Object Classification Using Convolutional Neural Network (CNN)

By

Gebeyaw Tigabu

Introduction to Image Classification

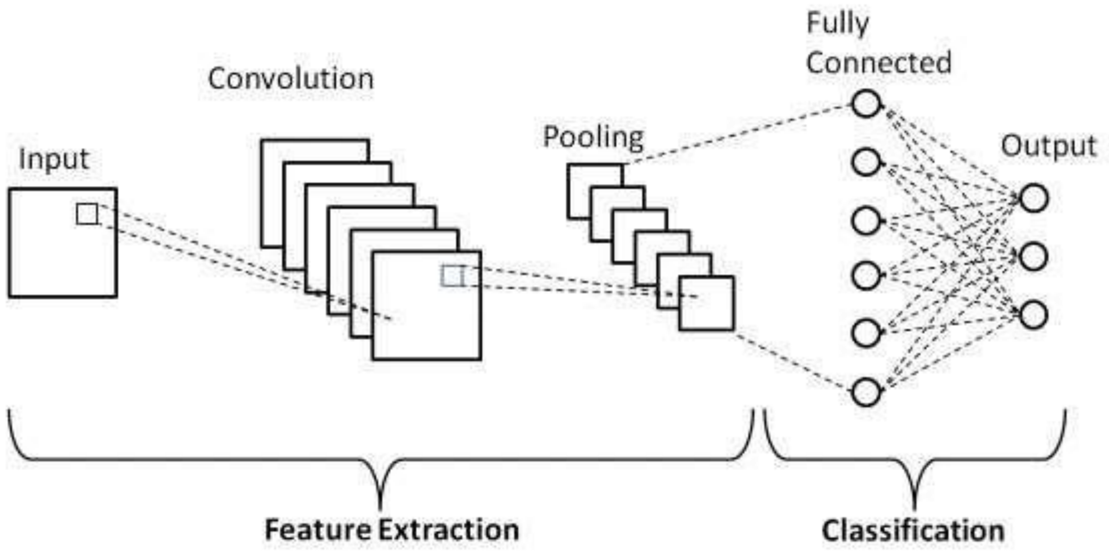
Image classification is the process of categorizing and labeling groups of pixels or vectors within an image based on specific rules. It refers to a process in computer vision classifying an image according to its visual content.

Today, with the increasing volatility, necessity and applications of artificial intelligence, fields like machine learning, and its subsets, deep learning and neural networks have gained immense momentum. Image classification is perhaps the most important part of digital image analysis. Classification between objects is a complex task and therefore image classification has been an important task within the field of computer vision. Image classification refers to the labelling of images into one of a number of predefined classes.

Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN, or ConvNet) is a special kind of multi-layer neural network, designed to recognize visual patterns directly from pixel images with minimal pre-processing. It is a special architecture of artificial neural networks. Convolutional neural networks are comprised of two very simple elements, namely convolutional layers and pooling layers. Although simple, there are near-infinite ways to arrange these layers for a given computer vision problem. The elements of a convolutional neural network, such as convolutional and pooling layers, are relatively straightforward to understand.

The challenging part of using convolutional neural networks in practice is how to design model architectures that best use these simple elements. The reason why convolutional neural network is hugely popular is because of their architecture, the best thing is there is no need of feature extraction. The system learns to do feature extraction and the core concept is, it uses convolution of image and filters to generate invariant features which are passed on to the next layer. The features in next layer are convoluted with different filters to generate more invariant and abstract features and the process continues till it gets final feature/output which is invariant to occlusions.



Basic Architecture

There are two main parts to a CNN architecture

A *convolution tool* that separates and identifies the various features of the image for analysis in a process called as Feature Extraction

A *fully connected layer* that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.

Convolution Layers

There are three types of layers that make up the CNN which are the convolutional layers, pooling layers, and fully-connected (FC) layers. When these layers are stacked, a CNN architecture will be formed. In addition to these three layers, there are two more important parameters which are the dropout layer and the activation function which are defined below.

Convolutional layer:-creates a feature map to predict the class probabilities for each feature by applying a filter that scans the whole image, few pixels at a time.

Pooling layer (downsampling):-scales down the amount of information the convolutional layer generated for each feature and maintains the most essential information (the process of the convolutional and pooling layers usually repeats several times).

Fully connected input layer:-“flattens” the outputs generated by previous layers to turn them into a single vector that can be used as an input for the next layer.

Fully connected layer:-applies weights over the input generated by the feature analysis to predict an accurate label.

Fully connected output layer:-generates the final probabilities to determine a class for the image.

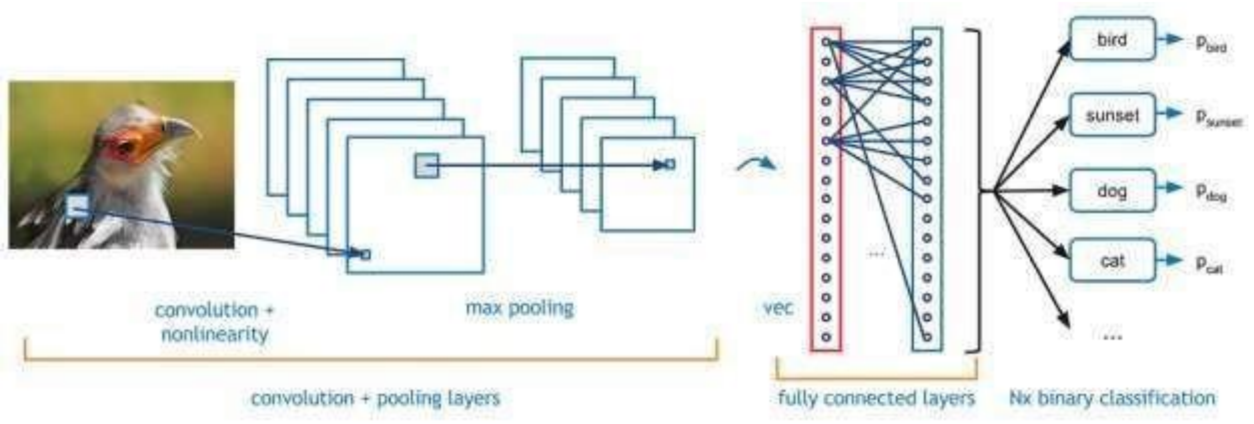
Dropout

Usually, when all the features are connected to the FC layer, it can cause overfitting in the training dataset. To overcome this problem, a dropout layer is utilised wherein a few neurons are dropped from the neural network during training process resulting in reduced size of the model. For example, on passing a dropout of 0.3, 30% of the nodes are dropped out randomly from the neural network.

