Machine Learning Project

REPORT

NAME: UMID NARZIEV (20023832)
MODULE: MACHINE LEARNING (EE4108)
ACADEMIC YEAR: 2020-2021

Contents

Abstract

To implement one of the popular CNN architecture and train it continuously and optimize hyperparameters to achieve high results in classification. Model suffers from high overfitting and this can be overcome by the optimal choice of dropout and image augmentation.

Introduction

The AlexNet contains eight layers with weight coefficients. The first five of them are convolutional, and the remaining three are fully connected. The output data is passed through the softmax loss function, which forms a distribution of class labels. The network maximizes a multilinear logistic regression, which is equivalent to maximizing the average for all training cases of the logarithm of the probability of correct labelling over the expected distribution. Relu is applied after each convolutional and fully connected layer. The dropout is applied before the first and second fully connected layers. It is capable of studying more complex objects and their hierarchies. Features of this solution: [1]

- Using linear rectification (ReLU) as nonlinearities.
- Using a drop technique to choose to ignore individual neurons during training, thus avoiding model overfitting.

Methods

The implementation of AlexNet was done by Keras open–source library. The initial architecture is depicted in below figure 1.

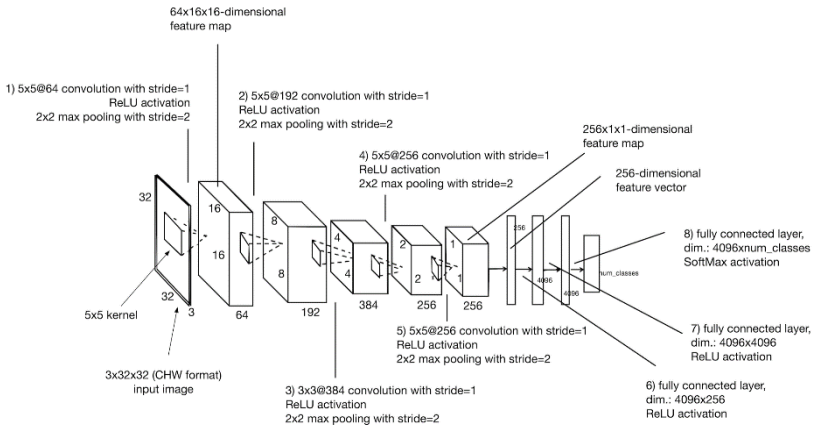

Figure 1: AlexNet architecture

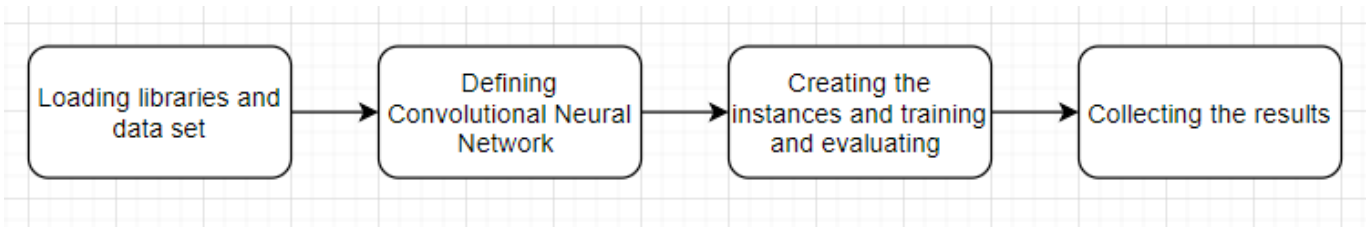

Figure 2: Neural Network Development Stages

Figure 2 shows the 4 stages of code implementation. They are described as follows:

1. **Loading libraries and data set:**
   To increase code readability, several modules are imported from Keras API directly into the notebook. The cifar-10 dataset is used to train and validate the model. The set consists of 10 classes with 50000 training and 10000 test images. The split ratio is done by sklearn with 20% validation and 80% test ratio.

1. **The architecture of convolution neural network:**
   The model is initially constructed as sequential flow, 5 convolution layers are added by Conv2D function with required kernel size and layer density. The ReLU activation function is added by Activation(). Additional features like dropout (SpatialDropout2D and Dropout), Pooling (MaxPooling2D), Batch normalization (BatchNormalization) are added and modified as per requirement and tuning. The input for fully connected layers is reshaped by Flatten function and by the Dense function the other connected layers are constructed.

2. **Model training and evaluating:**
   As the proposed architecture required extensive optimization and tunning, it is needed to create python dictionaries and lists to pass and record the training and evaluation results. Moreover, loss (categorical_crossentropy) and metrics (accuracy) are defined during model compilation. Thankfully,

an object of the model, after creation one has access to methods like ‘fit’ and ‘evaluate’ to train and predict.

3. **Collecting the result:**
   The result of iterations are collected into the dictionaries, and then they are used to produce graphs by ‘matplotlib’ library functions and saved into excel with the help of pandas and ‘xlsxwriter’ libraries, for further use.

