# UNIVERSITA DEGLI STUDI DI NAPOLI,FEDERICO II

Corso Umberto I 40 - 80138 Naples

A PROJECT REPORT

ON

## "450 Birds Species classification with CNN"

Submitted in partial fulfillment of the requirements for the award of the Degree of

**Master's Degree**
**In**
**Data Science**

Submitted by


| | |
|---|---|
| **Pooja Priya** | **P37000097** |
| **Aryan Garg** | **P37000111** |
| **Shubham Kanwar** | **P37000100** |


**Under The Guidance Of**


**Prof. Giuseppe Longo**
Department of Physics


**Data Science**

2021-2023

**Abstract**

In this project we were expected to do birds species classification on the 450 species of birds images taken from kaggle datasets(kaggle link). The data is actually is huge in size taking around 1.8GB of disk space. It contains 450 species of birds that we have to classify. Here, we we expected to use Deep Learning with CNN to classify images in particular. So, we used the Convolutional Neural Networks(CNN) with transfer learning to train our classification model. Although we're not the first one to use transfer learning in classification of images, but this method really comes handy when we don't have much images at our disposal to train the network, we can use already trained model to transfer the already learned features and use it for our model and get better performance. In this project we used Keras API which runs on top of machine learning platform Tensorflow. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible. Keras fast, flexible and Powerful.

Keywords - Tensorflow, Keras, Deep Learning, Transfer learning, Neural Networks, VGG16, Data loading, Image Classification.

# Contents

# Chapter 1

# Introduction

In this project we will explore how to use Tensorflow library which is a popular machine learning platform to solve an image classification problem of many classes. It comes with broad range of tools and packages, which range from high level abstraction using Keras API. Keras API is build on top of the tensorflow platform. It allows to design and fast implementation of machine learning architectures.It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible. Keras is fast, flexible and Powerful. It combines 4 key abilities.

- Efficiently executing low-level tensor operations on CPU, GPU, or TPU.
- Computing the gradient of arbitrary differentiable expressions.
- Scaling computation to many devices, such as clusters of hundreds of GPUs.
- Exporting programs ("graphs") to external runtimes such as servers, browsers, mobile and embedded devices.

Workflow is based on the instructions mentioned below.

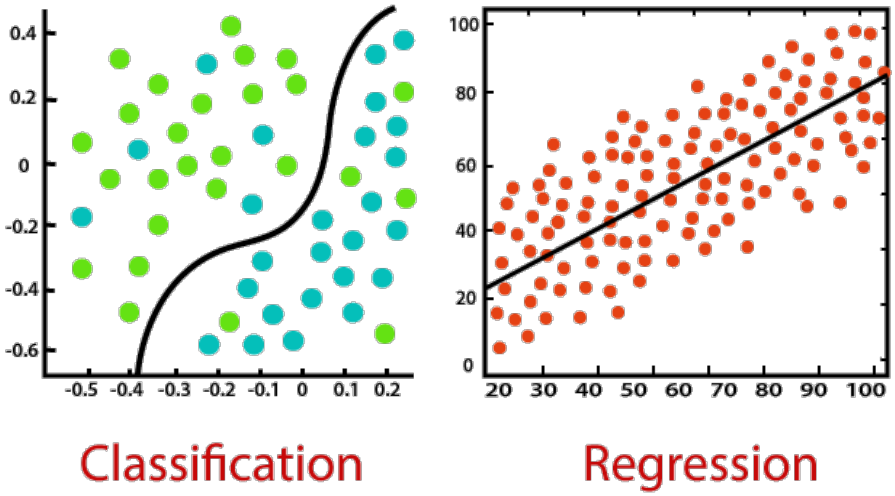
- Data Exploration and visualizing the data we have.
- Explaining the data with looking at classes/species.
- Visualize the random images to see how our classes look.
- Downloading a pretrained model VGG-16.
- Using the image data generator for data loading.
- Preparing an image classification Convolutional Neural Network(CNN) and train it on our keras pretrained model VGG16.
- Fine tuning the model with training last 5 layers of our VGG16 model.
- Assessment of classification model on test data with accuracy measure.
- Check the model prediction with visualization some images.

In this introduction we will try to review the classification with traditional methods which is by machine learning and what is image classification as a whole, and what is image classification with deep learning. Following that, we will introduce our 450 species birds datasets and later we will review and choose those techniques to produce the images classification model for our birds datasets.

# Chapter 2

# What is Classification ?

## 2.1 Predictive model and classification




Predictive modelling is problem of developing the model using the historic data to make prediction on new data(which is not seen by the model). It can also be defined by the mathematical problem of approximating a mapping function(f) from in put variables(X) to the output variables (y). The job of modelling algorithm is to find the best mapping function we can given the time and the resources available. **Classification** is when we want to predict object label/class and **Regression** is when we want to predict the magnitude of specific quantity. As evident from the figure above the difference between the 2 approaches, where classification predicts the class value for a given data point on basis of its location w.r.t. class boundary. The regression model is designed to predict the actual quantity of something give the set of predictors. **Classification** modelling is task of approximating the function(f) from input variable(X) to discrete output variables(y). The output variables are often called labels or categories. The mapping function predicts the labels of class for a given observation. For example email of text can be classified as belonging to one to 2 categories: "Spam" or "not spam".

- A Classification problem required that the examples can be classified into one of two or more classes.

- A classification can have real values or discrete input variables.
- A problem with 2 classes are called binary classification, and problem with more than 2 classes are called multi-class classification problem.
- A problem where an example is assigned multiple classes is called a multi label classification problem.

It common for classification models to predict continuos values as a probability of given example belonging to each output class. The probabilities can be interpreted as the likelihood or confidence of a given example belonging to each class. A predicted probability can be converted into a class value by selecting the class label that has the highest probability.

