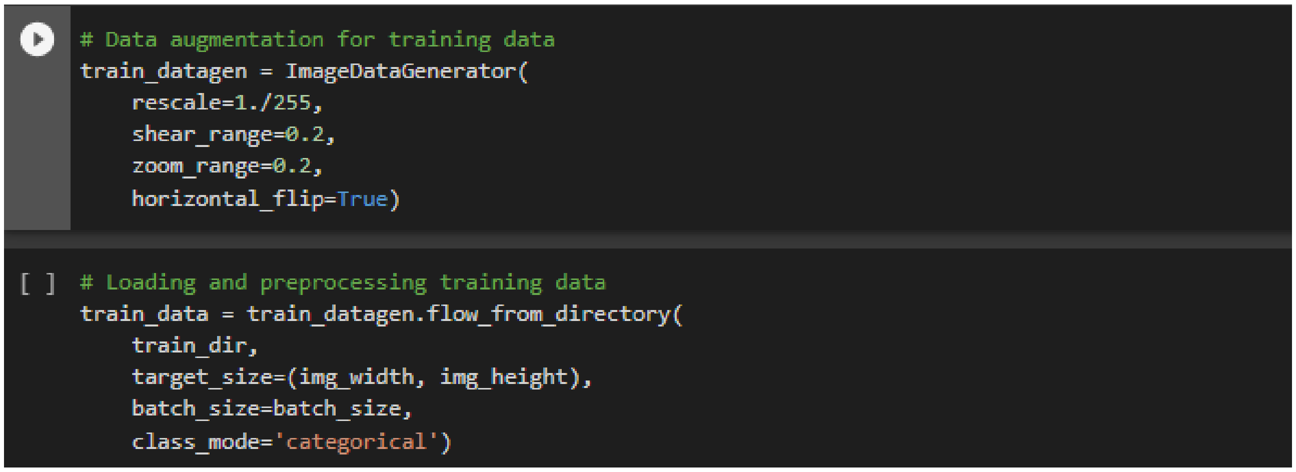
When Should You Use Data Augmentation?

1. To prevent models from overfitting.
2. The initial training set is too small.
3. To improve the model accuracy.
4. To Reduce the operational cost of labeling and cleaning the raw dataset.

In our code, we have done only **Geometric augmentation**

Geometric transformations: randomly flip, crop, rotate, stretch, and zoom images.

Thankfully tensorflow.keras.preprocessing.image import ImageDataGenerator helps us make the process easier.



```
# Data augmentation for training data
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)
```

```
[ ] # Loading and preprocessing training data
train_data = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')
```

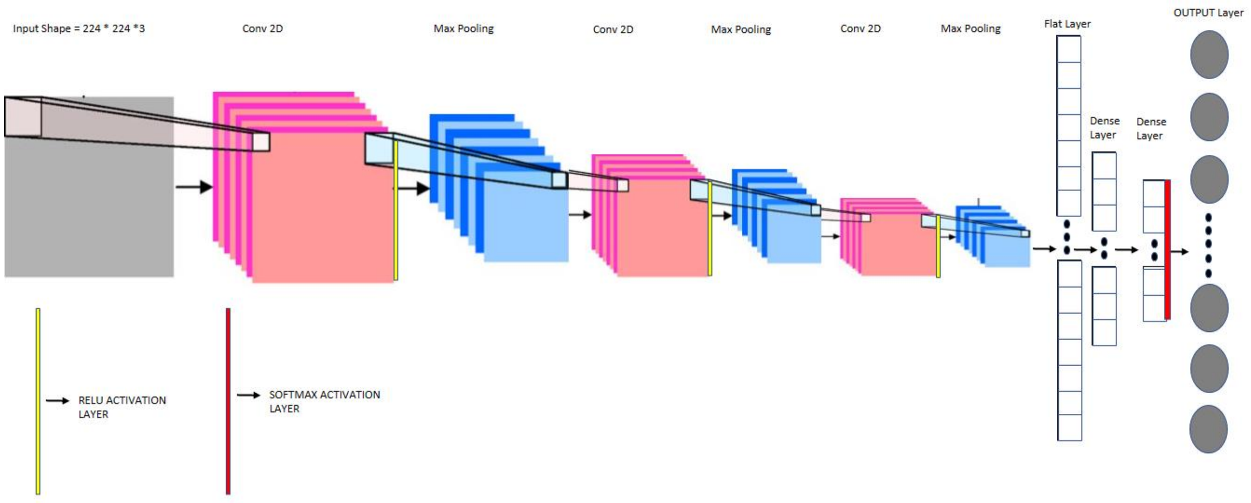
Explanation of the code:

Parameters in ImageDataGenerator:

- Shear_range : Shear angle in counter-clockwise direction in degrees
- Rescaling : Rescaling is the process of normalizing the pixel values of an image. Rescaling is typically done by dividing each pixel value by a constant value, such as 255, which is the maximum value that a pixel can take in an 8-bit grayscale image. Rescaling have maintain the width to height ratio. Hence, rescaling do not cause distortion or skewing. The rescale parameter in ImageDataGenerator allows you to specify the rescaling factor for the pixel values in your image data.
- Zoom_range: if the zoom range is set to (0.8, 1.2), a randomly chosen zoom level of 0.9 would mean that the image is zoomed out by 10% (i.e., displayed at 90% of its original size), while a zoom level of 1.1 would mean that the image is zoomed in by 10% (i.e., displayed at 110% of its original size).
- Horizontal_flip : flips the image horizontally.

MODEL BUILDING AND COMPILATION

```
# Defining the CNN model architecture

model = tf.keras.Sequential([

    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(img_width, img_height, 3)),

    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),

    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(128, activation='relu'),

    tf.keras.layers.Dense(196, activation='softmax')

])


# Compiling model

model.compile(optimizer='adam',

              loss='categorical_crossentropy',

              metrics=['accuracy'])
```



We have used a basic CNN model which has the following components and parameters:

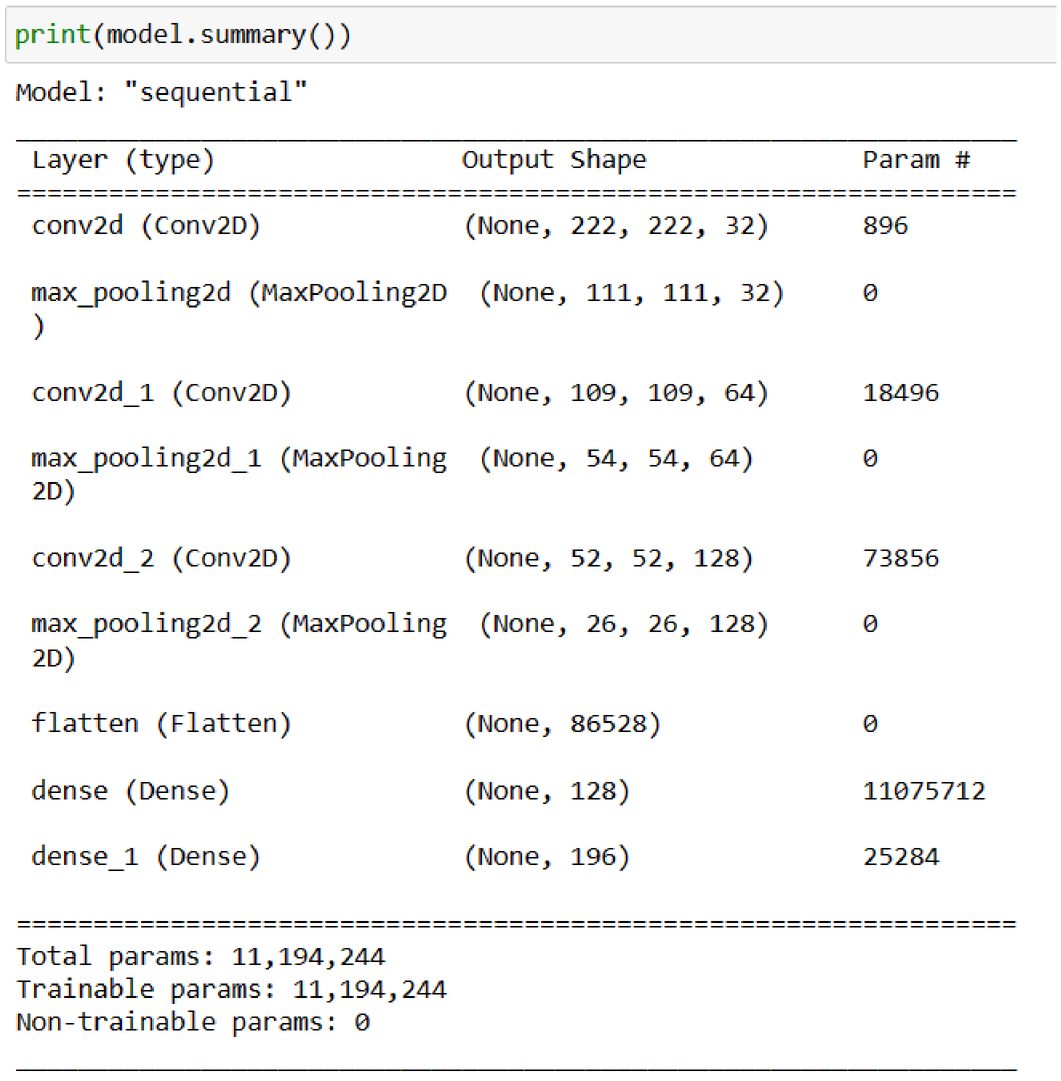
 * As we see in the above figure our model consists of 3 Conv 2D layer, 3 max pooling layer and 3 flat layers.
* The "Adam optimizer" is used because usually it's the most efficient among the algorithm available benefits of using Adam on non-convex optimization problems, as follows:

- ➢ Straightforward to implement.
- ➢ Computationally efficient.
- ➢ Little memory requirements.
- ➢ Invariant to diagonal rescale of the gradients.
- ➢ Well suited for problems that are large in terms of data and/or parameters.
- ➢ Appropriate for non-stationary objectives.
- ➢ Appropriate for problems with very noisy/or sparse gradients.
- ➢ Hyper-parameters have intuitive interpretation and typically require little tuning.

* Default learning rate of 0.01 is used in the model

*This model has been trained for 10 epochs and accuracy is plotted

Model Summary:



```
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 111, 111, 32)     0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 54, 54, 64)       0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 52, 52, 128)       73856     
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 26, 26, 128)      0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 86528)             0         
                                                                 
 dense (Dense)               (None, 128)               11075712  
                                                                 
 dense_1 (Dense)             (None, 196)               25284     
                                                                 
=================================================================
Total params: 11,194,244
Trainable params: 11,194,244
Non-trainable params: 0
_________________________________________________________________
```

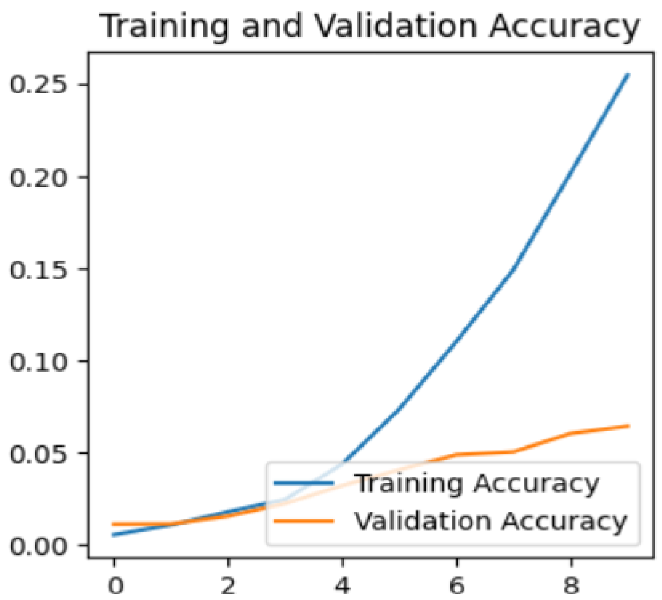
The number of weights that gets updated during training = 11194244

