Analysis of CNN Architectures on Image Classification Tasks

Kiana Hadysadegh

***Abstract*— Image Classification with different CNN Architectures.**

I. Introduction

Recently much progress has been made in the Image classification task. Tuning parameters have an important role in classifying images. Nowadays researchers use data augmentation techniques on the training dataset to bring more diversity and improve the accuracy of unseen data. In this project, we will train GoogLeNet, Resnet18, and VGG16 on MNIST, CIFAR10, and CIFAR100 in different settings. Achieving the highest possible accuracy in these experiments is a solved problem undoubtedly. Our goal is to examine the effect of different hyperparameters and training tricks on model performance in practice. We trained more than 50 models in different training conditions. In the next section, we will discuss two main trends in the image classification task. Model architectures are described in detail in section 3. Section 4 is dedicated to training tricks discussion. We will explain our experiments in detail in section 5. Conclusions and Limitations of this work are given in the last two sections.

II. Related Work

The goal of the image classification problem is to classify the input image within the expected label correctly. We can divide the most common and recent image classification techniques into two main trends:

A. Methods based on Convolution Neural Networks

Convolutional Neural Networks (CNN) are a class of deep, feed-forward artificial neural networks. They can be successfully applied for analyzing images. The main structure of CNN is the convolutional layer, pooling layer, nonlinear activation layer, and fully connected layer. Through the input layer, the image is preprocessed and then input into the network. Then processed by several alternately arranged convolutional layers and pooling layers, and classified by the fully connected layer.

B. Methods based on Transfer Learning

In transfer learning the knowledge of an already trained model is transferred to a different linked problem; For example, if we trained a simple classifier to predict whether an image contains food, we could use the model's training knowledge to identify other objects such as drinks [9]. The idea is to use the existing model and only adjust it to our data. Those models can predict images with very high accuracy. There are lots of models that can recognize thousands of labels and in which millions of images are used during the learning process. The number of images used in those big models (like the ImageNet database used for the Inspection v3 model) is often very hard to obtain in private (not benchmarked) image recognition problems [4]. The idea is to use features for millions of objects from bigger projects to predict different labels than in an original project.

III. Model Description

In this section, the architecture of the different models is going to be discussed.

A. Model Architecture

i. VGG16

VGG is a standard deep Convolutional Neural Network (CNN) architecture with multiple layers. It consists of tiny convolutional filters. The VGG-16 (Fig.1) is constructed from 13 convolutional layers and three fully connected layers.

Input: An image size of 224×224 is used as an input of the network.

Convolutional Layers: The network has some 1×1 convolution filters. These filters do linear transformations on the input. To reduce the training time, the previous part is followed by a ReLU unit. It should be noted that to keep the spatial resolution after convolution, the stride of the convolution is fixed at 1 pixel.

Hidden Layers: The ReLU activation function is used in all hidden layers of the network.

Fully-Connected Layers: There are three fully connected layers in the network. Each of the first two has 4096 channels, while the third one has one channel per class (1000 channels).

The number 16 in the name VGG refers to the fact that it is a 16-layer deep neural network (VGGnet).

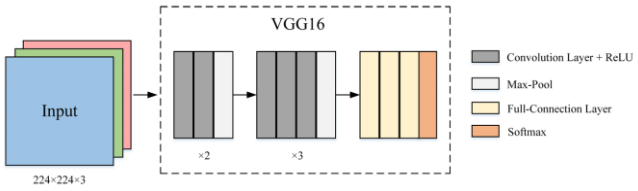


Fig.1. The architecture of the VGG-16 network [9]

ii. Resnet 18

The basic building blocks of ResNet are named residual blocks or residual units. These blocks are repeated throughout the network, with variations in their configuration depending on the specific ResNet architecture (e.g., ResNet-18, ResNet-50, etc.).

Residual Blocks (Fig. 2) is a solution created to address the issue of the vanishing/exploding gradient. These blocks have skip connections. The skip connection bypasses some levels in between to link-layer activations to subsequent layers. Skip connections allow the network to learn residual mappings by skipping one or more layers and directly adding the input from an earlier layer to the output of the block.

As a result of this connection, a leftover block is created. Resnests are built by stacking these leftover blocks. Instead of having layers to learn underlying mapping, we let the network fit the residual mapping:

$$F(x) := H(x) - x \rightarrow H(x) := F(x) + x$$

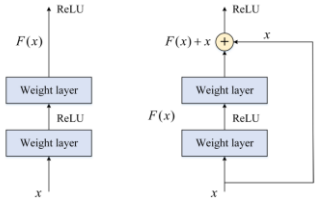


Fig.2 Comparison of ordinary CNN(left) and Residual learning (right) [9]

The advantage of skip connections is that regularization will skip any layer that degrades architecture performance. Therefore we can train extremely deep neural networks without facing vanishing or expanding gradients.
ResNet-18 specifically consists of 18 layers and is known for its effectiveness in image classification tasks. Each block is constructed from multiple convolutional layers, batch normalization layers, and activation functions, along with skip connections.

iii. GoogLeNet (InceptionV1)

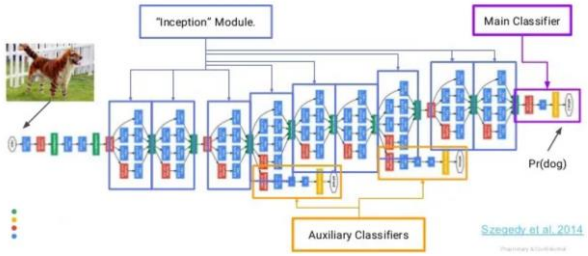


Fig.3 InceptionV1 Architecture [16]

The idea of this architecture (Fig.3) is that individual devices can run the model (even with low computational resources). GoogLeNet (Appendix. Fig.4) has 22 layers (6M parameters). it uses different methods to create deeper architecture such as 1×1 convolution, and max pooling.

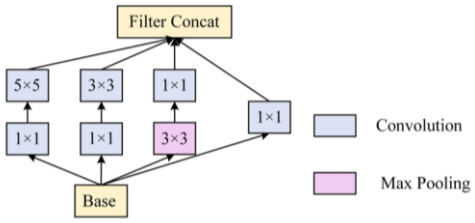


Fig.4 Inception V1 module [9]

The basic module of the network is the Inception module (Fig.4). This module contains 4 parallel branches. The first three branches use convolutional layers with different sizes to extract information under different spatial sizes. We have 1×1 convolutions that can reduce the number of channels and compress information to reduce model complexity. The max-pooling effect of the last branch is to reduce the resolution, followed by 1×1 convolution to adjust the depth after the pooling.
They perform parallel to the input, and the output of each is stacked together to generate the final output. The idea is convolution filters of different sizes will handle objects at multiple scale better.
Auxiliary Classifier: In the middle of the architecture, there are some intermediate classifier branches that are only used in the training time.
These branches consist of: a 5×5 average pooling layer with a stride of 3, 1×1 convolutions with 128 filters, two fully connected layers of 1024 outputs and 1000 outputs, and a softmax classification layer. The generated loss of these layers added to the total loss with a weight of 0.3. The role of these layers is to combat the gradient vanishing problem and provide regularization.
This unique design increases the width of the network model and adapts it to different scales or resolutions.

IV. TRAINING TRICKS

In this section, we will analyze some training tricks to increase the accuracy of our models. We also dive deeper into the effects of each trick.

A. Adapt models to the dataset

In order to use GoogLeNet, Resnet18, and VGG16 we had to resize the images of our datasets to 224×224 pixels and normalize them. We also had to make some modifications in model architectures: (a) we changed the output of the last layers of each model based on the numbers of the total classes of each dataset. (b) for the MNIST dataset, we only have one color channel therefore we changed the number of the input channels of the models to one for experiments with this specific dataset.

