Exploring Deeper Networks for Multiclass Image Classification

Aniruddha Mulay
Queen Mary University of London
a.mulay@se20.qmul.ac.uk

Abstract—This experiment discusses the effect of different deep learning architectures on the training and test set accuracies on two very popular image datasets namely MNIST and CIFAR-10 datasets. This experiment evaluates the performance of 3 different deep learning architectures i.e., VGG-16, ResNet-34, GoogleNet on the CIFAR-10 and MNIST datasets for different optimizers, epochs and learning rate values.

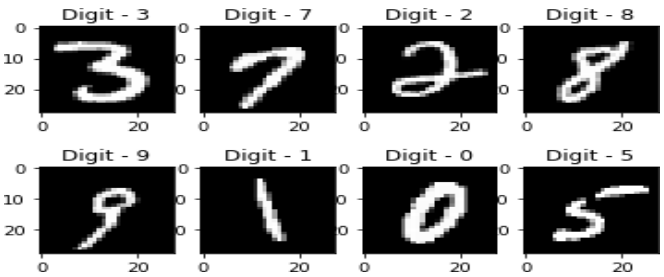
I. INTRODUCTION

Convolutional Neural Networks (CNN) form the backbone of many of today's modern computer vision systems. CNN were first used for handwritten character recognition in the 1990s by Yann LeCun[1]. To be more specific, a CNN model was trained for recognition of handwritten digits in the MNIST database. CNN demonstrated that it was able to combine several different simpler features and create more complex features from them enabling it to successfully recognize handwritten digits. Over the course of time, deeper and more complex state-of-the-art CNN models have been developed which have been able to recognize hand written digits with an extremely high degree of accuracy. In this report I have demonstrated the usage of three extremely popular deep learning architectures namely VGG-16, ResNET-34, GoogleNet and applied them 2 popular datasets (MNIST and CIFAR-10) to evaluate their performance over different optimizers, epochs and learning rates.

II. DATASETS

A. MNIST Dataset

The MNIST database[2] is a dataset which contains a large number of handwritten digits. It is one of the most popular datasets used for training and evaluation of different deep learning architectures. The MNIST dataset contains 60,000 training images and 10,000 testing images of handwritten digits numbered from 0 to 9. These images are single channel grayscale images carrying the dimensions 28x28. This dataset has been the go-to dataset when it comes to benchmarking different Machine Learning and Deep Learning architectures since 1999.



B. CIFAR-10 Dataset

CIFAR stands for Canadian Institute for Advanced Research. Along with MNIST dataset, it is one of the most common and popular datasets used for training and evaluating the performance of different Machine Learning algorithms and Deep learning architectures. The CIFAR-10 dataset[3] contains approximately 60,000 images of dimensions 32x32 spread out over ten different classes. The 10 different classes represent airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, trucks. Each class contains about 6000 images. The dataset is designed in such a way that there is no overlap between two different image classes. Unlike the MNIST dataset which is a single channel grayscale data, the CIFAR-10 dataset is a 3-channel dataset containing Red, Green, Blue channels with each channel containing 1024 color values.

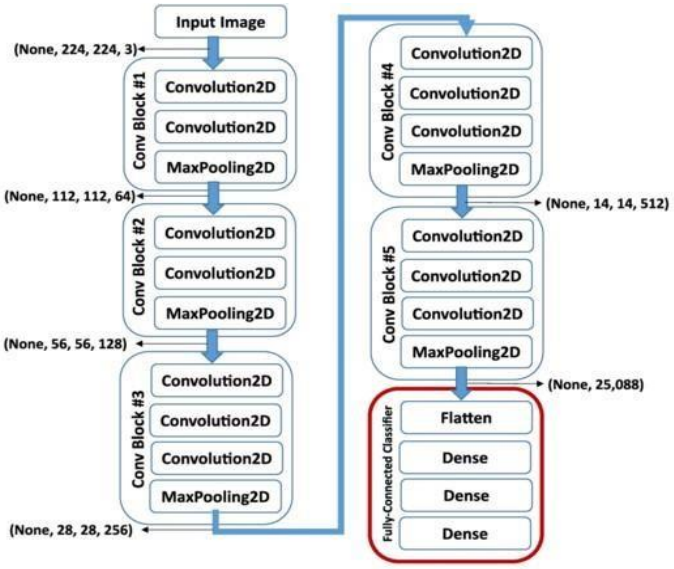
III. DEEP LEARNING ARCHITECTURES

A. VGG-16

VGG architecture[4][5] is a type of Convolutional Neural Network (CNN) architecture which was proposed by Karen Simonyan and Andrew Zisserman belonging to the Visual Geometry Group of the University of Oxford in 2014. The architecture derives its name from the group which first proposed this architecture. The authors detailed their work on the architecture in the paper [2]. The VGG architecture was the 1st runner-up in the 2014 ILSVRC competition. The VGG16 architecture consists of 13 convolutional layers and 3 fully connected layers Compared to other CNN architectures, VGG has comparatively smaller filter sizes at 3x3. The 3 fully connected layers in the VGG16 architecture consists of two FC layers with 4096 channels each and one layer having 1000 channels to detect 1000 different class labels in the default configuration. For this proposed study, we have made changes to the last fully connected layer by changing the total number of output classes from 1000 to 10 as both MNIST and CIFAR-10 have 10 different image

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

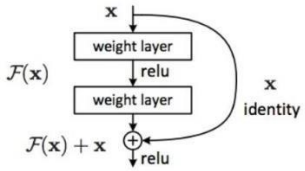
classes which we have to classify. The VGG-16 architecture structure is as follows:



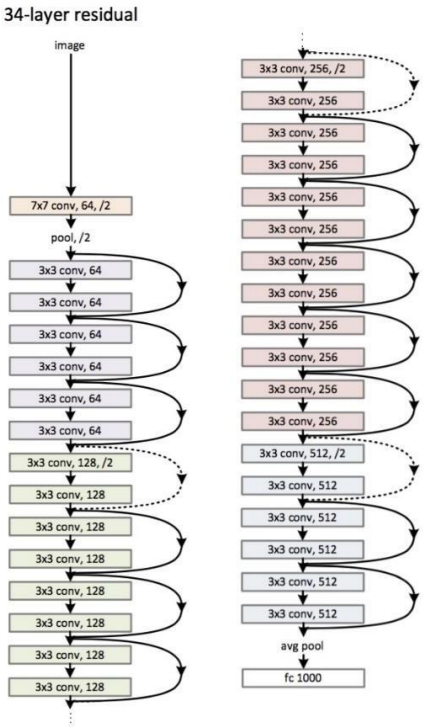
Input image is passed to the 1st conv block where convolution operation is performed with a 3x3 kernel and stride of 1. Convolution operation generates feature maps. The feature maps store the information of extracted features from the image. Max pooling operation is then performed on the convolved image which halves its dimensions. Max pooling operation is used to reduce the computational and also to prevent the network from overfitting. The resultant image is then passed through the 2nd, 3rd, 4th and 5th convolutional blocks after which it is passed through the fully connected layers. The image vector is first flattened and then passed through two dense layers of 4096 units each. ReLU is used as an activation function for these layers. The final dense layers output the number of classes within the dataset which we have to classify. The final dense layer uses softmax activation which outputs a value between 0 and 1 based on the confidence of the model class which the image belongs to.

B. ResNet-34

ResNet or Residual Networks forms the backbone of many of today's computer vision, image recognition tasks. ResNet model[6][7] was proposed in the paper to address the problems faced during training deeper neural networks. Earlier, deeper neural networks suffered from the problem known as vanishing gradient when convolutional layers were simply stacked on top of each other to create a deep neural network. During training, a 56-layer CNN had a much higher training and test set accuracy compared to a 20-layer CNN due to the problem of vanishing gradients. The vanishing gradient problem basically happens when repeated multiplication of the gradient makes the gradient extremely small in the backpropagation stage. This affects the performance of the deeper network. The ResNet architecture counters this problem by the introduction of residual block.



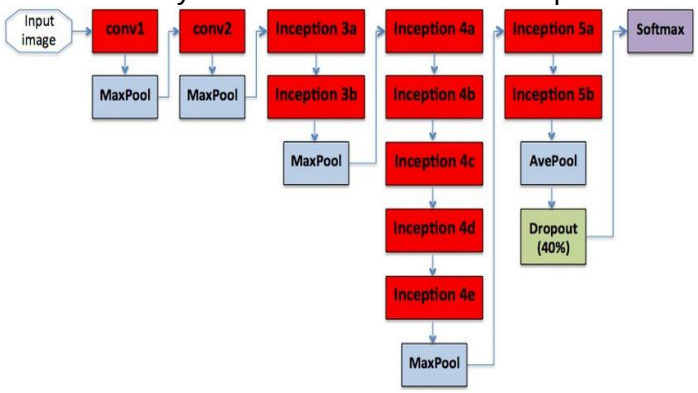
The skip connection between the layers sums up the output of the previous layers to the output of the stacked layers. This helps in training the deeper networks more efficiently compared to training deeper networks without the skip connection and also mitigates the problem of vanishing gradient by allowing the gradient to flow through an alternate path. The different versions of ResNet are ResNet18, ResNet34, ResNet50, ResNet101 and ResNet152. We have used ResNet34 architecture for this experiment. The structure of the ResNet34 architecture is as follows:



C. GoogleNet

GoogleNet is a variant of the Inception architecture which was developed and proposed by the researchers at Google in 2014 in the paper[8][9]. The GoogleNet architecture was the winner of the ILSVRC 2014 image classification competition. The architecture successfully demonstrated a lower error rate when compared to other architectures like AlexNet and VGG. The problem with deeper CNN was that the increase in layers suffered from overfitting and vanishing gradient problems. The Googlenet architecture, to a large extent, solved these problems by making use of the Inception module. The inception module makes use of dimensionality reduction techniques to reduce the overall computational cost of the network. One of the ways in which Googlenet performs dimensionality reduction is by

adding 1x1 convolution layers before the 3x3, 5x5 convolution layers. This allows for faster computation.



The Googlenet architecture consists of 22 layers (27 layers including pooling layers). A part of these layers are the 9 inception modules from 3a to 5b. The input image is passed through the first convolution layer which has a patch size of 7x7 which is much higher compared to other architectures, this patch size helps in reducing the input image dimensions while recovering the spatial information. Maxpooling is then performed on the image. The resultant image is then passed to the 2nd convolutional layer which makes use of 1x1 conv layers for dimensionality reduction thereby decreasing the computational load. The purpose of the 9 convolutional layers is to down sample the input image as it goes through the network. This happens via reduction in the input image dimensions. This greatly helps in reducing the computational load on the network. After the 9 inception modules, average pooling is applied on the resultant image which computes the mean of all the features maps produced by the network till now while reducing the image dimensions to 1x1. Dropout is utilized to randomly drop some nodes within the network so as to prevent the network from overfitting.

IV. OPTIMIZERS UTILIZED

A. SGD with momentum

SGD[10] stands for Stochastic Gradient Descent. Unlike Gradient Descent & Batch Gradient Descent which update the weights for the network parameters for the entire training dataset and for a certain batch size respectively, SGD updates the weights for every single training instance. The word Stochastic in SGD means that the process is linked with a random probability. The SGD optimizer randomly selects a single datapoint for performing the gradient descent operation. However, the drawback to such a method is that SGD is slow to converge and the path to reach the minima is extremely noisy. The high oscillations means that SGD cannot be used with higher learning rates which further increases the time required for convergence. This problem can be alleviated by using SGD with momentum which makes use of exponentially weighted averages to compute the gradient and use this gradient value for updating the weights of the parameter.

- An equation to update weights and bias in SGD

$$w_t = w_{t-1} - \eta \frac{\partial L}{\partial w_{t-1}}$$

$$b_t = b_{t-1} - \eta \frac{\partial L}{\partial b_{t-1}}$$

- An equation to update weights and bias in SGD with momentum

$$w_t = w_{t-1} - \eta\, V_{dw_t}$$

$$\text{where } V_{dw_t} = \beta V_{dw_{t-1}} + (1-\beta)\frac{\partial L}{\partial w_{t-1}}$$

$$b_t = b_{t-1} - \eta\, V_{db_t}$$

$$\text{where } V_{db_t} = \beta V_{db_{t-1}} + (1-\beta)\frac{\partial L}{\partial b_{t-1}}$$

The addition of momentum parameter to SGD allows the SGD to converge faster while reducing the oscillations. This allows the usage of higher learning rates which further improves the speed of convergence.

B. Adagrad

In optimizers like SGD, the learning rate remains constant. Optimizers like Adagrad[10] make use of flexible learning rates as different parameters require different learning rate values i.e., for sparse feature parameters a higher learning rate won't affect the final result but for densely packed feature parameters a higher learning rate might not produce the desired results. Dense feature parameters require lower learning rate values compared to sparse feature parameters.

$$\text{SGD} \Rightarrow\ w_t = w_{t-1} - \eta \frac{\partial L}{\partial w_{t-1}}$$

$$\text{Adagrad} \Rightarrow w_t = w_{t-1} - \eta'_t \frac{\partial L}{\partial w_{t-1}}$$

$$\text{where } \eta'_t = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}}$$

$$\varepsilon \text{ is a small } +\text{ve number to avoid divisibilty by } 0$$

$$\alpha_t = \sum_{i=1}^{t} \left(\frac{\partial L}{\partial w_{t-1}}\right)^2 \quad \textit{summation of gradient square}$$

Adagrad eliminates the need to manually tune the learning rate & its convergence far faster and more reliable compared to other optimizers.

C. Adam

Adam[10] stands for adaptive movement estimation. The Adam optimizer serves as an extension of SGD which has recently become extremely popular for a variety of machine learning and deep learning tasks. The Adam optimizer can be seen as an extension of Adagrad and RMSprop such that it combines the benefits of both these optimizers. Adam uses momentum and adaptive learning rates to converge faster.

Some of the benefits of the Adam optimizer are:

- Straightforward and simple to implement.
- Computationally efficient.
- Memory requirements are low.
- Suited for solving problems with large datasets.
- Hyper-parameters have intuitive interpretation hence they require little to no tuning.

V. EXPERIMENTS AND RESULTS

A. MNIST

VGG-16, ResNet-34 & GoogleNet architectures were trained & validated using the popular MNIST dataset in this experiment. The tests were carried out using different optimizers, different learning rate values & different total number of epochs so as to evaluate and compare the performance of the results obtained. In the paper, it is mentioned that a standard 224x224 image is passed as an input to the VGG-16 and GoogleNet architectures. However, the images in the MNIST dataset are 28x28 and enlarging them to 224x224 will lead to artefacts which might affect the performance of the architectures, hence, the images were resized to 70x70 dimensions as such a dimension would not cause artefacts within the image. The results obtained for the 3 architectures are as follows:

Architecture: VGG-16, Dataset: MNIST

Optimizer	Learning Rate	Epochs	Train Acc(%)	Test Acc(%)	Loss
SGD	0.01	5	97.32	97.78	0.371
		10	98.93	98.74	0.073
	0.001	5	98.62	98.75	0.025
		10	99.36	99.13	0.002
Adagrad	0.01	5	21.80	31.02	1.839
		10	15.24	27.80	2.162
	0.001	5	98.83	98.74	0.0007
		10	99.58	99.02	0.0001

Architecture: ResNet-34, Dataset: MNIST

Optimizer	Learning Rate	Epochs	Train Acc(%)	Test Acc(%)	Loss
SGD	0.01	5	98.93	99.10	0.006
		10	99.48	99.12	0.030
Adagrad	0.01	5	98.76	98.91	0.0109
		10	99.47	99.32	0.0001
Adam	0.01	5	98.14	97.93	0.0281
		10	98.91	98.60	0.0331

Architecture: GoogleNet, Dataset: MNIST

Optimizer	Learning Rate	Epochs	Train Acc(%)	Test Acc(%)	Loss
SGD	0.01	5	99.54	99.29	0.0021
		10	99.77	99.38	0.00003
Adagrad	0.01	5	99.52	99.42	0.0060
		10	99.80	99.53	0.00004
Adam	0.01	5	98.00	98.97	0.0336
		10	98.96	98.24	0.0072

Tests were carried out on the VGG-16 architecture using SGD and Adagrad optimizers for learning rates of 0.001 and 0.01. The networks were tested for 5 epochs and 10 epochs respectively. VGG-16 using SGD was able to achieve 95%+ training and test set accuracies for both the learning rate and epoch values. VGG-16 struggled with Adagrad optimizer when the learning rate was set to 0.01 achieving only a 21.80% train accuracy and 31.02% test accuracy when tested for 5 epochs. The performance was even poorer when tested for 10 epochs. This might suggest the learning rate is too high and the network not being able to converge to the minima. With the learning rate set to 0.001, the network was able to achieve 95%+ train and test set accuracy with extremely low loss values. Adam optimizer was not tested with VGG-16 as the optimizer struggles to deal with the extremely high number of parameters(138 million) of the architecture. Further testing the dataset on ResNet-34 and GoogleNet architecture, we can clearly see that the networks are able to achieve train and test set accuracy values greater than 95%. With the number of epochs increased to 10, we can see that although there is not a significant increase in train and test set accuracy values, the loss value drop significantly when the number of epochs are increased. Both ResNet and GoogleNet performed exceedingly well with the MNIST dataset. In certain cases, we notice that the test set accuracy is greater than the train set accuracy, this is due to the less than ideal split of train and test set images of the dataset.

