A Comparative Study on CNN Architectures

Hari Chandana Mone
Department of Computer Science
University of Texas at Dallas
Richardson, U.S.A.
hxm210000@utdallas.edu

Meghana Sai Bijivemula
Department of Computer Science
University of Texas at Dallas
Richardson, U.S.A.
mxb210011@utdallas.edu

***Abstract*— In the modern world, the role of Image classification is widespread, from healthcare to e-commerce, transportation to marketing. Convolutional Neural Networks (CNN) are the best way to classify images. In this paper, we are going to perform a comparative study on different CNN architectures – classic, modern and a modified LeNets – on MNIST, Fashion MNIST and CIFAR100 datasets. Our aim is to analyze these different architectures by using two optimizers – Adam and Stochastic Gradient Descent and training the models for varying number of epochs to find the best performing model for each dataset based on their training and validation losses and accuracies.**

Keywords*— Artificial Neural Networks, Convolutional Neural Network, VGG, ResNet, LeNet, Principal Component Analysis, t-SNE, Convolutional Layer (CONV), Pooling Layer (POOL)

I. Introduction

Nowadays, the need for Image Classification can be found in almost every industry. As we are generating unprecedented amounts of data year after year in the last 5 years especially images, we have a huge demand for image classification. We have used these image classification technologies to detect COVID-19 from images of X-rays and CT-scans of lungs during the most recent calamity the world has experienced, the COVID-19 Pandemic. Another ubiquitous application is in our own hands - our phones are capable of sorting or organizing pictures based on the content, whether it is faces, documents, or things. Images can be classified using multiple Machine Learning techniques like K-nearest Neighbour, Naïve Bayes and Random Forest; however, convolutional neural networks are the widely utilized to classify images. The fact that CNN can quickly and easily recognize perceptible patterns in images is what makes it the most sought-after method.

Convolutional Neural Nets are comprised of Input Layer, Output Layer, Convolutional Layer, Pooling Layer, and Fully Connected Layer. Convolutional Layer includes a series of filters that perform convolution operations on the layer's input and extract specific features from them based on the filter. After performing the convolution then the result is activated through a nonlinear activation function and among all available activation functions, the most preferred and efficient out of all is Rectified Linear Unit (commonly referred to as ReLu). Now moving on to the Pooling Layer, it always follows the convolutional layer in order to downsize the result yielded by the conv layer as it decreases the size by performing commonly used pooling techniques.

The widely used techniques in the sub-sampling layer are max pooling operation and average pooling, these are applied to reduce the size of the image without losing any valuable information. This reduction in the size is advantageous to the network as the number of params that needed to be trained will also plummet significantly, which in turn improves the performance. It is normal to find a convolution layer followed by one or two pooling layers. This succession is rehashed on numerous occasions to perform extraction of the features and toward the end, the result of this is fed to fully connected layers (FC) after converting them from three dimensional tensors to a one-dimensional vector. Activation functions like squashing and softmax are used by these FC layers to categorise the features that were extracted by the layers before them. The logistic or squashing activation function is utilized by these layers when there are only two labels or categories, and the softmax activation function is utilized when there are more than two. The probability of each category is given by the output layer as it will be comprised of the same number of neurons as the labels for that application.

II. Related Work

Several architectures of ConvNets have been created over the years since its inception under the name Neocognitron by Kunihiko Fukushima. In this paper, we will be working with LeNet, AlexNet, Visual Geometry Group (VGG) and ResNet. Along with these standard architectures, we propose a new architecture in the report – a modified LeNet. We began this study when we were attempting to benchmark a CNN model for a standard dataset by utilizing different optimizers, loss functions and activation functions. We were curious about how different datasets affected the performance of various standard architectures after experimenting with those. Therefore, we examined the performance of the three aforementioned datasets and architectures using various optimizers and epochs.

A number of similar studies were conducted to find the best architectures for a dataset. Now, we will discuss a study that was done – "*COVID-Nets: deep CNN architectures for detecting COVID-19 using chest CT scans* [1]" where the paper explores ResNet50 and DenseNet121 for a SARS-CoV-2 detection on lung CT-scan dataset. The authors use sensititvity, specificity, AUC score, Accuracy and Presicion scores to benchmark the models.

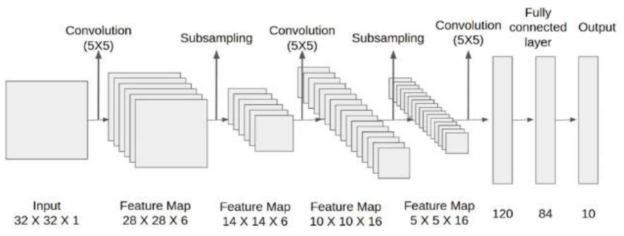
A conventional "*CNN and a AlexNet were compared in a study to find which is better for the detection of disease in Mango and Potato leaf* [2]". In this paper, the authors explore the networks on a mago and potato leaves datset containing 4004 images and compared these models based on the accuracy and efficiency of the models.

III. Initial Analysis

Now Let us understand each of these architectures in depth starting with *LeNet. LeNet is proposed by Y. LeCun, et al. in their paper – Gradient-based learning applied to document recognition* [2]. This architecture is comprised of seven layers: three CONV, two POOL and two FC layers. Consider CONV-1, CONV-2 and CONV-3 are the three CONV layers and POOL-1, POOL-2 are the two Pooling layers, and FC1, FC2 are the fully connected layers.

Figure 1

LeNet-5 architecture



Note: Adapted from Analytics Vidhya by Shipra Saxena — Published on March 18, 2021

The Input Layer can take in an image with one channel, which is a black and white image, and passes it on to CONV-1 layer with six filters of size 5×5. Here the first CONV layer generates a feature map $28 \times 28 \times 6$ and this is fed to the first POOL layer. POOL-1 generates a feature map of $14 \times 14 \times 6$, which will ne fed to the CONV-2 layer with 16 filters each if size 5×5. CONV-2 produces $10 \times 10 \times 16$ and it becomes $5 \times 5 \times 16$ after being processed in POOL-2. Now CONV-3 with 120 filters of size 5 takes in the previous output and produces a result of size 120 and it will be fed to the FC layers and the output layer will have softmax activation in order to classify images into multiple categories. In this architecture, we have 60K parameters in the 5 layers which extracts features.

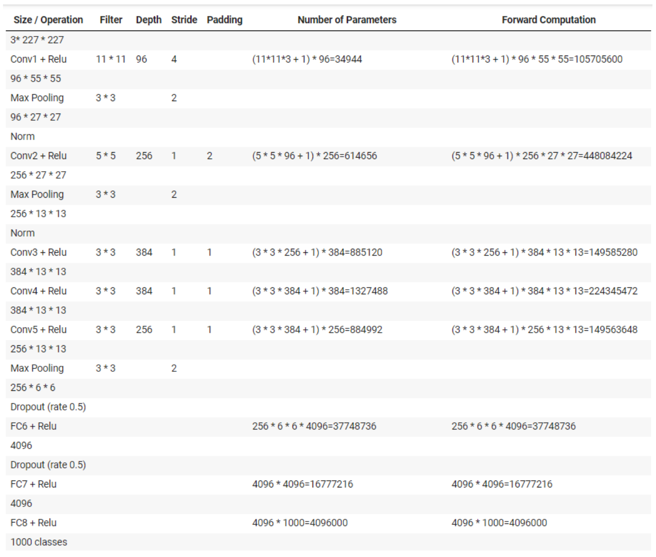
Modified Lenet Model - The modified Lenet has the same layers as the Lenet5 with extra convolutional layer and Maxpooling layer. Added convolutional layer. The parameters to the extra convolutional layer are as follows. The filter size is 128, and the kernel size is 5 and the padding is same. There is maxpooling layer that follows the CNN layer with parameters being pool size and stride is 2 with padding remaining same.

Now we will move on to AlexNet, it was created by Hinton and his student Alex Krizhevsky. AlexNet has 5 CONV layers, 3 POOL layers, 3Dropout layers and 3 FC layers. It is very similar to LeNet with some modifications like addition of a dropout layer which helps on avoiding overfitting by performing regularization. Dropout is incorporated into the network by altering its inner skeleton. It revises the params in a certain layer according to the training process and removing some with a certain possibility while keeping all the perceptrons intact in the input and output layers. The activation function used is ReLu as they converge significantly quicker compared to squashing function and hyperbolic tangent function.