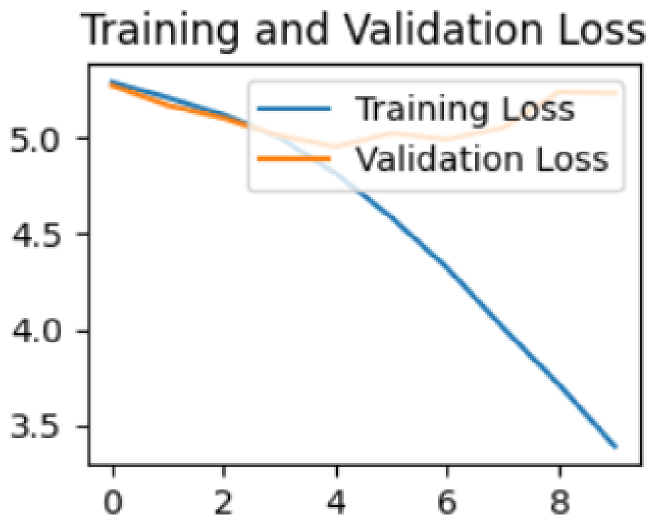
RESULTS

Test accuracy: 6.45442083477974 %

Model accuracy if the model was random selecting an image from the classes ≅1/number of classes *100 = 1/196 *100 = 0.510204082 %

This means we are doing at least 10 times better than guessing the model but there is still a lot of scope for improvements



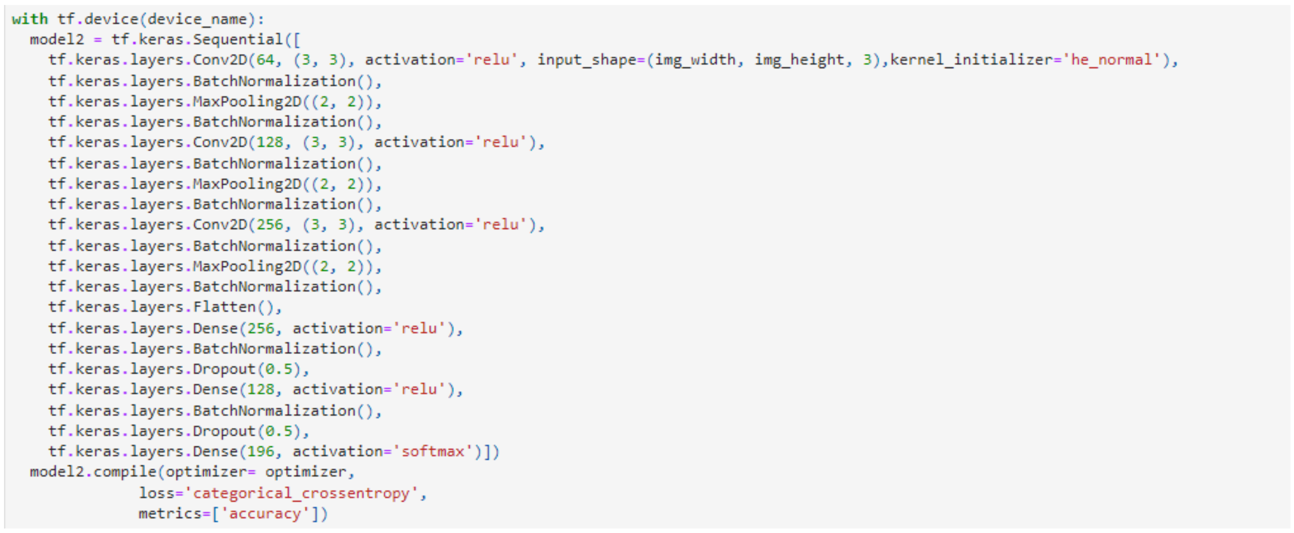


The graph deviates after 3 epochs when the accuracy is 2.5% and the loss is around 4.85 the rate of increase of accuracy of training data is much higher than validation accuracy. This is clearly a case of ***overfitting.***

NOTE *:-* Further in the project we will try to improve the accuracy by working on the hyper parameter tuning and applying transfer learning.

Fine tuning the Basic CNN model

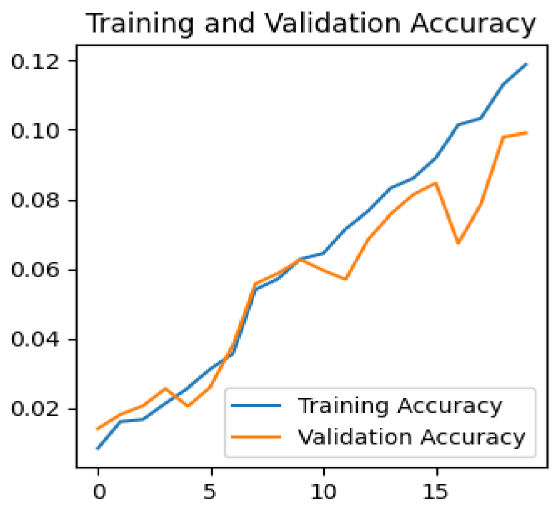
- In order to fine tune the previous model, we have changed the architecture a little bit and we have used two different callbacks in Keras, namely *EarlyStopping and ReduceLROnPlateau.*
- *EarlyStopping* is a callback that can be used to stop the training of a model when a monitored metric, specified by the 'monitor' argument, has stopped improving. In this case, the monitored metric is the validation loss ('val_loss'), and the training will stop if the validation loss does not improve for 'patience' number of epochs. The 'patience' argument specifies how many epochs the training will continue without improvement before EarlyStopping stops the training. The 'verbose' argument controls the verbosity of the output.
- *ReduceLROnPlateau* is a callback that can be used to dynamically adjust the learning rate of the optimizer during training. It monitors a specified metric, in this case the validation loss ('val_loss'), and when the monitored metric stops improving for 'patience' number of epochs, the learning rate is reduced by a factor of 'factor'. This can help the optimizer converge more smoothly to the optimal solution. The 'verbose' argument controls the verbosity of the output.
- The optimizer used in this code is the *Adam optimizer* with a learning rate of 0.01.
- Model architecture has been changed and it is as follows

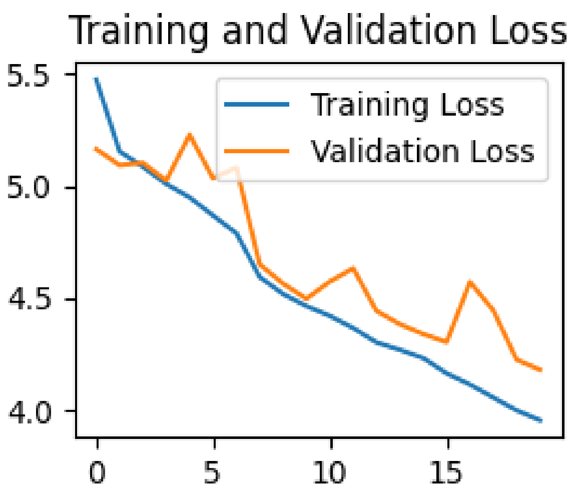


```
with tf.device(device_name):
  model2 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', input_shape=(img_width, img_height, 3),kernel_initializer='he_normal'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(256, (3, 3), activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(196, activation='softmax')])
  model2.compile(optimizer= optimizer,
            loss='categorical_crossentropy',
            metrics=['accuracy'])
```

Performance of the model:

```
Epoch 20/20
255/255 [==============================] - 50s 192ms/step - loss: 3.9566 - accur
acy: 0.1189 - val_loss: 4.1812 - val_accuracy: 0.0991 - lr: 1.0000e-03
```





Observations:

The model doesn't show much improvement even after changing the model and early stopping the epochs based on learning , so employing other algorithms like transfer learning,RCNN could be a the right thing to in the upcoming steps.

Transfer learning:

Since with a traditional CNN model, we are getting a validation accuracy around 6% which is nowhere close to the efficiency we would like to see in our model.

Transfer learning is a machine learning (ML) method that reuses a trained model designed for a particular task to accomplish a different yet related task. The knowledge acquired from task one is thereby transferred to the second model that focuses on the new task.

Transfer learning speeds up the overall process of training a new model and consequently improves its performance. It is primarily used when a model requires large amount of resources and time for training. Due to these reasons, transfer learning is employed in several deep learning projects, such as neural networks that accomplish NLP or CV tasks, such as sentiment analysis.

Transfer learning can accomplish tasks by undertaking different approaches.

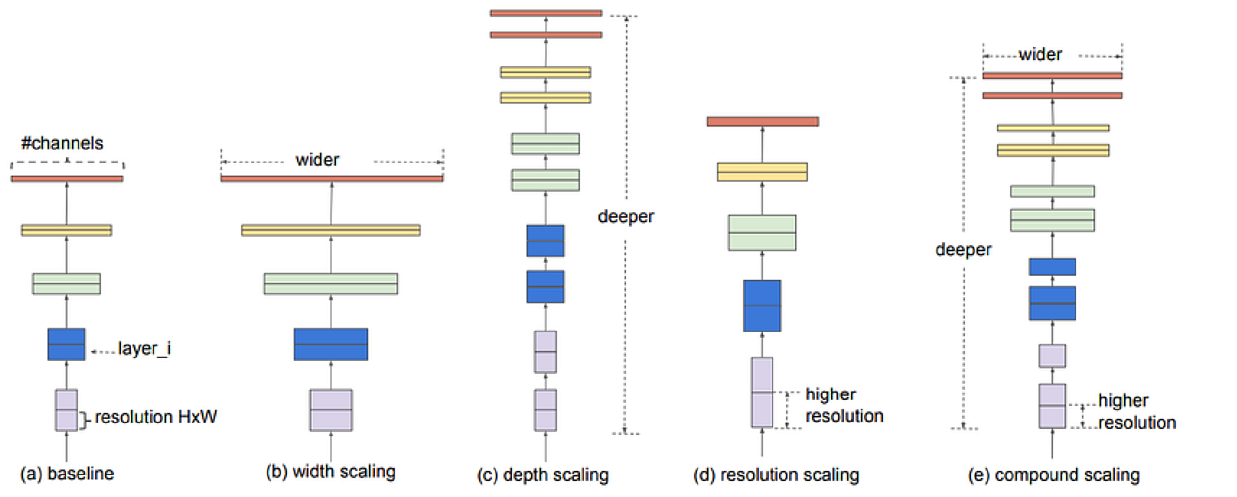
1. Train 'similar domain' models
2. Extract features
3. Use pre-trained models

The following models are used for our project

EfficientNet B1

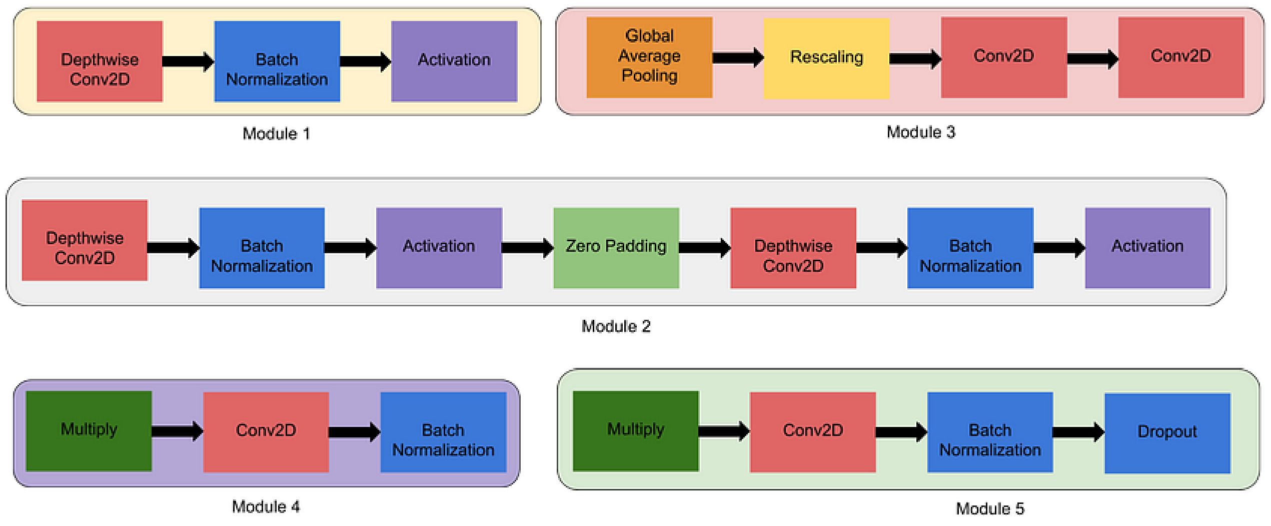
EfficientNets rely on AutoML and compound scaling to achieve superior performance without compromising resource efficiency. The AutoML Mobile framework has helped develop a mobile-size baseline network, EfficientNet-B0, which is then improved by the compound scaling method to obtain EfficientNet-B1 to B7. The EfficientNet we have chosen to train our model is B1.