B. CIFAR-10 Dataset

VGG-16, ResNet-34, GoogleNet were also tested on the CIFAR-10 dataset to evaluate how these architectures perform when provided with 3 color channel dataset with more distinct classes as compared to the MNIST dataset. Similar to MNIST, the images were resized to 70x70 dimensions so as to provide a common scale for evaluation and also not to enlarge the images to such an extent that it would cause artefacts within the image. The results obtained were as follows:

Architecture: VGG-16, Dataset: CIFAR-10

Optimizer	Learning Rate	Epochs	Train Acc(%)	Test Acc(%)	Loss
SGD	0.01	20	71.31	73.25	0.498
		30	91.33	86.15	0.056
	0.001	20	95.09	84.02	0.025
		30	97.79	85.43	0.015
Adagrad	0.01	20	10.18	10.73	2.265
		30	21.46	23.59	1.960
	0.001	20	92.27	66.61	0.747
		30	95.45	76.84	0.007

Architecture: ResNet-34, Dataset: CIFAR-10

Optimizer	Learning Rate	Epochs	Train Acc(%)	Test Acc(%)	Loss
SGD	0.01	20	96.76	83.69	0.0126
		30	98.91	84.79	0.0303
Adagrad	0.01	20	98.40	82.48	0.0834
		30	98.89	82.80	0.0110
Adam	0.01	20	96.39	83.17	0.0035
		30	97.45	82.37	0.0772

Architecture: GoogleNet, Dataset: CIFAR-10

Optimizer	Learning Rate	Epochs	Train Acc(%)	Test Acc(%)	Loss
SGD	0.01	20	95.92	85.27	0.0682
		30	98.02	86.02	0.1057
Adagrad	0.01	20	96.19	81.97	0.0227
		30	98.74	83.93	0.0180
Adam	0.01	20	89.82	84.48	0.4173
		30	92.94	84.25	0.3629

Testing the dataset on VGG-16 using the SGD optimizer and with the learning rate set to 0.01, the model is able to achieve 71.31% train accuracy and 73.25% test accuracy and loss value of 0.498. Increasing the epochs by a margin of 10, we see a significant rise in performance with the model now able to achieve 90%+ train accuracy and 85%+ test accuracy and the loss value has significantly decreased to 0.056. Setting the learning rate to 0.001, we significant improvement in performance of the model when run for 20 epochs as compared to when the learning rate was 0.01. The model showcases extremely poor performance with Adagrad optimizer with learning rate of 0.01 but performance significantly improves when learning rate is reduced to 0.001. Adam optimizer was not tested with VGG-16 as its not recommended to use it with such a parameter intensive architecture. Testing the dataset on ResNet-34 architecture, the model is consistently able to achieve 95%+ train accuracy and 80%+ test accuracy across all optimizers when the model is ran for both 20 epochs and 30 epochs with learning rate set to 0.01. Testing CIFAR-10 with GoogleNet, the model showcases high training loss value for Adam optimizer for the learning rate of 0.01 as compared to other optimizers. One other interesting thing which is noticed is that the loss value increases when SGD is tested for 30 iterations as compared to when SGD is tested for 20 iterations. With the increase in the number of iterations for which the architecture is trained upon, we see marginal improvement in train and test accuracy values.

VI. CRITICAL ANALYSIS

Deep Neural Networks(DNN) are amongst the most popular set of artificial neural network architectures out there, particularly when it comes to computer vision tasks like image classification, image recognition, etc. The history of such deep neural networks dates back to almost the 1950s when these structures were used for performing a variety of biological experiments. It was not until the 1990s that the first usage of neural networks for image classification was recorded. The usage of Convolutional Neural Networks(CNN) for recognizing a set of handwritten digits was written in the paper[11]. In this paper, Yann LeCun, a French computer scientist trained a CNN with MNIST dataset of handwritten digits. The MNIST database[2] is a dataset of 60,000 train images and 10,000 test images of handwritten digits numbered from 0-9. The CNN architecture which Yann LeCun used for training the network was LeNet-5, a simple CNN architecture containing a total of 60,000 parameters made up of 7 layers(3 convolutional layers, 2 subsampling layers, 2 fully connected layers) which takes a 32x32 image as an input. The LeNet-5 architecture was trained by providing a sample image which the network has to predict, based on the prediction results, the model settings are updated and this process is repeated until the optimal setting have been achieved with the lowest cost function value. The implementation of a neural network architecture for image classification set the standard for most of today's image recognition and image classification applications. Throughout the early 2000s, 2010s, the process of building such neural network architectures was streamlined which facilitated the building of more complex architectures which were able to handle more complex image recognition and classification tasks. In 2010, Fei-Fei Li collaborated with PASCAL VOC team to create the ImageNet dataset, a dataset containing more than 15 million images spread out over 1000 different classes. This dataset formed the backbone of ILSVRC challenge where every year researchers were invited to test their DNN architectures. While MNIST and CIFAR-10 datasets are great for testing out the perfromance of neural network architectures, these datasets fail to provide an ideal representation of the challenges faced while training neural nets in the real world. Due to this reason, the ImageNet dataset was created and used by the ILSVRC competition for a thorough evaluation of deep learning architectures. The VGG-16 architecture which was proposed by K. Simonyan & A.Zisserman belonging to the University of Oxford in the paper[4]. VGG stands for Visual Geometry Group, a group belonging to the department of Engineering Science at the University of Oxford. This architecture was able to achieve a top-5 test accuracy of

92.7% in the ImageNet dataset. The major improvement which this architecture made over the AlexNet architecture proposed in 2012 was that it replace the kernel sizes of 5x5, 7x7 with kernel size of 3x3 in convolutional layers. The smaller kernel size allowed the researchers to build a more deeper network thus improving the network's performance. Although it was a commonly understood phenomenon amongst researchers that a deeper network performs better, but it was noticed that after a particular depth, the performance of the network degrades. This problem was related to the problem of vanishing gradients. As the network depth increased, the gradients using which the loss function was calculated would shrink to zero. The ResNet architecture[6] proposed in 2015 addressed this problem by making use of a skip connection which through which the gradient value can flow through. Alongside the VGG architecture which was proposed in 2014, a variant of the Inception network, GoogleNet[8] was presented at the ILSVRC 2014 challenge. GoogleNet remains at the forefront of a variety of computer vision tasks like image recognition, image classification, face recognition, etc. Amongst all the use cases for which GoogleNet is used for, the task of image classification using GoogleNet forms the lion share of the papers published on GoogleNet. As of today, the 3 architectures discussed in this paper continue to be used for solving a variety of computer vision tasks, most recently for the detection and classification of COVID-19. Several improvements over the vanilla architectures have also been proposed by the researchers. One of the improvements proposed over the original VGG-16 is the design of a 9 layer architecture called as IVGG[12]. IVGG model adds max pooling & dropout after multiple conv layers to extract features from the input image and save on training time. The addition of dropout and batch normalization is proposed after every fully connected layer to accelerate convergence and improve the classification,detection accuracy. Improvements to ResNet have been proposed in the paper[13]. The improved ResNet facilitates the use of adjustable shortcut connections which addresses the problem of gradient fading. The improvements made to the architecture results in the ResNet being able to achieve a increase of 2.85% in the test accuracy when tested on CIFAR-10. Another benefit of this improvement other than train/test accuracy is that it does not increase the computational load on the network. So its safe to say that ever since the original architectures of VGG, ResNet and GoogleNet were proposed in 2014, 2015 and 2014 respectively, significant progress has been made in improving these networks such that these networks achieve near perfect results, especially when the MNIST and CIFAR-10 datasets are taken into consideration.

VII. CONCLUSION

In this paper, we have demonstrated the train,test accuracy and loss values obtained for VGG-16, ResNet-34 and GoogleNet architectures tested over different optimizers, learning rate values and the total number of epochs for which the three architectures have been tested for.

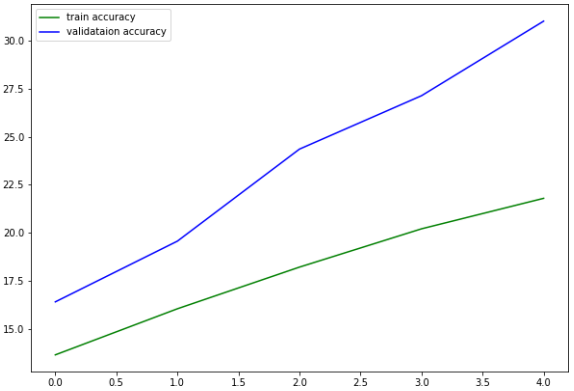
VIII. REFERENCES

1. "The History of Convolutional Neural Networks" https://towardsdatascience.com/a-short-history-of-convolutional-neural-networks-7032e241c483
2. "The MNIST database of handwritten digits" http://yann.lecun.com/exdb/mnist/
3. "The CIFAR-10 dataset" https://www.cs.toronto.edu/~kriz/cifar.html
4. Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).
5. "VGGnet architecture explained" https://medium.com/analytics-vidhya/vggnet-architecture-explained-e5c7318aa5b6
6. He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.
7. "An Overview of ResNet and its Variants" https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035
8. Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1-9. 2015.
9. "Deep Learning: GoogLeNet Explained" https://towardsdatascience.com/deep-learning-googlenet-explained-de8861c82765
10. "An overview of gradient descent optimization algorithms" https://ruder.io/optimizing-gradient-descent/
11. LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.
12. Zhou, Shuren, et al. "Improved VGG model for road traffic sign recognition." *Comput., Mater. Continua* 57.1 (2018): 11-24.
13. Li, Baoqi, and Yuyao He. "An improved ResNet based on the adjustable shortcut connections." *IEEE Access* 6 (2018): 18967-18974.

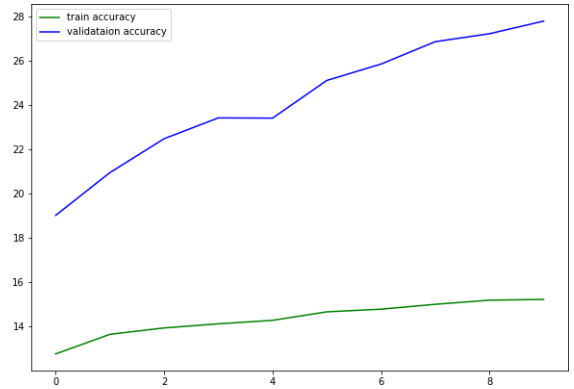
IX. APPENDIX

Train Accuracy vs Validation Accuracy Plots for MNIST dataset:

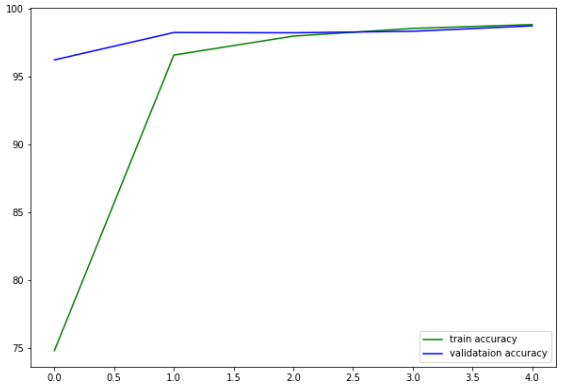
1. Optimizer: Adagrad, LR = 0.01, Epochs: 5
   Architecture: VGG-16



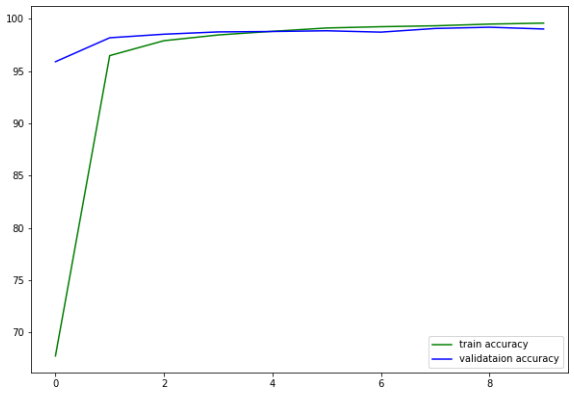
2. Optimizer: Adagrad, LR = 0.01, Epochs: 10
   Architecture: VGG-16



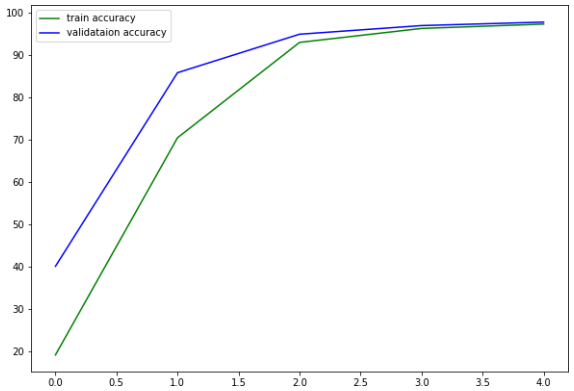
3. Optimizer: Adagrad, LR = 0.001, Epochs: 5
   Architecture: VGG-16



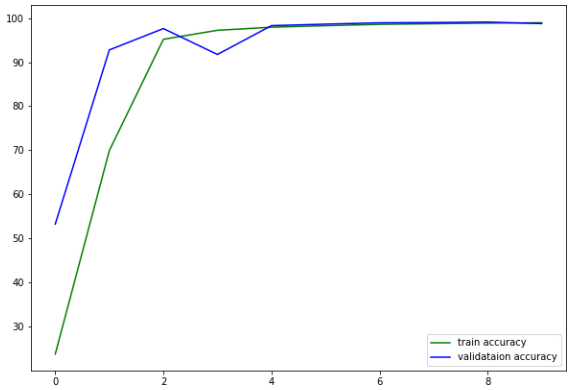
4. Optimizer: Adagrad, LR = 0.001, Epochs: 10
   Architecture: VGG-16



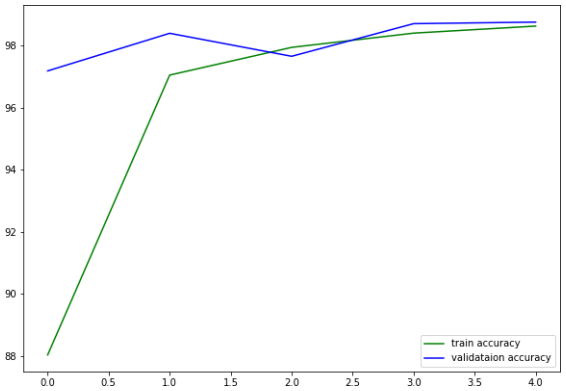
5. Optimizer: SGD, LR = 0.01, Epochs: 5
   Architecture: VGG-16



