CSE 676:DEEP LEARNING

JYOTI SINHA
Department of Computer Science
University at Buffalo
Buffalo, New York, 14221
Email:- jsinha@buffalo.edu

Abstract

In this project, Implementation, Details and Comparison is done between three of the most common CNN architectures that are VGG16, Resnet18 and InceptionV2. Tried to change the parameters and activation functions of the CNN architectures to maximize the accuracy of the network. Table of accuracy, Precision and Recall is created for comparison between various CNN architectures on SGD and Adam optimizer to get the more detailed and precise conclusion. Plot of accuracy and Loss vs Epoch is also performed to get better visualization of the CNN architecture.

1 Introduction

This project deals with the details, implementation and comparison between three of the most common and powerful CNN architectures. The most common CNN architectures that are used in this project are VGGNet, Inception and Resnet. These CNN laid the foundation of today's Computer Vision achievements which are achieved using Deep Learning. For the implementation part, VGG16, Resnet18 and InceptionV2 versions of different CNN are used to compare the accuracy of the three CNN models. An evaluation of the comparison is done in the last section. The CNN models are evaluated through a multiclass statistical analysis based on accuracy, precision and recall. This comparison would be useful in helping people to conclude which CNN architectures they should use as per their requirement to get more success.

2 Implementation of the architecture

2.1 VGG16

2.1 VGGNet16 (Original Implementation)

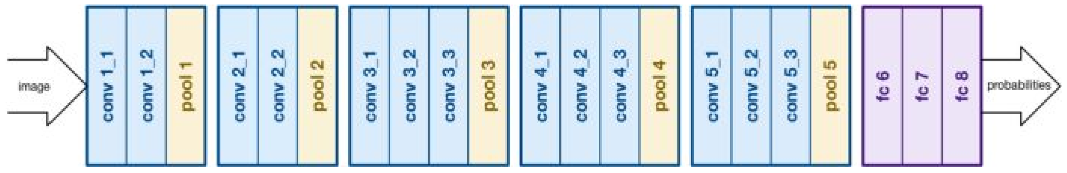


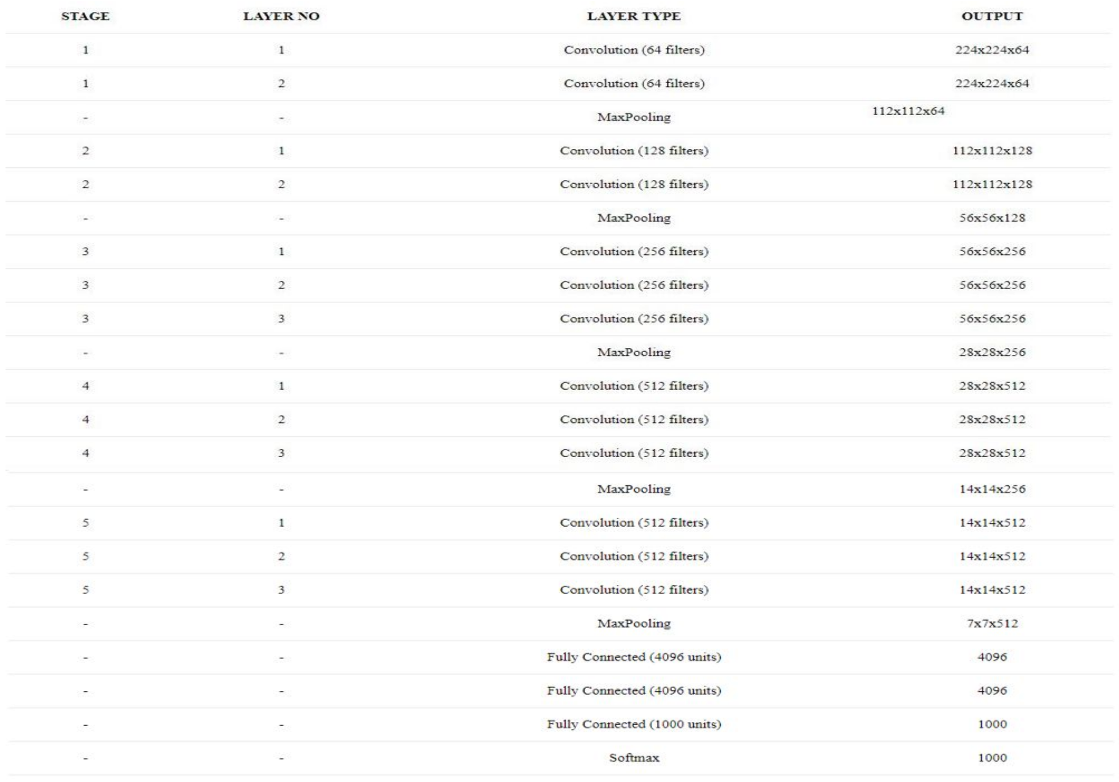
Fig a:- Layer details of VGG16

In VGG16 architecture two rules of thumb needs to follow :-

- Each Convolutional layer has the same configuration of kernel size = 3x3, stride = 1x1 and padding = ‘same’. The only thing which differs is the number of filters.
- The Max Pooling layer has also the same configuration of window size = 2x2 and stride = 2x2. In this way, the size of the image gets half at every Pooling layer.

The VGG16 was initially implemented on ImageNet dataset.

The size of each RGB image was 224x224 pixels. So, the input size = 224x224x3.



STAGE	LAYER NO	LAYER TYPE	OUTPUT
1	1	Convolution (64 filters)	224x224x64
1	2	Convolution (64 filters)	224x224x64
-	-	MaxPooling	112x112x64
2	1	Convolution (128 filters)	112x112x128
2	2	Convolution (128 filters)	112x112x128
-	-	MaxPooling	56x56x128
3	1	Convolution (256 filters)	56x56x256
3	2	Convolution (256 filters)	56x56x256
3	3	Convolution (256 filters)	56x56x256
-	-	MaxPooling	28x28x256
4	1	Convolution (512 filters)	28x28x512
4	2	Convolution (512 filters)	28x28x512
4	3	Convolution (512 filters)	28x28x512
-	-	MaxPooling	14x14x256
5	1	Convolution (512 filters)	14x14x512
5	2	Convolution (512 filters)	14x14x512
5	3	Convolution (512 filters)	14x14x512
-	-	MaxPooling	7x7x512
-	-	Fully Connected (4096 units)	4096
-	-	Fully Connected (4096 units)	4096
-	-	Fully Connected (1000 units)	1000
-	-	Softmax	1000

Fig b:- Layer Details with their filter and output size

2.1.1 Implementation(In this Project) different from the Original Implementation

In this project, there is no significant difference in the layer details of VGG16 compared to the original Implementation of VGG16 performed by the runner up of 2014 imagenet challenge. In this project, VGG16 is implemented on the CIFAR100 dataset.

CIFAR100 Dataset Details:-

CIFAR data sets are the well known data sets in computer vision tasks created by Geoffrey Hinton, Alex Krizhevsky and Vinod Nair. This dataset consists of 100 different category labels containing 600 images for each (1 testing image for 5 training images per class). There are a total of 100 classes in the CIFAR100. Each image contains a label which indicates their class. There are 50000 training images and 10000 testing images. It contains a total of 60000 images having 32x32 pixels in 100 classes.

This dataset was taken in the project from Alex Krizhevsky's homepage at University of Toronto.

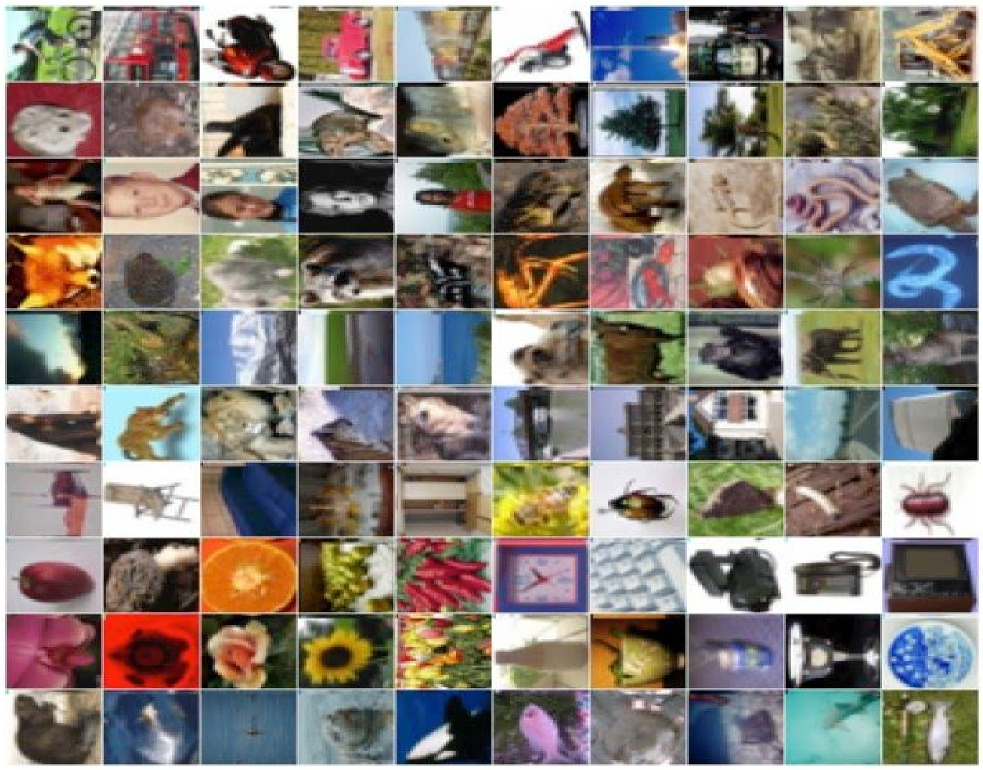
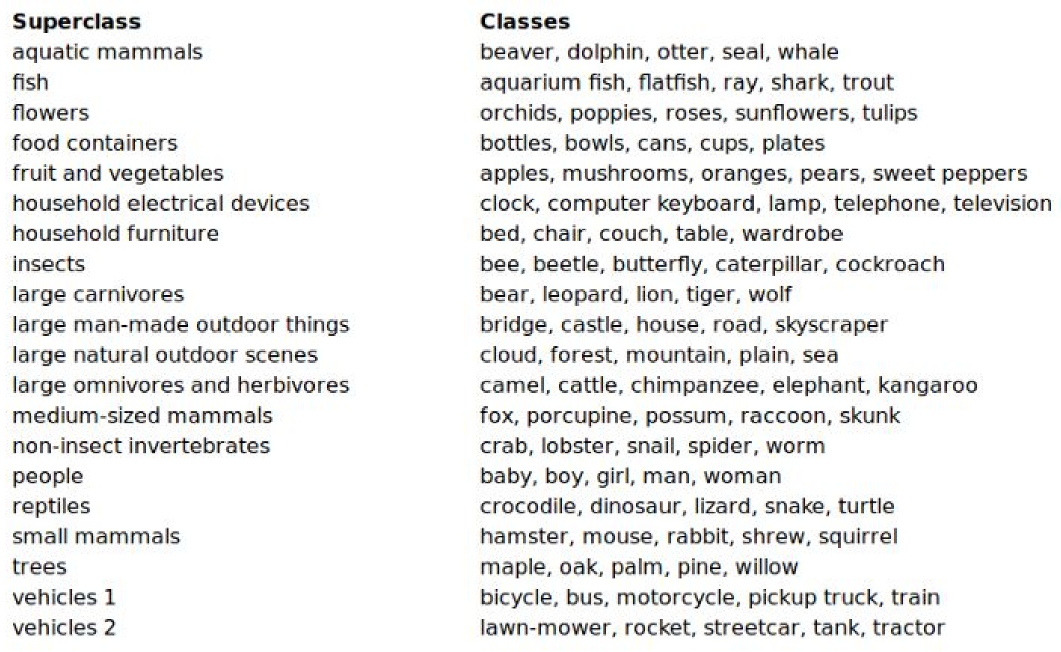


Fig c:- CIFAR100 dataset

The 100 classes in the CIFAR100 dataset are grouped into 20 super classes. These are as follows:-



Superclass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Fig d:- Super classes of CIFAR100 dataset

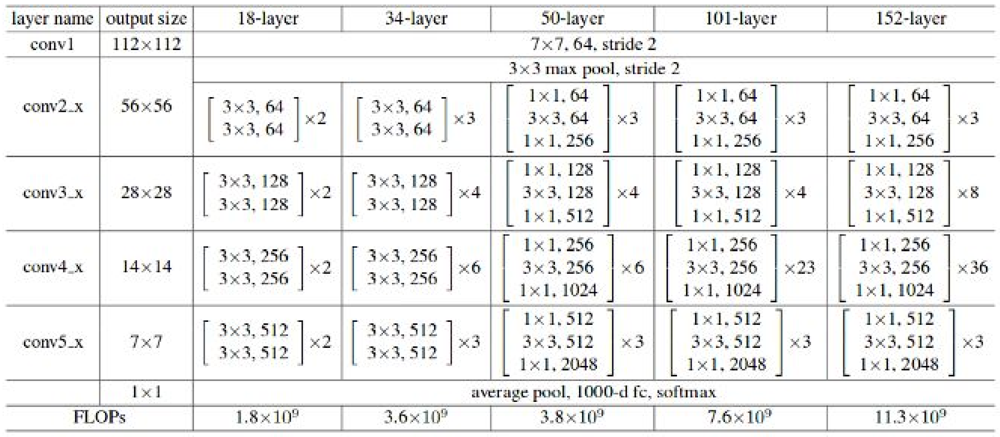
Implemented VGG16 using ADAM and SGD optimizer without regularisation, Batch Normalisation and Dropout on both of the two optimizers.