Generally, the models are made too wide, deep, or with a very high resolution. Increasing these characteristics helps the model initially but it quickly saturates and the model made just has more parameters and is therefore not efficient. In EfficientNet they are scaled in a more principled way i.e. gradually everything is increased.



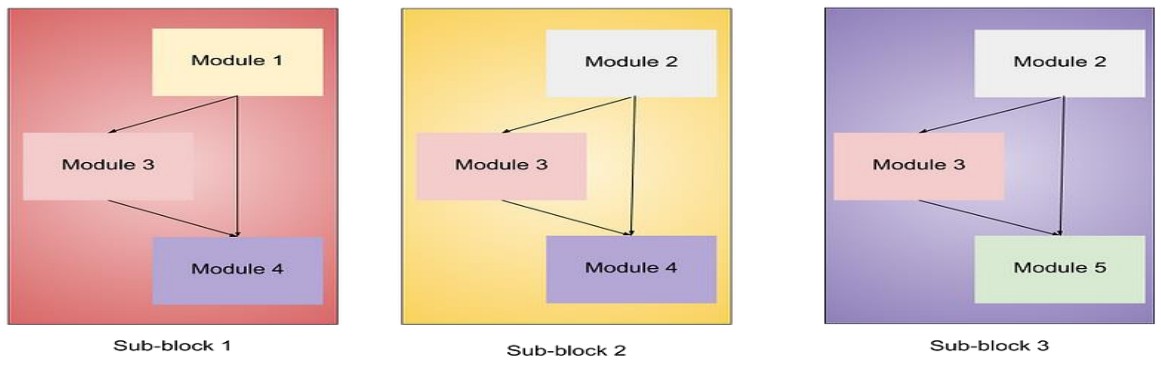
Architecture

5 modules we will use to make the architecture.

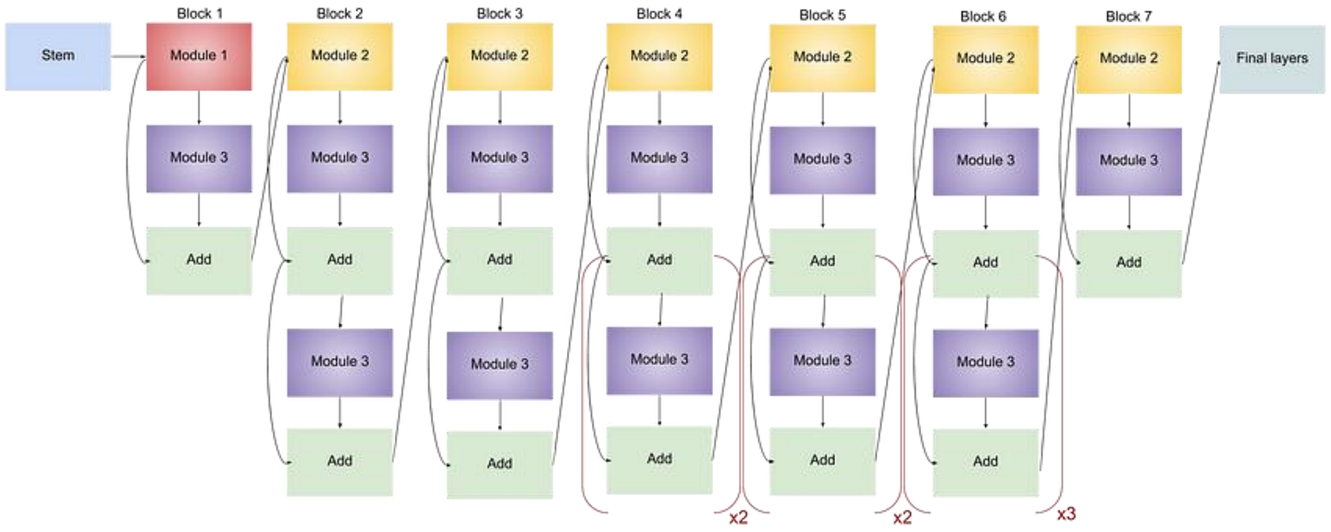


- **Module 1** — This is used as a starting point for the sub-blocks.
- **Module 2** — This is used as a starting point for the first sub-block of all the 7 main blocks except the 1st one.
- **Module 3** — This is connected as a skip connection to all the sub-blocks.
- **Module 4** — This is used for combining the skip connection in the first sub-blocks.
- **Module 5** — Each sub-block is connected to its previous sub-block in a skip connection and they are combined using this module.

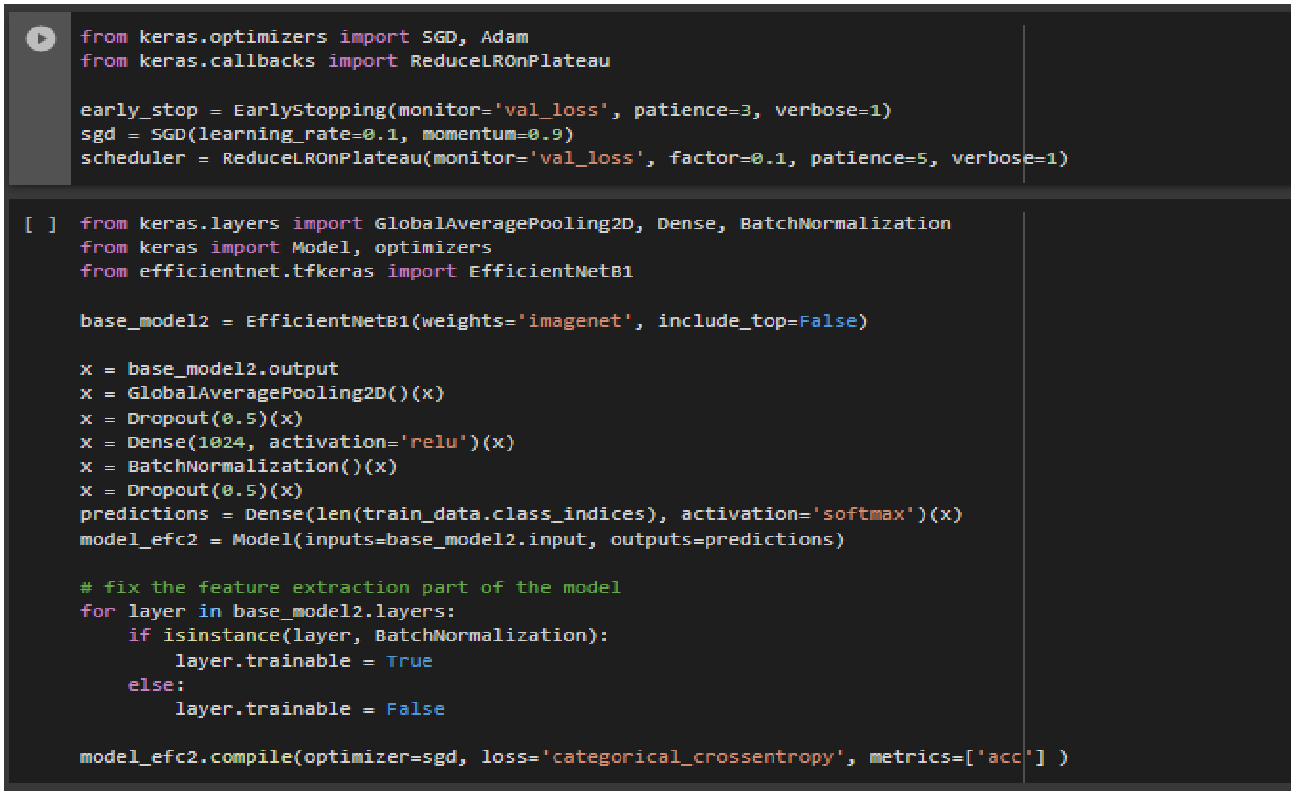
These modules are further combined to form sub-blocks which will be used in a certain way in the blocks.



- **Sub-block 1 —** This is used only used as the first sub-block in the first block.
- **Sub-block 2 —** This is used as the first sub-block in all the other blocks.
- **Sub-block 3 —** This is used for any sub-block except the first one in all the blocks.



Code and explanation:

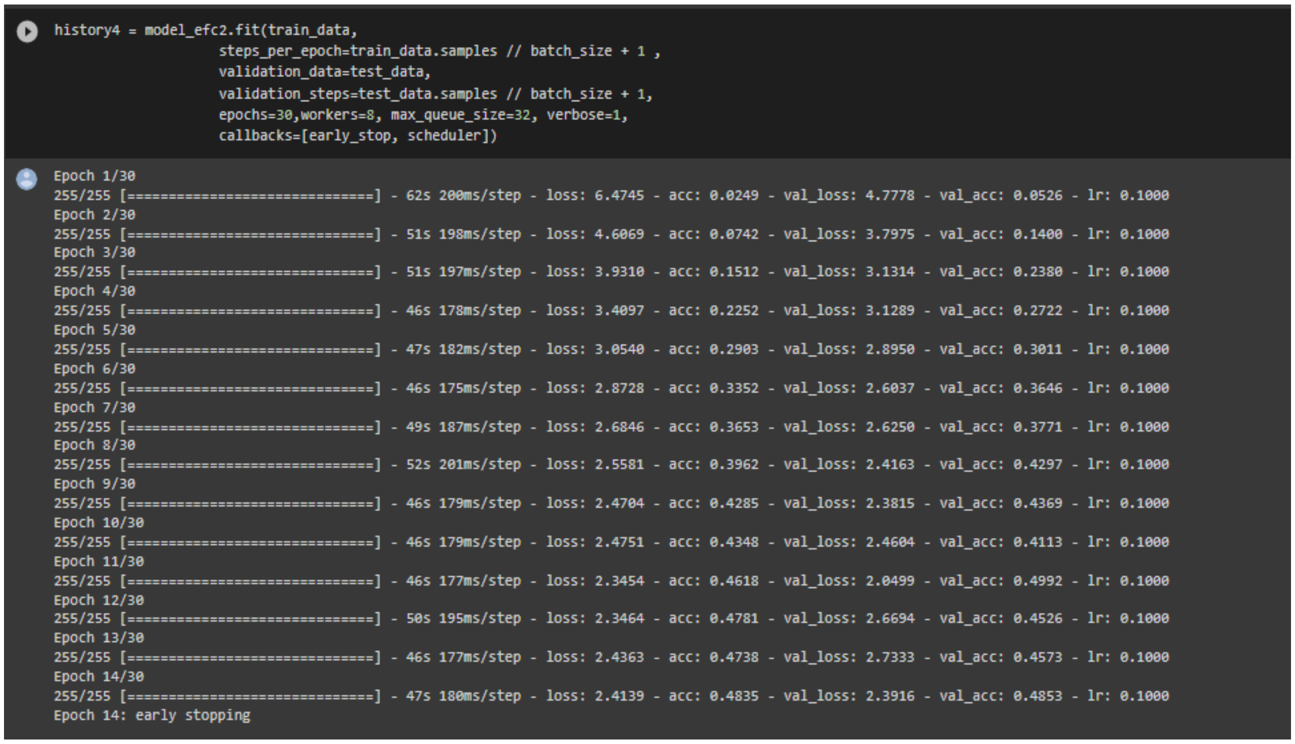


```
from keras.optimizers import SGD, Adam
from keras.callbacks import ReduceLROnPlateau

early_stop = EarlyStopping(monitor='val_loss', patience=3, verbose=1)
sgd = SGD(learning_rate=0.1, momentum=0.9)
scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5, verbose=1)
```

```
[ ] from keras.layers import GlobalAveragePooling2D, Dense, BatchNormalization
    from keras import Model, optimizers
    from efficientnet.tfkeras import EfficientNetB1

    base_model2 = EfficientNetB1(weights='imagenet', include_top=False)

    x = base_model2.output
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.5)(x)
    x = Dense(1024, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)
    predictions = Dense(len(train_data.class_indices), activation='softmax')(x)
    model_efc2 = Model(inputs=base_model2.input, outputs=predictions)

    # fix the feature extraction part of the model
    for layer in base_model2.layers:
        if isinstance(layer, BatchNormalization):
            layer.trainable = True
        else:
            layer.trainable = False

    model_efc2.compile(optimizer=sgd, loss='categorical_crossentropy', metrics=['acc'] )
```

- Model: we have chosen the weights of imagenet, and finetune the model with some customization according to our data and need.
- Optimizer is chosen to be SGD and we have categorical_crossentropy as we have classification problem with softmax as activation layer. The output from softmax will be binary indicating only the predicted class index as 1.
- In order to reduce the overfitting, Global average pooling layers, batch normalization and dropout layers with 50% are used on top of downloaded Effnet model. **Note:** Dropout layers doesnt reduce the output to 50% instead makes 50% of its output to zero.
- Learning rate has been used along with the scheduler in order to monitor the performance.

