CSE 676 – Deep Learning (Fall 2020)

Project One

University at Buffalo, The State University of New York

Buffalo, NY 14620

UBIT: pranayre

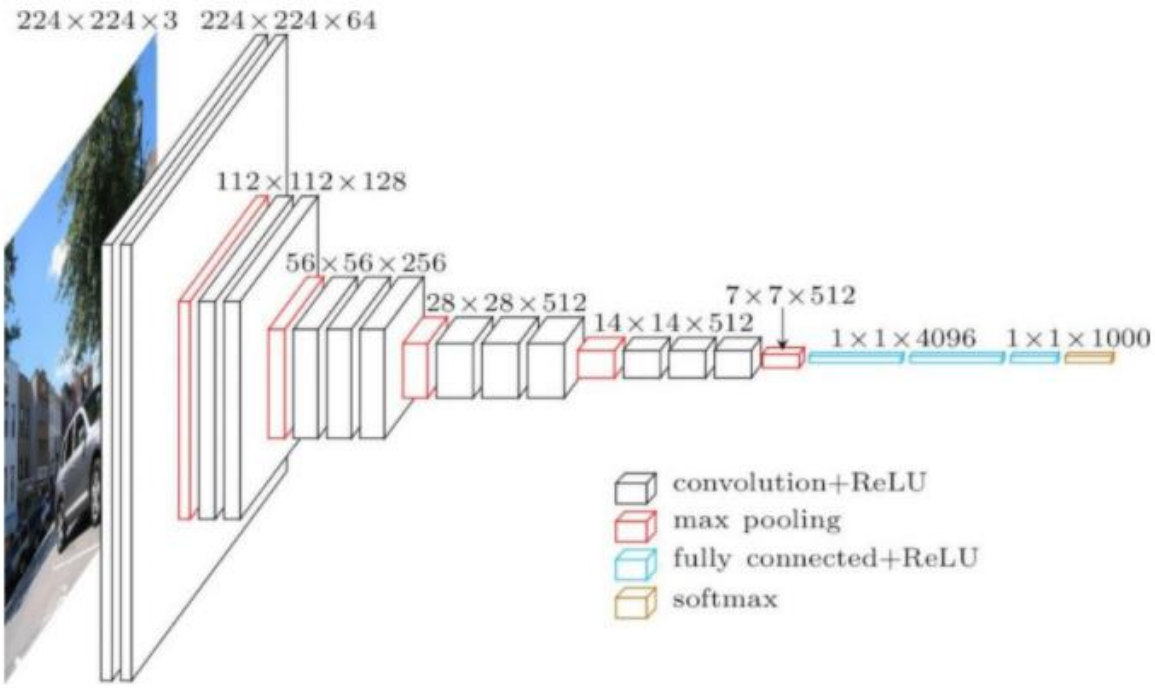
Introduction:

In Machine learning, Convolution Neural Network (CNN) architectures are used to solve many complex problems like image recognition, video analysis, natural language processing. The main aim of this project is to compare three different CNN architectures namely VGGNet, ResNet, InceptionNet with different optimization techniques namely SGD (Stochastic gradient descent), Adam optimization and different regularizations for image classification. The dataset used for this project is CIFAR 100 which consists of 60000 images with 100 different categories. The CIFAR-100 dataset consists of 60000 32x32 colour images in 100 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The CIFAR-100 is a labelled subset of the 80 million tiny images dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

VGG16:

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”.

Architecture:



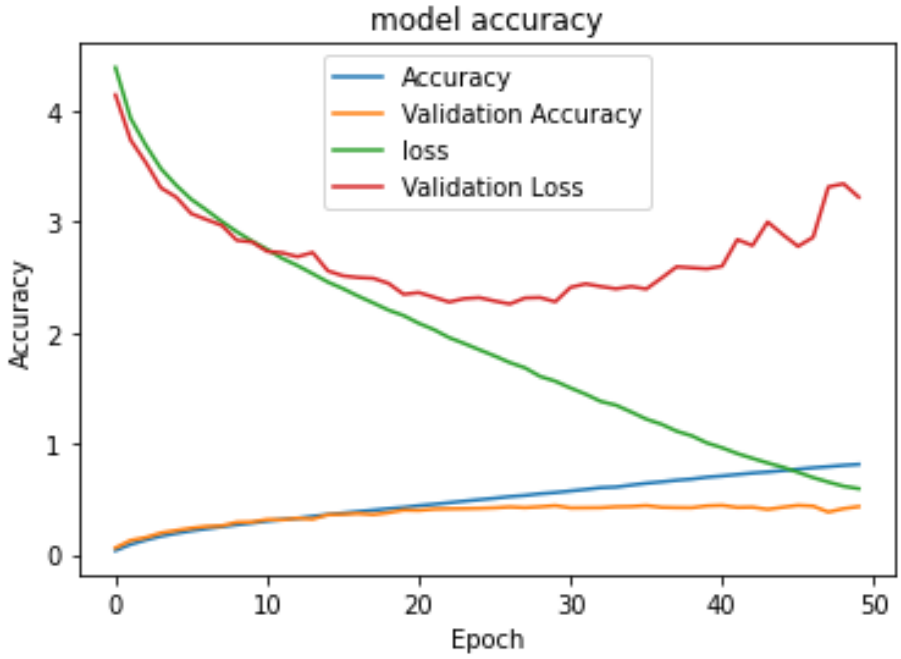
VGG16 Architecture

In this Architecture we can see that the input to the first convolution layer is fixed of fixed size (224,224) RGB image but CIFAR100 dataset consists of (32,32) size images. So, I have changed the input size to be fixed of (32,32). Three Fully-Connected (FC) layers follow a stack of convolutional layers, the first two have 4096 channels each, the third performs 1000 category classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. To this FC layers I have changed it to be 100 because of 100 classes dataset.

Implementation:

Using VGG16 I have tried 6 different models with 2 optimizers (SGD and Adam) and with No regularization, Batch Normalization and Drop out regularization.

VGG16 with SGD and NoRegularization:



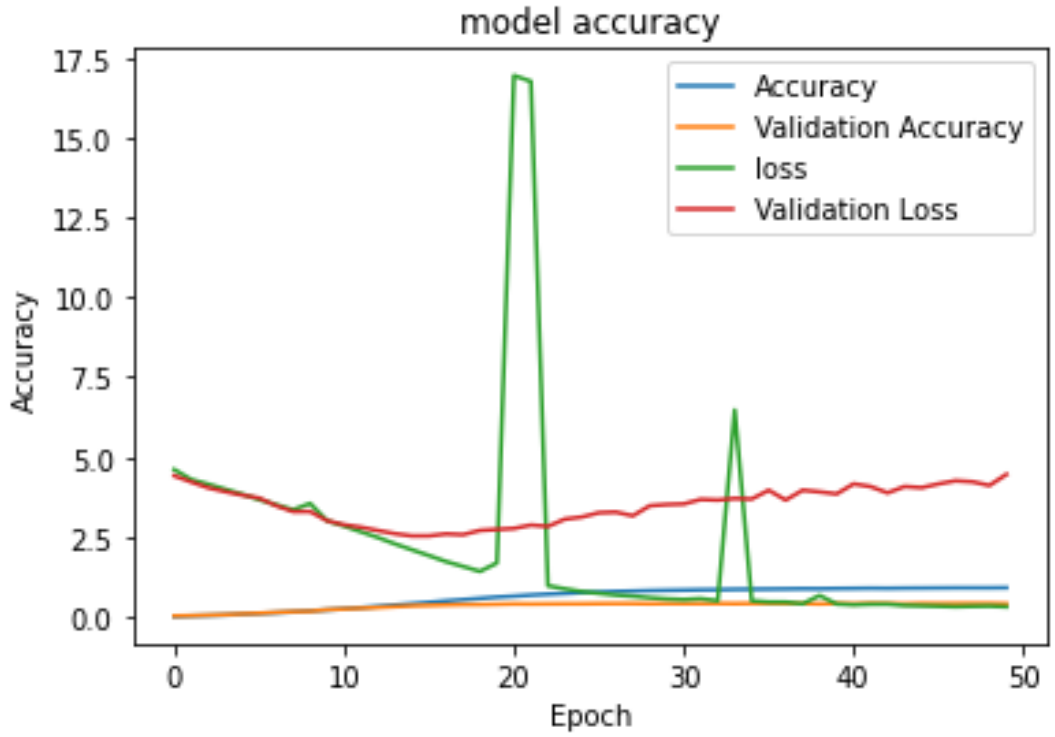
During my implementation of VGG16 plain version, I have noticed the dead weights in the architecture that is there is no increase in the accuracy and no decrease in the validation loss. This may be because of the activation function 'ReLu' because it just produces zero if the value is negative. Instead of changing the activation function, I have provided some momentum to the optimizer so that whenever the gradient becomes 'zero' it provides some momentum and the weights get updated accordingly. I have initialized weights according to He Normal() to prevent layer activation outputs from exploding or vanishing during forward propagation in the network. I have used gradient clipping as well to prevent gradient problem so that the gradient value stays in the range provided in the '+, - clipvalue'. Next observation is the overfitting of the training data, this is as expected because the Neural Network is so complex and dense, it tries to learn every feature in the training data. To overcome this, I have done Data Augmentation in order to provide more data for training. By doing this over fitting is decreased up to some extent.