Important points:-

- 'ELU' activation function is used for both SGD and ADAM optimizer for proper comparison.
- Used 'RELU' activation function initially but it was found that 'RELU' activation function leads to lesser accuracy because of vanishing gradient problem as compared to the 'ELU' activation function. So,I used the 'ELU' activation function throughout the project.
- Used data augmentation to increase the diversity of data available for training models by cropping, padding and horizontal flipping to improve the performance of VGG16 network.
- Used the same layer configuration for VGG16 without regularisation.
- For VGG16 with Batch Normalisation and Dropout, used 1 less Dense layer as compared to the original layer configuration of VGG16.
- Implemented VGG16 having Only Dropout. Later, found that VGG16 having only Dropout was giving less accuracy. So used Dropout as well as Batch normalisation together. But, the notebook folder contains VGG16 having Dropout only and Dropout(Both Dropout and Batch Normalisation) both.
- VGG16 SGD only Dropout was giving accuracy:- 27% accuracy while VGG16 ADAM was giving an accuracy of 1% only. Later used both Batch Normalisation and Dropout to improve the accuracy of VGG16 architecture.

2.2 RESNET18

2.1 RESNET18 (Original Implementation)



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Fig e:- Resnet architecture

The original Resnet model consists of one convolution and pooling step followed by four convolutional layers of the same behaviour.

In Resnet18 architecture two rules of thumb needs to follow :-

- Each Convolutional layer has the same configuration of kernel size = 3x3, stride = 2x2. The only thing which differs is the number of filters.
- Downsampling with CNN layers with stride 2.
- Global average pooling layer and a fully connected layer with Softmax in the end.

2.1.1 Implementation(In this Project) different from the Original Implementation

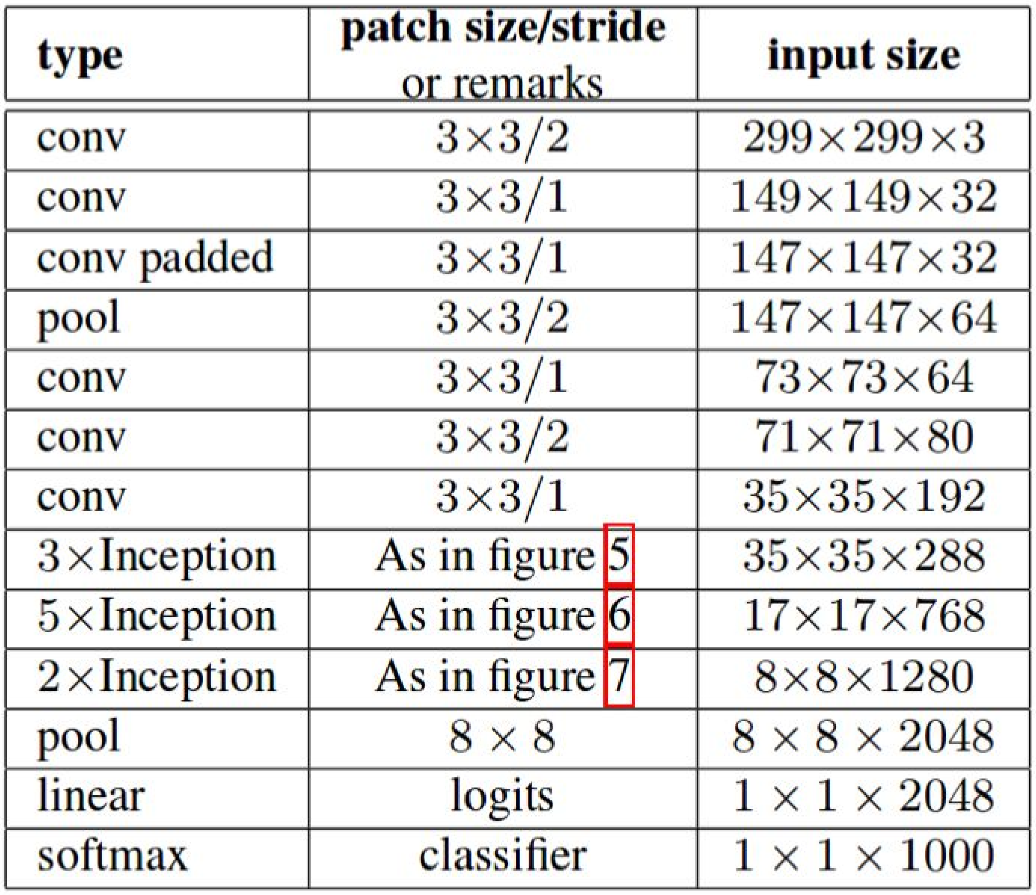
In Resnet18, there is no significant difference in the layer details of Resnet18 compared to the original implementation. Resnet18 that was performed on ImageNet dataset. In this project, Resnet18 is implemented on the CIFAR100 dataset.

Important Points:-

- 'ELU' activation function is used for both SGD and ADAM optimizer for proper comparison.
- 'ELU' activation function was giving more accuracy over Relu activation function. So, I used Relu throughout Resnet18 architecture.
- Copied layer details from the architecture diagram and written in form of code.
- Getting very less accuracy when used only Dropout. So, used Dropout with Batch Normalisation together which led to increase the accuracy of the model.
- In the Identity block of Resnet18, I used padding as "same" rather than "valid"

2.3 InceptionV2

2.1 InceptionV2(Original Implementation)



type	patch size/stride or remarks	input size
conv	$3\times3/2$	$299\times299\times3$
conv	$3\times3/1$	$149\times149\times32$
conv padded	$3\times3/1$	$147\times147\times32$
pool	$3\times3/2$	$147\times147\times64$
conv	$3\times3/1$	$73\times73\times64$
conv	$3\times3/2$	$71\times71\times80$
conv	$3\times3/1$	$35\times35\times192$
$3\times$Inception	As in figure 5	$35\times35\times288$
$5\times$Inception	As in figure 6	$17\times17\times768$
$2\times$Inception	As in figure 7	$8\times8\times1280$
pool	8×8	$8\times8\times2048$
linear	logits	$1\times1\times2048$
softmax	classifier	$1\times1\times1000$

Fig f:- Inception Architecture

Important points :-

- 'ELU' activation function is used for both SGD and ADAM optimizer for proper comparison.
- Used 'RELU' activation function initially but later on found that 'RELU' activation function leads to lesser accuracy as compared to the 'ELU' activation function. So, I used the 'ELU' activation function throughout the project.
- Implemented InceptionV2 having Only Dropout. Later, found that InceptionV2 having only Dropout was giving less accuracy. So used Dropout as well as Batch normalisation together.
- I was getting less accuracy of 26 % for Inception.So, I reduced the number of times each module is called. In the original Inception architecture, Module A should call 3 times, ModuleB should call 5 times and ModuleC should call 2 times. I reduce the calling times of ModuleA by 2, ModuleB by 3 and ModuleC remains constant. This led to improving my accuracy.

3 Network Explanation

3.1 VGGNet16

3.1.1 Evolution of VGG

VGG16 was developed to reduce the number of parameters in the convolutional layer.
VGG16 is a convolutional neural network architecture which was used to win "The ImageNet Large Scale Visual Recognition Challenge"(Imagenet) which is an annual computer vision competition. This competition is held every year where each team competes on two tasks. The first task of this competition is to detect objects within an image coming from 200 classes, which is called object localisation. The second task of this competition is to classify images, each labelled with one of 1000 categories, which is called image classification. VGG 16 was proposed by two researchers named Karen Simonyan and Andrew Zisserman of the Visual Geometry Group Lab of Oxford University in 2014. They published their work in the paper "Very Deep Convolutional Networks For Large-Scale Image Recognition". This model won the 1st and 2nd place in the above categories in the 2014 ILSVRC challenge.

3.1.2 Dataset Used

ImageNet is a dataset which consists of more than 15 million images which is labelled with more than 22 thousand classes. The images were collected from the web and it is labeled by human labellers using Amazon's Mechanical Turk crowd-sourcing tool. The dataset contains approximately 1.2 million training images, 50,000 validation images, and 150,000 testing images. ImageNet dataset consists of images with variable resolution. Hence, the images were down-sampled in order to make a fixed resolution of 256×256.

3.1.3 Architecture of VGG16

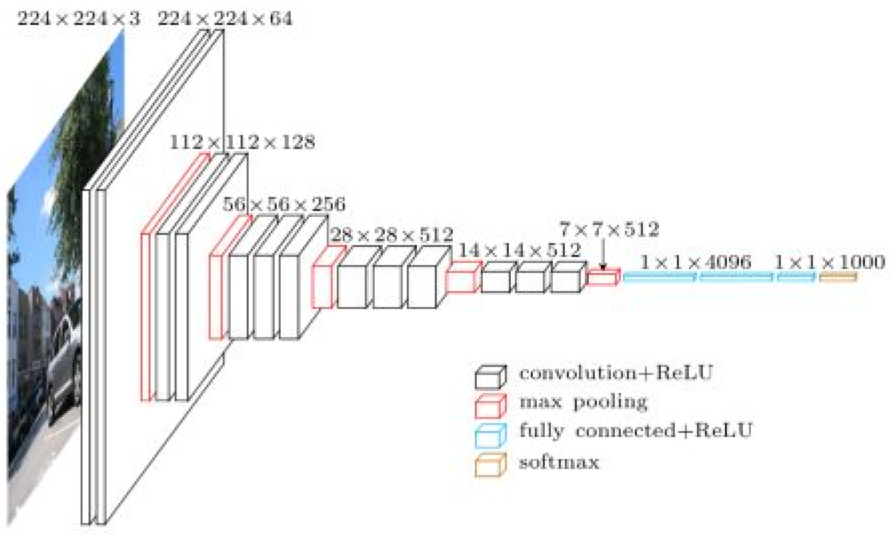


Fig g:- Architecture of VGG16

The input to the network are RGB images with a dimension of (224, 224, 3) where each image is of 224x224 pixels.There are total 138M parameters. The 224 x 224 RGB images are served as the input to the conv1 layer. The image is passed through multiple convolutional layers. The first two layers of VGG16 have 64 channels of 3x3 filter size. Both layers have the same padding. Then, a max-pool layer of stride (2,2) is applied. After the maxpool layer, two convolutional layers are applied which have 256 filter and filter size (3, 3). This layer is followed by the max pooling layer of stride (2, 2) which is similar to the previous layer. After these layers, there are 2 convolution layers of filter size(3,3) and 256 filters. Next, there are 2 sets of 3 convolutional layers and a max pool layer. After that, the image is passed to the stack of two convolutional layers and max pooling layers.In order to prevent the spatial feature of the image, same padding is done after each convolutional layer. We get the (7,7, 512) feature map then this output is flatten to make the feature vector. At the end, three fully connected layers are present where the first layer takes input from the last feature vector and outputs a vector of size(1, 4096) , second layer also gives a vector of size (1,4096) and the third layer output a 1000 channels for 100 classes. Then the output of the third fully connected layer is passed to the softmax layer to normalise the classification error. All the hidden layers use ReLU as the activation function. ReLU is preferred as it is more computationally efficient as it decreases the issue of vanishing gradient problems and it also results in fast learning.

3.1.4 Advantage of VGG over other models

- This model focused on having convolution layers of 3x3 filter with a stride 1 and always used the same padding and maxpool layer of 2x2 filter of stride 2 and hence avoids having a large number of hyper-parameters. This makes VGG16 as one of the excellent vision model architecture till date.
- It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another.

3.1.5 Challenges of VGG16

- VGG16 is very slow to train(The original VGG model was train on Nvidia Titan GPU for 2-3 weeks)
- VGG16 takes lots of bandwidth and space which makes it inefficient.

3.2 RESNET18

3.2.1 Advantage of Resnet

- Resnet is able to solve the vanishing gradient problem. In Resnet, the gradient can easily flow through the skip connection in backward from later layer to initial filters. While in other networks, the gradient from where the loss function is calculated got shrunk to zero and as a result the weights are never updated.