In this project we are going to focus more on images classification problem.

## 2.2 What is image classification?

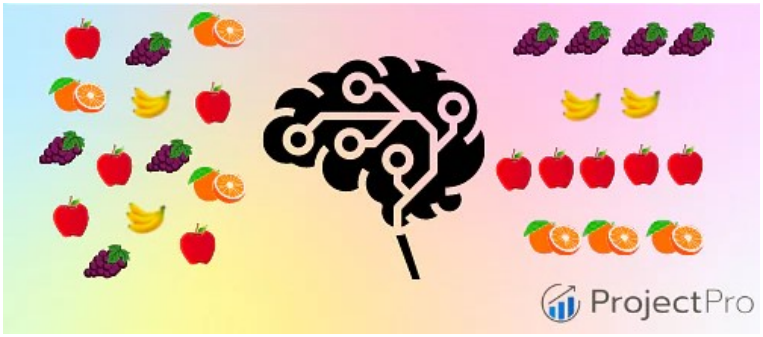



Figure 2.1: image classification of objects, where image classification is based on contents of image

Image classification is sub domain of the general problem of classification. An algorithm designed for image classification accepts images as input and produces the prediction of the class of image as output. The output can take the form of label or category, or it can be set of real-values probabilites representing the likelihodd of each potential class belonging to an image.For example in image above an algorithm is designed to identify the type of fruits it is by taking an image of set of fruits as an input.

In order to successfully classify an image the algorithm needs to extract the important features and allows it learn the prespective patterns that allows it differentiate between the classes. These features can be manually created through an iterative and interactive design process, with the aim of producing the features that best separate the class. Although that method is **Human Intensive**, **Prone to error**, **Exhaustive** and there's no gurantee of unique features with optimally distinguish between classes.

Another approach has been developed from almost more than a decade, which has seen a considerable improvement in image classification is the use of deep learning, or in particular the Convolutional Neural Networks(CNN). These type of approaches tend to solve both the problem of carefully designing of optimal

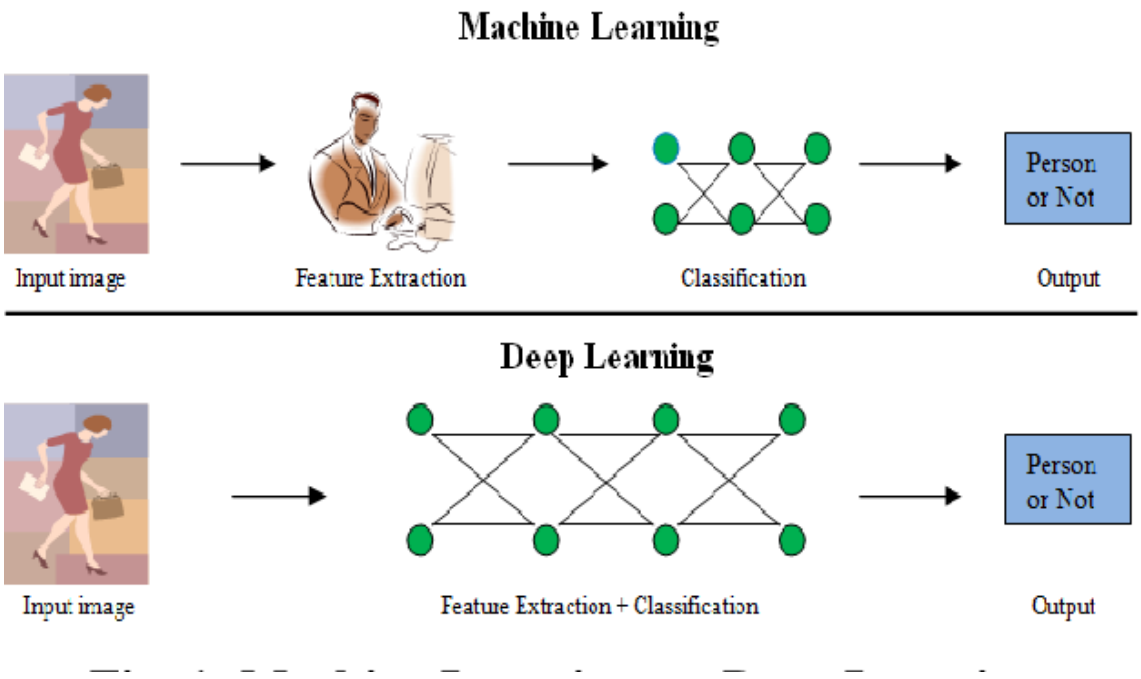



Figure 2.2: Different approaches to image classification(top) machine learning with manually crafted features and (bottom) deep learning using CNN's to automatically derive both idealized features and classifier in one process

features and the classifier that predicts the label, at the same time. In the simplest form CNN is huge, automatically learnt feature extraction system, with a classifier tagged on the end.

In fact it is this very structure that allows for methodology of **Transfer Learning**(figure2.3). This is the process of removing the classification layer at the base of the network, and using the network as the feature extractor. The outputs of these features and then be fed up into any machine learning classifier, to preform class perdictions. In this project we used the pre-trained famous VGG16 model which is trained on the huge set of images from the "imagenet". VGG16 is object detection and classification algorithm which is able to classify 1000 images of 1000 different categories with more than 93 percent accuracy.