The architecture has 8 layers performing feature extraction and it can also take colour images that is input with three channels and similar to the previous architecture softmax and sigmoid are used in the last layer to label the images. This Network architecture has around sixty million parameters.

Figure 2

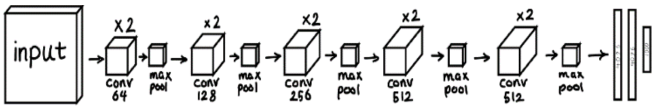
AlexNet Network Structure



Size / Operation	Filter	Depth	Stride	Padding	Number of Parameters	Forward Computation
3* 227 * 227						
Conv1 + Relu	11 * 11	96	4		(11*11*3 + 1) * 96=34944	(11*11*3 + 1) * 96 * 55 * 55=105705600
96 * 55 * 55						
Max Pooling	3 * 3		2			
96 * 27 * 27						
Norm						
Conv2 + Relu	5 * 5	256	1	2	(5 * 5 * 96 + 1) * 256=614656	(5 * 5 * 96 + 1) * 256 * 27 * 27=448084224
256 * 27 * 27						
Max Pooling	3 * 3		2			
256 * 13 * 13						
Norm						
Conv3 + Relu	3 * 3	384	1	1	(3 * 3 * 256 + 1) * 384=885120	(3 * 3 * 256 + 1) * 384 * 13 * 13=149585280
384 * 13 * 13						
Conv4 + Relu	3 * 3	384	1	1	(3 * 3 * 384 + 1) * 384=1327488	(3 * 3 * 384 + 1) * 384 * 13 * 13=224345472
384 * 13 * 13						
Conv5 + Relu	3 * 3	256	1	1	(3 * 3 * 384 + 1) * 256=884992	(3 * 3 * 384 + 1) * 256 * 13 * 13=149563648
256 * 13 * 13						
Max Pooling	3 * 3		2			
256 * 6 * 6						
Dropout (rate 0.5)						
FC6 + Relu					256 * 6 * 6 * 4096=37748736	256 * 6 * 6 * 4096=37748736
4096						
Dropout (rate 0.5)						
FC7 + Relu					4096 * 4096=16777216	4096 * 4096=16777216
4096						
FC8 + Relu					4096 * 1000=4096000	4096 * 1000=4096000
1000 classes						

Figure 3

VGG-13 Network Structure



Note: Adapted from Medium article by Amir Hossein — Published on Feb 23, 2018

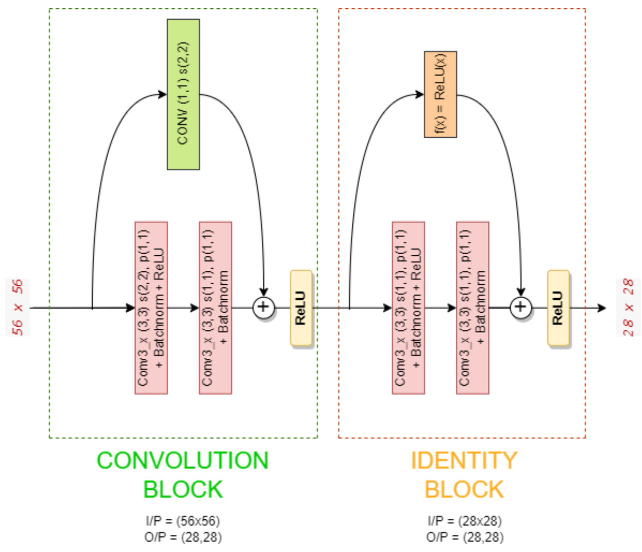
Now we will explore VGG architecture, it was created by *A. Zisserman and others in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" which belongs to the Department of Science and Engineering of Oxford University* [2]. They have also created multiple VGG architectures ranging from VGG11 VGG19. VGG11 contains eight CONV layers and 3 FC layers and VGG19 is comprised of sixteen CONV layers and 3 FC layers. Furthermore, each Conv layer is not followed by POOL layer in VGG, but five POOL layers are allocated throughout the network. Here every CONV represents a 3×3 kernel and every POOL layer is of size 2×2 and as we go deeper into the network the number of feature maps generated increases as the network becomes wider. VGG uses small filters in the convolutional layers, and it helps in decreasing the params also according to the paper cited above, the writer implies that it is the same as nonlinear mapping. VGG can process pictures of size $225 \times 225 \times 3$ which means it can accept rgb images.

As VGG is deep neural network, during the gradient calculation while backpropagation, at some point it reaches an unstable state due to endless product operations due to its depth, so the value will be significantly high or insignificantly low. Multiple techniques have been developed to resolve this issue, some of them are batch normalization, using Rectified Linear Units. Additionally, with deeper nets, we also encounter a deterioration in the performance of the nets which in turn results in the plummeting of accuracy. These problems that we run into with deep neural nets are addressed by Residual Networks.

Now let us move on to Residual Network (ResNet) architecture, it was developed by "*He Kaiming, and others of Microsoft Research Asia in 2015". ResNet is different from the architectures that we have discussed before because of the skip connections. These connections incorporate residual learning which overcomes the issues VGG runs in to like vanishing gradient and accuracy deterioration due to its depth. Here residual learning refers to the two main ideas that are incorporated, one is the skip or identity where the connection is in its own block and the other one is residual which means between different blocks.

Figure 4

Building blocks of ResNets


56 x 56
CONV (1,1) s(2,2)
Conv3_x (3,3) s(2,2), p(1,1) + Batchnorm + ReLU
Conv3_x (3,3) s(1,1), p(1,1) + Batchnorm
ReLU
f(x) = ReLU(x)
Conv3_x (3,3) s(1,1), p(1,1) + Batchnorm + ReLU
Conv3_x (3,3) s(1,1), p(1,1) + Batchnorm
ReLU
28 x 28
CONVOLUTION BLOCK
I/P = (56x56)
O/P = (28,28)
IDENTITY BLOCK
I/P = (28x28)
O/P = (28,28)

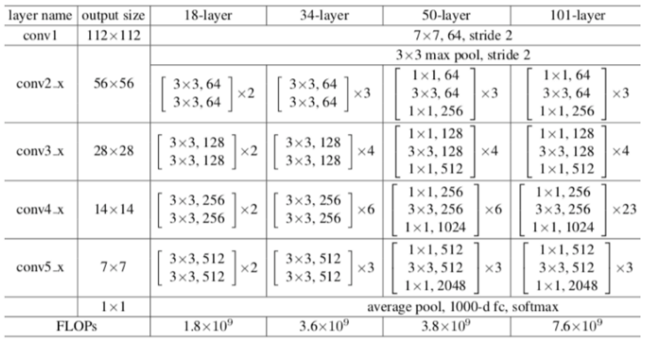
Note: Adapted from Analytics Vidhya article by Rohit Modi — Published on Dec 1, 2021

The above convolution block accepts a 56 × 56 × 64 feature map, and the initial CONV, which has a 3 × 3 filter, a stride of two, and padding of one, generates a 28 × 28 × 128 feature map and then applies the Rectified Linear Unit activation function. Similarly, the next CONV receives a 28 ×28 ×128 feature map and as the shape of the feature map does not match, to add the residue, we utilize a CONV layer to alter the original input from 56 ×56 ×64 to 28 ×28 ×128 that is why we use a CONV layer with kernel size-1 and stride-2. Once the residue is added then its activated using Rectified Linear Unit function.

In Identity block, no data modification is required to the output from convolutional block as the input & output for an identity block are identical. This infers that no transformation is needed in order to apply Skip/Residual connection. We have to just apply ReLU on the output from convolutional block applied to the fourth convolution layer in the block. Consider using identity block when input and output are identical otherwise use Convolution Block where output adjustment is needed to apply a Residual connection.

Figure 5

ResNet architecture for 18, 34, 56 and 101 layers



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

Note: Adapted from Deep Residual Learning for Image Recognition by He, K, and others — Published in 2015

Using the convolutional blocks, Identity block and the above defined architecture for ResNets, we bult ResNet-18 and ResNet34 models.