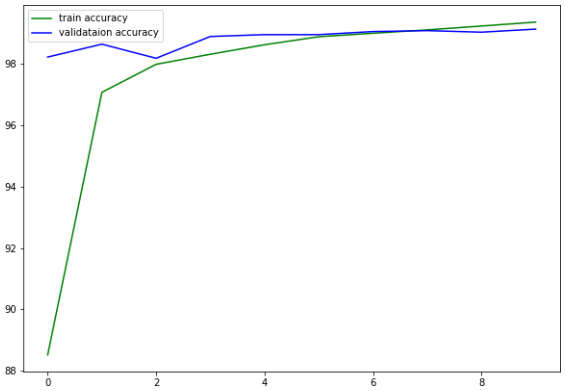
6. Optimizer: SGD, LR = 0.01, Epochs: 10
   Architecture: VGG-16



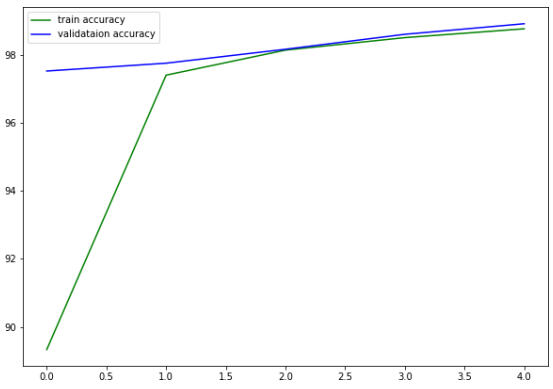
7. Optimizer: SGD, LR = 0.001, Epochs: 5
Architecture: VGG-16



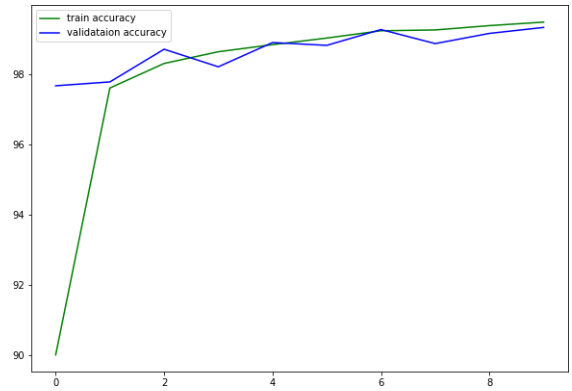
8. Optimizer: SGD, LR = 0.001, Epochs: 10
Architecture: VGG-16



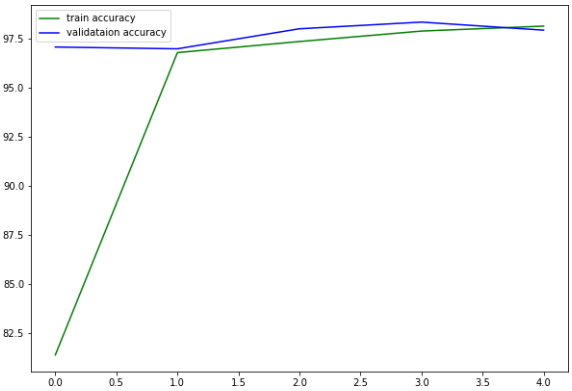
9. Optimizer: Adagrad, LR = 0.01, Epochs: 5
Architecture: ResNet-34



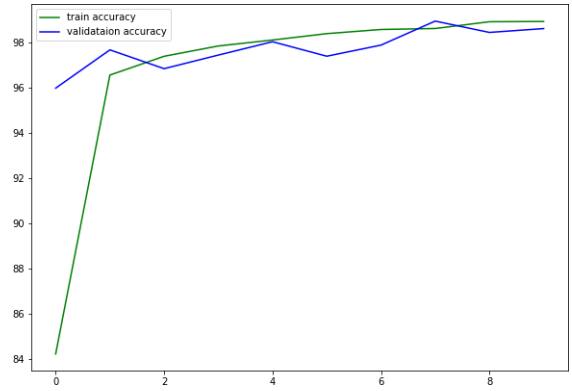
10. Optimizer: Adagrad, LR = 0.01, Epochs: 10
Architecture: ResNet-34



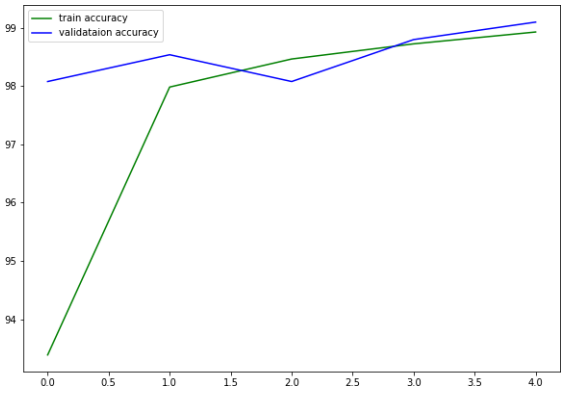
11. Optimizer: Adam, LR = 0.01, Epochs: 5
Architecture: ResNet-34



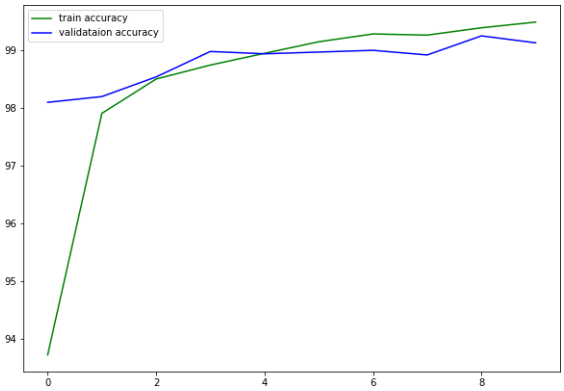
12. Optimizer: Adam, LR = 0.01, Epochs: 10
Architecture: ResNet-34



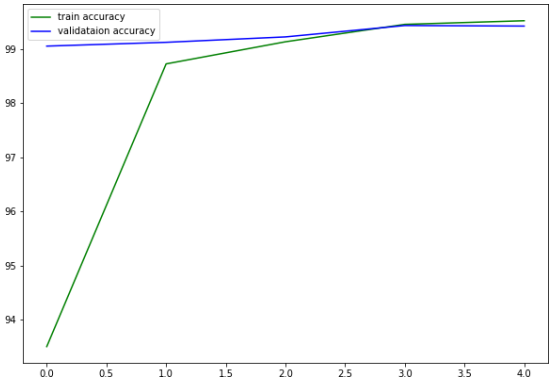
13. Optimizer: SGD, LR = 0.01, Epochs: 5
Architecture: ResNet-34



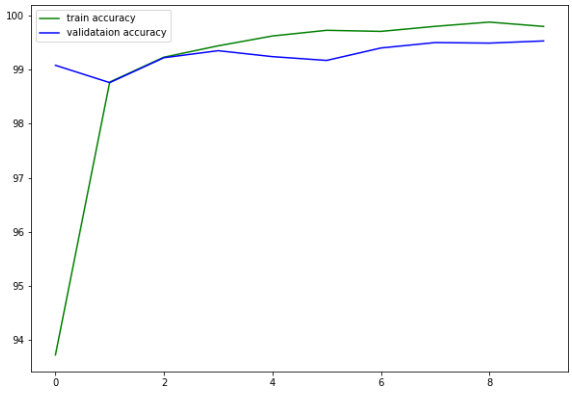
14. Optimizer: SGD, LR = 0.01, Epochs: 10
Architecture: ResNet-34



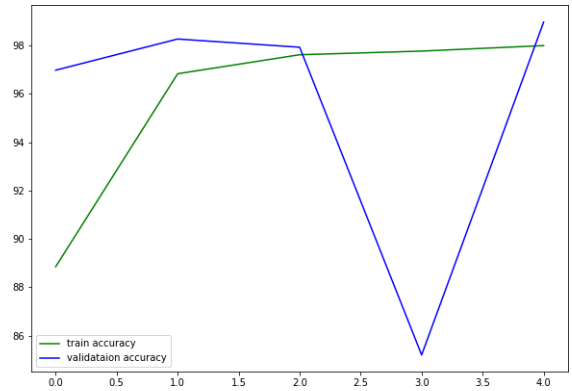
15. Optimizer: Adagrad, LR = 0.01, Epochs: 5
Architecture: GoogleNet



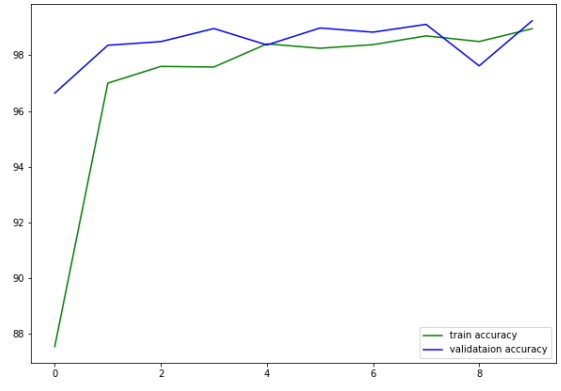
16. Optimizer: Adagrad, LR = 0.01, Epochs: 10
Architecture: GoogleNet



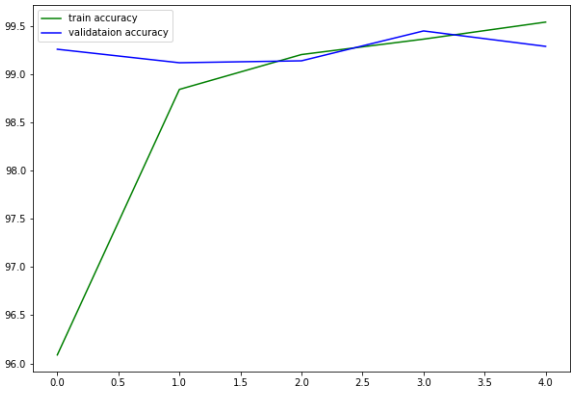
17. Optimizer: Adam, LR = 0.01, Epochs: 5
Architecture: GoogleNet



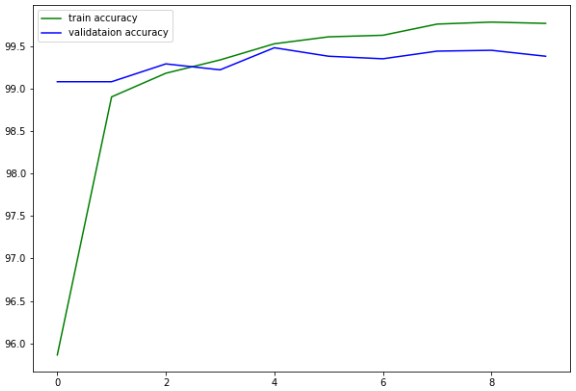
18. Optimizer: Adam, LR = 0.01, Epochs: 10
Architecture: GoogleNet



19. Optimizer: SGD, LR = 0.01, Epochs: 5
Architecture: GoogleNet

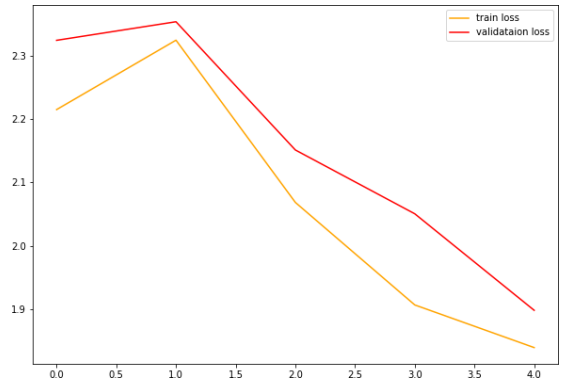


20. Optimizer: SGD, LR = 0.01, Epochs: 10
Architecture: GoogleNet

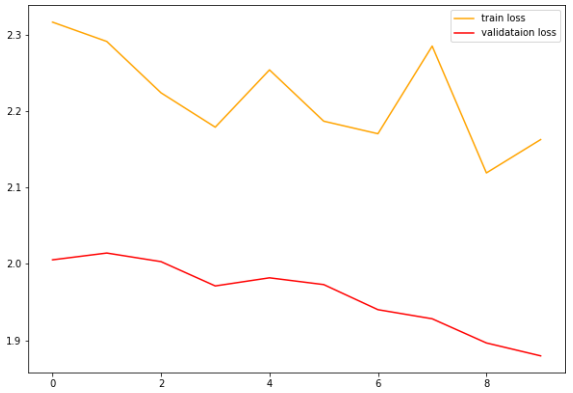


Train Loss vs Validation Loss Plots for MNIST dataset:

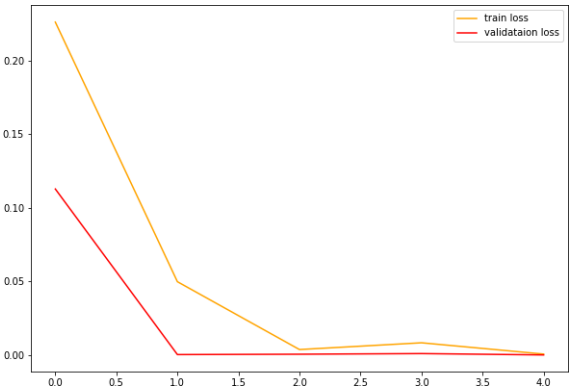
1. Optimizer: Adagrad, LR = 0.01, Epochs: 5
Architecture: VGG-16



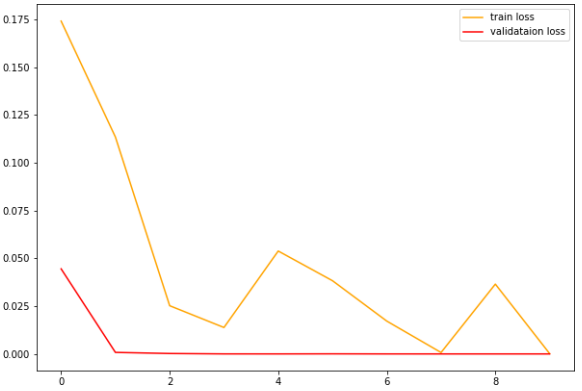
2. Optimizer: Adagrad, LR = 0.01, Epochs: 10
Architecture: VGG-16



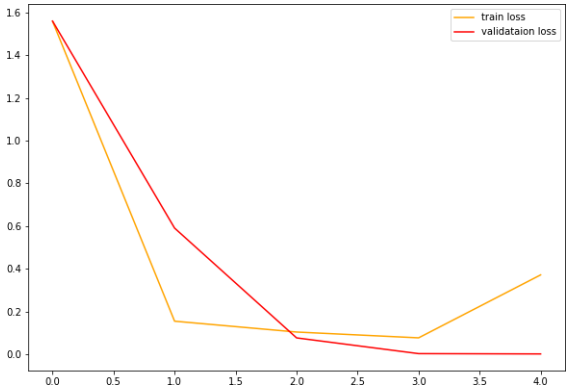
3. Optimizer: Adagrad, LR = 0.001, Epochs: 5
Architecture: VGG-16



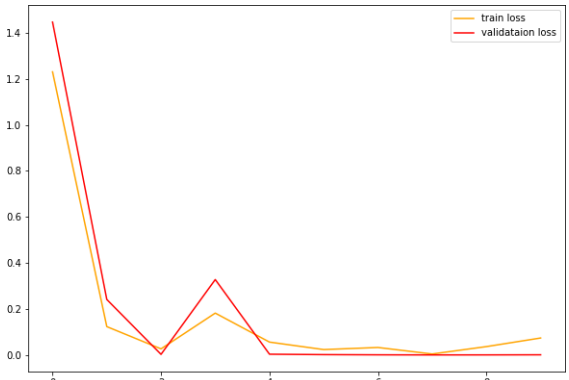
4. Optimizer: Adagrad, LR = 0.001, Epochs: 10
Architecture: VGG-16



