Deep Learning-Project-1

Soumith Reddy Chinthalapally

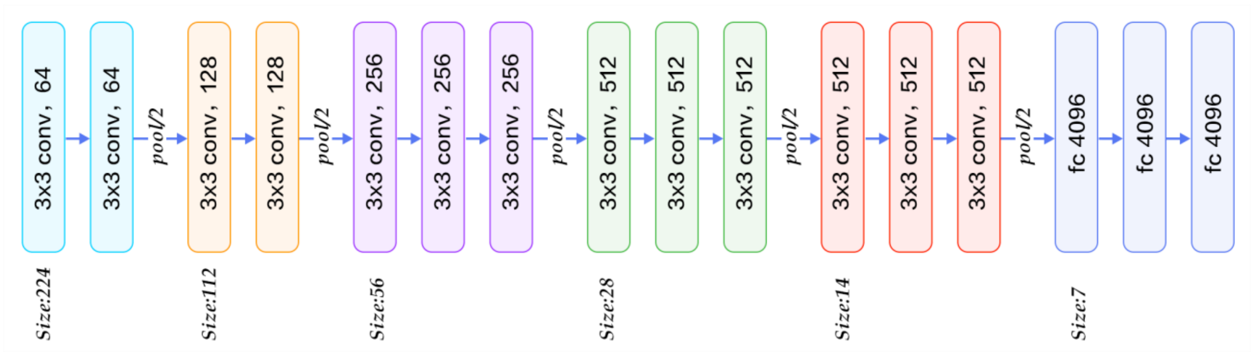
UBIT: soumithr

UB#: 50336650

Introduction

We have implemented VGGNet, ResNet and InceptionNet architectures on CIFAR-100 dataset for image classification. This gives an idea about the existing architectures and how each architecture performs on the dataset. We have created variations of each architecture by implementing various optimization and regularization schemes which helps us understand how each optimization and regularization affects the accuracy of the model. This also gives us an understanding of which variation of model to use depending on the input data and how to increase the accuracy of the model at different stages. Comparing different models gives us an overview of how to use different layers and ways of combining layers to increase accuracy. By viewing the graphs of each model, we get to know how the training accuracy and validation accuracy are varying and how the tradeoff is done between loss and accuracy.

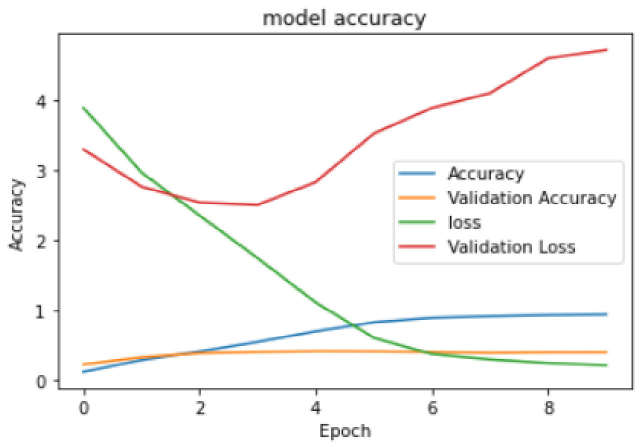
Implementation of VGG 16 [1]



The basic architecture of VGG-16 which I implemented is like the one above except the size of the image is different. I have used the similar convolution layers of 3*3 filter with strides 1 but for max pool layer I used 2*2 filter with strides 1 and 2. The original VGG uses strides of 2 at max pool layers, but the image size is comparatively low hence to get the model learn features, I have used strides 1. This allows the model to train on larger images and with improved training on extracting features. The fully connected layer size is reduced from the original 4096 to 512 as the input size is less and increasing dense layer size makes the model overfit. The output from this dense layer is sent to a dense layer of size 100 with softmax activation to find out the image classification. Early stopping is added to all the models with a patience of 10 and model checker is added to save the best weights.

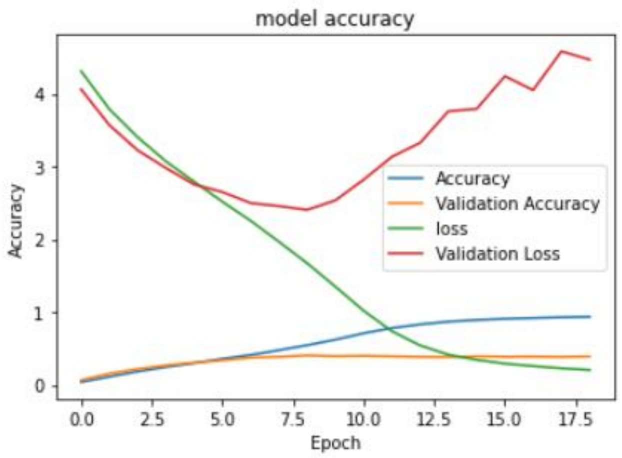
VGG_16_NoRegularization_ADAM

I have implemented the above-mentioned architecture with 'LeakyReLU' activation, as the accuracy with basic relu is low comparatively. Using relu there might be the case that gradient might vanish, with LeakyReLU we can overcome such cases. I have also observed that after replacing relu with LeakyReLU the training time increases when compared to relu as it ignores the negative part completely which is not the case with LeakyReLU. I have observed that the VGG without any regularization the model overfits quickly and hence the validation accuracy is low. As you can see in the below graph the Training accuracy shoots up very fast by 6th epoch. I have used the Adam optimizer with learning rate 0.0001 and clip norm 1. Validation accuracy is 40.71.



