**PROJECT PHASE-III REPORT (IT401)**

**ON**

**Image Classification using Deep Learning**

*A report submitted in partial fulfilment of the requirement for the award of*

*The degree of*

**BACHELOR OF TECHNOLOGY**

**In**

**INFORMATION TECHNOLOGY**



**Submitted to:**

Dr. Mitali Srivastava

Assistant Professor

**Submitted by:**

Anubhav Negi-170105021

Ayush Kumar Sinha-170105013

Akash Raj-170105019

**SCHOOL OF COMPUTING**

**DIT UNIVERSITY, DEHRADUN**

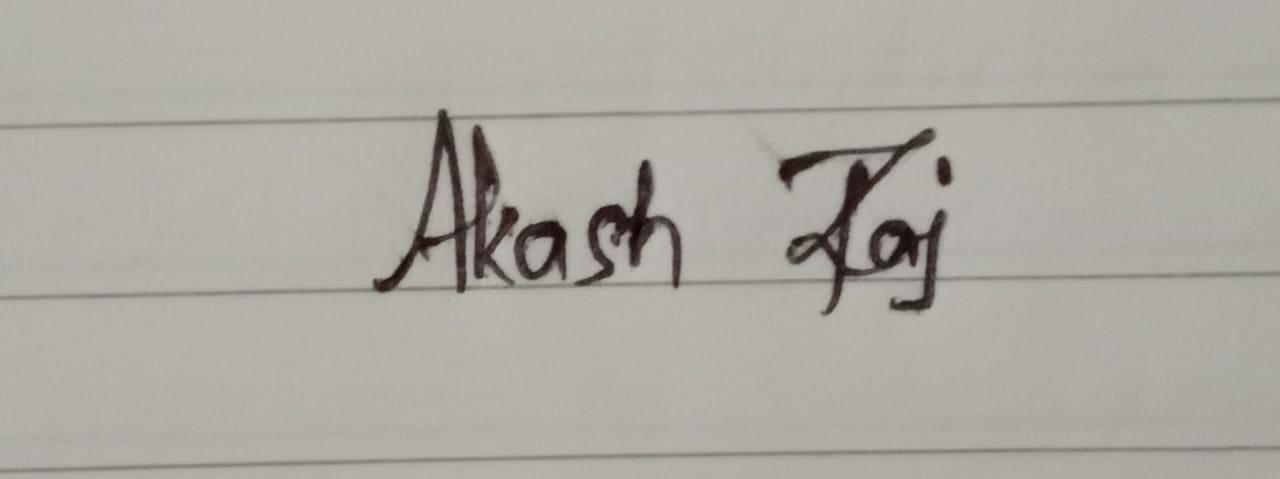
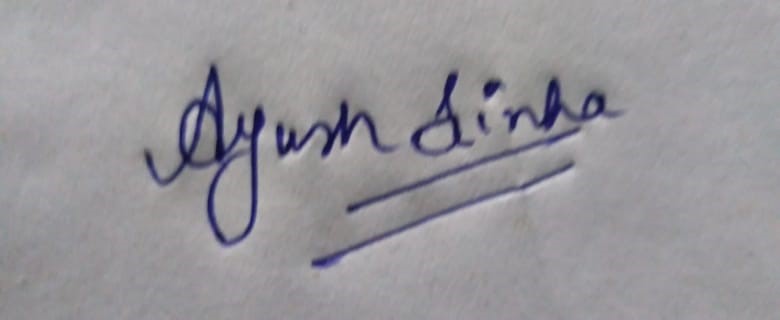
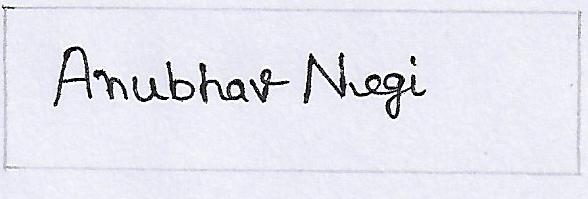
(State Private University through State Legislature Act No. 10 of 2013 of Uttarakhand and approved by UGC)

**Mussoorie Diversion Road, Dehradun, Uttarakhand - 248009, India.**

**2020**

**CANDIDATES DECLARATION**

I here by certify that the work, which is being presented in the Report, entitled **Image Classification using Deep Learning**, in partial fulfilment of the requirement for the award of the Degree of **Bachelor of Technology** and submitted to the DIT University is an authentic record of my work carried out during the period ***from 19-10-2019*** to ***01-11-2020*** under the guidance of **Dr. Mitali Srivastava.**



**Date: 01-11-2020 Signature of the Candidates**

**Signature of Guide**

**Certificate**

This is to certify that Project entitled “***Image Classification using Deep Learning***” in partial fulfilment of the requirement for the award of the degree of **Bachelor of Technology in Information Technology**, submitted to **DIT University, Dehradun, Uttarakhand, India**, is an authentic record of bonafide research work carried out by

Anubhav Negi Roll No 170105021

Ayush Kumar Sinha Roll No 170105013

Akash Raj Roll No 170105019

under my supervision/ guidance.

Signature of Guide Signature of HOD

Dr. Mitali Srivastava Dr. Amit Kumar Mishra

Assistant Professor Head of IT

School of Computing School of Computing

DIT University DIT University

**ACKNOWLEDGEMENT**

We would like to express our sincere gratitude to our guide Dr. Mitali Srivastava for her extensive support and guidance through entire last phase of the project. We would also like to thank other faculties who gave crucial advice through span of the project. Secondly, we would like to thank my seniors and people from community who were ready to help us every time.

**Name:** Anubhav Negi

**Roll No:** 170105021

**SAP ID:** 1000008615

**Name:** Ayush Kumar Sinha

**Roll No:** 170105013

**SAP ID:** 1000009251

**Name:** Akash Raj

**Roll No:** 170105019

**SAP ID:** 1000009227

**ABSTRACT**

The task of classifying an image is simple for human beings and we can recognise most perceivable objects in the world. This task becomes quite difficult for a computer which only sees the image as an array of numbers. If the object in the image changes its position, a human can still classify the object, but a computer treats it as a completely new image. Therefore, image classification becomes important for visual analysis of image data. With the availability of large benchmarking datasets such as ImageNet it has become possible to test and improve the classification models. There was need for models with large learning capacity, models such as a CNN came into existence with advancements in Deep Learning. Though being invented in late 90’s they gained importance in 2012 with the advent of AlexNet which reduced error rates to approximately half. Now CNN find their application in robotics, self-driving car, recommender systems, medical imaging etc. Our project is an attempt to test the generalizability of these models and learn some key insights which would help in applying image classification on some real-world application. The project started with experimentation on the cat-dog dataset from Kaggle. Several configurations were tested which showed that model’s limit was 83% accuracy on unseen data. Next, we experimented with the Food-101 dataset on Kaggle by using the VGGNet architecture and train the model from scratch which could not yield good results. Later we use transfer learning, using pre-trained weights of VGGNet which reached an accuracy of 98% on the first dataset and 58% on the Food-101. We carry out the process of fine tuning the model, determining the number of layers to train and the one’s that needed to be freeze. There is a certain difference in accuracy when images are padded with zeroes to preserve their original dimensions after convolution. This happens only when border features are helpful in identifying the images.

**TABLE OF CONTENT**

CHAPTER PAGE No.

Candidate’s Declaration 1

Certificate 2

Acknowledgement 3

Abstract 4

Chapter 1 –Introduction 9 1.1. Purpose 14

1.2. Motivation 15

1.3. Problem statement 15

1.4. Project Perspective 15

1.5. Objectives(Phase-1 to 3) 16

Chapter 2 –Literature Review 17

Chapter 3 –Description of Phase-2 21

Chapter 4–Findings of Phase-3 25

Chapter 5 –Conclusion &Future scope 31

Bibliography 32

**List of Figures**

FIGURE NAME PAGE

Perceptron 9

MLP 10

An example of 2-D convolution without kernel flipping 12

Max-Pooling Operation 13

Flattening of a 3 x 3 feature map into a long feature vector 13

Fully Connected Layers 14

Structure of a CNN 14

Alex Net 18

ZF Net 19

Architecture of VGGNet 20

Convnet model 23

Accuracy and Loss Graphs (Phase-2| Model-1) 23

Accuracy and Loss Graphs (Phase-2| Model-2) 24

Representation of zero padding 26

Phase-3 Model Architecture 27

Accuracy and Loss Graphs 29

Accuracy and Loss Graphs 29

**List of Tables**

Page

Different Convnet Configurations 26

Accuracy Results 27

**List of Abbreviations**

AI – Artificial Intelligence

CNN – Convolutional Neural Network

Convnet- Convolutional Neural Network

MLP – Multi Layer Perceptron

Adam – Adaptive Movement Algorithm

ReLu – Rectified Linear Units

ILSVRC- Image Large Scale Recognition Contest

VGG- Visual Geometry Group

LRN- Local Response Normalization

RGB- Red Green Blue

GPU- Graphics Processing Unit

**Chapter-1 Introduction**

Artificial Intelligence aims at developing algorithms which enable the machines to demonstrate human like intelligence. AI is already transforming industries such as healthcare, automobiles, robotics, retail etc. To implement AI systems Deep Learning is now becoming the most preferred choice. One of the reasons being that it is biologically inspired by the neural networks inside human brain makes it better at learning tasks. Other several reasons being the rapid rise in data generation over the internet, availability of high processing power, and advent of technologies such as Big Data and Cloud Computing.