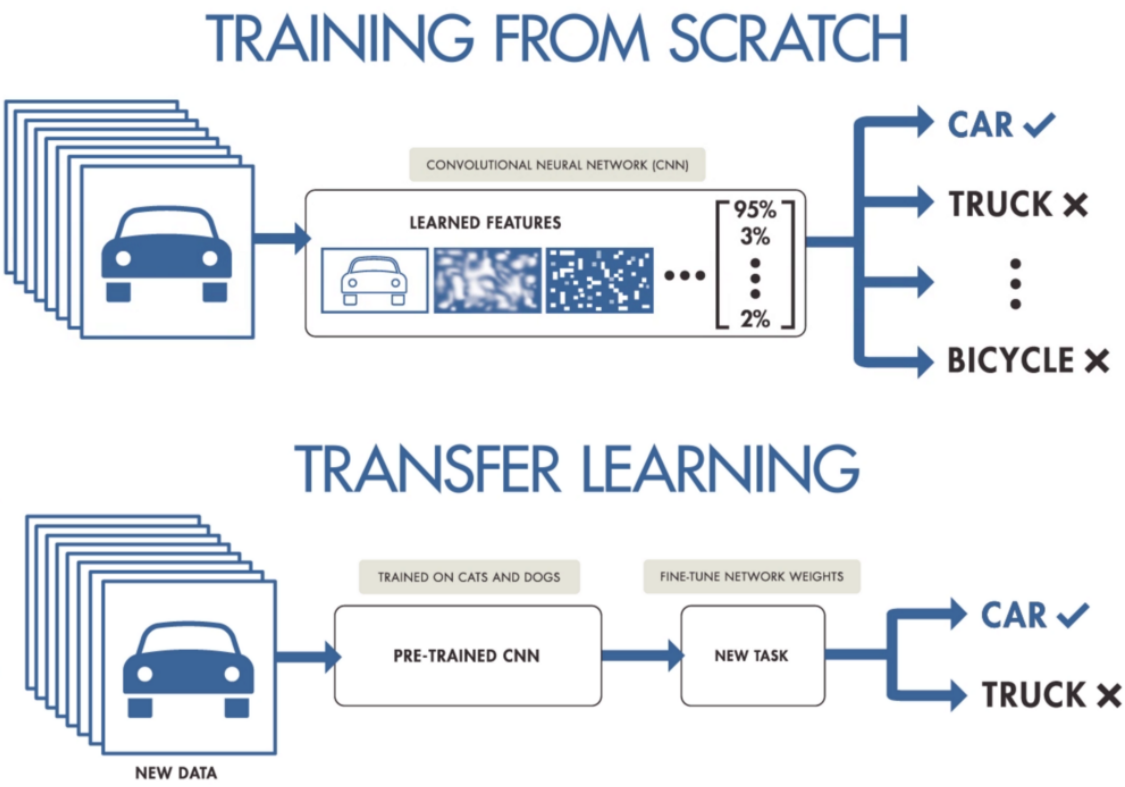



Figure 2.3: Concepts of training a CNN from scratch(top) versus the transfer learning(bottom). In transfer learning the pre-trained network is used the feature extraction network, by removing the final layer classifier and the using the extracted feature outputs as input into a new classifier.

# Chapter 3

# The Data- 450 birds species

This dataset was taken from the kaggle repository of datasets. This is data of 450 species of brids from all over the world. This dataset is a challenging in many ways as the most of the birds of some species has the same visual appearance. Birds species identification can itself be problem for humans let alone for a computer vision algorithms, Hence this type of problem is often reffered to as large scale fine-grained. As is evident form the picture that follows, it is practically challenging to tell form the picture which species of humming bird are those?

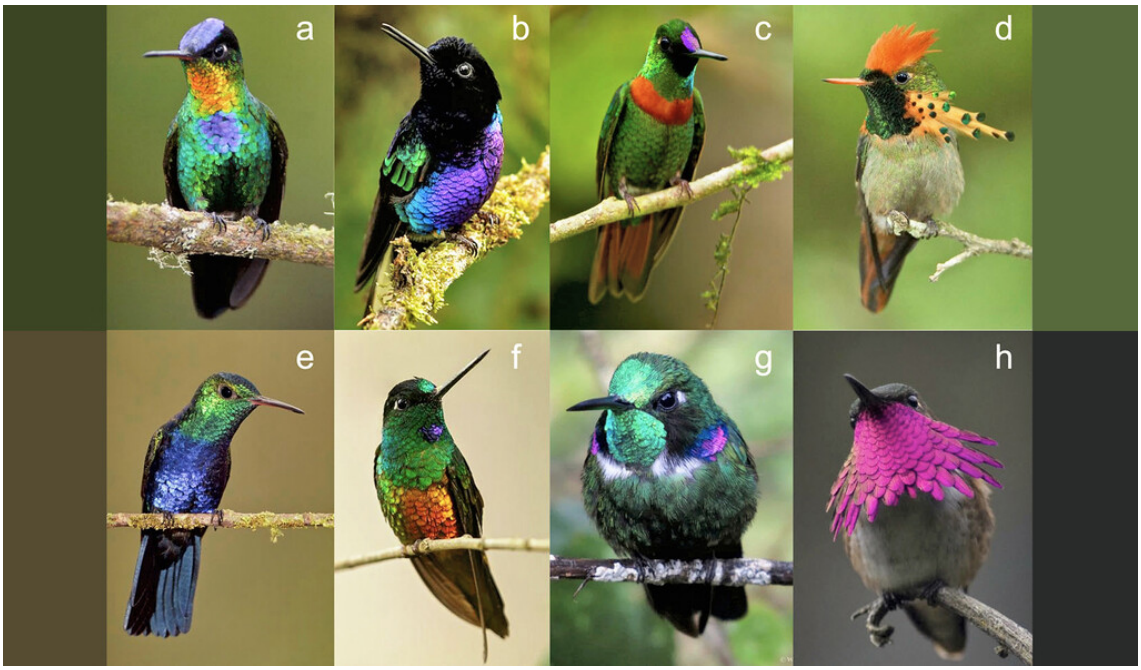



Figure 3.1: Different species of hummingbird looking not very different from each other

The data contains, 70626 training images, 2250 test images(5 images per class), and 2250 valid images(5 images per class). This is a very high quality dataset where there is only one bird in each image and the bird typically takes up at least 50% of the pixels in the image. As a result even a moderately complex model will achieve training and test accuracies in the mid 90% range. All images are 224 X 224 X 3 color images in jpg format. Data set includes a train set, test

set and validation set. Each set contains 450 sub directories, one for each bird species. Images were gather from internet searches by species name. Once the image files for a species was downloaded they were checked for duplicate images using a python duplicate image detector program I developed. All duplicate images detected were deleted in order to prevent their being images common between the training, test and validation sets.