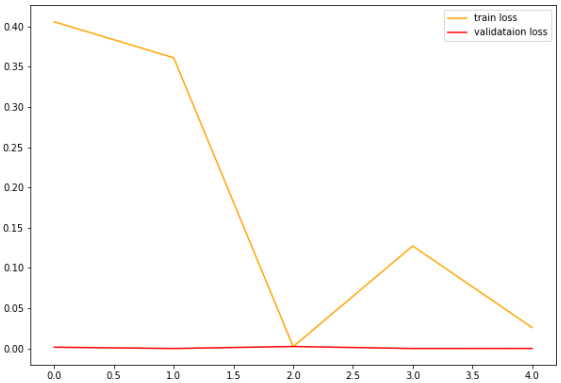
5. Optimizer: SGD, LR = 0.01, Epochs: 5
Architecture: VGG-16



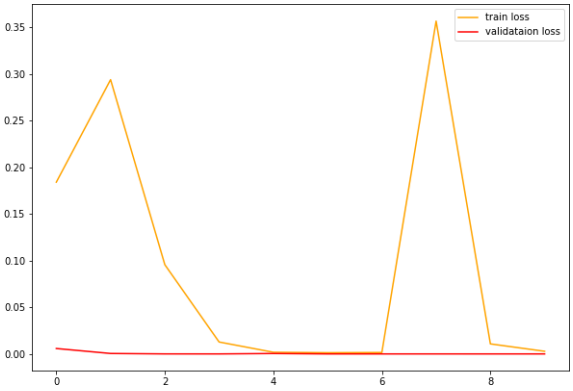
6. Optimizer: SGD, LR = 0.01, Epochs: 10
Architecture: VGG-16



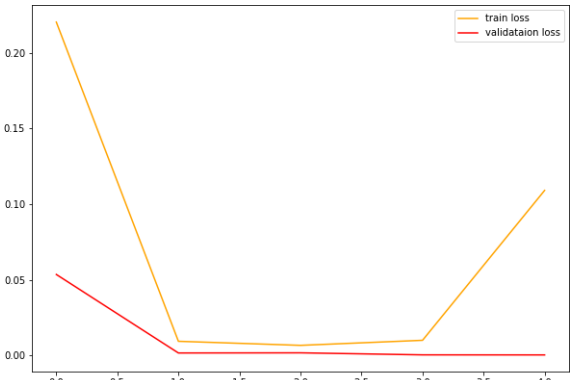
7. Optimizer: SGD, LR = 0.001, Epochs: 5
Architecture: VGG-16



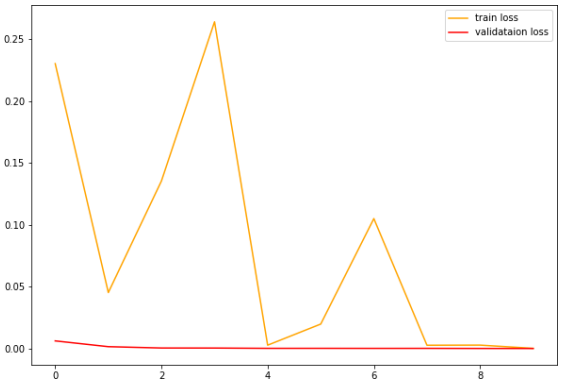
8. Optimizer: SGD, LR = 0.001, Epochs: 10
Architecture: VGG-16



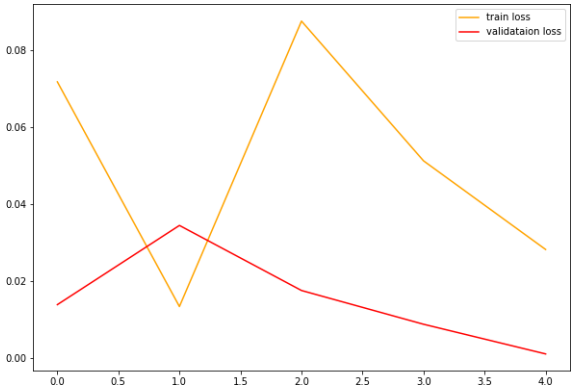
9. Optimizer: Adagrad, LR = 0.01, Epochs: 5
Architecture: ResNet-34



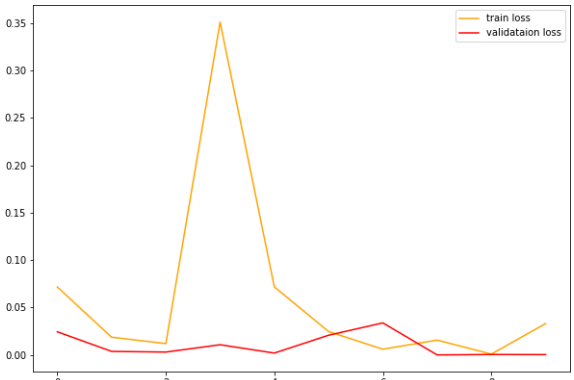
10. Optimizer: Adagrad, LR = 0.01, Epochs: 10
Architecture: ResNet-34



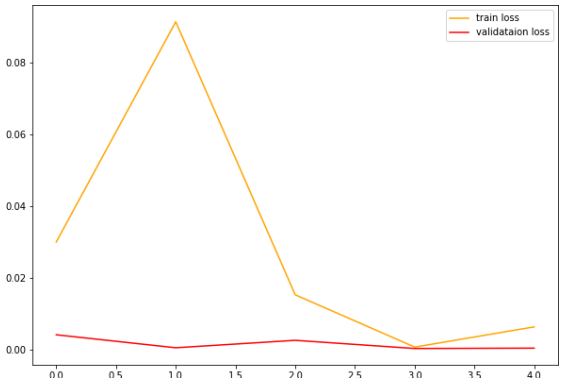
11. Optimizer: Adam, LR = 0.01, Epochs: 5
Architecture: ResNet-34



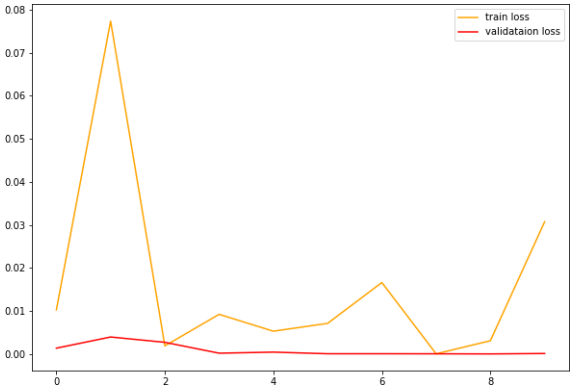
12. Optimizer: Adam, LR = 0.01, Epochs: 10
Architecture: ResNet-34



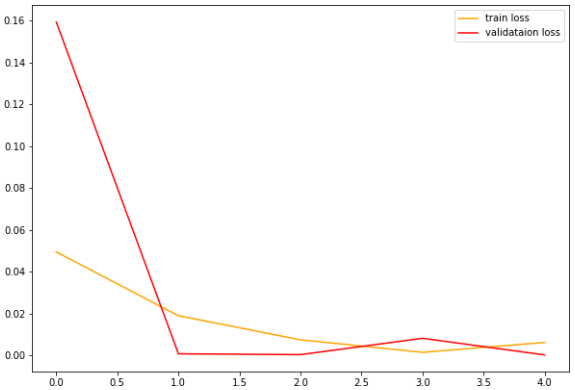
13. Optimizer: SGD, LR = 0.01, Epochs: 5
Architecture: ResNet-34



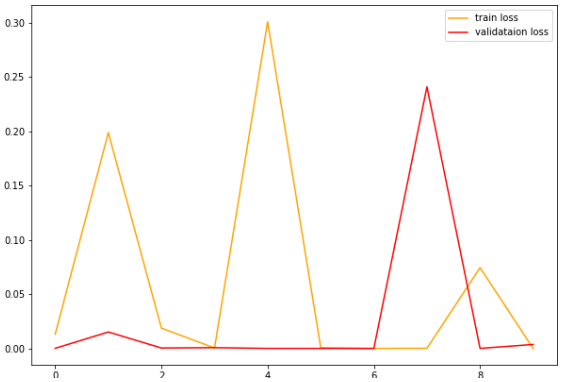
14. Optimizer: SGD, LR = 0.01, Epochs: 10
Architecture: ResNet-34



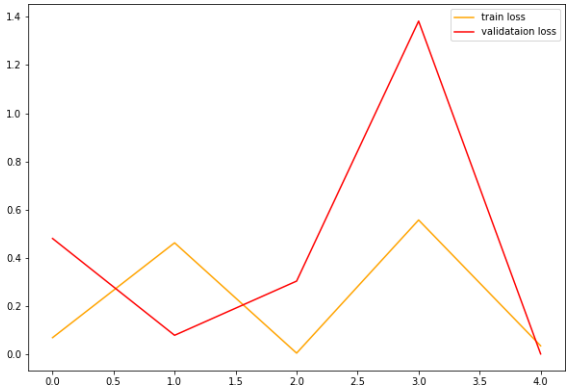
15. Optimizer: Adagrad, LR = 0.01, Epochs: 5
Architecture: GoogleNet



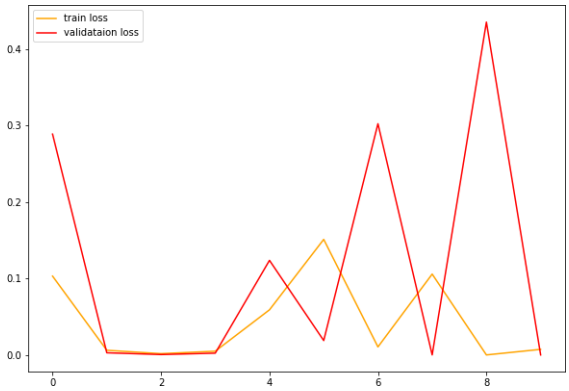
16. Optimizer: Adagrad, LR = 0.01, Epochs: 10
Architecture: GoogleNet



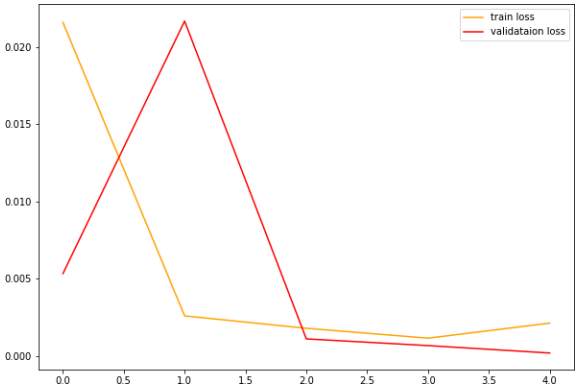
17. Optimizer: Adam, LR = 0.01, Epochs: 5
Architecture: GoogleNet



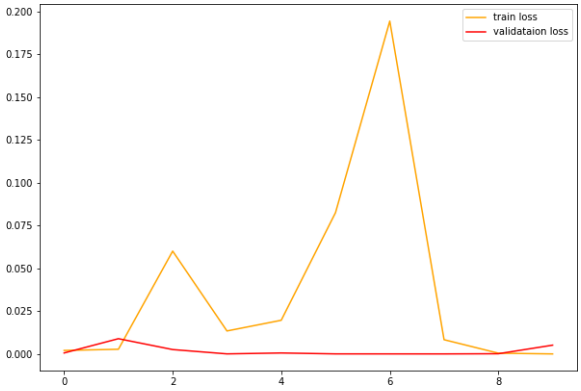
18. Optimizer: Adam, LR = 0.01, Epochs: 10
Architecture: GoogleNet



19. Optimizer: SGD, LR = 0.01, Epochs: 5
Architecture: GoogleNet



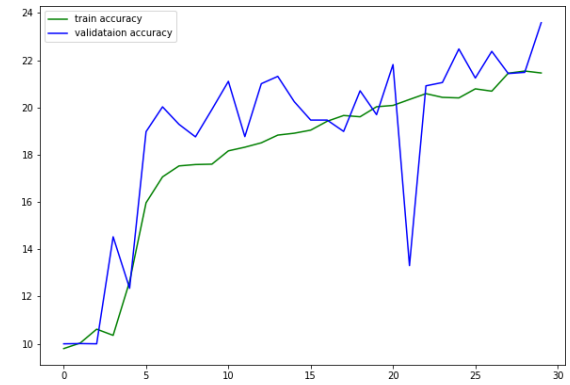
20. Optimizer: SGD, LR = 0.01, Epochs: 10
Architecture: GoogleNet



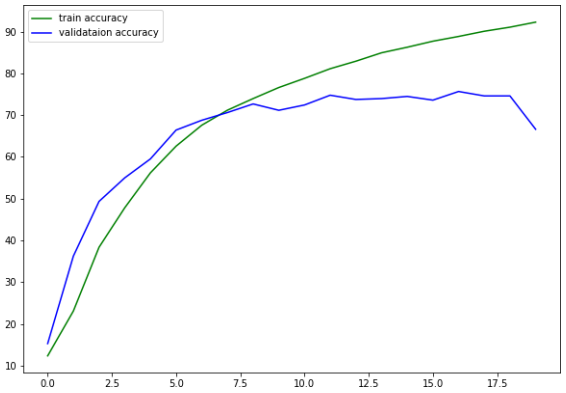
Train Accuracy vs Validation Accuracy Plots for CIFAR-10 dataset:

1. Optimizer: Adagrad, LR = 0.01, Epochs: 20
Architecture: VGG-16

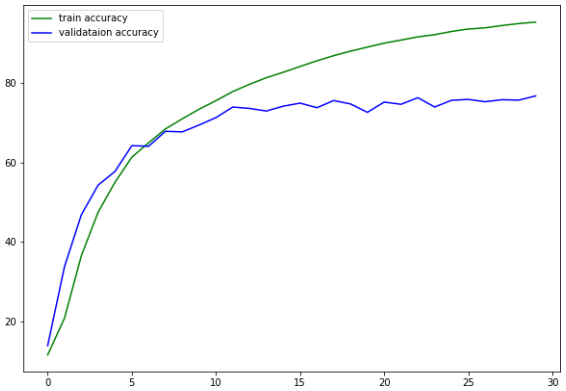
2. Optimizer: Adagrad, LR = 0.01, Epochs: 30
Architecture: VGG-16



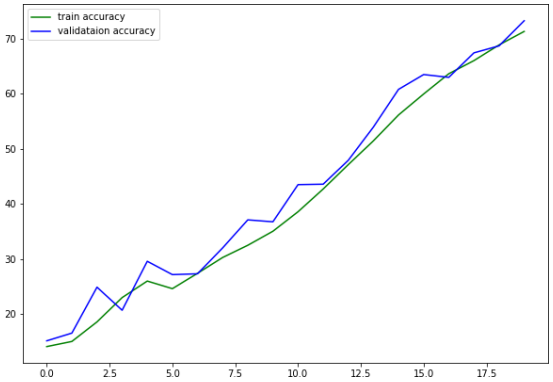
3. Optimizer: Adagrad, LR = 0.001, Epochs: 20
Architecture: VGG-16



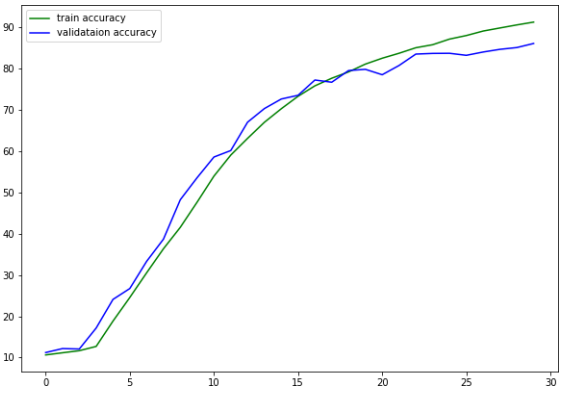
4. Optimizer: Adagrad, LR = 0.001, Epochs: 30
Architecture: VGG-16