Activation Functions

Finally, one of the most important parameters of the CNN model is the activation function. They are used to learn and approximate any kind of continuous and complex relationship between variables of the network.

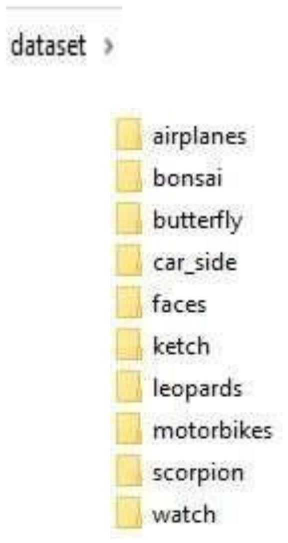
It adds non-linearity to the network. There are several commonly used activation functions such as the ReLU, Softmax, tanH and the Sigmoid functions. For a multi-class classification, generally softmax is preferred.



Procedures we follow

1) Setting up Our Image Data

From Caltech 101 dataset, we select randomly only 10 objects. A total of 3012 images were used from Caltech 101 dataset, which was divided into train and test. we performed an 80-20 split with the train folder having 2408 images and the test folder has 604.
Our dataset structure is as follow:



dataset ›

- airplanes
- bonsai
- butterfly
- car_side
- faces
- ketch
- leopards
- motorbikes
- scorpion
- watch

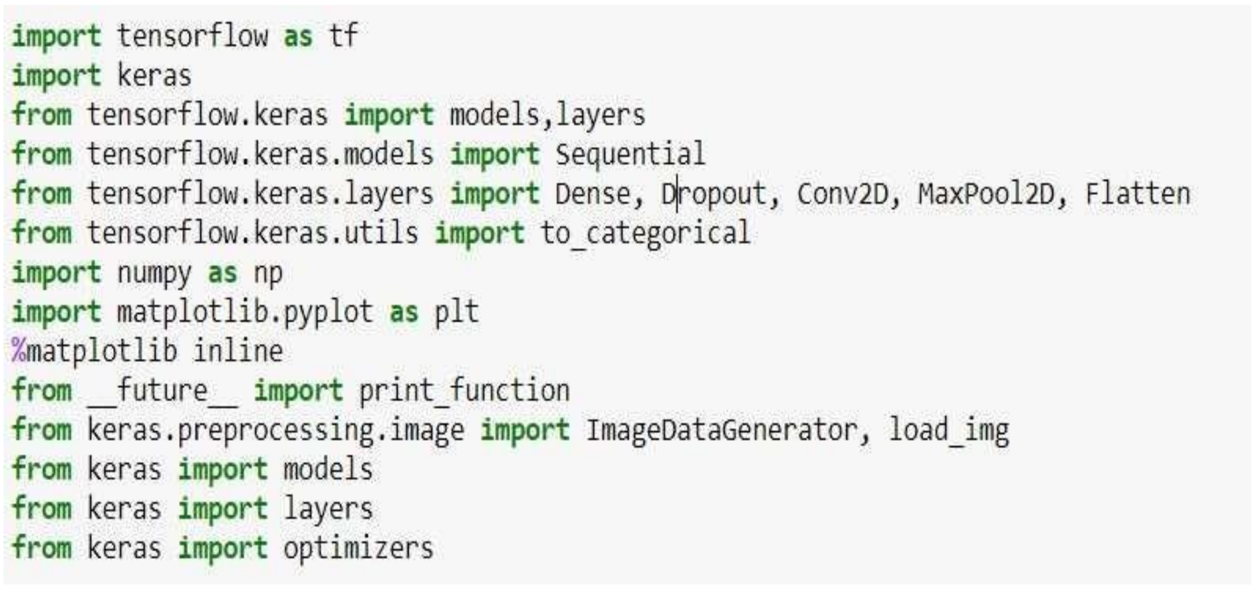
Input – 3012

Train -2408

Test -_604

2) Import the required libraries

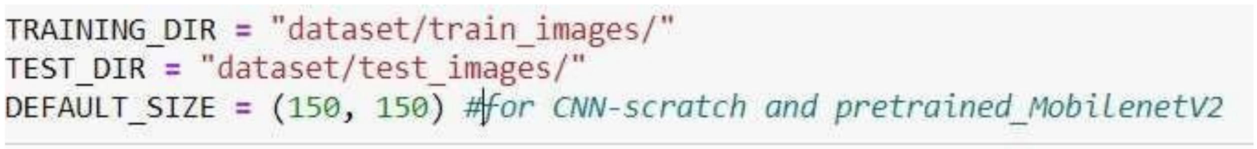
Here we are used the Keras library for creating our model and training it. We also use Matplotlib and sklearn for use of metrics and to display the confusion matric of our model.



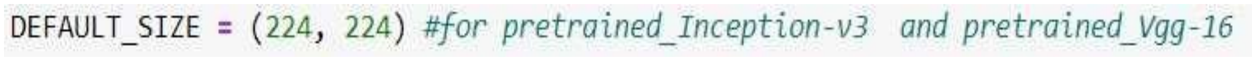
```
import tensorflow as tf
import keras
from tensorflow.keras import models,layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv2D, MaxPool2D, Flatten
from tensorflow.keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from __future__ import print_function
from keras.preprocessing.image import ImageDataGenerator, load_img
from keras import models
from keras import layers
from keras import optimizers
```

3) Loading the data

Next, we define the path to our data and resize the images to our desired width and height and all the images are in the RGB format.



```
TRAINING_DIR = "dataset/train_images/"
TEST_DIR = "dataset/test_images/"
DEFAULT_SIZE = (150, 150) #for CNN-scratch and pretrained_MobilenetV2
```



```
DEFAULT_SIZE = (224, 224) #for pretrained_Inception-v3  and pretrained_Vgg-16
```

4) Data Augmentation

We applied data augmentation for training and testing before we were proceeded with building the model for our pre-trained model such as inception-v3 and vgg-16.