IV. Dataset Analysis

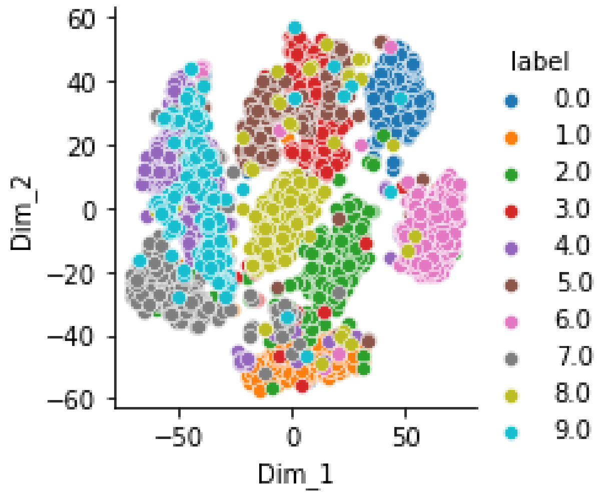
For the comparative study, we used three standard datasets-MNIST, Fashion MNIST and CIFAR100 and train the models with different CNN architectures.

Dataset	Size	Classes	Data Split	Notes
MNIST	28×28×1	10 classes with 7000 images each	Training-60,000 Testing-10,000	images of hand-written single digits between 0 and 9
Fashion MNIST	28×28×1	10 categories with 7000 images each	Training-60,000 Testing-10,000	images of fashion products from 10 categories
CIFAR 100	32x32x3	100 classes with 600 images each grouped into 20 super classes	Training-50000 Testing-10000	Images are either labeled fine or coarse.

We have used two techniques - T-distributed neighbor embedding (t-SNE) and Principal component analysis (PCA) to visualize these high dimensional datasets. The techniques mentioned above are used to visualize multi-dimensional data by decreasing their dimensions. Applying t- SNE on MNIST gives the following plot showing the 10 classes representing ten digits and the clustering of these classes is shown in the plot.

Figure 6

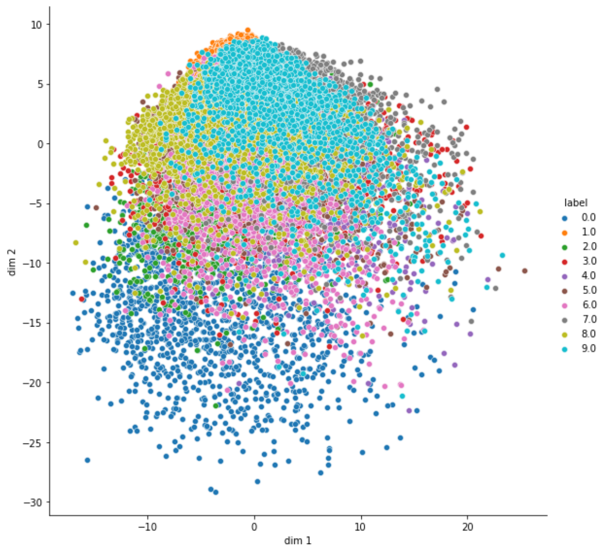
Visualization of MNIST using t-SNE



We also visualized the dataset using PCA, here the clustering is not shown that clearly due to its linearity.

Figure 7

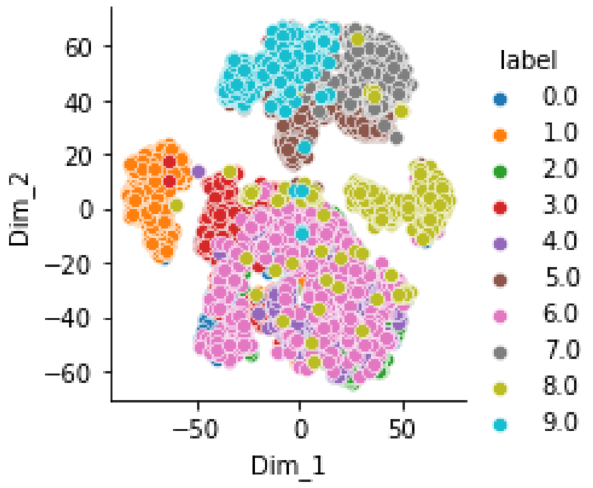
Visualization of MNIST using PCA



The following is the plot showing the visualization of Fashion MNIST using t-SNE for all the ten labels with different colours. There is some overlap in the clusters the clothing products meant for upper body have commonality and similarly for lower body clothing.

Figure 8

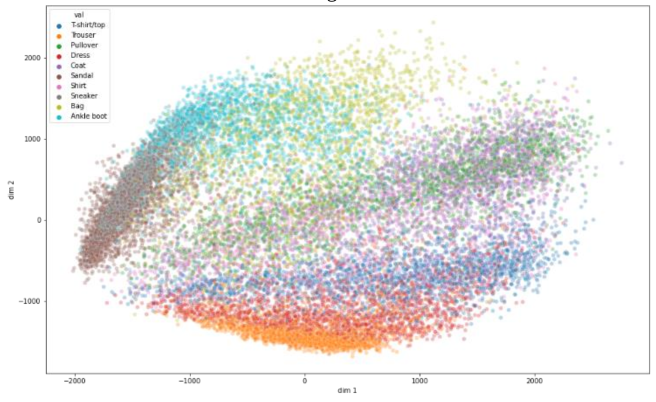
Visualization of Fashion MNIST using t-SNE



Using PCA, we cannot obtain the clearly defined clusters as we did with t-SNE as it is not linear.

Figure 9

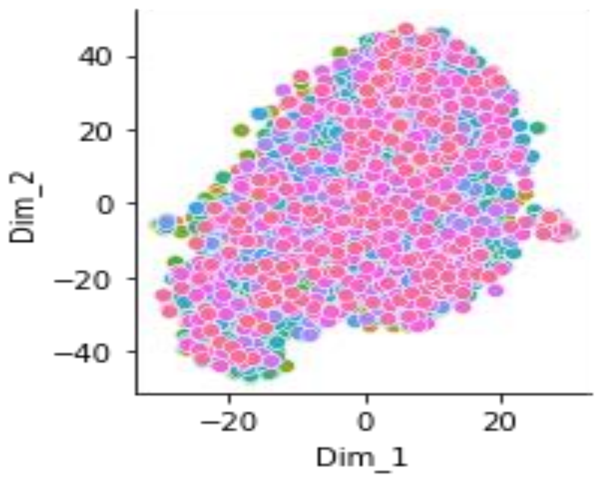
Visualization of Fashion MNIST using PCA



CIFAR-100 dataset visualization with t-SNE shows the overlapped clusters and there is not clear boundary as the data contains classes and subclasses which are related and because of that there is no clear distinction between labels when plotted.

Figure 10

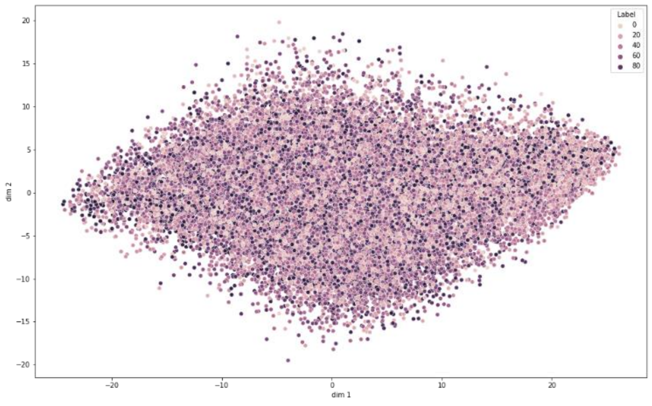
Visualization of CIFAR100 using t-SNE



Visualization of CIFAR-100 is done using Principal Component Analysis and below plot show all the 100 classes.

Figure 11

Visualization of CIFAR100 using PCA



V. Implementation and Analysis

We have implemented CNN architectures with on MNIST, Fashion MNIST and CIFAR100 datasets for optimizers – Adam and Stochastic Gradient Descent, and also number of epochs. For Adam Optimizer, we used the learning rate of 0.01 and for Stochastic Gradient Descent, we used the learning rate of 0.1 and momentum of 0.9. With these settings, we

A. *LeNet*

From the below plot, the training accuracy has reached maximum when trained with both the optimizers, but the validation accuracy has decreased for Adam with epochs 10. The accuracy is highest for the model when trained with Adam optimizer with epoch's 20.