B. Hyperparameter tune

Hyperparameter tuning is the process of selecting the optimal values for a machine learning model's hyperparameters. This process has significant importance in training models. It should be noted finding the optimal parameters is a time-consuming process. In this project, we tried to tune the below parameters in different experiments: Number of Epochs, Batch size, learning rate

C. Optimizers

We analyzed two types of optimizers: Stochastic Gradient Descent, and Adam. The key difference between these two optimizers is that SGD uses a fixed learning rate but Adam adjust it on a per-parameter basis. Also, Adam converges faster than SGD especially when we have large datasets with many parameters. This is because of its adaptive learning rate techniques. SGD is effective in many cases, however, Adam is often preferred in training deep neural networks due to its robustness. Adam is capable of handling challenges like vanishing or exploding gradients.

D. Learning rate scheduler

Learning rate schedules seek to adjust the learning rate during training by reducing the learning rate according to a pre-defined schedule. The learning rate is a crucial hyperparameter when training neural networks. This is because each time the model weights are updated it controls how much to change the model in response to the estimated error.

In this project, we used the Cosine Annealing learning rate scheduler. This method adapts the learning rate during the training of neural networks. It follows a cosine curve between an initial learning rate and a final, lower boundary over a specified number of training epochs or steps.

E. Data Augmentation

Data augmentation techniques are used to increase the amount and diversity of the dataset. Using this technique we add slightly modified copies of the existing images to the dataset. Data augmentation helps to overcome overfitting and improve model performance. In this project, we used random horizontal flips, random crops, random rotations, and color augmentation to overcome overfitting and improve model generalization.

V. EXPERIMENTS

Table 1 (Appendix) and Figure 5 (Appendix) demonstrate the results of our experiments in detail. The accuracy and loss curves for each trial have been provided in figures 12-17 (Appendix). Table 1 and Figures (8-10) demonstrate the results and performance of the top-trained models, respectively. The cross-entropy loss was used in all trials.

A. Datasets

In this project, we used three datasets for the classification task: MNIST, CIFAR10, and CIFAR100.

MNIST: The MNIST dataset is a large database of handwritten digits (0 to 9) that is commonly used for training various image processing systems. It contains 60,000 training images and 10,000 testing images. The images are 28x28 pixels and in one channel (black and white) (Fig.1. Appendix). There are some labeling issues in this dataset that we will discuss in the testing results.

CIFAR10: The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class (Fig.2 Appendix). There are 50000 training images and 10000 test images. The classes are completely mutually exclusive.

CIFAR100: This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 super classes. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs).

B. Experiments on MNIST

Figure 5. (a) (Appendix) demonstrates all experiments done using this dataset. It is worth noting that in every examination we resized our input image to 224×224 pixels, and changed the number of classes of the last linear layer to 10. Since this dataset has only one color channel we changed the number of channels in the first convolution layer to 1 (except in one experiment that we try to change dataset channels to 3 and create redundancy). It should be noted that the MNIST dataset is too simple for these deep architectures, therefore observing significantly high accuracies in the first epochs was expected to happen.

GoogLeNet: We started from 10 epochs, with a learning rate of 0.01, batch size of 64, and SGD optimizer, and we observed 99.52% test accuracy. To increase the accuracy of test data we picked two different approaches. The first approach was to increase the number of epochs to 50 which improved it to 99.64 and the second one was to use data augmentation techniques. Using random flip, crop, and rotate methods we increased dataset size and diversity to improve generalization. After applying this method the accuracy of the test data decreased to 98.86% which can indicate that we should increase the number of epochs and batch size or decrease the learning rate to achieve higher accuracy for the new and diverse dataset. This is because we have a larger dataset therefore more interaction with the dataset and slower training may be needed.

*Resnet18:*We started from 10 epochs, with a learning rate of 0.01, batch size of 64, and SGD optimizer, and observed 99.54% test accuracy. We increased the epochs to 50 and the test accuracy improved by 0.03%. As a result of using data augmentation techniques (flip, crop, rotate) with 40 epochs, we observed that the test accuracy dropped to 99.02%. This can indicate that due to the dataset diversity and size increment, the model may need more interaction with data through training or a slower training process to generalize better.

*VGG16:*We started from 10 epochs, with a learning rate of 0.001, the batch of size 4, and SGD optimizer, and observed 99.53% test accuracy. We increased the epochs to 50 and the test accuracy dropped 0.01%. Using the data augmentation technique we increased the size and diversity of the dataset and trained a new model. The test accuracy dropped to 98.90% that can be because of the same reasons as the other architectures.

We also tried to use another approach and instead of changing the number of input channels of the first convolution layer, we increased the channels of the dataset to 3 using different grayscales to see the result (test accuracy 99.37%).

C. Experiments on CIFAR10

For all experiments, we resized our input image to 224×224 pixels, changed the input convolution layer channels to 3, and changed the number of classes of the last linear layer to 10.

GoogLeNet: Figure 5. (b) (Appendix) shows the results of our experiments. We started with 10 epochs and a learning rate of 0.01 with a batch size of 64 and SGD optimizer. The test accuracy was 84.84% which is relatively high due to the small size of CIFAR10 and complexity of GoogLeNet. To enhance the result and examine the influence of different training tricks we did some other experiments; we slightly increased the number of epochs and decreased the learning rate and the test accuracy improved by around 3 percent. Then in another test, we changed the optimizer to Adam, increased the epoch numbers to 30, and slightly decreased the learning rate but

the test accuracy dropped by 1 percent. As a result, we keep using SGD for the rest of the experiments and use data augmentation techniques such as random flip, crop, and rotate to increase the dataset size and variety. Due to this change, we increased the batch size and epoch numbers and observed the more augmentation techniques we used; the higher test accuracy we have. (The best test accuracy we recorded was 92.04%)

Resnet18: Figure 5. (c) (Appendix) shows our result for this model. For all experiments, we resized our input image to 224×224 pixels and changed the number of classes of the last linear layer to 10. Using random horizontal flip we added variety to our dataset in all tests. We started with 20 epochs, a learning rate of 0.001, a batch size of 32, and an SGD optimizer and we got 81.32% test accuracy. To monitor the influence of other tricks, we did some other tests; we increased the number of epochs to 30 and observed there was no considerable improvement in the test accuracy meaning that the training time was sufficient. Therefore we changed the optimizer to Adam and got a 4% improvement in the test accuracy. We increased the number of epochs and batch size and the test accuracy increased by 1%. By applying other data augmentation techniques to our dataset and increasing the batch size and epoch numbers we achieved a 3% improvement in the last model (89.18%).

VGG16: Figure 5. (d) (Appendix) demonstrates the results of experiments that have been done using this model. For all experiments, we resized our input image to 224×224 pixels, and changed the number of classes of the last linear layer to 10. Due to the first interactions with the model, we realized that it needs more training time in comparison to previous models. Therefore we started with 50 epochs, a learning rate of 0.01, batch size of 64, and SGD optimizer (We used Adam optimizer and observed that it is not efficient for this case) and got 76.42% test accuracy which is significantly less than two other architectures. To enhance it, we continuously add different data augmentation techniques (flip, crop, rotate) to increase the variety of our dataset in the next trials. Due to this dataset size increase, we decreased the learning rate and increased the number of epochs and batch size to give more training time and interaction with the data. We observed that the more we used the previous technique, the better test accuracy we achieved (best model test accuracy 84.19%).

D. Experiments on CIFAR100

CIFAR100 is a significantly larger dataset than the previous two, therefore, it is expected that training the models on this dataset and achieving higher test accuracy is more challenging. It should be noted that due to this difficulty we did many experiments and explaining all of them is out of the scope of this report. Table 1 (Appendix) shows the exact details of every experiment has been done by us. In all trials, we resized the input image to 24x24 pixels and changed the number of classes in the last linear layer of the model architecture to 100.

*GoogLeNet:*The experiment started from 10 epochs, a 0.01 learning rate, a batch size of 64, and the SGD optimizer The test accuracy of 54.79% has been observed (train accuracy of 70.39%). We changed the optimizer to Adam and increased the learning rate and number of epochs and the test accuracy improved to 58.69% (train accuracy 98.68%). According to the achieved train accuracy, it is obvious that overfitting has happened. To reduce the influence of overfitting other experiments have been done by us. We use random horizontal flips and rotates to increase the diversity of the dataset and retrain the model. The train and test accuracy was 88.50% and 64.72% respectively. This indicates the effect of the data augmentation techniques in improving the generalization of the model and avoiding overfitting. We increased the batch size to 128 and the number of epochs to 40 and did two other experiments with learning rates of 0.001 and 0.006. We observed that the test accuracy has improved by 2 percent. Meanwhile, the results produced by the model with the lower learning rate (0.001) increased the overfitting effect potentially due to being stuck in the local minima.