I have achieved 0.434 Accuracy in 50 epochs by this implementation.

Results:

```
Prec: 0.46998147346974545
Recall: 0.434
Accuracy: 0.434
```

VGG 16 with Adam and No Regularization:



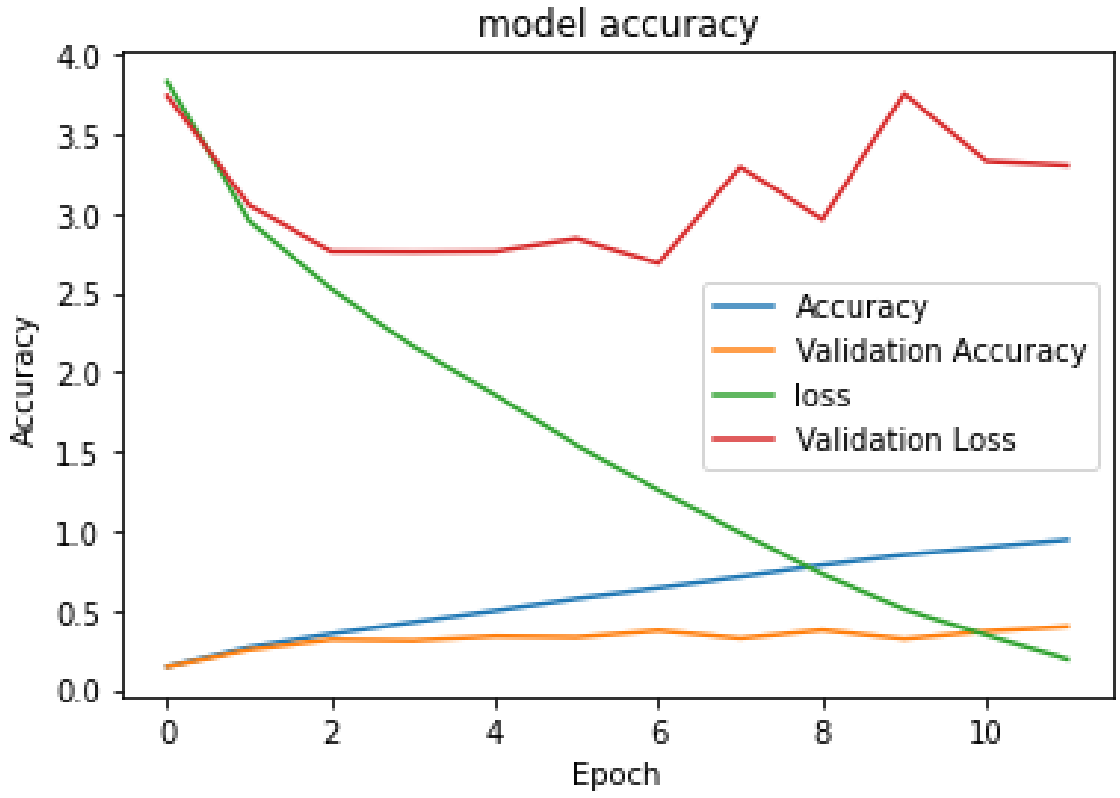
During the training process of this network with Adam optimizer, I have noticed that it does not require any tuning of hyperparameters for the gradient vanishing or exploding problem because the Adam optimization is designed in such a way that it computes different learning rates for different parameters. It uses first and second moments of gradients to adapt the learning rate. Adam Optimization is nothing but the combination of SGD and momentum so the momentum initialization is not required for Adam. Also, I have observed the results to be same with SGD (with gradient clipping and momentum) without any weight initialization and without any gradient clipping. Also, no data Augmentation is done during the training process so the data got over fitted during training.

Achieved 0.4108 accuracy with this implementation.

Results:

```
Prec: 0.4557957523455909
Recall: 0.4108
Accuracy: 0.4108
```

VGG16 with SGD and BatchNormalization:

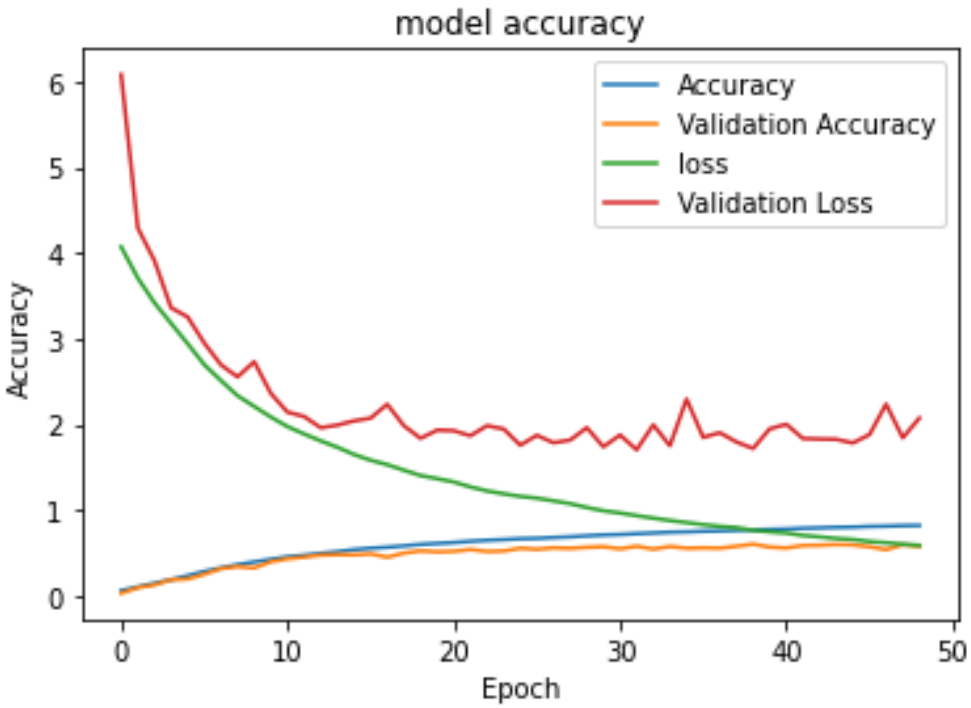


During the training process of this model I observed the increase in the speed of the network training process. This is done without any Data Augmentation so I can observe there is overfit of data. There is no increase in the Accuracy of the model but the training process speed got increased. I am sure that if we use data augmentation there would be increase in the accuracy. I just want to differentiate the speed of the training process. Also, I can see that the 0.3957 accuracy is achieved in just 12 epochs where as No regularization models took over 50 epochs for training to achieve the same accuracy.