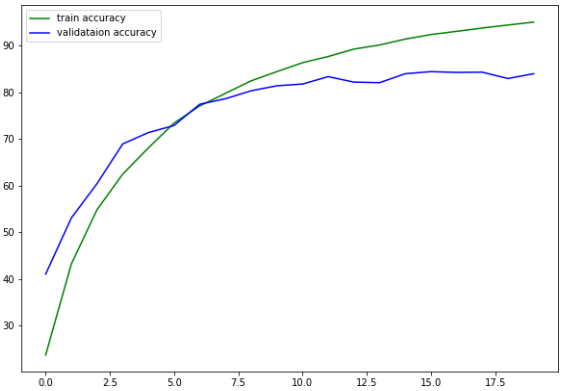
5. Optimizer: SGD, LR = 0.01, Epochs: 20
Architecture: VGG-16



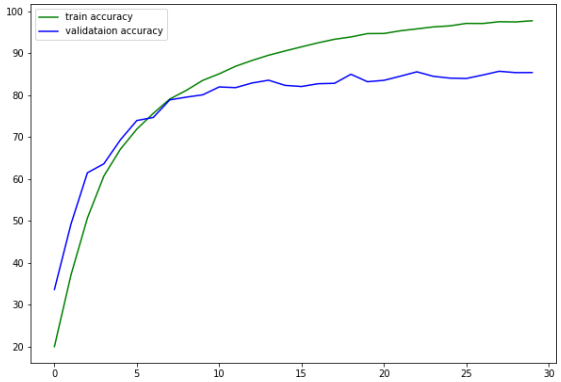
6. Optimizer: SGD, LR = 0.01, Epochs: 30
Architecture: VGG-16



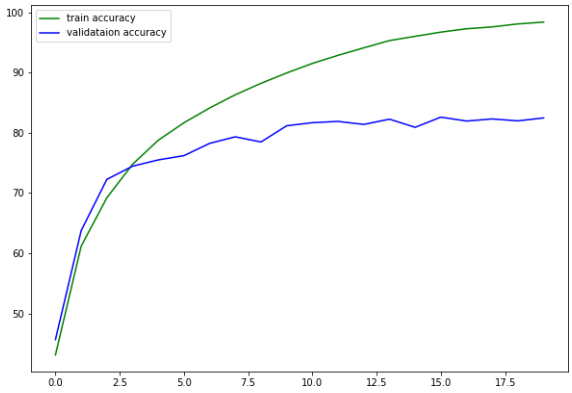
7. Optimizer: SGD, LR = 0.001, Epochs: 20
Architecture: VGG-16



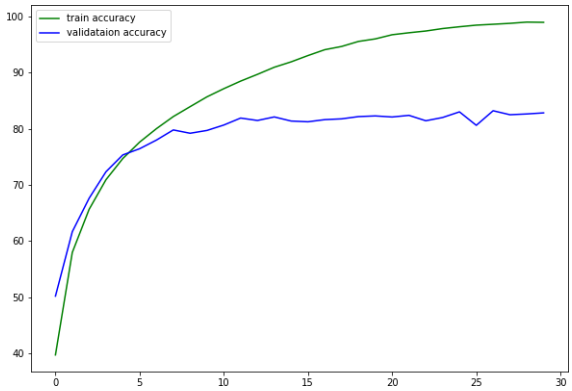
8. Optimizer: SGD, LR = 0.001, Epochs: 30
Architecture: VGG-16



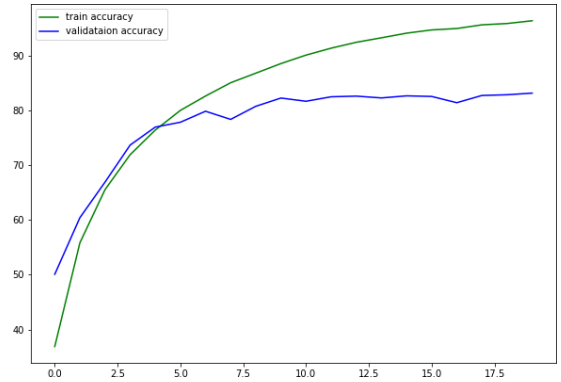
9. Optimizer: Adagrad, LR = 0.01, Epochs: 20
Architecture: ResNet-34



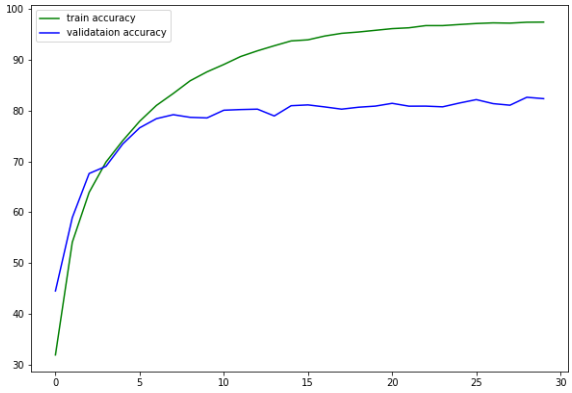
10. Optimizer: Adagrad, LR = 0.01, Epochs: 30
Architecture: ResNet-34



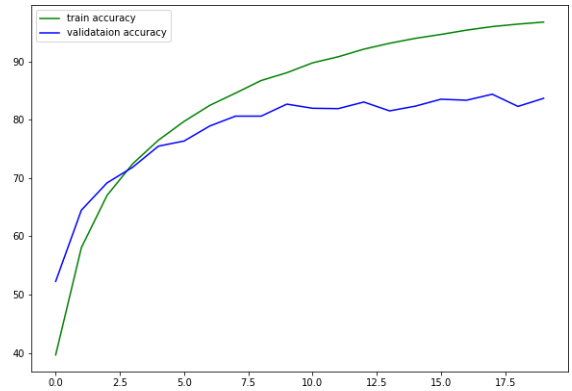
11. Optimizer: Adam, LR = 0.01, Epochs: 20
Architecture: ResNet-34



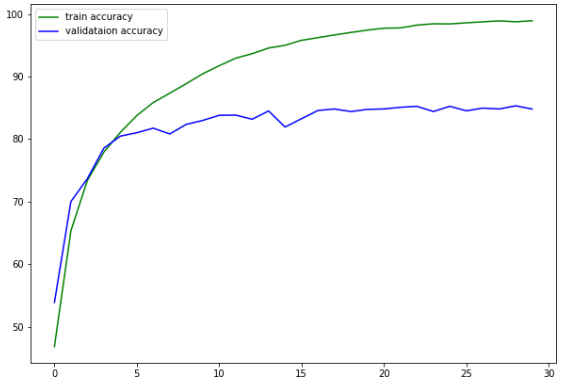
12. Optimizer: Adam, LR = 0.01, Epochs: 30
Architecture: ResNet-34



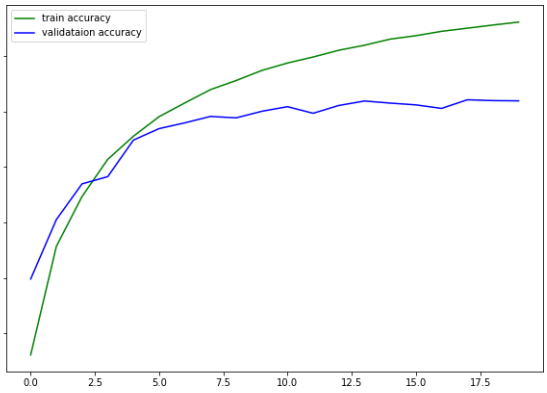
13. Optimizer: SGD, LR = 0.01, Epochs: 20
Architecture: ResNet-34



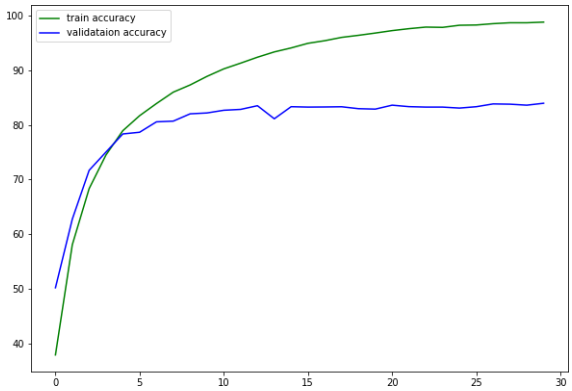
14. Optimizer: SGD, LR = 0.01, Epochs: 30
Architecture: ResNet-34



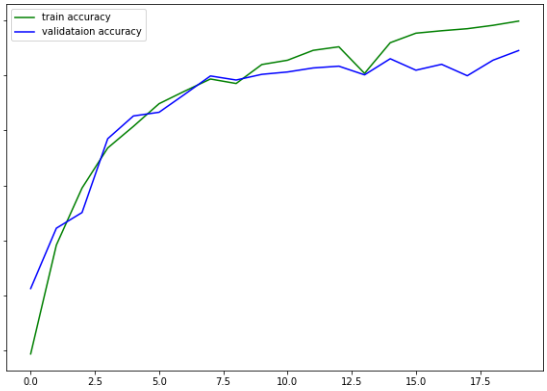
15. Optimizer: Adagrad, LR = 0.01, Epochs: 20
Architecture: GoogleNet



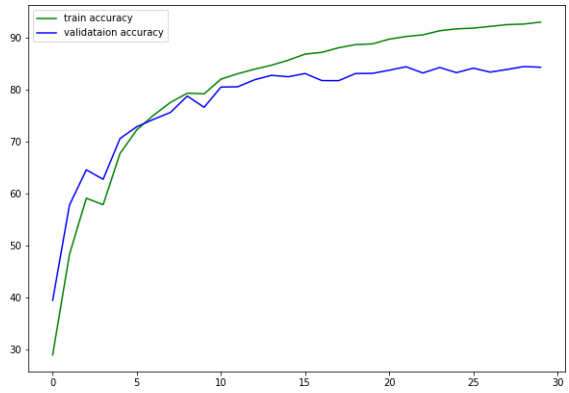
16. Optimizer: Adagrad, LR = 0.01, Epochs: 30
Architecture: GoogleNet



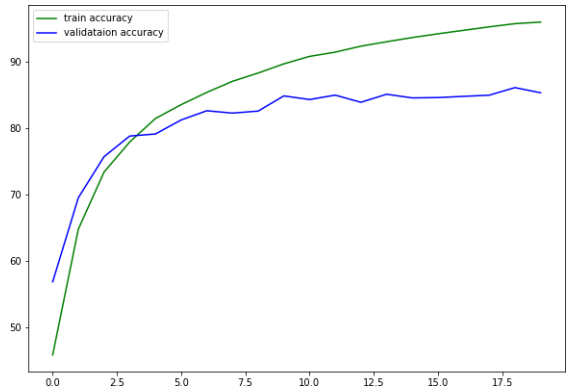
17. Optimizer: Adam, LR = 0.01, Epochs: 20
Architecture: GoogleNet



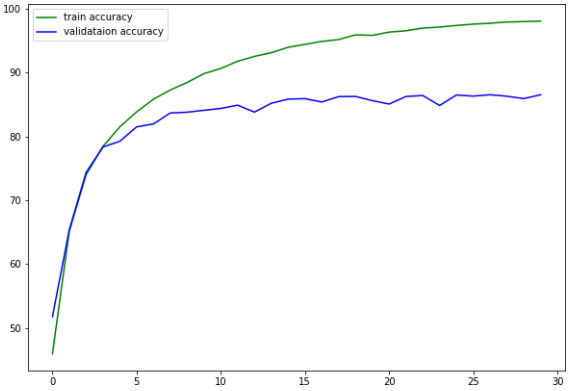
18. Optimizer: Adam, LR = 0.01, Epochs: 30
Architecture: GoogleNet



19. Optimizer: SGD, LR = 0.01, Epochs: 20
Architecture: GoogleNet



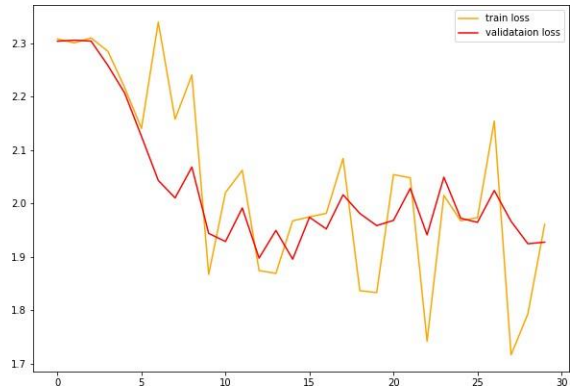
20. Optimizer: SGD, LR = 0.01, Epochs: 30
Architecture: GoogleNet



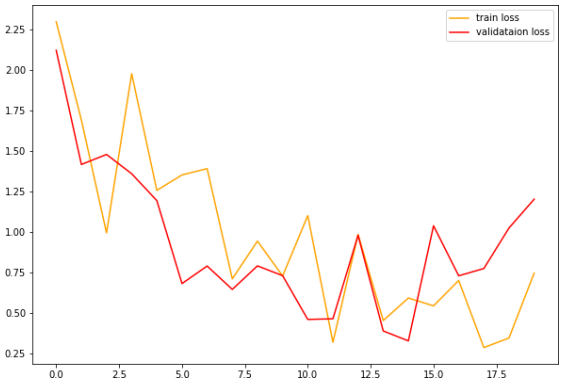
Train Loss vs Validation Loss Plots for CIFAR-10 dataset:

1. Optimizer: Adagrad, LR = 0.01, Epochs: 20
Architecture: VGG-16

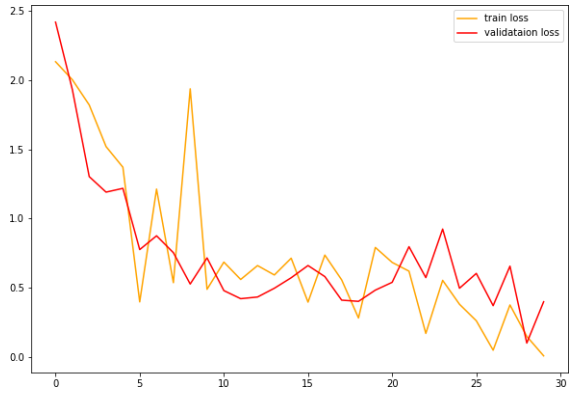
2. Optimizer: Adagrad, LR = 0.01, Epochs: 30
Architecture: VGG-16



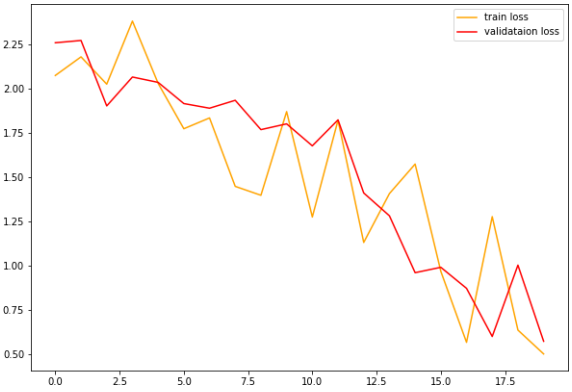
3. Optimizer: Adagrad, LR = 0.001, Epochs: 20
Architecture: VGG-16



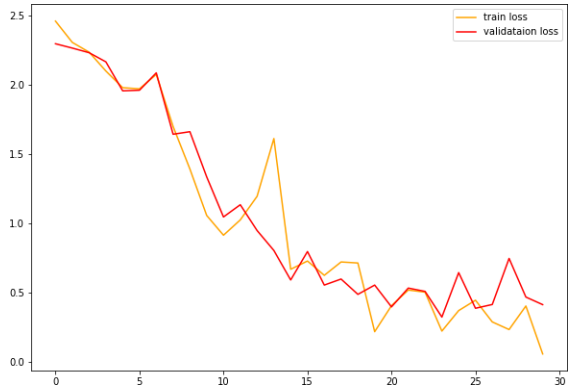
4. Optimizer: Adagrad, LR = 0.001, Epochs: 30
Architecture: VGG-16