Deep Learning uses Artificial Neural Networks which is a mathematical model of a biological neuron combined with huge amounts of training data and multiple processing layers. The term ‘deep’ refers to the large number of processing layers. Deep Learning is effectively applied in processing of image, video, sound, and text data. Unlike traditional machine learning where “feature extraction” is performed manually, a deep learning algorithm learns the features by itself.

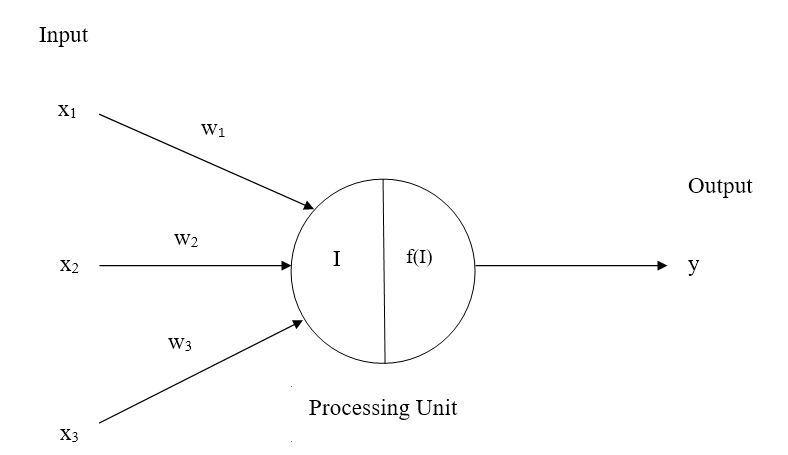


Fig 1. Perceptron

The perceptron is a mathematical model of the biological neuron. It acts as a linear classifier which performs binary classification. It consists of a single layer i.e. one input and one output layer. The single layer being referred here is the output layer. The input to a perceptron is a real valued vector. The output is always binary i.e. a 0 or 1. This model introduced weights and network learns the weight. With weights {w1, w2, w3, …, wn} being associated with each of the input. The neuron either fires (produces 1 as output) or not fire (produces 0 as output) based on the net input. The net input is given by

Net input,

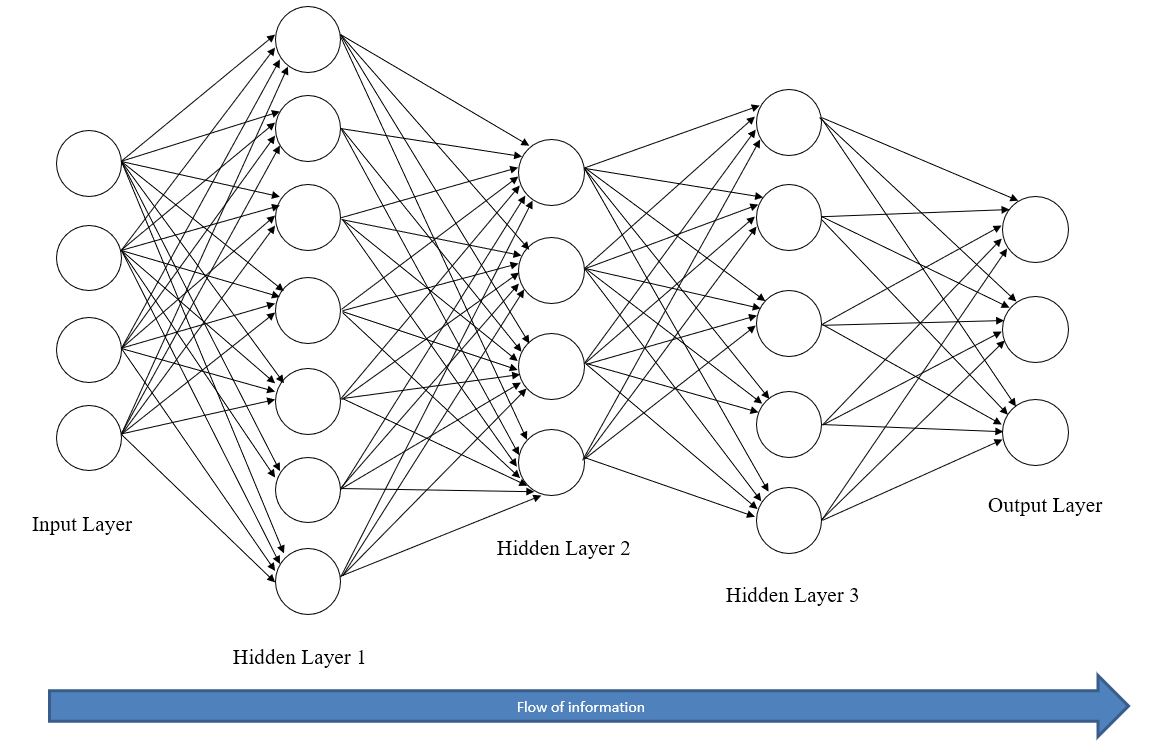


Fig 2. Multi Layer Perceptron

An improvement over the perceptron model is the multilayer perceptron model. A multi-layered perceptron is a multi-layered directed graph in which data flows from the first layer to the second layer, from the second layer to the third layer and so on. This type of network in which information flows from one direction through the intermediate layers is known as *feedforward network*. The first and last layers are input and the output layer. The intermediate layers are known as the hidden layers. The hidden layer and output layers use units with non-linear activation functions. This helps the MLP to solve non-linearly separable problems.

The network shown above is a four layered network consisting of three hidden layers and one output layer. The hidden layers could contain arbitrary number of nodes in each layer. Each node in one layer is connected to every node in the previous layer. Every edge in the network has some weights which are the parameters of the MLP that needs to be learnt. More the number of hidden layers, the more parameters we need to learn.

The nodes in the input layer is determined by the number of features and the number of output nodes depend on the number of outputs. MLP uses the backpropagation algorithm for learning representations.

One type of artificial neural network is Convolution Neural Network, which is a deep, feed-forward network commonly used for visual data analysis. These networks use little pre-processing as compared to other image classification algorithms. They were inspired from biological connectivity patterns of neurons in the animal visual cortex. Each neuron having its own *receptive field* which partially overlaps with other neurons.

CNN’s are made up of neurons which receive several inputs, performs weighted sum, apply and activation function over the sum and produce an output. They have very promising application in deep learning. Their use in computer vision has applications such as robotics and self-driving cars.

CNN’s main operation is a convolution, which is an element-wise matrix multiplication between the input volume and the filters. The filters are formed by the weights of the neuron present in the convolution layer. The first layer in a CNN can detect edges such as straight lines with higher layers detecting complex features as curves. The convolution operation produces different feature maps which are then combined by higher layers.

A CNN has a hierarchical structure which provides for fast feature extraction. The main aim is to extract features from the input image and convert it into an output vector holding class scores. A CNN consists of stacked convolution, pooling, and fully connected layers.

The various layers of a CNN are given below:

a) Convolutional Layer: This layer is used for feature extraction with the help of filters. Each filter learns or detects a certain feature in the input volume. The filters have depth dimension equal to the input volume’s depth dimension, as they work on every depth slice of the image. The initial convolution layers learn very simple features such as straight lines and edges. The output of convolution layer is feature maps having depth dimension equal to the number of filters. Each neuron in the convolution layer has its own receptive field which overlaps with that of other neurons i.e. these neurons are not fully connected but have sparse connections which reduces the dimensionality.