Results:

```
Prec: 0.4507767478540414
Recall: 0.3957
Accuracy: 0.3957
```

VGG16 with ADAM and BatchNormalization:

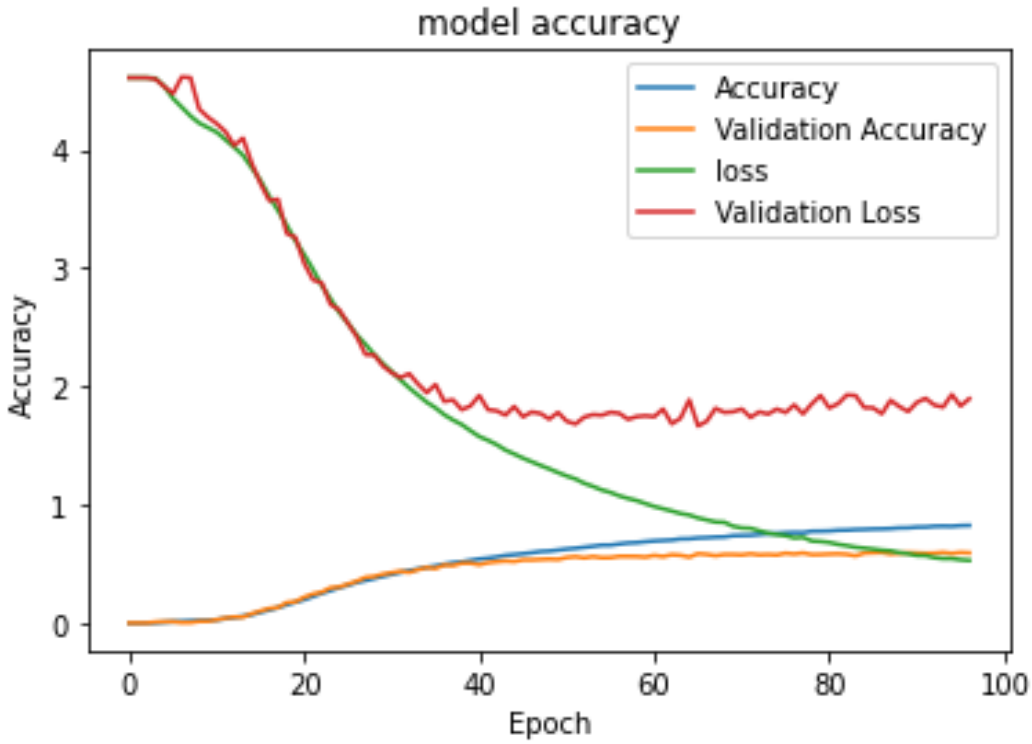


During the training of this model I observed the dead weights again even though I was using Adam optimization, I can say this is because of 'ReLu' activation function because it produces zero output when the input is less than zero. To overcome this, I changed the activation function to 'Elu' with alpha constant as 0.1. As soon as I used 'Elu' I noticed the training process got started and there is an increase in the accuracy. Elu is nothing but the modified form of 'ReLu', Elu produces some value when the input is less than zero so we can overcome the vanishing gradient problem. Achieved 0.6047 accuracy in this model.

Results:

```
Prec: 0.6314587318430532
Recall: 0.6047
Accuracy: 0.6047
```

VGG with SGD and DropOut:



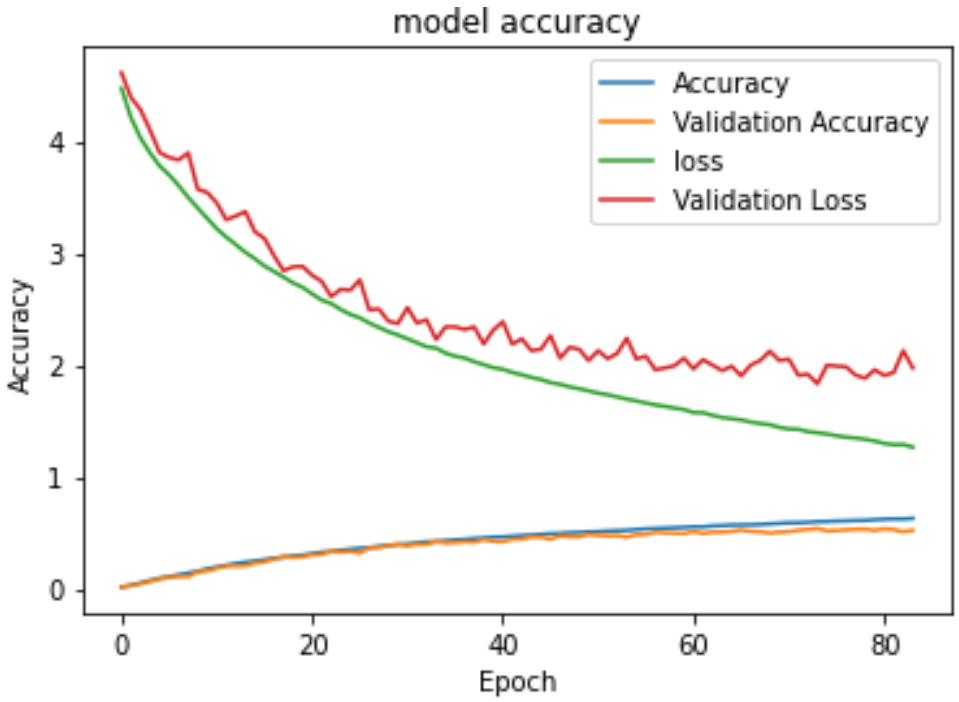
DropOut is a regularization technique where randomly selected neurons are ignored during training process that is the output of those neurons become 'zero'.

In this model I have not used the data augmentation because 'DropOut' is the technique used to overcome the overfitting of data. Also, I used 'LeakyRelu' instead of 'ReLu' because I observed that weights are not being updated that is may be because of gradient vanishing issue. When I used 'LeakyRelu' I observed the increase in accuracy. Also, there is an overfit of data with 0.80 as training accuracy but not high as compared to 0.99. Achieved 0.6044 accuracy at 97 epoch as early stopping.

Results:

```
Prec: 0.6111953935180988
Recall: 0.6044
Accuracy: 0.6044
```

VGG16 with Adam and DropOut:

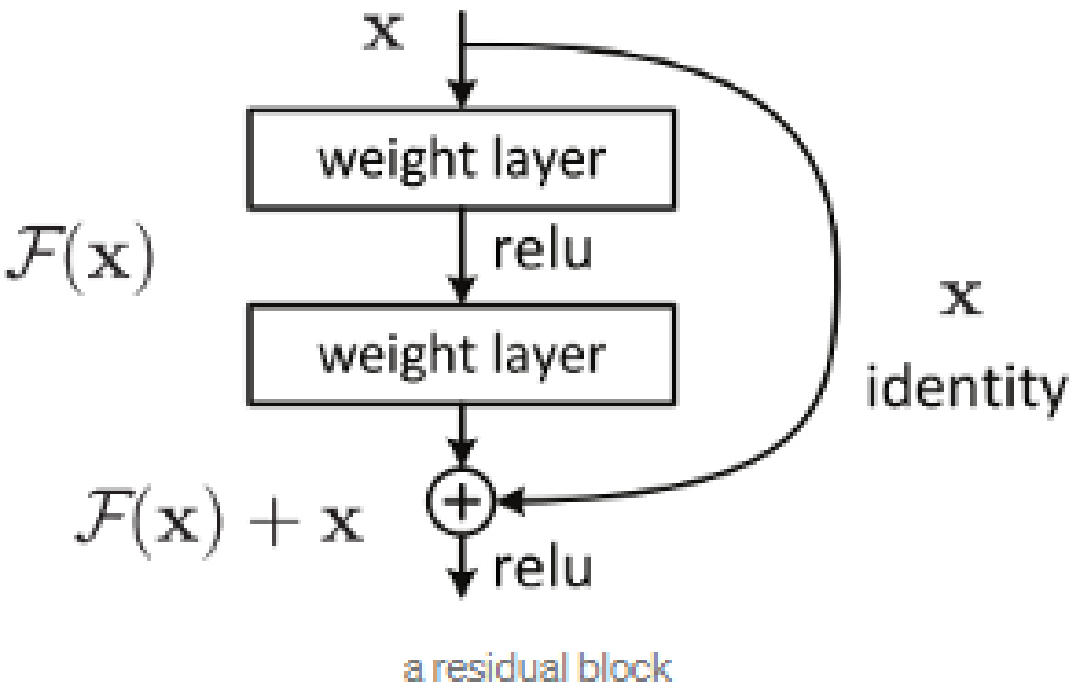


As there was overfit of the data in previous model, I used data augmentation for this model in order to increase training data. In this we can observe there is consistent decrease in loss and validation loss. There is no overfit of data even though it ran for 84 epochs as early stopping criteria. Achieved 0.5049 accuracy in this implementation.