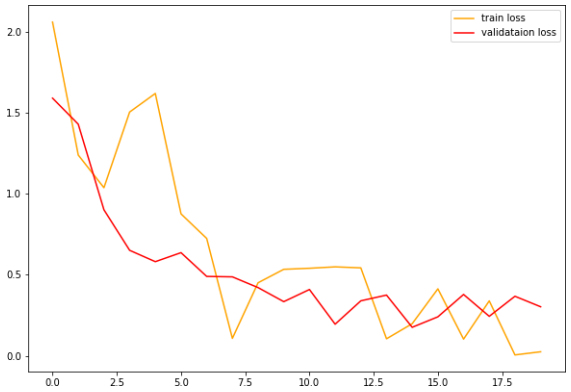
5. Optimizer: SGD, LR = 0.01, Epochs: 20
Architecture: VGG-16



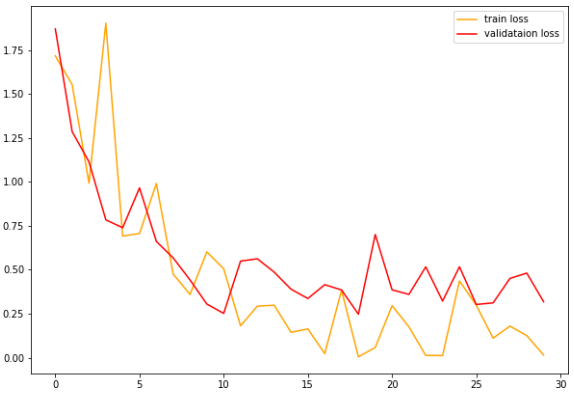
6. Optimizer: SGD, LR = 0.01, Epochs: 30
Architecture: VGG-16



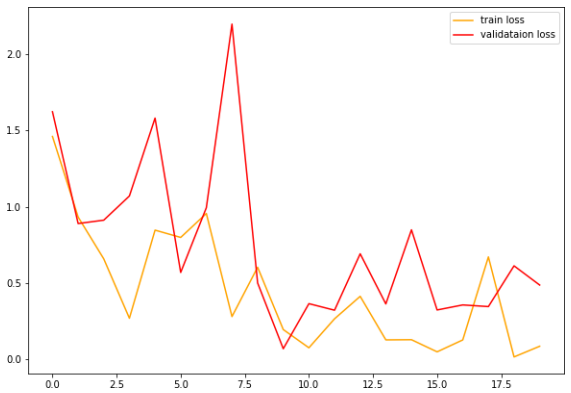
7. Optimizer: SGD, LR = 0.001, Epochs: 20
Architecture: VGG-16



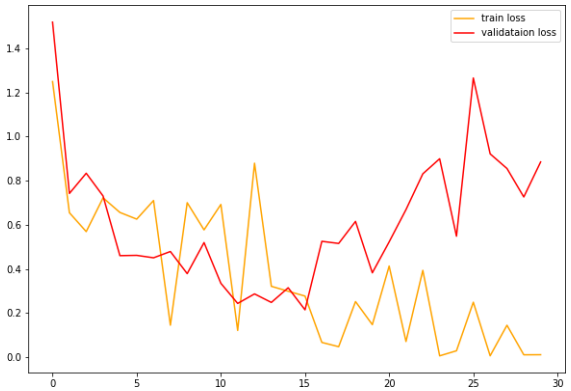
8. Optimizer: SGD, LR = 0.001, Epochs: 30
Architecture: VGG-16



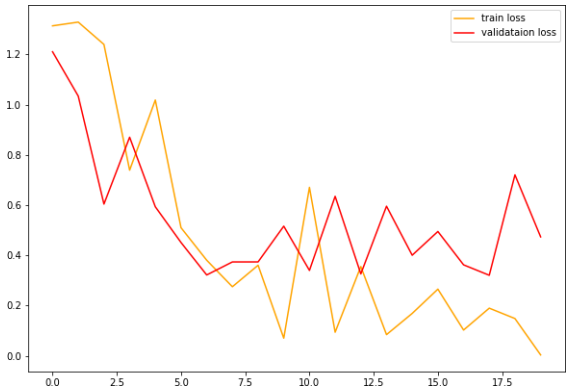
9. Optimizer: Adagrad, LR = 0.01, Epochs: 20
Architecture: ResNet-34



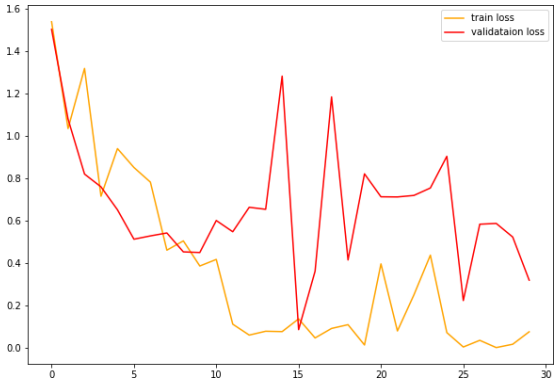
10. Optimizer: Adagrad, LR = 0.01, Epochs: 30
Architecture: ResNet-34



11. Optimizer: Adam, LR = 0.01, Epochs: 20
Architecture: ResNet-34

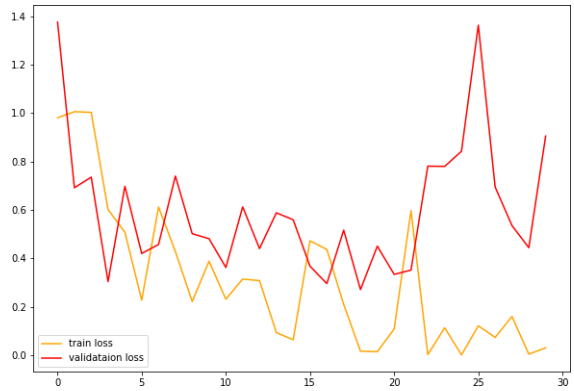


12. Optimizer: Adam, LR = 0.01, Epochs: 30
Architecture: ResNet-34

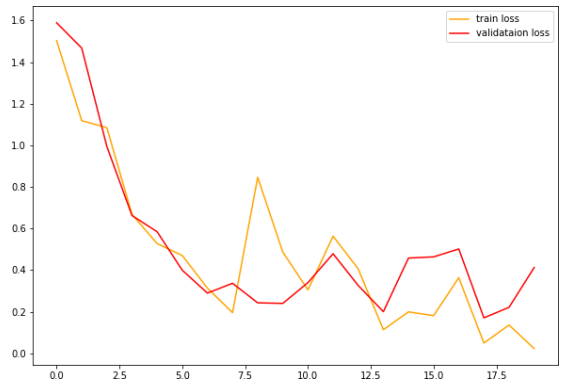


13. Optimizer: SGD, LR = 0.01, Epochs: 20
Architecture: ResNet-34

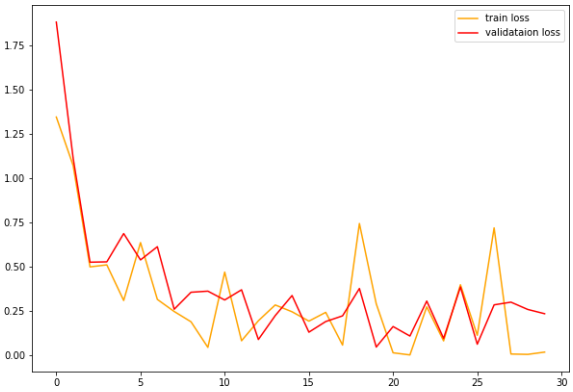
14. Optimizer: SGD, LR = 0.01, Epochs: 30
Architecture: ResNet-34



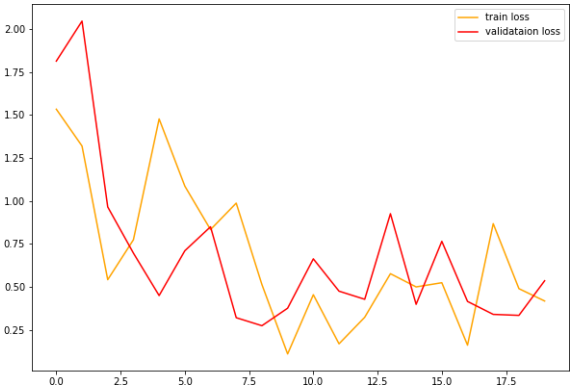
15. Optimizer: Adagrad, LR = 0.01, Epochs: 20
Architecture: GoogleNet



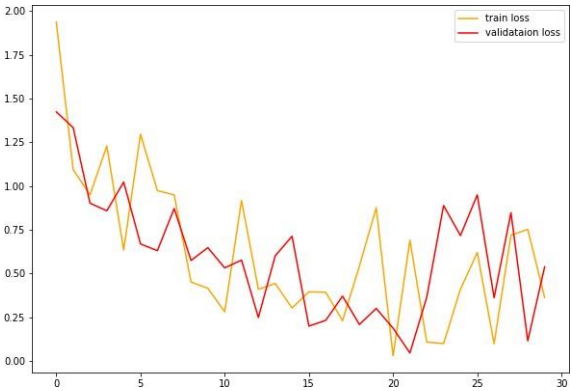
16. Optimizer: Adagrad, LR = 0.01, Epochs: 30
Architecture: GoogleNet



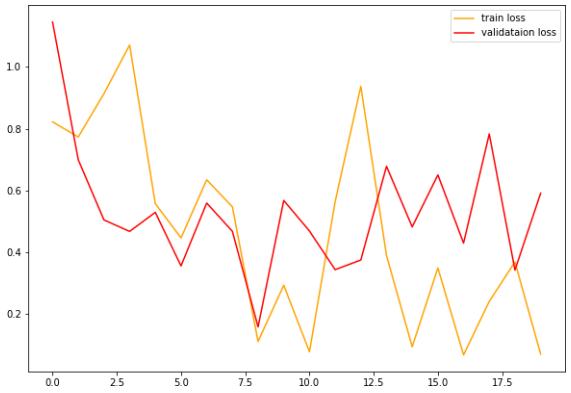
17. Optimizer: Adam, LR = 0.01, Epochs: 20
Architecture: GoogleNet



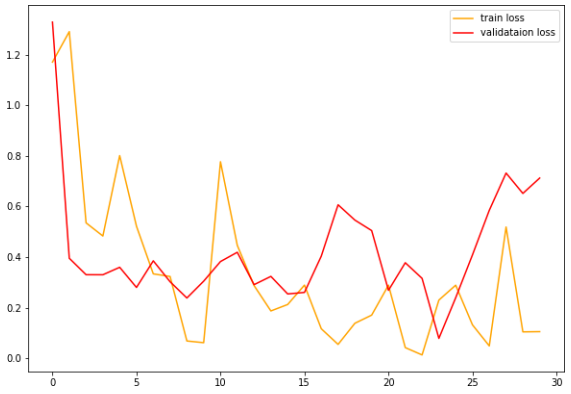
18. Optimizer: Adam, LR = 0.01, Epochs: 30
Architecture: GoogleNet



19. Optimizer: SGD, LR = 0.01, Epochs: 20
Architecture: GoogleNet

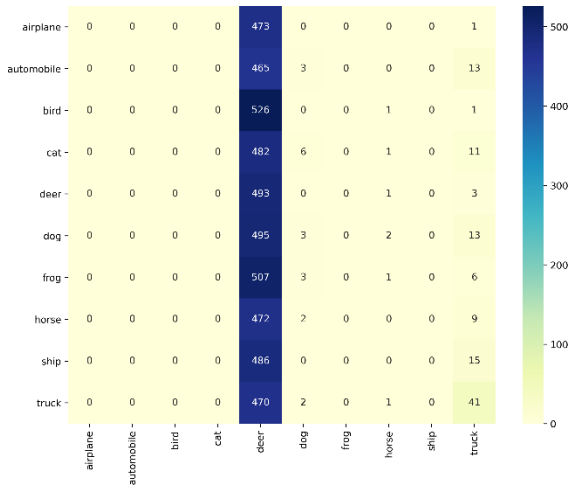


20. Optimizer: SGD, LR = 0.01, Epochs: 30
Architecture: GoogleNet

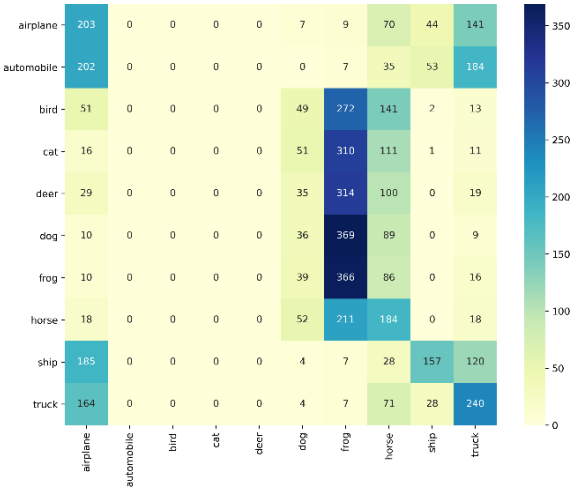


Heatmap Visualization of classified & misclassified examples of CIFAR-10 dataset:

1. Optimizer: Adagrad, LR = 0.01, Epochs: 20
Architecture: VGG-16

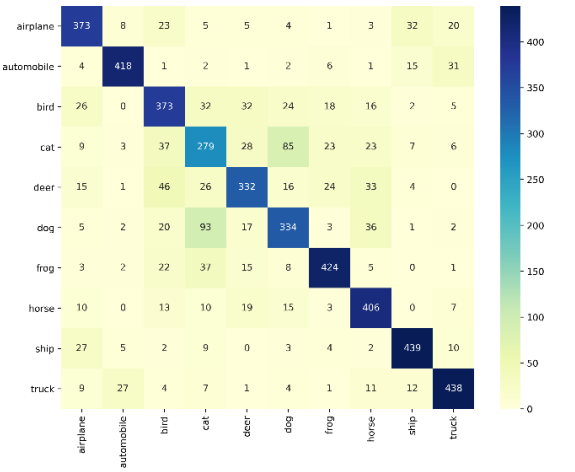


2. Optimizer: Adagrad, LR = 0.01, Epochs: 30
Architecture: VGG-16

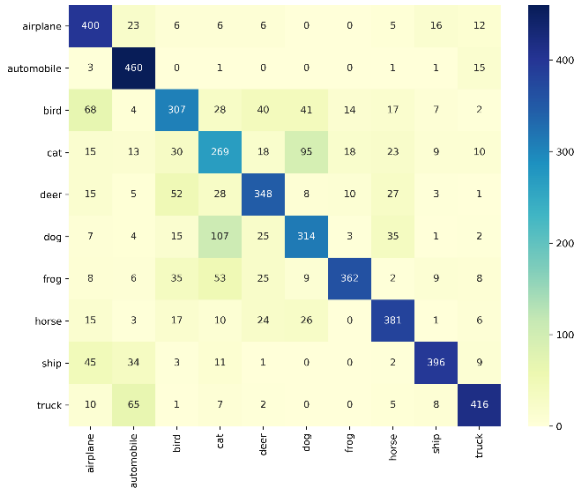


3. Optimizer: Adagrad, LR = 0.001, Epochs: 20
Architecture: VGG-16

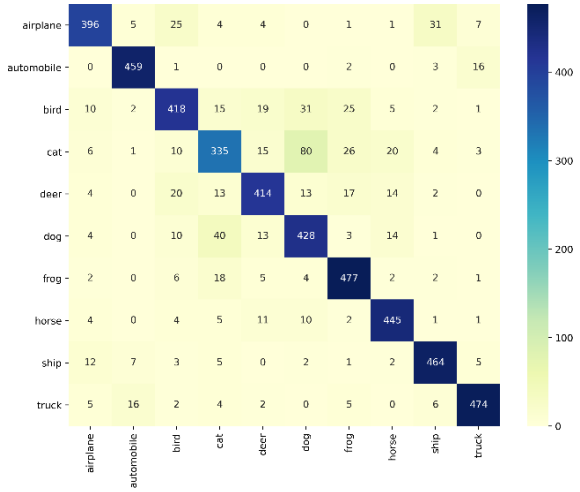
4. Optimizer: Adagrad, LR = 0.001, Epochs: 30
Architecture: VGG-16