3.2.2 Building Residual Network

In Resnet, the gradient are directly back propagated by two ways:-

- Shortcut Connection
- Skip Connection

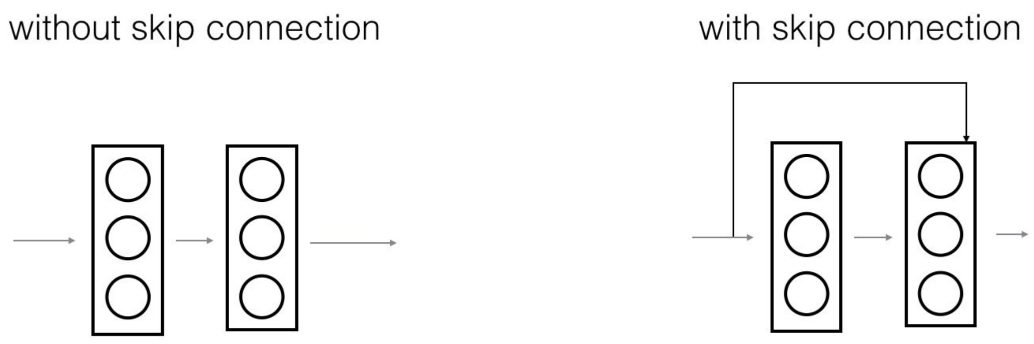


Fig h :- Residual Connection

The image which is at the left side is the "main path" whereas the image at the right side is the "shortcut" to the main path. More deeper and deeper networks can be formed by stacking Resnet blocks on the top of each other. Different types of Resnet blocks are used depending upon the dimension of input/output. It is easy for one of the blocks to learn the identity function due to shortcut connection in Resnet which makes little risk of harming training set performance while stacking the layers.

3.2.2.1 Identity Block

It is the standard block of Resnet and it is used in the case where the input activation has the same dimension as the output dimension.

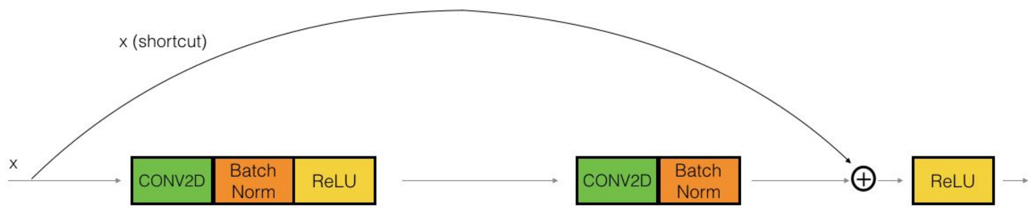


fig i:- shortcut path

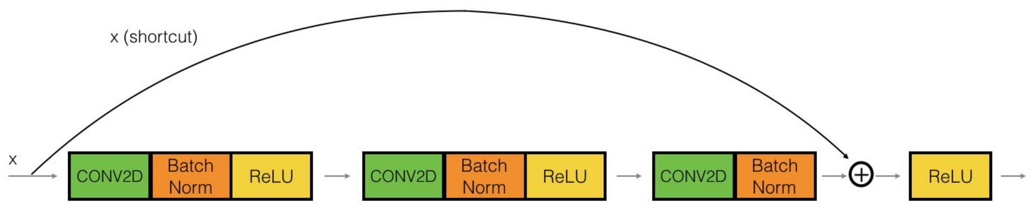


fig j:- Main path

Basically, this diagram is of **Resnet50**. In Resnet18, there are only two filters in the main path.

Details of Identity Block-

First component of Main path :-

- The first Conv2D has F1 filters of shape (1,1) and stride of (1,1) and padding is 'valid'
- The first BatchNorm is normalising the channel axis
- Then the Relu activation function is applied

Second component of Main path :-

- The second Conv2D has F2 filters of size (f,f) and stride of (1,1) and padding is 'valid'. Used 0 as the seed for random initialization.
- The second BatchNorm is normalising the channel axis
- Then the Relu activation function is applied

Final step:-

- The shortcut and input are added together
- Then apply the Relu activation function.

3.2.2.2 Convolutional Block

This block is used when the input and output dimensions are not equal. The main difference between the identity and the convolutional block is that there are a Conv2D layer in the shortcut path.

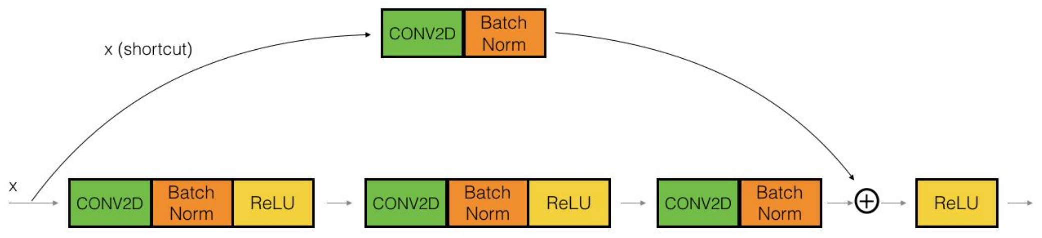


Fig k :- Convolutional Block

Details of Convolutional Block -

First component of Main path :-

- The first Conv2D has F1 filters of shape (1,1) and stride of (s,s) and padding is 'valid'
- The first BatchNorm is normalising the channel axis
- Then the Relu activation function is applied

Second component of Main path :-

- The second Conv2D has F2 filters of size (f,f) and stride of (1,1) and padding is 'valid'. Used 0 as the seed for random initialization.
- The second BatchNorm is normalising the channel axis

- Then the Relu activation function is applied

Shortcut path:-

- The Conv2D has F2 filters of shape (1,1) and stride of (s,s) and padding is ‘valid’
- The BatchNorm is normalising the channel axis

Final step:-

- The values of shortcut and main path are added together
- Then apply the Relu activation function.

3.2.2.3 Resnet Model

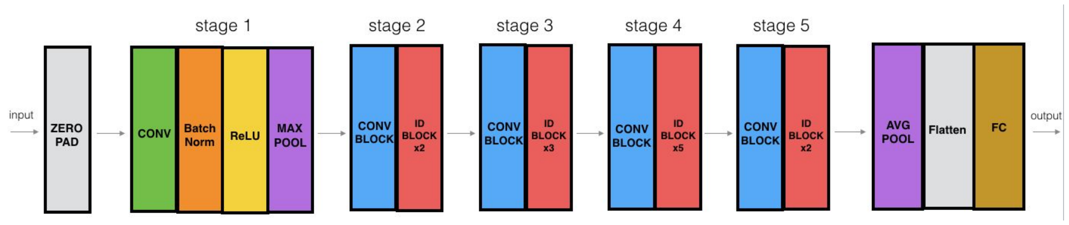


Fig 1: - Resnet model

Details of Resnet Model-

- Zero padding pads the input with a pad of (3,3)
- Stage 1:

1) The 2D convolution has a filter of shape (7,7) and stride of (2,2). There are a total of 64 filters.

2) The BatchNorm is applied to the channel axis of the input

3) Max Pooling uses a (3,3) window and a stride of (2,2)

- Stage 2:

1) The convolutional block uses two sets of filters of size [64,64]

2) The two identity blocks use two sets of filters of size [64,64]

- Stage 3:

1) The convolutional block uses two sets of filters of size [128,128]

2) The two identity blocks use two sets of filters of size [128,128]

- Stage 4:

1) The convolutional block uses two sets of filters of size [256,256]

2) The two identity blocks use two sets of filters of size [256,256]

- Stage 5:

  1) The convolutional block uses two sets of filters of size [512,512]

  2) The two identity blocks use two sets of filters of size [512,512]

- The 2D average Pooling uses a window of shape (2,2)
- Then Flatten is used
- The fully connected layers reduce the input to the number of classes using Softwax.

3.3 InceptionV2

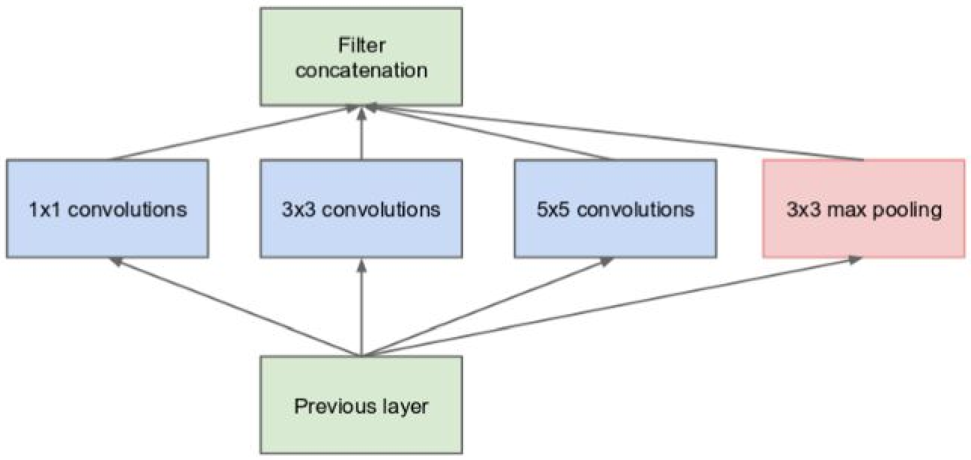
3.3.1 Overview of Inception

Inception was developed based on GoogLeNet architecture.The most important criteria of Inception architecture is it's adaption of "Network in Network" approach which increased the representational power of the neural networks. This had additionally saved them for computational bottlenecks by dimension reduction to 1×1 convolutions. The inception network was an important turning point for the development of CNN classifiers. Earlier, generally CNN are just stacked convolutional layers deeper and deeper aiming to get better performance.

3.3.2 Versions of Inception:-

- InceptionV1
- InceptionV2 and InceptionV3
- InceptionV4 and Inception-Resnet

3.3.3 Basic Architecture of Inception



(a) Inception module, naïve version

Fig m :- Naive version of Inception Module

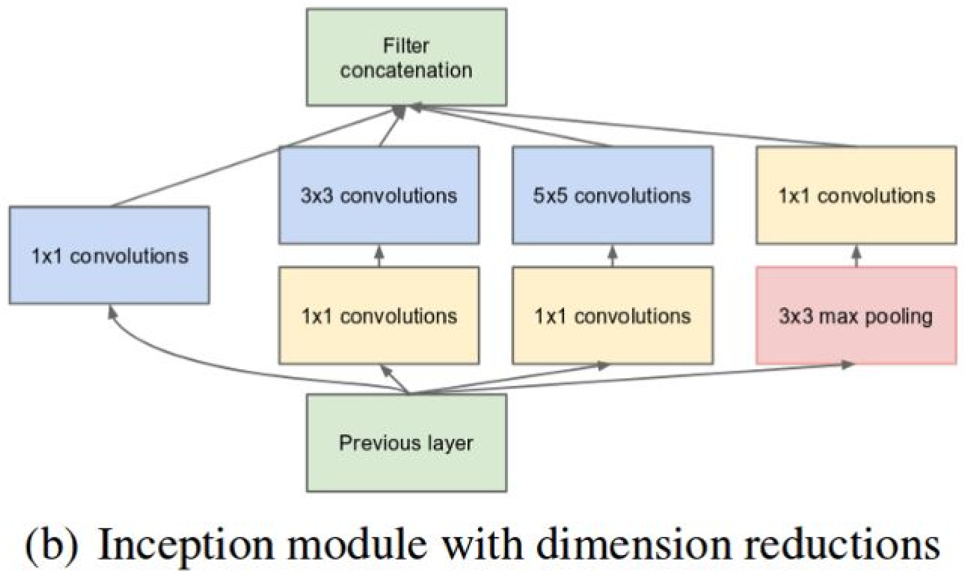


Fig n: Inception Module with reduced dimension

The researchers made lots of upgrades which has increased the accuracy of the inception. It also reduced the computational complexity.

The Premise:-

- Neural networks perform better than convolutional networks when the convolutional network didn't alter the dimensions of the input very much. On reducing the dimensions much may lead to loss of information.
- Convolutions can be made more efficient in the term of computational complexity using smart factorization methods.

Solution:-

- In order to improve the computational speed, 5x5 convolution can be factorize into two 3x3 convolution as 5x5 convolution is 2.78 times more expensive than a 3x3.Thus, stacking of the two 3x3 convolution instead of 1 5x5 convolution leads to a boast in the performance.

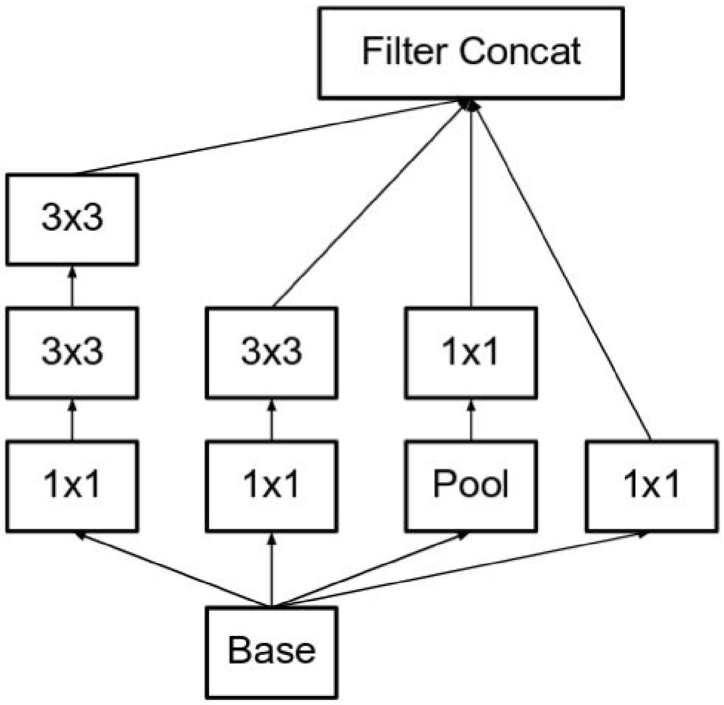


Fig o:- a 5x5 convolution is factored into two 3x3 convolution

- Factorize nxn convolution into a combination of 1xn and nx1 convolutions. For example, a 3x3 convolution is equivalent to first performing a 1x3 convolution and then performing 3x1 convolution.This way of factoring a 3x3 convolution into a combination of 1x3 and 3x1 convolutions are 33% more cheaper than a single 3x3 convolution.