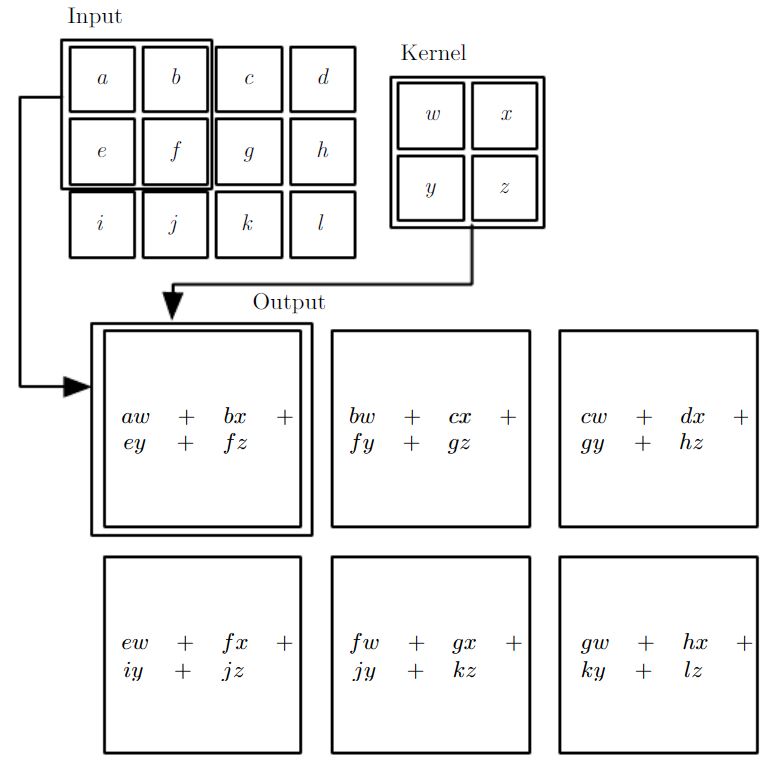


Fig 3. An example of 2-D convolution without kernel-flipping.

b) Pooling Layer: The pooling operation reduces the dimensions of the feature maps. It uses a receptive field (say 2 x 2) with certain stride (number of pixels to move ahead) to perform some computation over the field to reduce the pixel values to a single value. There are many types of pooling such as sum pooling, average pooling, and max pooling. The recent CNN architectures nowadays use max pooling. The idea behind max pooling is that, if a certain feature is detectable in a certain receptive field then taking only the maximum among those values would also make that feature detectable.

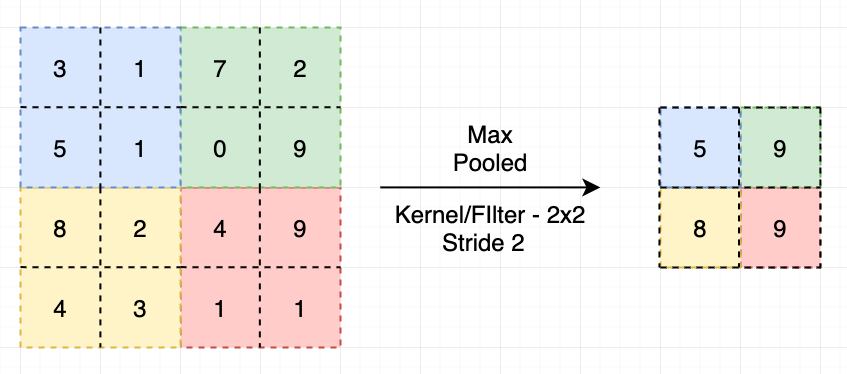


Fig 4. Max-Pooling Operation

c) Flattening Layer: The multidimensional output from the convolution + pooling layer cannot be fed to the fully connected layer. The fully connected layer expects a one-dimensional output, so the feature maps are flattened to a long one-dimensional feature vector. The values are flattened in a row wise fashion.

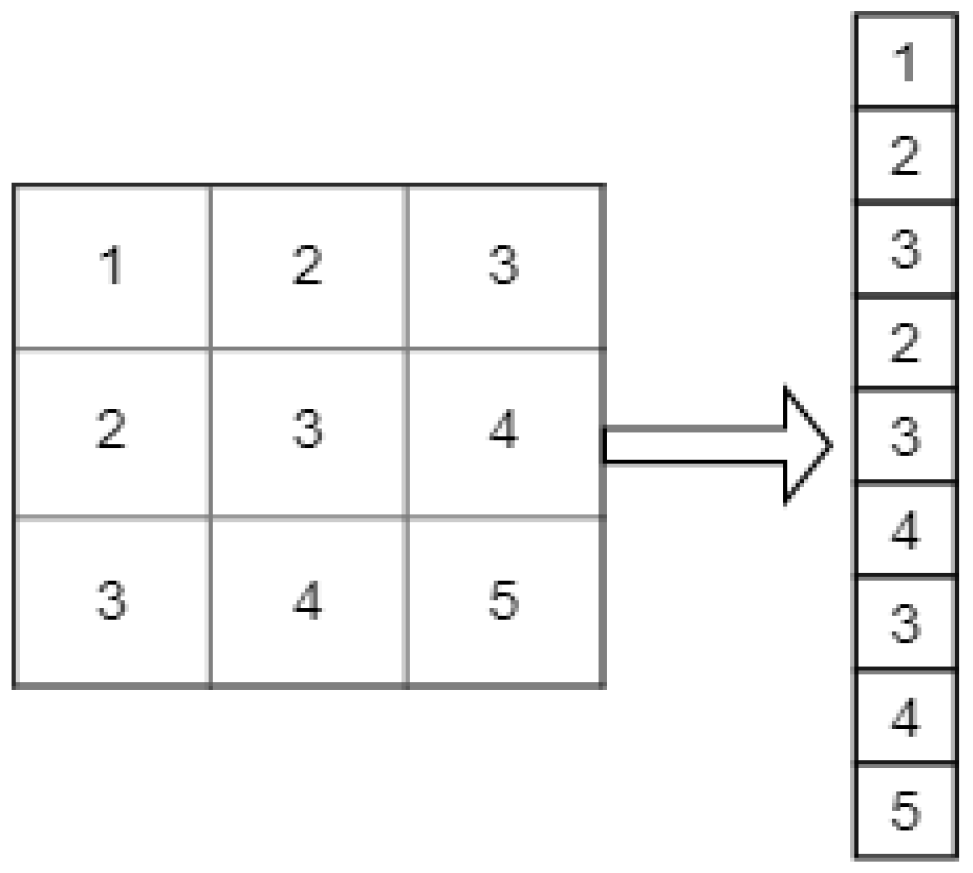


Fig 5. Flattening a 3 x 3 Feature Map into a long feature vector

d) Fully Connected Layer: This layer is MLP containing the hidden layers and the output layer. There is no restriction on the number of hidden layers and hidden neurons contained in them. The output layer contains neurons equivalent to the number of classes. These fully connected layers are responsible for combining individual features and adjusting the weights via backpropagation. The output layer probability scores for each class which is then passes through an activation function such as softmax which reduces the scores so that they sum to one.