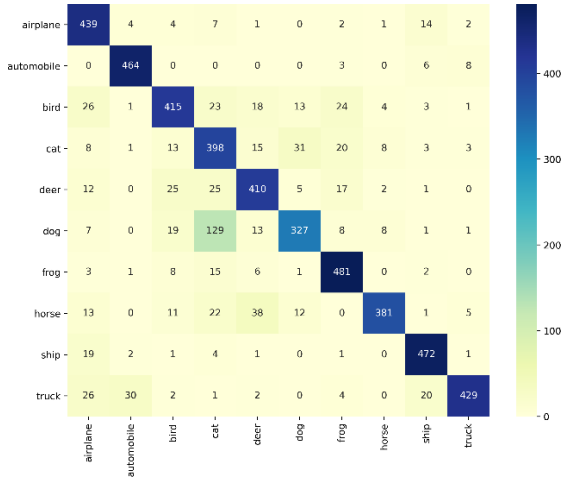
5. Optimizer: SGD, LR = 0.01, Epochs: 20
Architecture: VGG-16



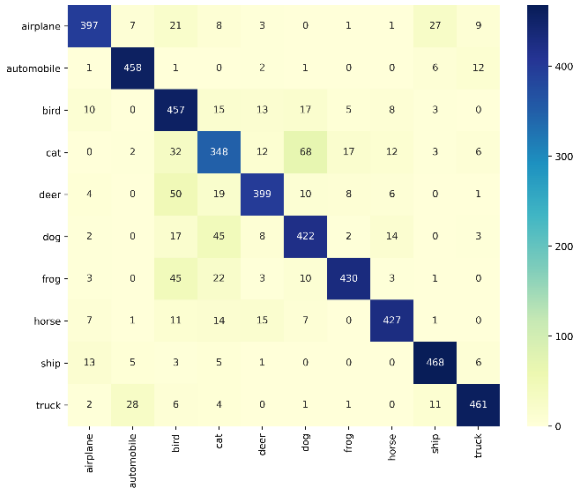
6. Optimizer: SGD, LR = 0.01, Epochs: 30
Architecture: VGG-16



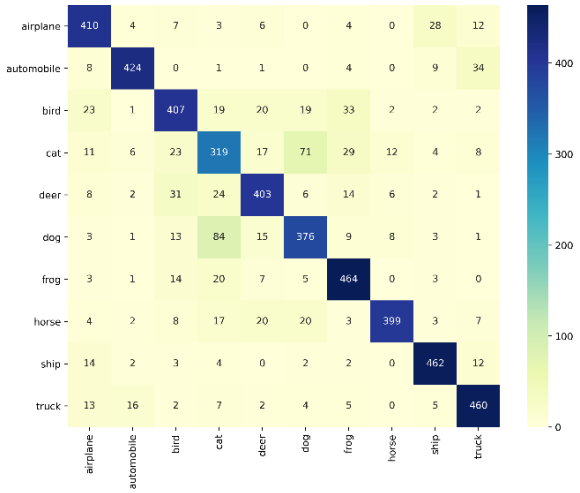
7. Optimizer: SGD, LR = 0.001, Epochs: 20
Architecture: VGG-16



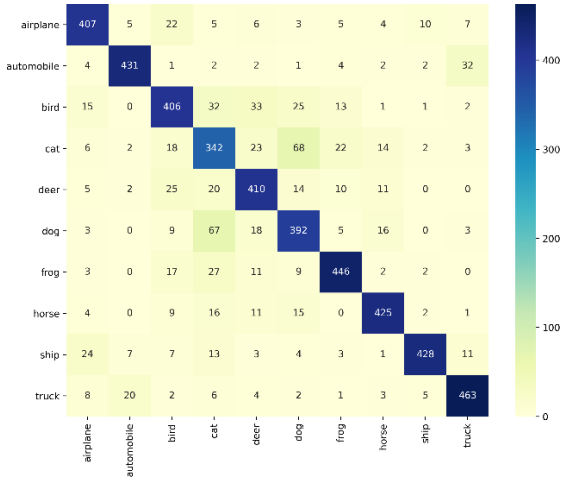
8. Optimizer: SGD, LR = 0.001, Epochs: 30
Architecture: VGG-16



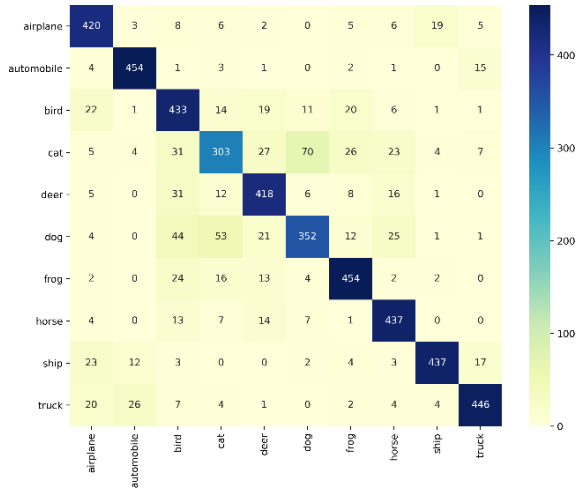
9. Optimizer: Adagrad, LR = 0.01, Epochs: 20
Architecture: ResNet-34



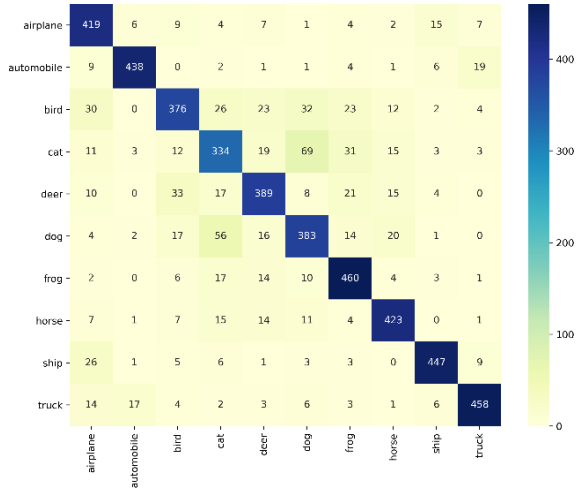
10. Optimizer: Adagrad, LR = 0.01, Epochs: 30
Architecture: ResNet-34



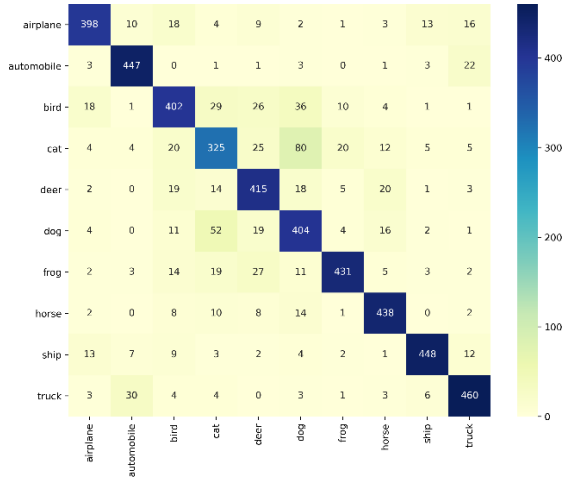
11. Optimizer: Adam, LR = 0.01, Epochs: 20
Architecture: ResNet-34



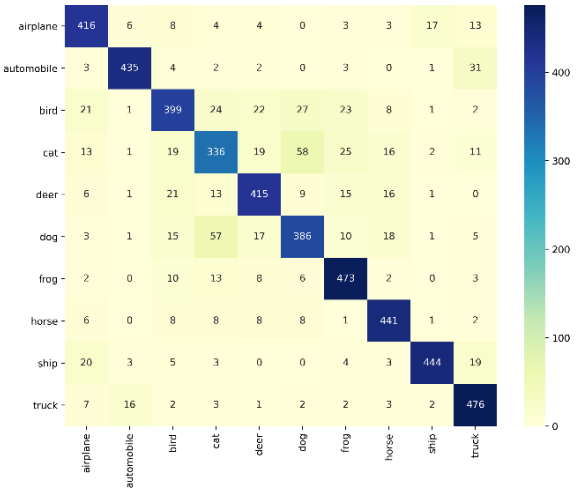
12. Optimizer: Adam, LR = 0.01, Epochs: 30
Architecture: ResNet-34



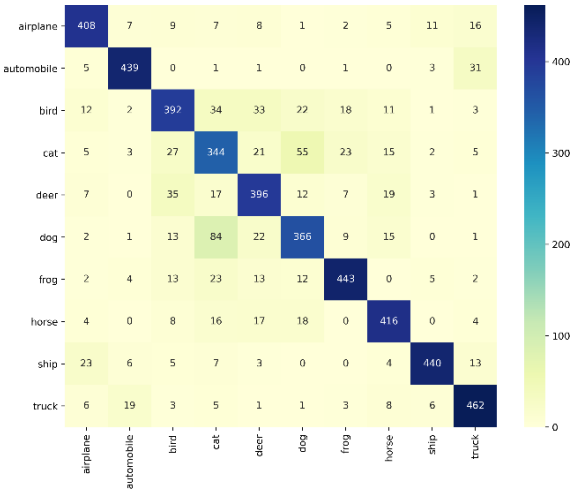
13. Optimizer: SGD, LR = 0.01, Epochs: 20
Architecture: ResNet-34



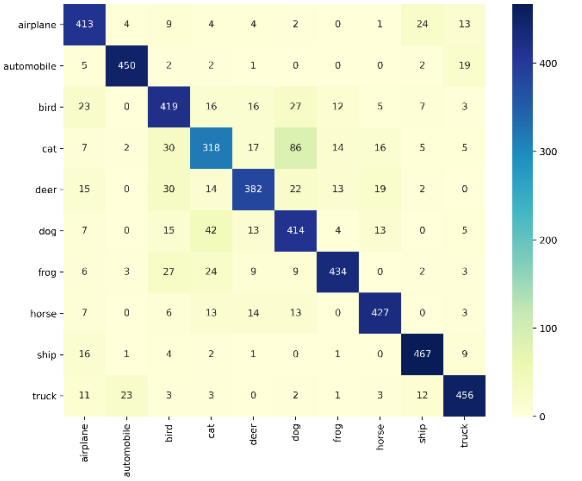
14. Optimizer: SGD, LR = 0.01, Epochs: 30
Architecture: ResNet-34



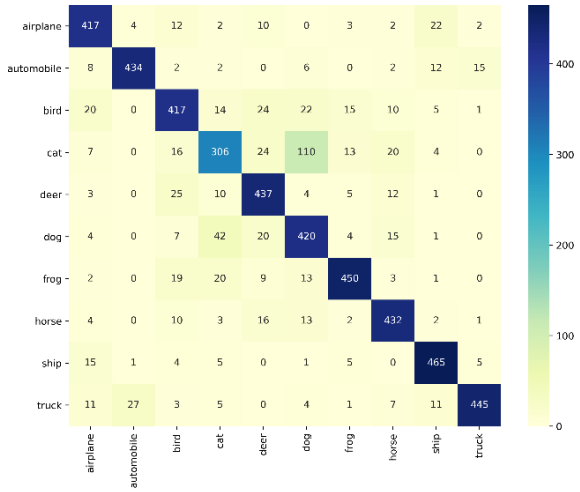
15. Optimizer: Adagrad, LR = 0.01, Epochs: 20
Architecture: GoogleNet



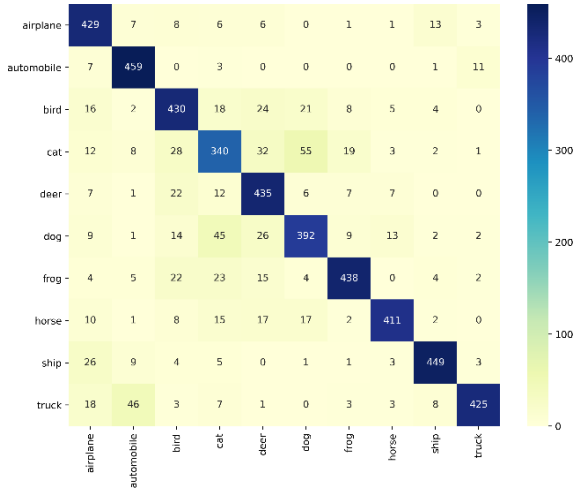
16. Optimizer: Adagrad, LR = 0.01, Epochs: 30
Architecture: GoogleNet



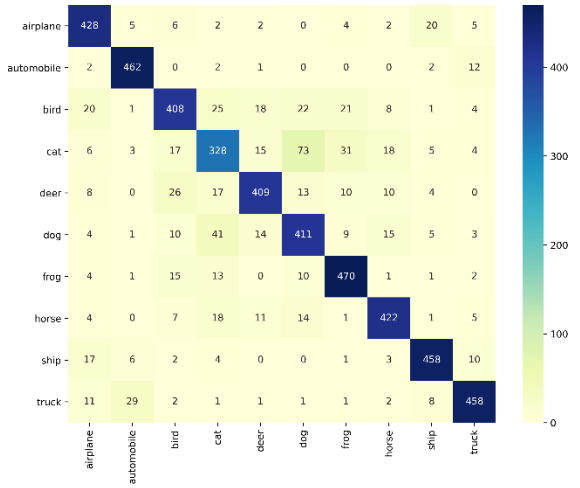
17. Optimizer: Adam, LR = 0.01, Epochs: 20
Architecture: GoogleNet



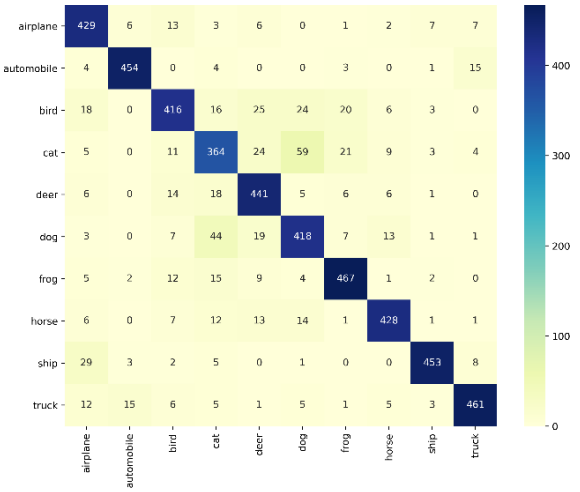
18. Optimizer: Adam, LR = 0.01, Epochs: 30
Architecture: GoogleNet



19. Optimizer: SGD, LR = 0.01, Epochs: 20
Architecture: GoogleNet

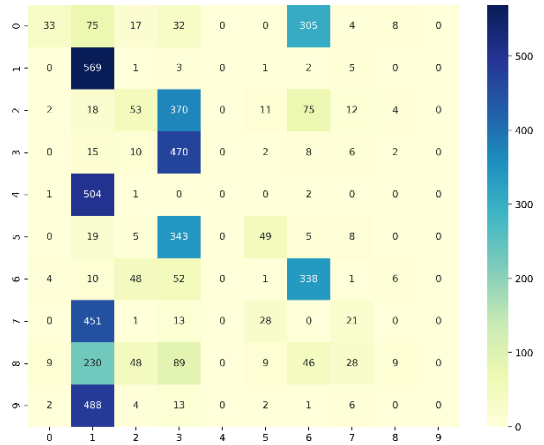


20. Optimizer: SGD, LR = 0.01, Epochs: 30
Architecture: GoogleNet

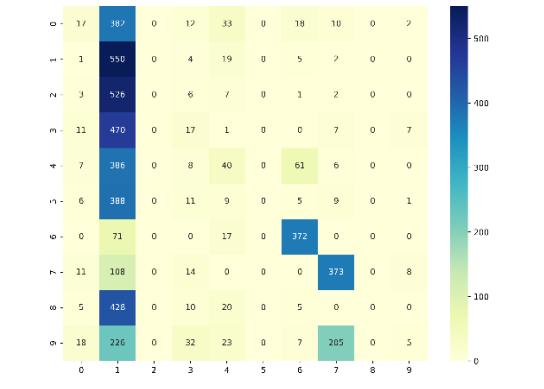


Heatmap Visualization of classified & misclassified examples of MNIST dataset:

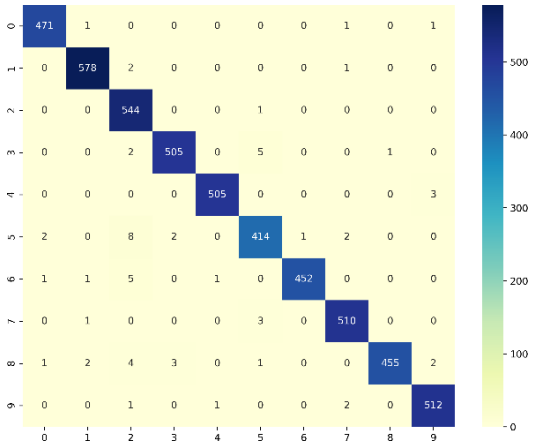
1. Optimizer: Adagrad, LR = 0.01, Epochs: 5
Architecture: VGG-16



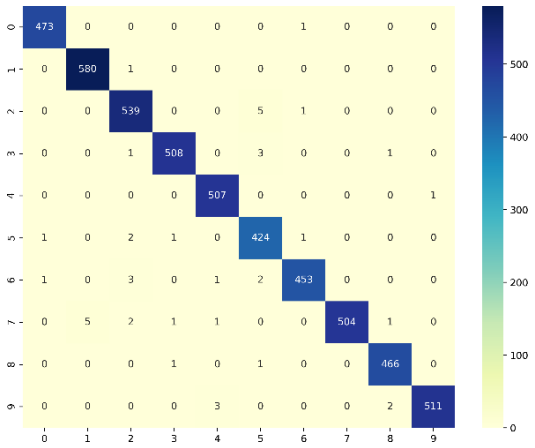
2. Optimizer: Adagrad, LR = 0.01, Epochs: 10
Architecture: VGG-16



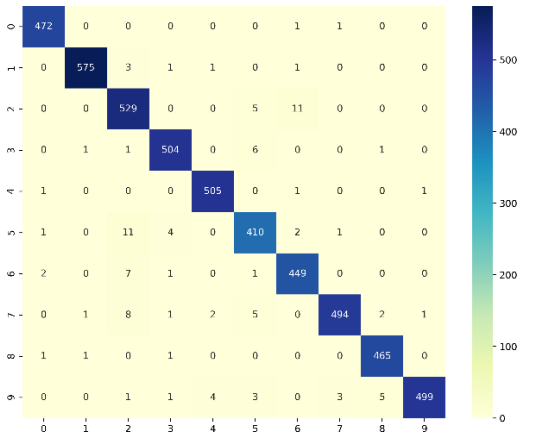
3. Optimizer: Adagrad, LR = 0.001, Epochs: 5
Architecture: VGG-16



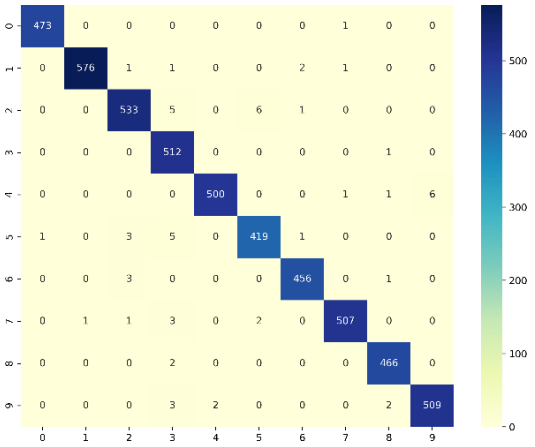
4. Optimizer: Adagrad, LR = 0.001, Epochs: 10
Architecture: VGG-16



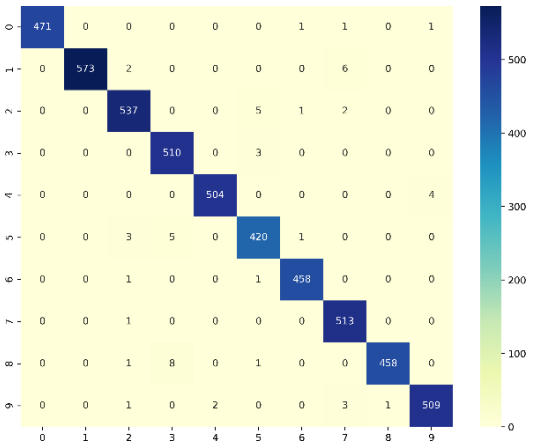
5. Optimizer: SGD, LR = 0.01, Epochs: 5
Architecture: VGG-16



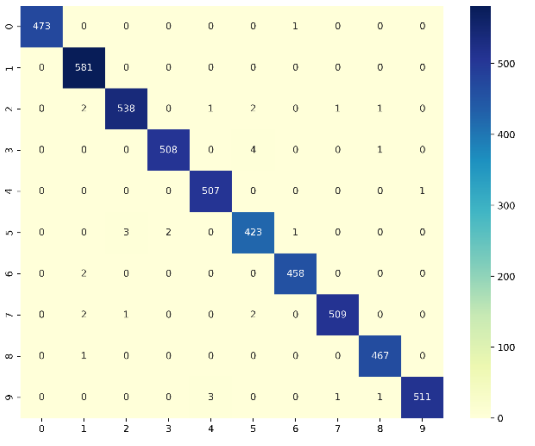
6. Optimizer: SGD, LR = 0.01, Epochs: 10
Architecture: VGG-16



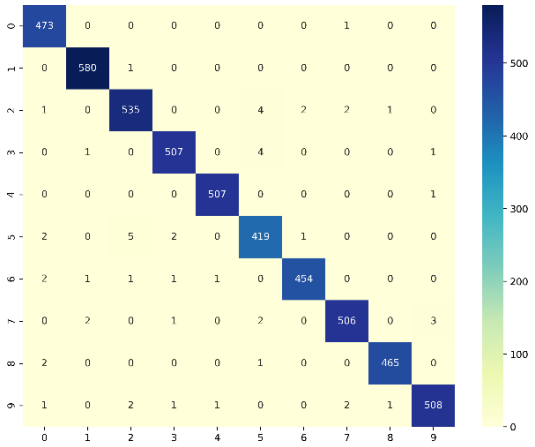
7. Optimizer: SGD, LR = 0.001, Epochs: 5
Architecture: VGG-16



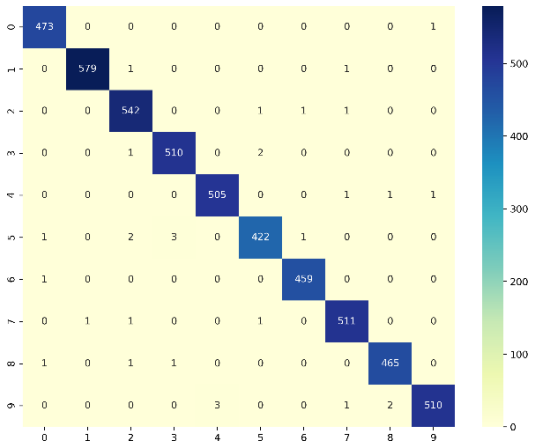
8. Optimizer: SGD, LR = 0.001, Epochs: 10
Architecture: VGG-16



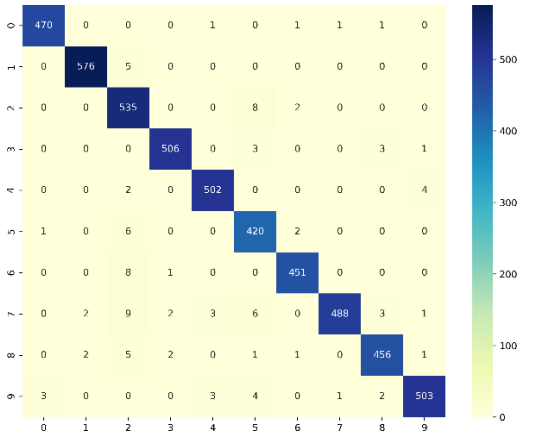
9. Optimizer: Adagrad, LR = 0.01, Epochs: 5
Architecture: ResNet-34



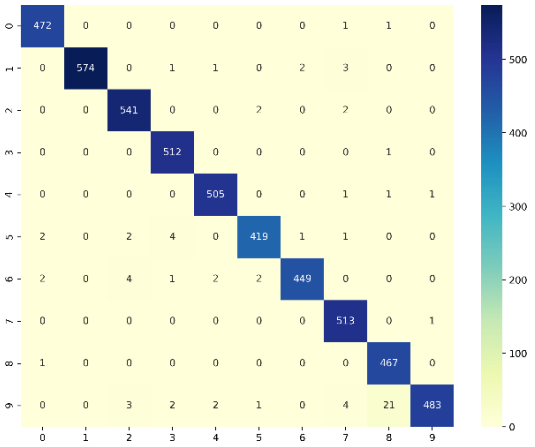
10. Optimizer: Adagrad, LR = 0.01, Epochs: 10
Architecture: ResNet-34



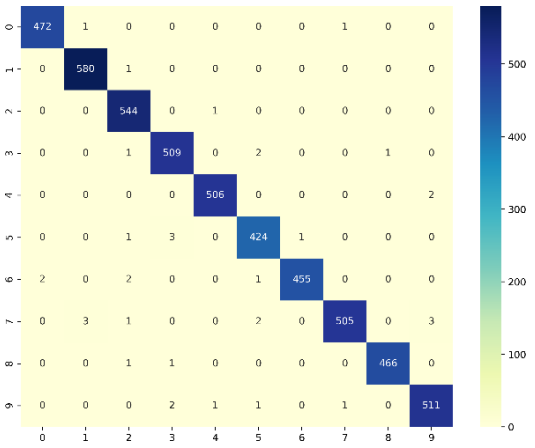
11. Optimizer: Adam, LR = 0.01, Epochs: 5
Architecture: ResNet-34



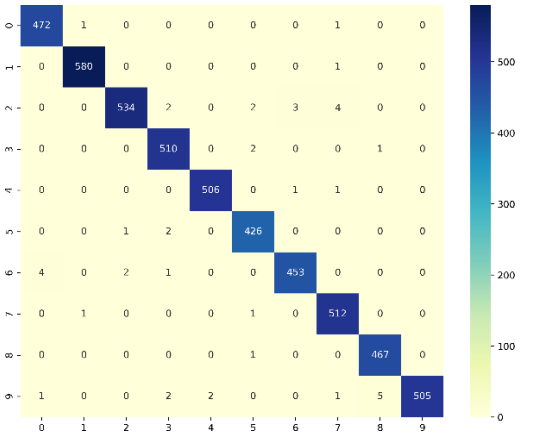
12. Optimizer: Adam, LR = 0.01, Epochs: 10
Architecture: ResNet-34



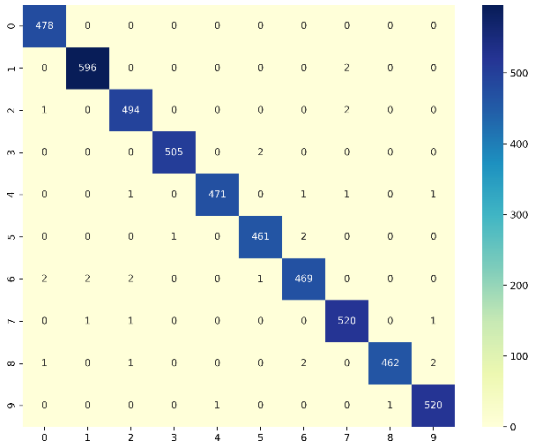
13. Optimizer: SGD, LR = 0.01, Epochs: 5
Architecture: ResNet-34



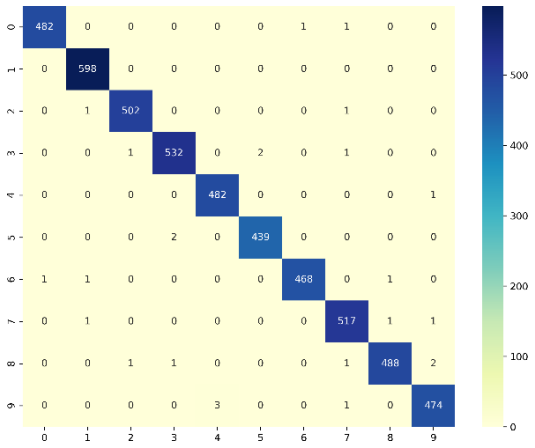
14. Optimizer: SGD, LR = 0.01, Epochs: 10
Architecture: ResNet-34



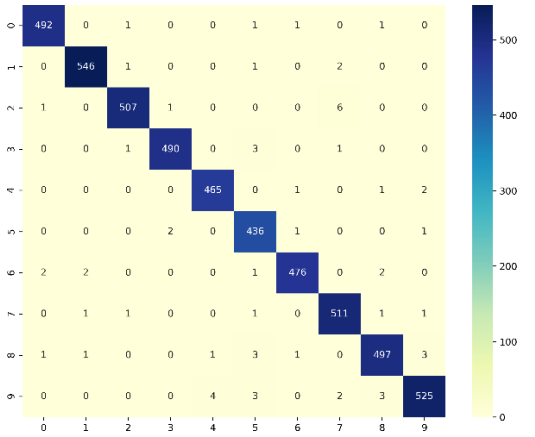
15. Optimizer: Adagrad, LR = 0.01, Epochs: 5
Architecture: GoogleNet



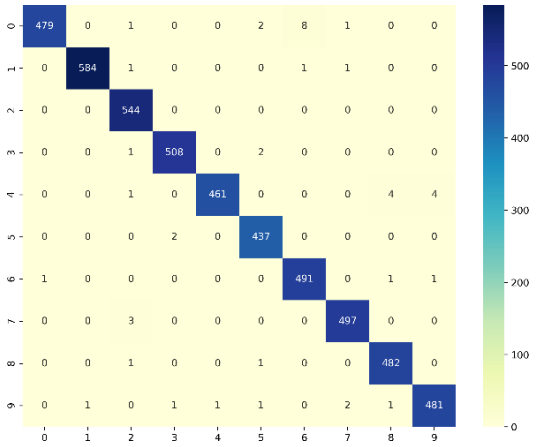
16. Optimizer: Adagrad, LR = 0.01, Epochs: 10
Architecture: GoogleNet



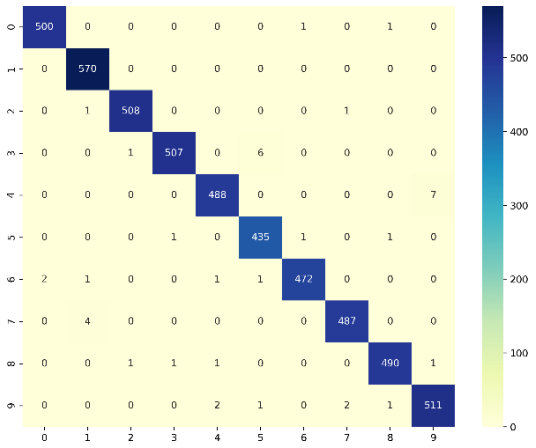
17. Optimizer: Adam, LR = 0.01, Epochs: 5
Architecture: GoogleNet



18. Optimizer: Adam, LR = 0.01, Epochs: 10
Architecture: GoogleNet



19. Optimizer: SGD, LR = 0.01, Epochs: 5
Architecture: GoogleNet



20. Optimizer: SGD, LR = 0.01, Epochs: 10
Architecture: GoogleNet