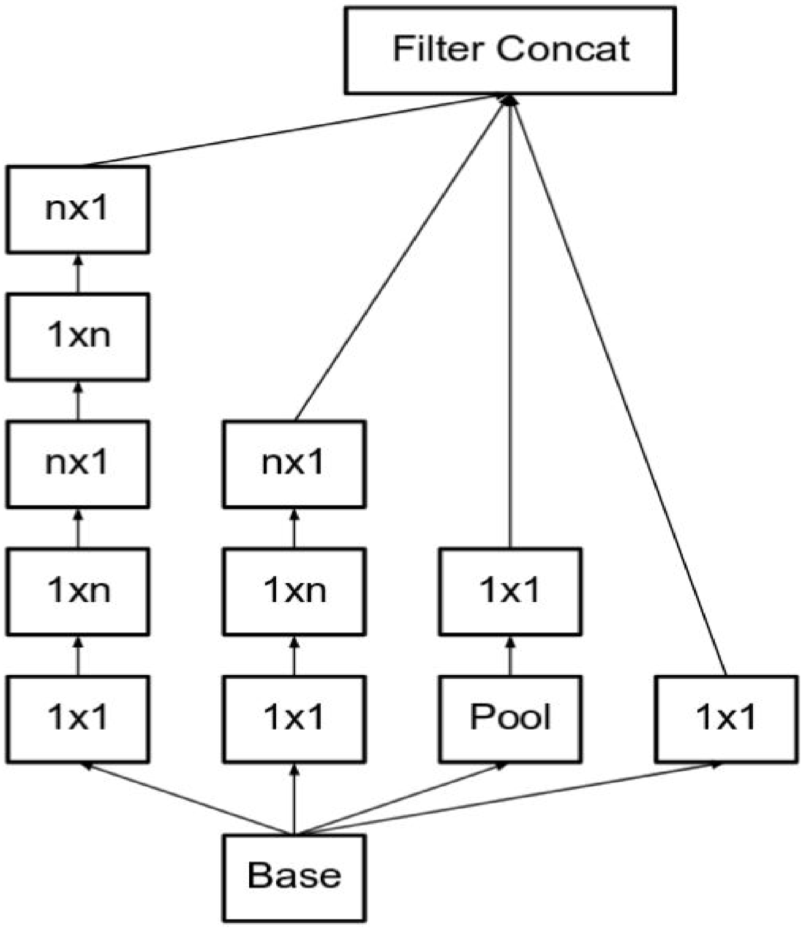


Fig p:- A 5x5 convolution can be represented as two 3x3 convolutions which in turn can be represented as 1x3 and 3x1 convolutions

- The filters in the modules were made more wider than the deeper.If the modules are made more deeper instead of wider than it would lead to excessive reduction in dimensions and hence loss of information.

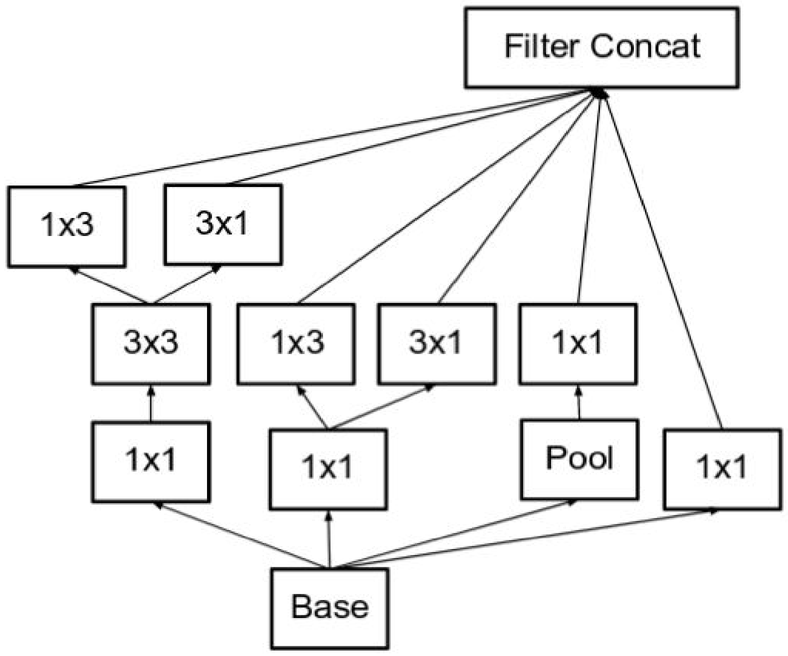


Fig q:- Filters are arranged in wider rather than deeper way

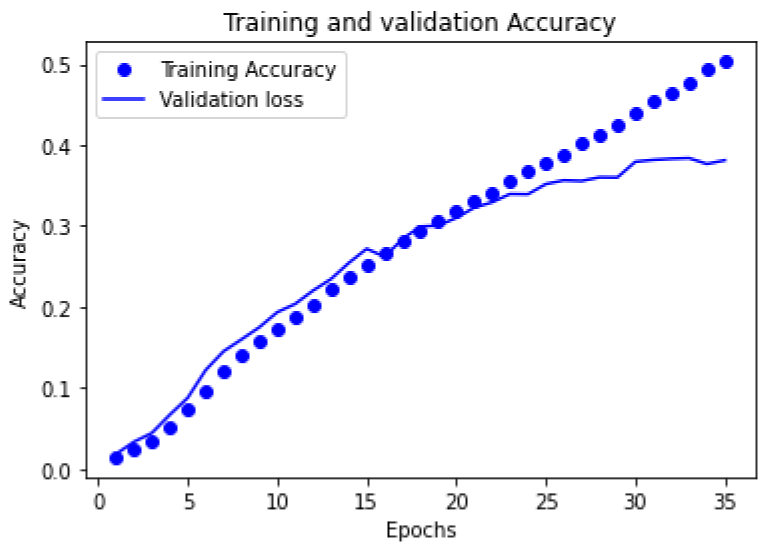
These three principles are used to make InceptionV2. The InceptionV2 architecture are mentioned in the Section3(Implementation of Architecture Inception).

4 Plots

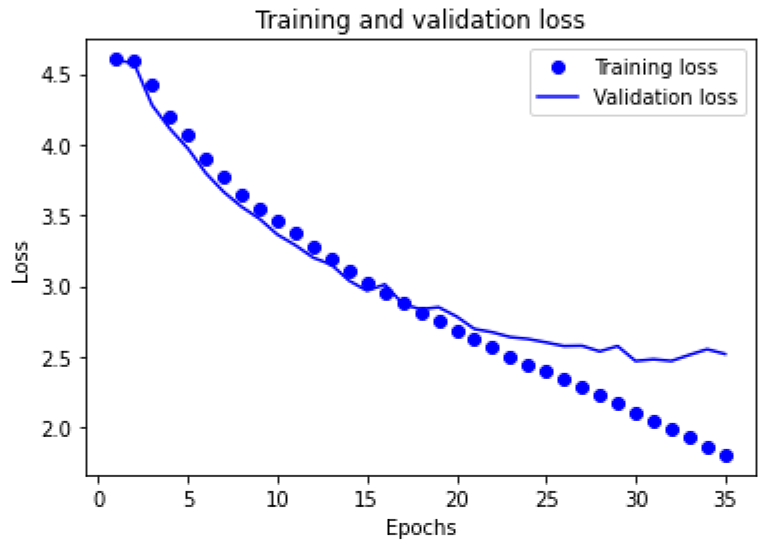
The Plots between accuracy and Loss with Epoch are mentioned below for each individual case using both ADAM and SGD optimizer.

4. VGG16

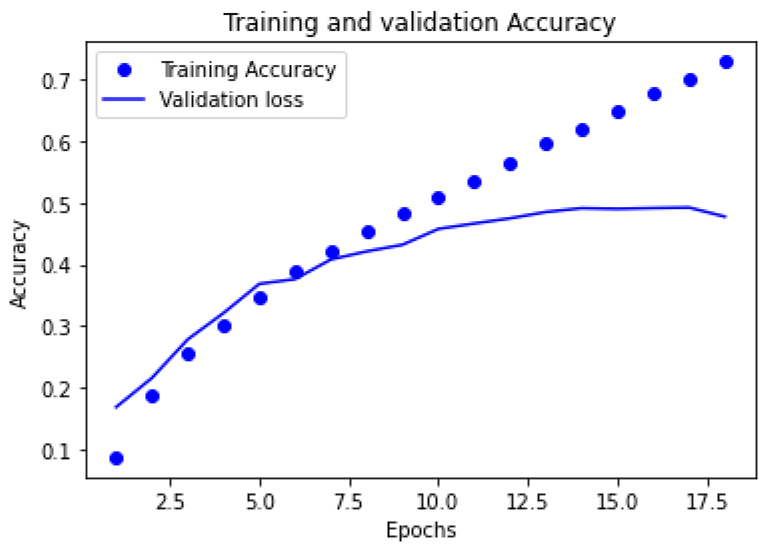
4.1.1.a Plot between Accuracy and Epoch for SGD No Regularisation



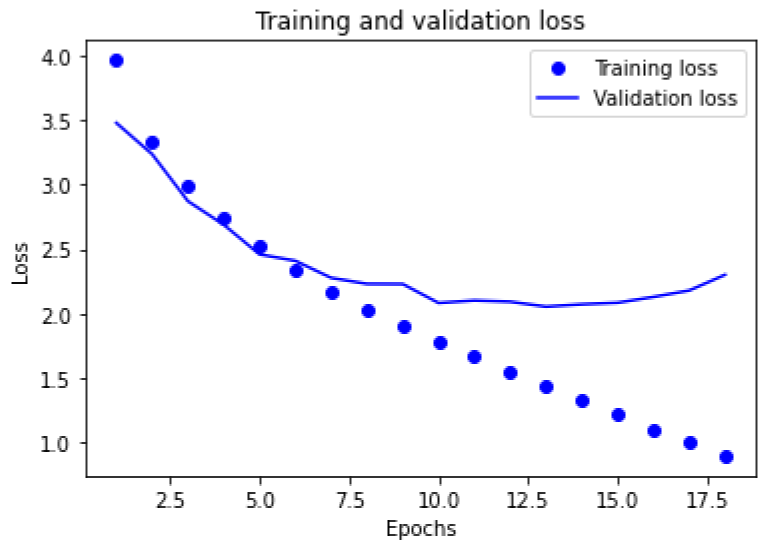
4.1.1.b Plot between Loss and Epoch for SGD without Regularisation



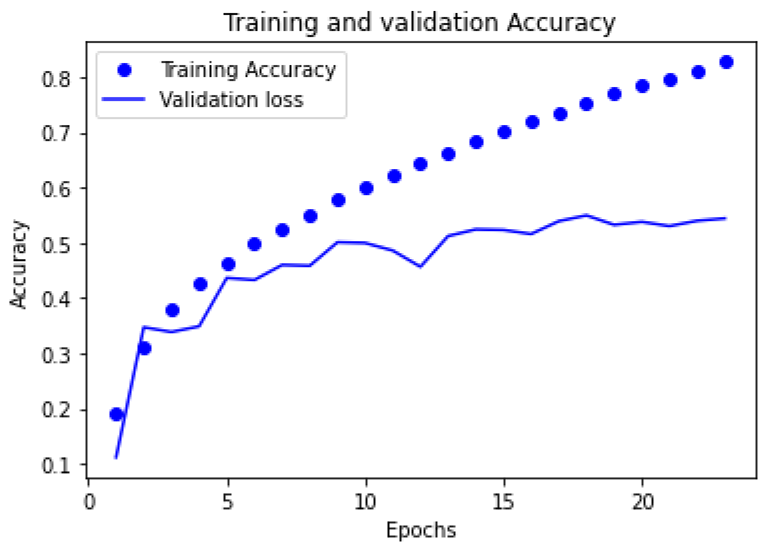
4.1.2.a Plot between Accuracy and Epoch for ADAM No Regularisation



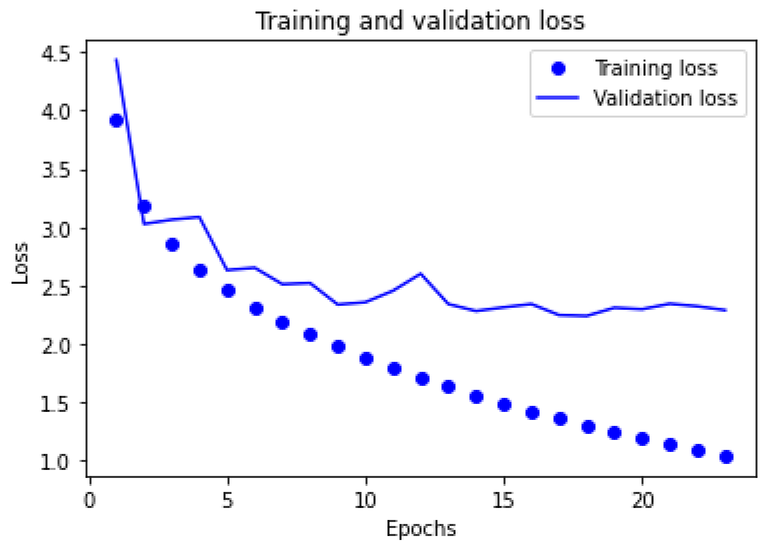
4.1.2.b Plot between Loss and Epoch for ADAM without Regularisation