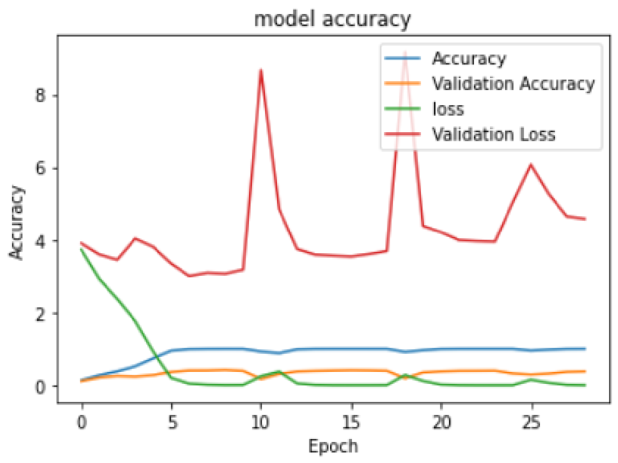
VGG_16_NoRegularization_SGD

The implementation is similar to ADAM except here I am using SGD instead of ADAM.I have added momentum to the optimizer with a value of 0.9. This helped in making the model train faster since SGD alone takes time to train, SGD along with momentum converges faster. Other than time constraint the training accuracy and validation accuracy vary like Adam. Validation accuracy is 42.73.



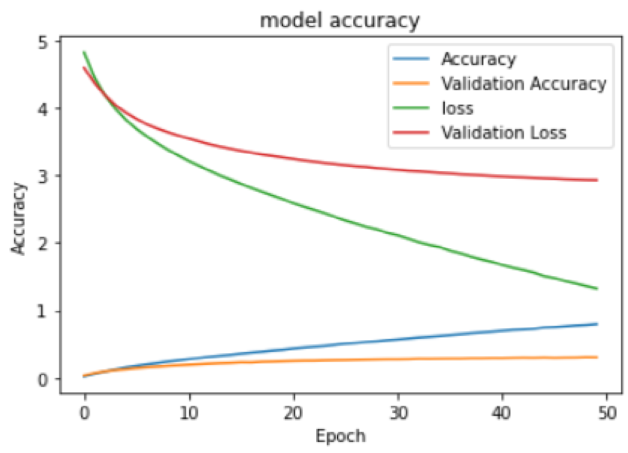
VGG_16_WithBatchNorm_ADAM

I am adding Batch Normalization layer to the initially implemented VGG architecture. In a convolution layer we can add Normalization at two places one before activation and another way is after activation. Both ways the accuracy is better compared to basic VGG but in comparison Batch normalization before activation (LeakyReLU) worked well for me. The same has been applied to Dense layers. One of the reason for this can be since Batch Normalization is used between the linear and non-linear layers in the network, it normalizes the input to the activation function, so that it is centered in the linear section of the activation function. In this case also model overfits but it takes some time to overfit and validation accuracy is more comparatively with Normalization. Validation accuracy is 62.32.



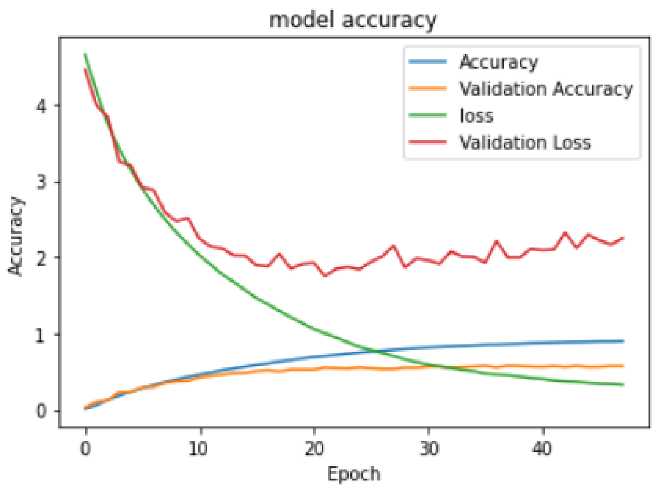
VGG_16_WithBatchNorm_SGD

I have implemented Batch before activation (LeakyReLU) in convolution layers and Dense layers. But the accuracy is low when compared to Batch Norm with ADAM. The accuracy did not improve much, and the model is still overfitting, hence I have added Data Augmentation which helps in increasing the accuracy of the model. Post adding the data augmentation the accuracy has increased considerably. Validation accuracy is 60.73.



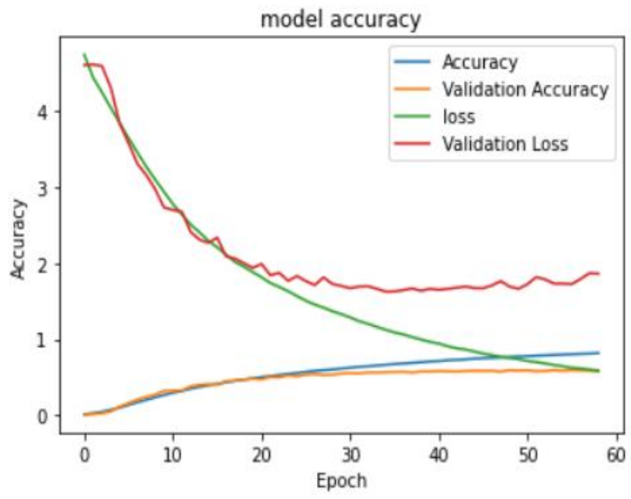
VGG_16_WithDropOut_ADAM

Since our model is overfitting DropOut is another Regularization to reduce overfitting in our model. Dropout is a technique where randomly selected neurons are ignored during training.
I have added the dropouts after each maxpool layer with value of 0.5 and the results show a considerate increase in the accuracy compared to the basic model. Since our model is overfitting, removing a portion of cells does not impact the model much and in turn increases the accuracy.
Validation accuracy is 57.58.