Figure 12

Accuracy plots for MNIST using LeNet

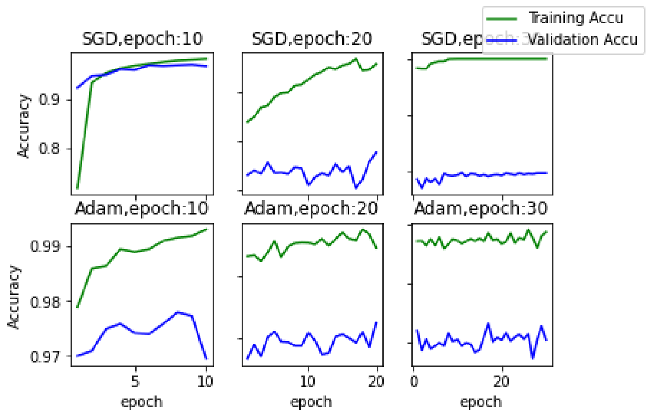
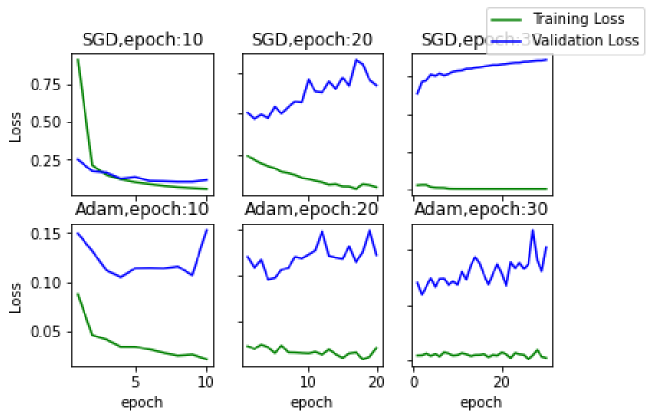


Figure 13

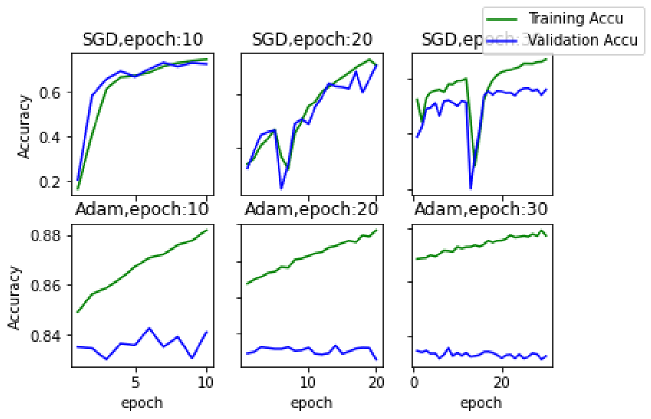
Loss plots for MNIST using LeNet



The figure above clearly states that the validation loss and training loss followed a trend for SGD optimizer with epochs 10. The graph actually depicts the best fit for the above mentioned hyper parameters although increasing epochs did not help in reducing the validation loss of the graph. With Adam optimizer the model did try to learn the training dataset for the less epochs and the graph is stable for more epochs. The validation loss is fluctuating as seen in the graph above.

Figure 14

Accuracy plots for Fashion MNIST using LeNet

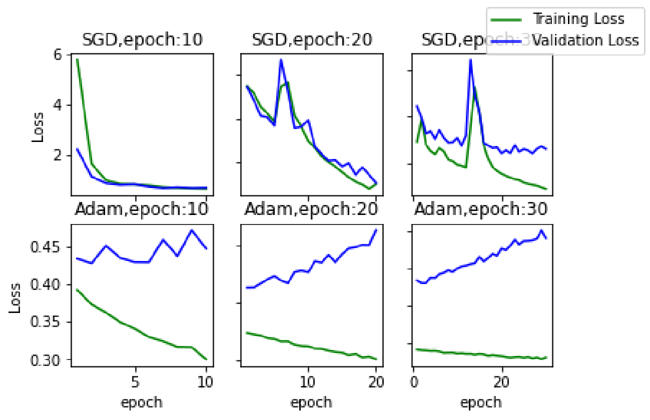


From the above plot, the accuracy plots for SGD optimizer are the best fit models graph for all the epochs but when it comes to the adam there is a huge gap between the training accuracy and validation accuracy.

From the below graph, the loss values are less for the Adam optimizer for all the epochs and as well for the above parameters that model learned the dataset well when compared to SGD optimizer

Figure 15

Loss plots for Fashion MNIST using LeNet



From the below graph it can be inferred that the optimizers did not try to learn the training data well and both optimizers did not do a good job.

Figure 16

Accuracy plots for CIFAR100 using LeNet

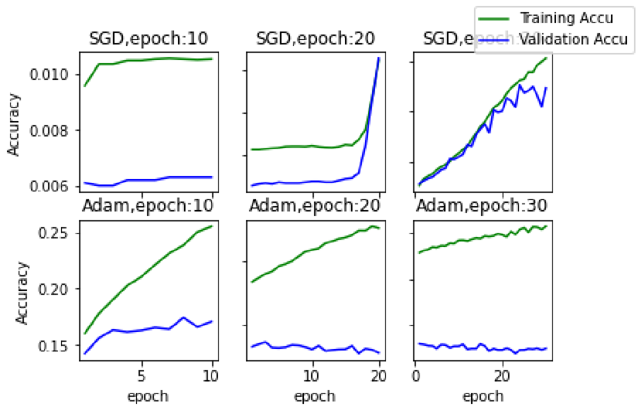
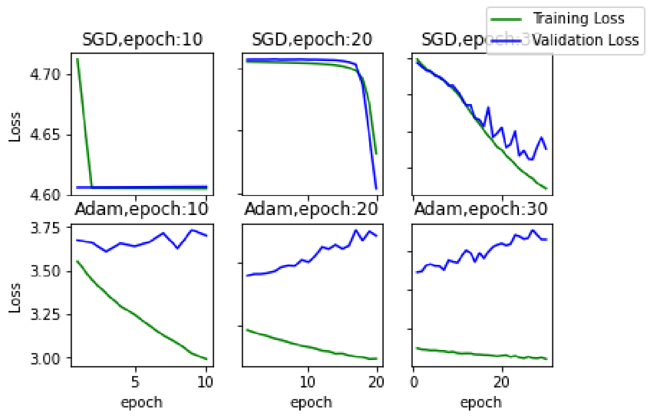


Figure 17

Loss plots for CIFAR100 using LeNet



The training loss is less for the graph with the optimizer Adam and epoch 30. The loss is high for the model with optimizer SGD and epoch 20.

B. AlexNet

The below plot depicts that the training accuracy is high for SGD optimizer when compared to the Adam optimizer and the validation accuracy is high for the models with the Adam optimizer.

Figure 18

Accuracy plots for MNIST using AlexNet

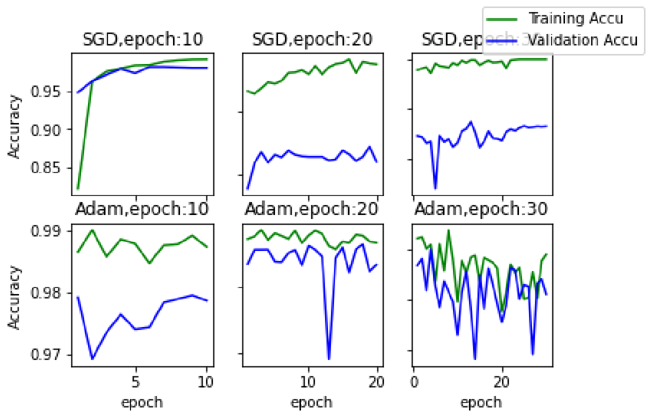
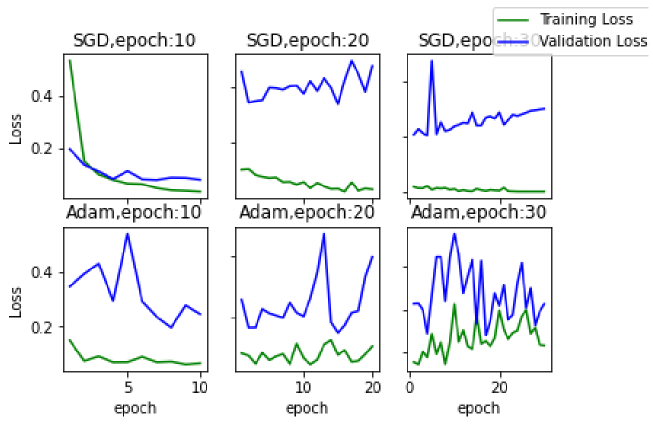