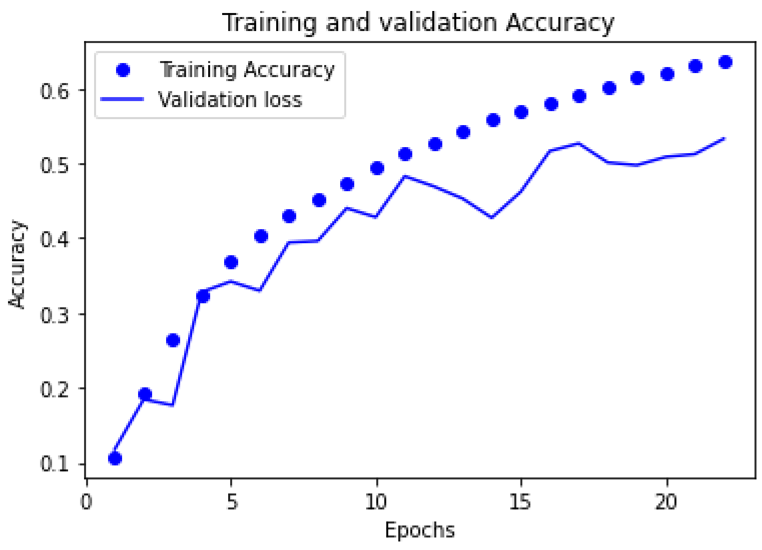
4.1.3.a Plot between Accuracy and Epoch for SGD with BATCH Normalisation



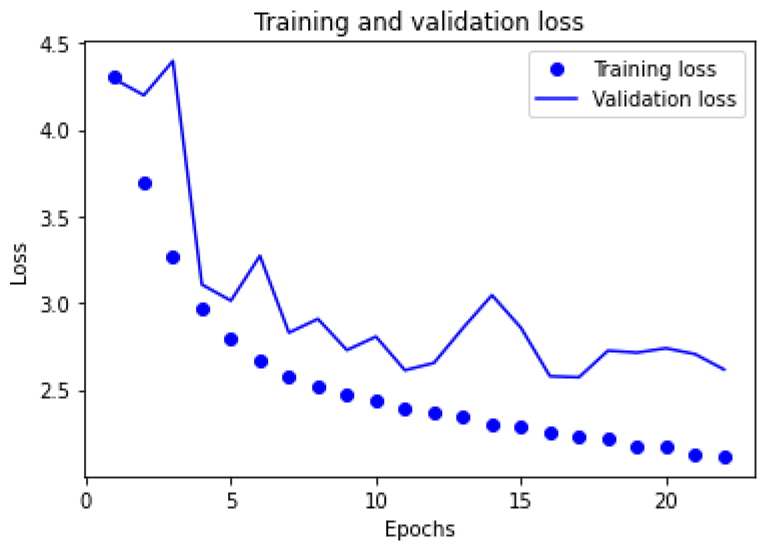
4.1.3.b Plot between Loss and Epoch for SGD with BATCH Normalisation



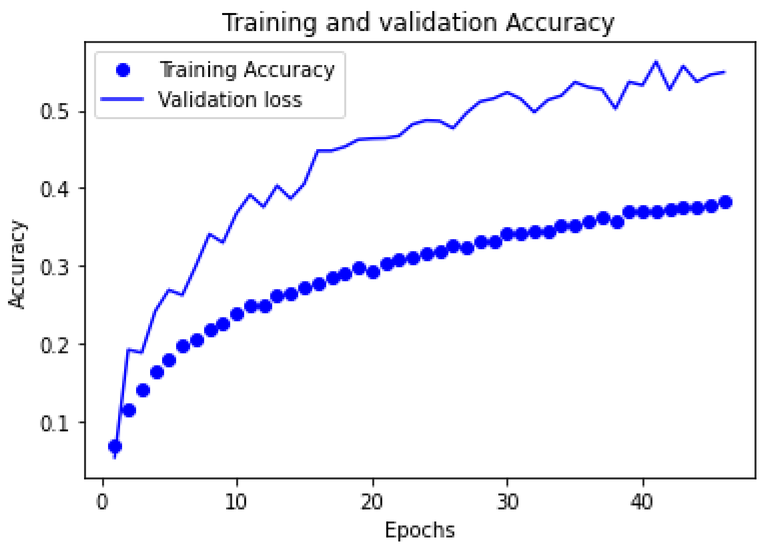
4.1.4.a Plot between Accuracy and Epoch for ADAM with BATCH Normalisation



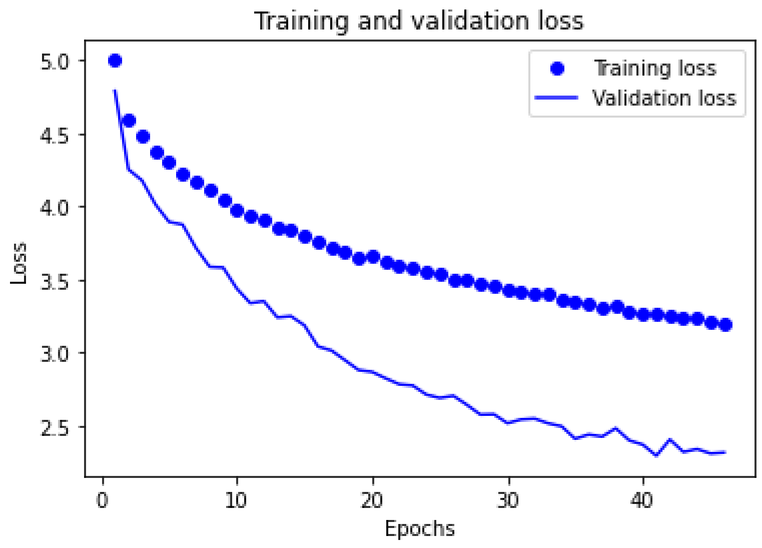
4.1.4 .b Plot between Loss and Epoch for ADAM with BATCH Normalisation



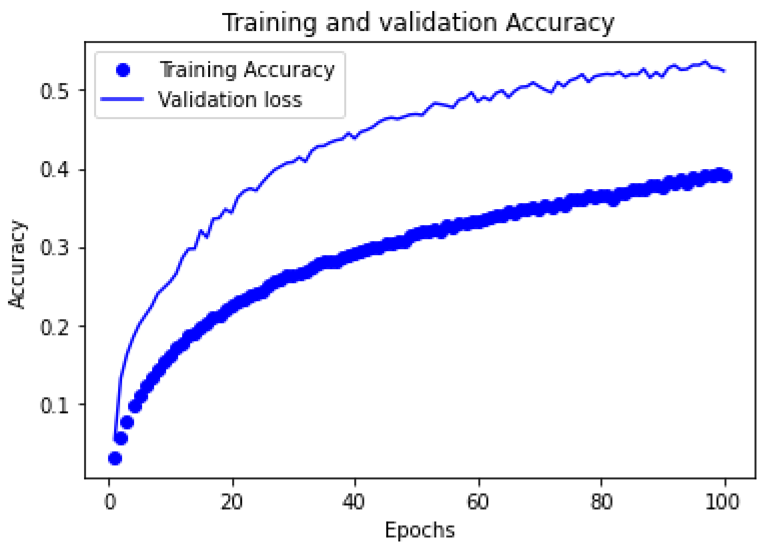
4.1.5.a Plot between Accuracy and Epoch for SGD with BATCH Normalisation and Dropout Together(Dropout)



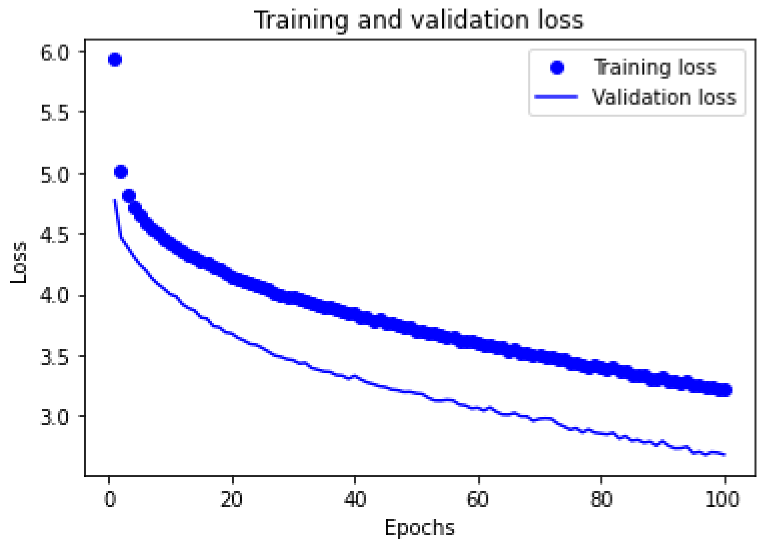
4.1.5.b Plot between Loss and Epoch for SGD with BATCH Normalisation and Dropout Together(Dropout)(Done)



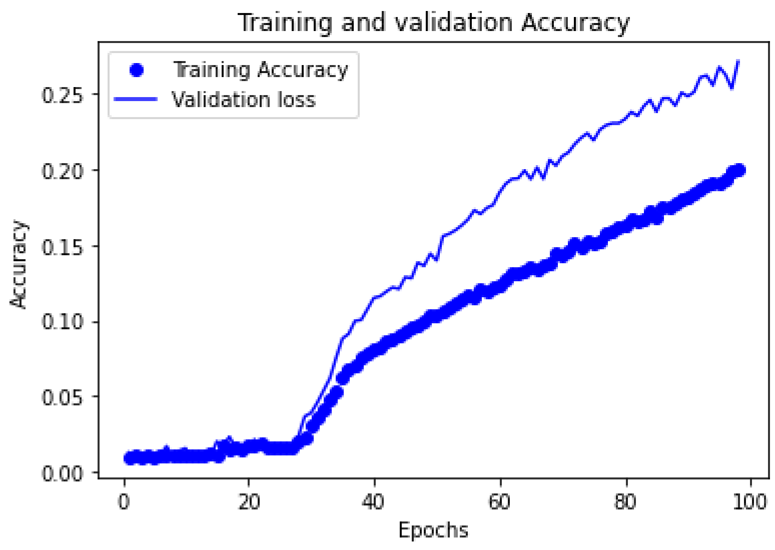
4.1.6.a Plot between Accuracy and Epoch for ADAM with BATCH Normalisation and Dropout Together(Dropout)



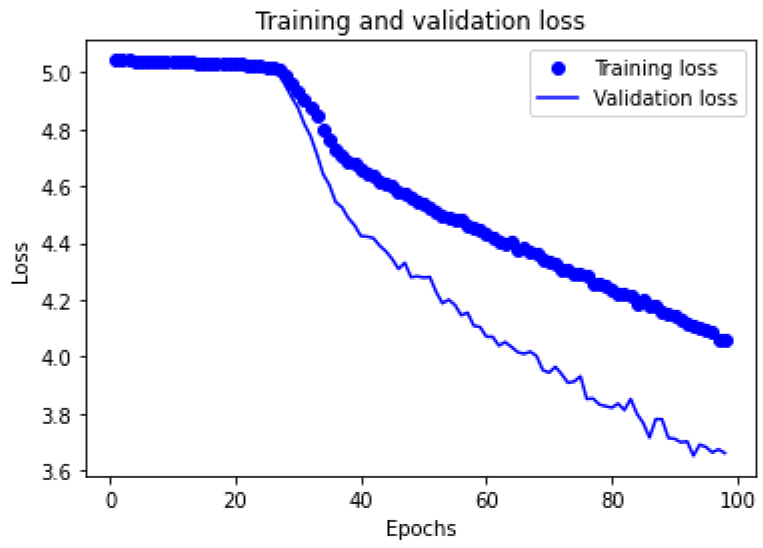
4.1.6.b Plot between Loss and Epoch for ADAM with BATCH Normalisation and Dropout Together(Dropout)



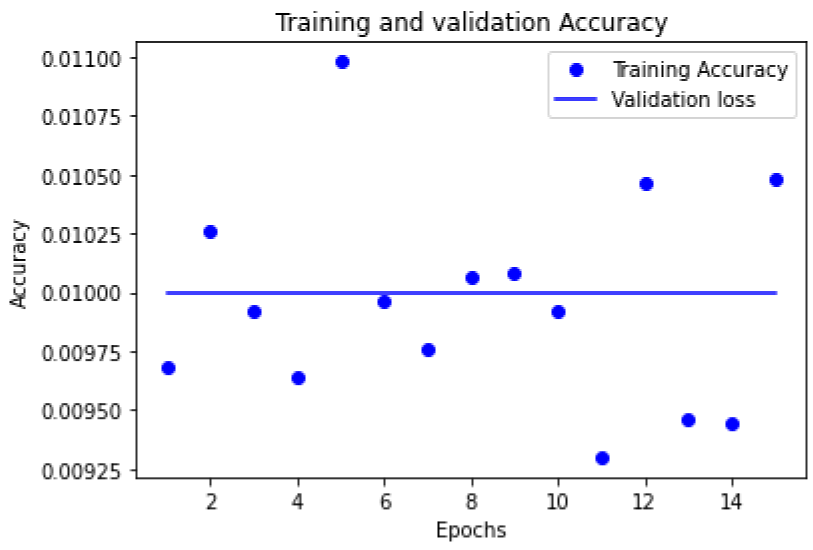
4.1.7.a Plot between Accuracy and Epoch for SGD with Dropout Only(OnlyDropout)



