Detection and classification of disease on plant leaf using deep learning

Rohan Sunil Sapkal – 10592020

Supervisor: Professor Anshu Mukherjee

Masters of Science in Business Analytics

Dublin Business School

This dissertation is submitted for the degree of Masters of Science in Business Analytics

September 2022

Declaration

All the content of this dissertation, except for citations to others, is original and has not been submitted in whole or in part for any other degree or qualification at this or any other institution. With the exception of the text and acknowledgments, this dissertation is all my own work, and no other work has been done in collaboration with others.

Rohan Sunil Sapkal – 10592020

September 2022

Acknowledgments

Thanks to my dissertation supervisor, I am grateful for his support Professor Anshu Mukherjee of the Dublin Business School's MSc in Business Analytics degree program. Whenever I had questions or needed help with my research or writing, Professor Anshu Mukherjee was always available to help me. Every time he reviewed my Masters research paper, he ensured that it was my own work and unique. Furthermore, he corrected me when he felt my research artefacts or writing had problems or issues. Whenever there was a question regarding the research, he was always open to discussing it. Since he asked me to explore many options in areas that were very new to me, every meeting with him was a great learning experience. Additionally, I must thank and be grateful to other teaching faculty at Dublin Business School for their assistance in helping me acquire and understand the knowledge, tools and technologies required for success in my dissertation.

Table of Contents

Table of Figures

Abstract

The agricultural industry, climate change, and the economy of the country depend on plants. Therefore, it is extremely important to maintain plants in good condition. In the same way that humans are prone to diseases caused by bacteria, fungus, and viruses, plants are susceptible to them as well. So, it is essential to identify these diseases timely and cure them if you want to prevent the entire plant from being destroyed. There are several research for the Identifying and classifying plant diseases that have been performed using many Machine Learning models. DL (Deep Learning) shows great promise for increased accuracy and is a very promising field of research. The symptoms of plant diseases are detected and classified using a variety of developed and modified DL architectures, along with several visualization techniques. This research aims to implement multiple Deep Learning models on plant disease datasets to find the most effective model for detecting and classifying plants. This research compares Convolution Neural Network (CNN) model layers designed manually, as well as VGG16 and Xception which are Transfer Learning models. Along with a comparison of models, we will deploy one model with high accuracy in the real world.

Keywords:

Deep Learning, Transfer Learning, Convolution Neural Network (CNN), VGG16, RESNET50, Plant Disease Dataset, Data Argumentation.

Introduction

Approximately 1.6% of the earth's population increases every year, and so does the demand for all kinds of plant products. In order to meet the growing demand for quality and quantity of food, the protection of plants from a number of plant diseases is important. In India, more than 35% of annual agriculture production is destroyed every year due to diseases. In addition to the diseases that affect plants, there are a number of other environmental factors that can lead to economic, social, and ecological losses.

There is usually no automated method of detecting plant diseases. Experts such as botanists and agricultural engineers conduct such processes, first by visual analysis and afterward tested in the laboratory. This traditional method is always been time-consuming and involves a great deal of complexity. As a result, machine learning and image processing have become increasingly important for identifying diseases automatically. For users with little or no knowledge of the product they are cultivating, automatic plant disease diagnosis with a visual inspection can be very helpful. Detection of plant diseases has been the subject of numerous studies in the literature. Color, shape, and texture have often been used as features in these studies over the past decade. But we can find some disadvantages to feature classification:

- The performance of this product is extremely low when used exclusively. Several methods have been employed in order to improve the performance of this product.
- Geometric and similar features must be extracted from plants by segmenting them from their roots.
- The method is applied to datasets containing images that are hard to get in real life.

This makes such methods and systems unsuitable for real-world use. Machine learning and visual object recognition have recently been improved by deep convolutional neural networks (CNNs). Feature extraction with improved models and methods can be performed without the use of segmented methods in these areas, which is one of their major advantages. Thus, real-life applications can easily make use of these methods and models.

The aim of the research is to implement the core CNN model with manually designed layers and also implement transfer learning models such as VGG16 and Xception. After implementation comparing the trained model based on their accuracy and other aspects that can help to improve the prediction of plant diseases in the real world. For this study, we have used the pest image dataset that has been collected in the survey paper "PlantDoc: A Dataset for Visual Plant Disease Detection". It contains 27 different classes which are "Apple Scab, Apple Black Rot, Cedar Apple Rust, Apple Healthy, Cherry Healthy, Cherry Powdery Mildew, Grape Black Rot, Grape Black Measles, Grape Healthy, Grape Leaf Blight, Peach Bacterial Spot, Peach Healthy, Potato Early Blight, Potato Healthy, Potato Late Blight, Strawberry Healthy, Strawberry Leaf Scorch, Tomato Bacterial Spot, Tomato Early Blight, Tomato Healthy, Tomato Late Blight, Tomato Leaf Mold, Tomato Septoria Leaf Spot, Tomato Spider Mites Two-spotted, Tomato Target Spot, Tomato Mosaic Virus, Tomato Yellow Leaf Curl Virus".

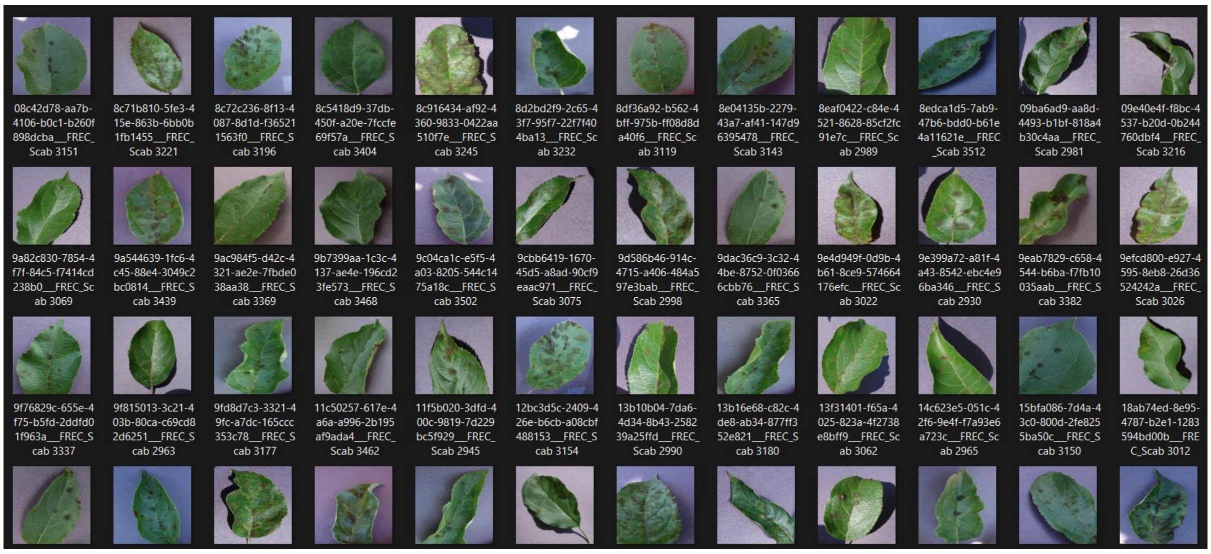

08c42d78-aa7b-4106-b0c1-b260f898dcba__FREC_Scab 3151
8c71b810-5fe3-415e-863b-6bb0b1fb1455__FREC_Scab 3221
8c72c236-8f13-4087-8d1d-f365211563f0__FREC_Scab 3196
8c5418d9-37db-450f-a20e-7fccfe69f57a__FREC_Scab 3404
8c916434-af92-4360-9833-0422aa510f7e__FREC_Scab 3245
8d2bd2f9-2c65-43f7-95f7-22f7f404ba13__FREC_Scab 3232
8df36a92-b562-4bff-975b-ff08d8da40f6__FREC_Scab 3119
8e04135b-2279-43a7-af41-147d96395478__FREC_Scab 3143
8eaf0422-c84e-4521-8628-85cf2fc91e7c__FREC_Scab 2989
8edca1d5-7ab9-47b6-bdd0-b61e4a11621e__FREC_Scab 3512
09ba6ad9-aa8d-4493-b1bf-818a4b30c4aa__FREC_Scab 2981
09e40e4f-f8bc-4537-b20d-0b244760dbf4__FREC_Scab 3216
9a82c830-7854-4f7f-84c5-f7414cd238b0__FREC_Scab 3069
9a544639-1fc6-4c45-88e4-3049c2bc0814__FREC_Scab 3439
9ac984f5-d42c-4321-ae2e-7fbde038aa38__FREC_Scab 3369
9b7399aa-1c3c-4137-ae4e-196cd23fe573__FREC_Scab 3468
9c04ca1c-e5f5-4a03-8205-544c1475a18c__FREC_Scab 3502
9cbb6419-1670-45d5-a8ad-90cf9eaac971__FREC_Scab 3075
9d586b46-914c-4715-a406-484a597e3bab__FREC_Scab 2998
9dac36c9-3c32-44be-8752-0f03666cbb76__FREC_Scab 3365
9e4d949f-0d9b-4b61-8ce9-574664176efc__FREC_Scab 3022
9e399a72-a81f-4a43-8542-ebc4e96ba346__FREC_Scab 2930
9eab7829-c658-4544-b6ba-f7fb10035aab__FREC_Scab 3382
9efcd800-e927-4595-8eb8-26d36524242a__FREC_Scab 3026
9f76829c-655e-4f75-b5fd-2ddfd01f963a__FREC_Scab 3337
9f815013-3c21-403b-80ca-c69cd82d6251__FREC_Scab 2963
9fd8d7c3-3321-49fc-a7dc-165ccc353c78__FREC_Scab 3177
11c50257-617e-4a6a-a996-2b195af9ada4__FREC_Scab 3462
11f5b020-3dfd-400c-9819-7d229bc5f929__FREC_Scab 2945
12bc3d5c-2409-426e-b6cb-a08cbf488153__FREC_Scab 3154
13b10b04-7da6-4d34-8b43-258239a25ffd__FREC_Scab 2990
13b16e68-c82c-4de8-ab34-877ff352e821__FREC_Scab 3180
13f31401-f65a-4025-823a-4f2738e8bff9__FREC_Scab 3062
14c623e5-051c-42f6-9e4f-f7a93e6a723c__FREC_Scab 2965
15bfa086-7d4a-43c0-800d-2fe8255ba50c__FREC_Scab 3150
18ab74ed-8e95-4787-b2e1-1283594bd00b__FREC_Scab 3012

Figure 1: Images of the dataset

Literature Review

Many studies have been conducted using machine learning and artificial intelligence techniques to identify plant diseases over the past decades, but each method has proved to be successful and provides different results. It has been implemented and improved continuously to achieve the desired results, including CNN, DBN, Deep Learning, and OpenCV Network. Each of these algorithms and models can be adapted for use on different datasets, resulting in different results and performances.

With the help of "PlantDoc: A Dataset for Visual Plant Disease Detection" published by Davinder Singh, Naman Jain, Pranjali Jain, Pratik Kayal, Sudhakar Kumawat, Nipun Batra we have found the dataset of plant diseases. The research is all about collecting the data for number of plant disease. There are total 2,598 images of plant disease across 13 different species and 27 classes in dataset. In this study, they used state-of-the-art object detection techniques to detect diseases in leaves in images. By applying image segmentation techniques, they were able to extract leaf information from the images, which could potentially enhance the dataset's utility which is also used in our research.

"Plant Disease Detection and Classification Using Deep Neural Networks" published by Aravindhan Venkataramanan, Deepak Kumar P Honakeri, Deepak Kumar P Honakeri. The paper presents the classification is conducted in more than one stage to eliminate possibilities at each stage, therefore allowing better accuracy when predicting the outcome. Leafs are extracted from input images using the YOLOv3 object detector. A series of ResNet18 models are used to analyze the extracted leaf which is a transfer learning model. Leaf types are identified on the first layer, and diseases associated with the plant are detected on the second layer. We have used ResNet reference from this research.

"Plant Disease Detection using Deep Learning" published by Murk Chohan, Adil Khan, Rozina Chohan, Saif Hassan Katpar, Muhammad Saleem Mahar. The information on the data augmentation and CNN model is been provided in this research which is also used while working on the dataset. In this research, they have done different experiments. Approximately 15 percent of data from PlantVillage is used for testing purposes, which includes both healthy and diseased plant images. There proposed model has achieved 98% of accuracy. We have referred to the work on the CCN model from this research.

"Comparison of VGG and ResNet used as Encoders for Image Captioning" published by Viktar Atliha and Dmitrij Seˇ sok. The researchers analyze both

VGG and ResNet convolution networks as encoders as part of an image captioning model to see which one produces the best captions. Based on the results of the study, ResNet outperformed VGG in image captioning, achieving higher BLEU-4 scores and also With less training epochs, ResNet is able to achieve comparable results to VGG-based models. From this research, we have referred to the pros and cons of the transfer learning model.

Methodology

DM-CRISP

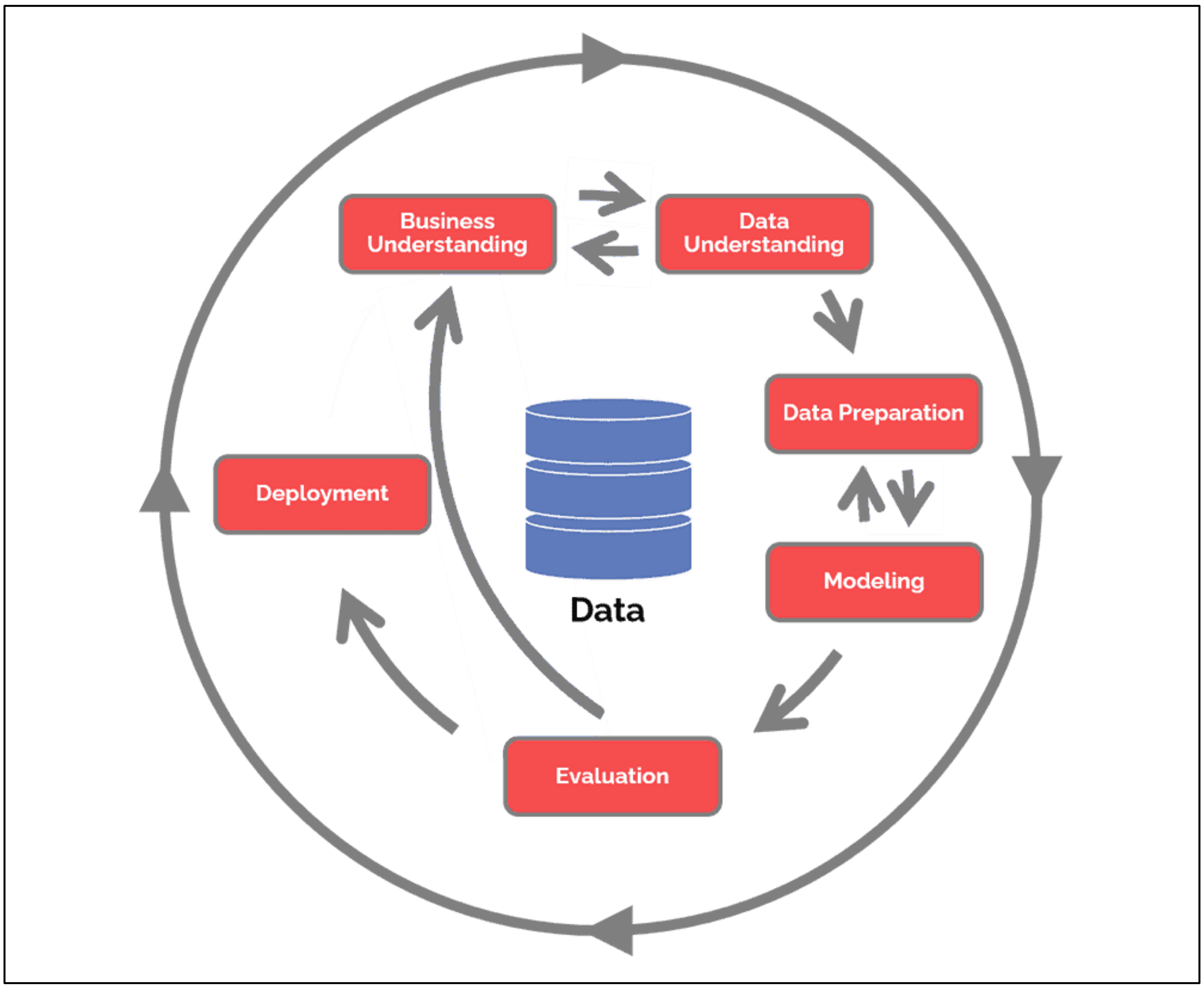


Figure 2: DM-CRISP Workflow

This process model serves as the foundation for a data science workflow. It is called CRISP-DM (CROSS Industry Standard Process for Data Mining). Phases are sequentially arranged as follows:

1. Business understanding
   Understanding the project's objectives and requirements is the purpose of the Business Understanding phase. This phase also includes three other tasks that are general in most projects, excluding the third:
   - Determine business objectives: In order to define business success criteria, you should first "understand what the customer is trying to accomplish from a business perspective." (CRISP-DM Guide).