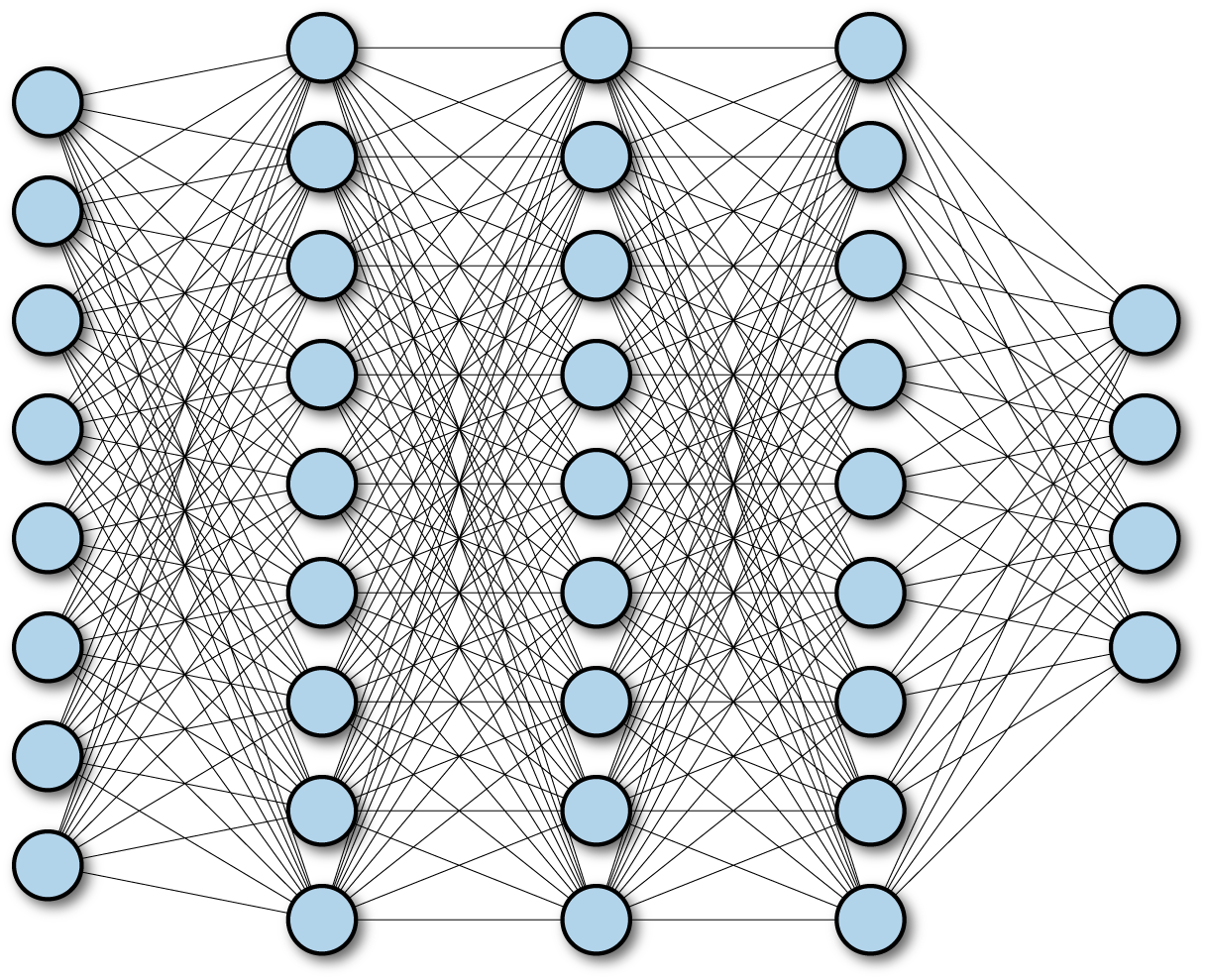


Fig 6. Fully Connected Layers

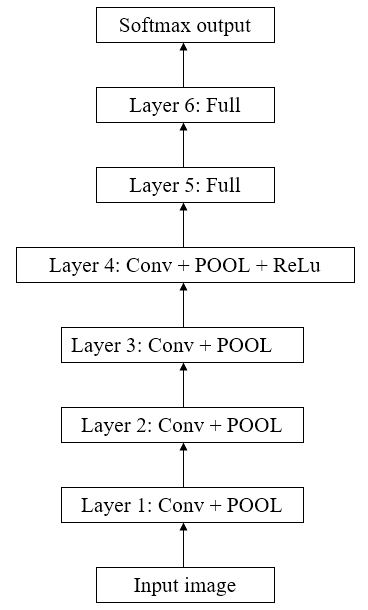


Fig 7. Structure of a CNN

**1.1 Purpose**

The purpose of undertaking this project is to learn and develop CNN’s which have been proven effective in computer vision tasks such as image classification. With a lot of visual data being produced and flowing over the internet, development of large datasets has become possible to test model’s performance over real world data. Although CNN’s being effective requires a lot of experimentation to make them generalise over the datasets by increasing its depth or width. Therefore, in this project we had tried to experiment with the network’s architecture, regularization, and tunings it hyperparameters. A good CNN model should be able to generalise over the given dataset and to other datasets as well by fine tuning them. Also, it is not always possible to develop classification models from scratch given shortage of computing resources, less visual data, and time. So, one could use previous architectures which worked can be used by application of transfer learning and fine tuning for generalizability. With this purpose we had proceeded in the project and tried to develop some insight into the working of the CNN’s.

**1.2 Motivation**

Image classification is one of the most important problems in visual recognition. CNN’s have become an important tool for such tasks. It was till 2012 these algorithms gained significant importance though they were lying around from 1990’s. Our motivation was to develop a computer program which could recognise some images and classify them correctly. The task is quite simple for human beings due to highly advanced visual system but for a computer it sees images just as arrays of numbers between 0 to 255. We came to know through our project guide that such task can be implemented using Deep Learning models. One such model is a Convolutional Neural Network which can be applied to such tasks. There have been rise in popularity of such models with availability of benchmarking datasets, use of these models for acceleration in medical imaging, advent of powerful hardware and betterment of algorithms. Also, the field of computer vision provide numerous varieties of tasks to carry on research activities.

**1.3 Problem Statement**

To classify an image based on its visual content into one of the pre-defined classes containing a large set of sample images. For example, we input an image and output is class label say a horse or a truck.

**1.4 Project Perspective**

The project aims at learning and implementation of CNN’s. CNN’s are a great tool for image and video analysis and are being used at Facebook for tagging purposes, at Google etc. The reason behind their renewed interest are Large datasets, Powerful GPU’s, and better regularization techniques. With the knowledge of these networks one can build powerful image classifiers which can be used in combination with medical imaging and produce results faster.

**1.5 Objectives (Phase-1 to 3)**

**Phase-1**

* Thinking a project idea.
* Get to know about the field of interest
* Learning Fundamentals of Deep Learning
* Introduction to Convnets

**Phase-2**

* Getting a dataset.
* Exploring and Implementing model using Keras Library.
* Training the model and assessing its performance.
* Adjusting the model.
* Retraining the model and drawing some inference.

**Phase-3**

* Trying out different configurations of the previous model.
* Applying regularization to address overfitting.
* Use of classic VGG16 net over Food-101 dataset.
* Application of Transfer Learning over previous two datasets.

**Chapter-2 Literature Review**

The performance of machine learning methods improves by collecting large datasets and making use of different regularization techniques. Though these techniques help reduce overfitting, but the task of object recognition is quite complex and cannot be even fully achieved with dataset as large as ImageNet. Only having a large dataset is not sufficient, we need to have model with large learning capacity. One such type of model is a Convolutional Neural Network (CNN) whose learning capacity can be increased or decreased by manipulating the network’s layers width or the network’s depth. Although CNN algorithms were invented in the early 1990’s but were restricted by computational power constraints at that time. However, in 2012, when Alex krizhevsky and his team proposed a deep CNN popularly known as AlexNet which was able to outperform previous records in the ImageNet Large Scale Visual Recognition Contest (ILSVRC) 2012. The dataset used was a subset of ImageNet having 1000 categories with 1000 images per category. This renewed the interest in CNN’s and was much credited to the availability of powerful GPU’s and better regularization techniques.

AlexNet consisted of five convolutional layers and three fully connected layers. Removing any of the layers resulted in reduced accuracy and therefore this depth was important. The model’s input were coloured images resized to have a dimension of 256 x 256. Instead of using traditional activation functions such as sigmoid and tanh(x) to introduce non-linearity, Rectified Linear Units were used. The neurons which implemented the ReLu activation function were termed as Rectified Linear Units which made learning faster. Performance of large models improves on large dataset when learning happens to be fast. The net also used Local Response Normalization at some layers which was termed as brightness optimization. The use of overlapping pooling with a stride of two and window dimension of 3 x 3 made overfitting difficult for the model. The architecture had total 8 weight layers, the last three fully connected whose output was passed to a 1000-way softmax. LRN was implemented after first and second convolution layer followed by max pooling, which was also implemented after fifth convolutional layer. The ReLu non-linearity was applied to all the neurons. The first conv layers contained 96 kernels of dimension 11 x 11 and convolved with a stride of four. The second conv layer used 256 kernels of dimension 5 x 5. The third, fourth, fifth conv layers had 384, 384, 256 kernels respectively, followed two fully connected layers having 4096 neurons each.