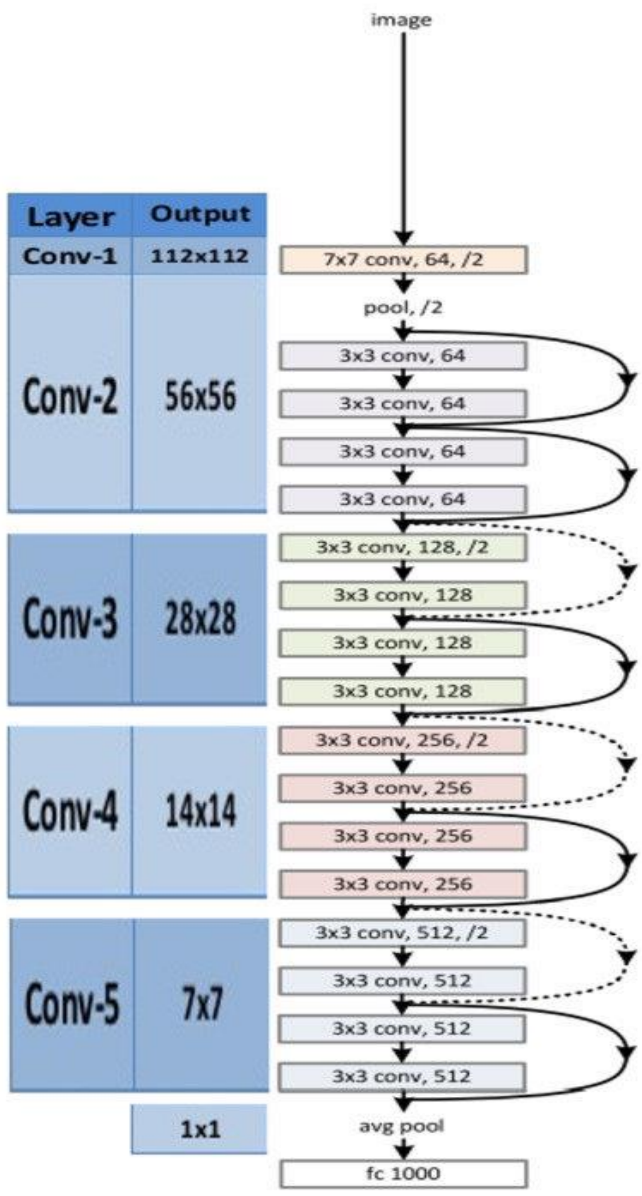


VGG_16_WithDropOut_SGD

Have added dropouts to the basic model after each maxpool layer to reduce the overfitting and the accuracy has increased considerably. Post adding DropOuts the model takes time to overfit as observed from the image it trains till 60 epochs. Validation accuracy is 59.38.



ResNet_18_Implementation [2]

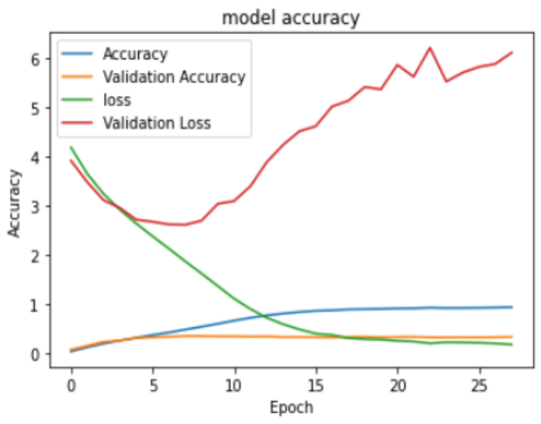


The above architecture is implemented for ResNet_18. The initial 7*7 convolution layer along with the maxpool layer is removed since our image size is small. The remaining convolution layers are implemented similar to the diagram. The initial kernels are of 64 size and then doubled for every 4 convolution layers. To add a convolution layer of 64 kernels to a convolution layer of 128 kernels, I have first increased the kernels of 64 size kernels to 128 and then added so that kernels of the same shape are added. After every four convolution layers I am increasing the stride to 2 which reduces the image size by half. After all the Convolution layers we add an Average pooling layer and then a Dense layer of size 500 with relu activation. The output of this Dense layer is sent to Dense layer of 100 units with softmax activation to get the outputs. Have used 'relu' activation for all the layers and implemented early stopping with model checker. Have implemented Data Augmentation for all the models in ResNet to reduce overfitting and to increase accuracy.

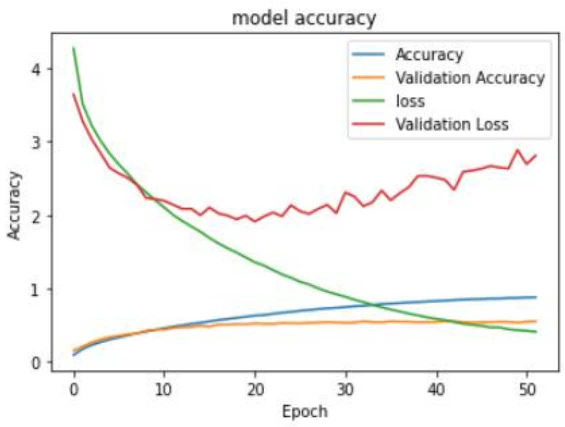
ResNet_18_NoRegularization_ADAM

Have implemented the above mentioned architecture without any regularization and with Data Augmentation. Initially I have started with 32 kernels and doubled the size for every 4 convolution layers and observed that that validation accuracy is low that is the model is unable to extract the features properly with 32 kernels hence increased to 64 kernels which increased the accuracy and in turn decreased validation loss which can be clearly seen in the below images. Validation accuracy is 55.35.