- Assess situation: An assessment of risks and contingencies, as well as a cost-benefit analysis, must be conducted in order to determine resource availability and project requirements.
- Determine data mining goals: It is important to define success in terms of technical data mining, as well as business objectives.
- Produce project plan: Each phase of the project should be defined with detailed plans based on selected technologies and tools.

2. Data understanding:
   The next step is to understand the data. The focus is on identifying, collecting, and analyzing the data sets required to accomplish the project goals. This adds to the foundation of Business Understanding. There are also four tasks in this phase:
   - Collect initial data: Load the data into your analysis tool after obtaining the necessary data.
   - Describe data: The surface properties of the data, such as their format, number, and field names, should be documented.
   - Explore data: Analyze the data in more detail. Identify relationships among the data by querying it, visualizing it, and analyzing it.
   - Verify data quality: Issues with quality should be documented.
3. Data preparation:
   Data munging, also known as "data preparation", prepares the final dataset for modeling. There are five tasks to be completed:
   - Select data: Document reasons for including or excluding certain data sets.
   - Clean data: The longest task is often this one. A garbage-in, garbage-out system is likely to result without it. During this task, erroneous values are often corrected, imputed, or removed.
   - Construct data: The purpose of this is to derive useful attributes. Based on someone's height and weight, calculate their body mass index.
   - Integrate data: Combining data from multiple sources creates new data sets.
   - Format data: Data should be reformatted as necessary. If you wish to perform mathematical operations on string values containing numbers, for example, you might convert them to numeric values.
4. Modelling:
   Several modelling techniques will likely be used to build and assess various models here. During this phase, there are four tasks to complete:

- Select modelling techniques: Make a decision as to which algorithm to use (for instance, regression or neural networks).
- Generate test design: The data might have to be split into training, testing, and validation sets depending on your modelling approach.
- Build model: Even though this sounds glamorous, it can be as simple as executing "reg = LinearRegression().fit(X, y)".
- Assess model: Data scientists interpret the model results based on domain knowledge, predefined success criteria, and test design in the context of multiple competing models.

Even though CRISP-DM recommends iterating model building and assessment until a team is certain that they have identified the best model(s), teams should iterate until they discover a "good enough" model, go through the CRISP-DM lifecycle, and then further refine it in the future.

5. Evaluation:
   A number of different aspects of the model are considered in the Evaluation phase, During the Modeling phase, the Assess Model task assesses the technical model. Three tasks are involved in this phase:
   - Evaluate results: How well do the models meet the criteria for business success? Should we approve one or more of them for the business?
   - Review process: Take a look at what has been accomplished. Correct any errors found and summarize findings.
   - Determine next steps: Make a decision regarding deployment, iteration, or initiating new projects based on the previous three tasks.
6. Deployment:
   In the absence of access to the results of a model, the model is not particularly useful. There is a wide range of complexity in this phase. In this final phase, four tasks must be completed:
   - Plan deployment: A deployment plan should be developed and documented.
   - Plan monitoring and maintenance: An operational phase (or post-project phase) control and maintenance plan should be developed in order to avoid issues.
   - Produce final report: Data mining results may be presented in a final summary by the project team.
   - Review project: Analyze the success of the project, the areas of improvement, and what could have been done better.

Implementation of work flow using DM-CRISP

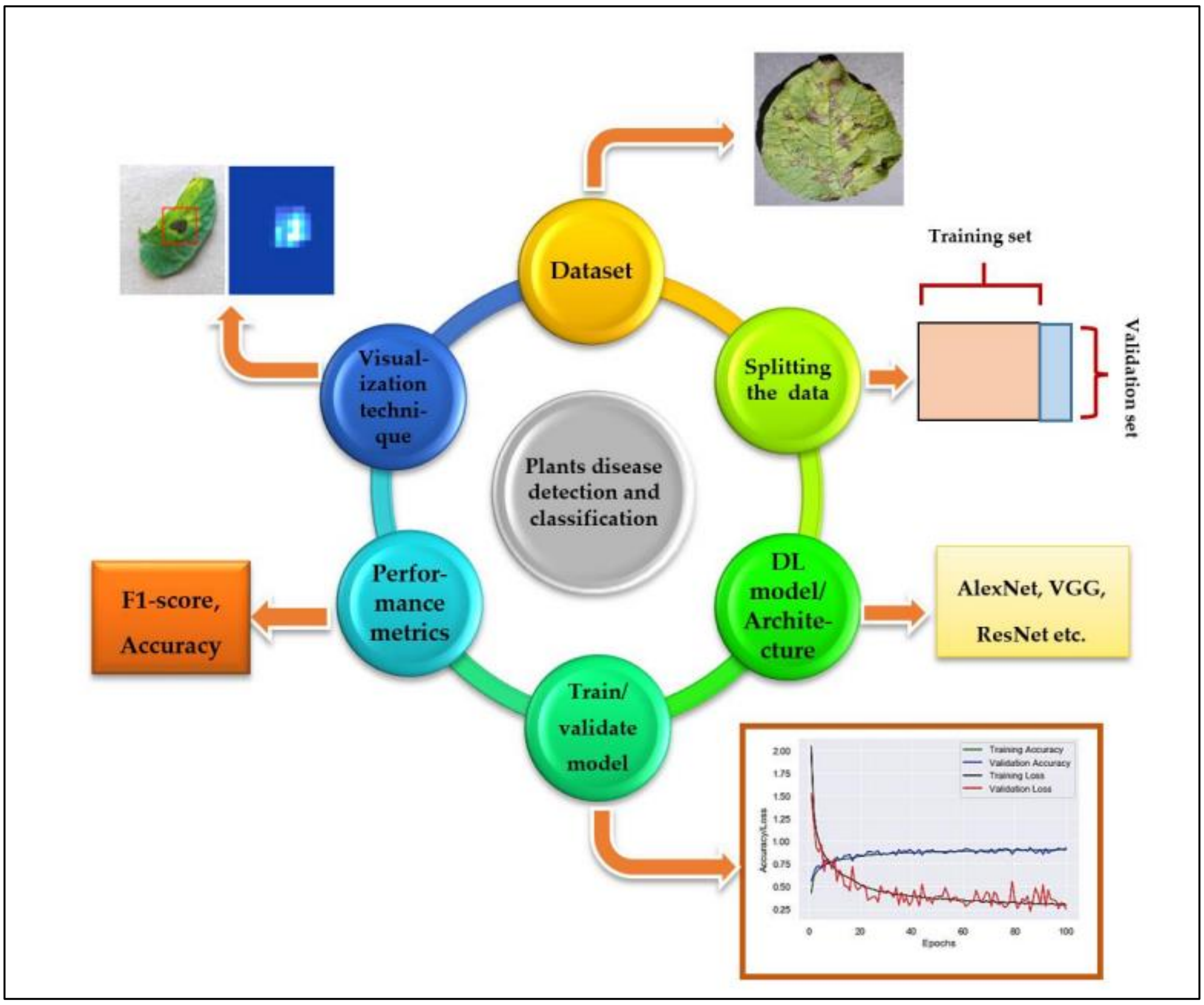


Figure 3: Workflow of system

Diagram of DL implementation: The dataset is first collected, then divided into training and validation sets, approximately 80% each. In order to determine the significance of DL models, training/validation plots are obtained for DL models evaluated from scratch or by using transfer learning techniques. Detection/localization/classification of images are performed using performance metrics, followed by visual mapping and visualization techniques.

Data Pre-processing

To prevent memory crashes, large sizes and colors of the images, and other potential issues, the acquired data must be pre-processed. Several different areas can benefit from design recognition and picture pre-processing. The rapid development of this discipline is a result of this factor. A variety of requirements created by resolving relevant problems rouse this discipline and accelerate its growth.

Reshaping Data

In order to use the acquired data for training, the data must be converted into a 3D tensor. In order to accomplish this, we will use the Keras API and TensorFlow package, both of which have been developed for years for such purposes and are freely available online. There are several parameters in a tensor, including: Grayscale images are indicated by the channel. Each step involves a neighborhood picture test on every picture in the preparation set and a decent sized window (e.g., 55) is ventured over the whole picture. The window moves four pixels at each progression (Khanzada et al. 2020).

Data Augmentation

A data augmentation technique generates new data points from existing data in order to artificially increase data volume. Creating new data points by using deep learning models or making small changes to data can be included. There has been a rapid increase in deep learning applications especially in the machine learning domain. Artificial intelligence faces challenges that can be overcome with the use of data augmentation techniques.

As new and different examples are formed to train datasets, data augmentation can enhance machine learning models' performance and outcomes. Machine learning models perform better and are more accurate if their datasets are rich and sufficient.

For machine learning models, collecting and labeling data can be exhausting and expensive. It is possible to reduce these operational costs by transforming datasets and utilizing data augmentation techniques.

A common method of augmented data is to alter visual data in a simple way. Data augmentation activities include:

- padding
- random rotating
- re-scaling,
- vertical and horizontal flipping
- translation (image is moved along X, Y direction)
- cropping
- zooming
- darkening & brightening/color modification
- grayscaling
- changing contrast
- adding noise
- random erasing

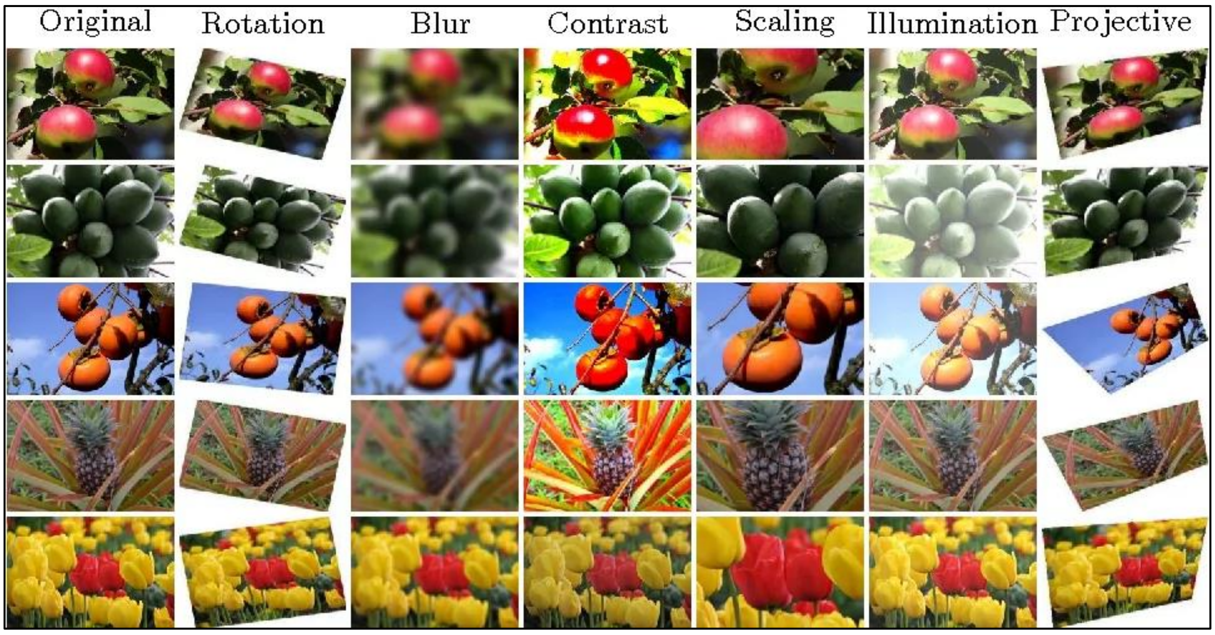


Figure 4: Examples of data augmentation techniques

As a result of data augmentation, the following benefits can be realized:

- Improvements can be made to the accuracy of model predictions.
    - The models should be trained with more data.
    - For better models, prevent data scarcity.
    - A function is overfitted if it correlates too closely with a limited number of observations (error in statistics).
    - The models will be able to generalize more.

    - Resolving issues related to classification imbalances.
- It is possible to reduce the costs associated with data collection and labeling.
- Detects rare events and predicts them.
- Makes sure data privacy is protected.

CNN [Convolutional Neural Networks]

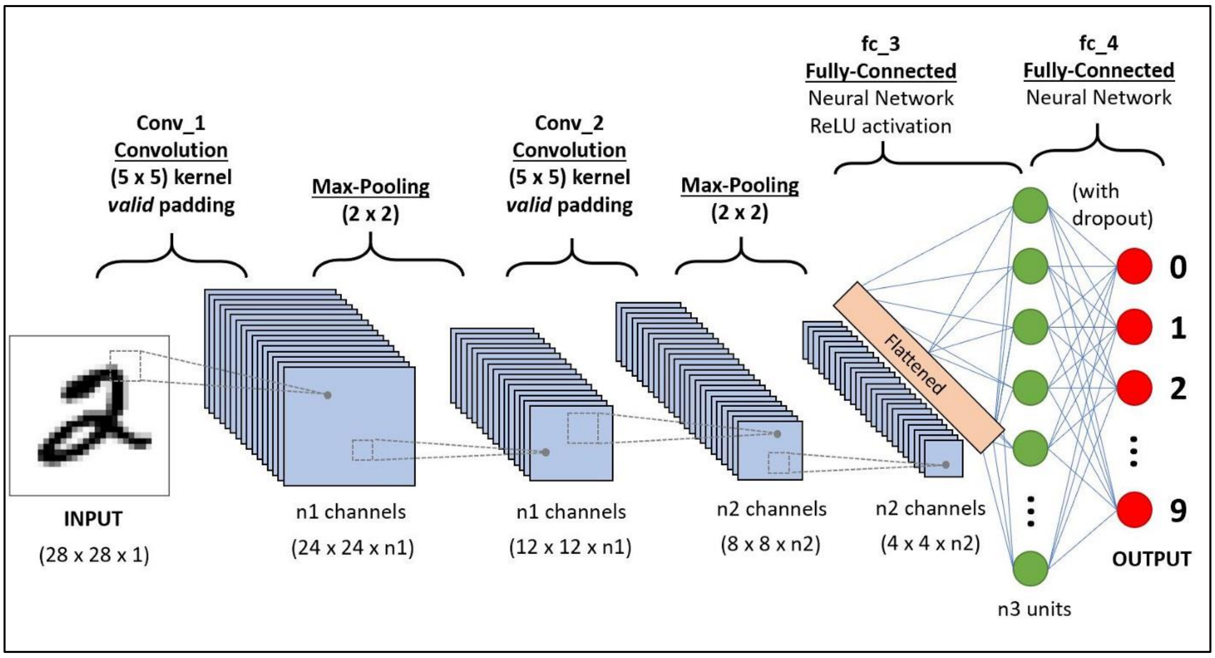


Figure 5: CNN Architecture

Convolutional Neural Networks (ConvNet/CNNs) are Deep Learning algorithms that can identify and distinguish various aspects in an image based on their importance (learnable weights and biases). Compared to other classification algorithms, ConvNet requires much less pre-processing. Thread filters are hand-engineered in primitive methods, but ConvNets can learn these filters with enough training.

Designed from the organization of the visual cortex to represent the connectivity pattern of neurons in the brain, the architecture of a ConvNet is analogous to that of neurons in the brain. Receptive Fields are restricted regions within the visual field where individual neurons respond to stimuli. The visual area is covered by a collection of such fields.

What makes ConvNets superior to FeedForward Neural Nets?

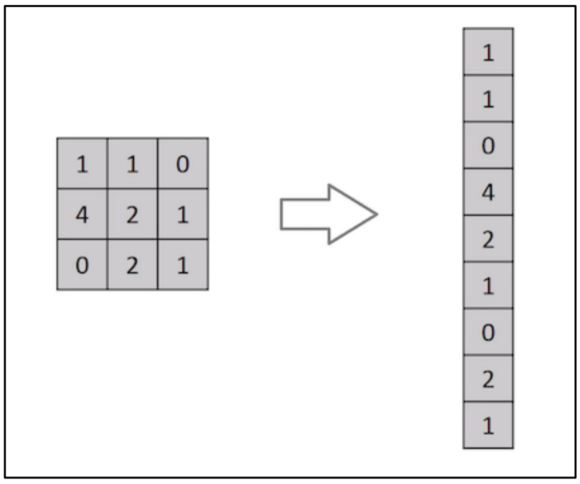


Figure 6: An image matrix of 3x3 is transformed into a vector matrix of 9x1

For extremely simple binary images, the method might perform class prediction reasonably well, but for complex images with pixel dependencies, it would have little to no accuracy. With the use of relevant filters, a ConvNet is capable of extracting spatial and temporal relationships from an image. As a result of reducing the number of parameters involved and reusing weights, the architecture performs better on the image dataset.