5) Define the Model

In this project, we tried to understand different CNN architectures. We defined five models for the same objects and datasets to understand and to make comparison which is better for image classification. We have used transfer learning to improve the performance of classification.

Our models are:

CNN-scratch with unbalanced_wegiht

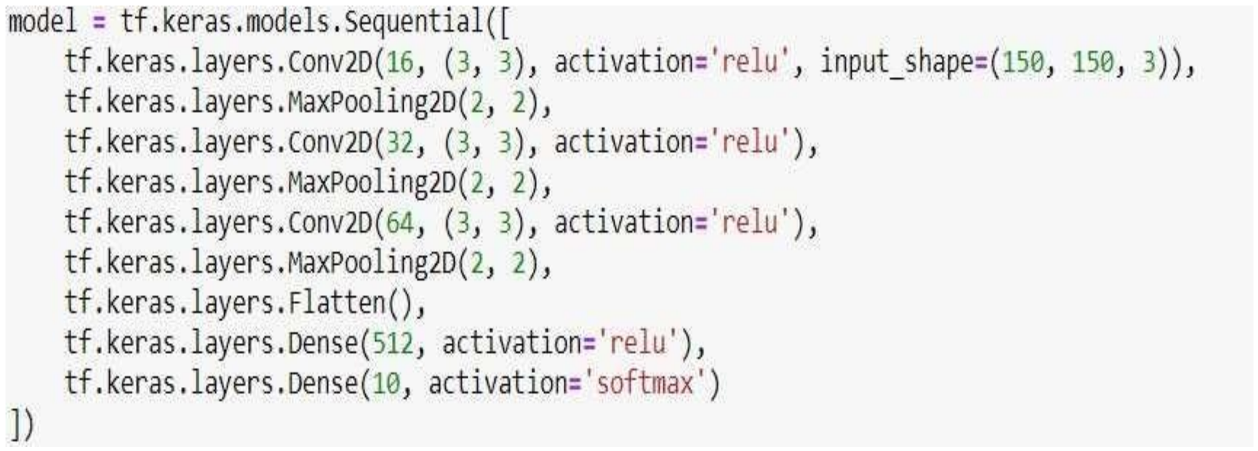
CNN-scratch with balanced_weight

Pretrained_MobilenetV2

Pretrained_Vgg-16

Pretrained_Inception-v3

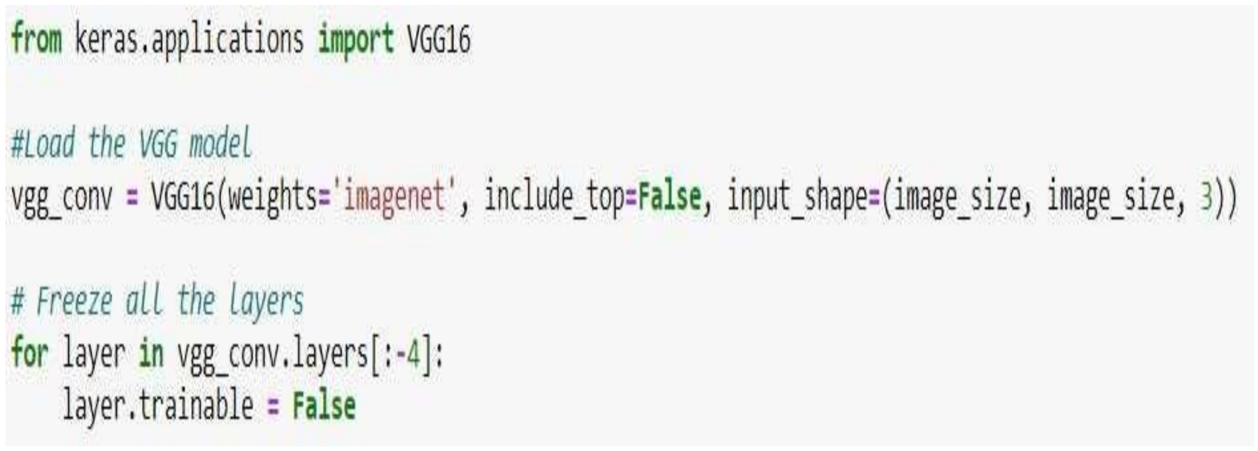
Here, we define a simple CNN model with 3 Convolutional layers followed by max-pooling layers and 3 fully connected layers. We used ‘relu and ‘softmax’ activation functions for our model.



```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

For transfer learning models we load the weights of pretrained model as follows:

For Pretrained _VGG-16:



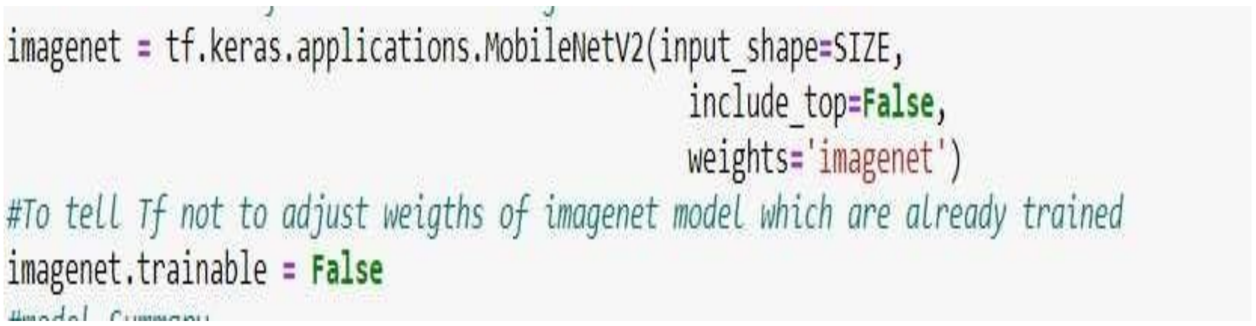
```
from keras.applications import VGG16

#Load the VGG model
vgg_conv = VGG16(weights='imagenet', include_top=False, input_shape=(image_size, image_size, 3))