32 Kernel

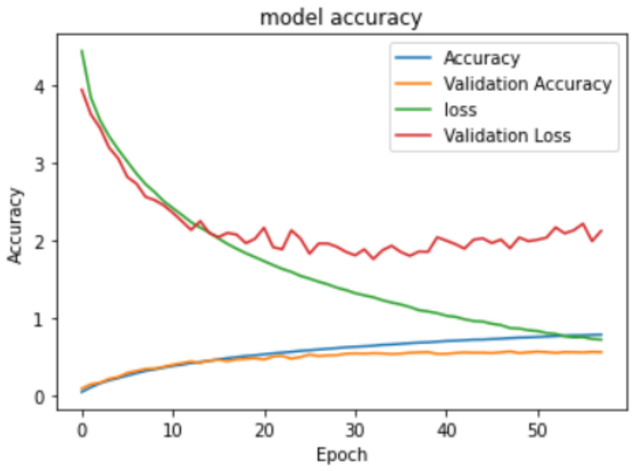


64 kernels



ResNet_18_NoRegularization_SGD

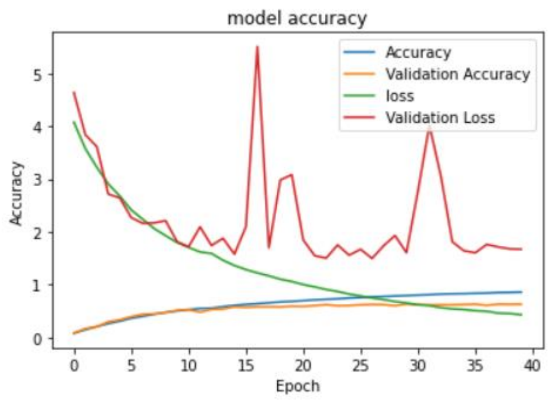
Have implemented a similar kind of architecture as above with SGD and the results are kind of similar except that with SGD initially both the accuracies are kind of equal and then slowly diverges which is not the case with ADAM. Have added clipnorm and momentum to the SGD optimizer. Validation accuracy is 57.4.



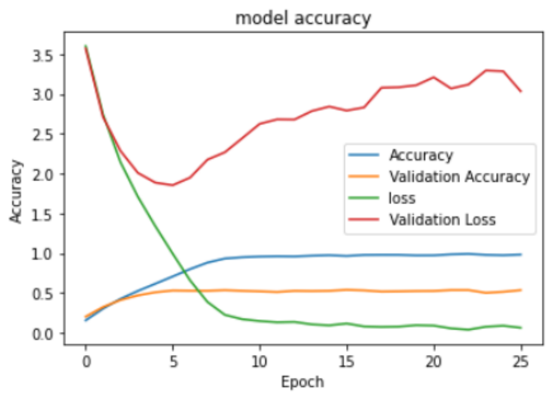
ResNet_18_WithBatchNorm_ADAM

Adding Batch Norm in Convolution Layers after activation, like in VGG did not provide similar results. Although the accuracy increased but not considerably, hence Batch Normalization added post activation function('relu'). This increased the model performance and model did not overfit and both the accuracies improved in a similar way. The below figure shows the variation of the model with and without Data Augmentation. Validation accuracy is 63.58.

With Augmentation

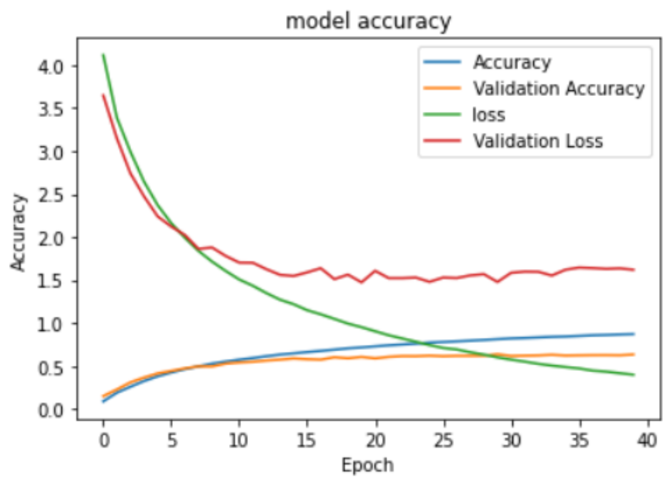


Without Augmentation



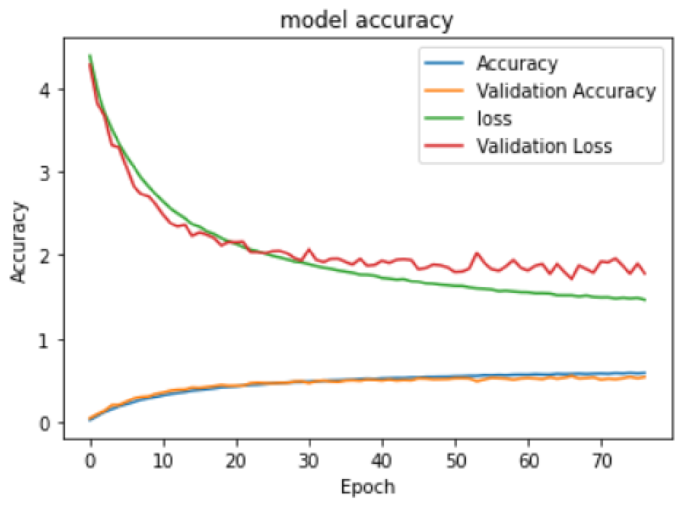
ResNet_18_WithBatchNorm_SGD

Have implemented a similar architecture with similar approach as above with SGD optimizer and the results were intact with the above model. Validation accuracy is 63.7.