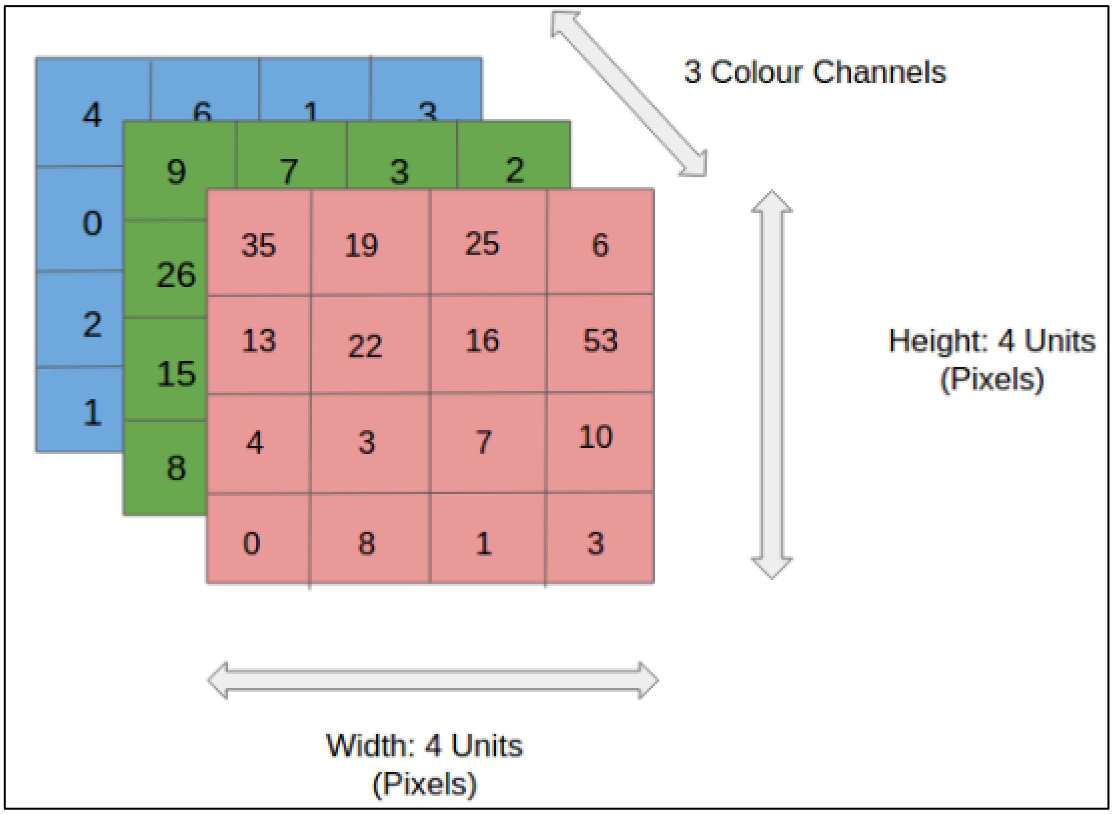


Figure 7: 4x4x3 RGB Image

Red, Green, and Blue are the three color planes of the RGB image in the figure. Among the color spaces that can be used for images are Grayscale, RGB, HSV, CMYK, etc. When the images reach 7680x4320 in resolution, you can imagine how computationally intensive things become. With ConvNet, images are reduced into a format that can be more easily processed, without losing important features that are critical to creating good predictions. When we are designing a scheme for learning features and scalability to large datasets, this is crucial.

Convolution Layer — The Kernel

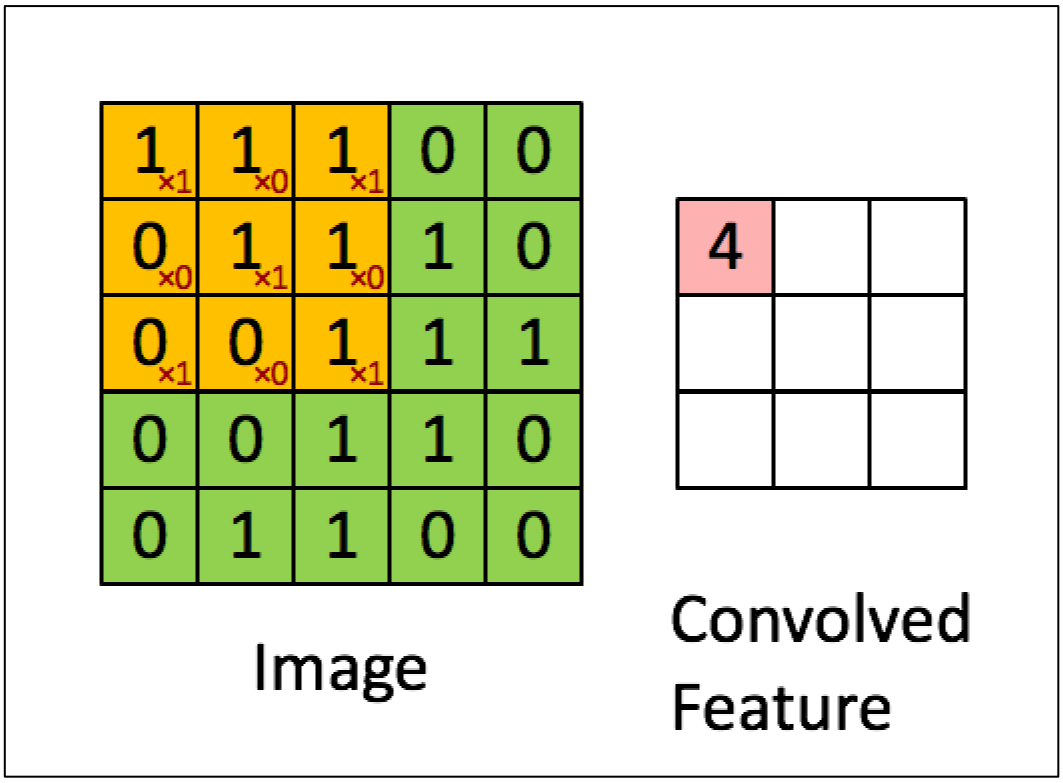


Figure 8: Getting a 3x3x1 convolved feature from a 5x5x1 image

Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB)

Our 5x5x1 input image appears in the green section of the above demonstration. Convolution is carried out by the Kernel/Filter, K, that is represented in yellow in the first part of a Convolutional Layer. K is a matrix of 3x3x1 dimensions.

As a result of the kernel shifting nine times, a matrix multiplication is performed between K and the portion P of the image that the kernel hovers over, every time.

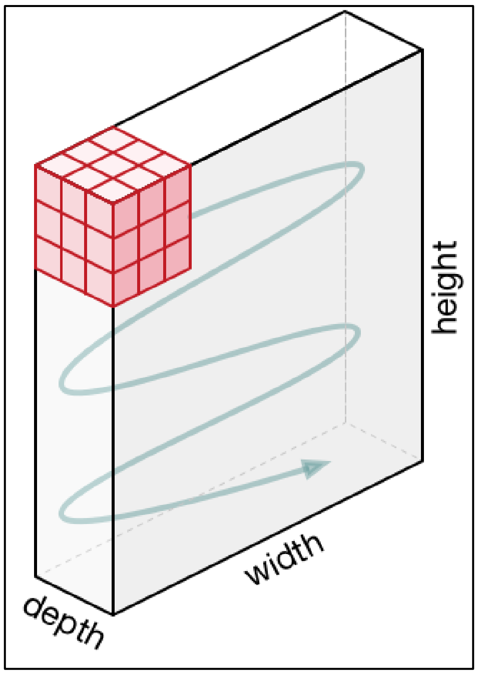
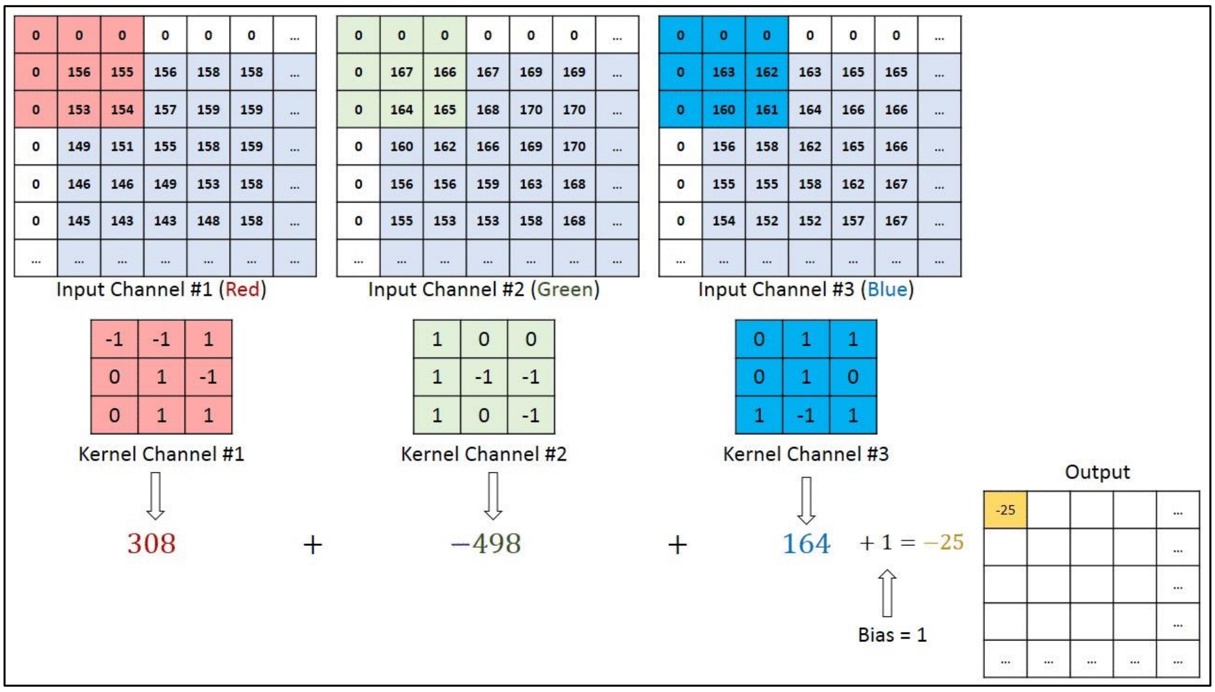


Figure 9: Kernel movement

When a certain Stride Value is set, the filter moves to the right until all of the width has been parsed. Continuing, it traverses the entire image starting at the left and repeating the process until the entire image is traversed.



Input Channel #1 (Red)

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Kernel Channel #1

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #2

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #3

0	1	1
0	1	0
1	-1	1

Output

-25				...
				...
				...
				...
...	...	...	...	...

Figure 10: A 3x3x3 Kernel is applied to a MxNx3 matrix to achieve convolution

The kernel has the same depth as the input image when it is applied to images with multiple channels (e.g. RGB). To give us a squashed one-depth channel

Convoluted Feature Output, Kn and In stacks are multiplied ([K1, I1], [K2, I2], [K3, I3]). Everything is combined with the bias.

A Convolution Operation extracts high-level features from an input image, such as edges. It is not necessary to limit ConvNets to one layer of convolution. According to convention, the first ConvLayer captures low-level features such as edges, colors, gradient orientations, and so on. As layers are added, the architecture adjusts to the High-Level features as well, so we have a network that is able to understand images based on the wholesome understanding that we would obtain using a neural network.

A reduction in dimensionality in the convolved feature as compared to the input is one result, whereas an increase or a decrease in dimensionality is another. When it comes to the former, you apply Valid Padding, and when it comes to the latter, you apply Same Padding.

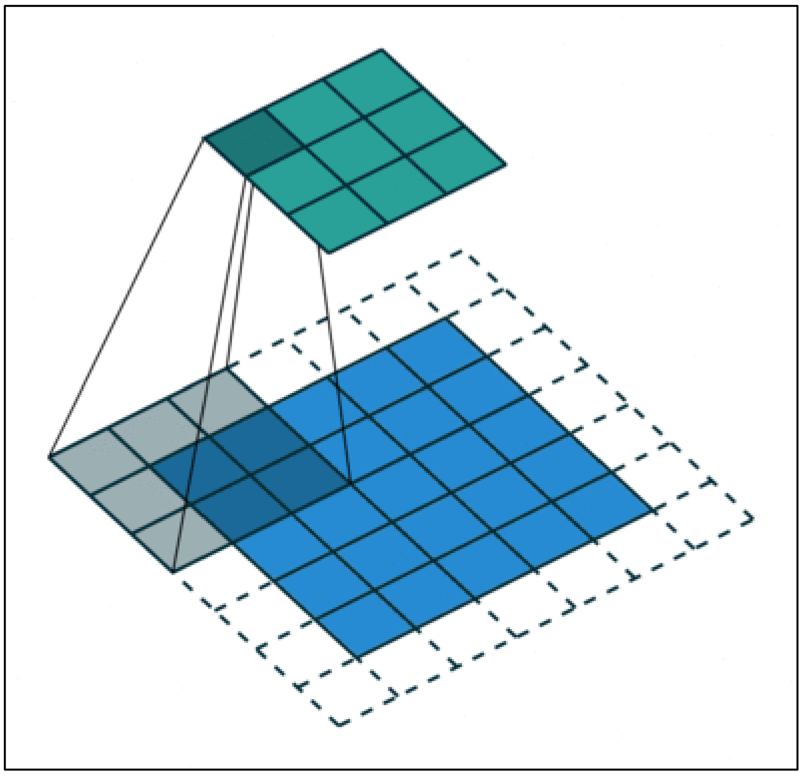


Figure 11: Padding

Adding the 3x3x1 kernel to the 5x5x1 image results in a 5x5x1 matrix when it is augmented to 6x6x1. As a result, the padding is called Same Padding.

The Valid Padding matrix, on the other hand, has dimensions of the Kernel (3x3x1) itself if the same operation is conducted without padding.

Pooling Layer

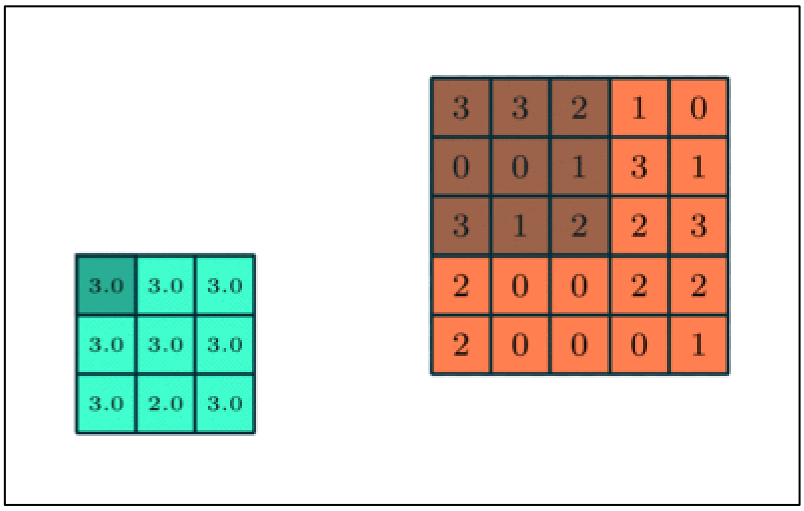


Figure 12: 5x5 convolved feature over 3x3 pooling

Pooling reduces the spatial size of the convolutional feature similarly to the Convolutional layer. Through dimensionality reduction, the data can be processed with less computing power. Further, it preserves the process of effectively training the model by extracting rotationally and positionally invariant features.

Max Pooling and Average Pooling are two types of pooling. The maximum value of the image covered by the kernel is returned by max pooling. The Average Pooling method, however, returns the average of all values within the kernel area.

Noise suppression is also achieved through max pooling. Noise activations are completely discarded and de-noising and dimension reduction are performed. The Average Pooling method allows noise to be suppressed by reducing dimensionality. Then we can say that max pooling outperforms average pooling by a wide margin.