# Freeze all the layers
for layer in vgg_conv.layers[:-4]:
    layer.trainable = False
```

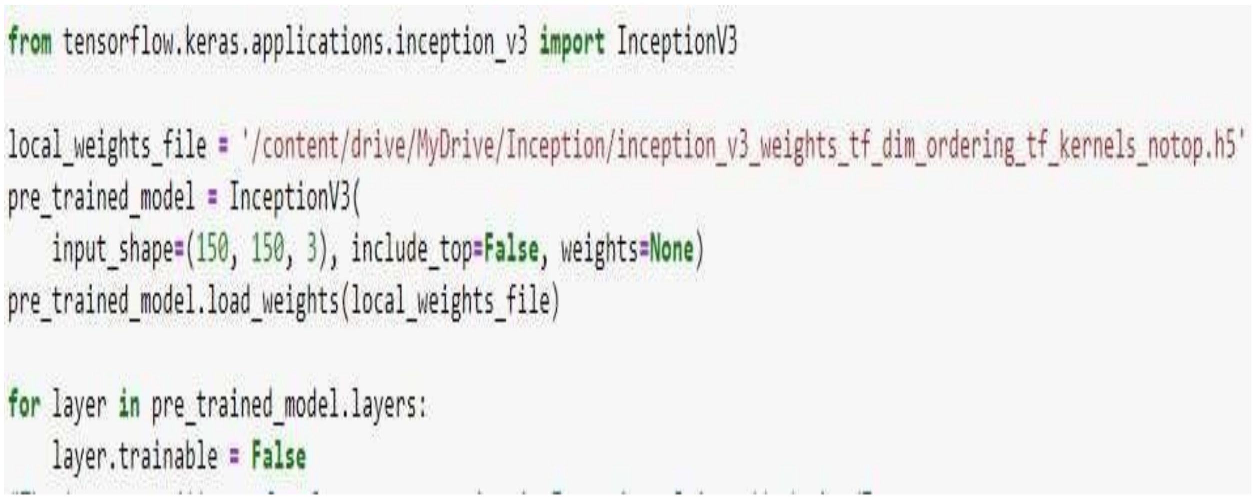
- ○ We used max pooling and ‘softmax’ activation function. ○ A drop out layer is added to avoid overfitting.

For Pretrained_MobilenetV2:


```
imagenet = tf.keras.applications.MobileNetV2(input_shape=SIZE,
                                             include_top=False,
                                             weights='imagenet')
#To tell Tf not to adjust weigths of imagenet model which are already trained
imagenet.trainable = False
```

- We used global average pooling and ‘softmax’ activation function.

For Pretrained_Inception-V3:


```
from tensorflow.keras.applications.inception_v3 import InceptionV3

local_weights_file = '/content/drive/MyDrive/Inception/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'
pre_trained_model = InceptionV3(
    input_shape=(150, 150, 3), include_top=False, weights=None)
pre_trained_model.load_weights(local_weights_file)