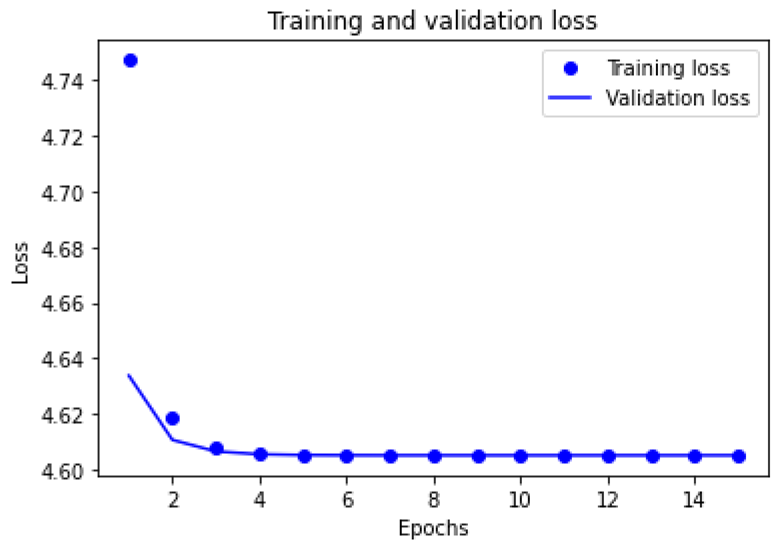
4.1.7.b Plot between Loss and Epoch for SGD with Dropout Only(OnlyDropout)



4.1.8.a Plot between Accuracy and Epoch for ADAM with Dropout Only(OnlyDropout)

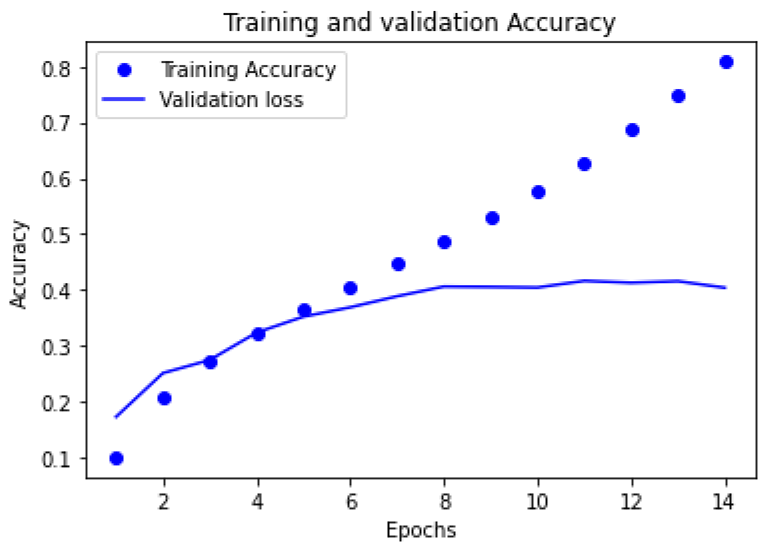


4.1.8.b Plot between Loss and Epoch for ADAM with Dropout Only(OnlyDropout)

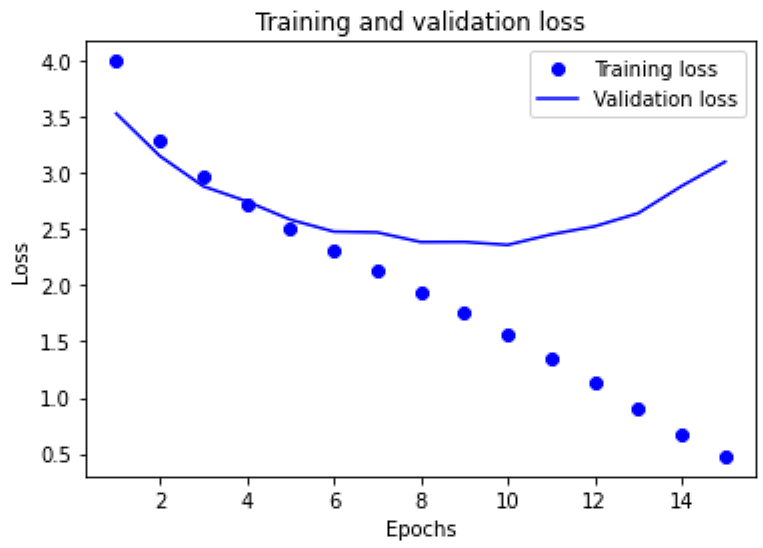


4.2 RESNET18

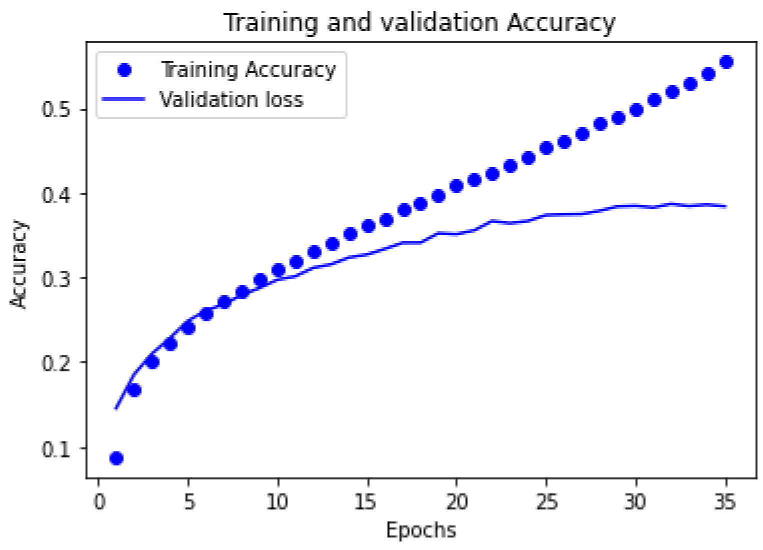
4.2.1.a Plot between Accuracy and Epoch for SGD No Regularization



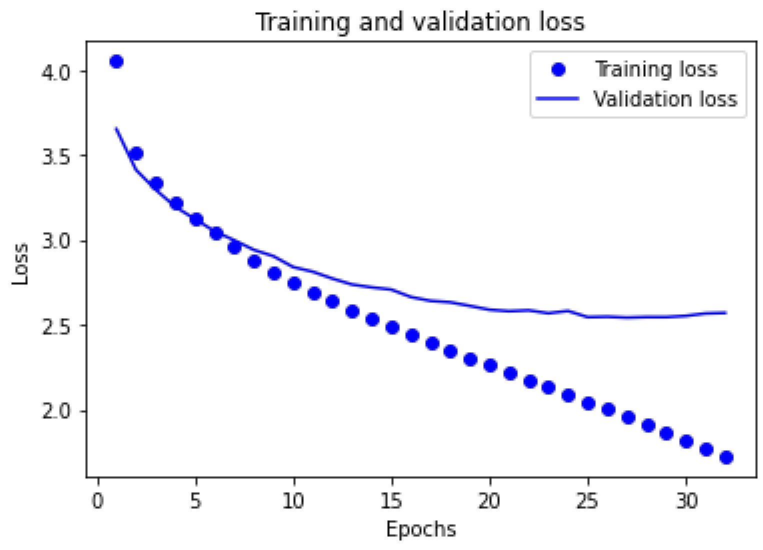
4.2.1.b Plot between Loss and Epoch for SGD No Regularisation



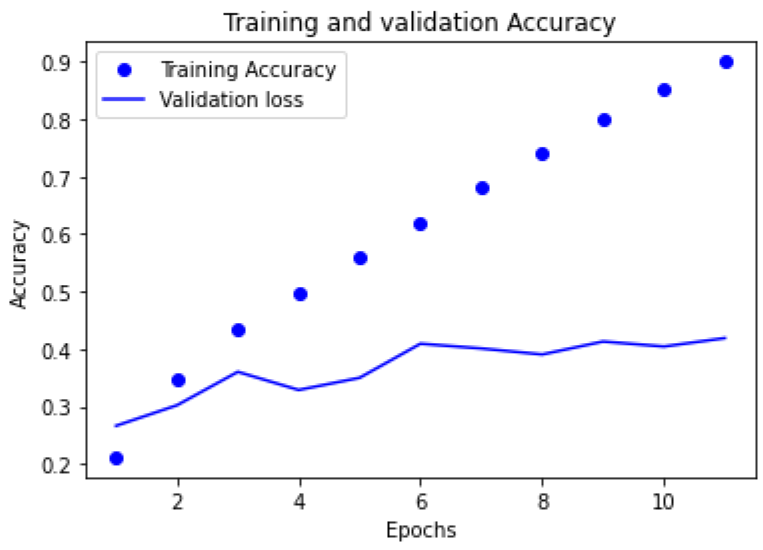
4.2.2.a Plot between Accuracy and Epoch for ADAM No Regularisation



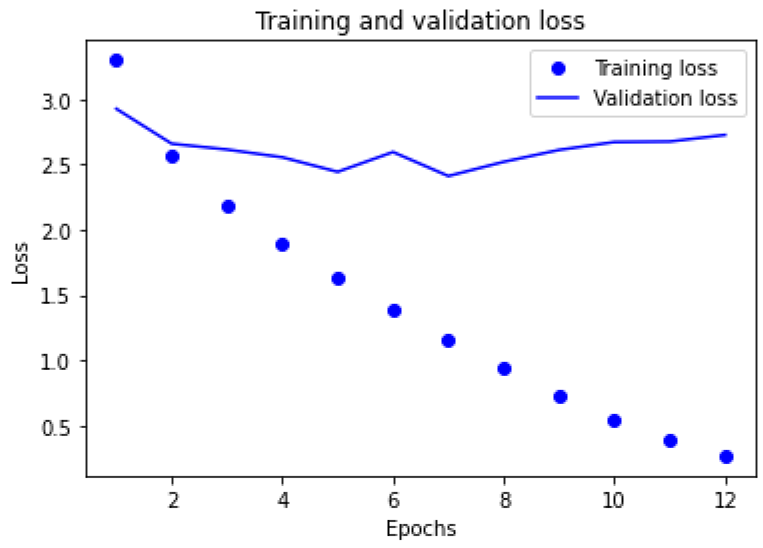
4.2.2.b Plot between Loss and Epoch for ADAM No Regularization



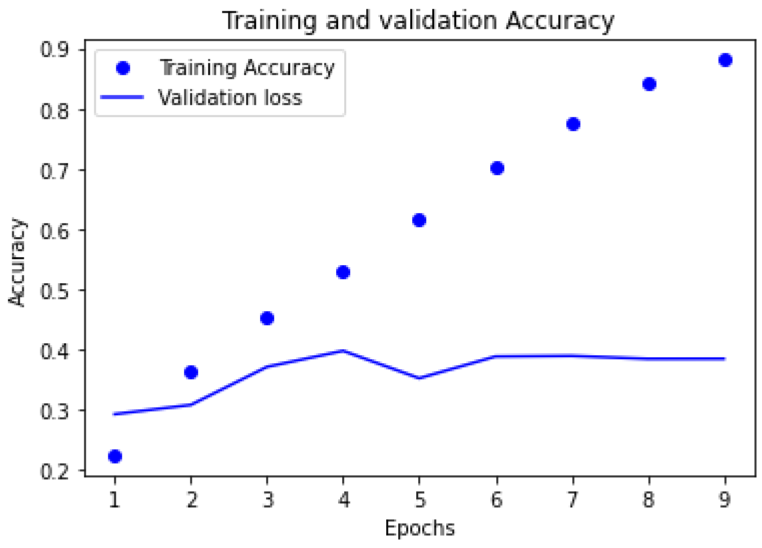
4.2.3.a Plot between Accuracy and Epoch for SGD with BATCH Norm



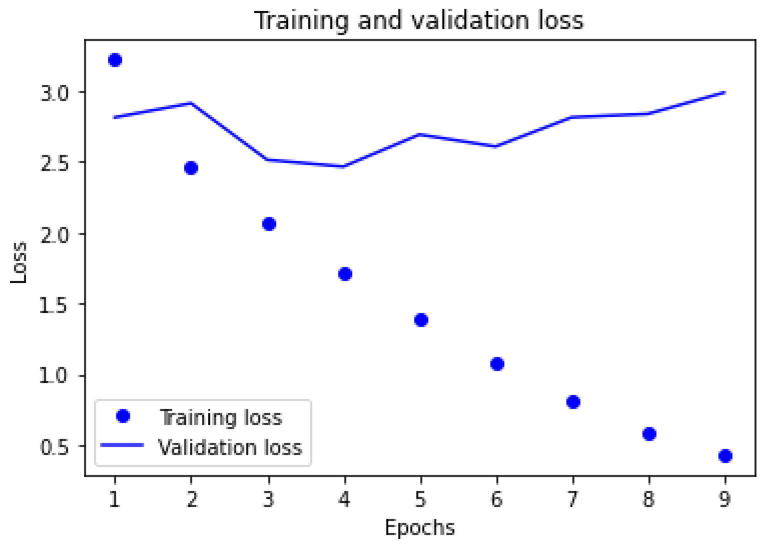
4.2.3.b Plot between Loss and Epoch for SGD with BATCH Norm