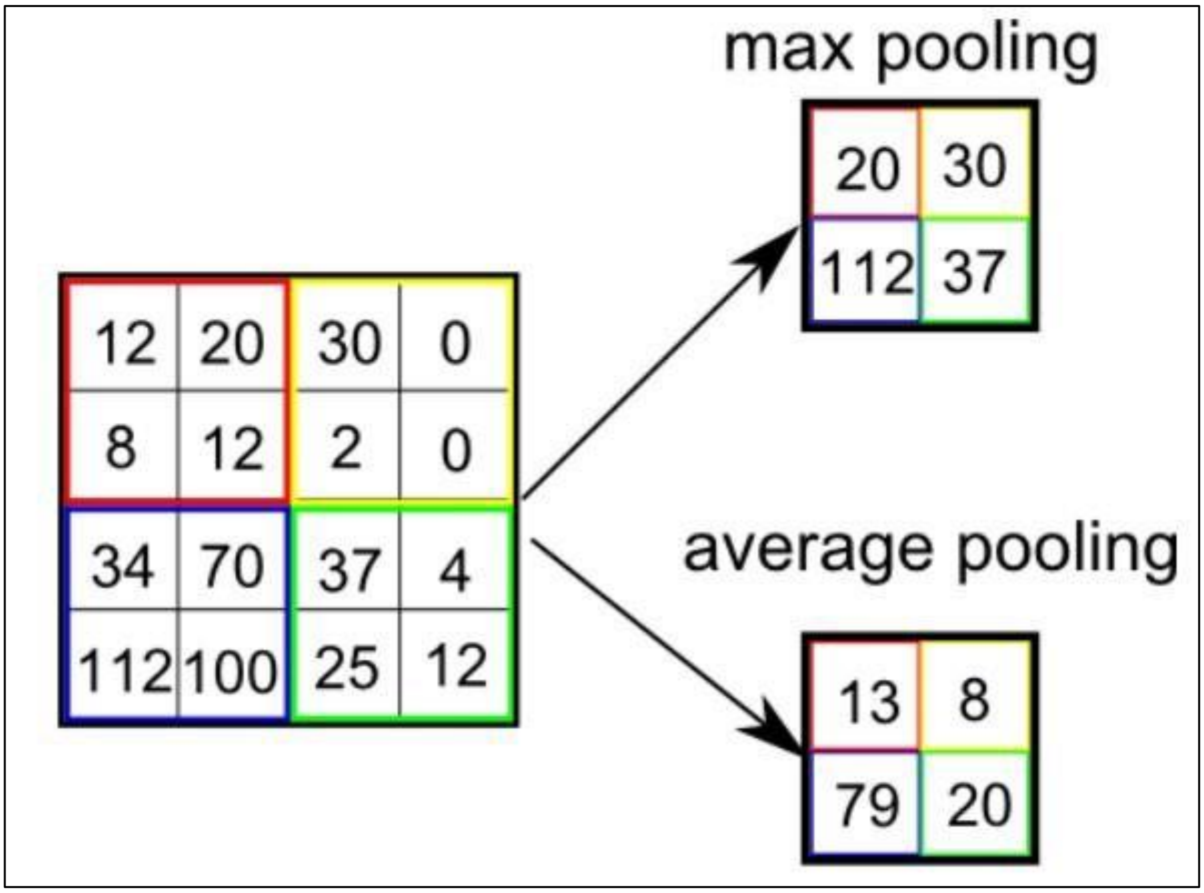
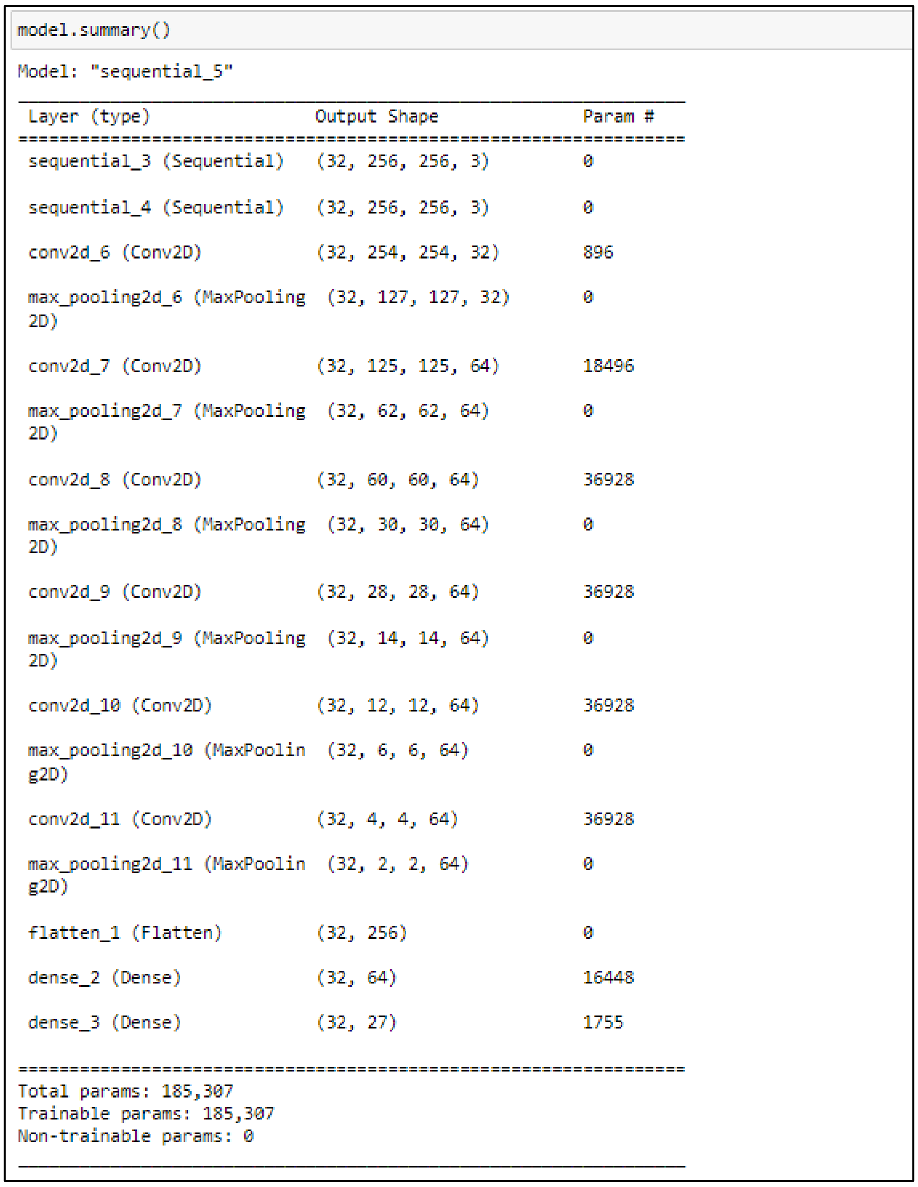


Figure 13: Types of Pooling

A Convolutional Neural Network has two layers, the Convolutional Layer and the Pooling Layer. In some cases, more layers may be added depending on the complexity of the images, but this will increase computational costs. Our model has been successfully enabled to understand the features after we went through the above process. The final output will then be flattened and fed into a standard neural network for classification.

We have designed the CNN layer manually as per below image:



```
model.summary()

Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_3 (Sequential)   (32, 256, 256, 3)         0

 sequential_4 (Sequential)   (32, 256, 256, 3)         0

 conv2d_6 (Conv2D)           (32, 254, 254, 32)        896

 max_pooling2d_6 (MaxPooling  (32, 127, 127, 32)       0
 2D)

 conv2d_7 (Conv2D)           (32, 125, 125, 64)        18496

 max_pooling2d_7 (MaxPooling  (32, 62, 62, 64)         0
 2D)

 conv2d_8 (Conv2D)           (32, 60, 60, 64)          36928

 max_pooling2d_8 (MaxPooling  (32, 30, 30, 64)         0
 2D)

 conv2d_9 (Conv2D)           (32, 28, 28, 64)          36928

 max_pooling2d_9 (MaxPooling  (32, 14, 14, 64)         0
 2D)

 conv2d_10 (Conv2D)          (32, 12, 12, 64)          36928

 max_pooling2d_10 (MaxPoolin  (32, 6, 6, 64)           0
 g2D)

 conv2d_11 (Conv2D)          (32, 4, 4, 64)            36928

 max_pooling2d_11 (MaxPoolin  (32, 2, 2, 64)           0
 g2D)

 flatten_1 (Flatten)         (32, 256)                 0

 dense_2 (Dense)             (32, 64)                  16448

 dense_3 (Dense)             (32, 27)                  1755

=================================================================
Total params: 185,307
Trainable params: 185,307
Non-trainable params: 0
_________________________________________________________________
```

Figure 14: Design layers of CNN

VGG16

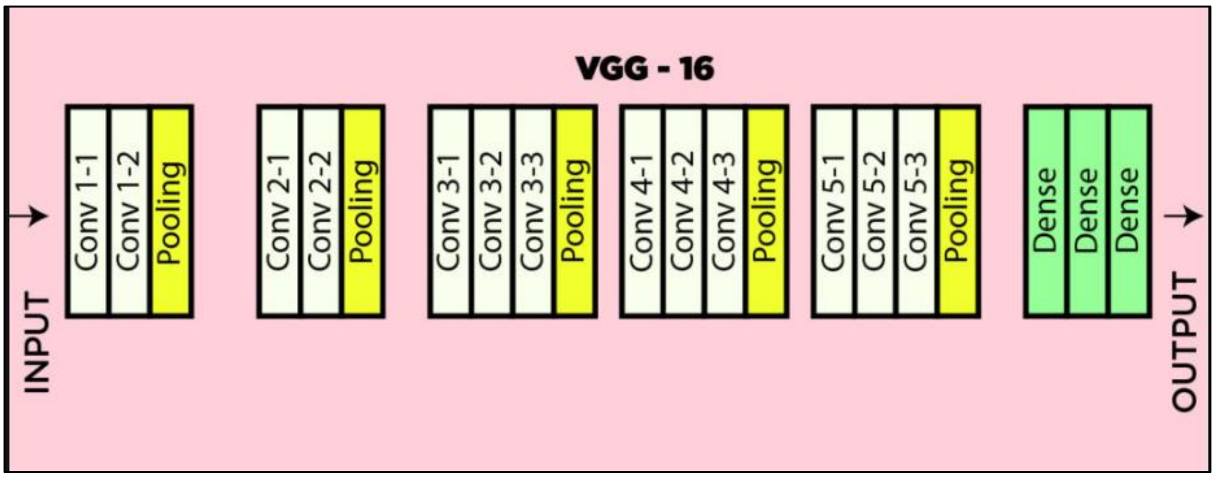
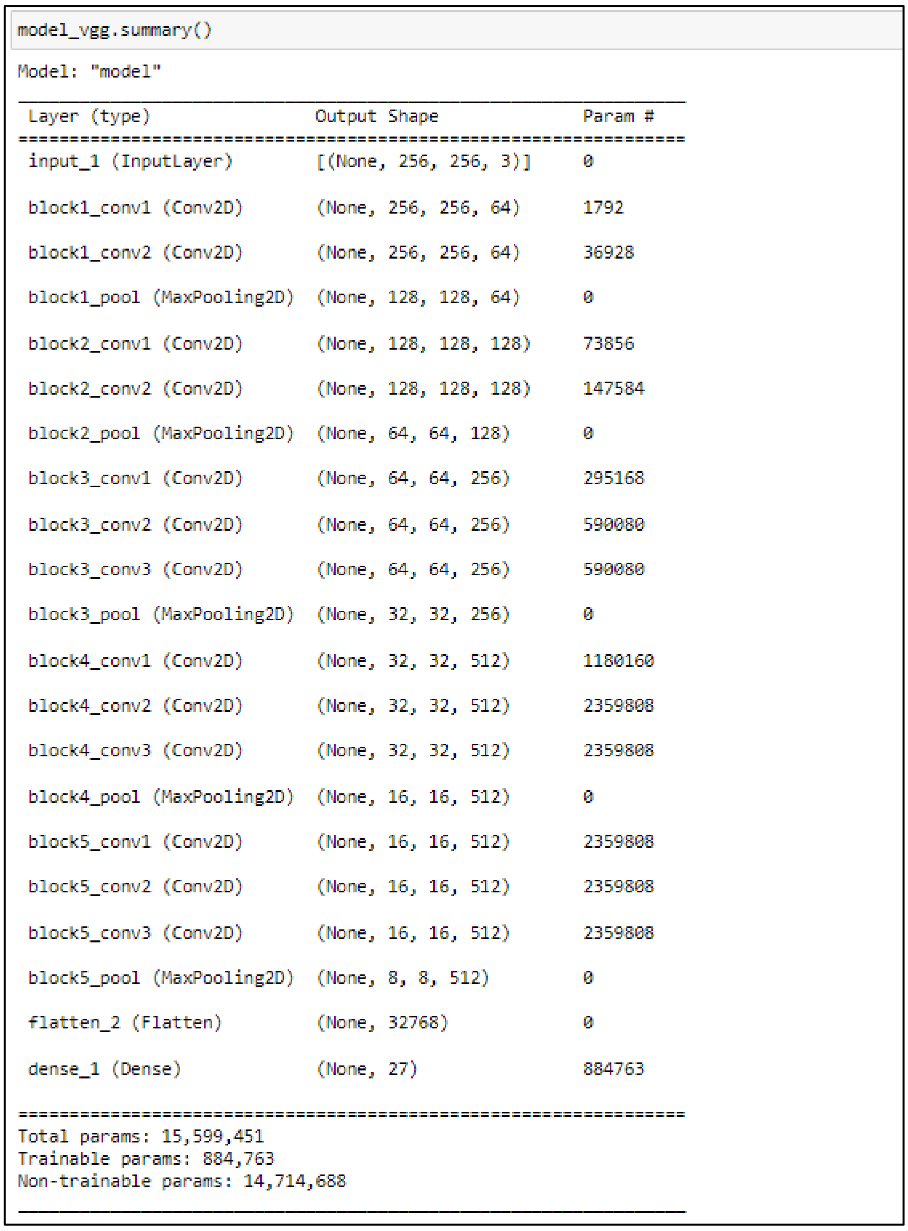


Figure 15: Architecture of VGG16

(224, 224, 3) is the dimension of the input image to the network. It consists of 64 channels and the same padding on the first two layers. Following the max pool layer, there are two convolution layers, each with a filter size of 128 (layer 2, layer 3) and a filter size of 512 (layer 3). As with the previous layer, this one follows with a max-pooling stride layer (2, 2). In addition, there are two convolution layers with filters of size 3 and 3 and 256 filters in total. A maximum pool layer follows, followed by two sets of three convolution layers. Filters with (3, 3) sizes and the same padding are included in each. A stack of two convolution layers is then applied to this image. The filters used in these layers of convolution and max-pooling are 3 x 3 rather than 11 x 11 in AlexNet and 7 x 7 in ZF-Net. A layer may also use 1x1 pixels if it has to manipulate the number of input channels. To prevent the spatial property of the image from being affected by the convolution layer, a pixel padding is applied to each convolution layer.

In the end, we got a feature map that is (7, 7, 512) after stacking convolution and max-pooling layers. Feature vectors (1, 25088) are created from this output. Three fully connected layers follow this process, the first taking input from the last feature vector and producing a (1, 4096) vector, the second also producing a (1, 4096) vector, while the third layer produces 1000 channels for 1000 classes of the ILSVRC challenge. In order to classify 1000 classes, the third fully connected layer implements the softmax function. ReLU is used as the activation function for all hidden layers. It reduces the likelihood of vanishing gradients, which makes it more computationally efficient.