for layer in pre_trained_model.layers:
    layer.trainable = False
```

- We used max pooling and ‘softmax’ activation function ○ A drop out layer is added to avoid overfitting.

For model **CNN-scratch** and **Pretrained _MobilenetV2**, we compile the model using Adam as our optimizer and SparseCategoricalCrossentropy as the loss function. we used a learning rate of 0.0001 for a smoother curve.

For model **Pretrained_Vgg-16** and **Pretrained_Inception-v3**, we compile the model using RMSprop and CategoricalCrossentropy as the loss function with a learning rate of 0.0001 for a smoother curve.

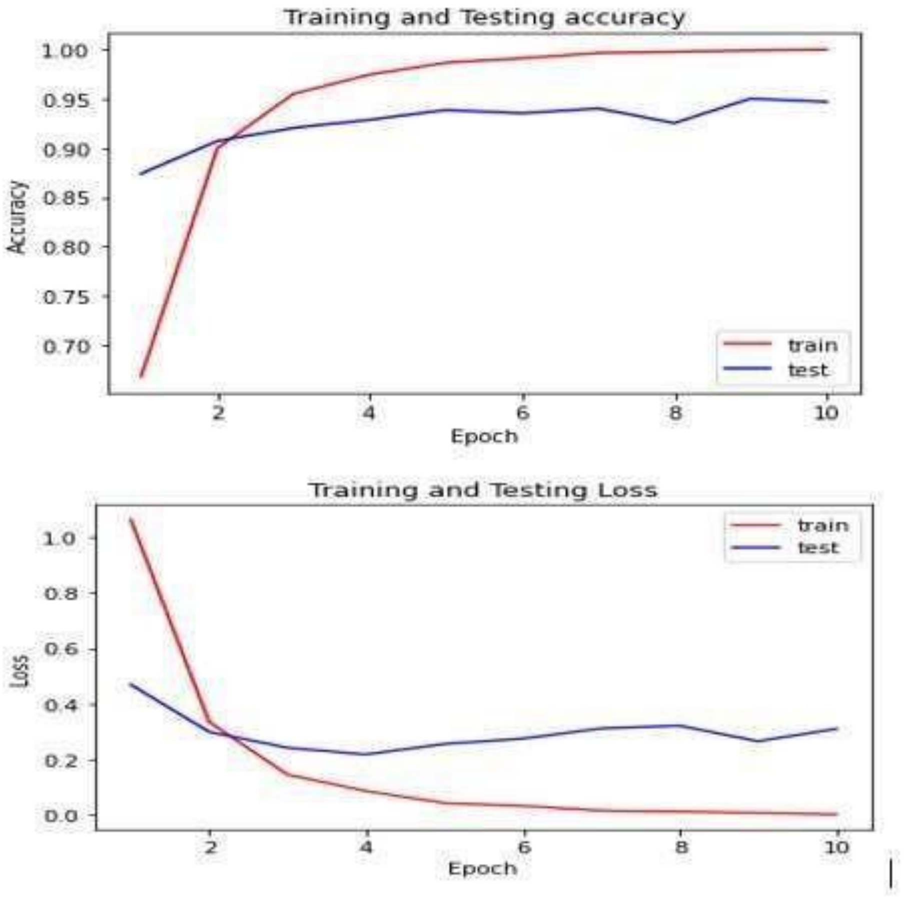
Now, we trained our model such as **CNN-scratch** and Pretrained**_MobilenetV2 for** 10 epochs and for **Pretrained_Vgg-16** and **Pretrained_Inception-v3** we trained for 50 epochs. Since our learning rate is small the number of epochs should be large but for our case, we couldn't run above 50 epochs due to our laptop performance because it needs more RAM.

6) Evaluating Results

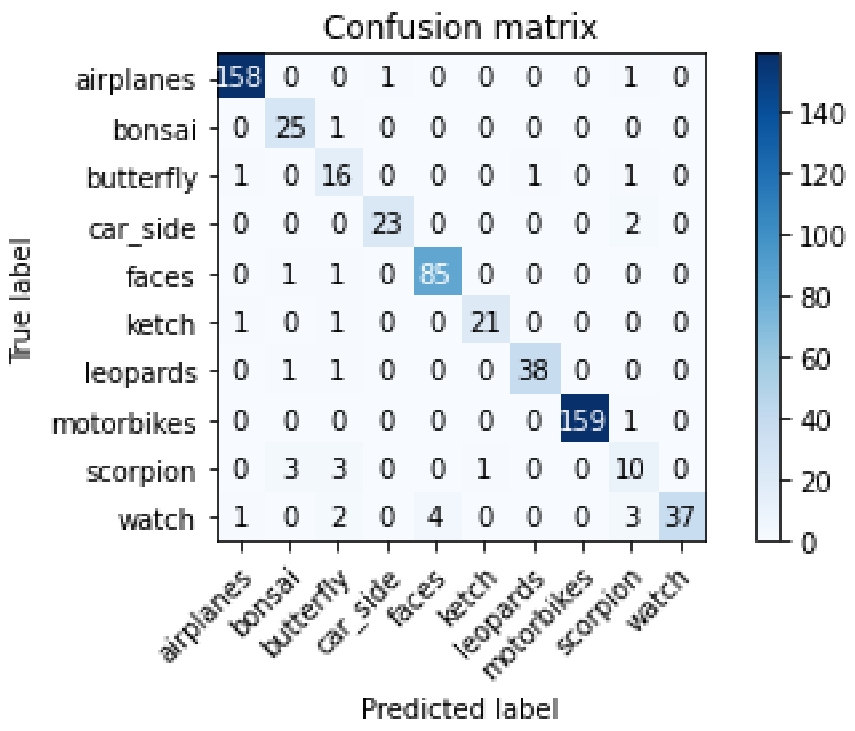
As we all know CNN is better for image classification and we get a better result compared to bag of visual words with the same objects.

Now, we plot our training and validation accuracy along with training and validation loss for each model as follows.

1) CNN-scratch with unbalanced_wegiht model

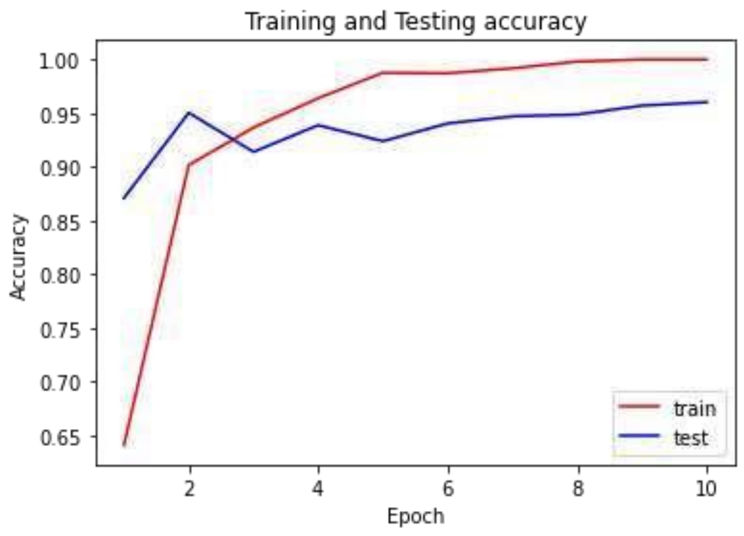


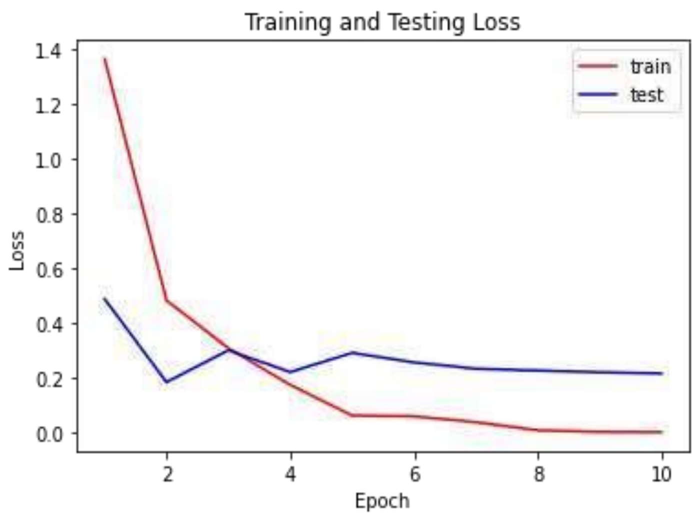
We get a total of accuracy **94.70 %.** From 604 test images this model predicts **32/604** of them incorrectly. The confusion matrix of the model is as follow:



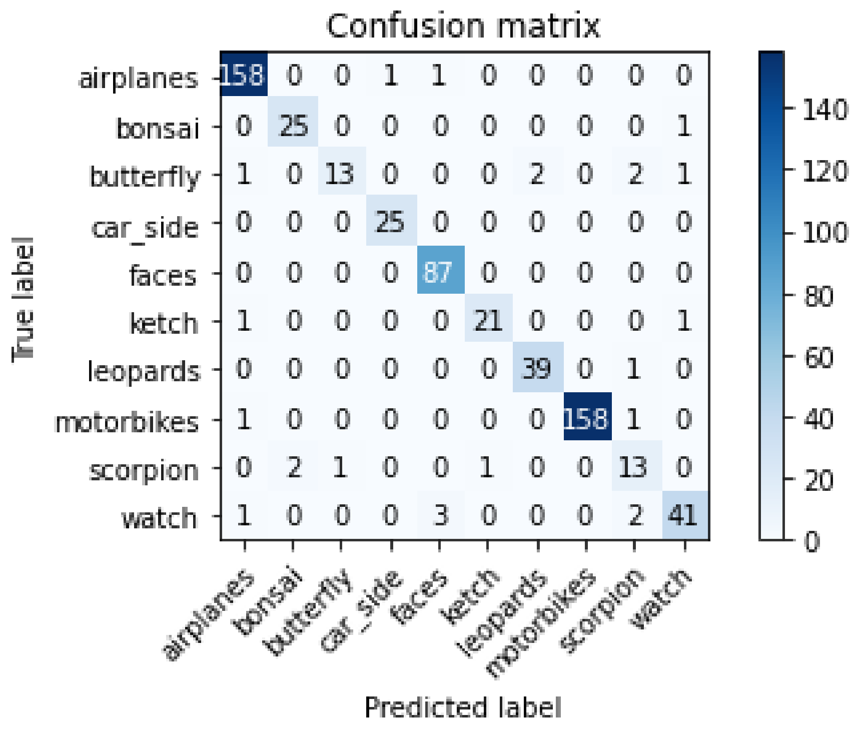
True label \ Predicted label	airplanes	bonsai	butterfly	car_side	faces	ketch	leopards	motorbikes	scorpion	watch
airplanes	158	0	0	1	0	0	0	0	1	0
bonsai	0	25	1	0	0	0	0	0	0	0
butterfly	1	0	16	0	0	0	1	0	1	0
car_side	0	0	0	23	0	0	0	0	2	0
faces	0	1	1	0	85	0	0	0	0	0