Results

Task 1:

Initial AlexNet architecture and different optimizers to choose batch size:

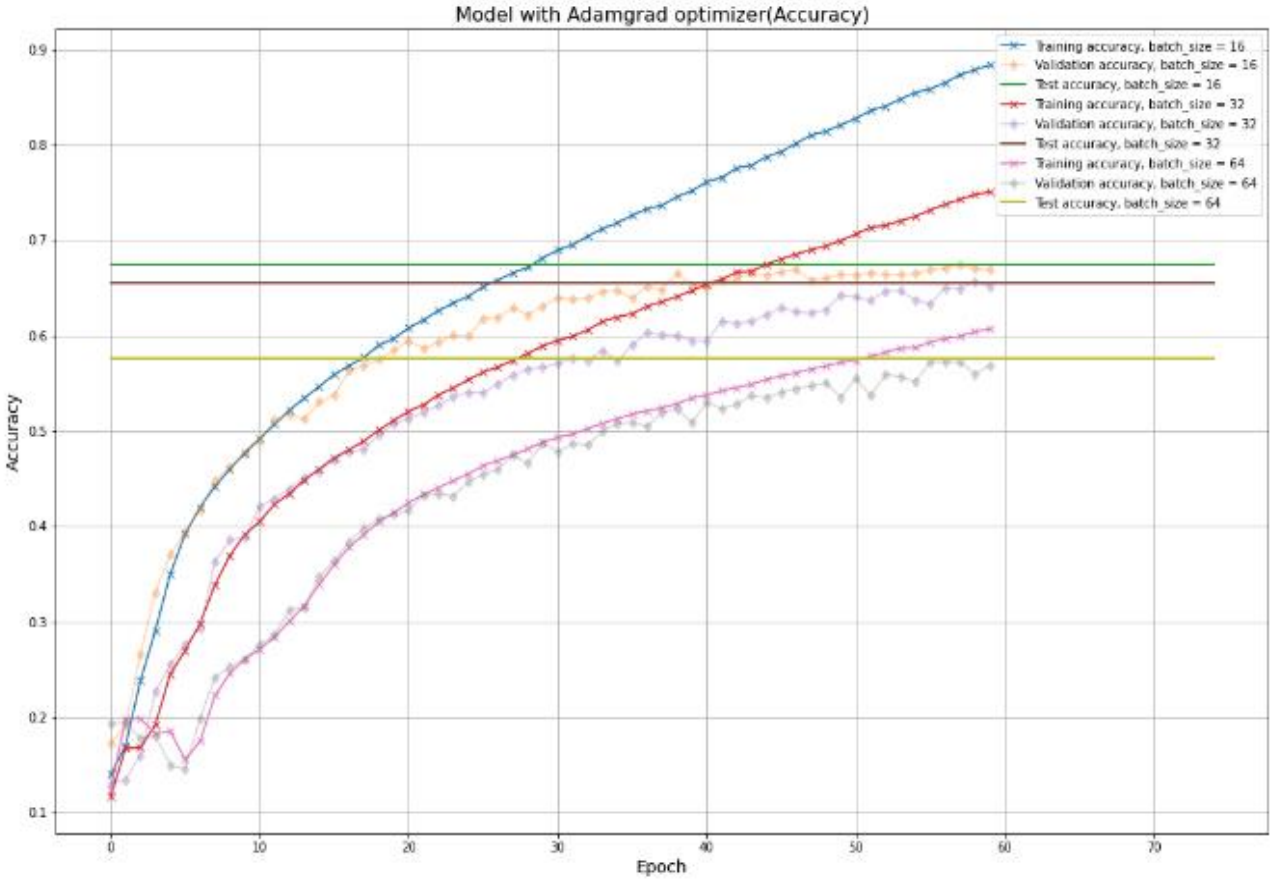

Figure 3: Adamgrad Optimizer with different batch sizes

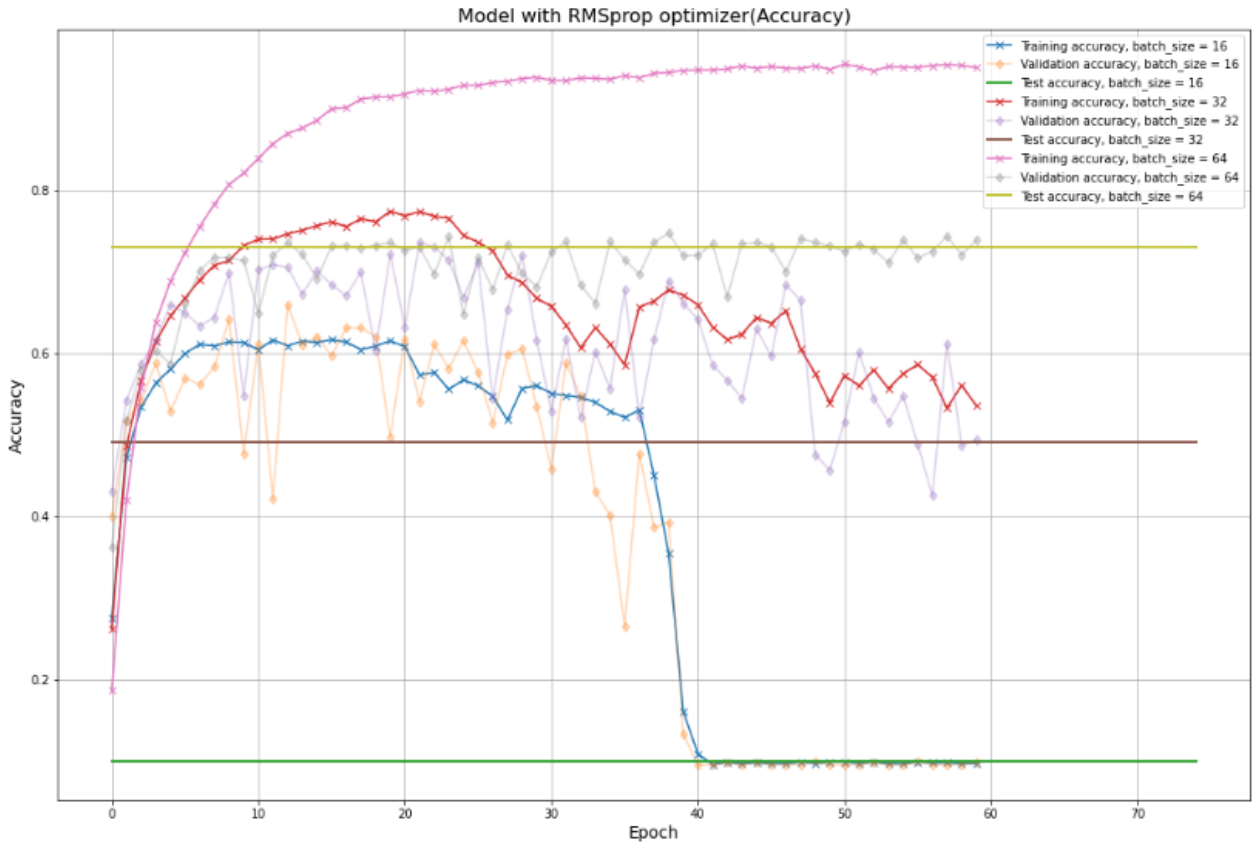

Figure 4: RMSprop with different batch sizes

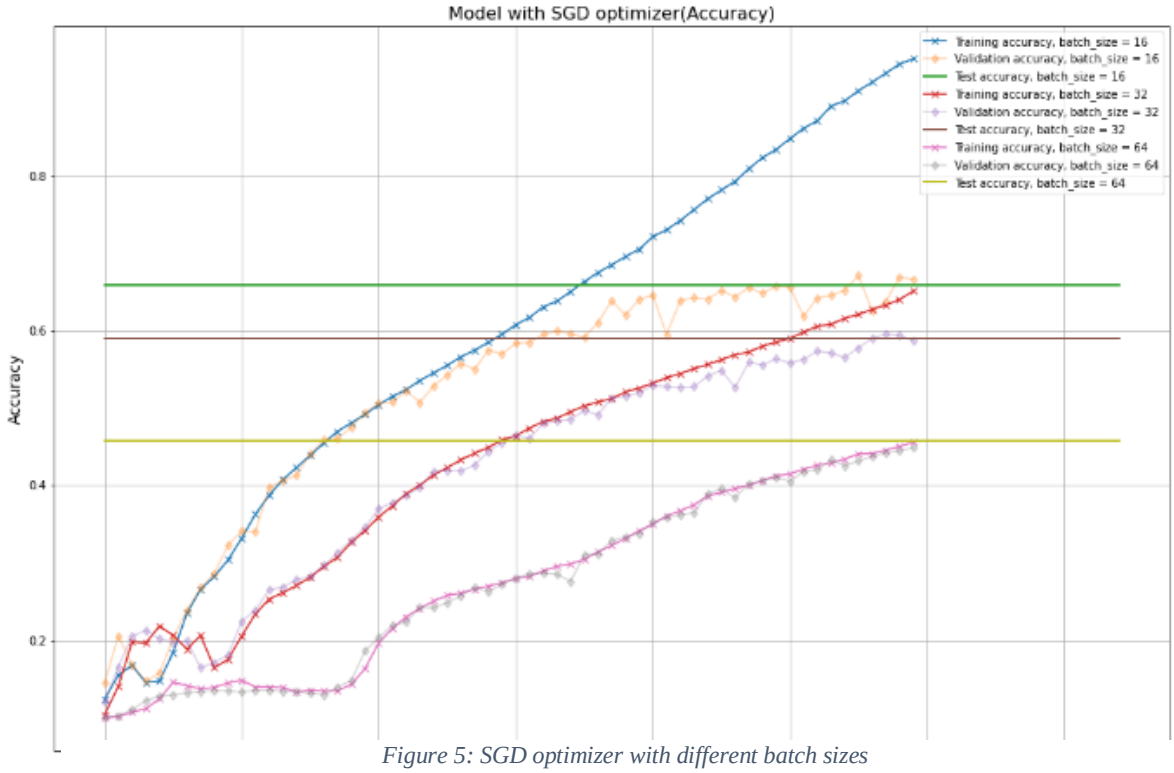

Figure 5: SGD optimizer with different batch sizes

Different Batch size		
Learning rate = 0.001	**Accuracy**	**Batch Size**
ADAMGRAD	0,67	16
	0,65	32
	0,58	64
RMSprop	0,10	16
	0,49	32
	0,73	64
SGD	0,66	16
	0,59	32
	0,46	64

Table 1: Accuracy result with batch size

The less the batch size is the better performance is shown for SGD and Adamgrad, and the opposite is true for RMSprop, the reason behind poor performance with the smaller batch size is that with a small learning rate it averages gradients over successive batches[2]. And the best performance is shown for RMSprop with batch size 64. And the second-best is Admgrad(67%).