VGG16 has a pre-define layer as per the below image:



```
model_vgg.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0         
                                                                 
 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0         
                                                                 
 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0         
                                                                 
 flatten_2 (Flatten)         (None, 32768)             0         
                                                                 
 dense_1 (Dense)             (None, 27)                884763    
                                                                 
=================================================================
Total params: 15,599,451
Trainable params: 884,763
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 16: Layers of VGG16

Xception

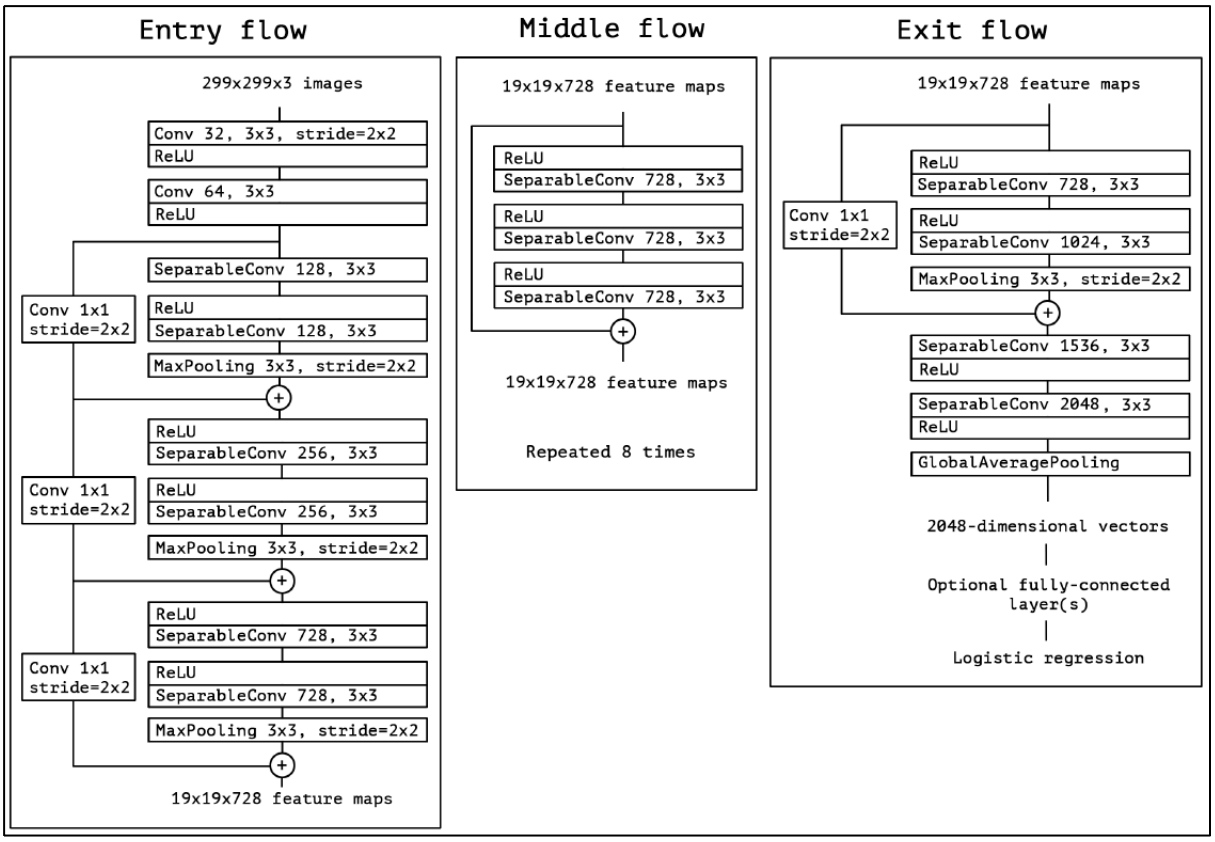


Figure 17: Flow of Xception

A modified depthwise separable convolution is shown in the figure above. Inclusion Modules such as SeparableConvs are placed throughout deep learning's architecture as Inception Modules.

A data stream is first entered, then repeated eight times in the middle flow, and then exited. A batch normalization is performed on all layers of Convolution and SeparableConvolution. A majority of classical classification challenges have been outperformed by the Xception architecture, as compared to the VGG-16, ResNet, and Inception V3 architectures.

What is the working principle of Xception?

There are two main points that make XCeption an efficient architecture:

- Convolution by depth-wise separation.
- Convolution block shortcuts similar to ResNet.

Convolutions that are depthwise separated are much faster and more efficient than classical convolutions.

Implementation of the Xception:

As with ResNet implementations, XCeption uses Depthwise Separable Convolution blocks along with Maxpooling.

The particularity of XCeption is that it reverses the order of the depthwise and pointwise convolutions, as shown here

Inception is a CNN model that has been inspired by the Extreme Inception or Xception model, an enhancement of the original model that is described as being "extreme". Inception's model consists of parallel layers of deep convolutions and wider convolutions. Three convolutional layers are present in each of the two levels of this model. A single layer is present in only one level of the Xception model, unlike the inception model. Upon passing the output to the next filter, it is sliced into three segments. At the first level, there is only a single convolutional filter level (1 x 1), while at the second level, there are three convolutional filters (3 x 3). Depthwise separable convolution is the defining characteristic of the Xception model. Convolutions are done depthwise and pointwise in the Xception model, but the deep CNN model takes care of spatial and channel distribution.

As shown in Figure, the Xception network model has a modular architecture. Based on an extreme interpretation of the Inception model from Google, the Xception model is a 71-layer deep CNN. Layers of depth-separable convolution are stacked in its architecture.

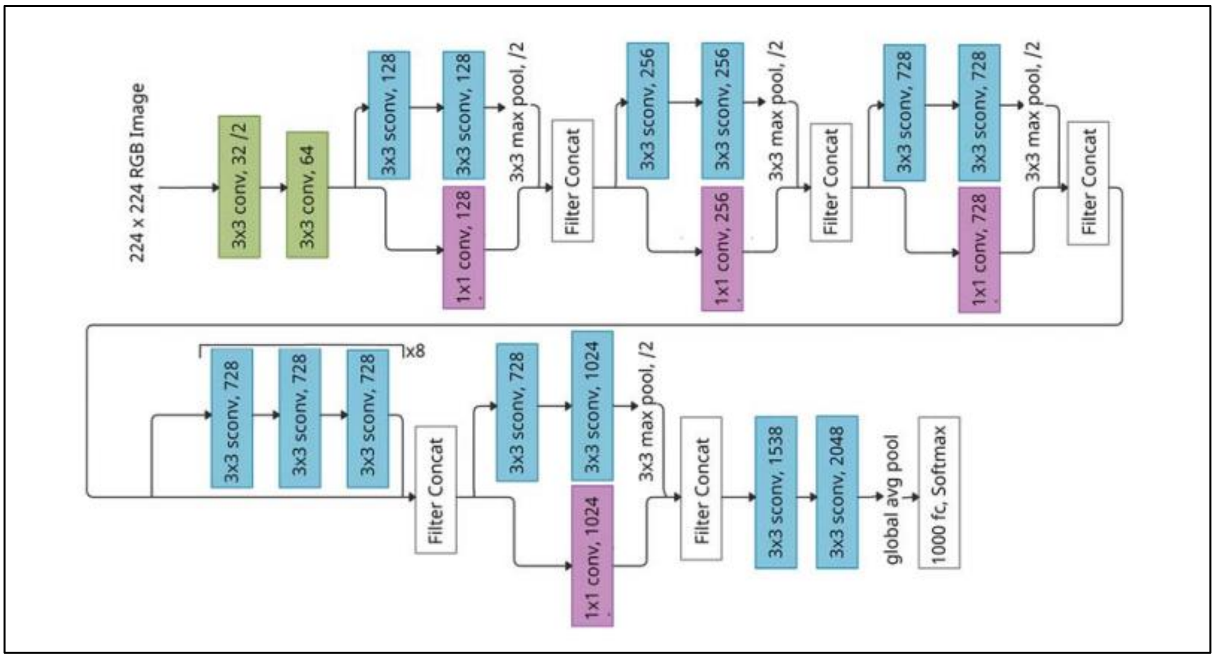
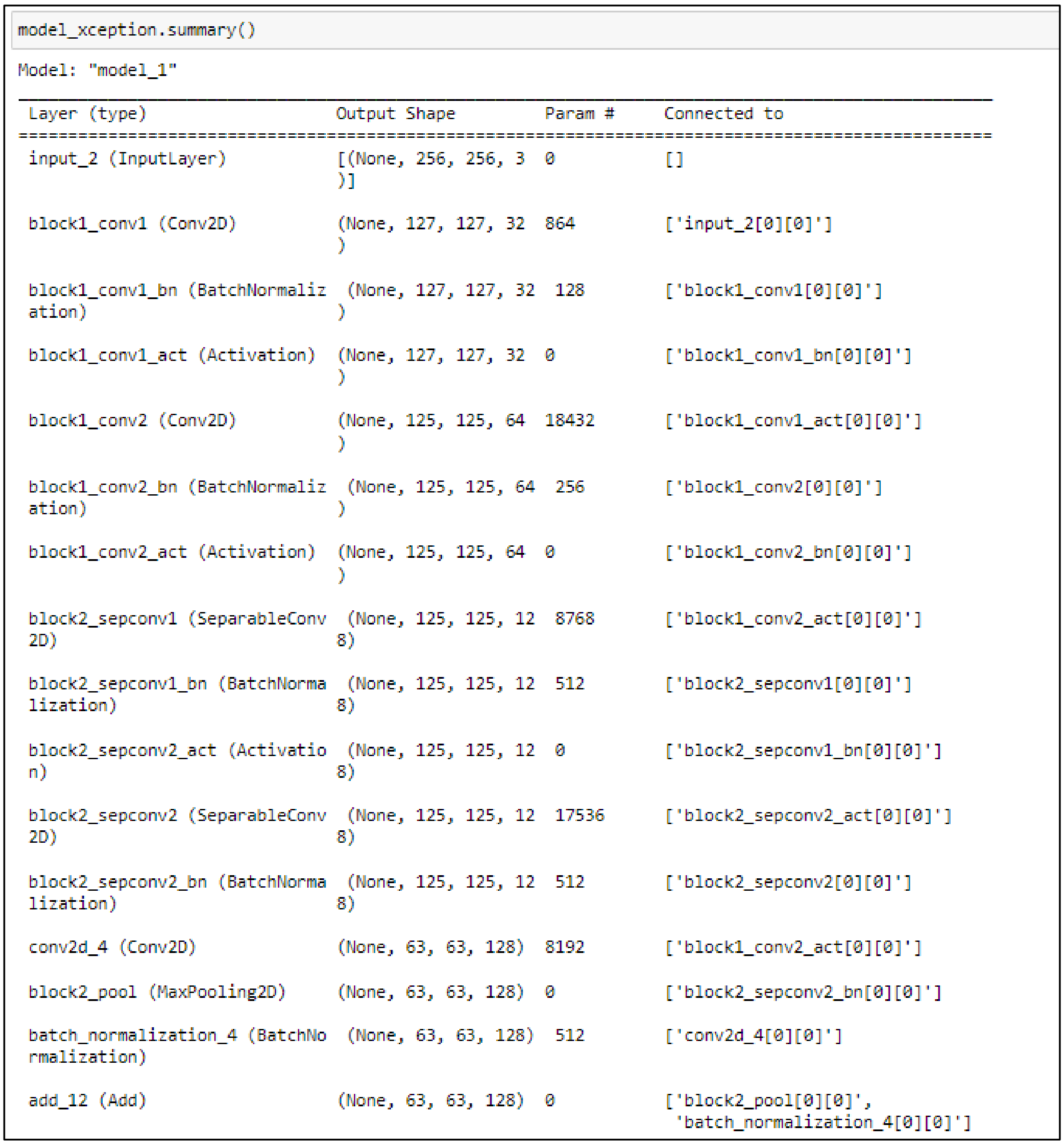


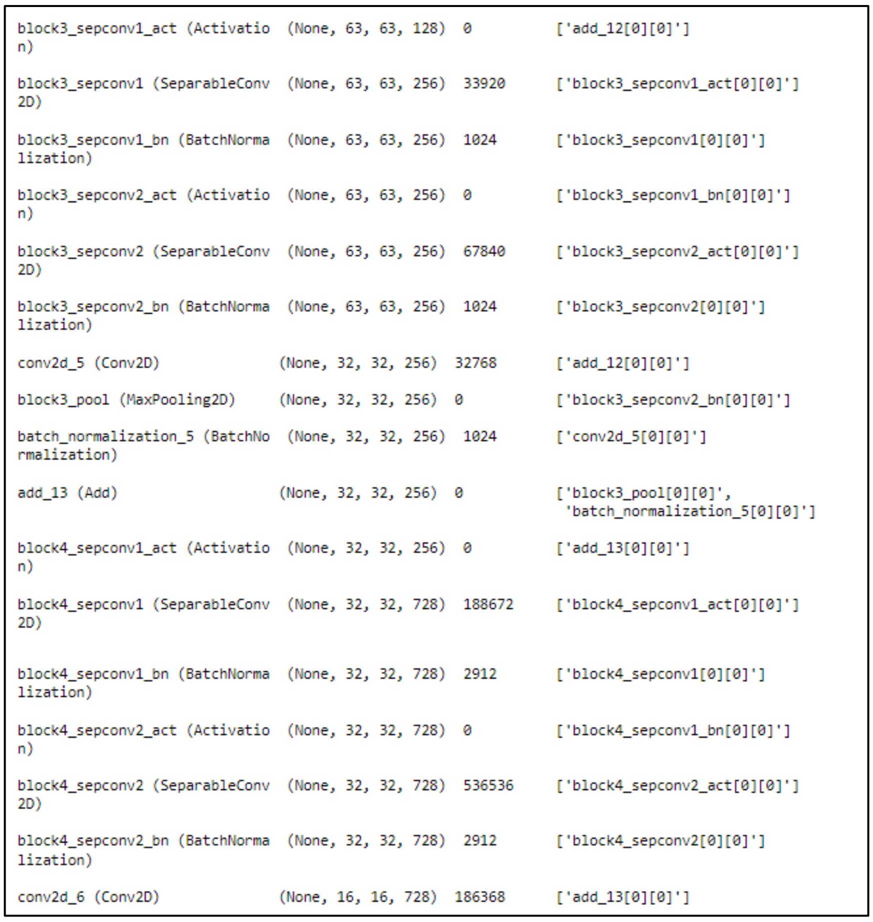
Figure 18: Architecture of Xception

Xception has a pre-define layer as per the below image:

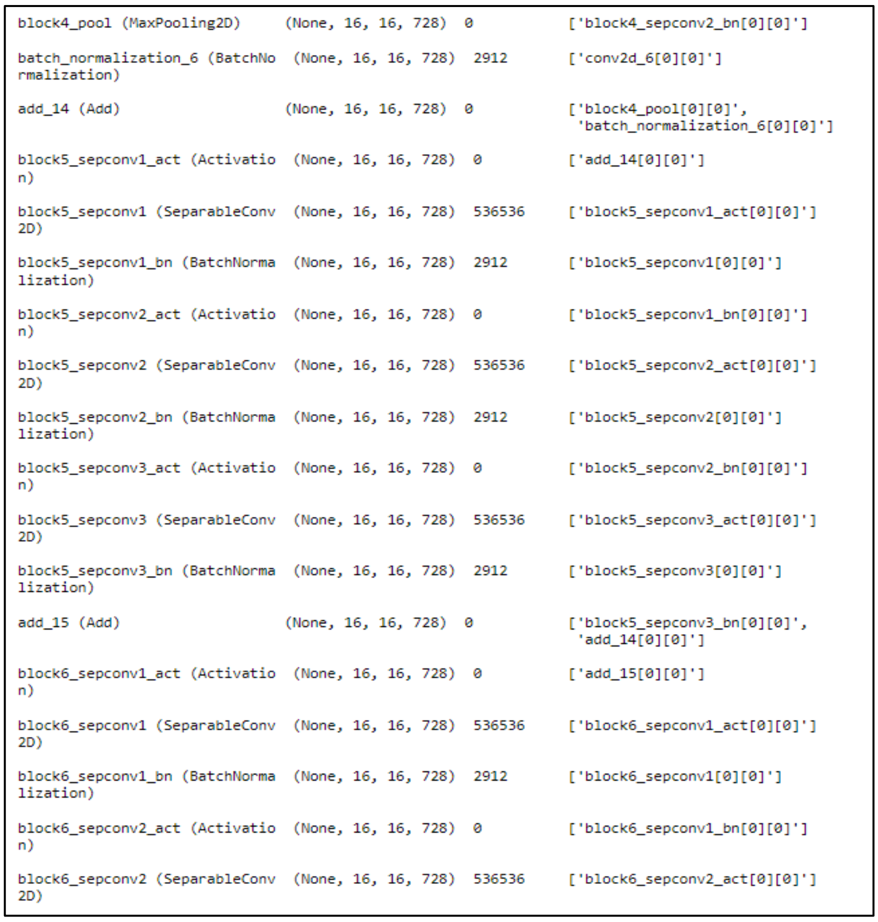