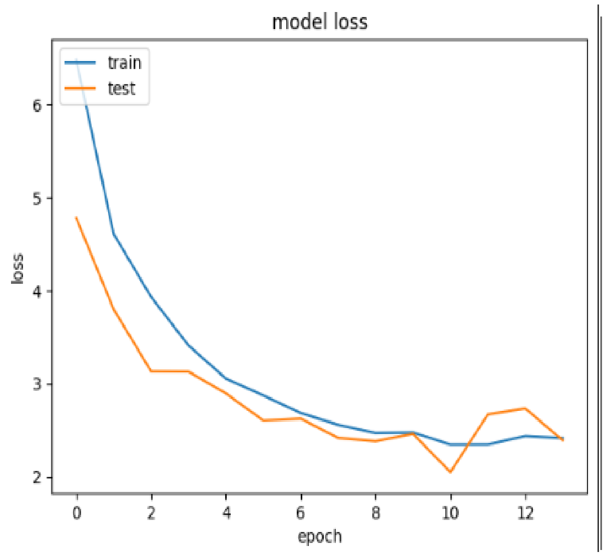
Scheduler : Reduce learning rate when a metric has stopped improving.Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

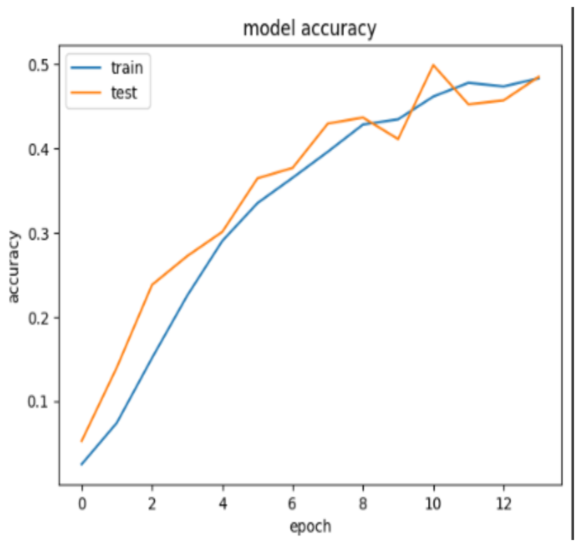


```
history4 = model_efc2.fit(train_data,
                    steps_per_epoch=train_data.samples // batch_size + 1 ,
                    validation_data=test_data,
                    validation_steps=test_data.samples // batch_size + 1,
                    epochs=30,workers=8, max_queue_size=32, verbose=1,
                    callbacks=[early_stop, scheduler])
```

```
Epoch 1/30
255/255 [==============================] - 62s 200ms/step - loss: 6.4745 - acc: 0.0249 - val_loss: 4.7778 - val_acc: 0.0526 - lr: 0.1000
Epoch 2/30
255/255 [==============================] - 51s 198ms/step - loss: 4.6069 - acc: 0.0742 - val_loss: 3.7975 - val_acc: 0.1400 - lr: 0.1000
Epoch 3/30
255/255 [==============================] - 51s 197ms/step - loss: 3.9310 - acc: 0.1512 - val_loss: 3.1314 - val_acc: 0.2380 - lr: 0.1000
Epoch 4/30
255/255 [==============================] - 46s 178ms/step - loss: 3.4097 - acc: 0.2252 - val_loss: 3.1289 - val_acc: 0.2722 - lr: 0.1000
Epoch 5/30
255/255 [==============================] - 47s 182ms/step - loss: 3.0540 - acc: 0.2903 - val_loss: 2.8950 - val_acc: 0.3011 - lr: 0.1000
Epoch 6/30
255/255 [==============================] - 46s 175ms/step - loss: 2.8728 - acc: 0.3352 - val_loss: 2.6037 - val_acc: 0.3646 - lr: 0.1000
Epoch 7/30
255/255 [==============================] - 49s 187ms/step - loss: 2.6846 - acc: 0.3653 - val_loss: 2.6250 - val_acc: 0.3771 - lr: 0.1000
Epoch 8/30
255/255 [==============================] - 52s 201ms/step - loss: 2.5581 - acc: 0.3962 - val_loss: 2.4163 - val_acc: 0.4297 - lr: 0.1000
Epoch 9/30
255/255 [==============================] - 46s 179ms/step - loss: 2.4704 - acc: 0.4285 - val_loss: 2.3815 - val_acc: 0.4369 - lr: 0.1000
Epoch 10/30
255/255 [==============================] - 46s 179ms/step - loss: 2.4751 - acc: 0.4348 - val_loss: 2.4604 - val_acc: 0.4113 - lr: 0.1000
Epoch 11/30
255/255 [==============================] - 46s 177ms/step - loss: 2.3454 - acc: 0.4618 - val_loss: 2.0499 - val_acc: 0.4992 - lr: 0.1000
Epoch 12/30
255/255 [==============================] - 50s 195ms/step - loss: 2.3464 - acc: 0.4781 - val_loss: 2.6694 - val_acc: 0.4526 - lr: 0.1000
Epoch 13/30
255/255 [==============================] - 46s 177ms/step - loss: 2.4363 - acc: 0.4738 - val_loss: 2.7333 - val_acc: 0.4573 - lr: 0.1000
Epoch 14/30
255/255 [==============================] - 47s 180ms/step - loss: 2.4139 - acc: 0.4835 - val_loss: 2.3916 - val_acc: 0.4853 - lr: 0.1000
Epoch 14: early stopping
```

- Model stopped at 14 epoch since we have used early stopping with a patience of 3.
- This gives us a accuracy of nearly 0.4835 for both train and test datas.
- Note: Only Classification of is done by this model.
- The performance is as follows: **Model seem to perform better than our previous Basic CNN model.**

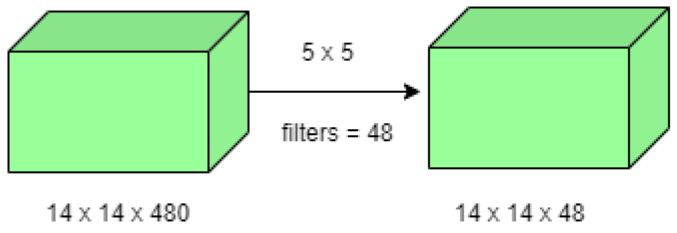




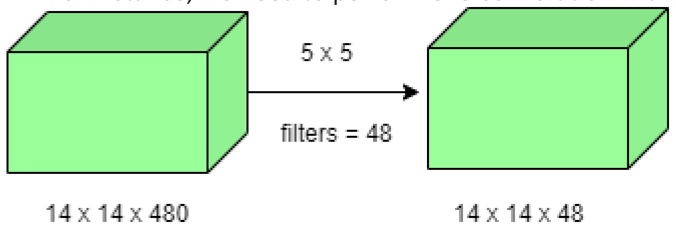
Inception Network V1

In the convolutional neural networks prior to the InceptionNet primarily focused on increasing the depth of the network to extract feature of features, for improving the learning capabilities of the model. The developers of the InceptionNet were the first to focus on increasing the width and the depth of the model simultaneously to attain better accuracy while keeping the computing resources constant.

- Before digging into Inception Net model, it's essential to know an important concept that is used in Inception network: 1 X 1 convolution: A 1×1 convolution simply maps an input pixel with all its respective channels to an output pixel. 1×1 convolution is used as a dimensionality reduction module to reduce computation to an extent.

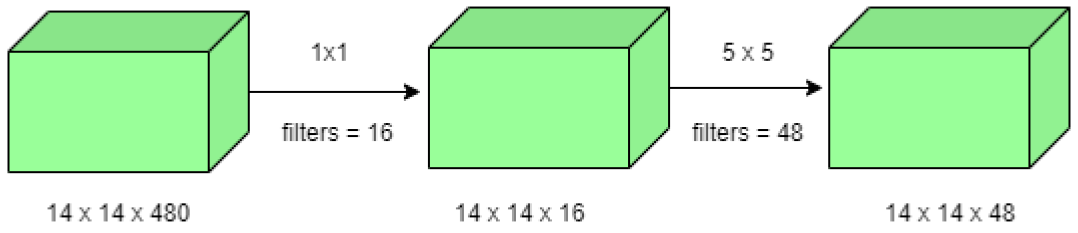


For instance, we need to perform 5×5 convolution without using 1×1 convolution as below:



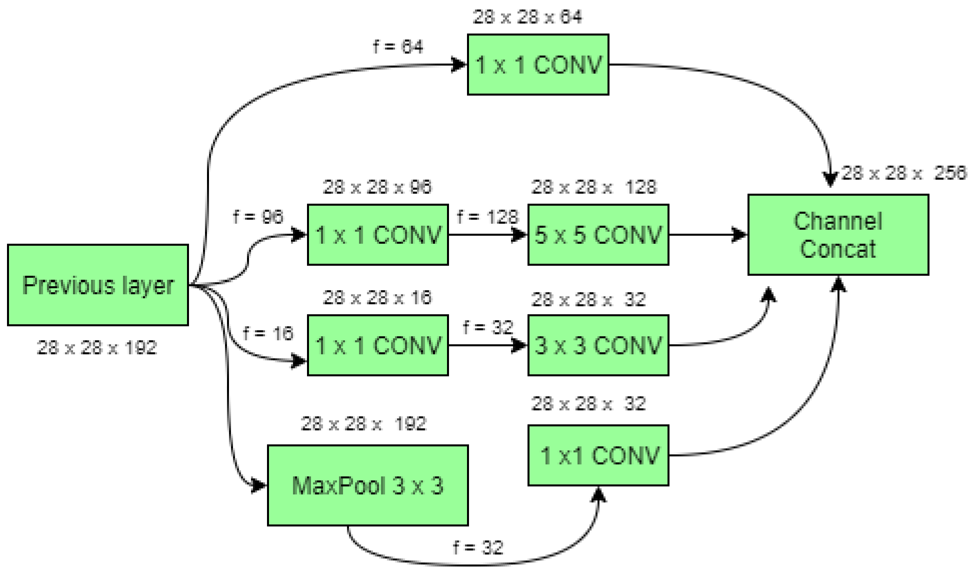
- Number of operations involved here is (14×14×48) × (5×5×480) = 112.9M

Using 1×1 convolution



- Number of operations for 1×1 convolution = (14×14×16) × (1×1×480) = 1.5M Number of operations for 5×5 convolution = (14×14×48) × (5×5×16) = 3.8M After addition we get, 1.5M + 3.8M = 5.3M