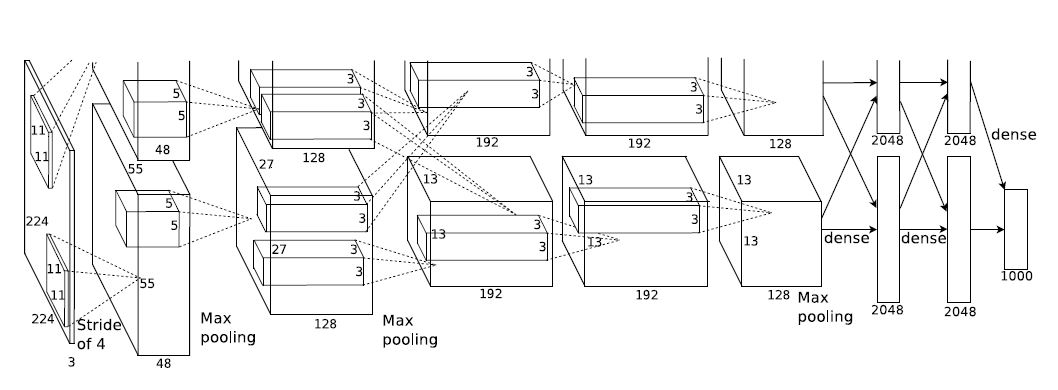


Fig 8. AlexNet

To fight overfitting AlexNet used augmentation and dropout. It uses two data augmentation techniques, first one just horizontally flips the image while the second one alters the pixel intensities using Principal Component Analysis. While dropout was to shutdown neurons with probability of (0.5). These neurons did not contribute during the forward and backward passes. So, at each iteration the network exhibited different architecture which shared the same weights. Dropout forces the neurons to learn features which were helpful when combined with other neurons. The network used stochastic gradient descent algorithm during training with a batch size of 128 and an initial learning rate of 0.01. The result was network with a top-5 error rate of 15.3% on ILSVRC-2012, where top-5 error corresponds to the number of images where the actual label was not among the top-5 probabilities generated by the model. This research paved a way for CNN’s application in computer vision.

While AlexNet had demonstrated its performance on large datasets but still there was no clear understanding as how these models performed so well or can improvements be made. There was a very little insight into the internal working of these models, so their development was only limited to trial and error. To overcome such problems Zieler and Fergus came up with their idea of a deconvolutional network. This network could perform visualization when attached to each of the conv layers of a CNN. Unlike a CNN which maps pixels to features, a deconvnet maps pixel to features. A deconvnet accomplish this task by unpooling the feature maps, rectifying using ReLu and then filters them. The filters are same as in the conv layers but just flipped both vertically and horizontally. It was an attempt to visualize the type of features which excited a feature map. Through their experimentation results it was demonstrated that the first conv layer detects very low-level features such as edges or colour. However, the later layers detect complex features such as curved edges. This kind of visualization was helpful in rectifying some problems in AlexNet. This rectified network came to known as ZF Net which was AlexNet with just some hyperparameter optimisation. ZF Net proposed use of 7 x 7 filters in the conv layers with a stride of two which contrasted with 11 x 11 filters with a stride of four in AlexNet. The reason being that larger filter sizes were skipping a lot of information and a smaller size improved the classification performance. It achieved a top-5 error rate of 14.8% in ILSVRC-2013.

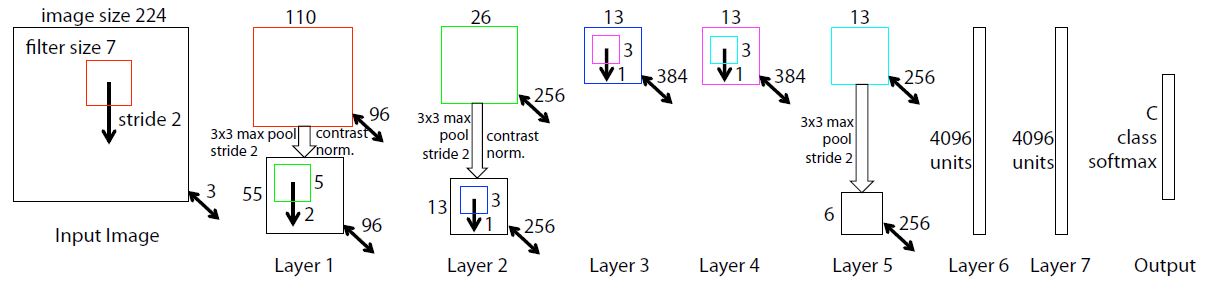


Fig 9. ZF Net

Another significant contribution to the development of deep CNN’s was made by the Visual Geometry Group, Oxford. As convnets were achieving high accuracy several attempts were being made to improve over AlexNet. VGG tried to experiment with the depth of the network which was a contributing factor in increasing the accuracy. The net architecture takes RGB images of dimension 224 x 224 and applying some pre-processing. The pre-processing subtracted the mean of the RGB values taken from each pixel, calculated over the training set. It used even more smaller receptive field of dimensions 3 x 3 with a stride of one. The convolution step involved padding the images such that the resultant conv layer output retains original dimensions. Max pooling was performed with pooling window of dimension 2 x 2 with a stride of two, which simply took a max out of four-pixel values. It uses the same fully connected layers as seen in AlexNet, each followed by ReLu. The VGG net does not incorporate LRN as it did not seem to benefit but instead increases space and time complexity. Initial conv layer starts with 64 number of filters and then increasing by a factor of two after each pooling layer. One important observation is that VGG net uses two stacked 3 x 3 conv layers which they stated to be in equivalence with a 5 x 5 convolution. Three such layers stacked back to back were equivalent to a 7 x 7 convolution. It gave an advantage of using three ReLu’s rather than just one. During training mini batch gradient descent was used with a batch size of 256. Learning rate was initially set to 0.01 and reduced with no increase in validation accuracy by a factor of 10. Despite its larger size model took less time to converge due to the use of smaller filter sizes and regularization. The model was able to achieve a top-5 error rate of 7.3% in ILSVRC-2014. With this the VGG net showed that traditional convolutional neural net with increased depth and smaller filter sizes can perform well on classification and localization tasks.

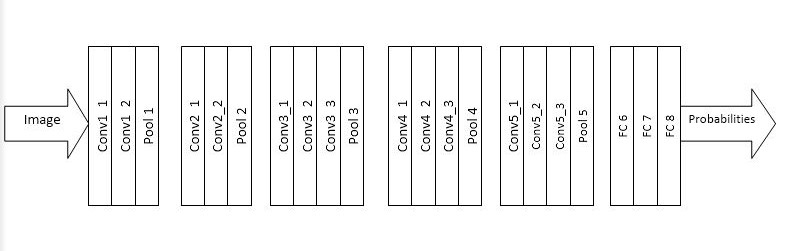


Fig 10. Architecture of VGGNet

The image classification and object detection capabilities have been increasing year on year with development of new ideas and improvement in network architectures. Most important aspect is the efficiency of algorithms i.e. the space and time complexity which matters when these are put to real world usage. The most straight forward way of increasing the performance of Deep Neural Networks is by increasing their size i.e. the number of layers (depth) and its width (number of units per layer). As increasing the depth seems advantageous, it has certain drawbacks. The increased depth means that huge number of parameters which makes the network vulnerable to overfitting when labelled samples in the training set are less. Also, it becomes computationally expensive, so computing resources need be efficiently distributed. These drawbacks could be overcome by replacing the fully connected layers and introducing sparsity into the network architecture. One such architecture that was aimed at overcoming these drawbacks is the GoogLeNet-2014 which used 9 inception modules. It was able to achieve an error rate of 6.67% with a depth of 22 layers and being very close to human level performance. It does use several but very small convolutions ranging between 1 x 1 to 5 x 5 in parallel.Their idea was to being able to cover larger area while keeping fine resolution on the images.

**Chapter-3 Description of Phase-2**

In Phase-2 we implemented a very simple Convolutional Neural Network to classify some images. The dataset we used first was a subset of the cat-vs-dog dataset from Kaggle. It consisted of 8000 samples for training and 2000 samples for validation/testing for each category. However at last we would also discuss our experimentation with the dataset consisting of 25K images of cats and dogs, 12.5K for each category .The model was implemented using Keras library which is a quite popular library used for implementing deep learning models. The library can be used with any of the three backends- TensorFlow, Theano and CNTK.

The implementation needed to learn and explore about different functions, classes and model structures built into Keras. We used Keras on top of TensorFlow which is a lower level deep learning framework, as Keras being a high-level library saves beginners from dealing directly with tensors. Now coming onto implementation, firstly we discuss the various modules and classes used inside them.