```
model_xception.summary()
```

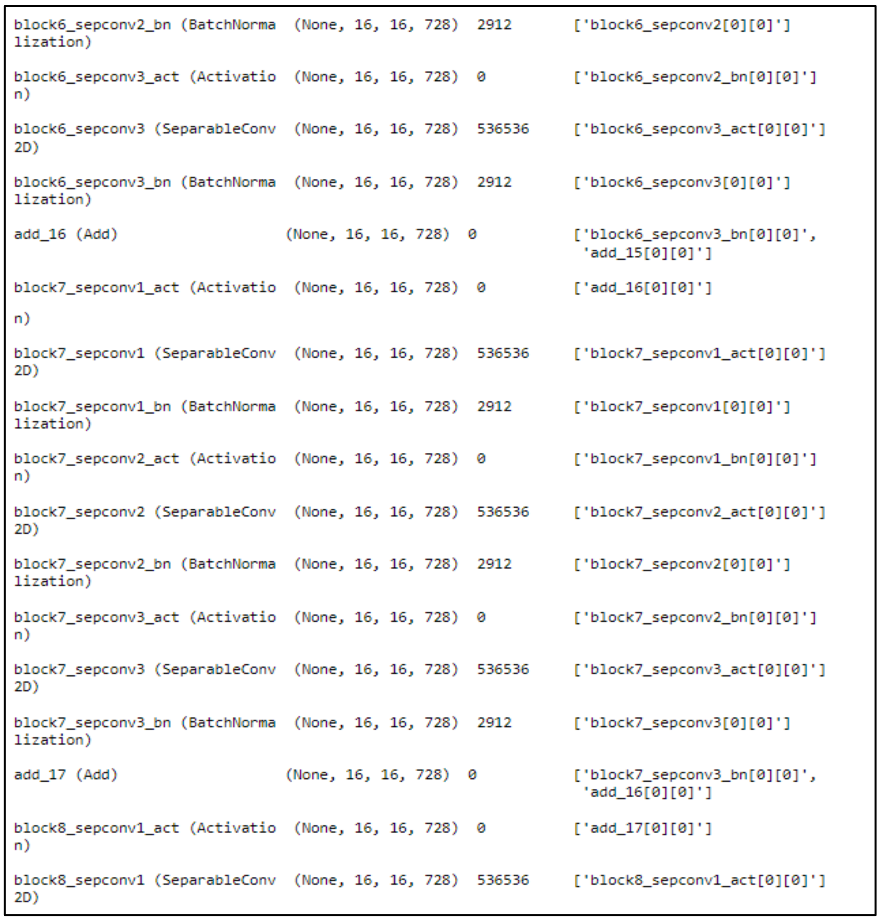
```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 256, 256, 3  0           []
                                )]

 block1_conv1 (Conv2D)          (None, 127, 127, 32  864         ['input_2[0][0]']
                                )

 block1_conv1_bn (BatchNormaliz  (None, 127, 127, 32  128        ['block1_conv1[0][0]']
 ation)                         )

 block1_conv1_act (Activation)  (None, 127, 127, 32  0           ['block1_conv1_bn[0][0]']
                                )

 block1_conv2 (Conv2D)          (None, 125, 125, 64  18432       ['block1_conv1_act[0][0]']
                                )

 block1_conv2_bn (BatchNormaliz  (None, 125, 125, 64  256        ['block1_conv2[0][0]']
 ation)                         )

 block1_conv2_act (Activation)  (None, 125, 125, 64  0           ['block1_conv2_bn[0][0]']
                                )

 block2_sepconv1 (SeparableConv  (None, 125, 125, 12  8768       ['block1_conv2_act[0][0]']
 2D)                            8)

 block2_sepconv1_bn (BatchNorma  (None, 125, 125, 12  512        ['block2_sepconv1[0][0]']
 lization)                      8)

 block2_sepconv2_act (Activatio  (None, 125, 125, 12  0          ['block2_sepconv1_bn[0][0]']
 n)                             8)

 block2_sepconv2 (SeparableConv  (None, 125, 125, 12  17536      ['block2_sepconv2_act[0][0]']
 2D)                            8)

 block2_sepconv2_bn (BatchNorma  (None, 125, 125, 12  512        ['block2_sepconv2[0][0]']
 lization)                      8)

 conv2d_4 (Conv2D)              (None, 63, 63, 128)  8192        ['block1_conv2_act[0][0]']

 block2_pool (MaxPooling2D)     (None, 63, 63, 128)  0           ['block2_sepconv2_bn[0][0]']

 batch_normalization_4 (BatchNo  (None, 63, 63, 128)  512        ['conv2d_4[0][0]']
 rmalization)

 add_12 (Add)                   (None, 63, 63, 128)  0           ['block2_pool[0][0]',
                                                                  'batch_normalization_4[0][0]']
```



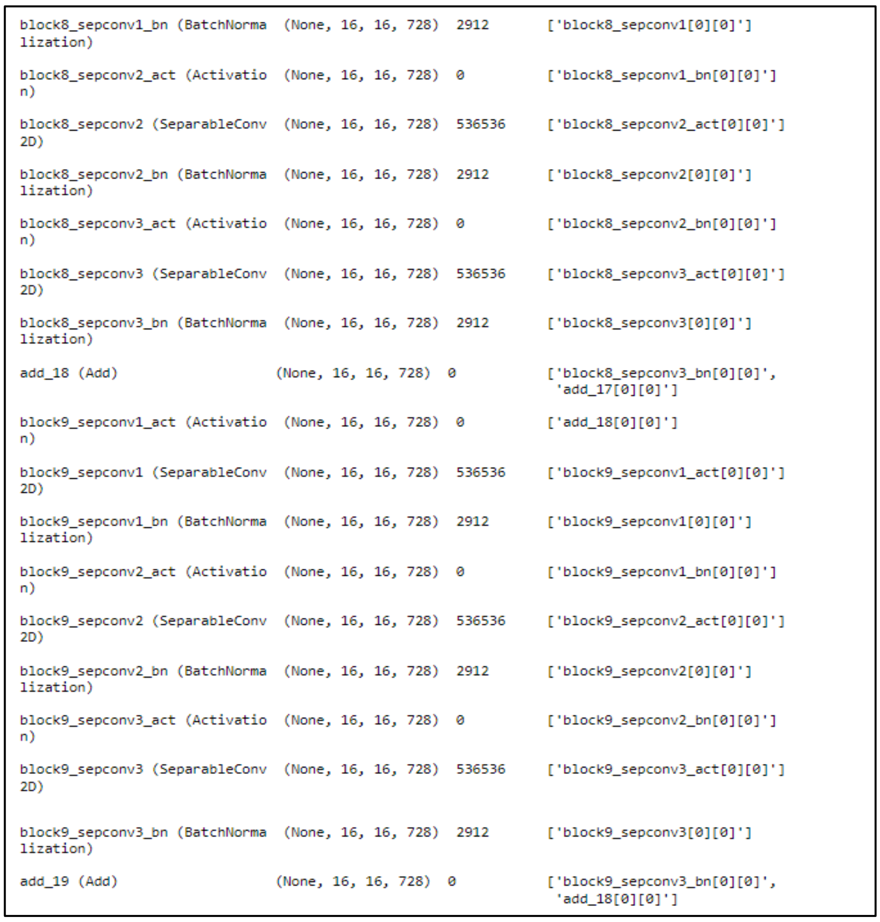
```
block3_sepconv1_act (Activatio  (None, 63, 63, 128)  0          ['add_12[0][0]']
n)

block3_sepconv1 (SeparableConv  (None, 63, 63, 256)  33920      ['block3_sepconv1_act[0][0]']
2D)

block3_sepconv1_bn (BatchNorma  (None, 63, 63, 256)  1024       ['block3_sepconv1[0][0]']
lization)

block3_sepconv2_act (Activatio  (None, 63, 63, 256)  0          ['block3_sepconv1_bn[0][0]']
n)

block3_sepconv2 (SeparableConv  (None, 63, 63, 256)  67840      ['block3_sepconv2_act[0][0]']
2D)

block3_sepconv2_bn (BatchNorma  (None, 63, 63, 256)  1024       ['block3_sepconv2[0][0]']
lization)

conv2d_5 (Conv2D)              (None, 32, 32, 256)  32768      ['add_12[0][0]']

block3_pool (MaxPooling2D)     (None, 32, 32, 256)  0          ['block3_sepconv2_bn[0][0]']

batch_normalization_5 (BatchNo  (None, 32, 32, 256)  1024       ['conv2d_5[0][0]']
rmalization)

add_13 (Add)                   (None, 32, 32, 256)  0          ['block3_pool[0][0]',
                                                                 'batch_normalization_5[0][0]']

block4_sepconv1_act (Activatio  (None, 32, 32, 256)  0          ['add_13[0][0]']
n)

block4_sepconv1 (SeparableConv  (None, 32, 32, 728)  188672     ['block4_sepconv1_act[0][0]']
2D)

block4_sepconv1_bn (BatchNorma  (None, 32, 32, 728)  2912       ['block4_sepconv1[0][0]']
lization)

block4_sepconv2_act (Activatio  (None, 32, 32, 728)  0          ['block4_sepconv1_bn[0][0]']
n)

block4_sepconv2 (SeparableConv  (None, 32, 32, 728)  536536     ['block4_sepconv2_act[0][0]']
2D)

block4_sepconv2_bn (BatchNorma  (None, 32, 32, 728)  2912       ['block4_sepconv2[0][0]']
lization)

conv2d_6 (Conv2D)              (None, 16, 16, 728)  186368     ['add_13[0][0]']
```



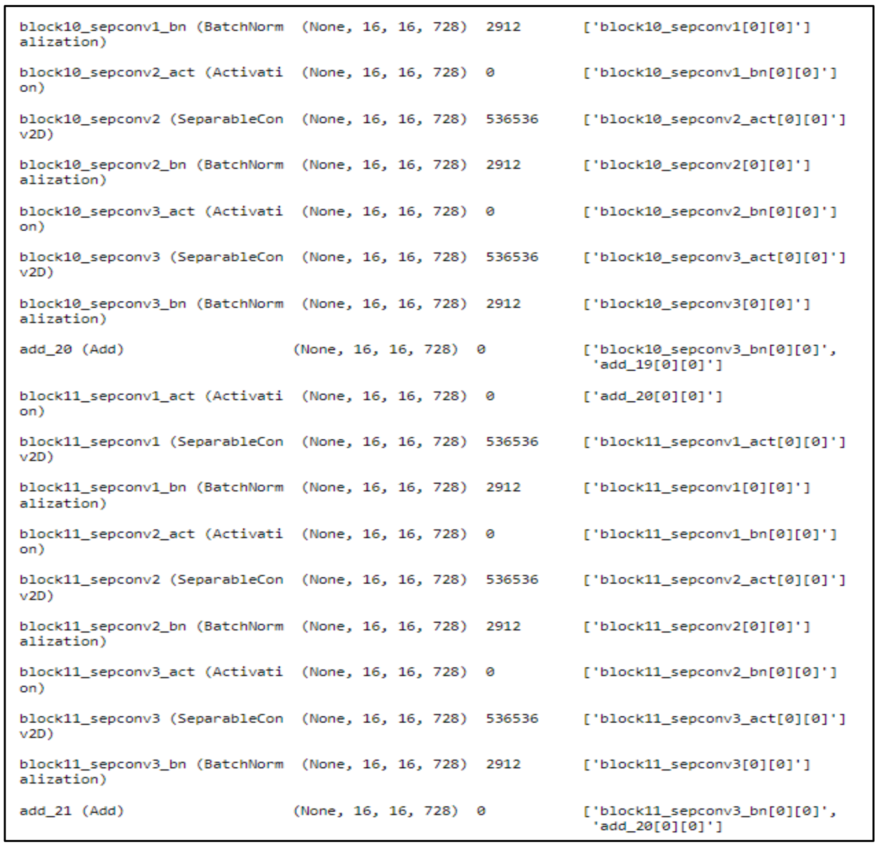
```
block4_pool (MaxPooling2D)     (None, 16, 16, 728)  0          ['block4_sepconv2_bn[0][0]']

batch_normalization_6 (BatchNo  (None, 16, 16, 728)  2912       ['conv2d_6[0][0]']
rmalization)

add_14 (Add)                   (None, 16, 16, 728)  0          ['block4_pool[0][0]',
                                                                 'batch_normalization_6[0][0]']

block5_sepconv1_act (Activatio  (None, 16, 16, 728)  0          ['add_14[0][0]']
n)

block5_sepconv1 (SeparableConv  (None, 16, 16, 728)  536536     ['block5_sepconv1_act[0][0]']
2D)

block5_sepconv1_bn (BatchNorma  (None, 16, 16, 728)  2912       ['block5_sepconv1[0][0]']
lization)

block5_sepconv2_act (Activatio  (None, 16, 16, 728)  0          ['block5_sepconv1_bn[0][0]']
n)

block5_sepconv2 (SeparableConv  (None, 16, 16, 728)  536536     ['block5_sepconv2_act[0][0]']
2D)

block5_sepconv2_bn (BatchNorma  (None, 16, 16, 728)  2912       ['block5_sepconv2[0][0]']
lization)

block5_sepconv3_act (Activatio  (None, 16, 16, 728)  0          ['block5_sepconv2_bn[0][0]']
n)

block5_sepconv3 (SeparableConv  (None, 16, 16, 728)  536536     ['block5_sepconv3_act[0][0]']
2D)

block5_sepconv3_bn (BatchNorma  (None, 16, 16, 728)  2912       ['block5_sepconv3[0][0]']
lization)

add_15 (Add)                   (None, 16, 16, 728)  0          ['block5_sepconv3_bn[0][0]',
                                                                 'add_14[0][0]']

block6_sepconv1_act (Activatio  (None, 16, 16, 728)  0          ['add_15[0][0]']
n)

block6_sepconv1 (SeparableConv  (None, 16, 16, 728)  536536     ['block6_sepconv1_act[0][0]']
2D)

block6_sepconv1_bn (BatchNorma  (None, 16, 16, 728)  2912       ['block6_sepconv1[0][0]']
lization)

block6_sepconv2_act (Activatio  (None, 16, 16, 728)  0          ['block6_sepconv1_bn[0][0]']
n)

block6_sepconv2 (SeparableConv  (None, 16, 16, 728)  536536     ['block6_sepconv2_act[0][0]']
2D)
```



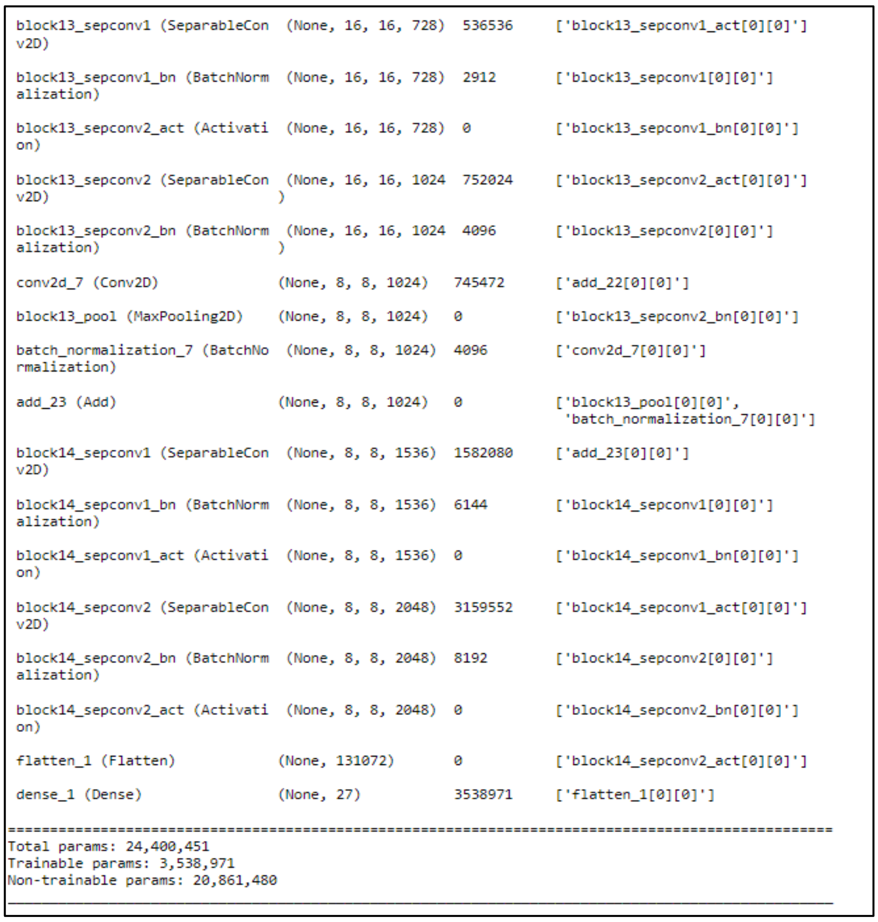
```
block6_sepconv2_bn (BatchNorma  (None, 16, 16, 728)  2912       ['block6_sepconv2[0][0]']
lization)

block6_sepconv3_act (Activatio  (None, 16, 16, 728)  0          ['block6_sepconv2_bn[0][0]']
n)

block6_sepconv3 (SeparableConv  (None, 16, 16, 728)  536536     ['block6_sepconv3_act[0][0]']
2D)

block6_sepconv3_bn (BatchNorma  (None, 16, 16, 728)  2912       ['block6_sepconv3[0][0]']
lization)

add_16 (Add)                   (None, 16, 16, 728)  0           ['block6_sepconv3_bn[0][0]',
                                                                 'add_15[0][0]']

block7_sepconv1_act (Activatio  (None, 16, 16, 728)  0          ['add_16[0][0]']

n)

block7_sepconv1 (SeparableConv  (None, 16, 16, 728)  536536     ['block7_sepconv1_act[0][0]']
2D)

block7_sepconv1_bn (BatchNorma  (None, 16, 16, 728)  2912       ['block7_sepconv1[0][0]']
lization)

block7_sepconv2_act (Activatio  (None, 16, 16, 728)  0          ['block7_sepconv1_bn[0][0]']
n)

block7_sepconv2 (SeparableConv  (None, 16, 16, 728)  536536     ['block7_sepconv2_act[0][0]']
2D)

block7_sepconv2_bn (BatchNorma  (None, 16, 16, 728)  2912       ['block7_sepconv2[0][0]']
lization)

block7_sepconv3_act (Activatio  (None, 16, 16, 728)  0          ['block7_sepconv2_bn[0][0]']
n)

block7_sepconv3 (SeparableConv  (None, 16, 16, 728)  536536     ['block7_sepconv3_act[0][0]']
2D)

block7_sepconv3_bn (BatchNorma  (None, 16, 16, 728)  2912       ['block7_sepconv3[0][0]']
lization)

add_17 (Add)                   (None, 16, 16, 728)  0           ['block7_sepconv3_bn[0][0]',
                                                                 'add_16[0][0]']

block8_sepconv1_act (Activatio  (None, 16, 16, 728)  0          ['add_17[0][0]']
n)

block8_sepconv1 (SeparableConv  (None, 16, 16, 728)  536536     ['block8_sepconv1_act[0][0]']
2D)
```