Optimizer learning rate:

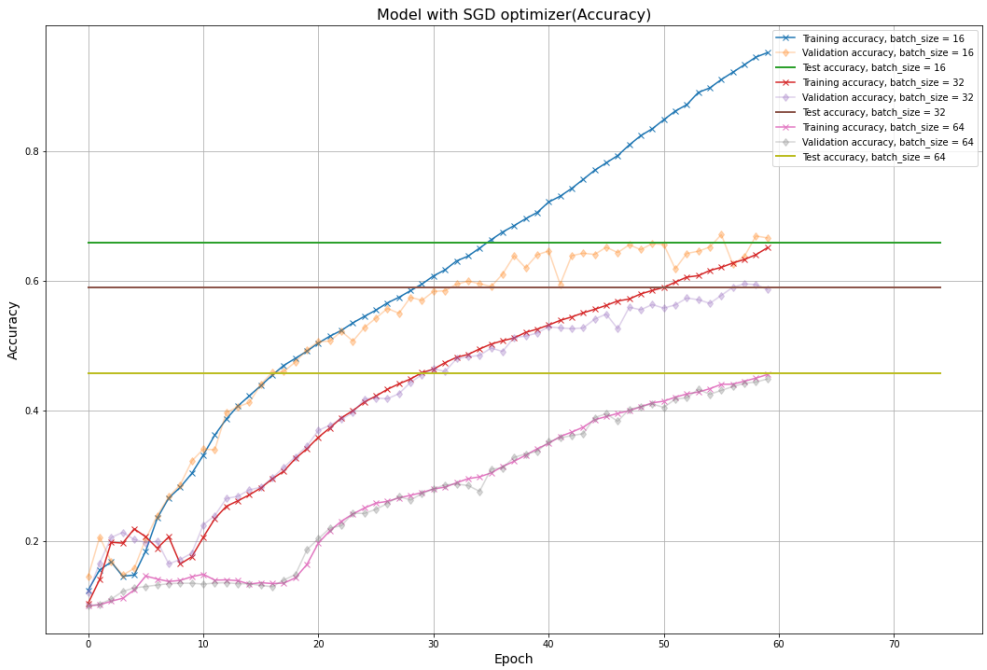

Figure 6: SGD optimizer with learning rate = 0.001

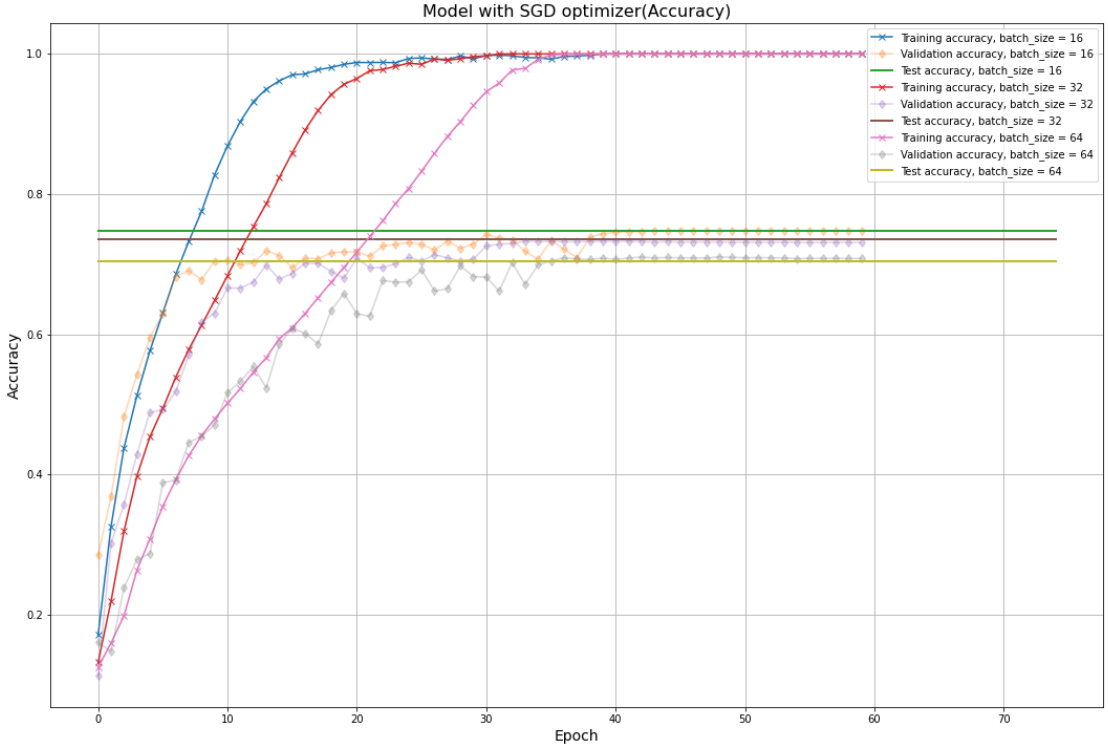

Figure 7: SGD optimizer with learning rate = 0.01

From, Figures 6 and 7, it is shown that α=0.01 converges faster than α=0.001, to an optimal value.