Results:

```
Prec: 0.563649189680288
Recall: 0.5409
Accuracy: 0.5409
```

Residual block:

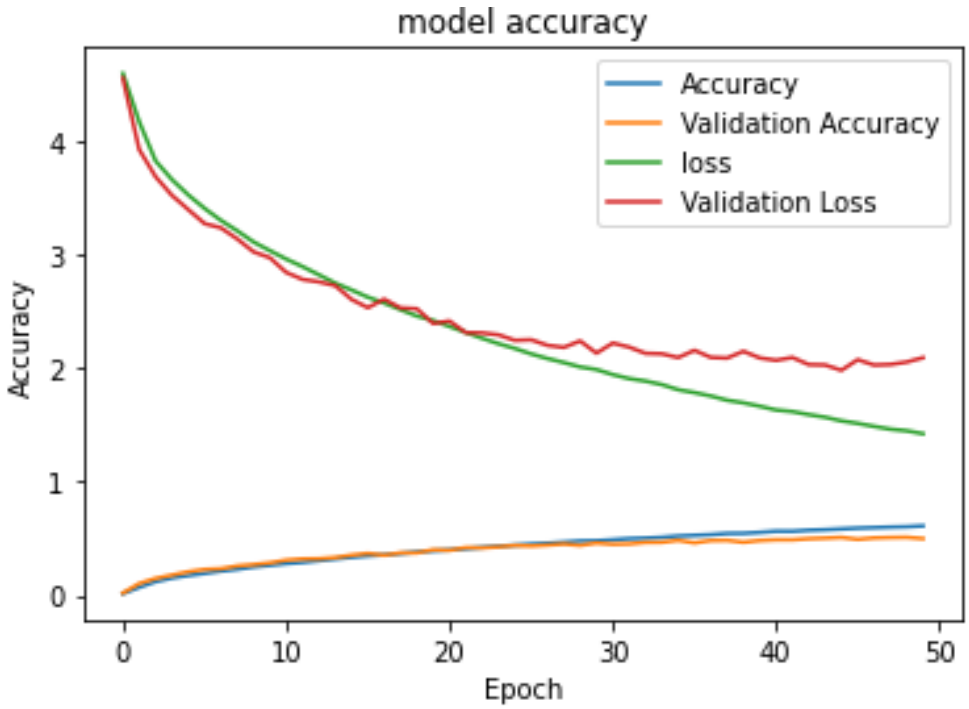


a residual block

The idea of ResNet is to introduce the "identity shortcut" that skips one or more layers as shown above. The above block is known as the residual block. The idea behind this architecture is that if the layers are stacked in the conventional neural networks, the performance gets degraded. Instead one can use such type of residual blocks instead of stacking layers which achieve the same outcome. It is much easier to get identity mapping in this type of networks.

In the same way I have implemented 6 different models with SGD, Adam optimization and with NoRegularization, BatchNormalization and DropOut.

ResNet-18 with SGD and NoRegularization:

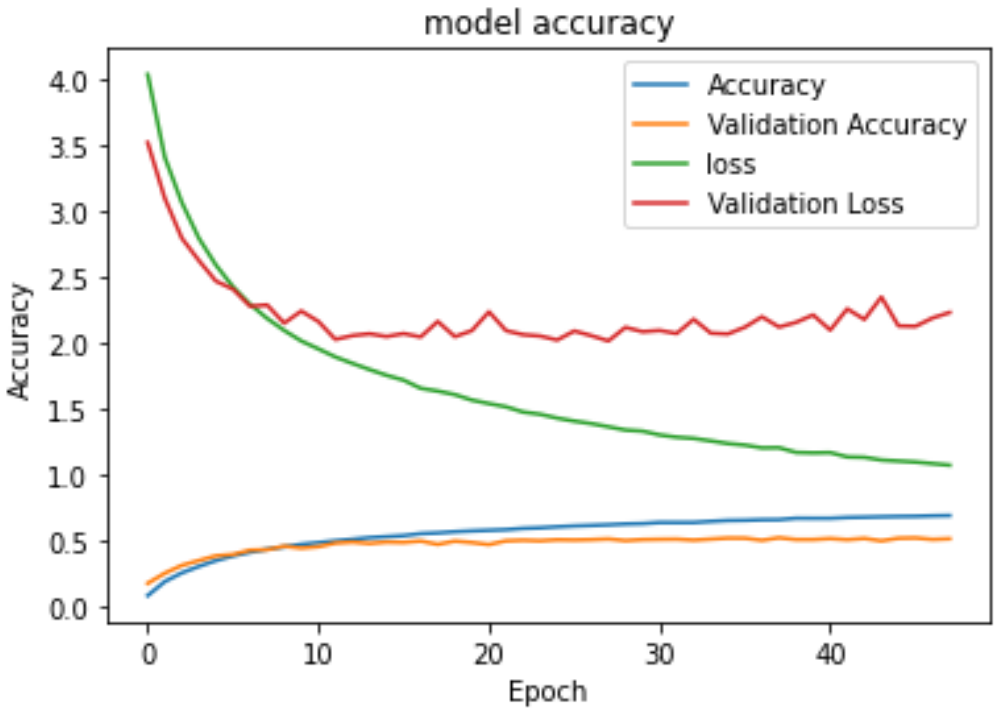


I have used data augmentation and 'Relu' as activation in this model, the model has been implemented same as the ResNet-18 and I could observe no overfit of the data and achieved an accuracy of 0.4986 in 50 epochs. I have decreased the neurons to half of the actual ResNet-18 as I thought the model becomes too complex for the data present here. Mostly no changes were done to the architecture.

Results:

```
Prec: 0.5231731225076414
Recall: 0.4986
Accuracy: 0.4986
```

ResNet18 with Adam and NoRegularization:

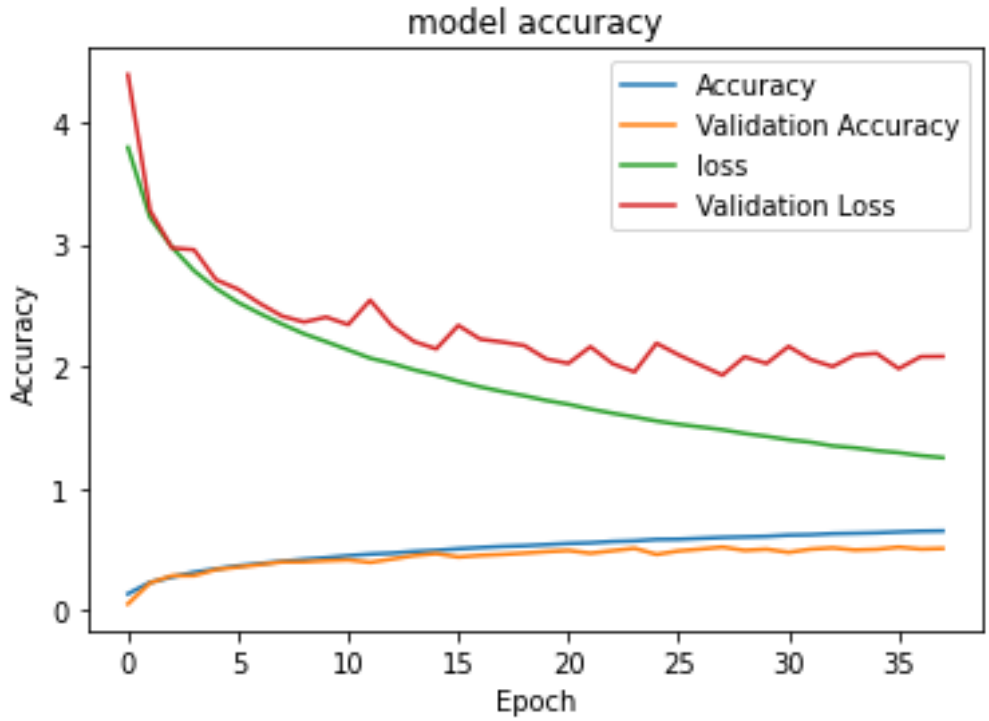


This model performed same as the SGD model as we could see every trend follows the SGD. No changes were done to the architecture. The only change done is same as to SGD, the number of neurons have been reduced to half of its actual number of neurons. Achieved 0.5248 accuracy in 48 epochs with early stopping.