ketch	1	0	1	0	0	21	0	0	0	0
leopards	0	1	1	0	0	0	38	0	0	0
motorbikes	0	0	0	0	0	0	0	159	1	0
scorpion	0	3	3	0	0	1	0	0	10	0
watch	1	0	2	0	4	0	0	0	3	37

2) CNN-scratch with balanced_weight model

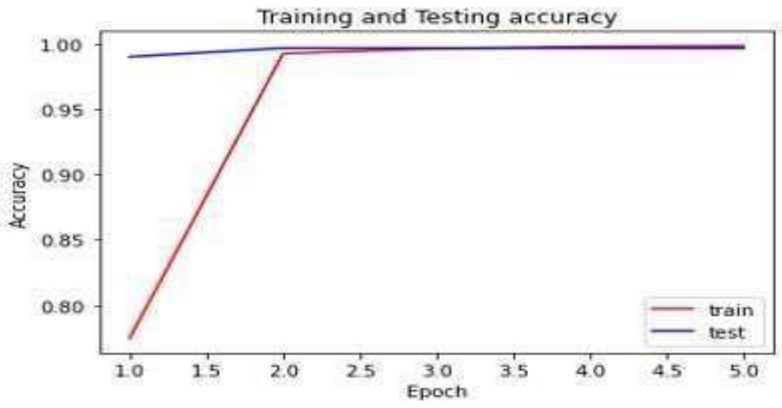


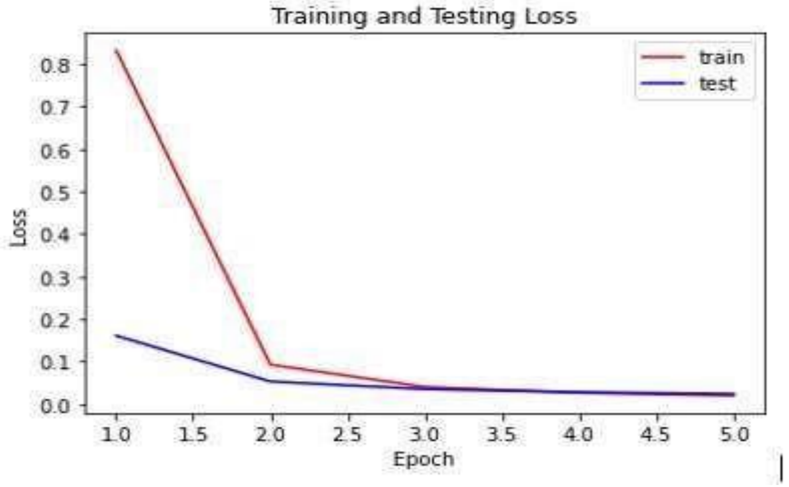


We get a total of accuracy **96.03%.** From 604 test images this model predicts 24 of them incorrectly. As we see from the result, if our dataset is not balanced, we should perform class weight to get better accuracy so we get **1.33 %** accuracy difference with class_wieght and without class_wieght. The confusion matrix of the model is as follow:

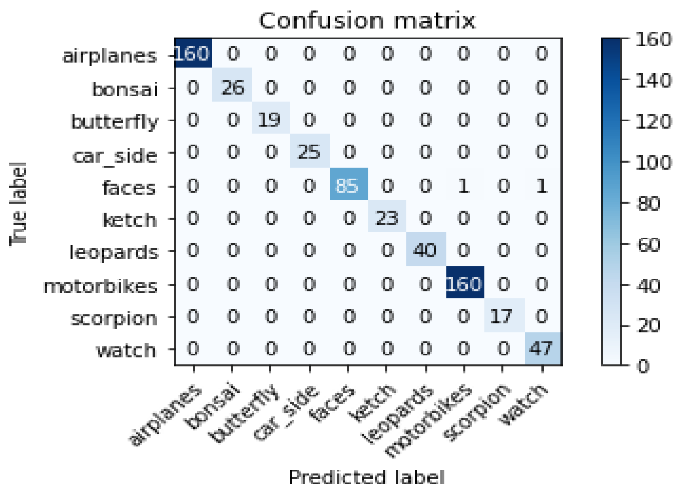


3) **Pretrained_Mobilenetv2 model**



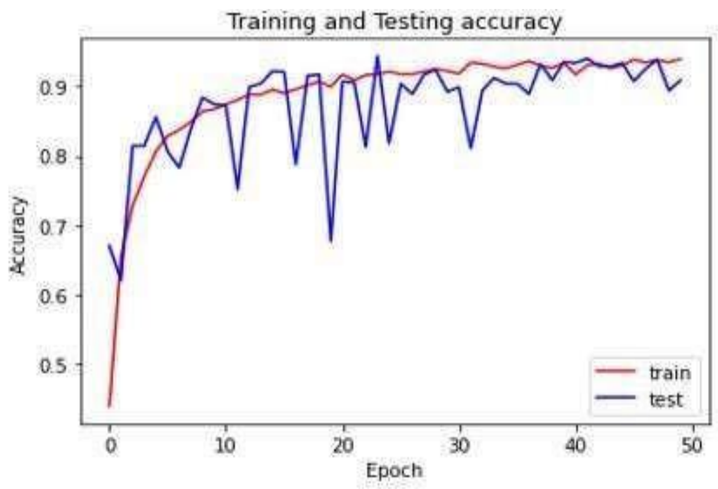


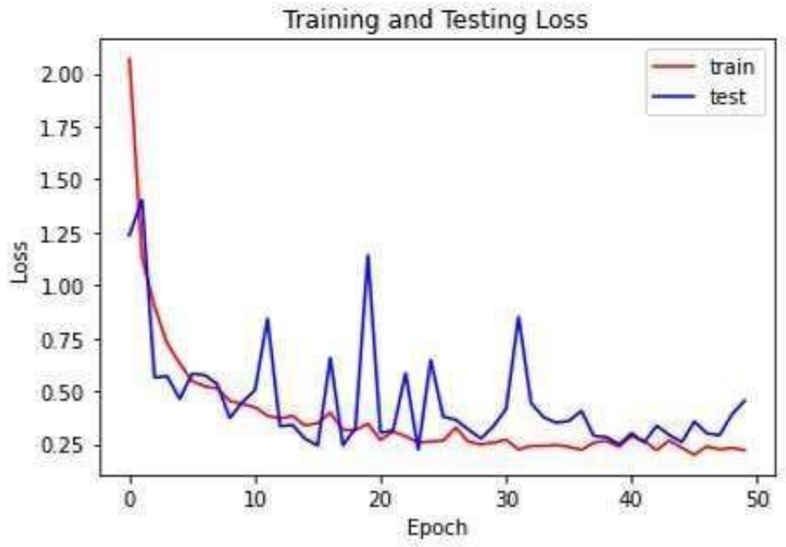
We get a total of accuracy **99.67 %.** From 604 test images this model predicts 2 of them incorrectly. As we can see with transfer learning, we were able to get a much better result. With a bit of hyperparameter tuning and changing parameters, we might be able to achieve a little better performance too! The confusion matrix of the model is as follow:



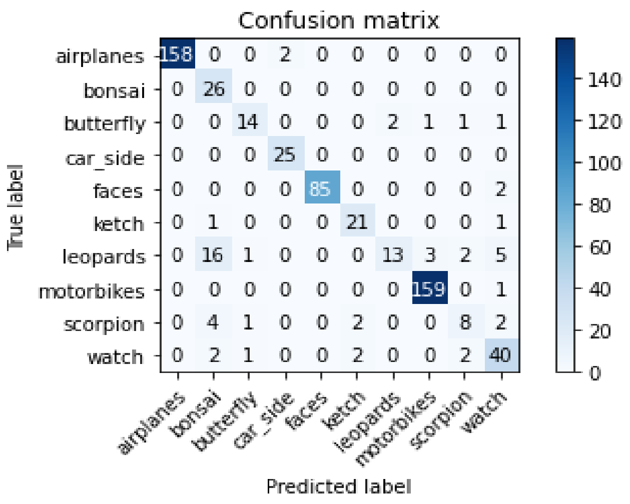
True label \ Predicted label	airplanes	bonsai	butterfly	car_side	faces	ketch	leopards	motorbikes	scorpion	watch
airplanes	160	0	0	0	0	0	0	0	0	0
bonsai	0	26	0	0	0	0	0	0	0	0
butterfly	0	0	19	0	0	0	0	0	0	0
car_side	0	0	0	25	0	0	0	0	0	0
faces	0	0	0	0	85	0	0	1	0	1
ketch	0	0	0	0	0	23	0	0	0	0
leopards	0	0	0	0	0	0	40	0	0	0
motorbikes	0	0	0	0	0	0	0	160	0	0
scorpion	0	0	0	0	0	0	0	0	17	0
watch	0	0	0	0	0	0	0	0	0	47

4) **Pretrained_Vgg-16 model**

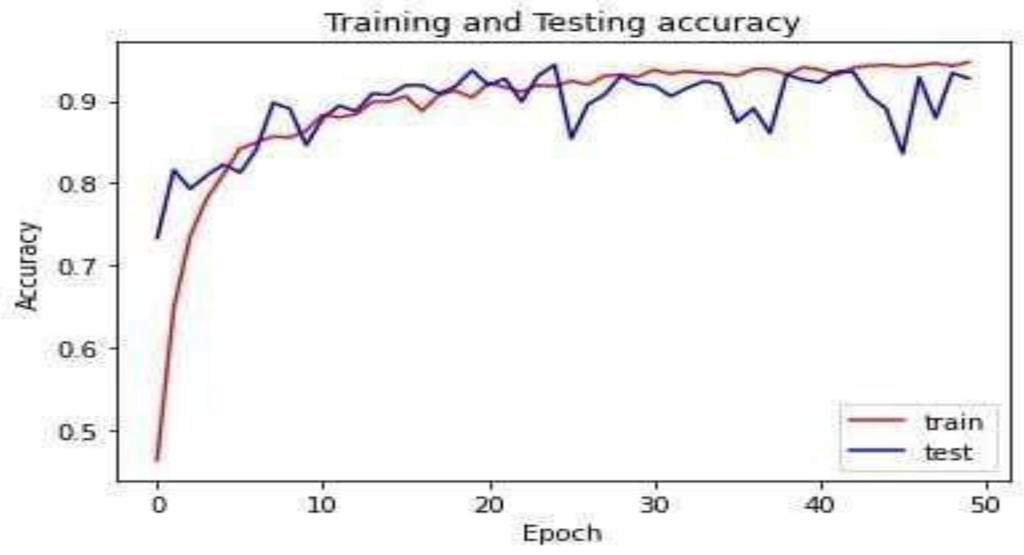


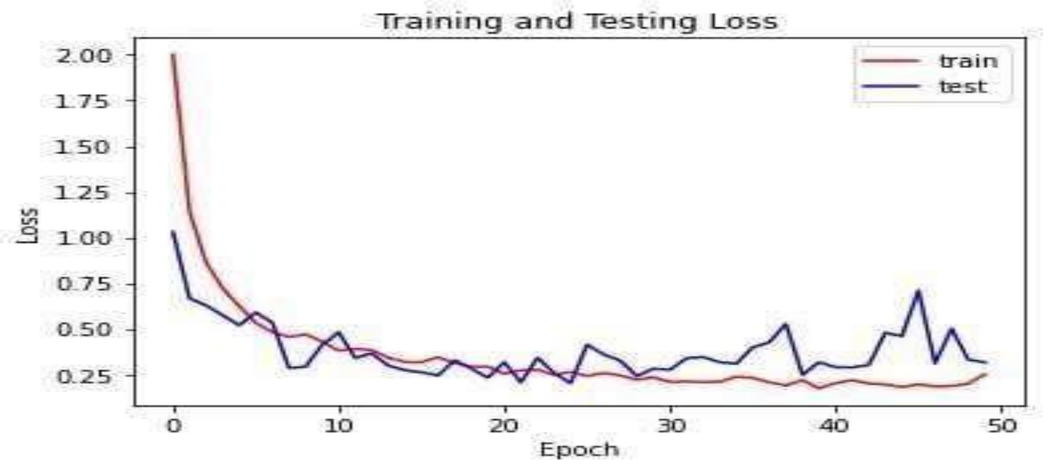