Task 2:

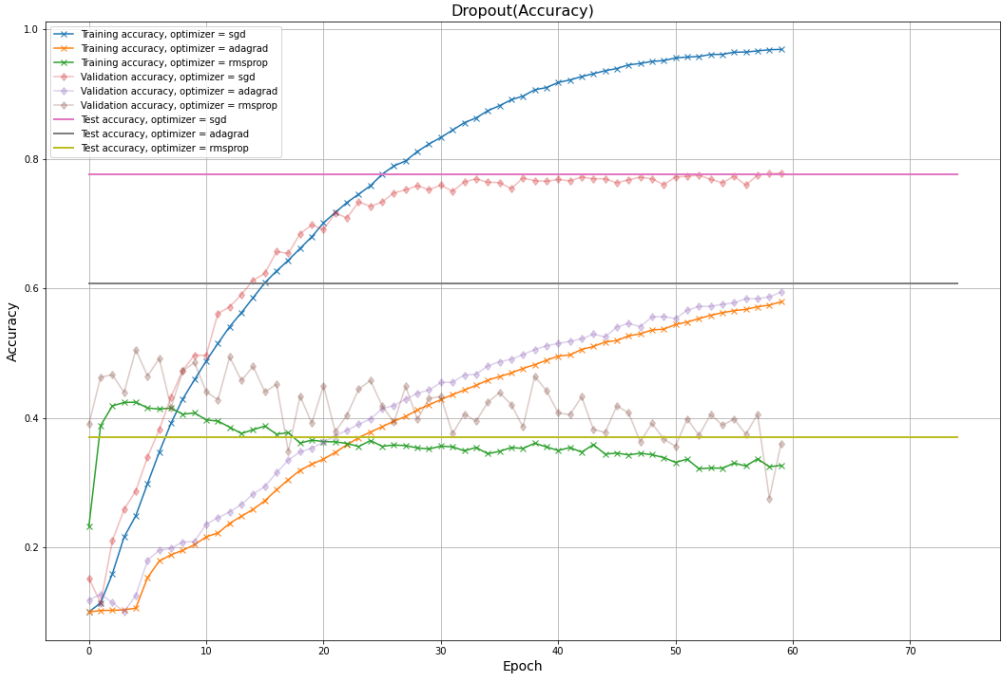

Figure 8: Different optimizer with dropout

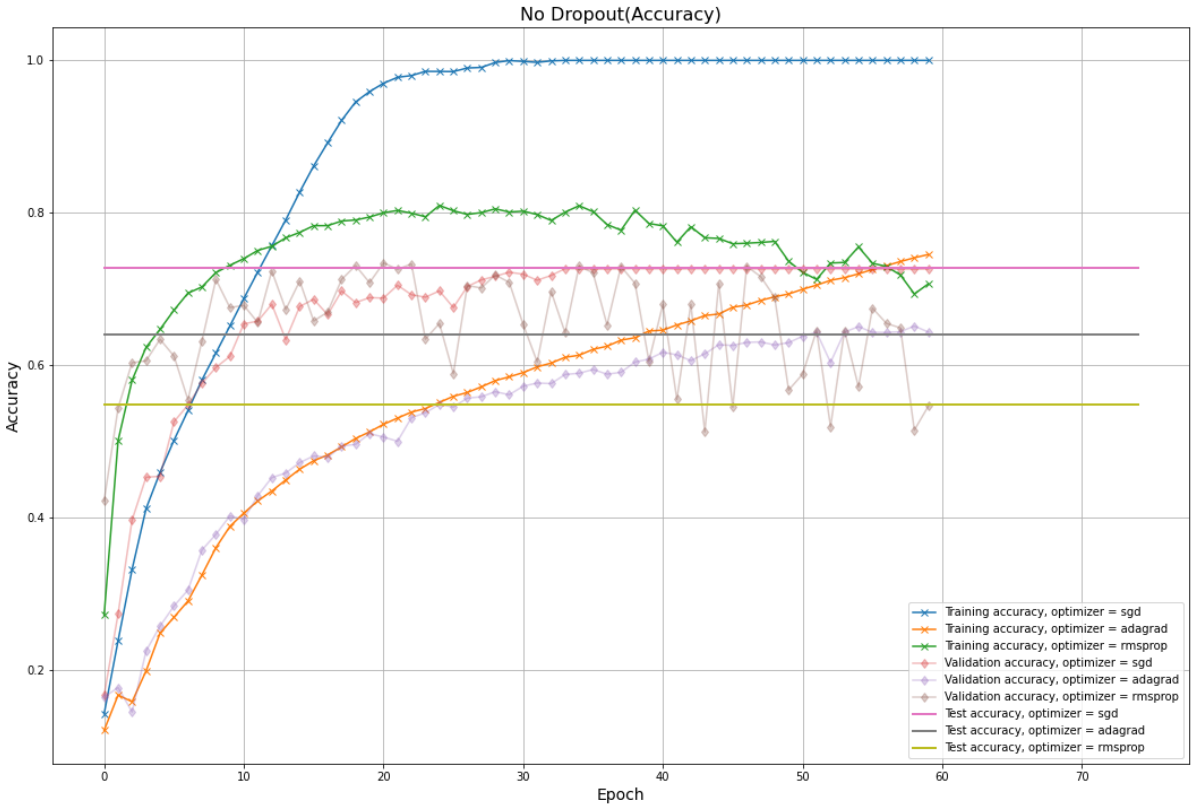

Figure 9: Different optimizers with no dropout

It is seen that in figure 9, the model is with SGD optimizer has reached saturation very quickly, and showed overfitting, however in figure 8 the dropout has significantly decreased the overfitting bringing accuracy from 75% to nearly 80%.