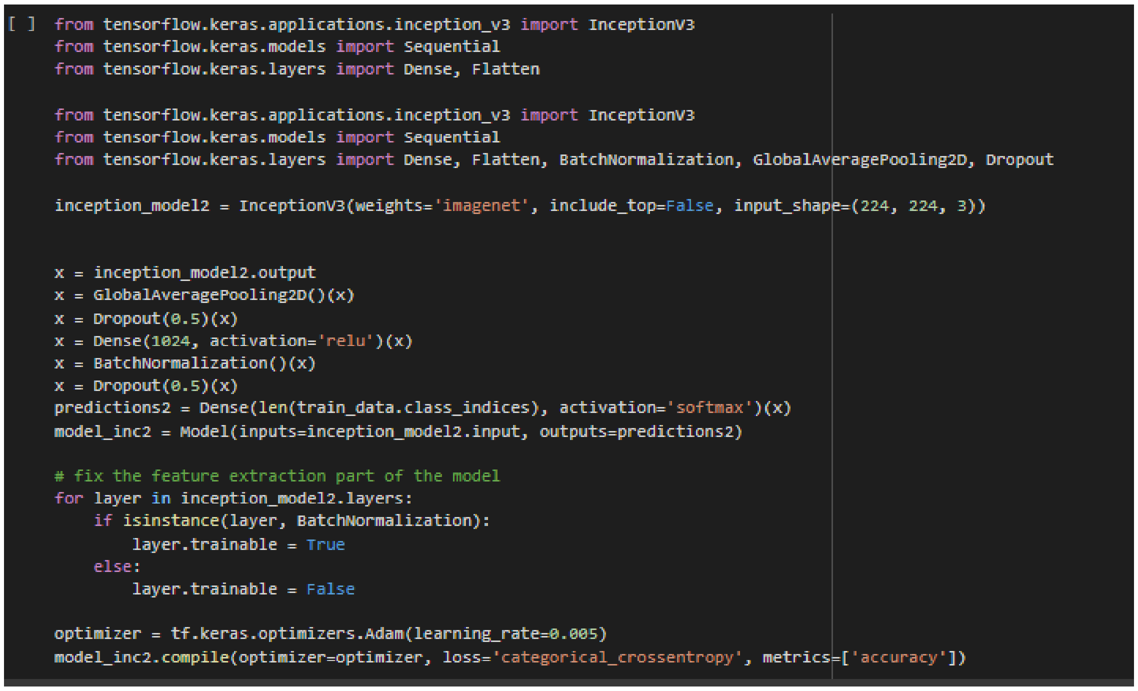
- **Which is immensely smaller than 112.9M** ! Thus, 1×1 convolution can help to reduce model size which can also somehow help to reduce the overfitting problem. Inception model with dimension reductions: Deep Convolutional Networks are computationally expensive. However, computational costs can be reduced drastically by introducing a 1 x 1 convolution. Here, the number of input channels is limited by adding an extra 1×1 convolution before the 3×3 and 5×5 convolutions. Though adding an extra operation may seem counter-intuitive but 1×1 convolutions are far cheaper than 5×5 convolutions. Do note that the 1×1 convolution is introduced after the max-pooling layer, rather than before. At last, all the channels in the network are concatenated together i.e. (28 x 28 x (64 + 128 + 32 + 32)) = 28 x 28 x 256



Inception Network: This architecture has 22 layers in total! Using the dimension-reduced inception module, a neural network architecture is constructed. This is popularly known as GoogLeNet (**Inception v1**). GoogLeNet has 9 such inception modules fitted linearly. It is 22 layers deep (27, including the pooling layers). At the end of the architecture, fully connected layers were replaced by a global average pooling which calculates the average of every feature map. This indeed dramatically declines the total number of parameters. Thus, Inception Net is a victory over the previous versions of CNN models. It achieves an accuracy of top-5 on ImageNet, it reduces the computational cost to a great extent without compromising the speed and accuracy.

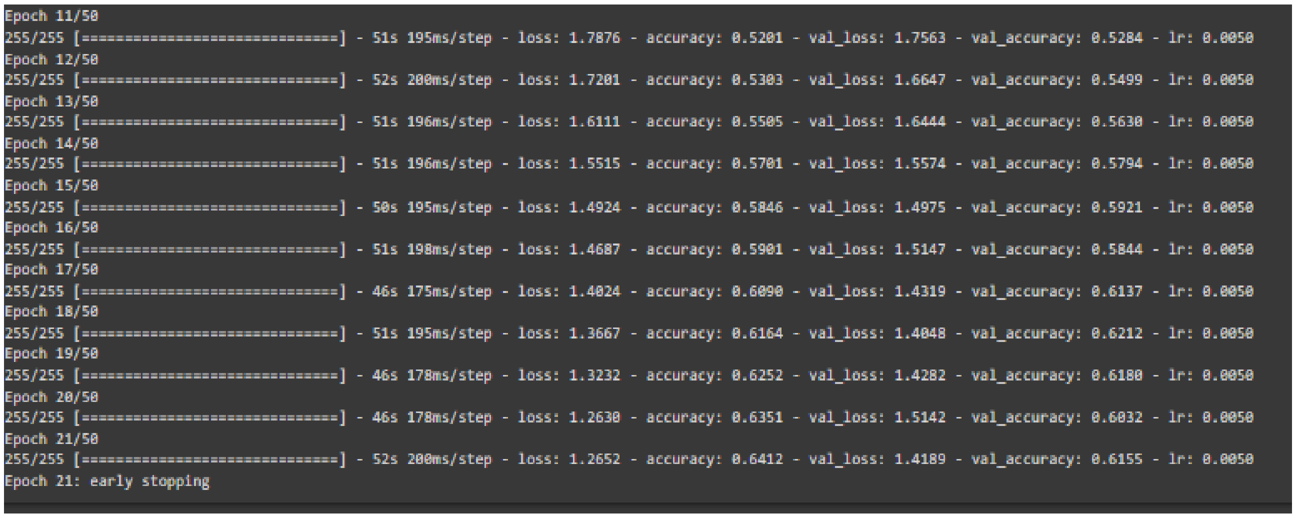
Code and explanation:

- Inception model uses very similar coding to the previously used Effnet model. We have not used the SGD optimizer instead chosen Adam without any scheduler with learning rate of 0.005. This model seem to be performing much better than Effnet model with accuracy of 63%



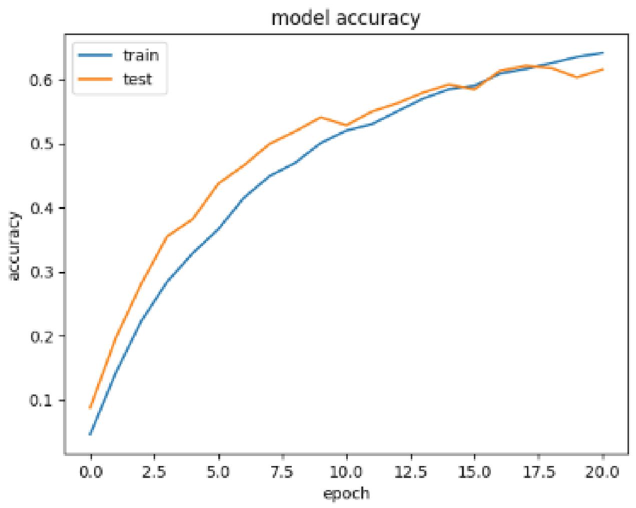
```
[ ] from tensorflow.keras.applications.inception_v3 import InceptionV3
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, Flatten

    from tensorflow.keras.applications.inception_v3 import InceptionV3
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, Flatten, BatchNormalization, GlobalAveragePooling2D, Dropout

    inception_model2 = InceptionV3(weights='imagenet', include_top=False, input_shape=(224, 224, 3))


    x = inception_model2.output
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.5)(x)
    x = Dense(1024, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)
    predictions2 = Dense(len(train_data.class_indices), activation='softmax')(x)
    model_inc2 = Model(inputs=inception_model2.input, outputs=predictions2)

    # fix the feature extraction part of the model
    for layer in inception_model2.layers:
        if isinstance(layer, BatchNormalization):
            layer.trainable = True
        else:
            layer.trainable = False

    optimizer = tf.keras.optimizers.Adam(learning_rate=0.005)
    model_inc2.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

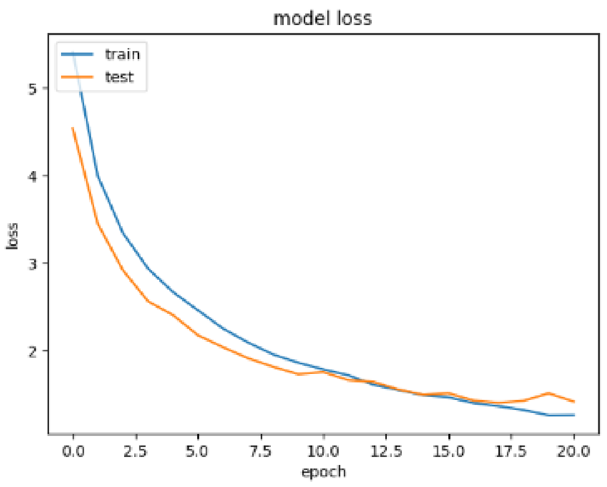
- Due to early stopping the model seem to stop at 20th epoch , as sufficient improvemnet was not there. The model doesn't seem to overfit and Gives almost equal accuracy in both training and testing of 63 %.
- This is the highest accuracy we have received in all the models we have tried.
- Note: this Model predicts only the classification.
- The Learning rates, optimizers are different in each model in order to experiment and find out which algorithm is outperforming .