Modules and their classes used-

1. The models module

This module consists of class Sequential which we use to instantiate a Sequential model object which helps us to linearly stack our layers. As we know CNN’s are mostly linear models in which layers are stacked on top of other in a predefined hierarchy.

2. The image module

The Keras library has modules which allow us to pre-process our data. One such module is image module. This module contains a class known as ImageDataGenerator which on instantiation help generate batches of augmented image data. Here our image is processed into a NumPy array, which then can be fed to our network. Data augmentation helps to artificially expand our dataset by introducing slight variations in each image. The images have quite less noticeable changes to humans but appears as a completely new image to computers.

The image batches are generated by using the function flow\_from\_directory() found inside ImageDataGenerator class. This function requires our dataset directory in a format as shown. It takes the directory path, target size to which images will be resized, batch size and a class mode. The function generates DirectoryIterator class objects of the (x, y) which are nothing but tuples. Here x is a NumPy array containing 32 images and y is a NumPy array containing class label. The function automatically encodes our label i.e. cat and dog now correspond to a 0 and 1 through value specified in the class mode. All the images are resized into 64 x 64 x 3 dimension and their pixel values rescales to be between [0, 1].

The ImageDataGenerator is passed some parameters such as rescale, shear\_range, zoom\_range and horizontal\_flip. We used the values recommended by the Keras library to effectively augment the images. Here rescaling is required as pixel values are between the inclusive range 0-255 and such values are quite large to fed to the network, so we multiply each pixel value by a factor of 1/255.

3. The layers module

The layers module contains the different classes required to instantiate a layer and add that to our model. The different layers used are Conv2D, MaxPooling2D, Dropout, Flatten and Dense.

4. The pyplot module

The pyplot module of matplotlib is used to visualize the results of model training. We plot the graphs, accuracy vs epochs and loss vs epochs.

Our first model was a 3 layers network consisting of a convolutional layer followed immediately after a max pooling layer. We used only 32 feature detectors in the convolutional layer which is starting point for most CNN architectures. The max pooling layer used 2 x 2 pooling window with a stride of 2 which has now become a default convention as used in state-of-the art models. The two Dense or Hidden layers contained 128 neurons each. Both the convolutional and dense layers were followed by a ReLu activation. Although the number of filters in the convolutional layers and hidden neurons are fixed but these hyperparameters can be adjusted to increase model performance. The model architecture is given below.

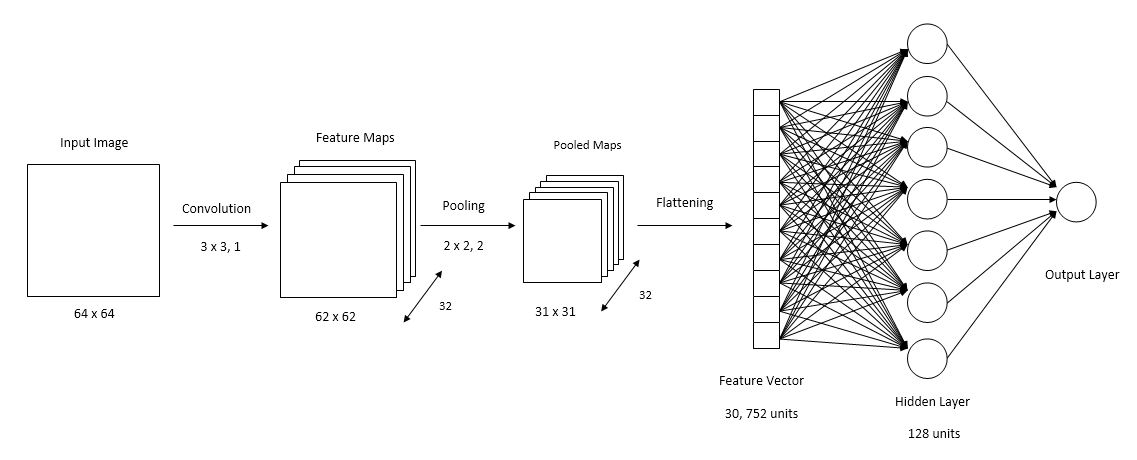


Fig 11. Convnet model

Our model was trained on 8K images for 25 epochs with 250 iterations per epoch. The model at epoch was receiving augmented images, each epoch model was fed with a randomly transformed image. We used an output layers containing just one neuron with sigmoid activation. The optimizer and loss function used were ‘adam’ and ‘binary cross entropy’. The training results are as given below.

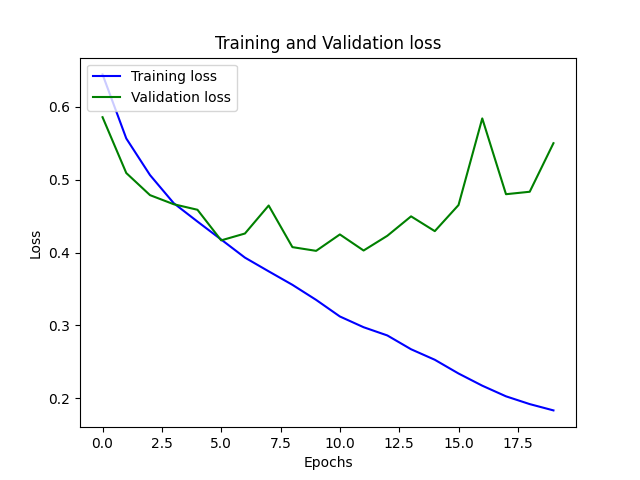
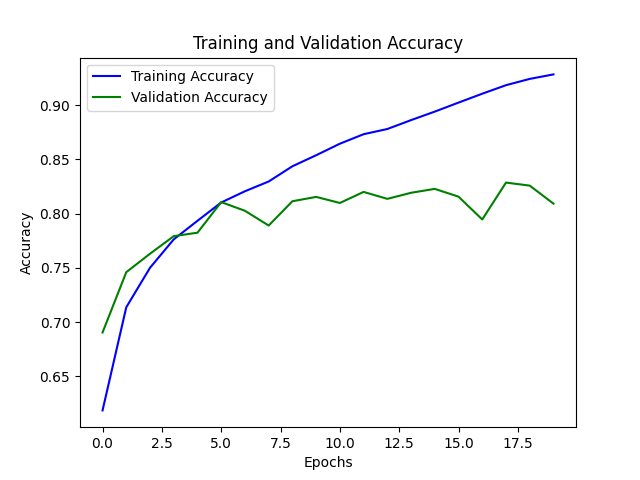


Fig 12. Accuracy and Loss Graphs (Phase-2|Model-1)

As it is clearly visible from both the graphs that model starts to overfit at around fifth epoch. The validation accuracy measured over 2K samples starts to flatten and the validation loss starts rising in the graph. However, in contrast to the validation accuracy and loss, the training accuracy kept increasing and training loss kept decreasing. This can be understood as model learned significant features at first then model started learning features that would not help in generalize over unseen data. To reduce overfitting, we can either use a larger dataset or apply some regularization techniques such as dropout.

We now increase the depth of the model by adding another convolution and pooling layer. The added layers have parameters identical to the convolution and pooling layers above. Each pooling and hidden layer is followed by a dropout with dropout rate being set to (0.2). This helps reduce overfitting and model now gives training accuracy as 81.61% and validation accuracy as 79.60%.

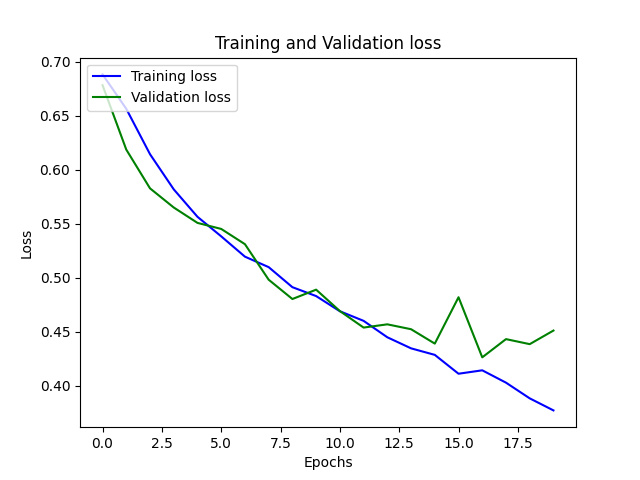
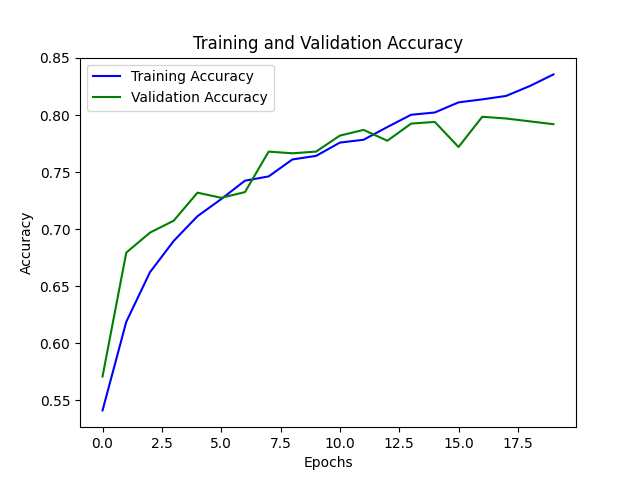


Fig 13. Accuracy and Loss Graphs (Phase-2|Model-2)

As it is evident from the graph that the model now starts to overfit after the 15th epoch as training accuracy is increasing but validation accuracy starts to dip. Also, the validation loss starts to shoot up at the 15th epoch while training loss is still decreasing. So, the real-world performance of the model would be around 79.60% when tested on unseen data.