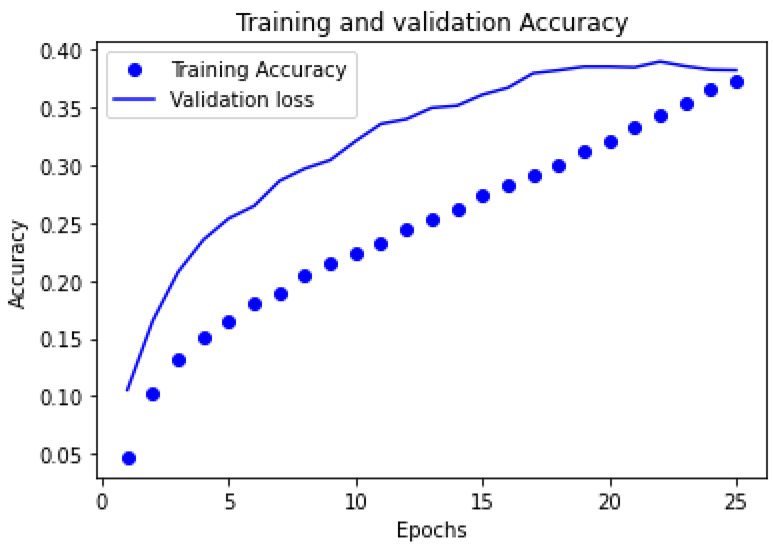
4.2.4.a Plot between Accuracy and Epoch for ADAM with BATCH Normalisation



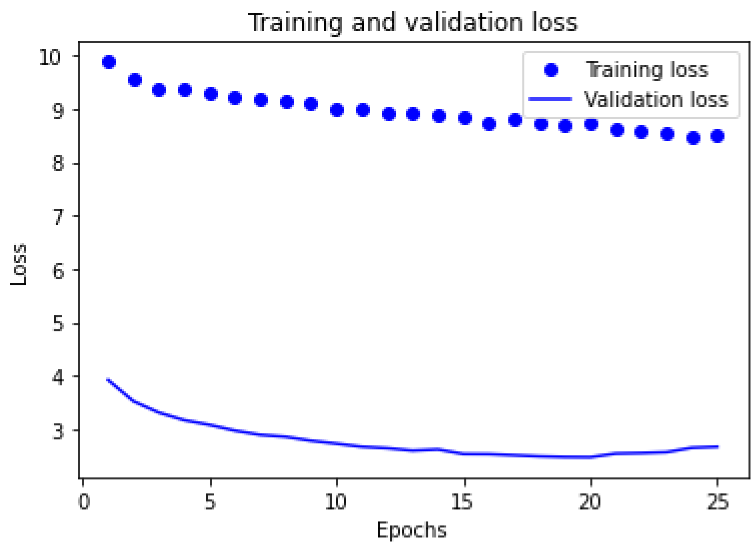
4.2.4.a Plot between Loss and Epoch for ADAM with BATCH Normalisation



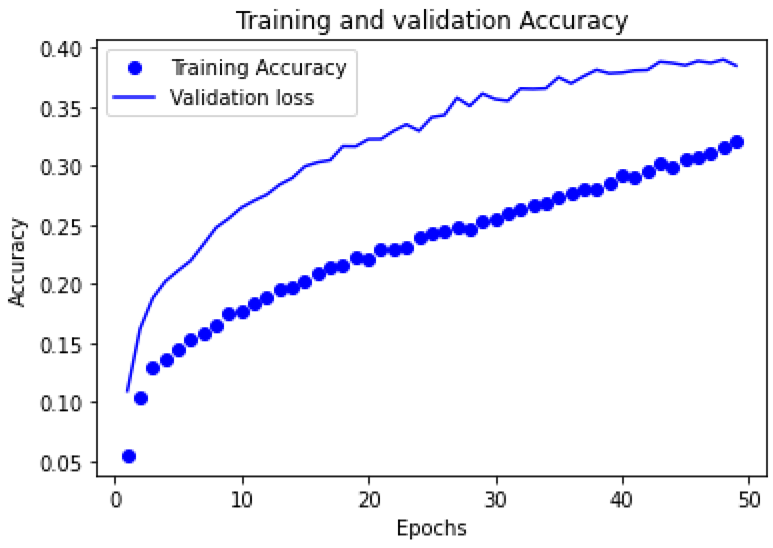
4.2.5.a Plot between Accuracy and Epoch for SGD with BATCH Normalisation and Dropout Together(Dropout)



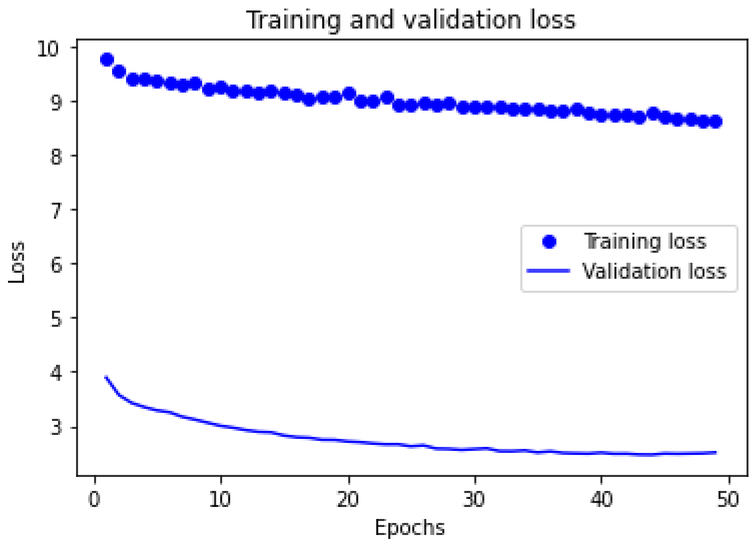
4.2.5.b Plot between Loss and Epoch for SGD with BATCH Normalisation and Dropout Together(Dropout)



4.2.6.a Plot between Accuracy and Epoch for ADAM with BATCH Normalisation and Dropout Together(Dropout)

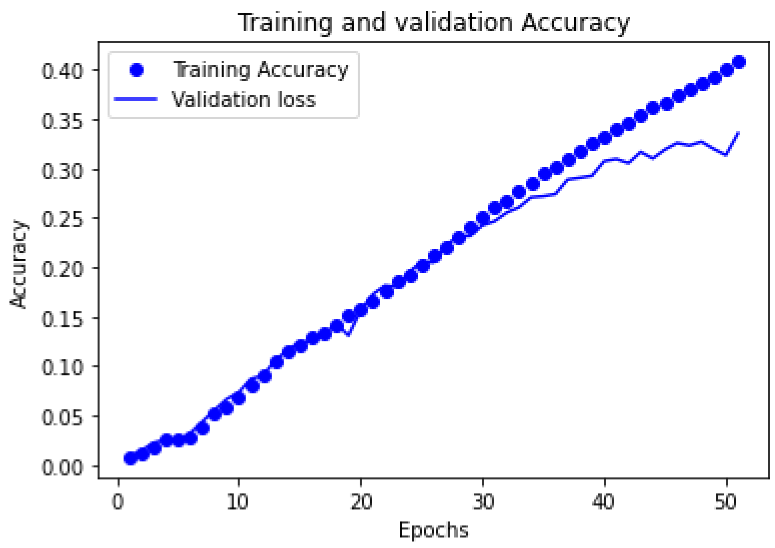


4.2.6.b Plot between Loss and Epoch for ADAM with BATCH Normalisation and Dropout Together(Dropout)

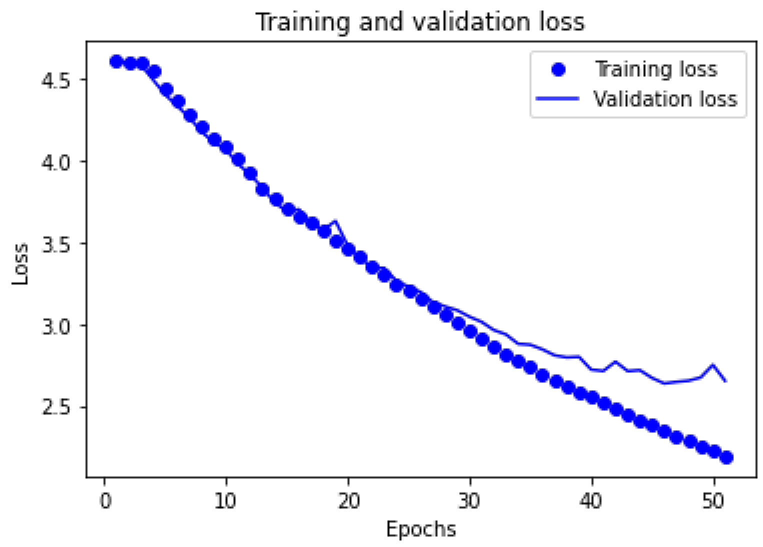


4.3 InceptionV2

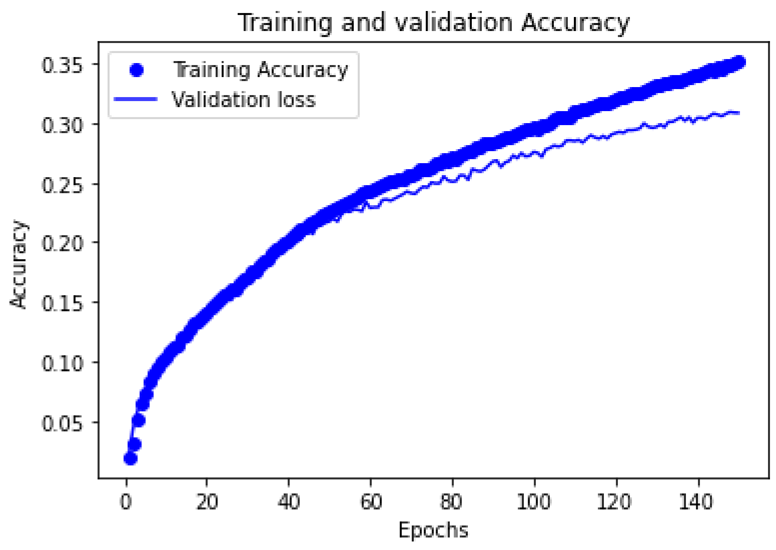
4.3.1.a Plot between Accuracy and Epoch for SGD No Regularization



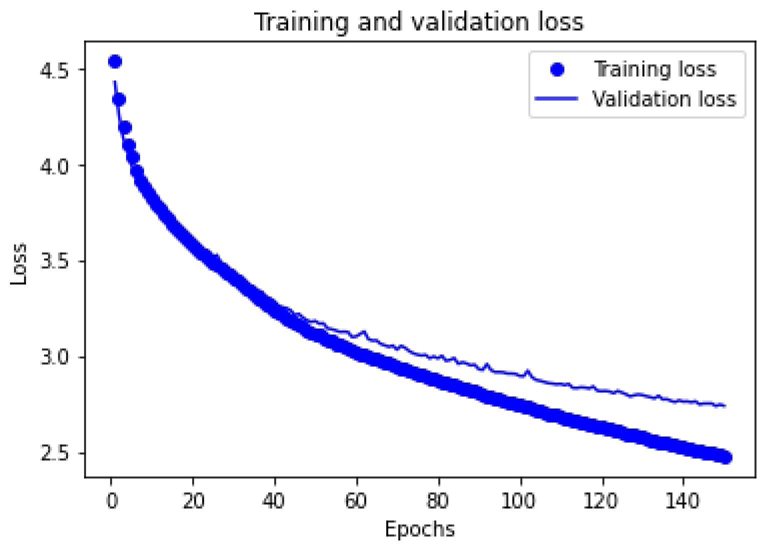
4.3.1.b Plot between Loss and Epoch for SGD No Regularization



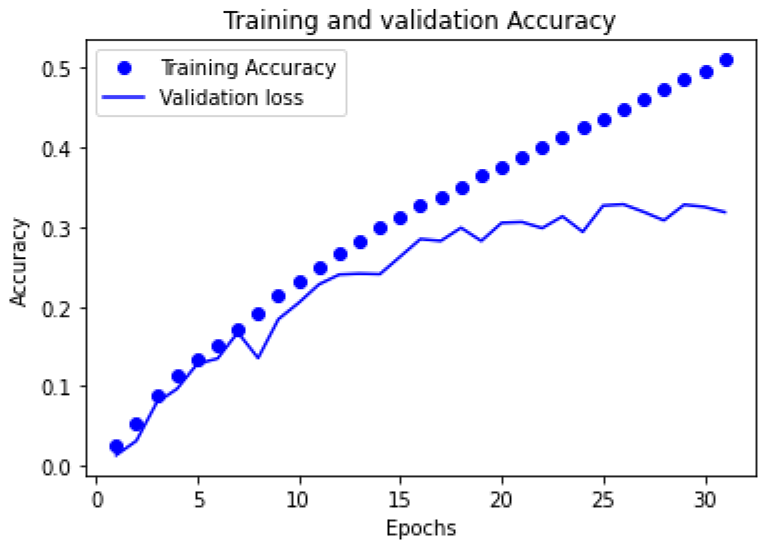
4.3.2.a Plot between Accuracy and Epoch for ADAM No Regularization