Figure 19

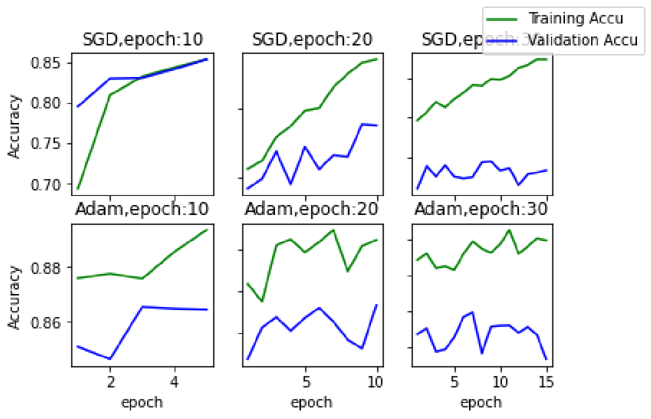
Loss plots for MNIST using AlexNet



The training and validation loss are less for the SGD optimizer when compared to other hyperparameters. The model learns the training data well for the SGD optimizer when compared to the Adam.

Figure 20

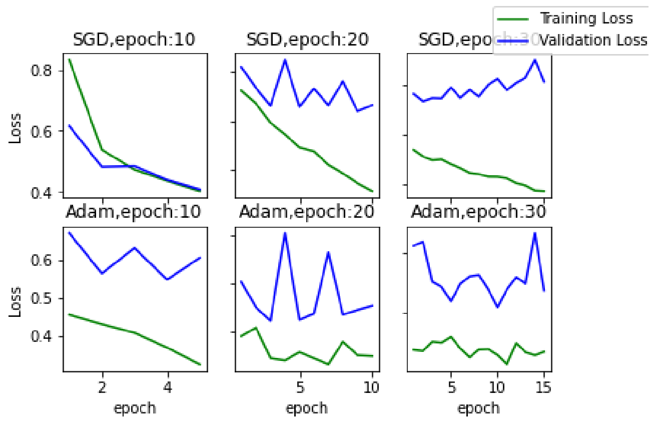
Accuracy plots for Fashion MNIST using AlexNet



The accuracy for the training is always increasing using SGD optimizer whereas there is no clear trend for the Adam optimizer. For SGD, the model tries to learn the training data well.

Figure 21

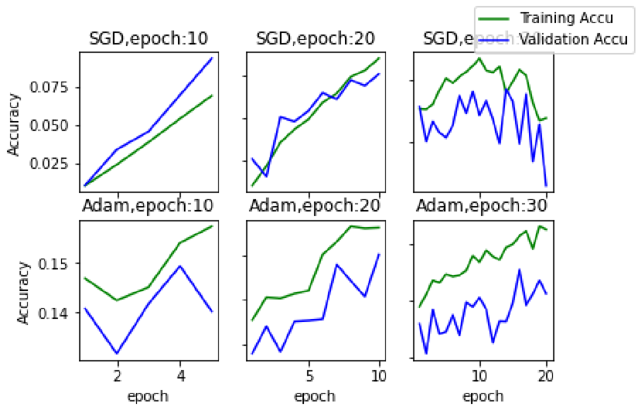
Loss plots for Fashion MNIST using AlexNet



The loss graphs dictate that the Adam hyperparameter performs well as the training loss has less values when compared to the SGD optimizer.

Figure 22

Accuracy plots for CIFAR-100 using AlexNet

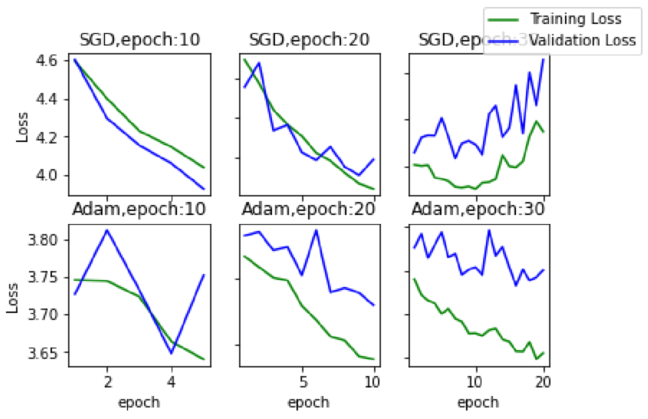


From the above plot the validation and training accuracy for the SGD and Adam start increasing and then decrease. The hyperparameters tuning did not have lot of impact as the network is shallow and the dataset is hard for the model to learn.

The training loss tend to decrease for both the optimizers initially whereas the validation is reaching maximum and hits minimum for some hyperparameters. Overall, the loss values are less for the Adam optimizer.

Figure 23

Loss plots for CIFAR-100 using AlexNet

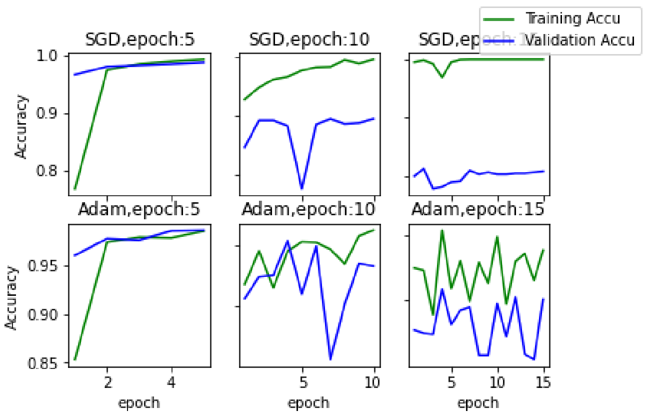


C. VGG

For the VGG, we implemented VGG-13 architecture with Adam and SGD optimizers and trained these models for 5, 10 and 15 epochs.

Figure 24

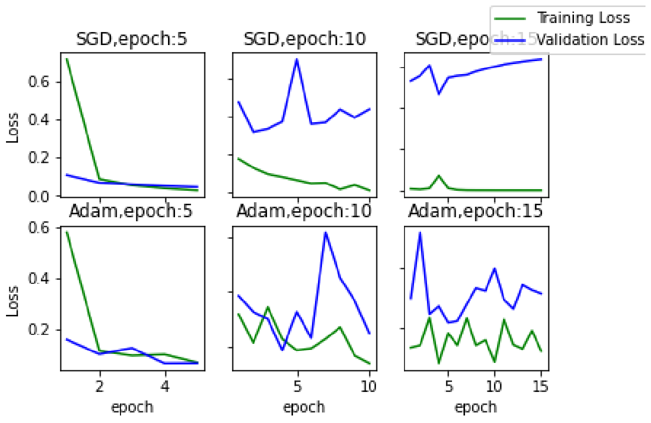
Accuracy plots for MNIST using VGG13



Here for the MNIST dataset, with 5 epochs the model was good fit because it did stabilize for both optimizers and when 10 epochs were run then for around 6 epochs it gives better result and then the performance is deteriorated.

Figure 25

Loss plots for MNIST using VGG13

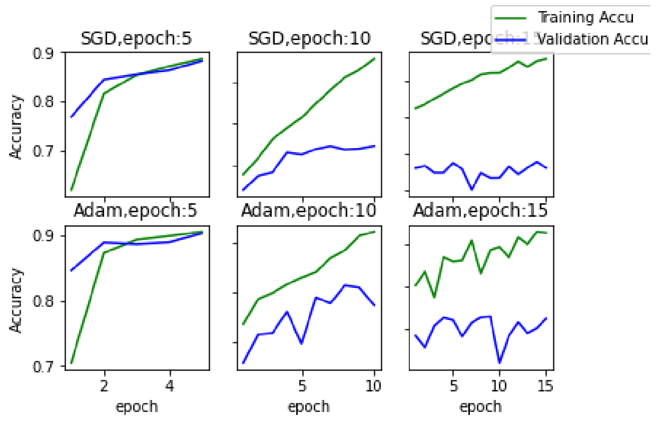


From the loss plots, we can infer that the SGD and Adam gives better performance after running for 5 epochs.