**Chapter-4 Findings of Phase-3**

The model built during the second phase of our project was able to generalise with an accuracy of 79.60%. Therefore, we went through some research papers on CNN architectures, articles, and video resources. We found that better accuracy can be achieved by experimenting with the depth of the model i.e. the number of convolution and hidden layers or by tweaking the hyperparameters of the model such as the number of filters in the convolutional layer, size of the filters (filter dimensions), dropout rates, number of hidden layers and the hidden neurons contained in them. Though we found that mostly the number of filters gets doubled at each subsequent convolutional layer and mostly taken as powers of 2 such as 32, 64, 128. Also, as we increase the filters the dimension of the filters also increases such as going from a size of 3 x 3 to a size of 5 x 5. The reason being that larger filters have a greater receptive field and can capture a lot more information such as smaller filters learn local features of an image and later bigger filters combine them to form global features. Also, we came to know about valid and same convolution, where valid convolution decreases the input size i.e. the output dimension is not equal to the input dimension. The two problems that valid convolution can cause are, first if the neural network has a greater depth then decreasing size of input to each layer will reduce the amount of information quite fast for the upper layers to work effectively, secondly if features at the border of the image are beneficial in classification would be skipped. Whereas a same convolution means that input dimension is preserved by the help of padding the image borders with zeroes. This helps in retaining the border features and the amount of information passed to the upper layers of the network. The formula’s used for calculating dimension of the feature maps produced and determining the amount of padding are and respectively where n is input dimension, p is the amount of padding, f is the filter dimension, and s being the stride. Usually the convention in computer vision is to take odd filter sizes such as 1 x 1, 3 x 3, and so on. The reason being that odd dimension filters tend to have a central pixel which is equidistant from top/bottom and left/right and even filters will cause asymmetric padding as is clear from the above formula. However, we found some relevant intel regarding convolution but there was no as such rules or principles to figure out the number of hidden layers and the neurons contained inside of them. These layer’s configuration seems to be pure experimental and good results can be obtained by trial and error only.

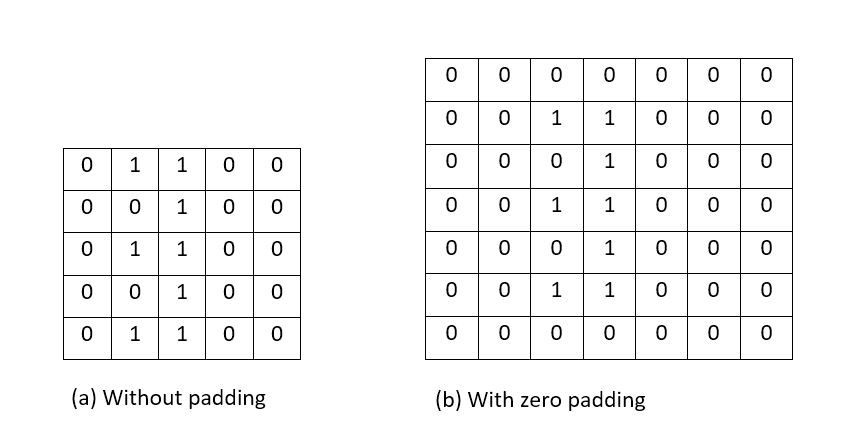


Fig 14. Representation of Zero padding

Using all the information gathered above we tried different versions of our model on the cat/dog dataset. Each version either had increased layers or different hyperparameter configuration. The configurations are given below:

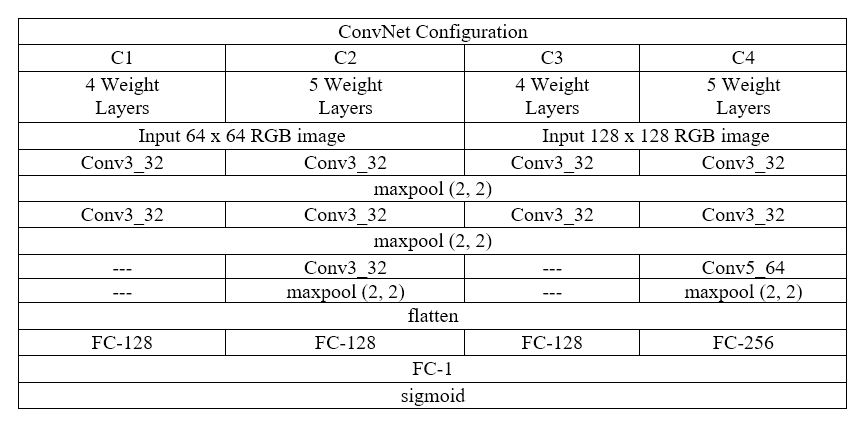


Table 1. Different Convnet Configurations

In the table shown above are the different convnet configurations which we tested to improve our accuracy. We ran all these models for 20 epochs and dropout if used was set to (0.2). The accuracy results for different runs of the model are given below.

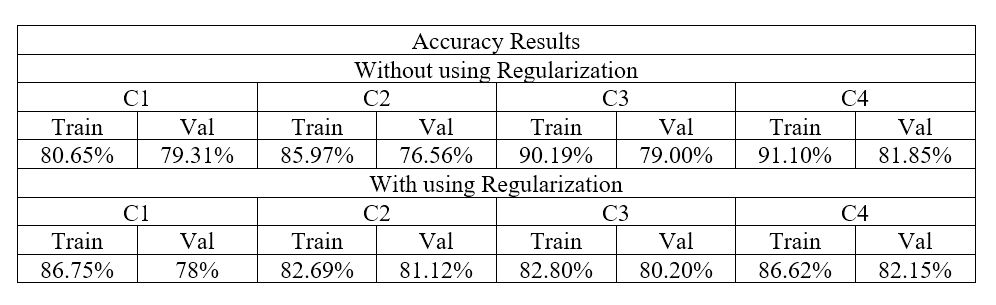
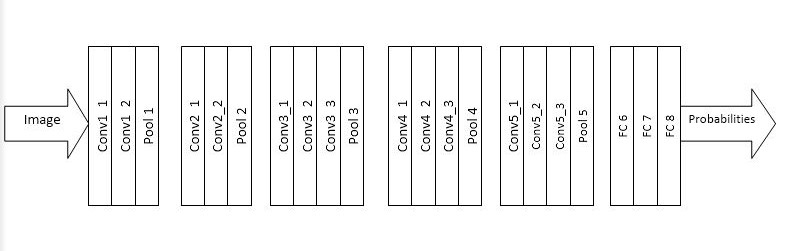


Table 2. Accuracy Results

The model configuration denoted by C4 had better validation accuracy than rest of the configurations trained. We also did train more configurations having increased amount of convolution and hidden layers, but it was not beneficial, and it only contributed to model overfitting. With all these experimentations performed we concluded that 82% was the model’s limit to generalize and it would not be able to squeeze out more accuracy with current architecture and the given dataset.

We now try to classify images with a much bigger network architecture that of VGGNet having 16 weight layers. This model was originally used to perform classification on the subset of ImageNet dataset that consisted of 1000 categories each having 1000 training samples.