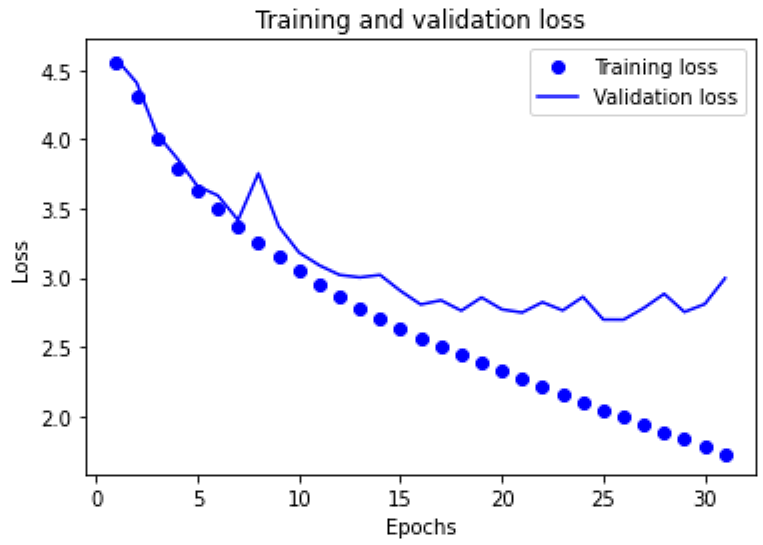
4.3.2.b Plot between Loss and Epoch for ADAM No Regularization



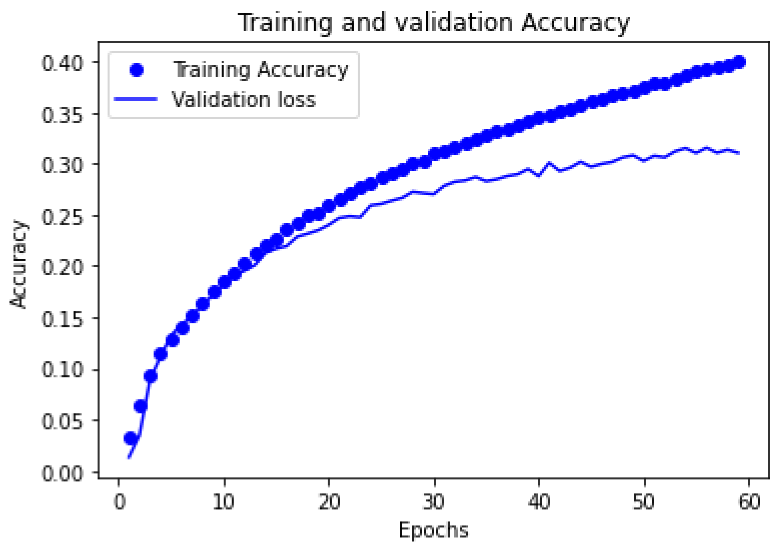
4.3.3.a Plot between Accuracy and Epoch for SGD with BATCH Normalisation



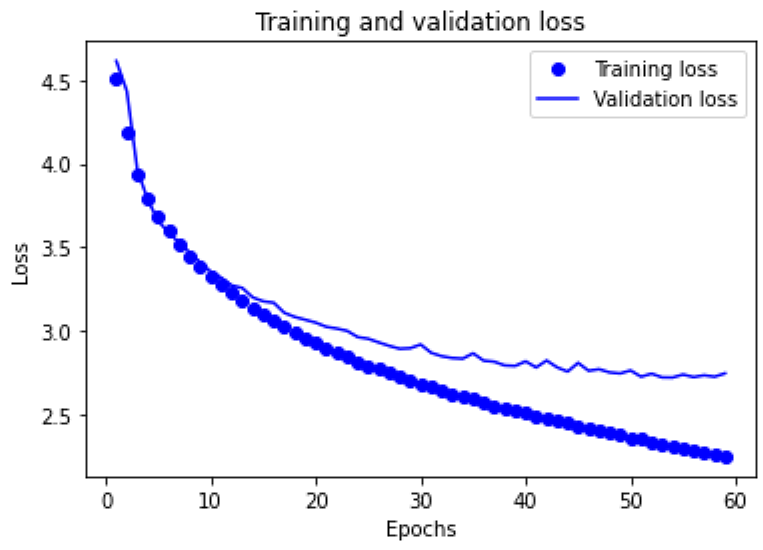
4.3.3.b Plot between Loss and Epoch for SGD with BATCH Normalisation



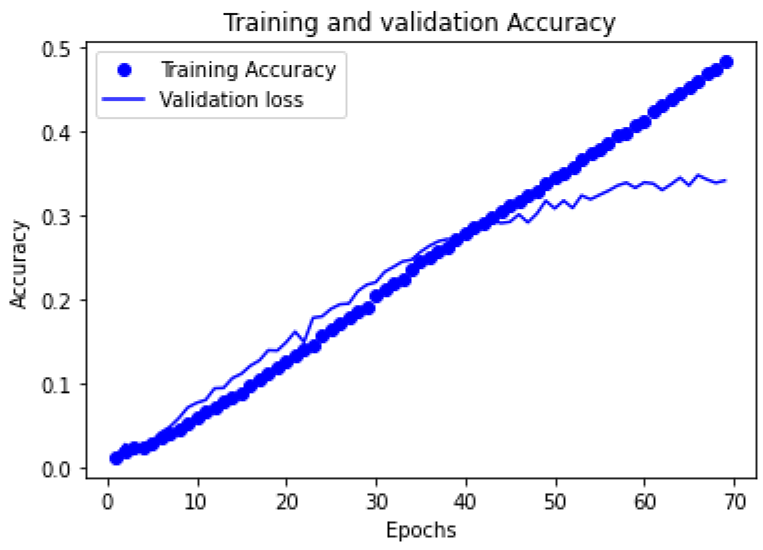
4.3.4.a Plot between Accuracy and Epoch for ADAM with BATCH Normalisation



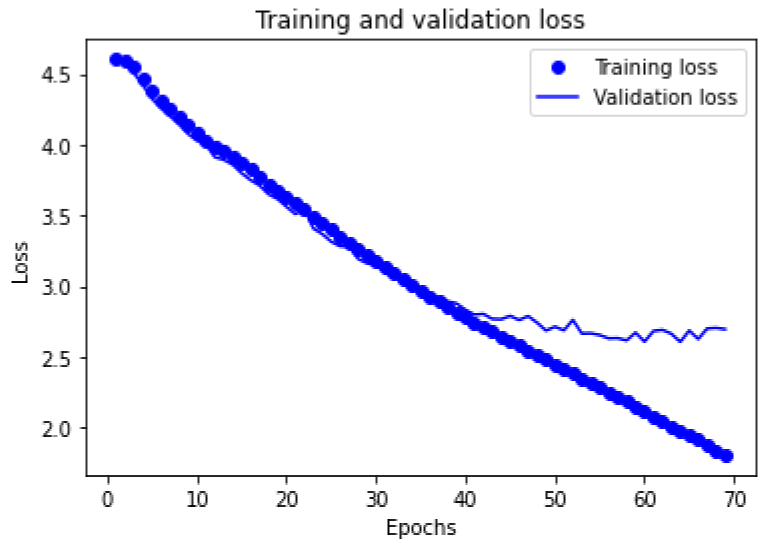
4.3.4.b Plot between Loss and Epoch for ADAM with BATCH Normalisation



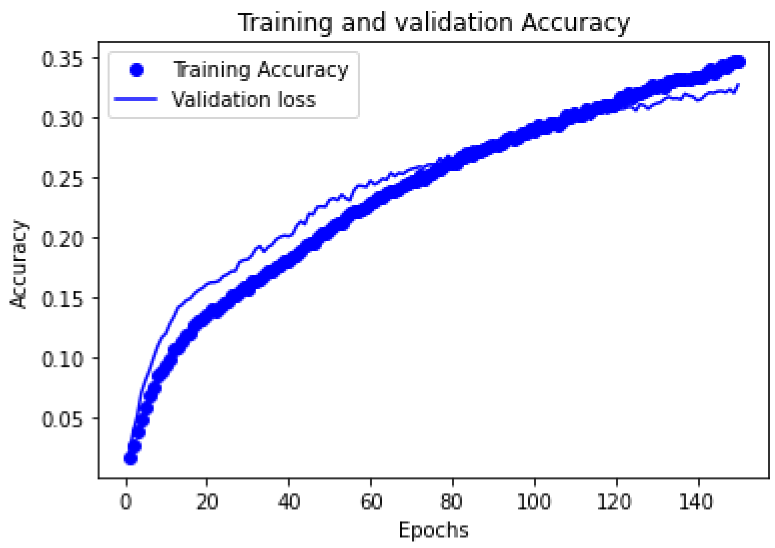
4.3.5.a Plot between Accuracy and Epoch for SGD with Dropout(With BatchNorm)



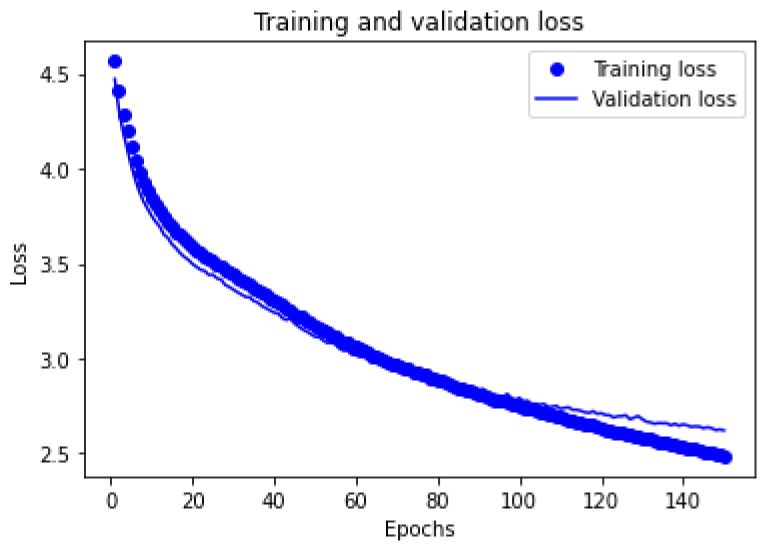
4.3.5.b Plot between Loss and Epoch for SGD with Dropout(With BatchNorm)



4.3.6.a Plot between Accuracy and Epoch for Adam with Dropout(With BatchNorm)



4.3.6.b Plot between Loss and Epoch for Adam with Dropout(With BatchNorm)



5 Table

	Arch	VGG16			Resnet18			InceptionV2		
Optimizer	Score / Setting	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy
SGD	With BatchNorm	0.56939	0.544	0.544	0.4465	0.43	0.43	0.3388	0.3285	0.3285

	With Dropout	0.5883	0.5492	0.5492	0.4185	0.3906	0.3906	0.34073	0.345	0.345
	No Regularization	0.4038	0.3812	0.3812	0.4384	0.4073	0.4073	0.3216	0.3258	0.3258
ADAM	With BatchNorm	0.5973	0.5328	0.5328	0.4151	0.3985	0.3985	0.3024	0.3151	0.3151
	With Dropout	0.5519	0.5243	0.5243	0.4060	0.3849	0.3849	0.3148	0.3271	0.3271
	No Regularization	0.5111	0.4777	0.4777	0.3949	0.3827	0.3827	0.2991	0.3092	0.3092

6 Conclusion

From the implementation of the project, it is clear that ‘ELU’ activation performs well as compared to the ‘RELU’ activation function. This is because ELU avoids the dead ‘relu’ problem. Moreover, It is observed that ‘No regularization’ has lower accuracy. And Only Dropout has lesser accuracy over others. Thus, I used the combination of Dropout and BatchNormalisation for every problem statement.The combination of Batch Normalisation and Dropout outperform in accuracy over the Batch Normalisation only and No Regularisation. Furthermore, I achieved more accuracy for VGG16 over Resnet18 and InceptionV2.

6 References

- **https://towardsdatascience.com/architecture-comparison-of-alexnet-vggnet-resnet-inception-densenet-beb8b116866d**
- **https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96**
- **https://github.com/geifmany/cifar-vgg/blob/master/cifar100vgg.py**
- **https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c**
- **https://neurohive.io/en/popular-networks/vgg16/**
- **https://www.geeksforgeeks.org/vgg-16-cnn-model/**
- **https://github.com/priya-dwivedi/Deep-Learning/blob/master/resnet_keras/Residual_Networks_yourself.ipynb**
- **https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202**
- **https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43022.pdf**
- **https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/**
- **https://github.com/deep-diver/DeepModels**