For Fashion MNIST dataset, Adam and SGD performs better for 5 epochs and the performance degrades beyond seven epochs.

Figure 26

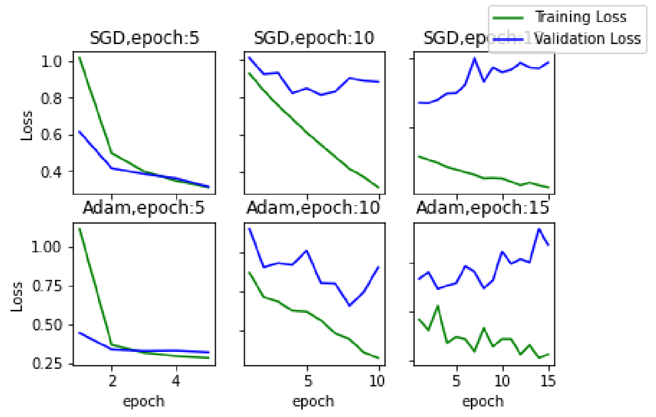
Accuracy plots for Fashion MNIST using VGG13



Adam optimizer with 5 epochs gives better performance for the Fashion MNIST dataset.

Figure 27

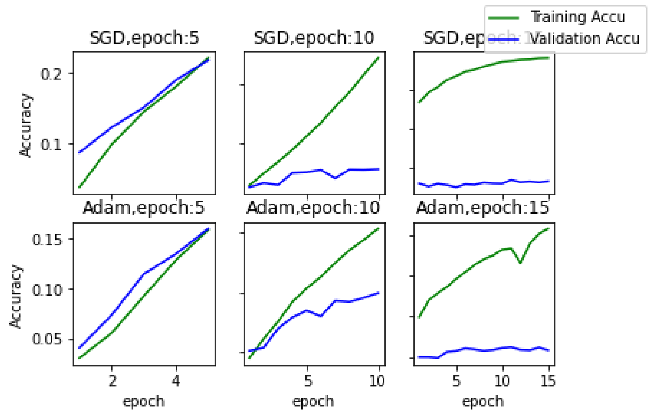
Loss plots for Fashion MNIST using VGG13



For CIFAR 100 dataset, we reach a maximum of around 20% accuracy scores for both train and validation accuracy.

Figure 28

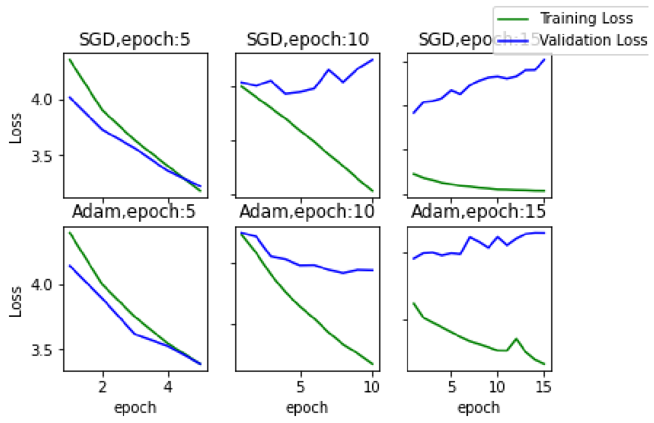
Accuracy plots for CIFAR100 using VGG13



Even from the loss plots, we can conclude that for CIFAR 100, both SGD and Adam performs similarly and both under perform as the dataset is too complex for the architecture.

Figure 29

Loss plots for CIFAR100 using VGG13

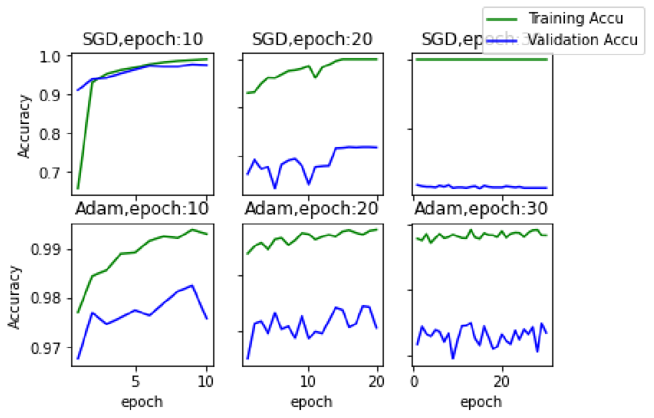


D. *Modified LeNet*

Below figure infers that SGD and Adam training accuracy increases and is closer to 99%

Figure 30

Accuracy plots for MNIST using Mod. LeNet



Below figure infers that as the epochs increase the training accuracy is decreasing and validation loss is increasing respectively

Figure 31

Loss plots for MNIST using Mod. LeNet

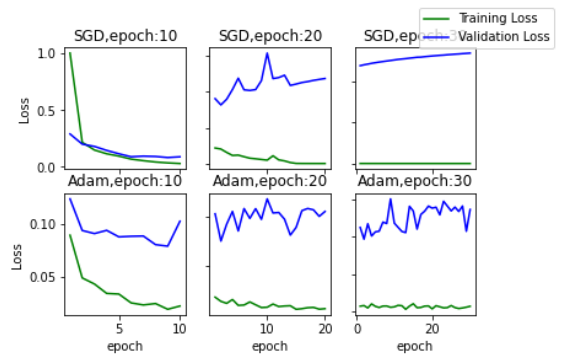
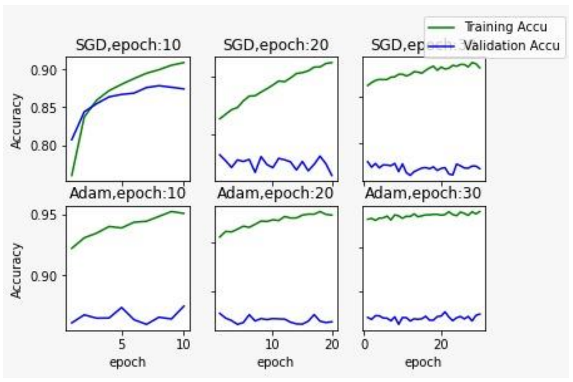


Figure 32

Accuracy plots for Fashion MNIST using Mod. LeNet

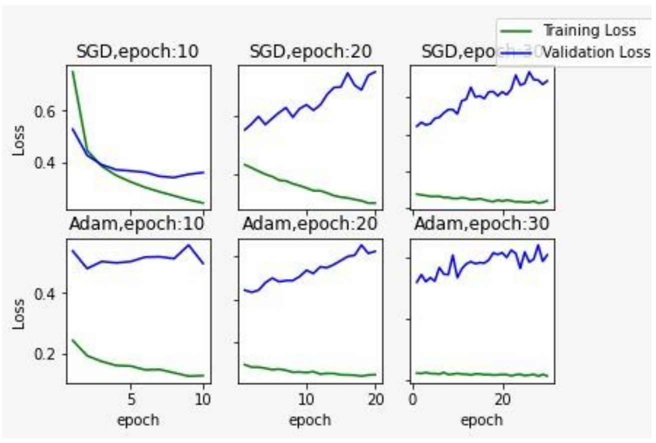


Above figure infers that for both the training accuracy is increasing and is high for epoch 30 and validation is decreasing for in all cases except for epoch 10 SGD.

Below figure infers that for both the training loss is decreasing and validation loss is increasing. Training loss is almost constant for higher epochs

Figure 33

Loss plots for Fashion MNIST using Mod. LeNet

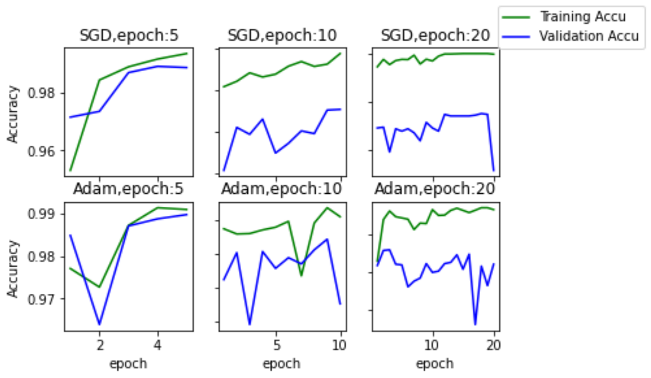


E. ResNet

Among all the ResNets with varying number of layers, we opted for ResNet18 and ResNet34. For ResNet-18, we have used Adam with learning rate of 0.01 and Stochastic Gradient Descent with the learning rate of 0.1 and momentum of 0.9, and also number of epochs -5, 10 and 20 to train the models using MNIST and the following are the plots for training and validation accuracies and training and validation losses.