Batch size = 32		
Learning rate = 0.001	**Accuracy**	**Dropout**
ADAMGRAD	0,64	No
	0,61	Yes
RMSprop	0,55	No
	0,37	Yes
SGD	0,73	No
	0,78	Yes

Table 2: Dropout rate comparisons for different optimizers

From Table 2 the other two optimizers have shown slower convergence and did not reach saturation point as fast as SGD despite they had similar learning rates.

Task 3:

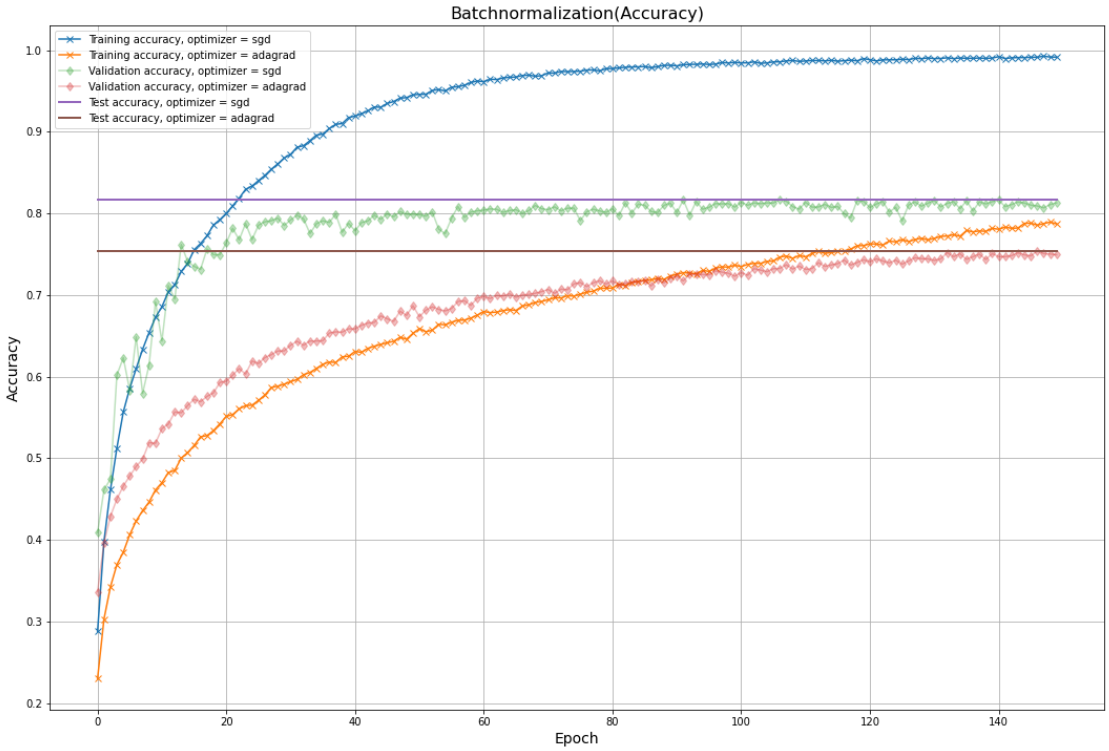

Figure 10: Batch normalization with SGD optimizers