```
Epoch 11/50
255/255 [==============================] - 51s 195ms/step - loss: 1.7876 - accuracy: 0.5201 - val_loss: 1.7563 - val_accuracy: 0.5284 - lr: 0.0050
Epoch 12/50
255/255 [==============================] - 52s 200ms/step - loss: 1.7201 - accuracy: 0.5303 - val_loss: 1.6647 - val_accuracy: 0.5499 - lr: 0.0050
Epoch 13/50
255/255 [==============================] - 51s 196ms/step - loss: 1.6111 - accuracy: 0.5505 - val_loss: 1.6444 - val_accuracy: 0.5630 - lr: 0.0050
Epoch 14/50
255/255 [==============================] - 51s 196ms/step - loss: 1.5515 - accuracy: 0.5701 - val_loss: 1.5574 - val_accuracy: 0.5794 - lr: 0.0050
Epoch 15/50
255/255 [==============================] - 50s 195ms/step - loss: 1.4924 - accuracy: 0.5846 - val_loss: 1.4975 - val_accuracy: 0.5921 - lr: 0.0050
Epoch 16/50
255/255 [==============================] - 51s 198ms/step - loss: 1.4687 - accuracy: 0.5901 - val_loss: 1.5147 - val_accuracy: 0.5844 - lr: 0.0050
Epoch 17/50
255/255 [==============================] - 46s 175ms/step - loss: 1.4024 - accuracy: 0.6090 - val_loss: 1.4319 - val_accuracy: 0.6137 - lr: 0.0050
Epoch 18/50
255/255 [==============================] - 51s 195ms/step - loss: 1.3667 - accuracy: 0.6164 - val_loss: 1.4048 - val_accuracy: 0.6212 - lr: 0.0050
Epoch 19/50
255/255 [==============================] - 46s 178ms/step - loss: 1.3232 - accuracy: 0.6252 - val_loss: 1.4282 - val_accuracy: 0.6180 - lr: 0.0050
Epoch 20/50
255/255 [==============================] - 46s 178ms/step - loss: 1.2630 - accuracy: 0.6351 - val_loss: 1.5142 - val_accuracy: 0.6032 - lr: 0.0050
Epoch 21/50
255/255 [==============================] - 52s 200ms/step - loss: 1.2652 - accuracy: 0.6412 - val_loss: 1.4189 - val_accuracy: 0.6155 - lr: 0.0050
Epoch 21: early stopping
```

The performance is as follows for the Inception model:



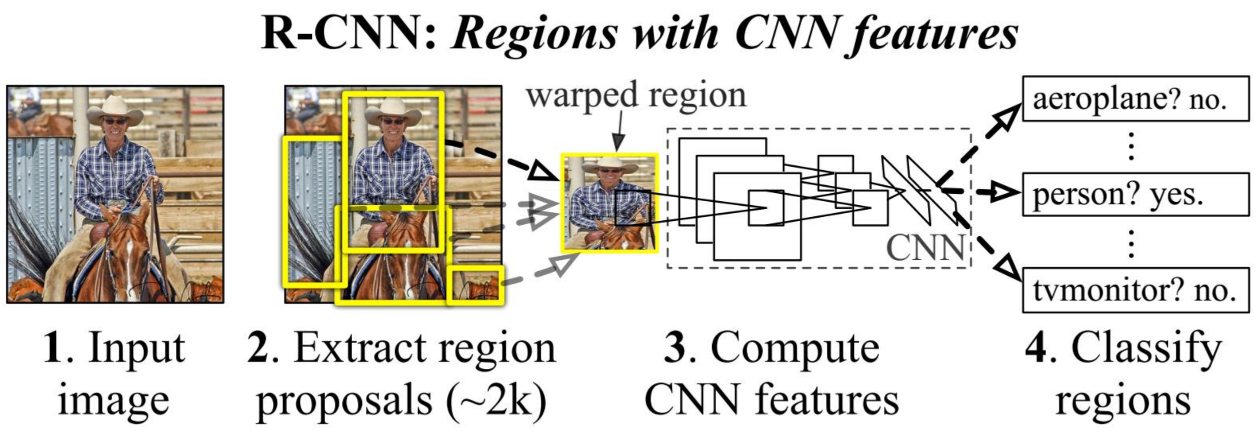


Observation:

Inception seem to be performing good compared to previous models for classification ., but for BBOX and regression of BBOX planned to try out Faster RCNN and Mask RCNN Model.

R-CNN

R-CNN(paper) one of the first large and successful application of convolutional neural networks to the problem of object localization, detection, and segmentation. The approach was demonstrated on benchmark datasets, achieving then state-of-the-art results on the VOC-2012 dataset and the 200-class ILSVRC-2013 object detection dataset.



Region based CNN consists of three modules — Region Proposal, Feature Extractor and Classifier.

Region Proposal : When an input image is given region proposal tries to detect different regions (~2000) in different sizes and aspect ratios. In other words, it draws multiple bounding boxes in input image as shown below.

Region Proposal Result

Feature Extractor: Each proposed region will be trained by a CNN network and the last layer (4096 features) will be extracted as features so the final output from Feature extractor will be **Number of proposed regions x 4096**

Classifier: Once the features are extracted we need to classify the objects inside each regions. To do this a linear SVM model is trained for classification, Specifically one SVM model for each class.

Cons of R-CNN:

- It takes a huge amount of time to train the network as you would have to classify 2000 region proposals per image.
- It cannot be implemented real time as it takes around 47 seconds for each test image.
- The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

Scope for Improvement in the future

The original dataset has roughly 50–50% train-test split. For further analysis, these images can be combined, and a train-test split of 80–20% can be used.

The dataset after consolidating to five classes has a class imbalance with Vans representing about 6% of the data and on the other extreme Convertible/Coupes and Sedans each representing about 32% of the data. To address this class imbalance the above classification methods can be re-iterated with oversampling techniques: random oversampling, and Synthetic Minority Oversampling TEchnique (SMOTE).

Image segmentation is one of the most popular image processing tasks. Some common pitfalls related to the most frequently used metrics in image segmentation, namely the Dice Similarity Coefficient (DSC), the Harsdorf Distance (HD),and the Intersection over Union (IoU); the problems related to segmentationmetrics are assigned to four categories, namely

(1) awareness of fundamental mathematical properties of metrics, necessary to determine the applicability of a metric,

a) Small structures - Segmentation of small structures, such as brain lesions, cells imaged at low magnification or distant cars, is essential for many image processing applications

b) Noise/errors in the reference annotations - Similar problems may arise in the presence of annotation artifacts. Even a single erroneous pixel in the reference annotation may lead to a substantial decrease in the measured performance, especially in the case of the HD.

c) Shape unawareness - Metrics measuring the overlap between objects are not designed to uncover differences in shapes. This is an important problem for many applications
d) Oversegmentation vs. undersegmentation - In some applications such as autonomous driving or radiotherapy, it may be highly relevant whether an algorithm tends to over- or under-segment the target structure.

(2) suitability for the underlying image processing task,
a) A common problem is that segmentation metrics, such as the DSC, are applied to detection and localization tasks. In general, the DSC is strongly biased against single objects, therefore not appropriate for a detection task of multiple structures

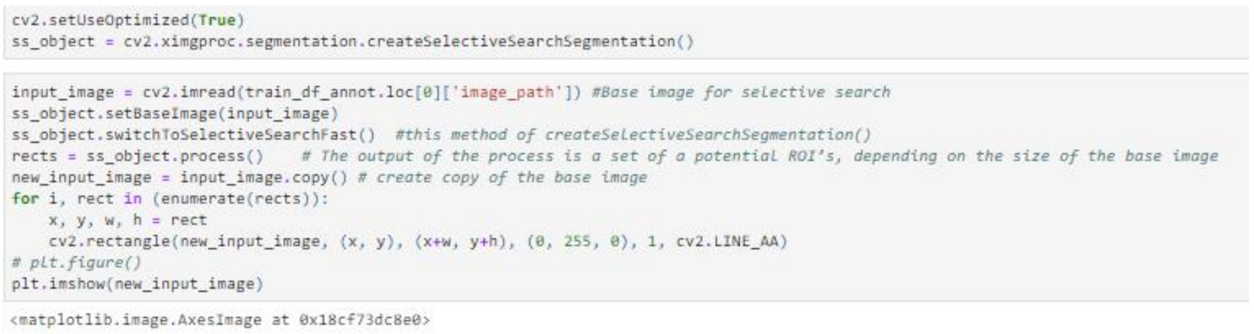
(3) metric aggregation to combine metric values of single images into one accumulated score
a) In the case of metrics with fixed boundaries, like the DSC or the IoU, missing values can easily be set to the worst possible value. For distance-based measures without lower/upper bounds, the strategy of how to deal with missing values is not trivial. In the case of the HD, one may choose the maximum distance of the image and add 1 or normalize the metric values and use the worst possible value. Crucially, however, every choice will produce a different aggregated value, thus potentially affecting the ranking.

(4) metric combination to reflect different aspects in algorithm validation.
a) A single metric typically does not reflect all important aspects that are essential for algorithm validation. Hence, multiple metrics with different properties are often combined. However, the selection of metrics should be well considered as some metrics are mathematically related to each other. A prominent example is the IoU – the most popular segmentation metric in computer vision – which highly correlates with the DSC – the most popular segmentation metric in image analysis. In fact, the IoU and the DSC are mathematically related. Mutually dependent metrics (DSC and IoU) will lead to the same ranking, whereas metrics measuring different properties (HD) will lead to a different ranking.

Plan:

Faster rcnn and masked Rcnn are planned to be executed for our dataset. Given that the Rcnn model Needs a base model --> Resnet 50 is planned to be used as base model.

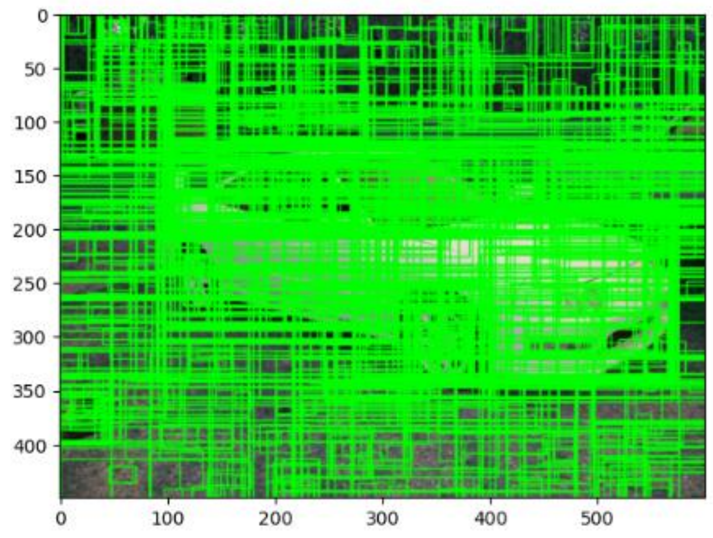
Code and explanation



```
cv2.setUseOptimized(True)
ss_object = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
```

```
input_image = cv2.imread(train_df_annot.loc[0]['image_path']) #Base image for selective search
ss_object.setBaseImage(input_image)
ss_object.switchToSelectiveSearchFast()  #this method of createSelectiveSearchSegmentation()
rects = ss_object.process()    # The output of the process is a set of a potential ROI's, depending on the size of the base image
new_input_image = input_image.copy() # create copy of the base image
for i, rect in (enumerate(rects)):
    x, y, w, h = rect
    cv2.rectangle(new_input_image, (x, y), (x+w, y+h), (0, 255, 0), 1, cv2.LINE_AA)
# plt.figure()
plt.imshow(new_input_image)
```

```
<matplotlib.image.AxesImage at 0x18cf73dc8e0>
```



This code performs selective search on an input image to generate a set of potential regions of interest (ROIs). The ROIs are then visualized by drawing rectangles around them on a copy of the input image.

- cv2.setUseOptimized(True): This enables optimized OpenCV functions, which can result in faster processing times.
- ss_object = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation(): This creates an instance of the SelectiveSearchSegmentation class for performing selective search.
- input_image = cv2.imread(train_df_annot.loc[0]['image_path']):This loads an input image from a specified file path.

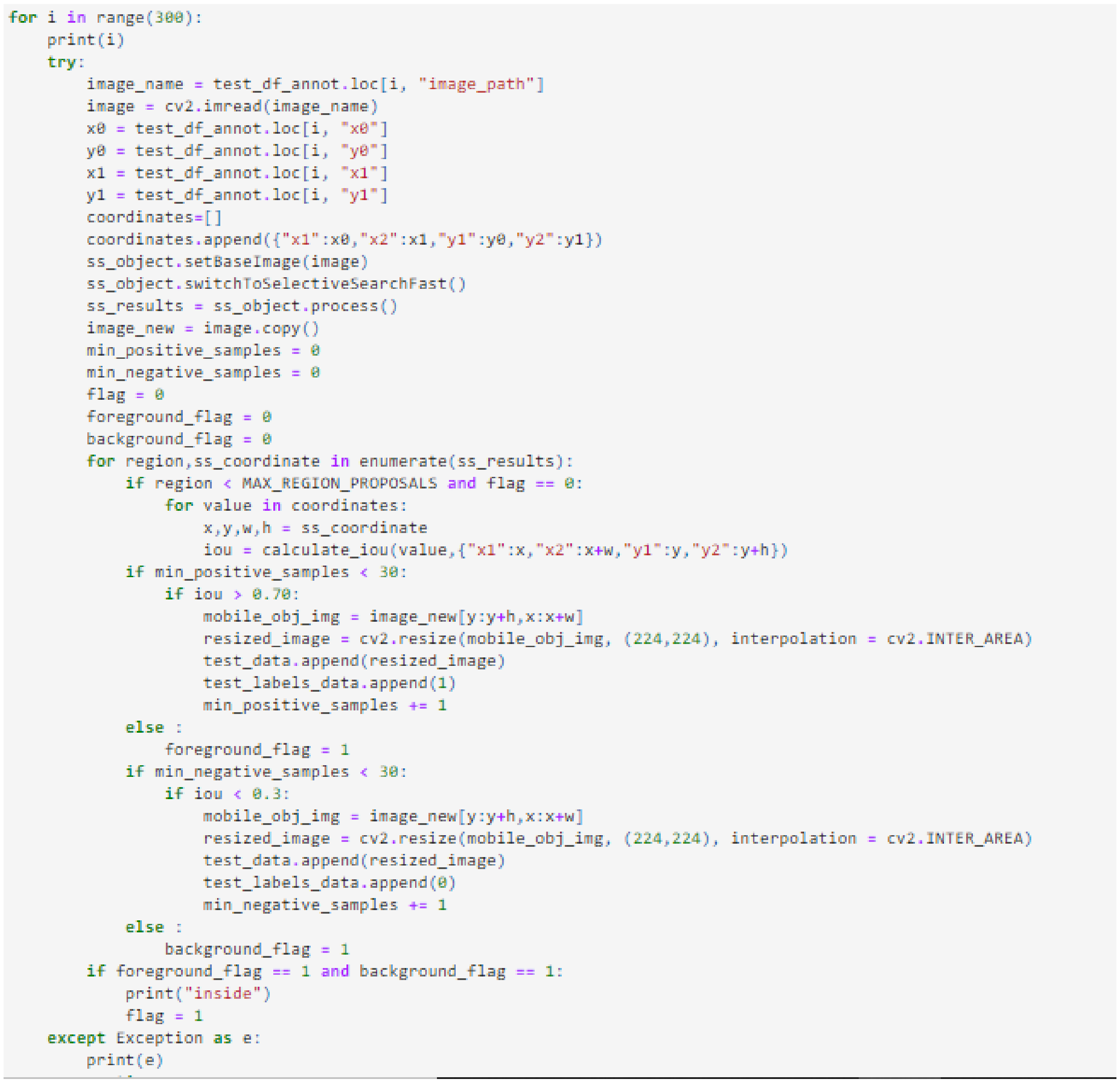
- ss_object.setBaseImage(input_image): This sets the base image for selective search to the input image.
- ss_object.switchToSelectiveSearchFast():This sets the selective search method to fast search.
- rects = ss_object.process(): This performs selective search on the base image to generate a set of ROIs, represented as rectangles.
- new_input_image = input_image.copy(): This creates a copy of the input image to draw rectangles on.
- for i, rect in (enumerate(rects)): ...: This loops over each ROI rectangle in rects and draws a green rectangle on new_input_image using the OpenCV function cv2.rectangle.
- plt.imshow(new_input_image): This displays the image with the drawn ROIs using Matplotlib's imshow function.

Function for Intersection over union: "calculate_iou" for Selective_search:

```
def calculate_iou(bb_1, bb_2):
    '''
  Now we are initialising the function to calculate IOU (Intersection Over Union)
  of the ground truth box from the box computed by selective search.
  To divide the generated ROI's, for example, we can use a metric called IoU.
  It's defined as the intersection area divided by area of the union of a predicted
  bounding box and ground-truth box.
  '''

    assert bb_1['x1'] < bb_1['x2']    # The assert keyword lets you test if a condition in your code returns True,
    assert bb_1['y1'] < bb_1['y2']    # if not, the program will raise an AssertionError.
    assert bb_2['x1'] < bb_2['x2']
    assert bb_2['y1'] < bb_2['y2']

    x_left = max(bb_1['x1'], bb_2['x1'])
    y_top = max(bb_1['y1'], bb_2['y1'])
    x_right = min(bb_1['x2'], bb_2['x2'])
    y_bottom = min(bb_1['y2'], bb_2['y2'])

    if x_right < x_left or y_bottom < y_top:
        return 0.0

    intersection = (x_right - x_left) * (y_bottom - y_top)

    bb_1_area = (bb_1['x2'] - bb_1['x1']) * (bb_1['y2'] - bb_1['y1'])
    bb_2_area = (bb_2['x2'] - bb_2['x1']) * (bb_2['y2'] - bb_2['y1'])

    iou_value = intersection / float(bb_1_area + bb_2_area - intersection)
    assert iou_value >= 0.0
    assert iou_value <= 1.0
    return iou_value
```

The calculate_iou function calculates the intersection over union (IOU) of two bounding boxes. It takes in two bounding boxes as input and returns a float value between 0 and 1, representing the IOU between the two bounding boxes. The IOU is a common evaluation metric used in object detection tasks. It measures the overlap between two bounding boxes and is defined as the ratio of the area of intersection to the area of the union of the two boxes. If the two boxes do not intersect, the function returns 0. If they overlap perfectly, the function returns 1.

Code for Selective search



```
for i in range(300):
    print(i)
    try:
        image_name = test_df_annot.loc[i, "image_path"]
        image = cv2.imread(image_name)
        x0 = test_df_annot.loc[i, "x0"]
        y0 = test_df_annot.loc[i, "y0"]
        x1 = test_df_annot.loc[i, "x1"]
        y1 = test_df_annot.loc[i, "y1"]
        coordinates=[]
        coordinates.append({"x1":x0,"x2":x1,"y1":y0,"y2":y1})
        ss_object.setBaseImage(image)
        ss_object.switchToSelectiveSearchFast()
        ss_results = ss_object.process()
        image_new = image.copy()
        min_positive_samples = 0
        min_negative_samples = 0
        flag = 0
        foreground_flag = 0
        background_flag = 0
        for region,ss_coordinate in enumerate(ss_results):
            if region < MAX_REGION_PROPOSALS and flag == 0:
                for value in coordinates:
                    x,y,w,h = ss_coordinate
                    iou = calculate_iou(value,{"x1":x,"x2":x+w,"y1":y,"y2":y+h})
            if min_positive_samples < 30:
                if iou > 0.70:
                    mobile_obj_img = image_new[y:y+h,x:x+w]
                    resized_image = cv2.resize(mobile_obj_img, (224,224), interpolation = cv2.INTER_AREA)
                    test_data.append(resized_image)
                    test_labels_data.append(1)
                    min_positive_samples += 1
            else :
                foreground_flag = 1
            if min_negative_samples < 30:
                if iou < 0.3:
                    mobile_obj_img = image_new[y:y+h,x:x+w]
                    resized_image = cv2.resize(mobile_obj_img, (224,224), interpolation = cv2.INTER_AREA)
                    test_data.append(resized_image)
                    test_labels_data.append(0)
                    min_negative_samples += 1
            else :
                background_flag = 1
        if foreground_flag == 1 and background_flag == 1:
            print("inside")
            flag = 1
    except Exception as e:
        print(e)
```

The code is performing selective search on a set of images and generating proposals for objects within those images. It is then using the annotations provided in the test_df_annot dataframe to extract positive and negative samples for a machine learning task, specifically object detection.

Explanation of the code:

- A loop is set up to iterate over a range of 300 images.
- The code reads in an image from the file path specified in the test_df_annot dataframe.
- The x0, y0, x1, and y1 values are extracted from the dataframe, which correspond to the bounding box coordinates for the object of interest within the image.
- A list called coordinates is created that contains a dictionary with the bounding box coordinates.
- The image is set as the base image for selective search using setBaseImage(image), and selective search is switched to the fast mode using switchToSelectiveSearchFast().
- The selective search algorithm is then applied to the image, and the proposed regions are stored in ss_results.
- A copy of the original image is created, and the code initializes variables to track the number of positive and negative samples that have been extracted so far.
- The code then loops through each proposed region, calculating the IoU (Intersection over Union) between the region and the bounding box coordinates.
- If the IoU is greater than 0.7 and the number of positive samples is less than 30, then the code extracts the region and resizes it to 224x224 pixels using cv2.resize(). The resized image is then appended to the test_data list, and the label "1" is appended to the test_labels_data list.
- If the IoU is less than 0.3 and the number of negative samples is less than 30, then the code extracts the region and resizes it to 224x224 pixels using cv2.resize(). The resized image is then appended to the test_data list, and the label "0" is appended to the test_labels_data list.
- If the number of positive or negative samples reaches 30, then the code sets flags to indicate that the minimum number of samples has been extracted for each class.
- If the number of positive and negative samples has been extracted for each class, then the loop is terminated.
- If an exception is raised during the loop, the code continues to the next iteration.
- **After the loop completes, the train_data and train_labels_data lists contain the extracted image samples and their corresponding labels for the object detection task. The same applied to test data as well.**

Creating a base model for Faster RCNN and Masked RCNN:

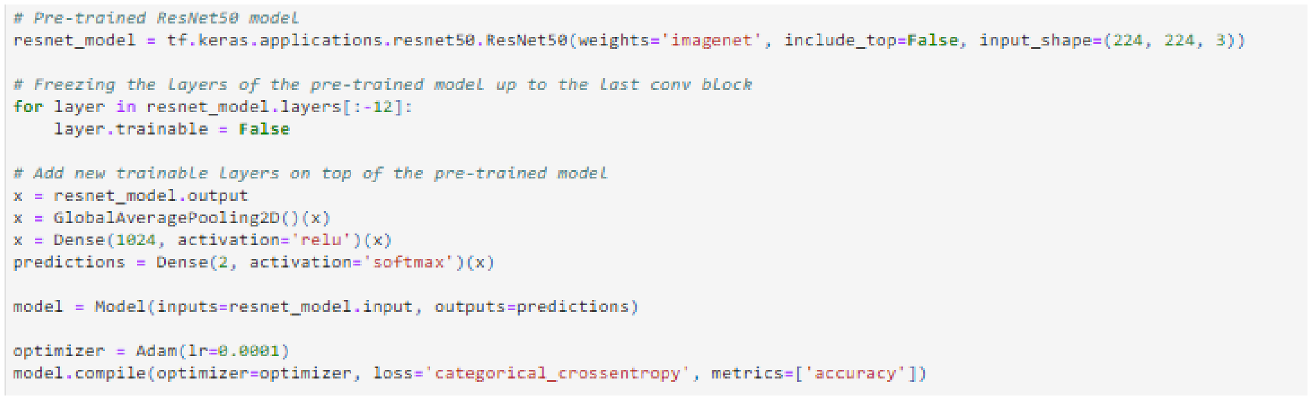
Basic preprocessing steps before sending the data to Base model (resnet):

My_Label_Binarizer class by inheriting from the LabelBinarizer class from sklearn.preprocessing. It overrides the transform and inverse_transform methods to handle binary and multi-class classification problems. The transform method binarizes the labels using the super() method and then stacks the binary and complement labels horizontally. The inverse_transform method checks if

the problem is binary and returns the inverse transform of the binary label by passing only the first column of the input, else it returns the inverse transform of the multi-class labels using the super() method.

Then an instance of My_Label_Binarizer i created and applies it to the training and testing labels y_train and y_test. Afterward, it defines train_generator and test_generator using the ImageDataGenerator class from tensorflow.keras.preprocessing.image to augment the training and testing data by flipping and rotating the images.Finally, it creates the train_data and test_data generators using the flow method of the train_generator and test_generator objects respectively, by passing the training and testing features and labels and the batch size.

Resnet Base Model:



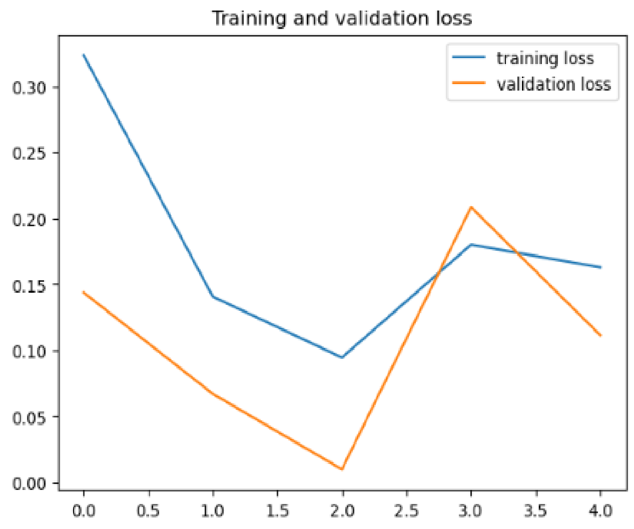
```
# Pre-trained ResNet50 model
resnet_model = tf.keras.applications.resnet50.ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freezing the layers of the pre-trained model up to the last conv block
for layer in resnet_model.layers[:-12]:
    layer.trainable = False

# Add new trainable layers on top of the pre-trained model
x = resnet_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(2, activation='softmax')(x)

model = Model(inputs=resnet_model.input, outputs=predictions)

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

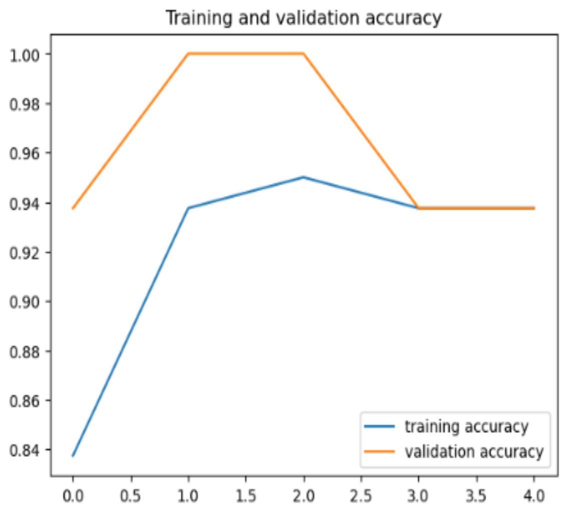
A pre-trained ResNet50 model is download and adds new trainable layers on top of it. The ResNet50 model is pre-trained on the ImageNet dataset and the last fully connected layer of the model is removed by setting include_top=False. The layers of the pre-trained model up to the last convolutional block are frozen so that they are not updated during training. The new trainable layers consist of a global average pooling layer, a fully connected layer with 1024 units and ReLU activation function, and a fully connected layer with 2 units and softmax activation function, which is the output layer. The Model function is used to create a new model that takes the same input as the pre-trained ResNet50 model and outputs the predictions of the new layers. The model is compiled with the Adam optimizer, a learning rate of 0.0001, categorical cross-entropy loss, and accuracy as the evaluation metric. Finally, the summary method is used to print a summary of the model architecture, including the input and output shapes of each layer and the number of trainable parameters.