Results:

```
Prec: 0.5466103215558665
Recall: 0.5248
Accuracy: 0.5248
```

ResNet18 with SGD and BatchNormalization:

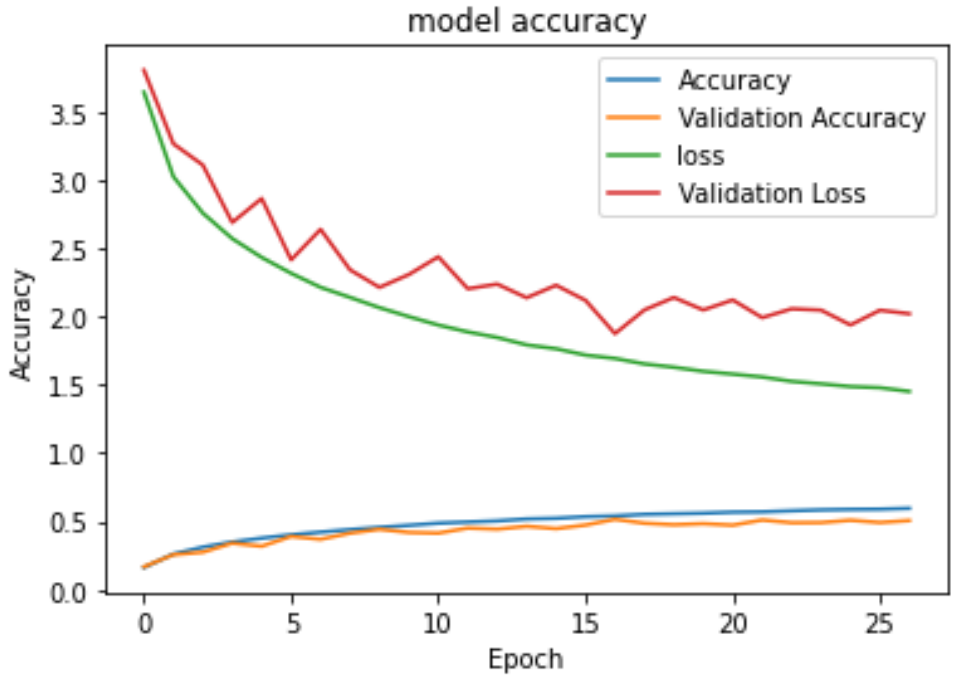


In this model Batch Normalization is used after every convolution layer. The training speed got increased in this model . There is no much difference in between these models. Achieved 0.5172 accuracy in 37 epochs with early stopping.

Resluts:

```
Prec: 0.5387424025261487
Recall: 0.5172
Accuracy: 0.5172
```

ResNet18 with Adam and BatchNormalization:

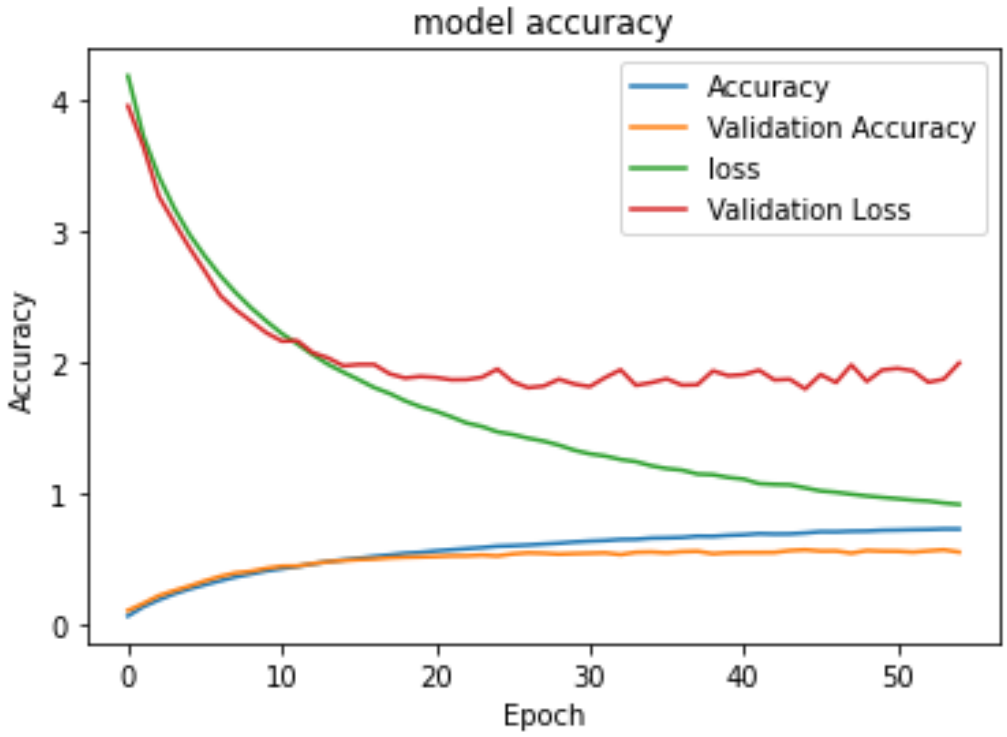


In this model, instead of 'RelU' I used 'Elu' with default alpha constant as activation function because of gradient vanishing problem as the weights were not being updated. Batch Normalization is used after every Covolution layer. We can observe that the model performed same as the above model with no overfitting and good amount of accuracy as well. Achieved 0.5157 accuracy in this model in 27 epochs with early stopping.

Results:

```
Prec: 0.5326537138214419
Recall: 0.5157
Accuracy: 0.5157
```

ResNet-18 with SGD and DropOut:

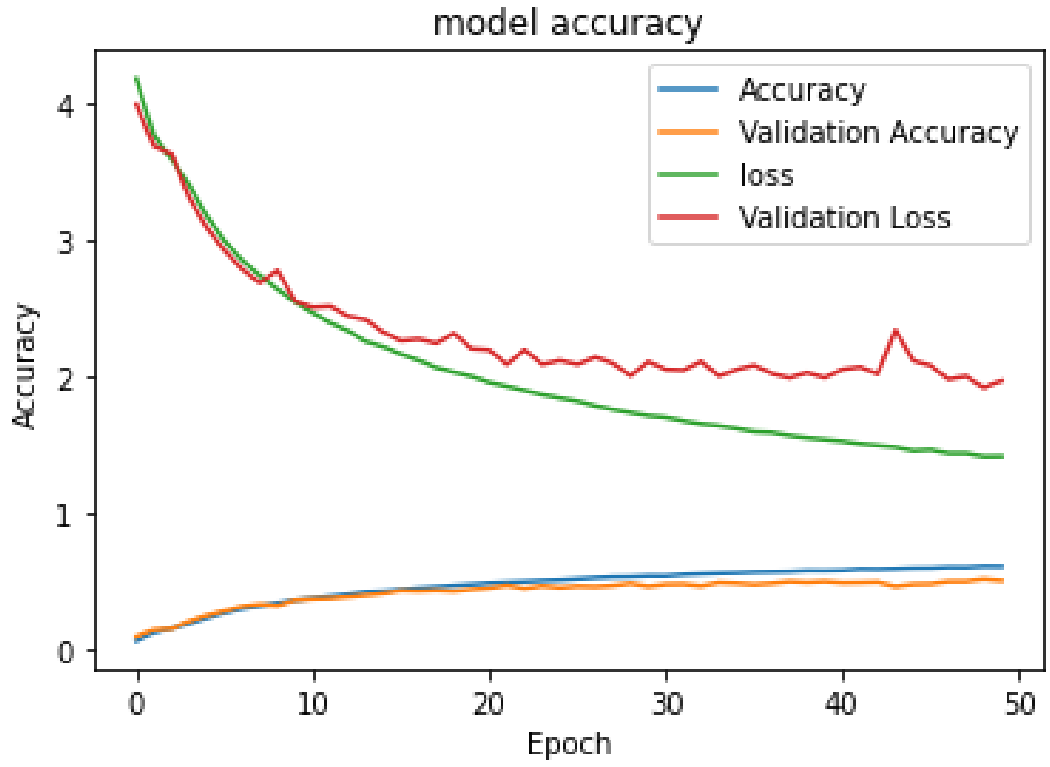


No changes were done during its implementation. Data Augmentation is used for increasing training data. Early I have implemented this model with Dropout layers after every convolution layer, then I have observed there is no increase in the accuracy. So, I have dropped some DropOut layers in between, then I observed there is increase in the accuracy over the epochs. Achieved 0.57 accuracy in 55 epochs with early stopping.