Figure 3.2: Random species brids clubbed together to have a rough look at the dataset

We can have a rough look at the images above of different species clubbed together. The training set is not balanced, having a varying number of files per species. However each species has at least 130 training image files. One significant shortcoming in the data set is the ratio of male species images to female species images. About 80% of the images are of the male and 20% of the female. Males typical are far more diversely colored while the females of a species are typically bland. Consequently male and female images may look entirely different .Almost

all test and validation images are taken from the male of the species. Consequently the classifier may not perform as well on female specie images.

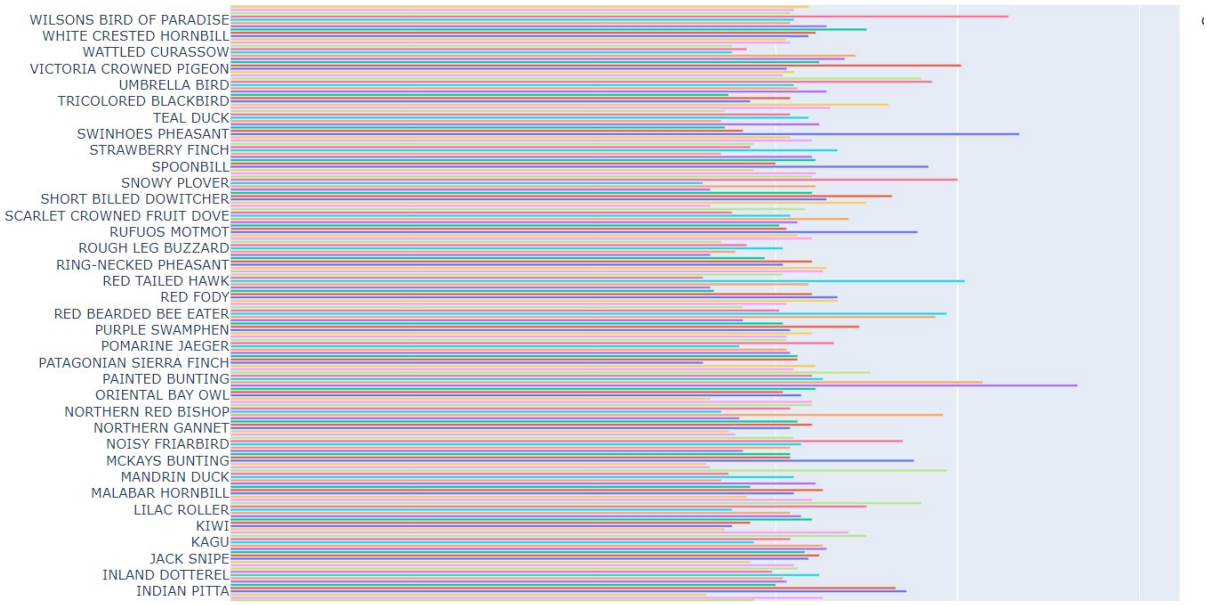



Figure 3.3: random distribution of birds species and their respective images(screenshot)

As evident from (figure3.3), our classes are not balanced, but average average images per class in training data is 156.
Although the data is balanced for test and validation images as all the classes contains 5 image each for testing and validation purposes. Distribution is shown in image below(figure3.4).
All the images are stacked in respective directories of train, test and validation, followed by the names of species in sub directories which contains the images of that particular class.
On top of that 5 random images of birds are provided to test our model in the end for those images, whether it predicts the correct class.


Figure 3.4: random distribution of test and valid classes with images

# Chapter 4

# Transfer Learning

In this project we used VGG16 model, which is also named as ConvNet which is kind of artificial neural network. A convolutional neural network has an input layer, and various hidden layers. VGG16 is a type of CNN(Convolutional Neural Network) that is considered to be one of the best computer vision models to date. We're using the pre-trained model because we have considerably less images per class for model to learn the most important features. As most of our class has around 150 images. We would need a model pretrained model where we can use the the pretrained weights to apply it to our model for better results.