Training of Resnet layers with our dataset

- Due to memory issues and frequent crashes., we have planned to take only 300 sample dataset and train those samples on our model.
- Thankfully , we have got good accuracy.

```
Epoch 5: val_accuracy did not improve from 1.00000
10/10 [==============================] - 9s 882ms/step - loss: 0.1629 - accura
cy: 0.9375 - val_loss: 0.1115 - val_accuracy: 0.9375
```



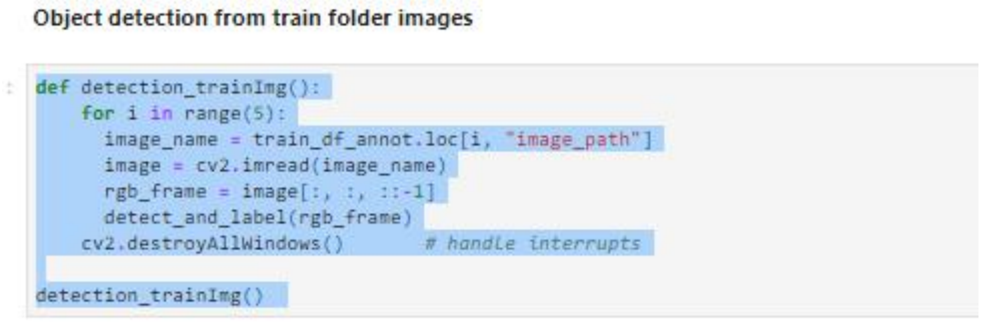


Faster RCNN Model to build on Resnet50

- Base RESNET model predicts the classification if it's a car or not. But bounding box will be predicted by Faster RCNN or Masked RCNN.
- Model zoo from gluoncv lets us download a Faster- RCNN model.

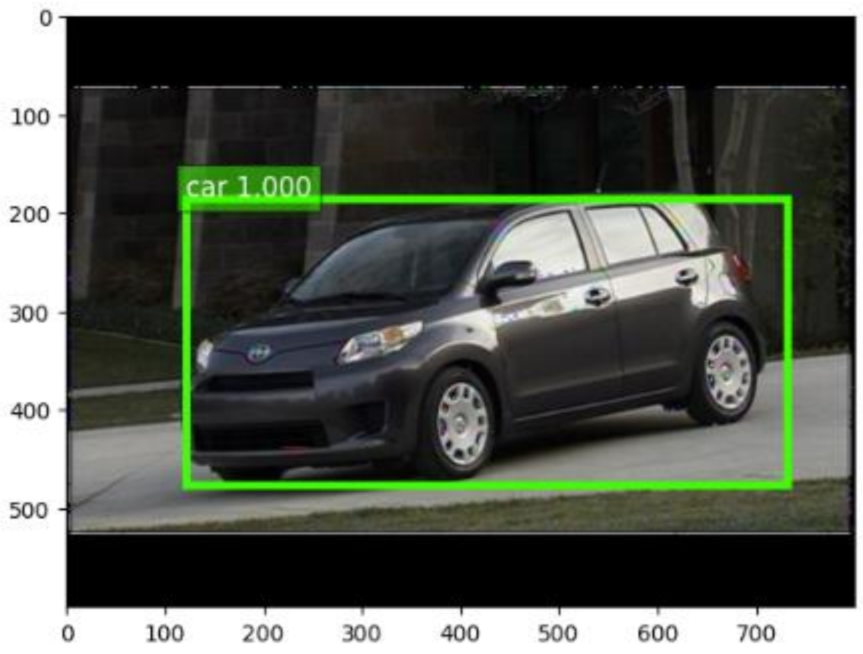
```
#pretrained Faster RCNN with Resnet50 as base model v3 model from model_zoo class of gluoncv
fcnn_net = model_zoo.get_model('faster_rcnn_resnet50_v1b_voc', pretrained=True)
```

- FasterRCNN : detect_and_label function is using a pre-trained Faster R-CNN model with ResNet50 as the base model (faster_rcnn_resnet50_v1b_voc from the model_zoo module of GluonCV). The function takes an RGB frame as input, converts it to an MXNet array and applies the necessary transformations for the model to perform object detection. The function then passes the transformed array to the model and obtains the class IDs, scores, and bounding boxes of the detected objects. Finally, the function uses the plot_bbox function from the utils.viz module to draw bounding boxes on the input image and display it. The displayed image will show the detected objects with their corresponding labels and confidence scores.



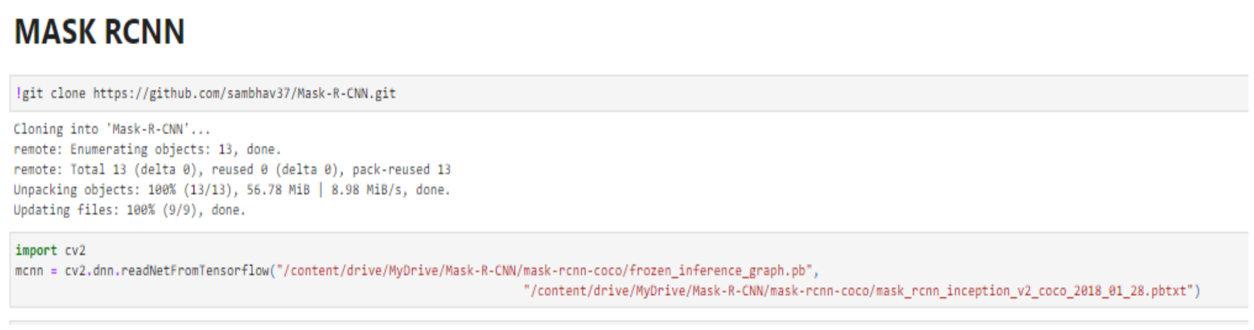
Object detection from train folder images

```
def detection_trainImg():
    for i in range(5):
      image_name = train_df_annot.loc[i, "image_path"]
      image = cv2.imread(image_name)
      rgb_frame = image[:, :, ::-1]
      detect_and_label(rgb_frame)
    cv2.destroyAllWindows()       # handle interrupts

detection_trainImg()
```



Mask- RCNN:

we are using Mask-R-CNN repository from GitHub and then used OpenCV's dnn module to load the pre-trained Mask R-CNN model on top of the Inception V2 architecture trained on the COCO dataset. The readNetFromTensorflow function from OpenCV's dnn module is used to load the frozen inference graph (frozen_inference_graph.pb) and the corresponding protobuf file (mask_rcnn_inception_v2_coco_2018_01_28.pbtxt) which contains the configuration information for the network. This creates an instance of the mcnn network object that can be used for inference on new images.



MASK RCNN

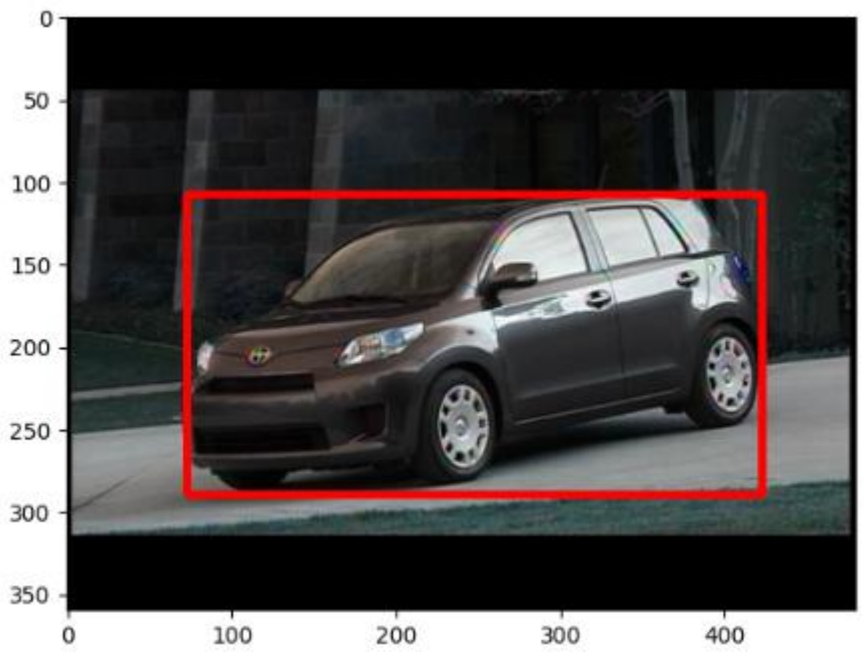
```
!git clone https://github.com/sambhav37/Mask-R-CNN.git
```

```
Cloning into 'Mask-R-CNN'...
remote: Enumerating objects: 13, done.
remote: Total 13 (delta 0), reused 0 (delta 0), pack-reused 13
Unpacking objects: 100% (13/13), 56.78 MiB | 8.98 MiB/s, done.
Updating files: 100% (9/9), done.
```

```
import cv2
mcnn = cv2.dnn.readNetFromTensorflow("/content/drive/MyDrive/Mask-R-CNN/mask-rcnn-coco/frozen_inference_graph.pb",
                                     "/content/drive/MyDrive/Mask-R-CNN/mask-rcnn-coco/mask_rcnn_inception_v2_coco_2018_01_28.pbtxt")
```

- We can Visualize the detections and segmentation masks on some sample images from the training dataset using Mask R-CNN. The code reads the frozen inference graph and pb txt file of the Mask R-CNN model and uses it to detect and segment objects in the images.
- The cv2.dnn.blobFromImage() function is used to preprocess the input image and create a blob from it, which is then passed to the Mask R-CNN model for detection and segmentation. The forward() method is called on the model to get the output boxes and masks.
- Then, for each detected object, the code extracts the bounding box coordinates, resizes the mask to the size of the ROI, and fills the mask in the ROI using the cv2.fillPoly() function. Finally, the code displays the detected objects and segmentation masks on the input image using the plt.imshow() and plt.show() functions.

BBOX predictions from Masked RCNN:



Pickling

The whole idea is to use the Pickle module to implement binary protocols for serialising and DE-serialising a Python object structure so that the Python object is converted into a byte stream. In summary Pickle helps you to be able to do that without having to rewrite everything or train the model all over again.

Conclusion:

- Inception Model seem to be performing great for Classification.
- Masked RCNN and Faster RCNN has predicted BBOX in a great way.
- while Faster R-CNN focuses on detecting objects and their bounding boxes, Mask R-CNN extends this to also include instance segmentation, which involves identifying the exact pixels that belong to each object instance.
- While Predicting the classification and BBOX with the proper code and dataloader has been difficult but we have multiple libraries to support and the learning was abundant in this Project.

References:

Transfer Learning Definition, Methods, and Applications | Spiceworks - Spiceworks

Region — Based Convolutional Neural Network (RCNN) | by Ramji Balasubramanian | Analytics Vidhya | Medium

Complete Architectural Details of all EfficientNet Models | by Vardan Agarwal | Towards Data Science

https://www.geeksforgeeks.org/ml-inception-network-v1/