Results:

```
Prec: 0.5845594954864525
Recall: 0.57
Accuracy: 0.57
```

ResNet-18 with Adam and DropOut:

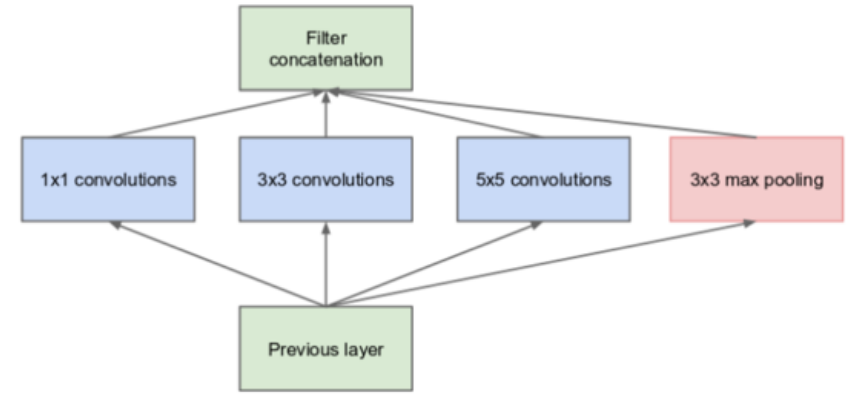


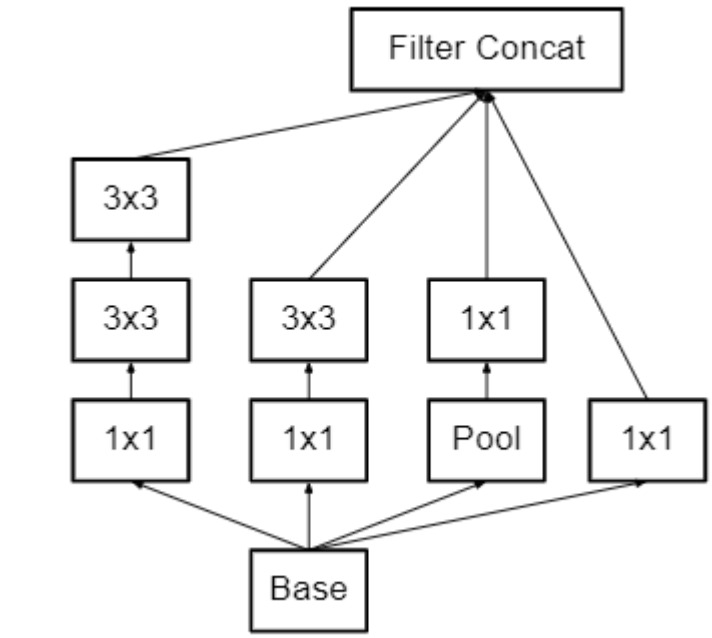
Even though DropOut is used in this model initially there was no increase in the accuracy that is there was no update in the weights. I have reduced the units to half of its actual number of units and implemented Data augmentation in the training, then I could see increase in the accuracy and there is no over fit of the training data. Achieved 0.5045 accuracy in 50 epochs.

Results:

```
Prec: 0.52100732250736
Recall: 0.5045
Accuracy: 0.5045
```

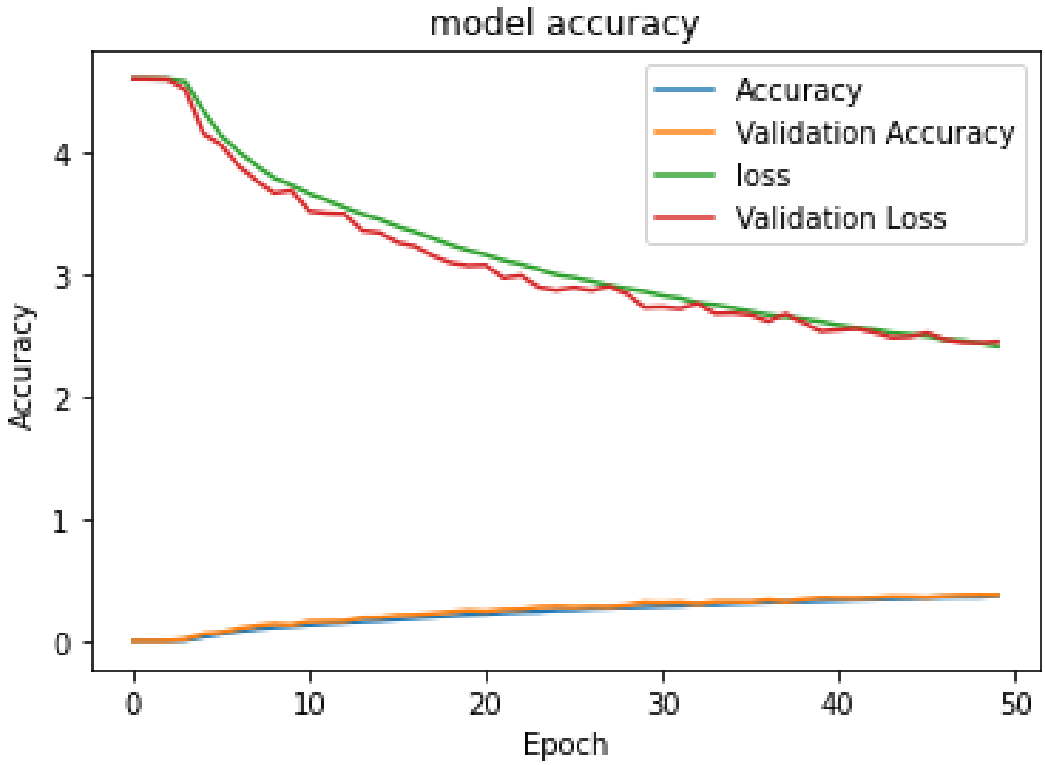
Inception:





Inception layer is the combination of (1,1) convolution layer, (3,3) convolution layer, (5,5) convolution layer with their outputs concatenated into a single output vector forming the input to the next layer. In Inception V2 the (5,5) convolution layer is replaced with two (3,3) convolution layers. These layers automatically pick the filter to learn the required features presented in the image because the dimension of the filter is very important to extract features from the image. In this way Inception layers learn the size of filters to be used based on the features present in the image. The computation cost is also reduced by this way of implementation than the conventional Neural networks with same type of learning activity.