```
block8_sepconv1_bn (BatchNorma  (None, 16, 16, 728)  2912       ['block8_sepconv1[0][0]']
lization)

block8_sepconv2_act (Activatio  (None, 16, 16, 728)  0          ['block8_sepconv1_bn[0][0]']
n)

block8_sepconv2 (SeparableConv  (None, 16, 16, 728)  536536     ['block8_sepconv2_act[0][0]']
2D)

block8_sepconv2_bn (BatchNorma  (None, 16, 16, 728)  2912       ['block8_sepconv2[0][0]']
lization)

block8_sepconv3_act (Activatio  (None, 16, 16, 728)  0          ['block8_sepconv2_bn[0][0]']
n)

block8_sepconv3 (SeparableConv  (None, 16, 16, 728)  536536     ['block8_sepconv3_act[0][0]']
2D)

block8_sepconv3_bn (BatchNorma  (None, 16, 16, 728)  2912       ['block8_sepconv3[0][0]']
lization)

add_18 (Add)                   (None, 16, 16, 728)  0           ['block8_sepconv3_bn[0][0]',
                                                                 'add_17[0][0]']

block9_sepconv1_act (Activatio  (None, 16, 16, 728)  0          ['add_18[0][0]']
n)

block9_sepconv1 (SeparableConv  (None, 16, 16, 728)  536536     ['block9_sepconv1_act[0][0]']
2D)

block9_sepconv1_bn (BatchNorma  (None, 16, 16, 728)  2912       ['block9_sepconv1[0][0]']
lization)

block9_sepconv2_act (Activatio  (None, 16, 16, 728)  0          ['block9_sepconv1_bn[0][0]']
n)

block9_sepconv2 (SeparableConv  (None, 16, 16, 728)  536536     ['block9_sepconv2_act[0][0]']
2D)

block9_sepconv2_bn (BatchNorma  (None, 16, 16, 728)  2912       ['block9_sepconv2[0][0]']
lization)

block9_sepconv3_act (Activatio  (None, 16, 16, 728)  0          ['block9_sepconv2_bn[0][0]']
n)

block9_sepconv3 (SeparableConv  (None, 16, 16, 728)  536536     ['block9_sepconv3_act[0][0]']
2D)

block9_sepconv3_bn (BatchNorma  (None, 16, 16, 728)  2912       ['block9_sepconv3[0][0]']
lization)

add_19 (Add)                   (None, 16, 16, 728)  0           ['block9_sepconv3_bn[0][0]',
                                                                 'add_18[0][0]']
```



```
block10_sepconv1_bn (BatchNorm  (None, 16, 16, 728)  2912       ['block10_sepconv1[0][0]']
alization)

block10_sepconv2_act (Activati  (None, 16, 16, 728)  0          ['block10_sepconv1_bn[0][0]']
on)

block10_sepconv2 (SeparableCon  (None, 16, 16, 728)  536536     ['block10_sepconv2_act[0][0]']
v2D)

block10_sepconv2_bn (BatchNorm  (None, 16, 16, 728)  2912       ['block10_sepconv2[0][0]']
alization)

block10_sepconv3_act (Activati  (None, 16, 16, 728)  0          ['block10_sepconv2_bn[0][0]']
on)

block10_sepconv3 (SeparableCon  (None, 16, 16, 728)  536536     ['block10_sepconv3_act[0][0]']
v2D)

block10_sepconv3_bn (BatchNorm  (None, 16, 16, 728)  2912       ['block10_sepconv3[0][0]']
alization)

add_20 (Add)                   (None, 16, 16, 728)  0          ['block10_sepconv3_bn[0][0]',
                                                                 'add_19[0][0]']

block11_sepconv1_act (Activati  (None, 16, 16, 728)  0          ['add_20[0][0]']
on)

block11_sepconv1 (SeparableCon  (None, 16, 16, 728)  536536     ['block11_sepconv1_act[0][0]']
v2D)

block11_sepconv1_bn (BatchNorm  (None, 16, 16, 728)  2912       ['block11_sepconv1[0][0]']
alization)

block11_sepconv2_act (Activati  (None, 16, 16, 728)  0          ['block11_sepconv1_bn[0][0]']
on)

block11_sepconv2 (SeparableCon  (None, 16, 16, 728)  536536     ['block11_sepconv2_act[0][0]']
v2D)

block11_sepconv2_bn (BatchNorm  (None, 16, 16, 728)  2912       ['block11_sepconv2[0][0]']
alization)

block11_sepconv3_act (Activati  (None, 16, 16, 728)  0          ['block11_sepconv2_bn[0][0]']
on)

block11_sepconv3 (SeparableCon  (None, 16, 16, 728)  536536     ['block11_sepconv3_act[0][0]']
v2D)

block11_sepconv3_bn (BatchNorm  (None, 16, 16, 728)  2912       ['block11_sepconv3[0][0]']
alization)

add_21 (Add)                   (None, 16, 16, 728)  0          ['block11_sepconv3_bn[0][0]',
                                                                 'add_20[0][0]']
```



```
block13_sepconv1 (SeparableCon  (None, 16, 16, 728)  536536     ['block13_sepconv1_act[0][0]']
v2D)

block13_sepconv1_bn (BatchNorm  (None, 16, 16, 728)  2912       ['block13_sepconv1[0][0]']
alization)

block13_sepconv2_act (Activati  (None, 16, 16, 728)  0          ['block13_sepconv1_bn[0][0]']
on)

block13_sepconv2 (SeparableCon  (None, 16, 16, 1024  752024     ['block13_sepconv2_act[0][0]']
v2D)                            )

block13_sepconv2_bn (BatchNorm  (None, 16, 16, 1024  4096       ['block13_sepconv2[0][0]']
alization)                      )

conv2d_7 (Conv2D)              (None, 8, 8, 1024)   745472     ['add_22[0][0]']

block13_pool (MaxPooling2D)    (None, 8, 8, 1024)   0          ['block13_sepconv2_bn[0][0]']

batch_normalization_7 (BatchNo  (None, 8, 8, 1024)  4096       ['conv2d_7[0][0]']
rmalization)

add_23 (Add)                   (None, 8, 8, 1024)   0          ['block13_pool[0][0]',
                                                                 'batch_normalization_7[0][0]']

block14_sepconv1 (SeparableCon  (None, 8, 8, 1536)  1582080    ['add_23[0][0]']
v2D)

block14_sepconv1_bn (BatchNorm  (None, 8, 8, 1536)  6144       ['block14_sepconv1[0][0]']
alization)

block14_sepconv1_act (Activati  (None, 8, 8, 1536)  0          ['block14_sepconv1_bn[0][0]']
on)

block14_sepconv2 (SeparableCon  (None, 8, 8, 2048)  3159552    ['block14_sepconv1_act[0][0]']
v2D)

block14_sepconv2_bn (BatchNorm  (None, 8, 8, 2048)  8192       ['block14_sepconv2[0][0]']
alization)

block14_sepconv2_act (Activati  (None, 8, 8, 2048)  0          ['block14_sepconv2_bn[0][0]']
on)

flatten_1 (Flatten)            (None, 131072)       0          ['block14_sepconv2_act[0][0]']

dense_1 (Dense)                (None, 27)           3538971    ['flatten_1[0][0]']