Figure 34

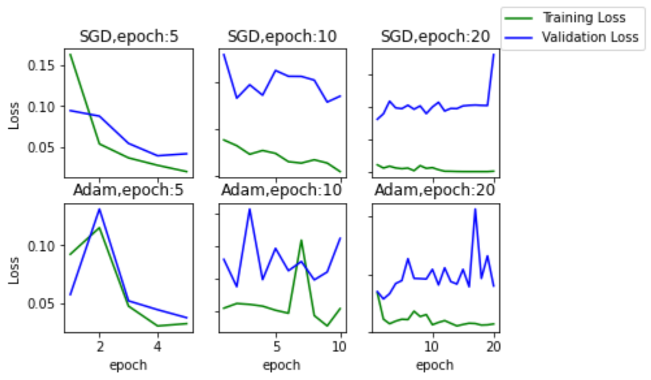
Accuracy plots for MNIST using ResNet-18



From the above figure, we can infer that with SGD and Adam - epoch 5's overall accuracy has reached an all time high for both test and validation with around 99%. However, it reduced for 10 epochs and validation plummeted at the end for 20 epochs.

Figure 35

Loss plots for MNIST using ResNet-18



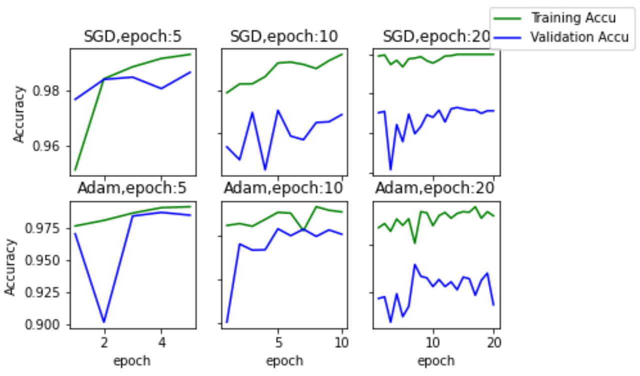
The figure above depicts that with SGD and Adam - epoch 5's overall loss deceased to an all-time low for both test and validation with less than 0.05. However, for 10 epochs and for 20 epochs, the validation loss never reached a low or was never even close to the training loss. From both the plots, we can conclude that epochs 5 with Adam and SGD with given hyperparameter tuning will result in a model with good fit.

For ResNet-34, we have used Adam with learning rate of 0.01 and Stochastic Gradient Descent with the learning rate of 0.1 and momentum of 0.9, and also number of epochs -5, 10 and 20 to train the models on MNIST and the following are the plots for training and validation accuracies and training and validation losses.

From the below figure, we can infer that with SGD and Adam - epoch 5's overall accuracy climbed to a peak for both test and validation with around 98%. However, the validation accuracy never got closer to training accuracy for all other combinations except for Adam with 10 epochs. Adam with 5, 10 and SGD with 5 are the models with better accuracy scores.

Figure 36

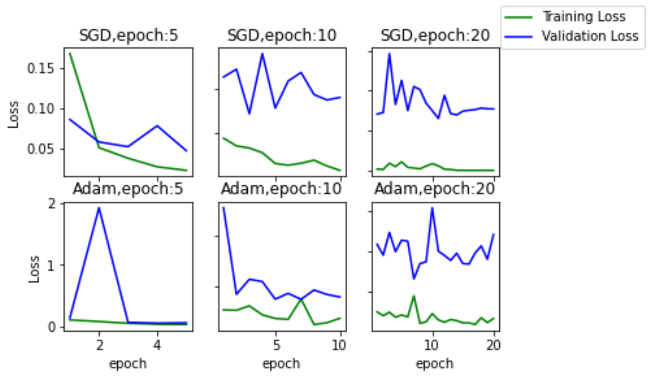
Accuracy plots for MNIST using ResNet-34



According to the graph below, with SGD and Adam, epoch 5's overall loss has dropped to an all-time low for both test and validation. However, for 10 and 20 epochs, the validation loss never reached a low or came close to matching the training loss.

Figure 37

Loss plots for MNIST using ResNet-34

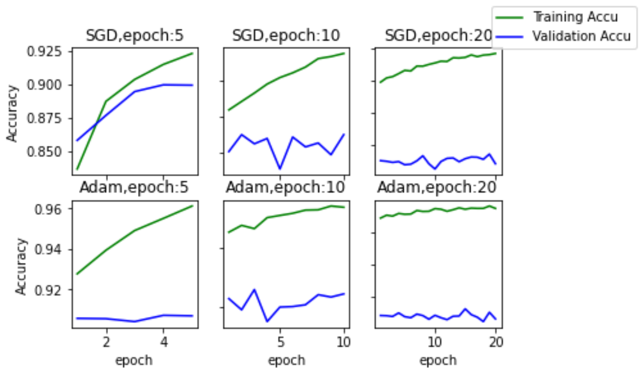


We can conclude from both plots that epochs 5 with Adam and SGD with the given hyperparameter tuning will result in a model with a good fit, but SGD with 5 epochs is a better fit for MNIST dataset.

Now for Fashion MNIST, we used Adam with a learning rate of 0.01 and Stochastic Gradient Descent with a learning rate of 0.1 and momentum of 0.9, as well as a number of epochs of -5, 10, and 20 to train the models using MNIST, and the plots for training and validation accuracies and training and validation losses are shown below. Now let us explore the settings with ResNet-18 architecture.

Figure 38

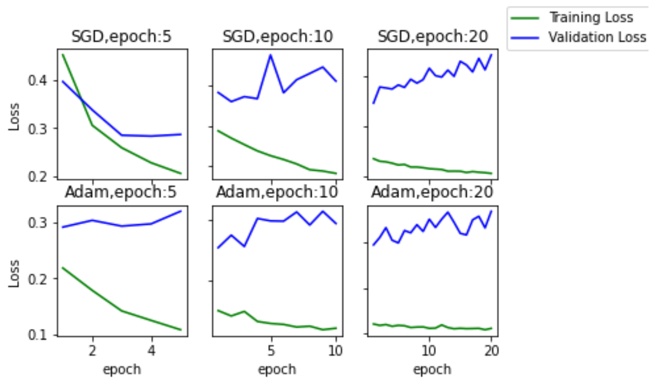
Accuracy plots for Fashion MNIST using ResNet-18



From both loss and accuracy plots, we can infer that with Adam by running 5 epochs overall accuracy and loss are at optimum. SGD with 5 epochs is the only other model which could reach closer to such performance, but its validation accuracy did not get as high as 91%. For the other combinations, the loss and accuracy of validation were very far from the corresponding losses and accuracies of training.

Figure 39

Loss plots for Fashion MNIST using ResNet-18

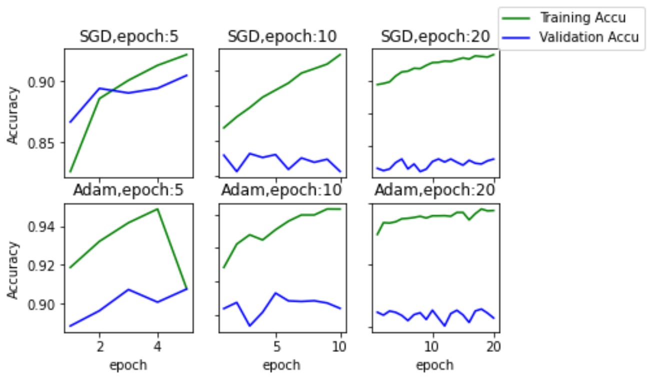


As the data set has only 10 classes and the model is complex, we do not need to run more than 5 to 10 epochs with the above models as proved above from the plots.