Inception with SGD and NoRegularization:

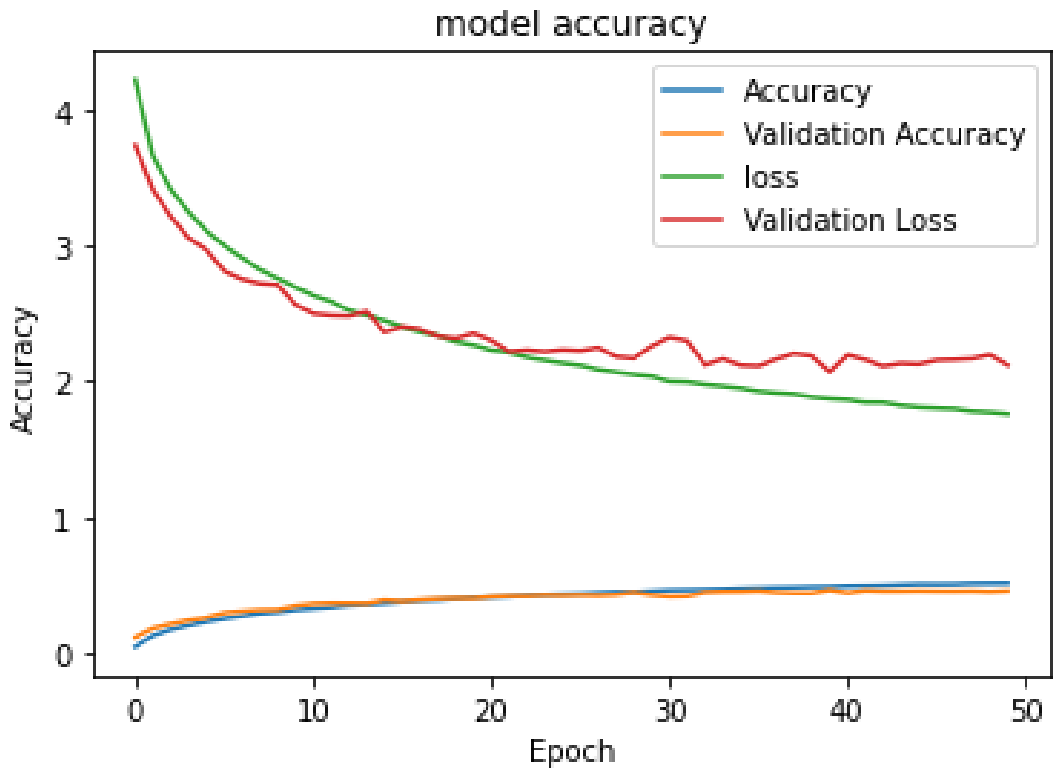


This model has been implemented with learning rate as 0.001 and momentum 0.9 because I observed no change in weights without momentum. I used 'ReLu' activation function also augmented the data. I could see there is no substantial increase in the accuracy. Achieved 0.3845 in 50 epochs. May be that would have been increase if I ran it for more number of epochs because there was no early stopping in the training process.

Results:

```
Prec: 0.4059654250597314
Recall: 0.3845
Accuracy: 0.3845
```

Inception with Adam and NoRegularization:

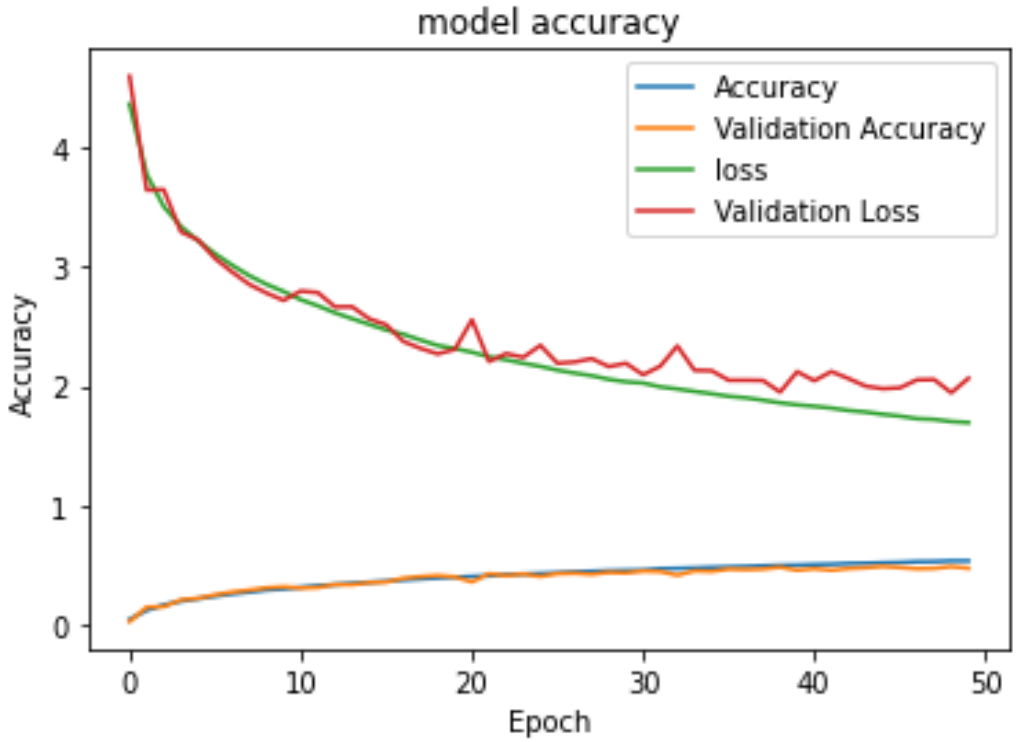


Initially I noticed no increase in the accuracy, then I used gradient clipping for this model to overcome the issue. In some of the epochs I observe the test accuracy to be more than the training accuracy but I was not able to figure out what was the cause for it but over the period of epochs the training accuracy dominated the testing accuracy but there was no overfit of data in this model even though we have not used any regularizer. Achieved 0.4681 accuracy over 50 epochs with early stopping.

Results:

```
Prec: 0.4759762613477358
Recall: 0.4681
Accuracy: 0.4681
```

Inception with SGD and BatchNormalization:

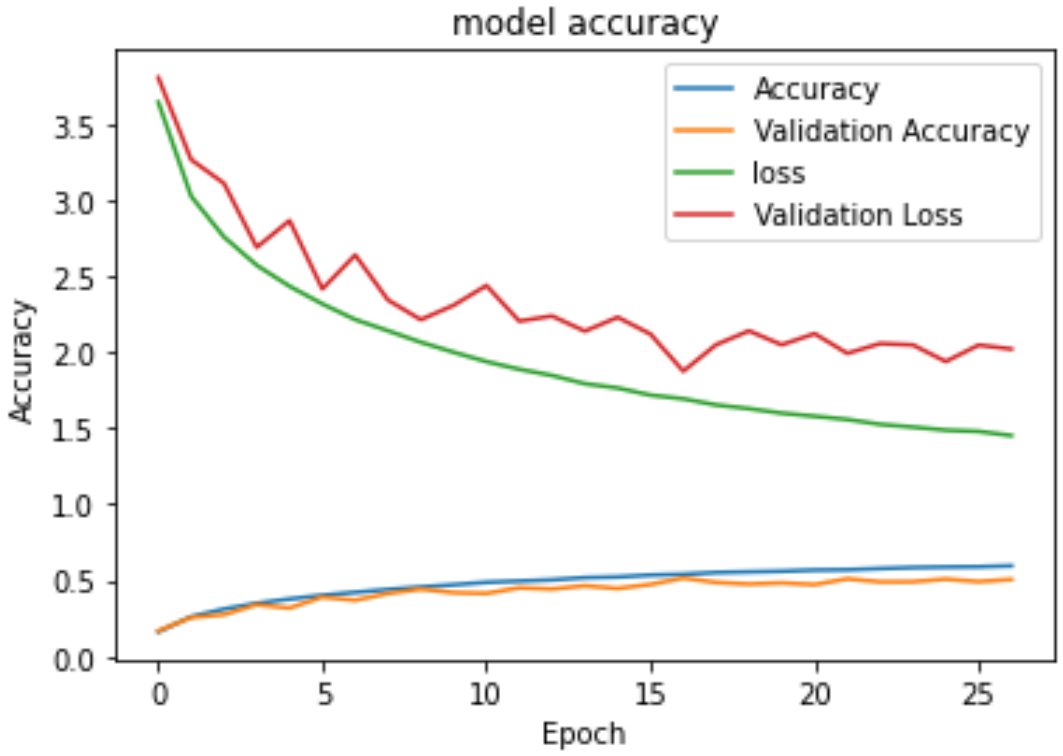


The BatchNormalization did not help much in this model because I got almost the similar results to the above models also there was no increase in the training speed. I used the same 'ReLu' as the activation function.