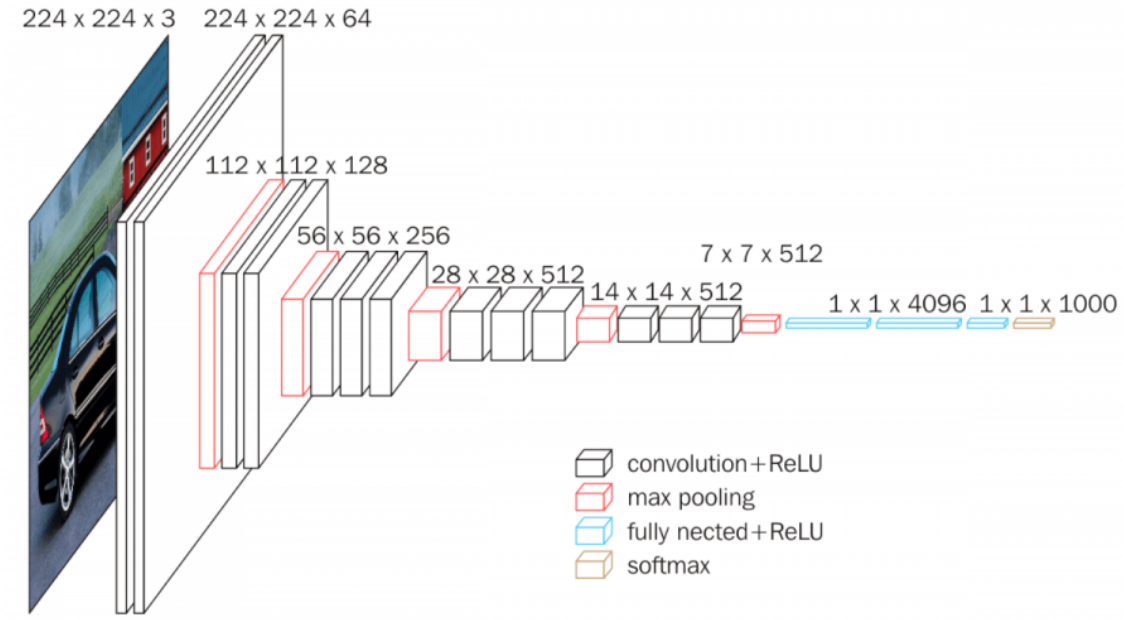



Figure 4.1: VGG16-architecture

Here let's explore VGG16 architecture closely In the figure above, all the white rectangles represent the convolution layers along with the non-linear activation function which is a rectified linear unit (or ReLU). As can be seen from the figure that there are 13 white and 5 red rectangles i.e there are 13 convolution layers and 5 max-pooling layers. Along with these, there are 3 blue rectangles representing 3 fully connected layers. So, the total number of layers having tunable parameters

is 16 of which 13 is for convolution layers and 3 for fully connected layers, thus the name is given as VGG-16. At the output, we have a softmax layer having 1000 outputs per image category in the imagenet dataset.

In this architecture, we have started with a very low channel size of 64 and then gradually increased by a factor of 2 after each max-pooling layers, until it reaches 512.

The architecture is very simple. It has got 2 contiguous blocks of 2 convolution layers followed by a max-pooling, then it has 3 contiguous blocks of 3 convolution layers followed by max-pooling, and at last, we have 3 dense layers. The last 3 convolution layers have different depths in different architectures.

The important thing to analyze here is that after every max-pooling the size is getting half.

## 4.1 Loading the model

We will start by downloading the model. Trained ImageNet models are available to download directly within the Keras library. You can see the available models and their details here . Any of these models would work for our exercise. We will pick a commonly used one called VGG16:

As we are downloading, there is going to be an important difference. The last layer of an ImageNet model is a dense layer of 1000 units, representing the 1000 possible classes in the dataset. In our case, we want it to make a different classification: 450 species of birds? Because we want the classification to be different, we are going to remove the last layer of the model. We can do this by setting the flag *include_top = False* when downloading the model. After removing this top layer, we can add new layers that will yield the type of classification that we want.

Before we add new layers to our model, we should first freeze the model pre-trained layers. This means that when we train, we will not update the base layers from the pre-trained model.Instead we will only update the new layers that we add on the end for our new classification. We freeze the initial layers because we want to retain the learning achieved from training on the ImageNet dataset. If they were unfrozen at this stage, we would likely destroy this valuable information. There will be an option to unfreeze and train these layers later, in a process called fine-tuning.

Now we will add new trainable layers to the pre-trained model. They will take the features from the pretrained layers and turn them predictions on the new dataset. We will add 2 layers to the model. First will be average pooling layer. Then we will need to add our final layer, which will classify our 450 classes. It will be densely connected layer with 450 output.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 dense (Dense)               (None, 450)               230850

=================================================================
Total params: 14,945,538
Trainable params: 230,850
Non-trainable params: 14,714,688
```

Figure 4.2: base model summary

## 4.2 Loading the Data

Here we use Keras ImageDataGenerator class to rescale our images by dividing it by the maximum pixel value i.e. 255.
After, we are going to load images directly from folders using Keras *flow_from_directory* function. We have set up our directories to help this process go smoothly as our labels are inferred from the folder names.In the */content/birds_dataset* directory, we have train and validation and test directories, which each have folders for images of all our classes. In train directory we have sub-folder with each class names contaning the images of birds for that particular class.

## 4.3 Compiling the model

We have to compile our model with loss and metrics options. Here, as we have many classes in our classification problem, we will use categorical crossentropy which is a logrithmic loss function for calculation of loss and efficient Adam gradient descent optimization.

```
# compiling the model
model.compile(loss = "categorical_crossentropy", optimizer = 'adam', metrics = ['accuracy'])
```

Figure 4.3: Compiling the model

# Chapter 5

# Training the base_model

Finally we arrived where we train our base model.
Key parts of the model.

- train data - the input data for model with labels
- model- a VGG16 model with pre-trained weights from imagenet training and output layer of 450 classes.
- epochs - Number of epochs to train our model on. An epoch is an iteration over the entire data provided.
- steps per epoch - total number of steps(batches of samples) before declaring one epoch finished and starting the new one.
- validation data - Data on which to evaluate the loss and accuracy in the end of each epoch.

Here we will train the model with the keeping the layers frozen and check how good our model is doing.
We will train our model for 10 epochs, batch size is 32 as default. Step size of each epoch is 2208.