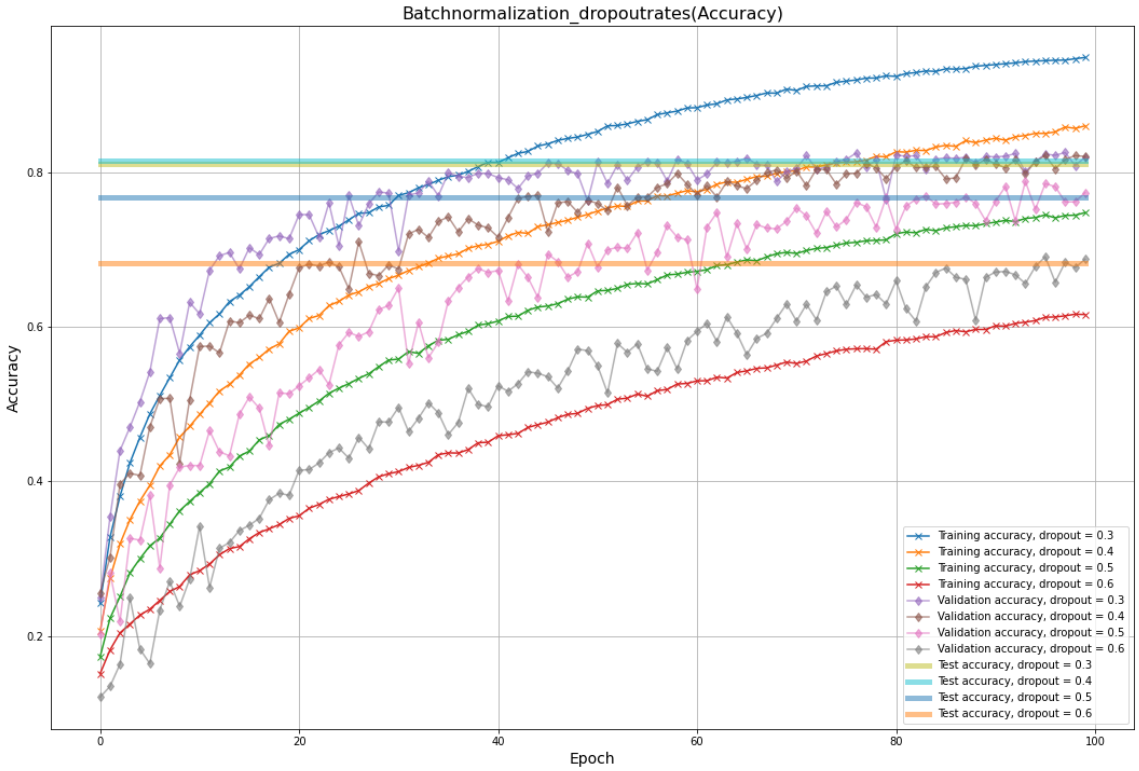

Figure 11: Batch Noralization with different dropout rates at Convolution layer

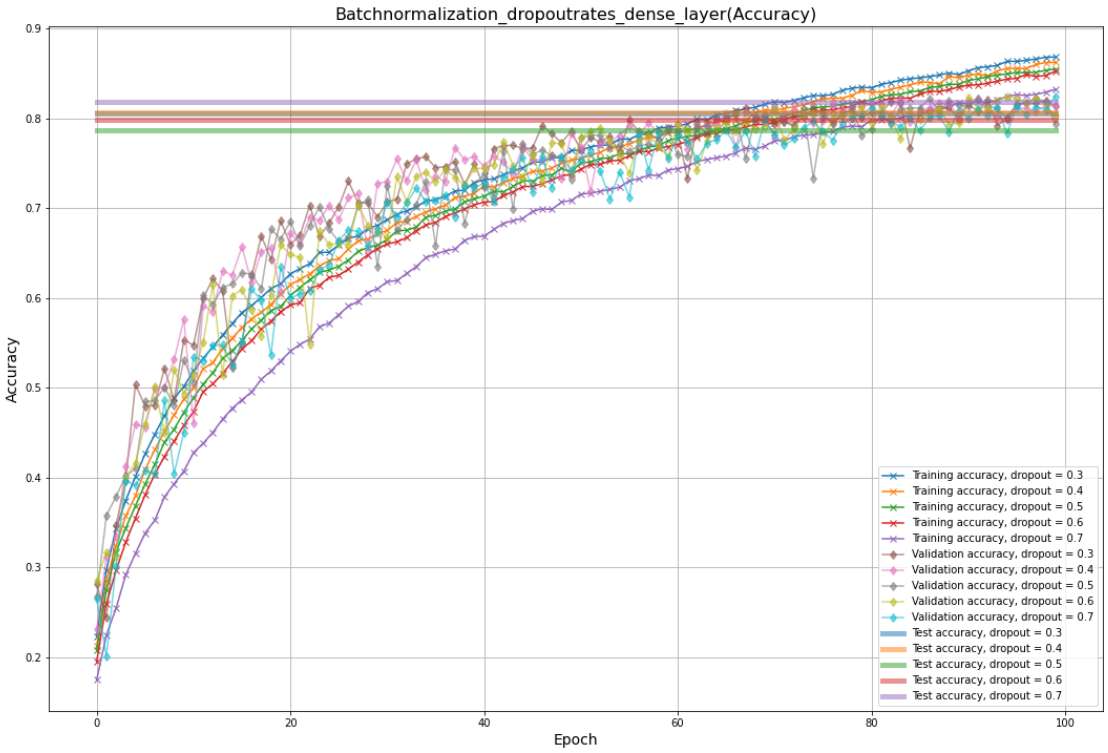

Figure 12: Batch Noralization with different dropout rates at Dense layer

Figure 10 depicts batch normalization with a dropout rate of 0.2, however, severe overfitting is observed, but dropouts 0.4 at the convolution layer and 0.7 at the dense layer perfectly eliminates overfitting increasing overall prediction accuracy to 82%.

Task 4:

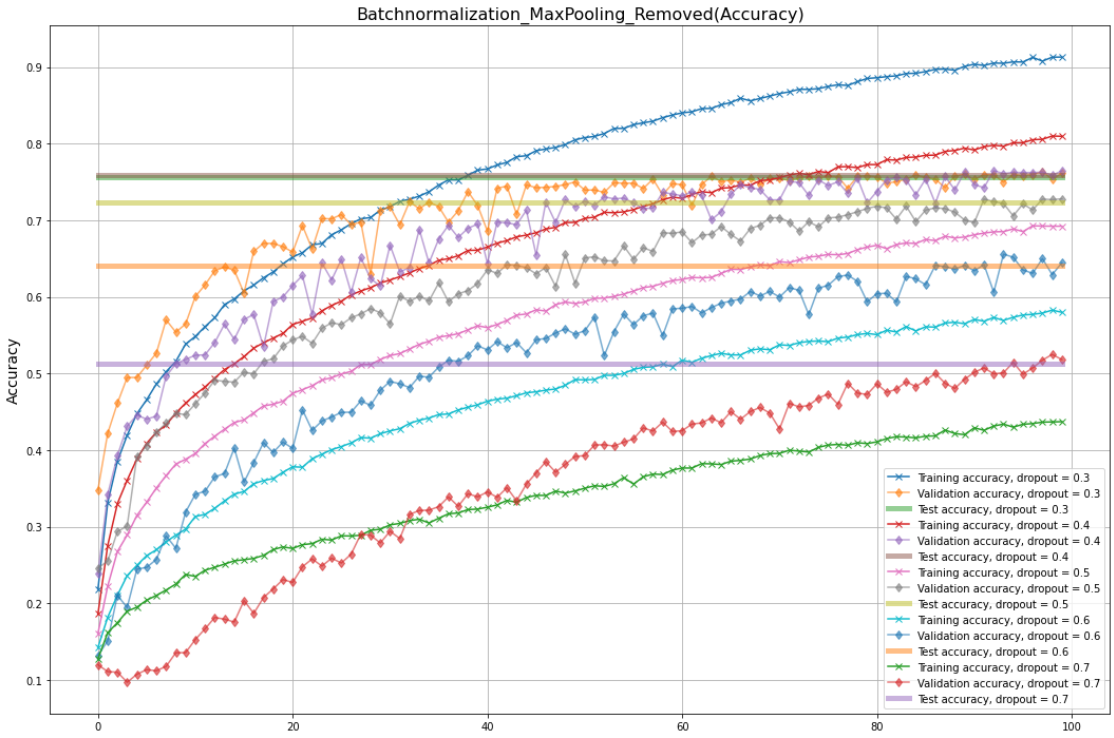

Figure 13: Maxpooling layer removed and different dropout rates

The new architectural modification result is shown in figure 13, this time max-pooling is removed, which means, a lot more features are introduced to the network, so in this case, 100 epochs with a learning rate of 0.01 were not enough. And dropout of 0.04 was a perfect choice in this case too.

Task 5:

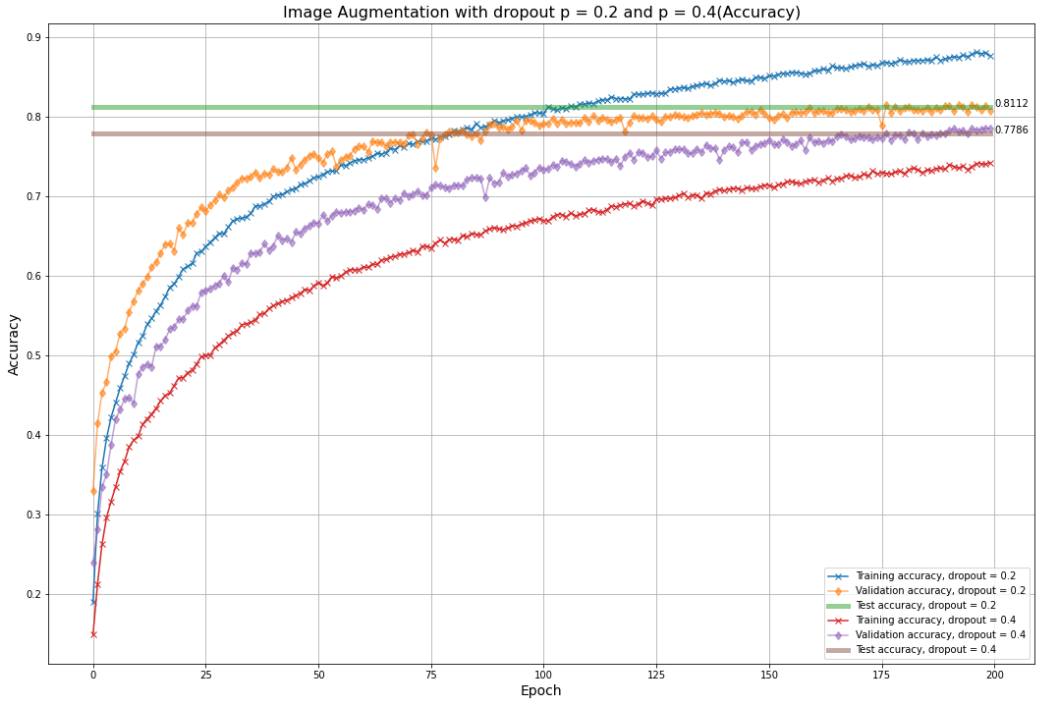

Figure 14: Image Augmentation

Another, nice feature like image augmentation was introduced to the network, to train a more robust model. So, with 200 epochs system almost build accuracy of 81%, and the augmentation image class has rotation, brightness, zoom and flip features.

Conclusion

Extensive optimization of parameters of convolutional neural network has been carried out. The following parameters were found optimal during the tuning process:

№	Parameters	Value
1	Optimizer	SGD(α=0.01 with max-pooling)
2	Dropout rate	0.4 @ convolutional layer and 0.9 @ dense layer.
3	Epochs	100-150
4	Image Augmentation	4 or more parameters
5	Accuracy	~80-82%

Bibliography

[1]A. Krizhevsky, I. Sutskever and G. Hinton, "ImageNet classification with deep convolutional neural networks", *Communications of the ACM*, vol. 60, no. 6, pp. 84-90, 2017.

[2]V. Bushaev, "Understanding RMSprop — faster neural network learning", *Medium*, 2018. [Online]. Available: https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a. [Accessed: 13- Aug- 2021]