Since the periodic adjustment of the cosine learning rate scheduler can help us escape from the local minima and overfitting, We did two other experiments. Due to the increase in the overfitting problem, the random crops have been used to increase the diversity and size of the dataset for better generalization: in the first examination we use 200 epochs, a learning rate of 0.01, batch size of 128, Adam optimizer and the test accuracy of 70.77% has been observed. The learning rate was reduced to 0.007 in the next test but the test accuracy didn't change considerably. Figure 5. (f) (Appendix) demonstrates the results of experiments done using this model.

*Resnet18:*The experiment started with 20 epochs, a 0.01 learning rate, a batch size of 32, and the SGD optimizer, and the test accuracy of 62.07% was observed (train accuracy of 95.50%). This was a case of overfitting so we increased the size and diversity of the dataset using data augmentation techniques (random horizontal flip, rotate, and color jitter) to improve generalization. Meanwhile, we increased the number of epochs and batch size to 30 and 64 in turn. The test accuracy improved by 3%. We changed the optimizer to Adam (learning rate 0.001) in the next experiment to observe its influence but the test accuracy decreased by 3%. To reduce the overfitting, we took another approach and increased the learning rate to 0.06 with SGD and 40 epochs and we got 62.72% test accuracy. This is similar to our first experiment but with less overfitting which indicates that we were probably stuck in the local minima in that experiment.

We performed two other experiments using a cosine learning scheduler to reduce overfitting and escape the local minima with its periodic change: the experiment was done with a batch size of 64, SGD optimizer (learning rates of 0.01, 0.007 and 100, 200 epochs respectively). This approach was successful and increased the test accuracy and model generalization by 10 percent. Figure

5. (g) (Appendix) demonstrates the results of experiments done using this model.

*VGG16:*Figure 5. (h) (Appendix) shows our result for this model. The experiments started with 50 epochs, a 0.006 learning rate, a batch size of 64, and SGD optimizer. We observed 31.80 % test accuracy (82.50% train accuracy) which was a case of overfitting. To reduce the overfitting and improve model generalization we use random horizontal flips data augmentation technique. As a result, the test accuracy increased to 42.68% (train accuracy 73.65 %) which indicates the influence of data augmentation techniques to reduce overfitting. Due to this size increase of the dataset, another model with 70 epochs, batch size of 128, and a learning rate of 0.004 was trained but the test accuracy dropped to 33.88%. Therefore, the learning rate has been increased to 0.007 and the test accuracy raised to 43.52% which can prove probably the model was probably stuck in the local minima.
To enhance the model's generalization, we used other augmentation techniques such as random crops and rotates in addition to random horizontal flips and increased dataset size. Meanwhile, we used a cosine learning scheduler to avoid overfitting due to its ability to periodically adjust the learning rate and pass the local minima. The experiment was done with 200 epochs, a 0.007 learning rate, and the SGD optimizer. A test accuracy of 66.99% has been observed (train accuracy of 99.25%).

E. Testing results

In this part, we will analyze the test results of our best-trained model for each model and dataset combination.
Fig.5 shows three cases of failure and success of classification tasks done by our best models on the MNIST dataset. By taking a deeper look at the cases of failure, we can observe that there are cases of labeling mistakes in the MNIST dataset. These wrong labels can be one of the reasons that we couldn't improve our test accuracy after a certain point.

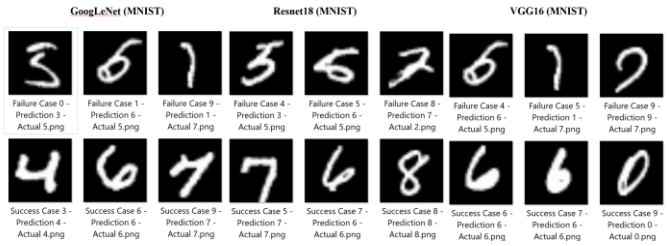


Fig.5 Models predictions on the MNIST dataset

Figures 6 and 7 demonstrate the predictions of our top three trained models on the CIFAR10 and CIFAR100 datasets, respectively. Three cases of successful and failed predictions of each model are shown in the figures:

Fig.6 Models predictions on the CIFAR10 dataset

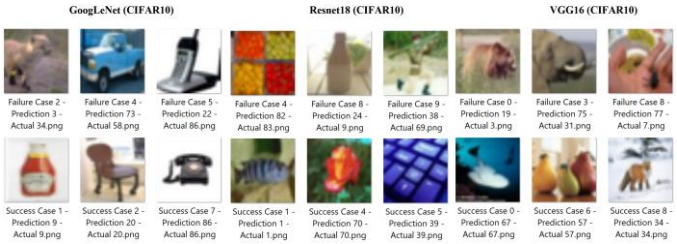


Fig.7 Models predictions on the CIFAR100 dataset

VI. CONCLUSION

As expected, there is a direct relationship between the size and diversity of the dataset and the challenges of training. We observed how fast our models achieved significantly high test accuracies on the MNIST dataset in the first epochs due to its small size and simplicity. It is worth mentioning that the complexity of these deep models also influences these results.
One of the biggest challenges we encountered was overfitting. Overfitting means that our model learns the training data too well and even captures the noises. Therefore, it has a great performance in train data but not in unseen data (test and validation accuracies are low). Our goal was to use different tricks to see if we could reduce this effect in our trials. We used different Data Augmentation techniques to increase the diversity and size of our dataset to improve the model's generalization ability on unseen data. As we mentioned earlier, our result confirmed the efficiency of this technique in reducing the influence of overfitting.
Being stuck in the local minima can cause overfitting. Therefore, we took two other approaches to escape from the minima:

1. We tune the learning rate hyperparameter manually to escape from the local minima; it should be noted that picking too high a learning rate can lead to passing the global minima and overshooting the loss. Therefore, we should increase the learning rate carefully.

2. We use the Cosine Annealing learning rate scheduler to use its periodic learning rate adjustments. These adjustments can help us escape the local minima.

Our results prove the efficiency of these two approaches.
One of the issues we faced was after a certain point we couldn't improve the models' generalization on the MNIST dataset. This was due to the mislabeled data in this dataset that we showed in Fig.5.
Table 1 demonstrates the settings in which our top models were trained and the results obtained. In terms of test accuracy, our GoogLeNet model has the highest accuracy on unseen data from MNIST and CIFAR10. However, in CIFAR100, our trained Resnet18 model achieved higher accuracy compared to the other top models.

Dataset	model	method	Epochs	Learning rate	batch size	lr scheduler	optimizer	Loss	data augmentation	test accuracy	train accuracy
MNIST	GoogLeNet	change input layer	50	0.01	64	no	SGD	Cross Entropy	no	99.64%	100.00%
MNIST	Resnet18	change input layer	50	0.01	64	no	SGD	Cross Entropy	no	99.57%	100.00%
MNIST	VGG16	change input layer	10	0.001	4	no	SGD	Cross Entropy	no	99.53%	99.79%
CIFAR10	GoogLeNet	change input layer	85	0.006	128	no	SGD	Cross Entropy	yes	92.04%	99.94%
CIFAR10	Resnet18	change input layer	100	0.001	64	no	Adam	Cross Entropy	yes	89.18%	94.00%
CIFAR10	VGG16	change input layer	100	0.001	128	no	SGD	Cross Entropy	yes	84.19%	95.08%
CIFAR100	GoogLeNet	change input layer	200	0.007	128	yes	Adam	Cross Entropy	yes	70.80%	99.91%
CIFAR100	Resnet18	change input layer	200	0.007	64	yes	SGD	Cross Entropy	yes	72.24%	99.98%
CIFAR100	VGG16	change input layer	200	0.007	128	yes	SGD	Cross Entropy	yes	66.99%	99.25%

Table 1. Training details of our top models

The comparison of our top 3 models on the MNIST dataset has been shown in Fig. 8. We can observe that although our GoogLeNet model converges slower to the lowest training loss than the other two, it has a better performance on unseen data. The fluctuations of the model indicate that its predictions are less confident.