Results:

```
Prec: 0.5259592575257784
Recall: 0.4733
Accuracy: 0.4733
```

Inception with Adam and BatchNormalization:

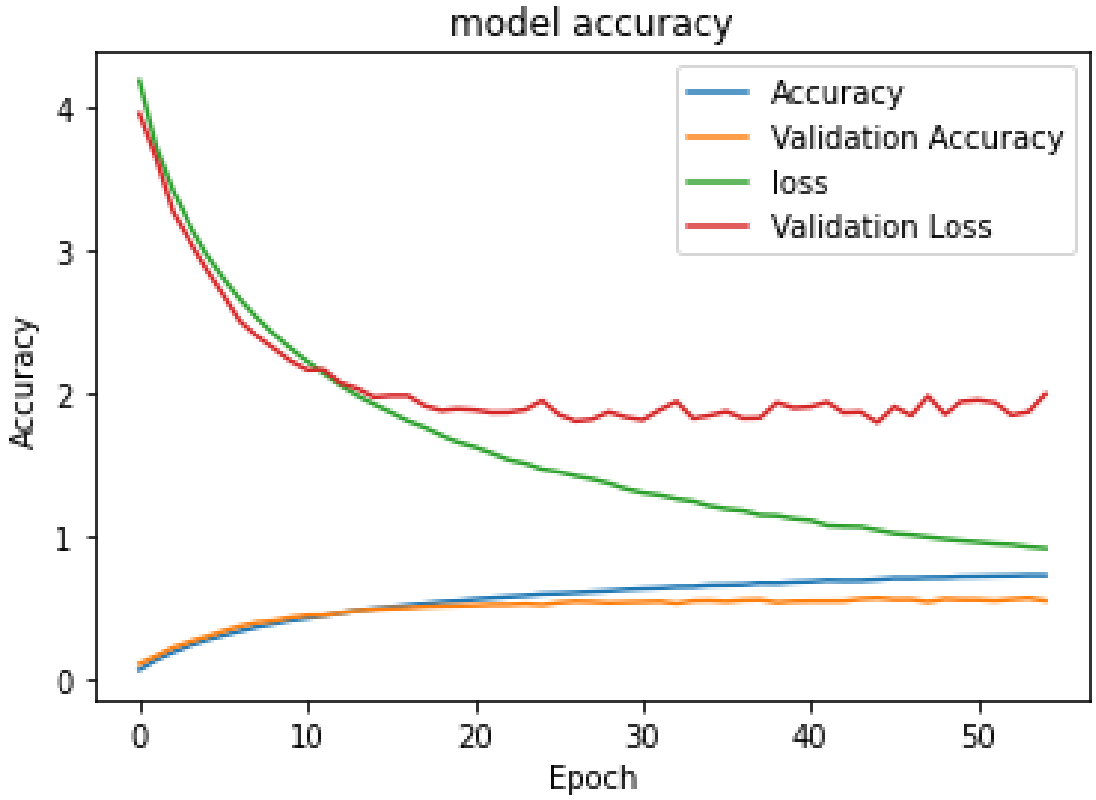


Even in this model I observed similar results but instead of 'ReLu' I used 'Elu' activation function to over the gradient issue and I observed there is increase in the accuracy of the model. In this Batch Normalization is used after every convolution layers. Achieved 0.5157 accuracy over 27 epochs with early stopping. In this there is slight increase in the training speed. So maybe the 'Elu' with Batch Normalizaion together helped in increasing training speed as some of the outputs will not be zero so that the model will be able to extract features easily.

Results:

```
Prec: 0.5326537138214419
Recall: 0.5157
Accuracy: 0.5157
```

Inception with SGD and DropOut:

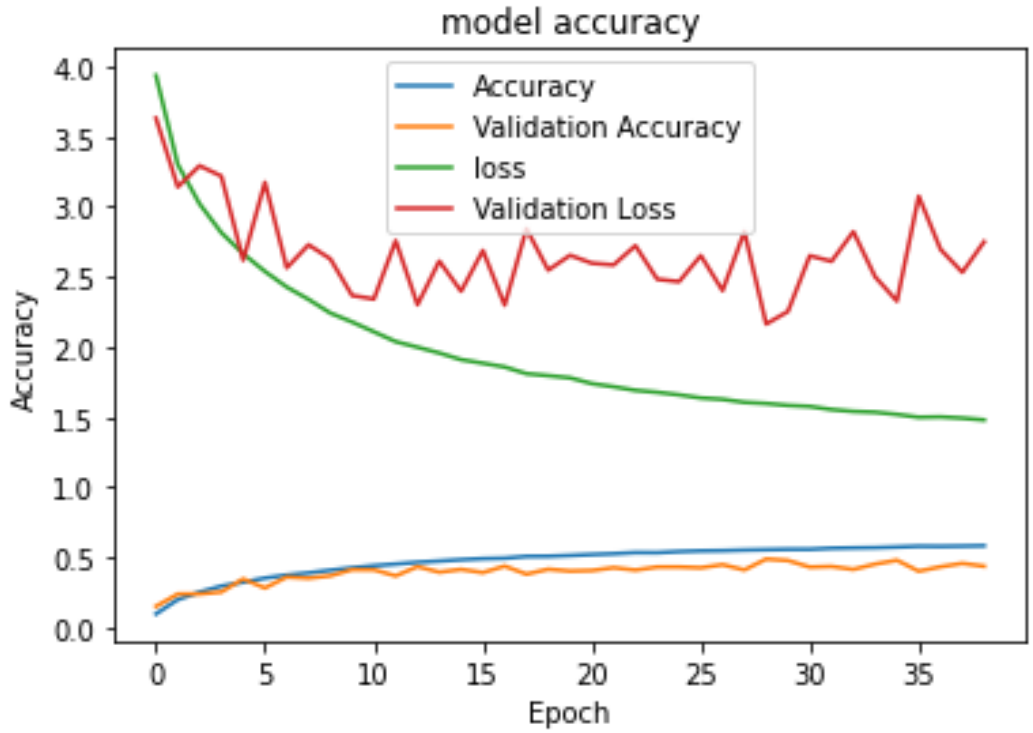


In this model I have used DropOut layer after every convolution layer and I also augmented the training data. There is also no overfit of the data in the model. I used the normal 'Relu' as activation function and could see increase in accuracy. Achieved 0.57 accuracy in 55 epochs with early stopping.

Results:

```
Prec: 0.5845594954864525
Recall: 0.57
Accuracy: 0.57
```

Inception with Adam and DropOut:



Initially I observed the validation accuracy to be only 0.01 over few epochs when I used 'ReLu' as activation function. Instead of 'ReLu' I used 'LeakyReLu' then the accuracy has been increasing over the period of epochs. I got to know because of the dead neurons the model behaved in that way. Achieved 0.4899 accuracy in 39 epochs with early stopping. I can also observe the increase in raining speed in this model.

Results:

```
Prec: 0.5755537106710966
Recall: 0.4899
Accuracy: 0.4899
```

Validation results:

	Arch	VGG_16			ResNet_18			Inception_v2		
Optimizer	Score / Setting	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy
SGD	With_BatchNorm	0.45	0.39	0.39	0.53	0.51	0.51	0.52	0.47	0.47
	With_DropOut	0.61	0.6	0.6	0.58	0.57	0.57	0.58	0.57	0.57
	No_Regularization	0.47	0.43	0.43	0.52	0.49	0.49	0.4	0.38	0.38
ADAM	With_BatchNorm	0.63	0.6	0.6	0.53	0.51	0.51	0.53	0.51	0.51
	With_DropOut	0.56	0.54	0.54	0.52	0.5	0.5	0.57	0.48	0.48
	No_Regularization	0.45	0.41	0.41	0.54	0.52	0.52	0.47	0.46	0.46

Conclussions:

- Batch Normalization increases the training speed.
- DropOut regularization prevents overfitting.
- No regularization models without tuning hyper parameters can lead to dead neurons.
- Sometimes 'ReLu' activation function cause 'zero'(Null) outputs from the layers.
- Batch Normalization and DropOut regularization does not have much effect on Inception V2.
- Overfitting will be observed if the network is too complex.
- Finally, tuning hyper parameters is required in any deep neural networks.

References:

- https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035
- https://www.cs.colostate.edu/~dwhite54/InceptionNetworkOverview.pdf
- https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/
- http://d2l.ai/chapter_convolutional-modern/resnet.html