```
Epoch 1/10
2208/2208 [==============================] - 364s 160ms/step - loss: 5.1505 - accuracy: 0.1368 - val_loss: 4.2864 - val_accuracy: 0.3143
Epoch 2/10
2208/2208 [==============================] - 362s 164ms/step - loss: 3.9421 - accuracy: 0.3323 - val_loss: 3.4353 - val_accuracy: 0.4430
Epoch 3/10
2208/2208 [==============================] - 362s 164ms/step - loss: 3.3077 - accuracy: 0.4263 - val_loss: 2.8003 - val_accuracy: 0.5276
Epoch 4/10
2208/2208 [==============================] - 363s 164ms/step - loss: 2.8999 - accuracy: 0.4857 - val_loss: 2.6056 - val_accuracy: 0.4963
Epoch 5/10
2208/2208 [==============================] - 362s 164ms/step - loss: 2.6095 - accuracy: 0.5267 - val_loss: 2.2955 - val_accuracy: 0.5754
Epoch 6/10
2208/2208 [==============================] - 362s 164ms/step - loss: 2.3904 - accuracy: 0.5581 - val_loss: 2.1175 - val_accuracy: 0.6011
Epoch 7/10
2208/2208 [==============================] - 362s 164ms/step - loss: 2.2166 - accuracy: 0.5866 - val_loss: 1.9320 - val_accuracy: 0.6011
Epoch 8/10
2208/2208 [==============================] - 362s 164ms/step - loss: 2.0766 - accuracy: 0.6068 - val_loss: 1.8177 - val_accuracy: 0.6489
Epoch 9/10
2208/2208 [==============================] - 362s 164ms/step - loss: 1.9587 - accuracy: 0.6269 - val_loss: 1.7948 - val_accuracy: 0.6471
Epoch 10/10
2208/2208 [==============================] - 362s 164ms/step - loss: 1.8578 - accuracy: 0.6423 - val_loss: 1.6186 - val_accuracy: 0.6728
```

Figure 5.1: training for 10 epochs

As we can see from figure 5.1, with initial training we reached around 67 percent of validation accuracy, which is less if we see the number of classes we have.

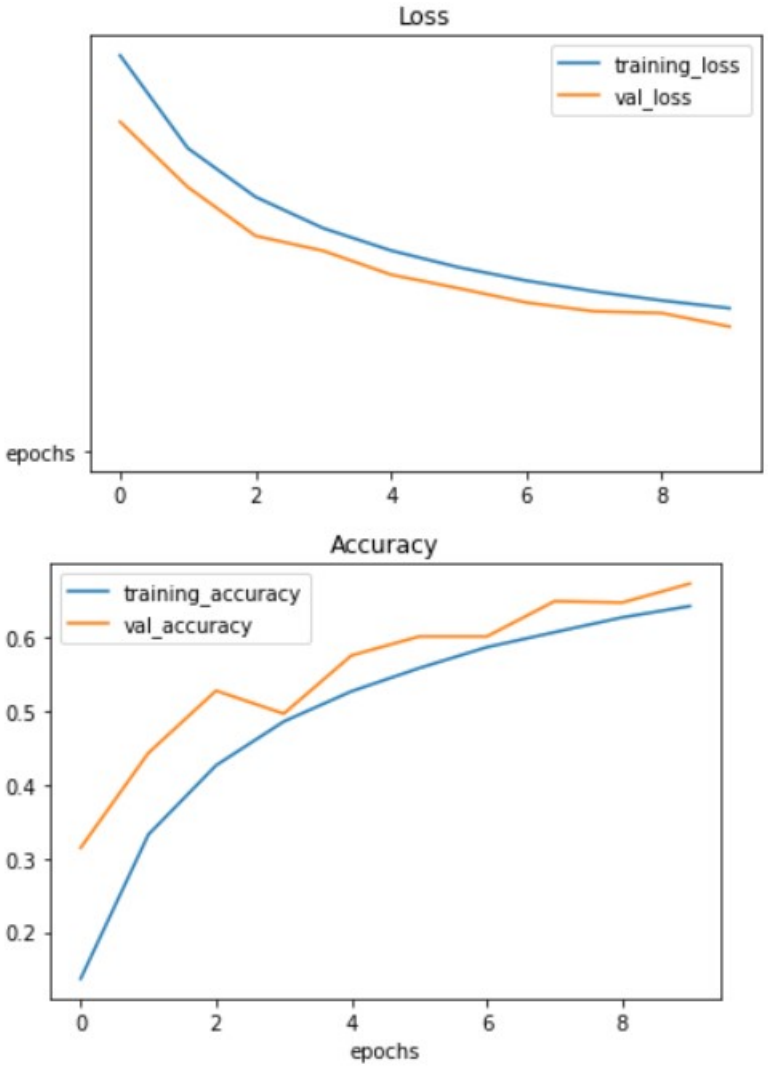



Figure 5.2: Visualization of loss and accuracy with epochs

As the model progresses its common to see both training loss and validation loss decrease with iteration. This indicates that the model is learning, and at the end of each epoch, the model is better at predicting the class labels on the images at both the training and test set. At some point, the validation loss will start to increase, whilst the training loss is continuing to decrease. This is point where model has stopped generalizing, i.e. it has stopped learning the features that will improve its ability to predict class labels on unseen data. in other way model is in a way starting to be over-trained, because the features its learning is becoming more specific to the training set. These features are not replicated in validation set, they may even the specific to the noise specific to the training set, and hence the model performs more poorly as training continue to overfit. Hence the best, model is the model that achieve the lowest validation loss.
As we can see from plot above the validation loss went down over the time but get constant around 6 epochs and above and minimize at 1.85. And accuracy metric increased over each epoch and reached 67 percent on the 10 epoch.

# Chapter 6

# Fine tuning the model

Now our model has stopped learning and generalizing as the validation loss has stopped decreasing and validation accuracy has stopped increasing. Now we will move further and unfreeze our model and make the last 5 layers trainable, and train it again with small learning rate. This will cause the base pretrained layers to take very small steps and adjust slightly, therefore improving the model. Note that it is important to only do this step after the model with frozen layers has been fully trained. The untrained pooling and classification layers that we added to the model earlier were randomly initialized. This means they needed to be updated quite a lot to correctly classify the images. Through the process of backpropagation, large initial updates in the last layers would have caused potentially large updates in the pre-trained layers as well. These updates would have destroyed those important pre-trained features. However, now that those final layers are trained and have converged, any updates to the model as a whole will be much smaller (especially with a very small learning rate) and will not destroy the features of the earlier layers.

Let's try unfreezing the pre-trained layers, and then fine tuning the model:

```
base_model.trainable = True  # let's unfreeze our base model and retrain our last 5 layers
```

```
for layer in base_model.layers[:-5]:
  layer.trainable = False
```

Figure 6.1: Unfreezing the model

```
# let's check which layers are trainable
for i, layer in enumerate(model.layers[1].layers):
  print(i, layer.name, layer.trainable)
```

```
0 input_1 False
1 block1_conv1 False
2 block1_conv2 False
3 block1_pool False
4 block2_conv1 False
5 block2_conv2 False
6 block2_pool False
7 block3_conv1 False
8 block3_conv2 False
9 block3_conv3 False
10 block3_pool False
11 block4_conv1 False
12 block4_conv2 False
13 block4_conv3 False
14 block4_pool True
15 block5_conv1 True
16 block5_conv2 True
17 block5_conv3 True
18 block5_pool True
```

Figure 6.2: Checking the trainable layers

Recompiling the model, we will same categorical crossentropy as our loss metric and optimizer as "ADAM" with small learning rate of 0.0001.
Its important to recompile the model after we make any changes to the trainable attribute of any inner layer, so that your changes are taken into account.

```
model.compile(loss = keras.losses.CategoricalCrossentropy(),
              optimizer = keras.optimizers.Adam(learning_rate= 0.0001),
              metrics = ["accuracy"])
```

Figure 6.3: Recompiling the model with smaller learning rate

## 6.1 Re-fiting the model

Now we will train our model for another 5 epochs over the 10 epochs done before. So our total epoch count will come to 15 epochs.

```
Epoch 10/15
2208/2208 [==============================] - 397s 180ms/step - loss: 0.0842 - accuracy: 0.9741 - val_loss: 0.4207 - val_accuracy: 0.9044
Epoch 11/15
2208/2208 [==============================] - 407s 184ms/step - loss: 0.0718 - accuracy: 0.9780 - val_loss: 0.4406 - val_accuracy: 0.8971
Epoch 12/15
2208/2208 [==============================] - 407s 184ms/step - loss: 0.0627 - accuracy: 0.9808 - val_loss: 0.4308 - val_accuracy: 0.9118
Epoch 13/15
2208/2208 [==============================] - 407s 184ms/step - loss: 0.0583 - accuracy: 0.9823 - val_loss: 0.4843 - val_accuracy: 0.9136
Epoch 14/15
2208/2208 [==============================] - 407s 184ms/step - loss: 0.0517 - accuracy: 0.9841 - val_loss: 0.5618 - val_accuracy: 0.9118
Epoch 15/15
2208/2208 [==============================] - 407s 184ms/step - loss: 0.0504 - accuracy: 0.9845 - val_loss: 0.4875 - val_accuracy: 0.9099
```

Figure 6.4: training summary for additional 5 epochs

As we can see our model has generalized better with validation accuracy of 91 percent.

## 6.2 Evaluate on test data

After refitting our model we evaluated our model performance on the unseen data. The data that is still unseen by the model, which is still not fed to the model the test data. We have the test data directory aside to test the performance of our model on data which is kept aside from the beginning. After evaluate our model on test data, we got the astonishing results of 93% accuracy on the test data. Which means our model has learned all the necessary features from the training to make the prediction on the data that is not seen by the model while training. Getting better results on the test data means that our model is ready to be deployed to classify the random birds images. Which we will do later in the project.