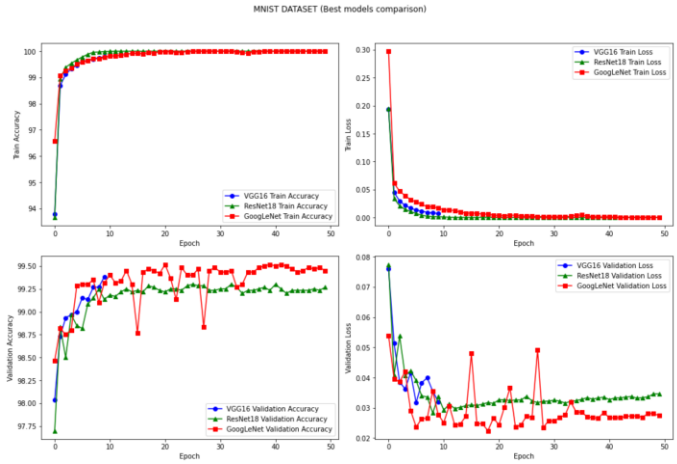


Fig. 8 Top trained models on the MNIST dataset

Fig.9 demonstrates the performance of our top three models on the CIFAR10 dataset. As shown below, our GoogLeNet model has the best performance and faster convergence in the both training and evaluation parts. We observe fewer fluctuations in this graph compared to the previous one, meaning that our predictions are more confident.

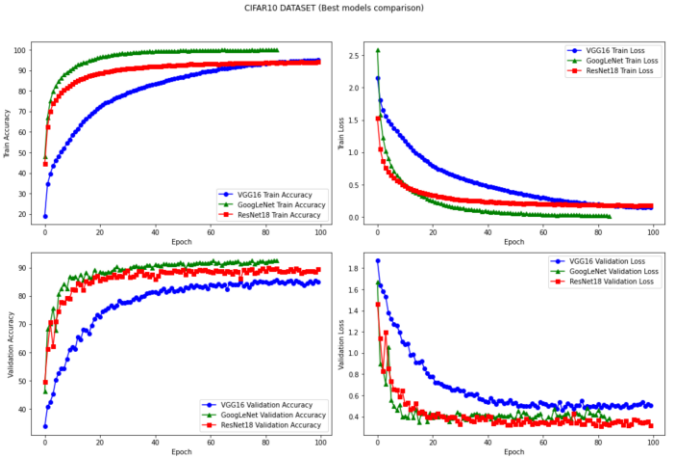


Fig.9 Top Trained models on the CIFAR10 dataset

The results of our top trained models on the CIFAR100 dataset have been shown in Fig. 10. This dataset is relatively larger than the other two and the training is more challenging. Our Resnet18 model has better performance and faster convergence in both training and evaluation tasks. The fluctuations in the GoogLeNet and Resnet18 models can indicate that they might be getting more correct answers but with less confidence. Meanwhile, our VGG16 model predictions can be less accurate (lower accuracy) but with more confidence (lower loss).

Another interesting phenomenon is that although we have fluctuations, we see different ranking patterns in loss and accuracy graphs. This can prove the lack of confidence in predictions of our GoogLeNet model; we can see that the model has a high loss (less confident predictions) but also high accuracy simultaneously. This means that although it predicts more correct answers, its predictions are not confident since the probabilities assigned to the correct classes are lower and the cross-entropy loss will be higher.

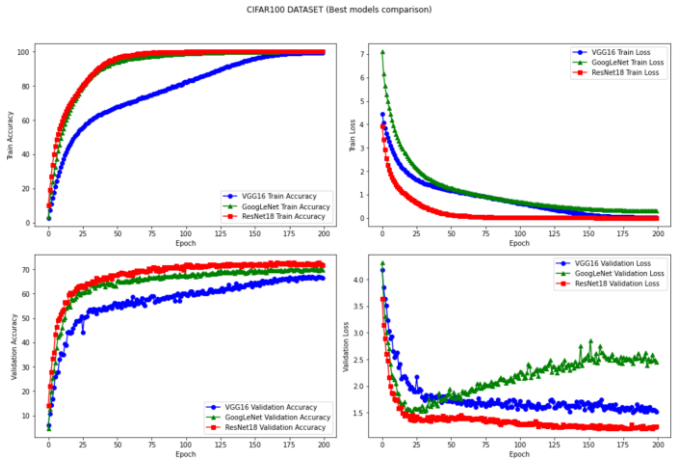


Fig.10 Top trained models on the CIFAR100 dataset

To conclude, the fluctuations in the graphs can indicate that our predictions are less confident. This is because the probabilities assigned to the correct classes are lower therefore the cross-entropy loss will be higher. This can occur even if the top prediction (highest probability) is correct, hence good accuracy but poor loss performance.

VII. Limitations and Future Trends

One of the main disadvantages of the VGG16 model is its high computational cost (around 138 million parameters) and long training time. Resnets can achieve high levels of accuracy in image classification tasks. However, potential disappearance of gradients in very deep networks may occur, which can make the gradient descent process slow. Although GoogLeNet brings the idea of efficient computational resource usage, its complex architecture and implementation should be taken into account. The image classification task is still an important research topic in the world. Recent Vision Transformer's achievements in image classification tasks cannot be underestimated. How to effectively combine convolution and Transformer has become one of the current hot spots for further research.

REFERENCES

[1] Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[2] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., 2015. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

[3] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[4] Szyc, K., 2018, June. Comparison of different deep-learning methods for image classification. In 2018 IEEE 22nd International Conference on Intelligent Engineering Systems (INES) (pp. 000341-000346). IEEE.

[5] Chen, L., Li, S., Bai, Q., Yang, J., Jiang, S. and Miao, Y., 2021. Review of image classification algorithms based on convolutional neural networks. Remote Sensing, 13(22), p.4712.

[6] Wang, P., Fan, E. and Wang, P., 2021. Comparative analysis of image classification algorithms based on traditional machine learning and deep learning. Pattern Recognition Letters, 141, pp.61-67.

[7] Rawat, W. and Wang, Z., 2017. Deep convolutional neural networks for image classification: A comprehensive review. Neural computation, 29(9), pp.2352-2449.

[8] Dovbnych, M. and Plechawska-Wójcik, M., 2021. A comparison of conventional and deep learning methods of image classification. Journal of Computer Sciences Institute, 21.

[9] Bichri, H., Chergui, A. and Hain, M., 2023. Image Classification with Transfer Learning Using a Custom Dataset: Comparative Study. Procedia Computer Science, 220, pp.48-54.

[10] He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J. and Li, M., 2019. Bag of tricks for image classification with convolutional neural networks. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 558-567).

[11] Shah, S.R., Qadri, S., Bibi, H., Shah, S.M.W., Sharif, M.I. and Marinello, F., 2023. Comparing inception V3, VGG 16, VGG 19, CNN, and ResNet 50: a case study on early detection of a rice disease. Agronomy, 13(6), p.1633.

[12] Krizhevsky, A., The CIFAR-10 dataset, viewed 20 April 2024, https://www.cs.toronto.edu/~kriz/cifar.html.

[13] VISo AI, VGG – Very Deep Convolutional Networks, viewed 20 April 2024, https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/.

[14] Siddhesh, B. 2020, ResNet Architecture Explained, Medium, viewed 20 April 2024, https://medium.com/@siddheshb008/resnet-architecture-explained-47309ea9283d.