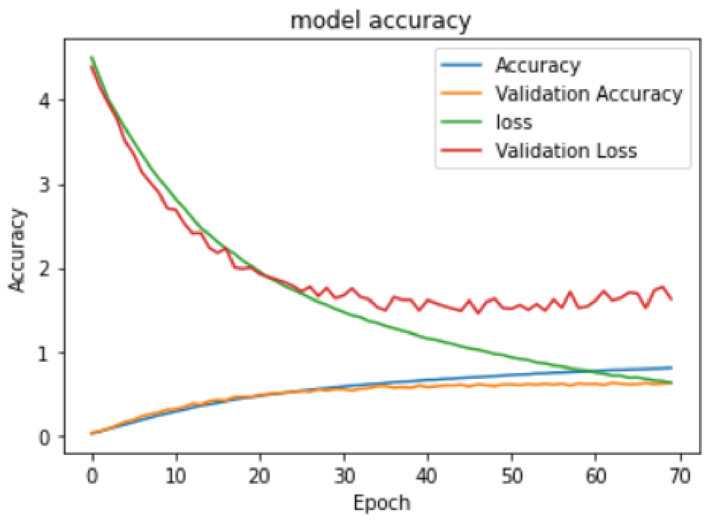
ResNet_18_WithDropOut_ADAM

In this model I have used the Batch Normalization Adam model and added dropouts to it. The Accuracy increased compared to the basic model but did not cross the Batch Normalization model. The model did not overfit either, but with both the regularizers in place the model took comparatively more time to execute. The validation accuracy started increasing to a point and is struck at the point. I observed that accuracy has dropped by using both the regularizers which indicates that using batch normalization the model is perfect and I believe that dropping some cells randomly deteriorating the model performance. Validation accuracy is 55.51.

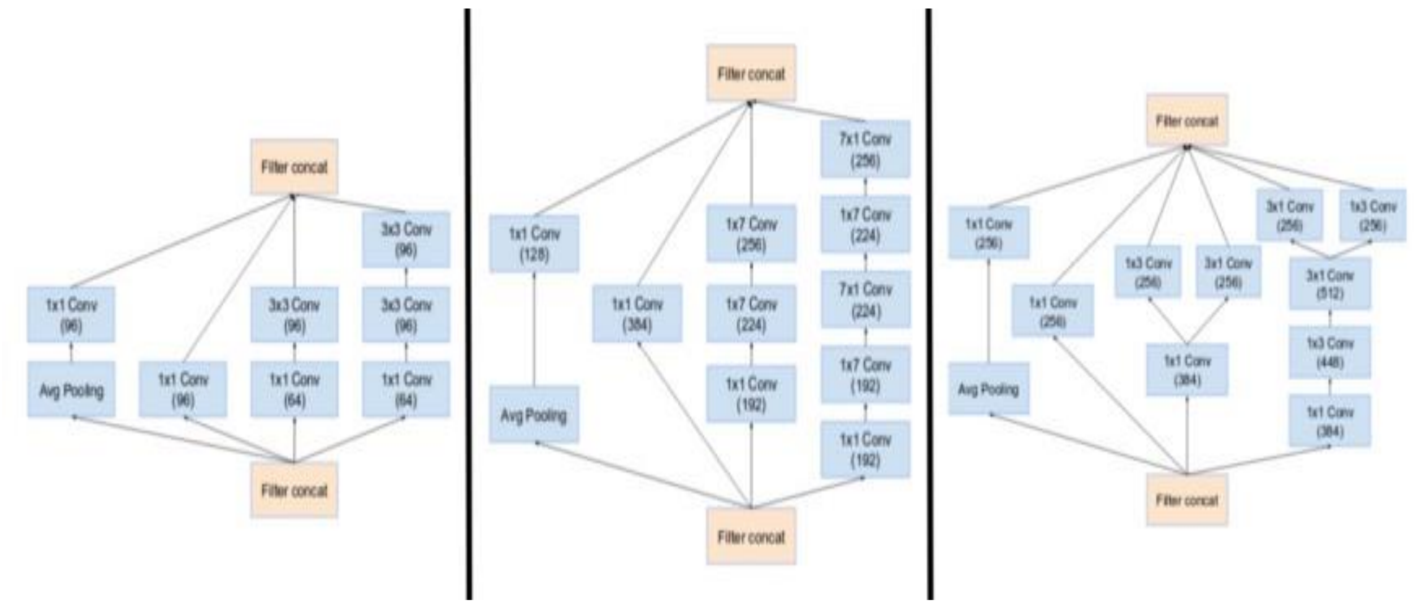


ResNet_18_WithDropOut_SGD

Have implemented dropouts for every call of residual block that is for every 4 convolution layers along with Batch normalization for every Convolution layer. The validation accuracy increased on par with training accuracy initially then after a point the validation accuracy became stable. Validation accuracy is 62.27.