Now let us explore the previously mentioned settings with ResNet-34 using the accuracy and loss plots. According to the attached figure, SGD and Adam - epoch 5's overall accuracy has reached an all-time high of around 91% for both test and validation. However, it remained constant for 10 epochs and 20 epochs.

Figure 40

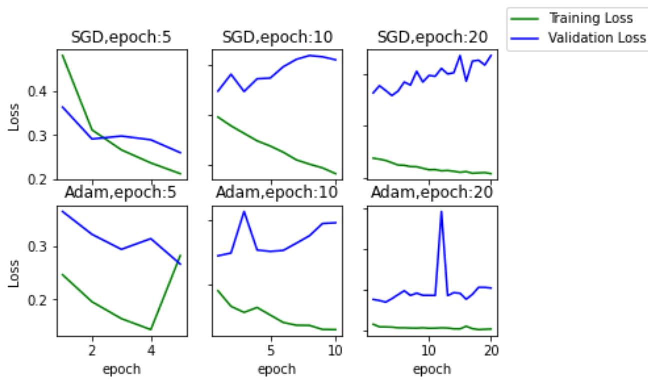
Accuracy plots for Fashion MNIST using ResNet-34



Now when we observe the corresponding loss plots for epoch 5 for both optimizers, it can be concluded that SGD is the best fit and Adam is next best fit. The other models are overfitted as the training loss in those is decreasing but validation loss is climbing up.

Figure 41

Loss plots for Fashion MNIST using ResNet-34

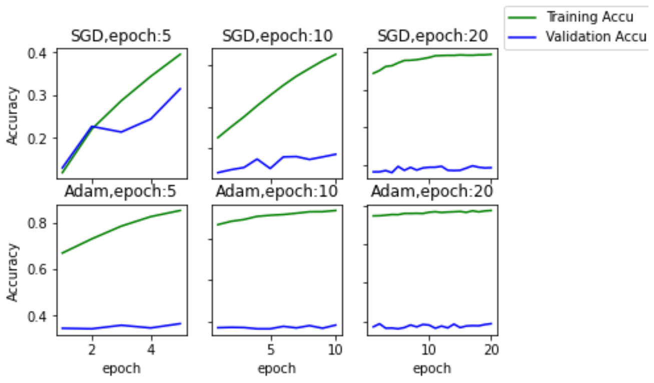


For Fashion MNIST data, as it is similar to the MNIST dataset however, the distribution is a little more complex, the models with SGD and Adam will be a good fit by running for 5. Running any more epochs will only result in overfitting of the model.

Lastly, we use CIFAR100 dataset and run the established model configurations on the dataset. Running ResNet-18 model on the dataset resulted in the below provided accuracy plot. From the plot, we can infer that validation could reach a high accuracy in Adam and SGD with 5 epochs and in Adam for 10 and 20 epochs.

Figure 42

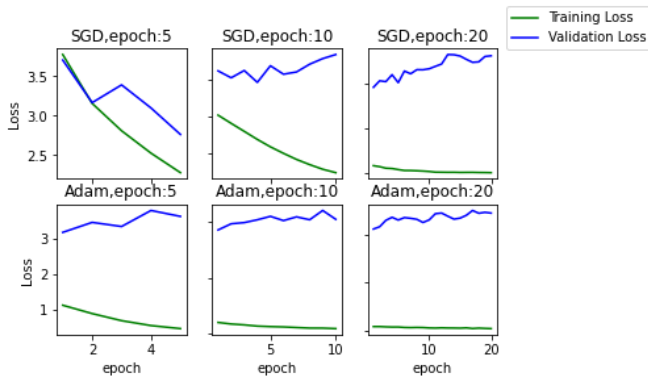
Accuracy plots for CIFAR100 using ResNet-18



Loss plots show that Adam achieves better results by running 5 and 10 epochs and all other configurations does not achieve any better results. ResNet 18 with Adam by running 10 epochs is a better fit for the dataset.

Figure 43

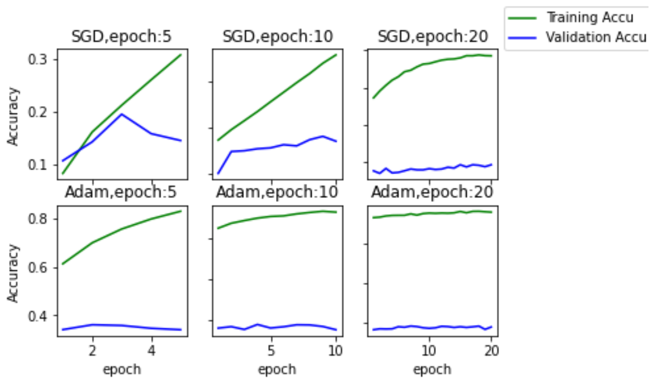
Loss plots for CIFAR100 using ResNet-18



ResNet-34 achieved a training accuracy of 80% and validation accuracy of 40% with Adam optimizers. However, SGD do not seem to get any closer to Adam for any of the epochs.

Figure 44

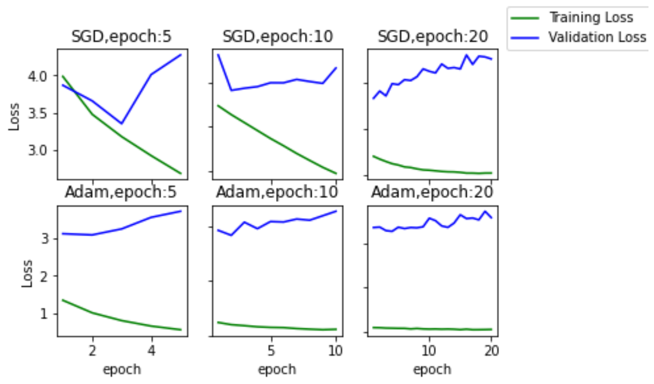
Accuracy plots for CIFAR100 using ResNet-34



Even the loss plots tell a similar story that Adam optimizer with 5 to 7 epochs is yielding the best scores out of all other configurations.

Figure 45

Loss plots for CIFAR100 using ResNet-34



For the CIFAR 100 dataset, ResNet-35 with Adam optimizer run for 5 epochs is yielding an efficient model out of all other models. However, this performance can be improved by tuning hyperparameters. As the data distribution is very complex with coarse and fine labels, these models which attained best results in MNIST and Fashion MNIST underperformed with CIFAR-100.

VI. Conclusion and Potential Work

For this comparative study, we used the following CNN architectures: LeNet, Modified LeNet, AlexNet, VGG-13, ResNet-18 and ResNet-34. We built models using these architectures and applying two different optimizers – Adam and SGD over numerous epochs. The following table shows the best model for image classification of each dataset:

Epoch size, OPT	Accuracy	Dataset	Loss	Model
Epoch 20 SGD	0.9941	MNIST	0.0281	ResNet 34

Epoch 10 SGD	0.9122	F MNIST	0.3319	ResNet-18
Epoch 5 Adam	0.3482	CIFAR100	3.6835	ResNet-34

In this paper, we have explored most of the CNN architectures in this study on three different datasets and analysed their performances on each dataset for different configurations. For MNIST and Fashion MNIST, many of the models performed well; however, for CIFAR100, ResNet 34 with Adam optimizer is the only model that could reach a training accuracy of 80% and testing accuracy of 40%. We plan to implement ResNet Models with more layers and ResNeXt architecture along with Inception and DenseNet.

REFERENCES

[1] Alshazly, H., Linse, C., Abdalla, M., Barth, E., & Martinetz, T. (2020). COVID-Nets: deep CNN architectures for detecting COVID-19 using chest CT scans. *PeerJ Computer Science*, *7*. https://doi.org/10.7717/peerj-cs.655

[2] S. Arya and R. Singh, "A Comparative Study of CNN and AlexNet for Detection of Disease in Potato and Mango leaf," 2019 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), 2019, pp. 1-6, doi: 10.1109/ICICT46931.2019.8977648.

[3] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

[4] Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv. https://doi.org/10.48550/arXiv.1409.1556

[5] https://www.cs.toronto.edu/~kriz/cifar.html

[6] https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/

[7] https://medium.com/@amir_hf8/implementing-vgg13-for-mnist-dataset-in-tensorflow-abc1460e2b93

[8] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. https://doi.org/10.48550/arXiv.1512.03385