[15] GeeksforGeeks, Understanding GoogLeNet Model – CNN Architecture, viewed 20 April 2024, https://www.geeksforgeeks.org/understanding-googlenet-model-cnn-architecture/

[16] Abheer, C., Inception V1 Architecture Explained, Medium. Viewed 20 April 2024, https://medium.com/@abheerchrome/inception-v1-architecture-explained-454b2eb66baf

Appendix

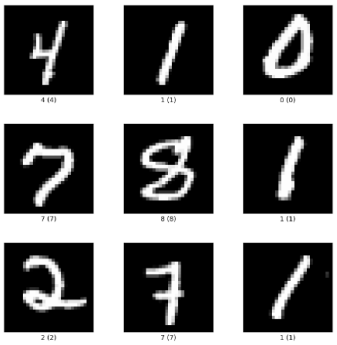


Fig.1 MNIST dataset

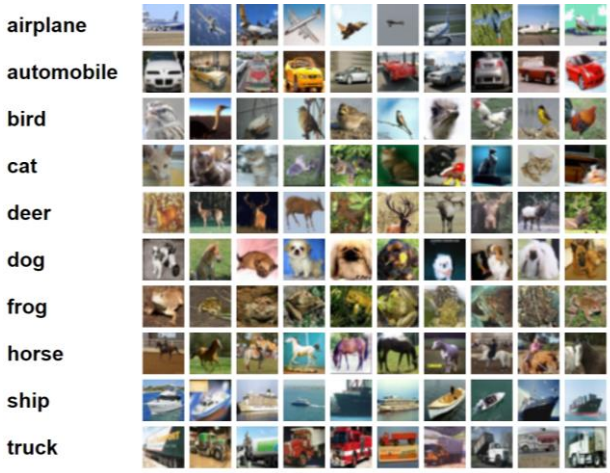


Fig. 2 CIFAR10 dataset

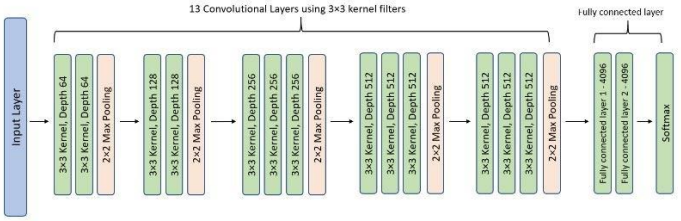


Fig.3 Resnet18 architecture

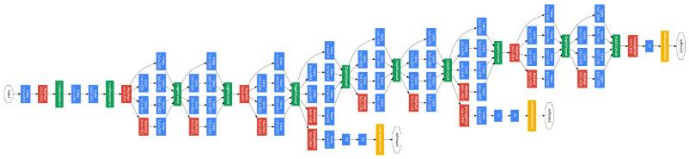


Fig.4 Inception V1 architecture

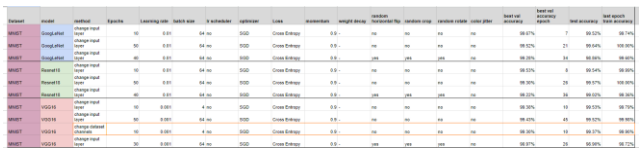


Fig.5 (a) Experiment details on MNIST

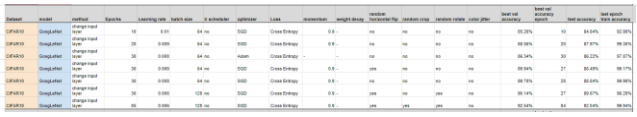


Fig.5 (b) Experiment details on CIFAR10 (GoogLeNet)

Fig.5 (c) Experiment details on CIFAR10 (Resnet18)

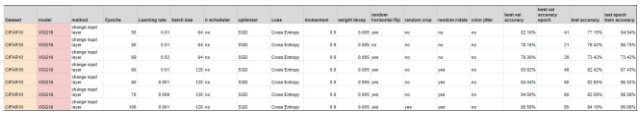


Fig.5 (d) Experiment details on CIFAR10 (VGG16)

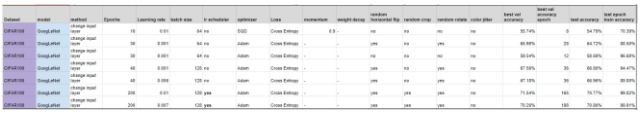


Fig.5 (e) Experiment details on CIFAR100 (GoogLeNet)

Fig.5 (f) Experiment details on CIFAR100 (Resnet18)

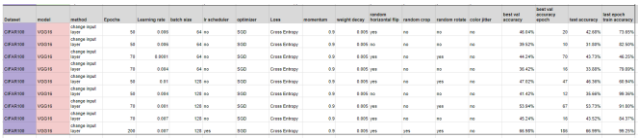


Fig.5 (g) Experiment details on CIFAR100 (VGG16)