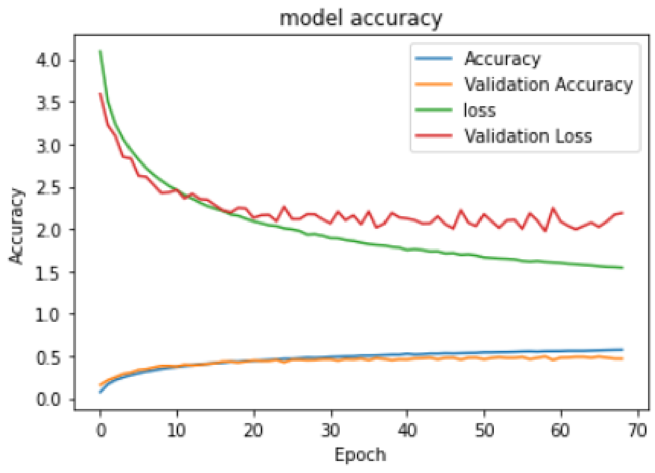
InceptionNet Implementation [3]



Deeper models are computationally expensive and tend to overfit easily. This model is wider and not so deep compared to other models and hence the model does not overfit unlike the previous model. Here we are adding multiple convolution layers on the same level and max pooling is performed on the same level, the outputs are concatenated and sent to the next inception module. I have implemented the above Inception V2 architecture, with three layers one after the other. The output of the first layer is sent to the second layer and similarly to the third layer. Initially the input is sent to three convolution layers with 3*3 filter size and 32 filters. Then a maxpool layer with 3*3 filter and stride 1 then 3 convolution layers with 3*3 filters and 32 filters. This output is sent to the first inception layer above and then the output from first inception layer is sent to second and output from second to third. Post the inception layers the output is sent to two convolution layers of 32,64 filters respectively. Then Average pooling is done, the output is sent to the Dense layer with 512 cells and then to the Output dense layer with 100 cells and softmax activation.
I am using early stopping, model checking and Data Augmentation for all the models.

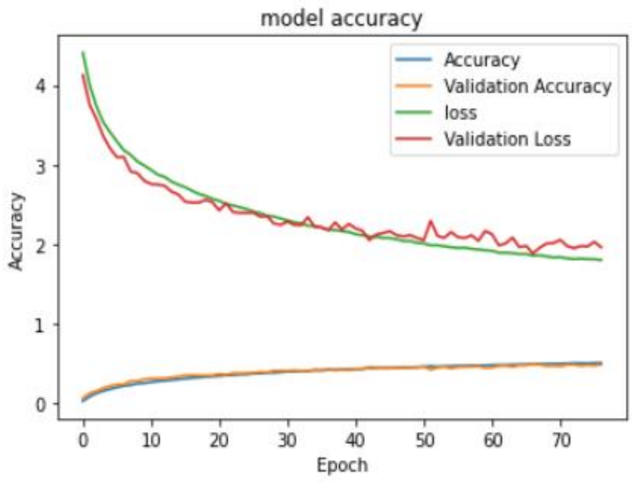
Inception_v2_NoRegularization_ADAM

I have used the above-mentioned architecture with Adam optimizer. Initially the Training and validation accuracy didn't diverge but after a point the validation accuracy became stable. With InceptionNet the model does not overfit so quickly compared to VGG and ResNet. As you can see in the below figure the model training accuracy is still increasing at 70th epoch. Validation accuracy is 49.46.



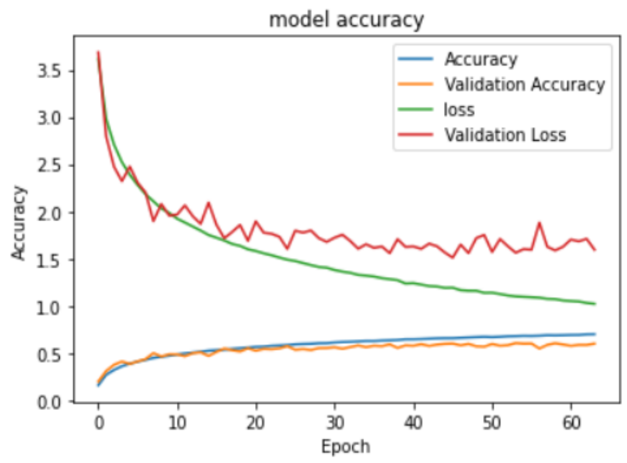
Inception_v2_NoRegularization_SGD

The implementation is similar to the above architecture with SGD and momentum. The accuracies do not diverge but after some point due to early stopping the model stops after reaching a stable validation accuracy. Validation accuracy is 50.12.



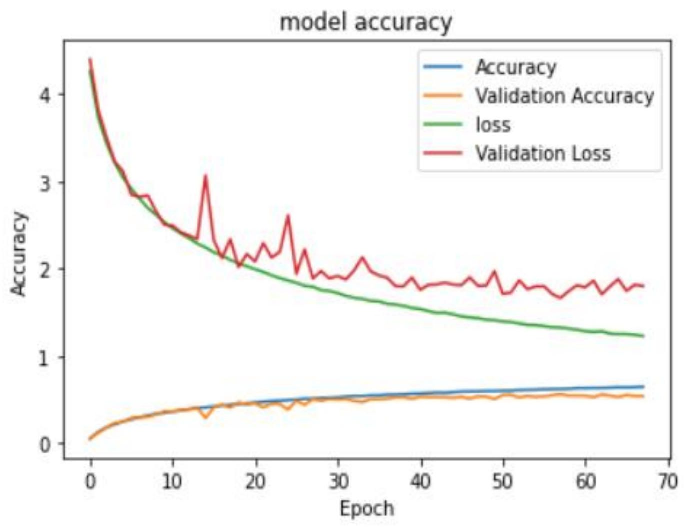
Inception_v2_WithBatchNorm_ADAM

Have used the basic Adam model and implemented the Batch Normalization to all the outputs which are sent to filter concatenation so that normalized weights are sent to the next Inception layer. I have used LeakyReLU activation for all the convolution layers and Dense layers, this increased the accuracy a little bit compared to normal relu. Validation accuracy is 60.37.