==================================================================================================
Total params: 24,400,451
Trainable params: 3,538,971
Non-trainable params: 20,861,480
__________________________________________________________________________________________________
```

Figure 19: Layers of Xception

Results of models

Using CNN model:

By using the CNN model with a manually designed layer we have achieved the accuracy of 0.9168 i.e., 91.68% and a loss of 0.2587. The time required to train a model using CNN was less compared to other models. The graph below shows that the model doesn't overfit or underfit. The lines are close to each other which shows that model is working perfectly with train data and test data.

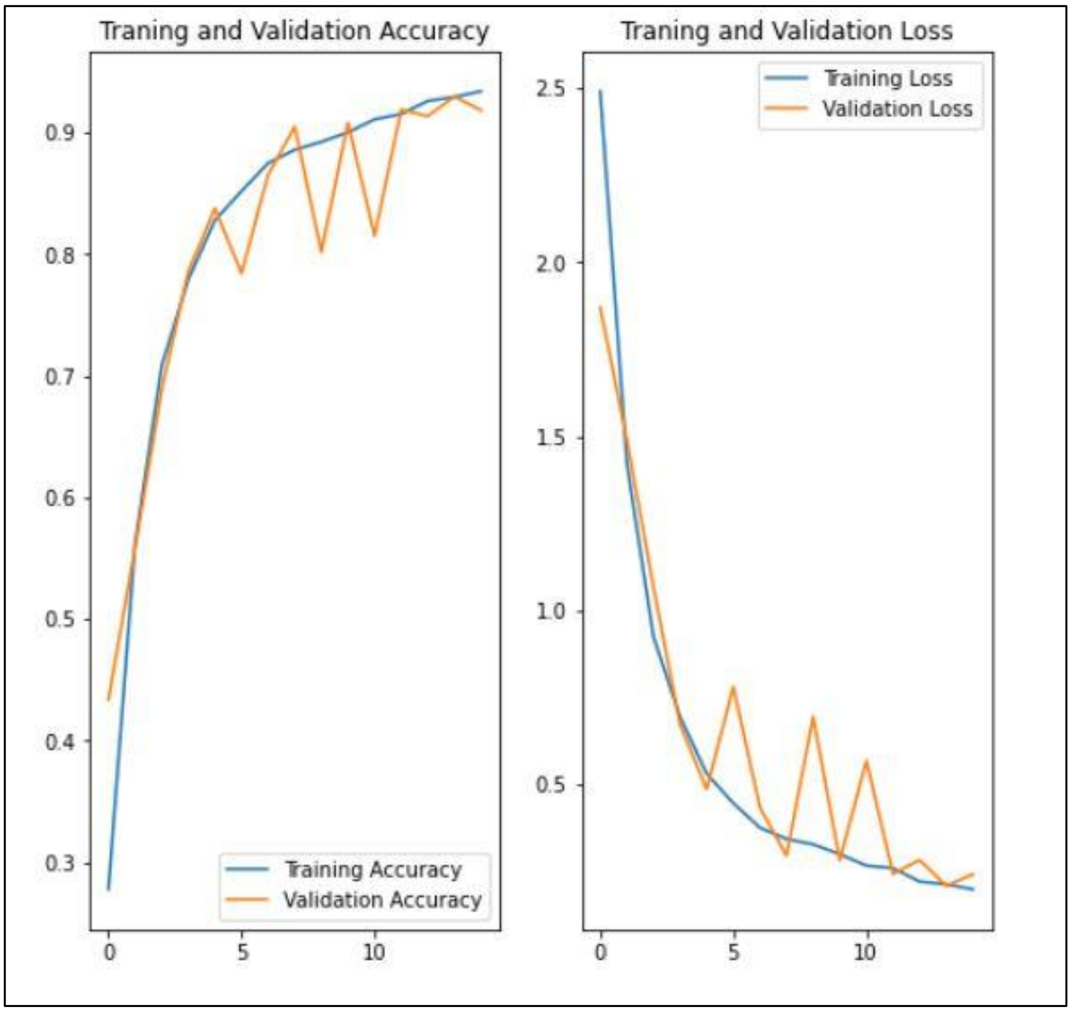


Figure 20: CNN working on train and test data

By using VGG

By using VGG16 which has pre-defined layers we have achieved the accuracy of 0.9337 i.e., 93.37%, and a loss of 7.4822. It has a lot of loss while capturing the features. The graph below shows that the model is overfitting.

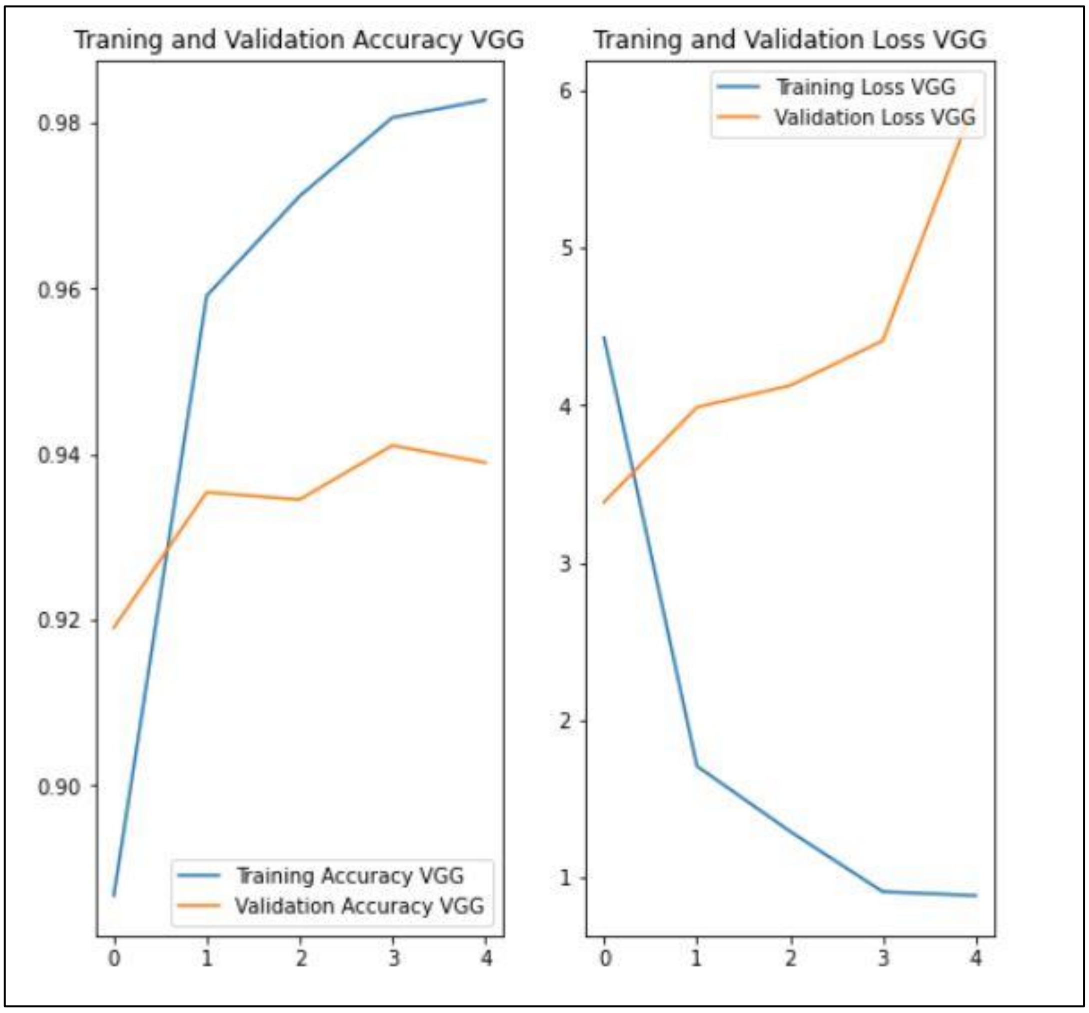


Figure 21: VGG16 working on train and test data

By using Xception

By using VGG16 which has pre-defined layers we have achieved the accuracy of 0.6168 i.e., 61.68%, and a loss of 103.4860. It has a huge loss while capturing the features. The graph below shows that the model is overfitting.

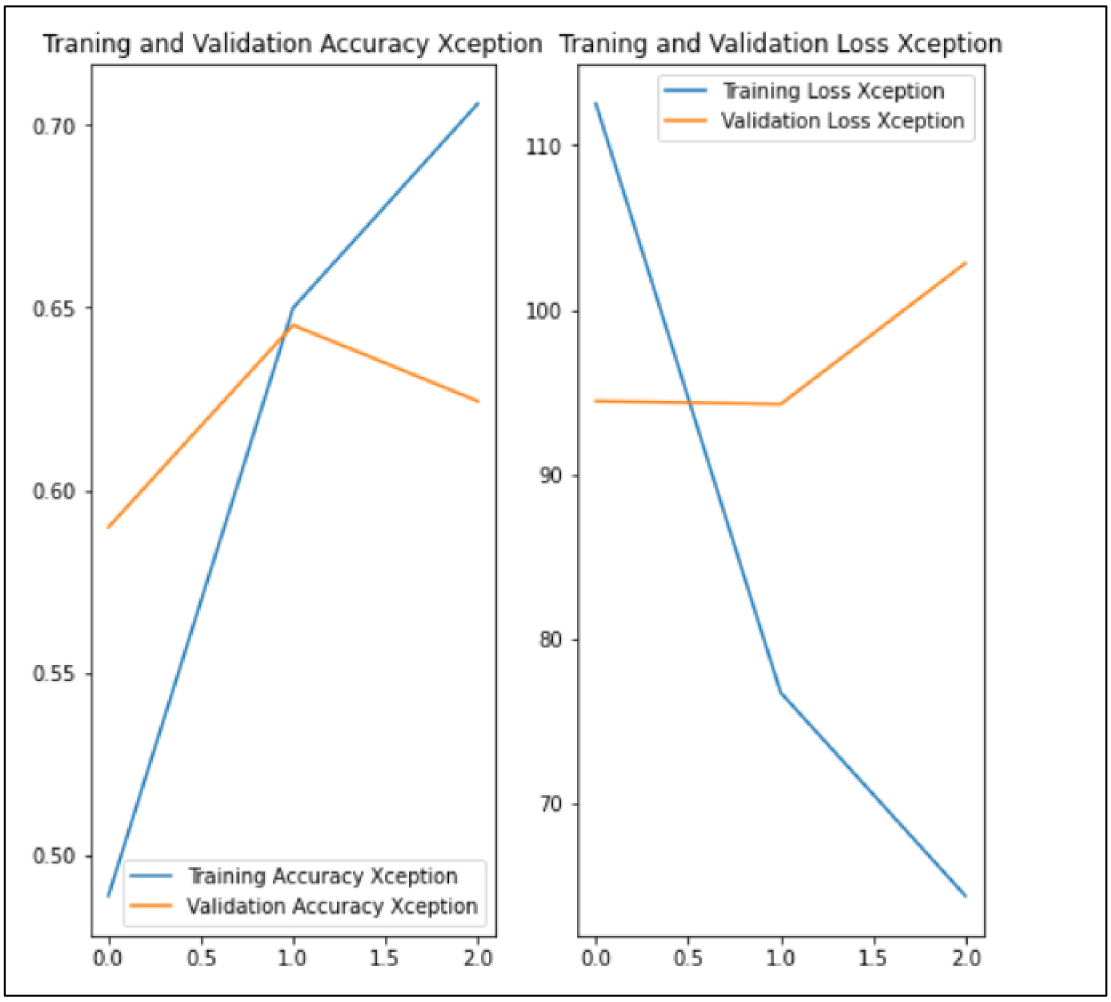


Figure 22: Xcpetion working on train and test data

Deployed model

We have decided to deploy a manually designed CNN model as it is a perfectly fitted model and has good accuracy over testing data. Also during training, they will be able to grasp relevant features. No matter what task you are working on, you can use the CNN which has been pre-trained and adjust the weights according to the data you feed it.

In contrast to regular neural networks, CNNs are more computationally efficient because of the procedure of convolution. By using parameter sharing and dimensionality reduction, CNN models can be deployed quickly and easily. Any device, including smartphones, can run them.

In image classification, state-of-the-art neural networks are not convolutional, such as those in image transformers. In most cases and for tasks such as recognizing images and videos, CNNs have now dominated for a very long time. When there is a lot of data, they are usually more accurate than non-convolutional NNs.

The reason for not selecting VGG and Xception:

The VGG16 and Xception model is a heavier model than the CNN.

The vanishing gradient problem is one major disadvantage of this model. As we can see from my validation loss graph, it is increasing over time. Other models did not exhibit this behavior.

In order to train a big model from scratch, you'll need a lot of resources (time and computing capacity).

Results

Prediction of disease

In our system, we use the CNN architecture model to detect plant diseases in real-time in order to detect them efficiently. It is our intention to build a system for detecting and identifying plant diseases as part of our research. Our proposed system recognizes plant diseases with a 91.68% success rate in a test dataset, suggesting that it performs well. We have created a website to detect plant diseases. We just need to upload an image of a plant leaf that has diseases on it. Then the website will show the results as the name of the disease and percentage of match. Below are the screenshot of the prediction.

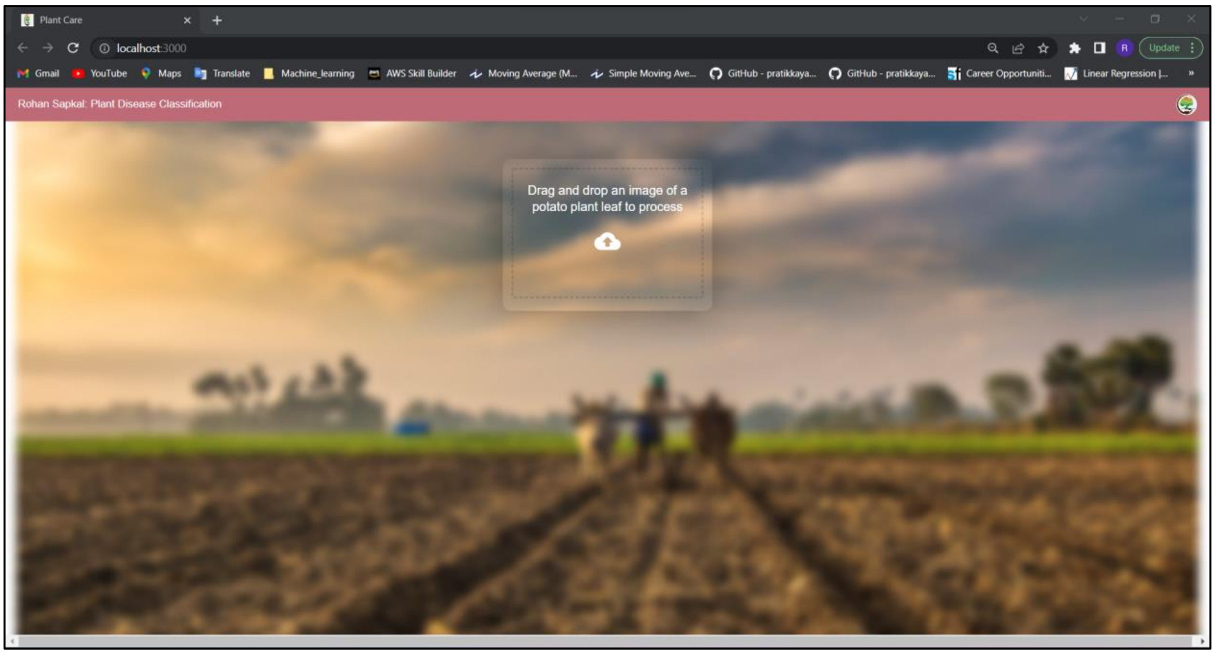


Figure 23: Home page of website

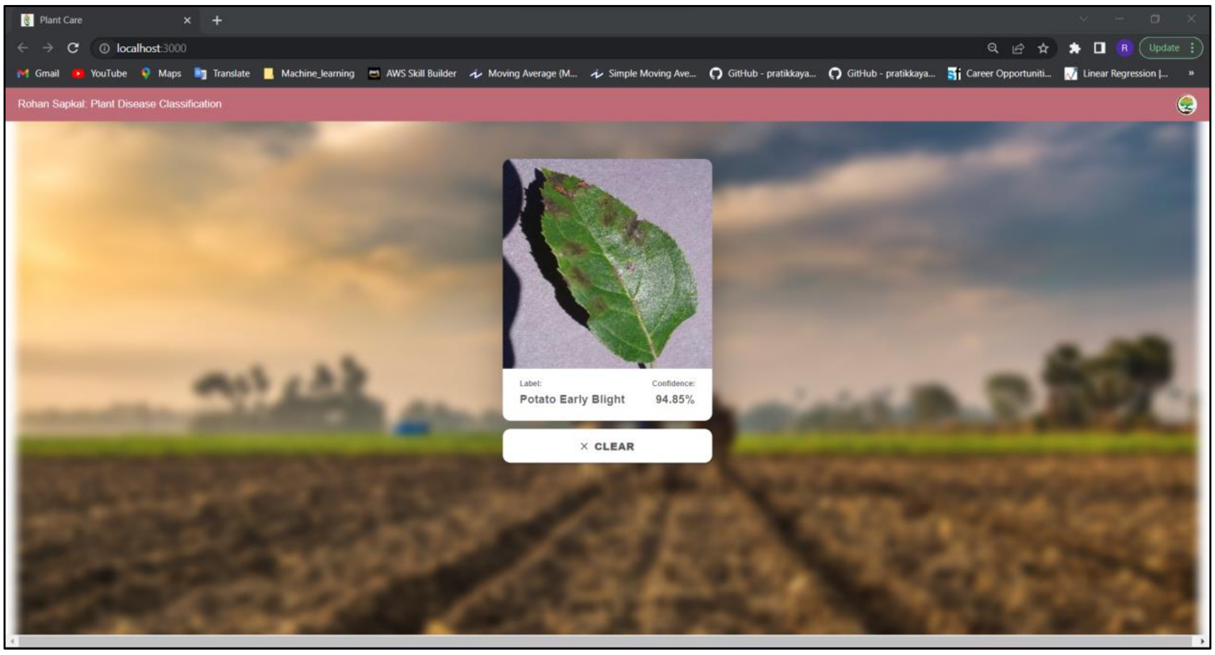

Plant Care
localhost:3000
Gmail YouTube Maps Translate Machine_learning AWS Skill Builder Moving Average (M... Simple Moving Ave... GitHub - pratikkaya... GitHub - pratikkaya... Career Opportuniti... Linear Regression |...
Rohan Sapkal: Plant Disease Classification
Label:
Potato Early Blight
Confidence:
94.85%
CLEAR

Figure 24: Prediction of Disease(1)

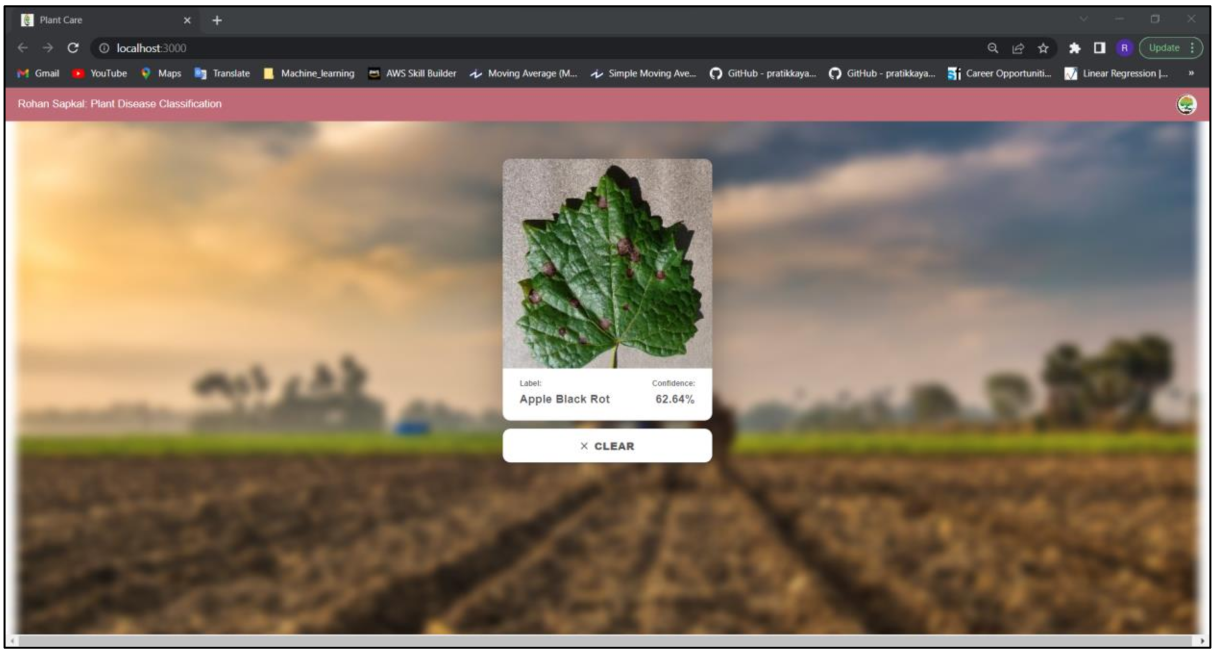

Plant Care
localhost:3000
Gmail YouTube Maps Translate Machine_learning AWS Skill Builder Moving Average (M... Simple Moving Ave... GitHub - pratikkaya... GitHub - pratikkaya... Career Opportuniti... Linear Regression |...
Rohan Sapkal: Plant Disease Classification
Label:
Apple Black Rot
Confidence:
62.64%
CLEAR

Figure 25: Prediction of Disease(2)

Conclusion

An automatic system for detecting plant diseases has been developed using deep learning capabilities. CNN feature extraction functionalities are exploited to build this simple classification system. A fully connected layer is used for prediction at the end of the model. The research was carried out using 26947 publicly available images, as well as 843 images taken in actual environments and experimental conditions. Overall, the system achieved a testing accuracy of 91.68% on publicly available datasets, and performed well when tested on images. According to accuracy, CNN is an excellent tool for detecting and diagnosing plants automatically. The other two models which are VGG16 and Xception also has high accuracy but both the models are overfitted so we can't use them in the real world. It could tell whether the plant has a disease or not, even though it was trained on a dataset with only 27 classes - somehow the symptoms are the same for all kinds of plants.

Future scope

Various advanced pre-processing and augmentation methods, such as data manipulation, will be replicated in future work, which includes the enhancement of reference image data. One of our future scopes is to train datasets with other models such as AlexNet, AlexNetOWTbn, GoogLeNet, Overfeat, GoogLeNet, ResNet-50, ResNet-101, Inception-v3, InceptionResNetv2, and SqueezeNet. So that we can find a perfect model for plant disease detection. Another future scope is that we can develop an mobile application which can predict plant disease in real time.

References

Aravindhan Venkataramanan, D. K. (n.d.). Plant Disease Detection and Classification Using Deep Neural Networks. *International Journal on Computer Science and Engineering (IJCSE)., e-ISSN : 0975-3397.*

Caglayan, A. G. ((2013, September)). A plant recognition approach using shape and color features in leaf images. *International Conference on Image Analysis and Processing (pp. 161-170).* Springer, Berlin, Heidelberg.

CNN model explanation. (n.d.). Retrieved from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Davinder Singh, N. J. (n.d.). PlantDoc: A Dataset for Visual Plant Disease Detection. *Indian Institute of Technology Gandhinagar*. Gujarat, India 382 355.

Fang, Y. &. (2015). *Current and prospective methods for plant disease detection.* . Biosensors, 5(3), 537-561.

Godliver Owomugisha, J. A. (2014). Automated Vision-Based Diagnosis of Banana Bacterial Wilt Disease and Black Sigatoka Disease . *Preceding of the 1'st international conference on the use of mobile ICT .* Africa .

Huntley, B. (1991). *How plants respond to climate change: migration rates, individualism and the consequences for plant communities.* Annals of Botany.

Isleib, J. &. (2018, October 2). *Signs and symptoms of plant disease: Is if fungal, viral or bacterial?* Retrieved from https://www.canr.msu.edu/news/signs_and_symptoms_of_plant_disease_is_it_fungal_viral_or_bacterial

Jia Deng, W. D.-J.-F. (2009). Imagenet: A large-scale hierarchical image database. *IEEE conference on computer vision and pattern recognition.*

Kshirsagar, P. R. (2015). Cotton Leaf Disease Identification using Pattern Recognition Techniques. *International Conference on Pervasive Computing (ICPC).*

Muhammad Hammad Saleem, J. P. (n.d.). *Plant Disease Detection and Classification by Deep Learning.* Retrieved from An Open Access Journal from MDPI.

Murk Chohan, A. K. (Volume-9 Issue-1, May 2020). Plant Disease Detection using Deep Learning. *International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878.*

Mustafa, M. S. (n.d.). Development of automated hybrid intelligent system for herbs plant classification and early herbs plant disease detection. Neural Computing and Applications, 1-23.

Resources for Plant Diseases. (APSNet. 2019.). Retrieved from https://www.apsnet.org/edcenter/resources/commonnames/Pages/default.aspx

S. S. Sannakki, V. S. (Vol2 Issue: 02 | May-2015). Classification of Pomegranate Diseases Based on Back. *International Research Journal of Engineering and Technology* .

S. Yun, W. X. (vol. 8, no. 4, p. 60, 2015). Pnn based crop disease recognition with leaf image features and meteorological data. *International Journal of Agricultural and Biological Engineering.*

Sanjay B Patil, K. S. (2011). Betel Leaf Area Measurement Using Image Processing. *International Journal on Computer Science and Engineering (IJCSE).*

Shruthi, U. N. (2019). A Review on Machine Learning Classification Techniques for Plant Disease Detection. *International Conference on Advanced Computing & Communication Systems (ICACCS) (pp. 281-284).*

Wang P., C. K. (2016). Multimodal classification of mild cognitive impairment based on partial least squares.

Williams, S. D. (2017, February 1). Retrieved from Plants Get Sick Too!: https://ohioline.osu.edu/factsheet/plpath-gen-1

Zhen, X. W. (2014.). Direct estimation of cardiac bi-ventricular volumes with regression forests. *Medical Image Com- puting and Computer-Assisted Intervention– MICCAI* .

Appendix A

Technical Specifications and Requirement

<u>System Requirements</u>

- 16 GB RAM (Minimum)
- 1 TB HDD / SDD
- MUlti-core processor
- Microsoft Windows 7 or higher

<u>Tools</u>

- Microsoft Visual Studio Code
- Anaconda Navigator
- Jupyter Notebook

<u>Programming Languages</u>

- Python programming
- Node js
- React Native
- HTML
- FastAPI