Fig.6 MNIST Cases of Success and Failure (a) GoogLeNet

Fig.6 MNIST Cases of Success and Failure (b) Resnet

(c) VGG16

Fig.6 MNIST Cases of Success and Failure (c) VGG16

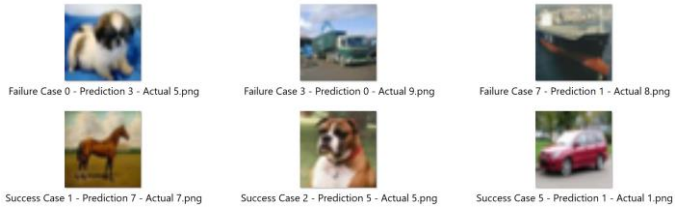


Fig.7 CIFAR10 Cases of Success and Failure (a) GoogLeNet

Fig.7 CIFAR10 Cases of Success and Failure (b) Resnet18

Fig.7 CIFAR10 Cases of Success and Failure (c) VGG16

Fig.8 CIFAR100 Cases of Success and Failure (a) GoogLeNet

Fig.8 CIFAR100 Cases of Success and Failure (b) Resnet18

Fig.8 CIFAR100 Cases of Success and Failure (c) VGG16

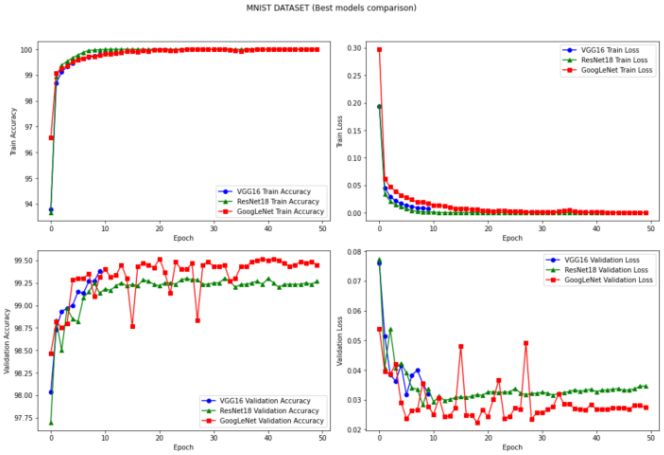


Fig.9 Top Trained models on the MNIST dataset

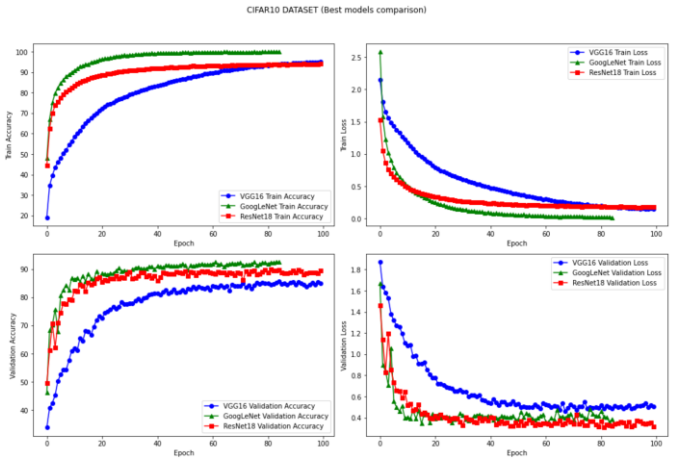


Fig.10 Top Trained models on the CIFAR10 dataset

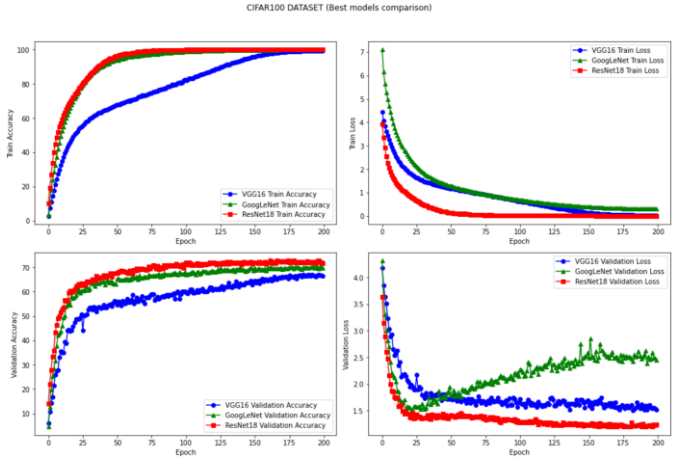
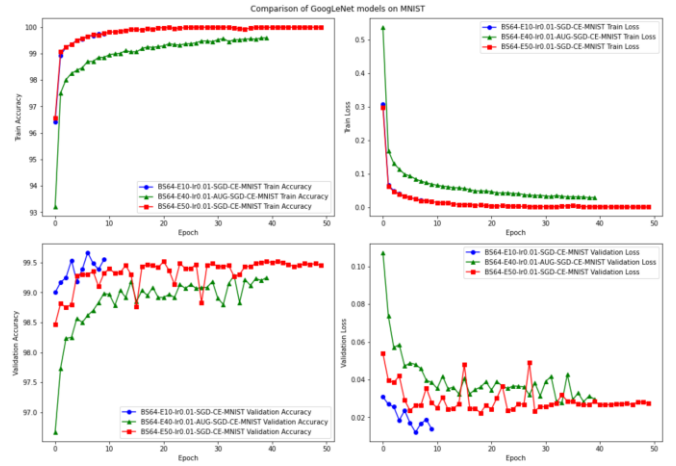