We get a total of accuracy **90.87 %**. From 604 test images this model predicts 55 of them incorrectly. Even though we apply transfer learning and data augmentation we get lower accuracy result compared the above three models. This is due to the lower learning rate and small number of epochs we used due to resource. If we train the model with hundreds of epochs, we will get higher accuracy compared to above models. The confusion matrix of the model is as follow:

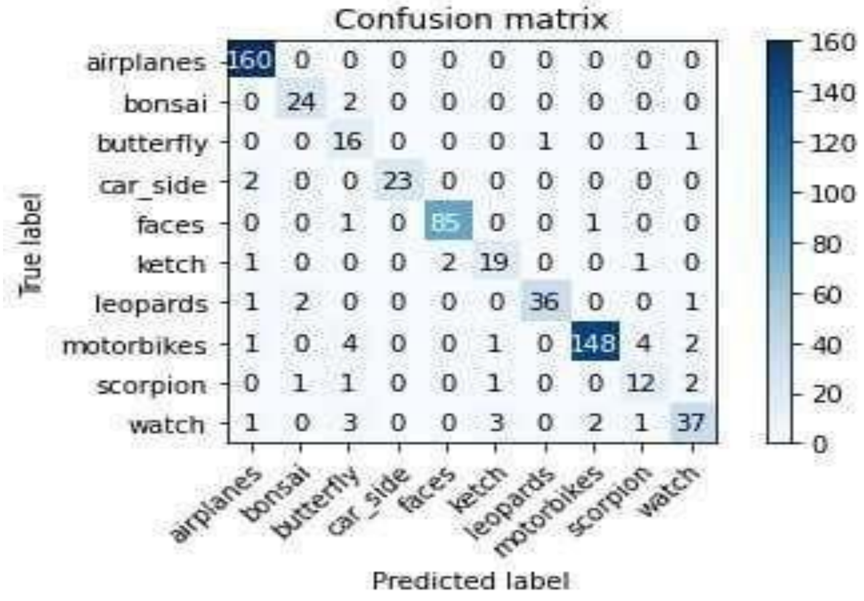


5) **Pretrained_Inception-V3 model**





We get a total of accuracy 92.72%. From 604 test images this model predicts 44 of them incorrectly. Compared to pretrained_Vgg-16 with equal number of epochs and batch size, we get better result using pretrained_Inception-v3. So, we can conclude that the inception-v3 is better for pre-trained image classification. Even though we apply transfer learning and data augmentation we get lower accuracy result compared to CNN_scratch and MobilenetV2 models. This is due to the lower learning rate and small number of epochs we used due to resource. If we train the model with hundreds of epochs, we will get higher accuracy compared to above models. The confusion matrix of the model is as follow:



	airplanes	bonsai	butterfly	car_side	faces	ketch	leopards	motorbikes	scorpion	watch
airplanes	160	0	0	0	0	0	0	0	0	0
bonsai	0	24	2	0	0	0	0	0	0	0
butterfly	0	0	16	0	0	0	1	0	1	1
car_side	2	0	0	23	0	0	0	0	0	0
faces	0	0	1	0	85	0	0	1	0	0
ketch	1	0	0	0	2	19	0	0	1	0
leopards	1	2	0	0	0	0	36	0	0	1
motorbikes	1	0	4	0	0	1	0	148	4	2
scorpion	0	1	1	0	0	1	0	0	12	2
watch	1	0	3	0	0	3	0	2	1	37

Conclusion

In deep learning, transfer learning is a technique whereby a neural network model is first trained on a problem similar to the problem that is being solved. Transfer learning has the advantage of decreasing the training time for a learning model and can result in lower generalization error.