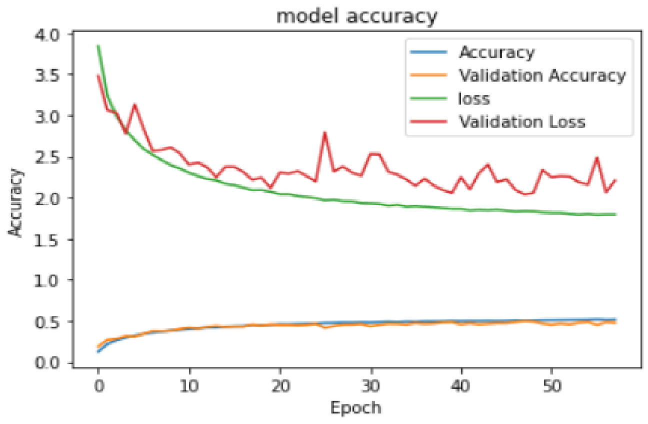
Inception_v2_WithBatchNorm_SGD

Have added the Batch Normalization similar to Adam except here I used relu activation instead of LeakyReLU. The model ran similarly except that it diverged a bit fast compared to Adam. The accuracy is little low compared to Adam, if I had implemented LeakyReLU the accuracy could have been better. Validation accuracy is 56.15.



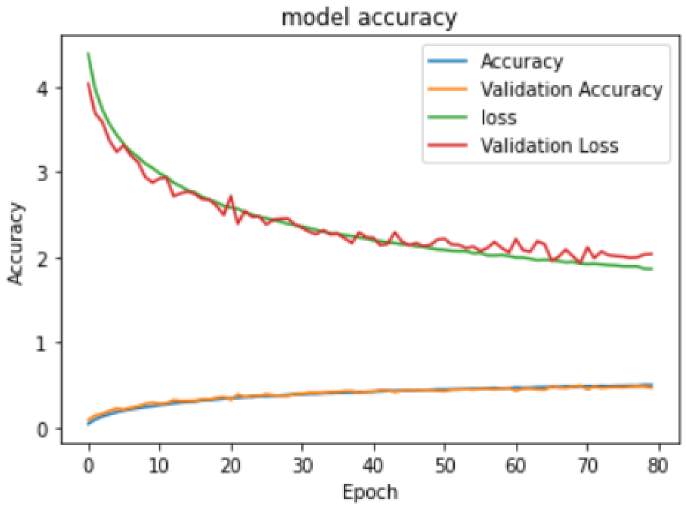
Inception_v2_WithDropOut_ADAM

Have implemented DropOuts with a value of 0.5 with relu activation the accuracy is low compared to the basic model. To increase accuracy have tried to use ‘elu’ as activation function, the accuracy improved little bit compared to relu. I believe that accuracy dropped with dropouts compared to Batch Normalization because since the model and not overfitting yet and the model is dense with each convolution layer being important, dropouts might be removing some of the necessary cells from the network. Validation accuracy is 49.48.



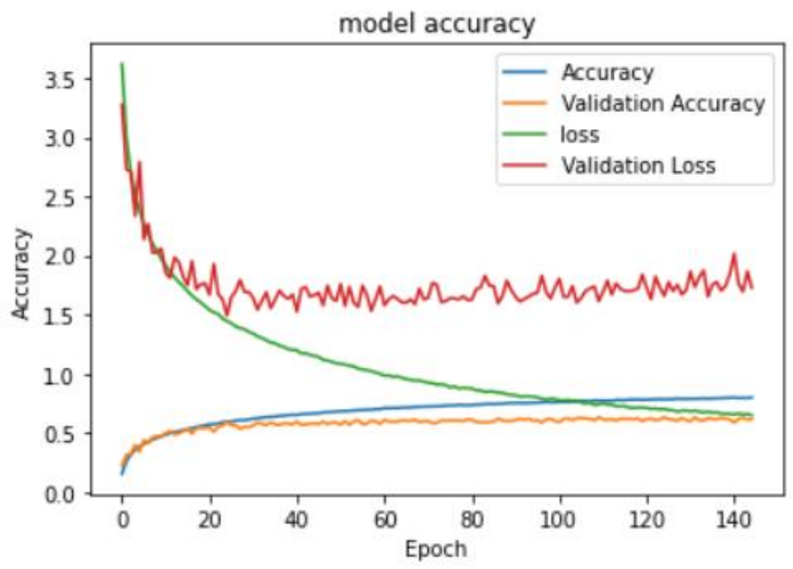
Inception_v2_WithDropOut_SGD

I have implemented DropOuts with SGD for the basic model and found the accuracy did not improve similar to the above case. Have used relu activation, other activations could have improved the accuracy a bit but not substantially. The reason could be the same as above that dropouts might be removing some of the necessary cells from the network. Validation accuracy is 49.08



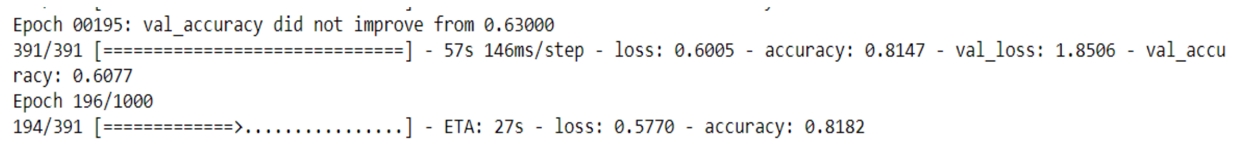
Inception_BatchNorm_ADAM_ExtraInceptionLayer