Fig.11 Top Trained models on the CIFAR100 dataset



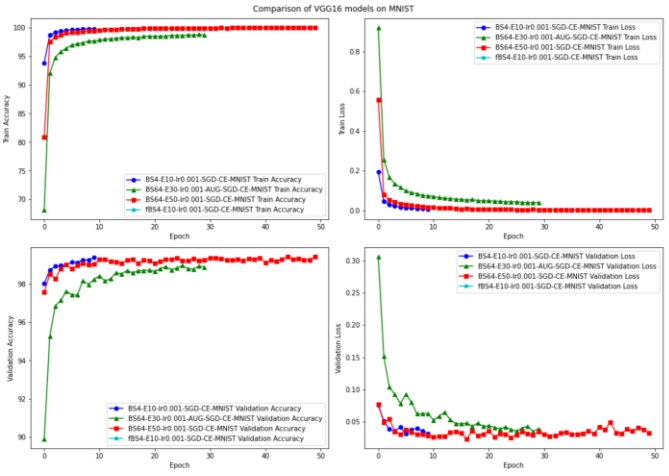


Fig.12 Comparison of different trained models on MNIST

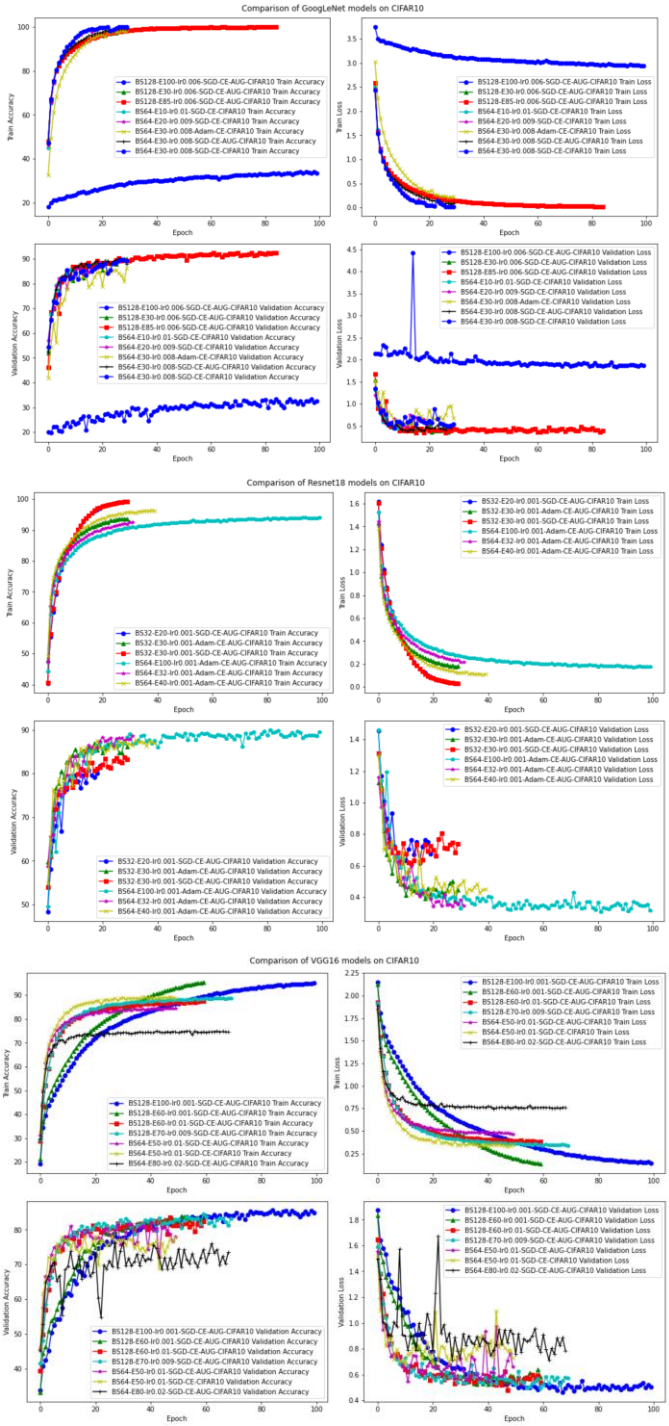


Fig.13 Comparison of different trained models on CIFAR10

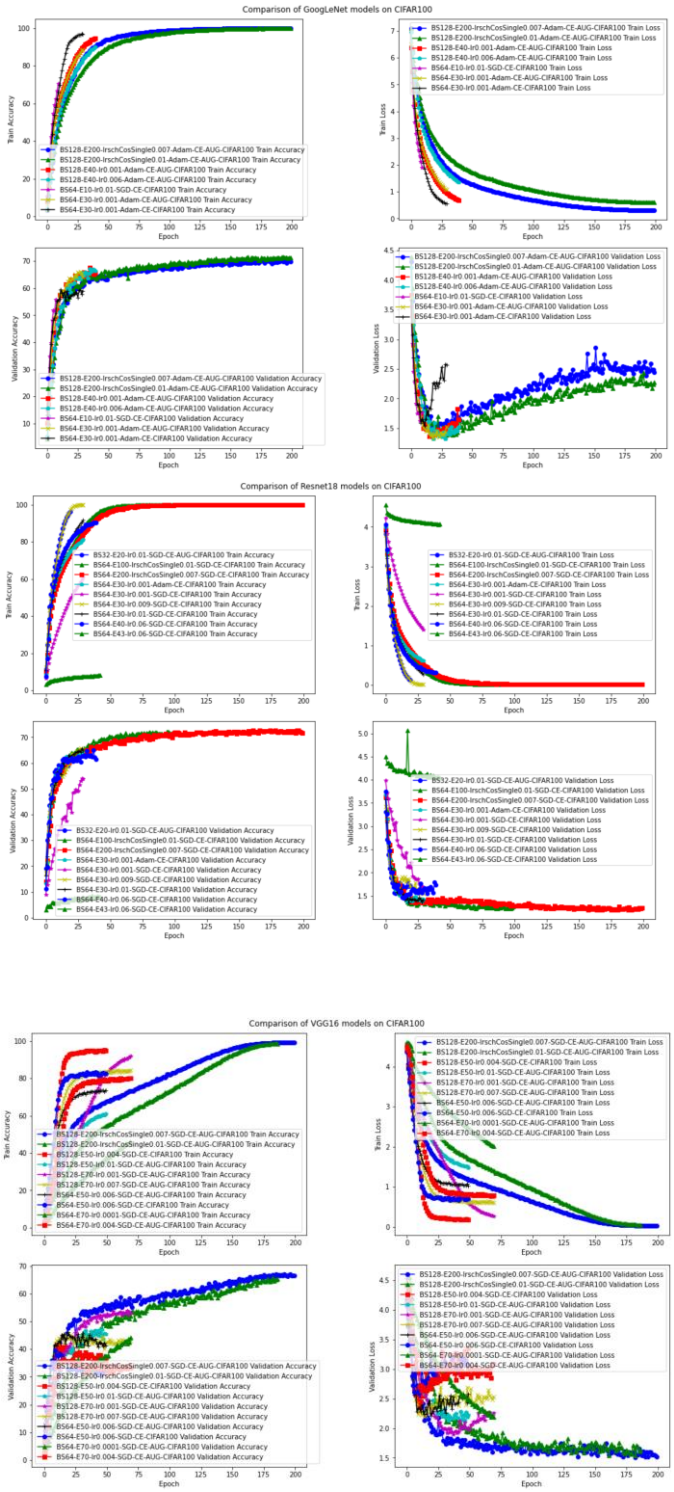
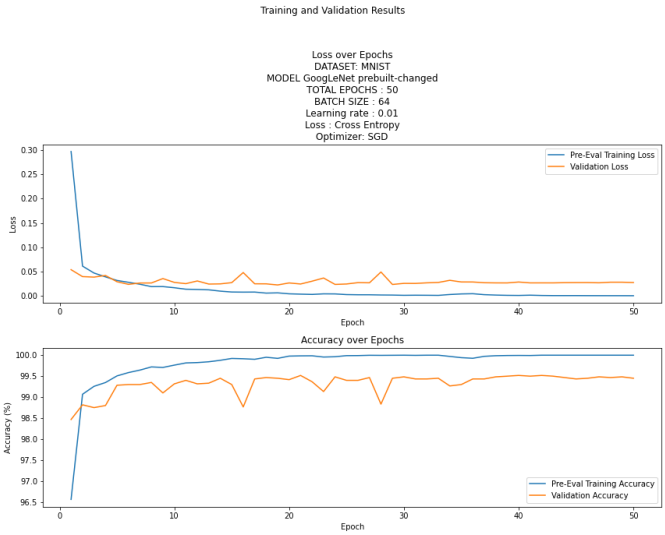
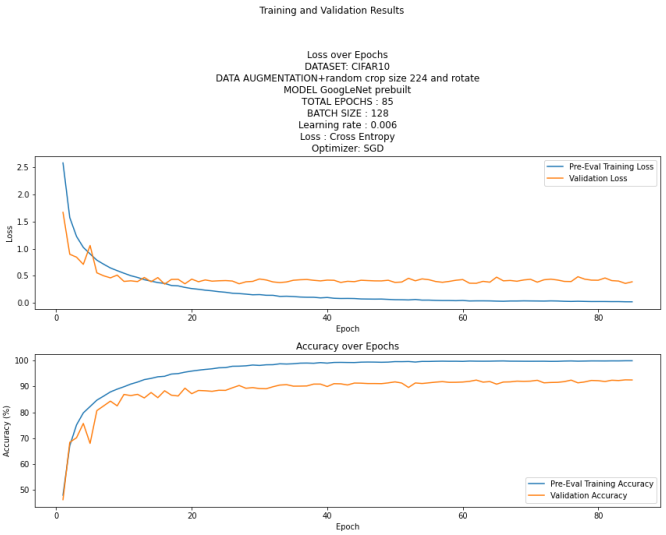
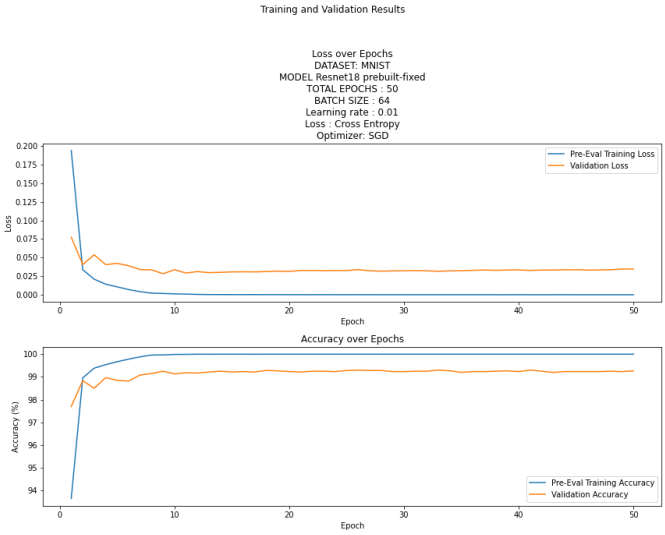
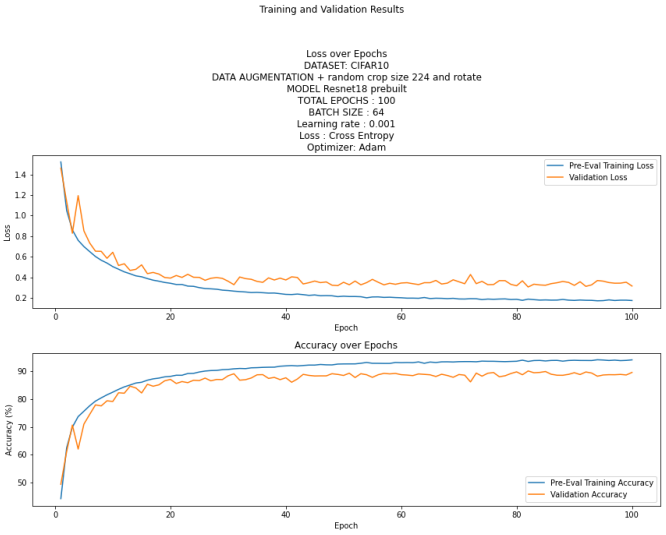


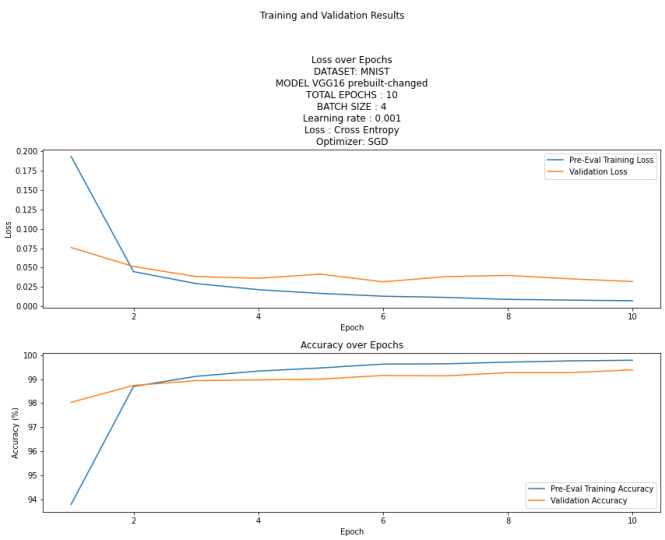
Fig. 14 Comparison of different trained models on CIFAR100