Fig 15. Phase-3 Model Architecture



We use this network architecture to perform classification over the Food-101 dataset that is available on Kaggle. This dataset consists of 101 categories of labelled food images with each category having 1000 samples. At first, we try to implement the model from scratch and iterate it over the dataset. Minimal data augmentation measures were applied, and model was trained for 20 epochs. The train-validation split was 80-20% i.e. 80,800 images for training and 20,200 for validation. The images were resized to dimension 224 x 224 x 3 during pre-processing because that is the image size expected by this model. The accuracy did not rose above 1% and we then continued the training from 21st epoch to 40th epoch. The model ran for approximate five to six hours but still the accuracy was not increasing. It was found that the model had 138 million trainable parameters which were not easy to train and could take days. Also we later found in the paper that VGGNet applies a certain kind of pre-processing over the images, it calculates the mean pixel values for the red, green and the blue channels of training images and then subtracts this value from each pixel in the images.

Since it was getting tougher to train the whole model from scratch, transfer learning could help us achieve better results. The technique of transfer learning involves using a pre-trained model or importing pre-trained weights into the model. We can add our output layer that suits our needs and training it. The rest layers are left freeze i.e. no weight updating takes place in those layers. The number of trained layers to incorporate or freezing the layers depends on the problem’s requirements. Finding the right numbers depends on experimentation with different setting and this process is known as fine tuning the model.

The pre-trained VGG16 model is available with keras applications is a functional model which can be used with weights set as imagenet. The layers from these pretrained model were copied into our sequential model, leaving out the last output layer. We first tried it over the cat-dog dataset which we were testing earlier in this phase. The pre-trained model showed very good results as it was already trained over the imagenet dataset that has a category for cat and dog images. The model gave an accuracy of 99.1% and 98.45% on training and validation sets. The results are shown in the graph below.

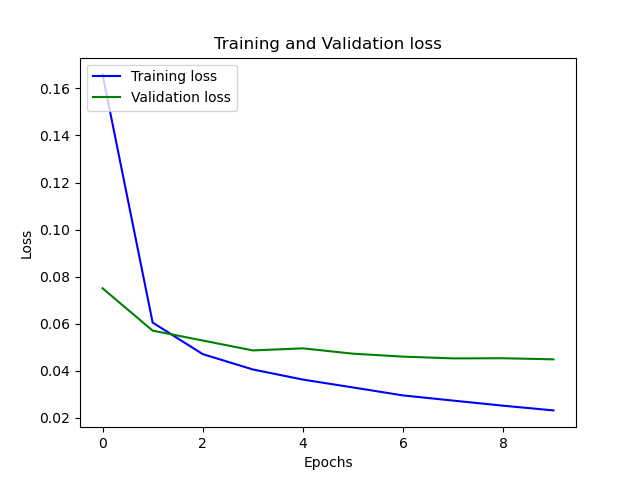
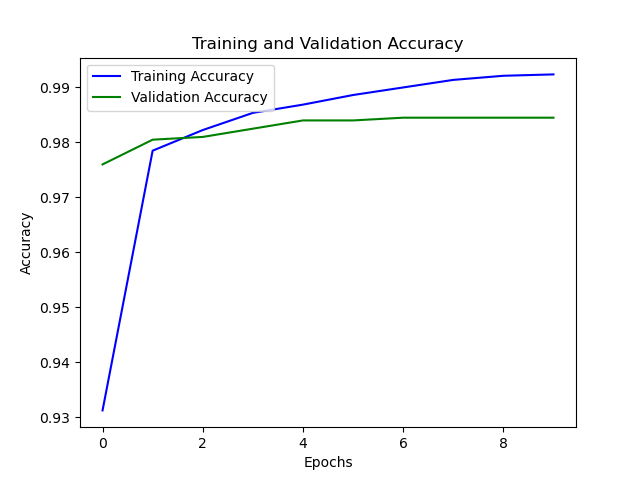


Fig 16. Accuracy and Loss Graphs

The same model configuration was used on the Food-101 dataset which gave an accuracy of 88.20% on training set and 47.49 on the validation set. The resulted graphs are shown below.

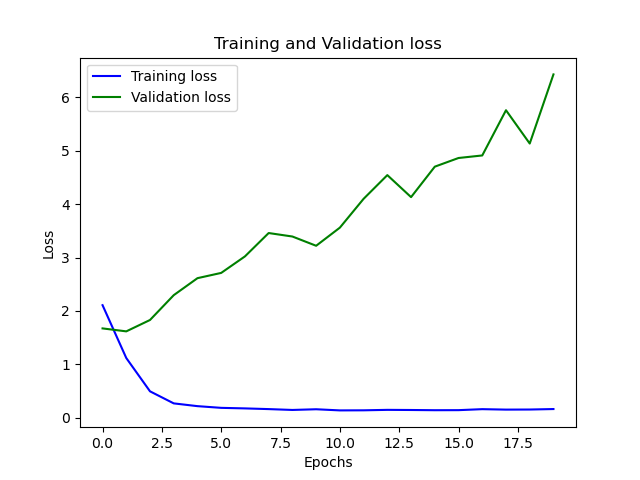
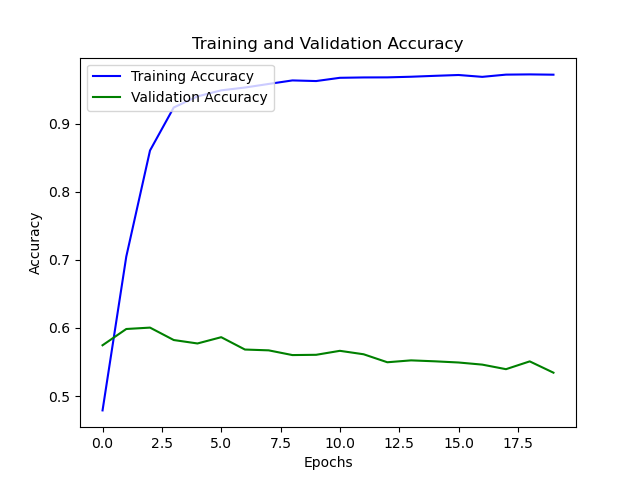


Fig 17. Accuracy and Loss Graphs

As we can see the training accuracy is increasing while the validation accuracy increases at starts but quickly flattens up around 2nd or 3rd epoch. Also, the training loss is decreasing but the validation loss is rising, the model is certainly overfitting. Since we trained only the output layer of the model, leaving rest of the layer weights intact can be fine-tuned for this dataset. We now unfreeze the two hidden layers the result drops down with 74.95% training accuracy and 35.55% validation accuracy. Decreasing the learning rate to 0.0001 does benefit a little with 98.22% training accuracy and 52.09% validation accuracy. While increasing the number of trainable layers and keeping a steady learning rate of 0.0001 the validation accuracy hovers around 52-58%.

**Chapter-5 Conclusion and Future Scope**

The CNN requires much more insight into its working. We were unable to find the right set of hyperparameters for our cat-dog classifier. There is no as such finding to find the right set of filters for each convolution layer or to find the hidden layers. This leads to random experimentation which could take a lot of time if you are low on computing resources. One should have to look towards transfer learning when dataset sizes are very small. Though this technique is helpful but requires a lot of fine tuning if model is seeing completely new set of images which do not resemble with images of its previous training set.

Limitations of image classification is only to predict the object. To overcome this, we use object localisation. In object localisation we predict object’s classification and its attributes i.e. what type of object it is and after identifying the object based on the training dataset make a boundary enclosing the object. This boundary varies with objects position and size in the given image. This helps in identifying multiple objects in the image and draw a boundary against them. This technique is totally dependent on the softmax function. It can also predict the background of the image.

**Bibliography**

1. ImageNet classification with Deep Convolutional Neural Networks, Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton.
2. Visualizing and Understanding Convolutional Networks, Matthew D. Zeiler, Rob Fergus.
3. Very Deep Convolutional Networks for Large-Scale Image Recognition, Karen Simonyan, Andrew Zisserman.
4. Deep Learning using Rectified Linear Units, Abien Fred M. Agarap.
5. Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition, Dominik Scherer, Andreas Muller, Sven Behnke.
6. Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Geoffrey Hinton.
7. Learning Representations by Back Propagating Errors, David E. Rumelhart, Geoffrey Hinton, Ronald J. Williams.
8. Gradient Based Learning Applied to Document Recognition, Yann LeCun.
9. Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville.
10. A Survey of the Recent Architectures of Deep Convolutional Neural Networks by Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi.
11. From ImageNet to Image Classification: Contextualizing Progress on Benchmarks by Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Andrew Ilyas, Aleksander Madry.