The biggest advantage of transfer learning is that you can load a pre-trained version of the network trained on more than a million images from the ImageNet database. A pre-trained network can classify images into thousands of object categories. Due to this advantage, we are going to apply this model on our image dataset that has 10 object categories.

Keras provides convenient access to many top performing models on the ImageNet image recognition tasks such as VGG, Inception, and MobileNet.
To improve the accuracy of the model we can increase our training set by using different augmentation techniques, building more complex layers, try using different regularization techniques suing a grid of parameters for momentum parameters, regularization parameters, rmsprop parameters, learning rates, run for a greater number of epochs trying different batch sizes.

Comparison Table

Model	**Optimizer**	**Activation function**	**Batch size**	**Epoch**	**Loss function**	**Accuracy**
CNN_scratch unbalanced_weight	Adam	softmax	32	10	sparse_categorical_ crossentropy	94.70%
CNN_scrach balanced_weight	Adam	softmax	32	10	sparse_categorical_ crossentropy	96.03%
Pretrained_Vgg-16	RMSprop	softmax	32	50	categorical_crossentropy	90.87%
Pretrained_ MobilenetV2	Adam	softmax	100	10	sparse_categorical_ crossentropy	99.67%
Pretrained_Inception-V3	RMSprop	softmax	32	50	categorical_crossentropy	92..72%

References

1. Brownlee, J., 2021. Convolutional Neural Network Model Innovations for Image Classification. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/review-of-architectural-innovations-for convolutional-neural-networks-for-image-classification/> [Accessed 07 March 2021].
2. Pythonprogramming.net. 2021. Python Programming Tutorials. [online] Available at: <https://pythonprogramming.net/convolutional-neural-network-deep-learning-pythontensorflow-keras/> [Accessed 10 March 2021]
3. Python Programming Tutorials. [online] Available at: /<https://pythonprogramming.net/convolutional-neural-network-deep-learning-pythontensorflow-keras/> [Accessed 10 March 2021].