```
results_ft = model.evaluate(test_data)
```

```
71/71 [==============================] - 11s 148ms/step - loss: 0.3233 - accuracy: 0.9329
```

Figure 6.5: Evaluation of model performance on Unseen data

## 6.3 Plotting the loss and accuracy curve

As evident form the figure 6.5, which is the plot of training and validation accuracy and loss of both base model and fine tuning model. As we can see our model has generalized better, that means its has learned the important features necessary enough to better predict the class labels of training as well as test set. As we can see form plot below that after fine tuning there is considerable decrease in loss and it went down to 0.48, which is great. Also, Accuracy took the sharp turn upwards with additional epochs and maxed out at around 91 percent and start tumbling there making the squiggly line.

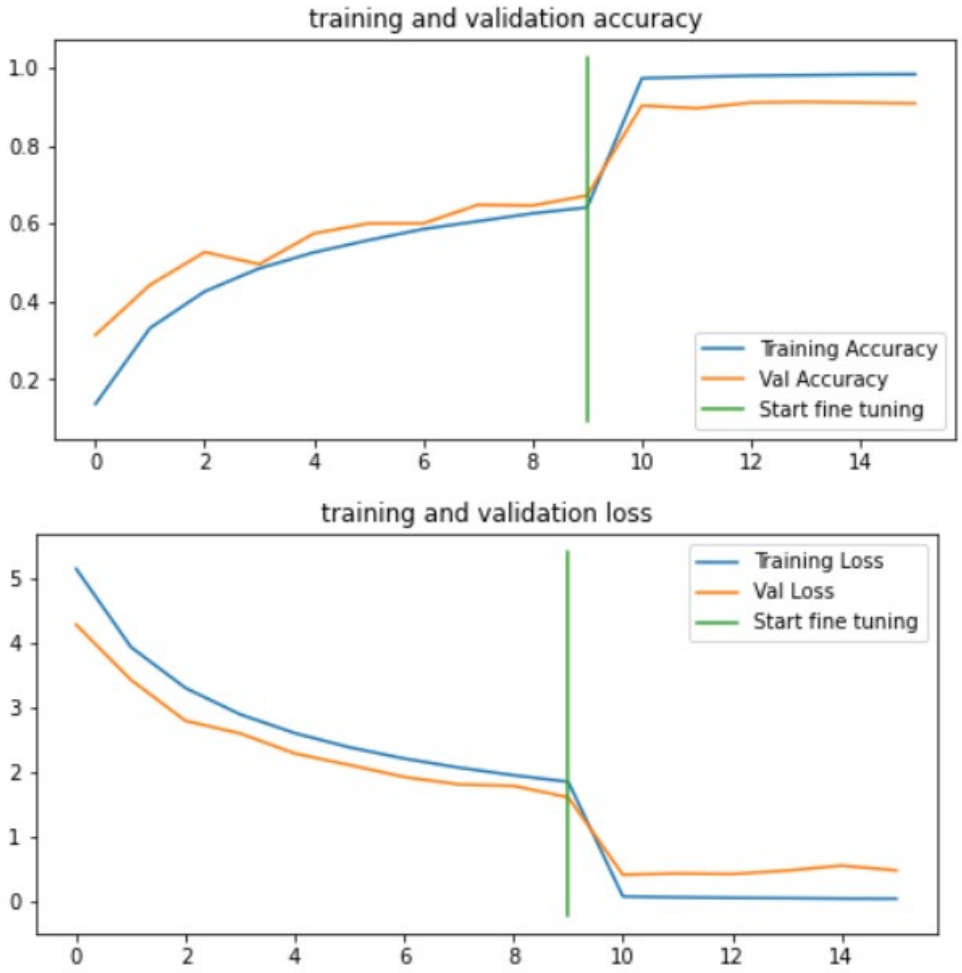



Figure 6.6: plotting the loss over epochs for both base model and training and fine tuning

# Chapter 7

# Predictions

## 7.1 Prediction visualization on Test data

Now is the time to see the prediction of our model and visualize the predictions. Also, we will test our model on some random birds images which are provided in the this dataset to have a look at how well is our model performing.

We made the prediction visualization for 20 random images, all images were correctly predicted by the model to be the correct class of that image.

## 7.2 Random image prediction

Image 2

Image 5


Figure 7.1: Visualization of random birds predictions, where green color label indicates that class was correctly and red color label means the birds was classified to be incorrect class

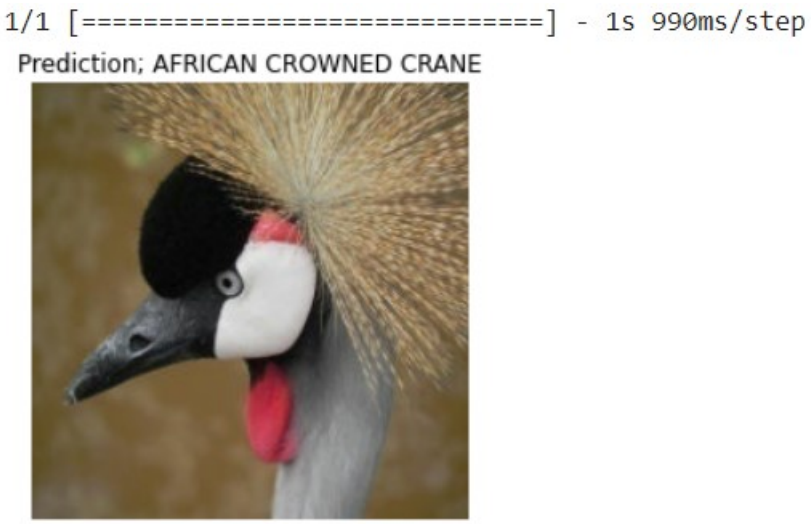



Figure 7.2: Prediction of random image

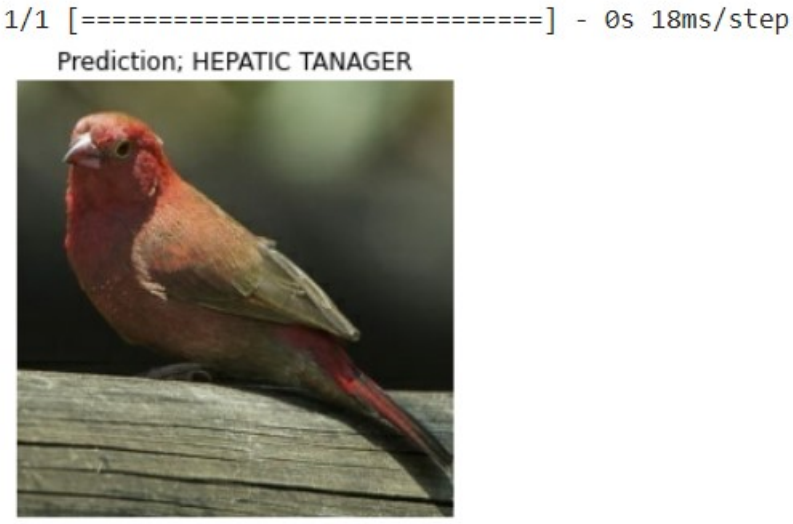



Figure 7.3: Prediction of random image

# Bibliography


[1] Architecture of VGG16. *https://pub.towardsai.net/the-architecture-and-implementation-of-vgg-16-b050e5a5920b*

[2] Birds species paper

*https://link.springer.com/chapter/10.1007/978-981-15-1387-9*$_3$

[3] Keras Api documentation

*https://keras.io/api/data*$_l$*oading/*

[4] tensorflow image classification *https://www.tensorflow.org/tutorials/images/classification*

[5] Image classification backened *https://setosa.io/ev/image-kernels/*

[6] Basic charts in python-plotly *https://plotly.com/python/basic-charts/*

[6] Fine tuning in pre-trained models *https://learnopencv.com/keras-tutorial-fine-tuning-using-pre-trained-models/*