For all InceptionNet models I have implemented 3 different inception layers all only once, that is total of three Inception layers. I thought to increase the number of inception layers and check how that works. I have added once more inception layer to the Batch Norm with Adam model and increased the patience of early stopping to 20 since all the Inception models did not overfit but are stopped because of early stopping. With this exploration I found that Validation accuracy improved from 60.37 to 63.23, but it took 145 epochs to stop whereas the one with patience 10 took 65 epochs. To increase the accuracy in Inception we need to increase the depth and have to be more patient to get the effective results. Validation accuracy is 63.23.



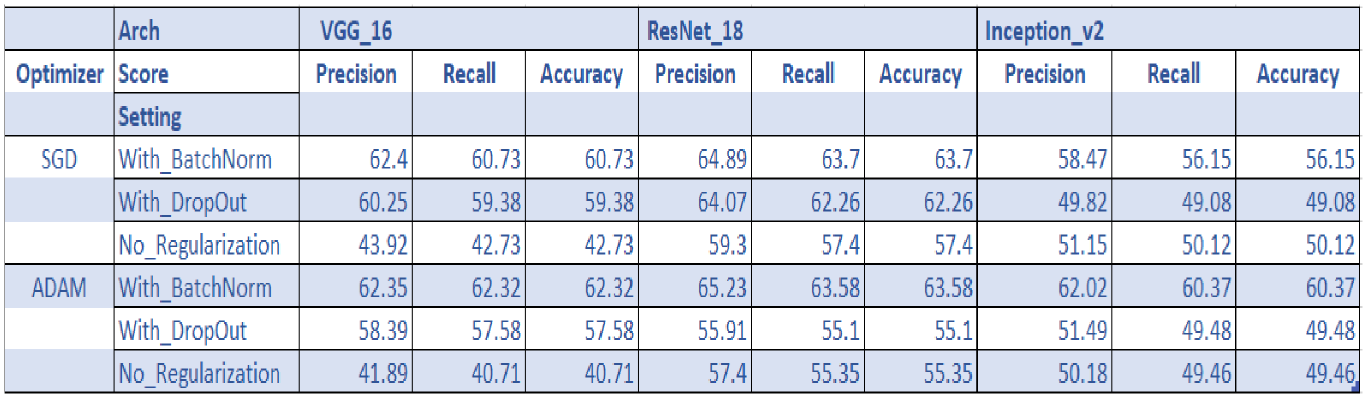
Inception_Without_EarlyStopping

I have tried to run an Inception model without early stopping to check when the model overfits and will there be any improvement in the accuracy. The model ran for 196 epochs with training accuracy of 81.82 but the test accuracy stopped at 60, I had to manually interrupt the model as it is taking forever since there is no early stopping.



```
Epoch 00195: val_accuracy did not improve from 0.63000
391/391 [==============================] - 57s 146ms/step - loss: 0.6005 - accuracy: 0.8147 - val_loss: 1.8506 - val_accu
racy: 0.6077
Epoch 196/1000
194/391 [=============>................] - ETA: 27s - loss: 0.5770 - accuracy: 0.8182
```

Results



	Arch	VGG_16			ResNet_18			Inception_v2		
Optimizer	Score	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy
	Setting									
SGD	With_BatchNorm	62.4	60.73	60.73	64.89	63.7	63.7	58.47	56.15	56.15
	With_DropOut	60.25	59.38	59.38	64.07	62.26	62.26	49.82	49.08	49.08
	No_Regularization	43.92	42.73	42.73	59.3	57.4	57.4	51.15	50.12	50.12
ADAM	With_BatchNorm	62.35	62.32	62.32	65.23	63.58	63.58	62.02	60.37	60.37
	With_DropOut	58.39	57.58	57.58	55.91	55.1	55.1	51.49	49.48	49.48
	No_Regularization	41.89	40.71	40.71	57.4	55.35	55.35	50.18	49.46	49.46

Conclusion

After running all the models with different variants following are my findings

- VGG model is simple but the computational cost is high since the number of parameters in VGG is very high compared to ResNet and InceptionNet. This is also observed from the size of weights file, which is 220MB for VGG, 40MB for ResNet and 4MB for InceptionNet.
- Inception layer is computationally very cheap compared to both but the model acts like a black box, we do not have the flexibility to add and remove layers.
- The InceptionNet model runs faster than ResNet and much faster than VGG for each epoch.
- The InceptionNet model takes time to overfit, whereas ResNet and VGG overfit quickly.
- A model with relu takes less time to run compared to a model with different activation functions.
- DropOut always does not increase Accuracy of the model and sometimes they might be removing some of the necessary cells from the network which reduces the model efficiency.
- VGG kind of architecture with proper placement of Convolution layers can also be productive and gives impressive results.
- Increasing the number of inception layers increases efficiency of the model to an extent.

References

[1] TDS_VGG, "step-by-step-vgg16-implementation-in-keras-for-beginners," [Online]. Available: https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c.

[2] TDS_ResNet, "building-a-resnet-in-keras," [Online]. Available: https://towardsdatascience.com/building-a-resnet-in-keras-e8f1322a49ba.

[3] TDS_Inception, "a-simple-guide-to-the-versions-of-the-inception-network-," [Online]. Available: https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202.