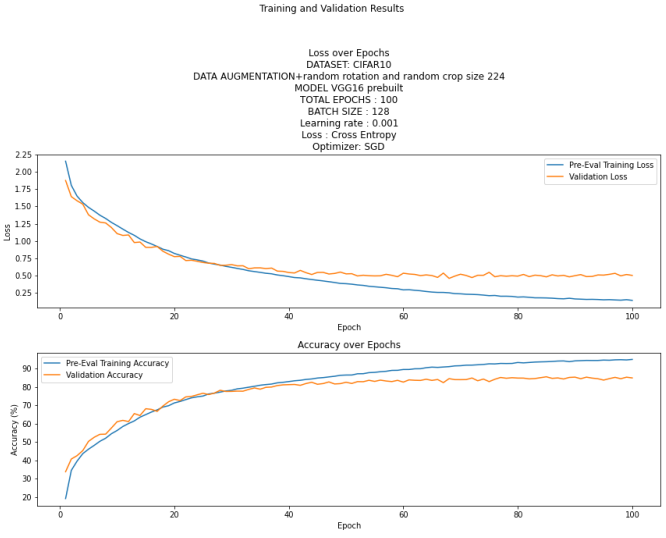
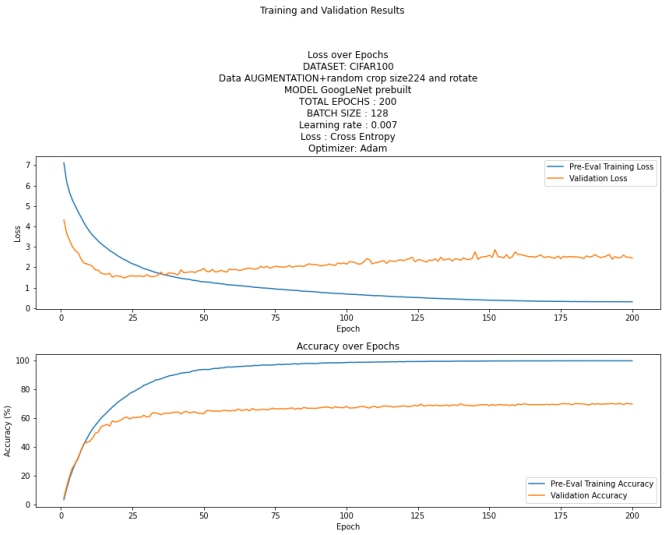
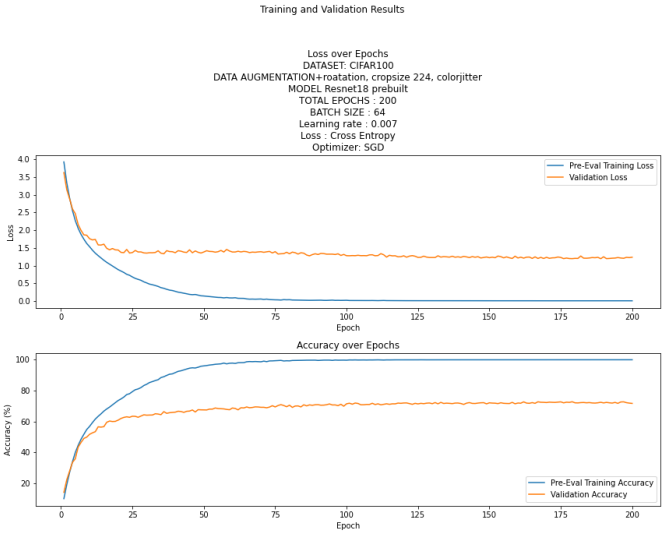


Fig. 15 Top trained models on MNIST single plots

Fig. 16 Top trained models on CIFAR10 single plots





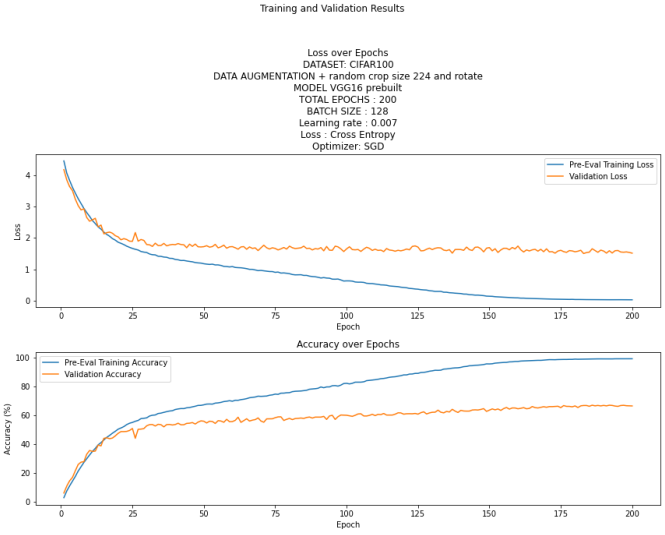


Fig. 17 Top trained models on CIFAR10 single plots

Table.1

Dataset	model	method	Epochs	Learning rate	batch size	lr scheduler	optimizer	Loss	momentum	weight decay	random horizontal flip	random crop	random rotate	color jitter	best val accuracy	best val accuracy epoch	test accuracy	last epoch train accuracy
MNIST	GoogLeNet	change input layer	10	0.01	64	no	SGD	Cross Entropy	0.9	-	no	no	no	no	99.67%	7	99.52%	99.74%
MNIST	GoogLeNet	change input layer	50	0.01	64	no	SGD	Cross Entropy	0.9	-	no	no	no	no	99.52%	21	99.64%	100.00%
MNIST	GoogLeNet	change input layer	40	0.01	64	no	SGD	Cross Entropy	0.9	-	yes	yes	yes	no	99.28%	34	98.86%	99.60%
MNIST	Resnet18	change input layer	10	0.01	64	no	SGD	Cross Entropy	0.9	-	no	no	no	no	99.53%	8	99.54%	99.99%
MNIST	Resnet18	change input layer	50	0.01	64	no	SGD	Cross Entropy	0.9	-	no	no	no	no	99.30%	26	99.57%	100.00%
MNIST	Resnet18	change input layer	40	0.01	64	no	SGD	Cross Entropy	0.9	-	yes	yes	yes	no	99.22%	36	99.02%	99.36%
MNIST	VGG16	change input layer	10	0.001	4	no	SGD	Cross Entropy	0.9	-	no	no	no	no	99.38%	10	99.53%	99.79%
MNIST	VGG16	change input layer	50	0.001	64	no	SGD	Cross Entropy	0.9	-	no	no	no	no	99.43%	45	99.52%	99.98%
MNIST	VGG16	change dataset channels	10	0.001	4	no	SGD	Cross Entropy	0.9	-	no	no	no	no	99.30%	10	99.37%	99.90%
MNIST	VGG16	change input layer	30	0.001	64	no	SGD	Cross Entropy	0.9	-	yes	yes	yes	no	98.97%	26	98.90%	98.72%
CIFAR10	GoogLeNet	change input layer	10	0.01	64	no	SGD	Cross Entropy	0.9	-	no	no	no	no	85.26%	10	84.84%	92.98%
CIFAR10	GoogLeNet	change input layer	20	0.009	64	no	SGD	Cross Entropy	0.9	-	no	no	no	no	88.56%	20	87.97%	99.30%
CIFAR10	GoogLeNet	change input layer	30	0.008	64	no	Adam	Cross Entropy	-	-	no	no	no	no	86.34%	30	86.22%	97.87%
CIFAR10	GoogLeNet	change input layer	30	0.008	64	no	SGD	Cross Entropy	0.9	-	yes	no	no	no	89.94%	27	88.49%	99.17%
CIFAR10	GoogLeNet	change input layer	30	0.008	64	no	SGD	Cross Entropy	0.9	-	no	no	no	no	89.78%	28	88.84%	99.98%
CIFAR10	GoogLeNet	change input layer	30	0.006	128	no	SGD	Cross Entropy	0.9	-	yes	no	yes	no	90.14%	27	89.67%	98.28%
CIFAR10	GoogLeNet	change input layer	85	0.006	128	no	SGD	Cross Entropy	0.9	-	yes	yes	yes	no	92.54%	84	92.04%	99.94%
CIFAR10	Resnet18	change input layer	20	0.001	32	no	SGD	Cross Entropy	0.9	1.00E-04	yes	no	no	no	81.78%	20	81.32%	97.09%
CIFAR10	Resnet18	change input layer	30	0.001	32	no	Adam	Cross Entropy	-	-	yes	no	no	no	87.50%	29	85.04%	93.69%
CIFAR10	Resnet18	change input layer	30	0.001	32	no	SGD	Cross Entropy	0.9	1.00E-04	yes	no	no	no	83.66%	29	81.97%	99.17%
CIFAR10	Resnet18	change input layer	32	0.001	64	no	Adam	Cross Entropy	0.9	1.00E-04	yes	no	yes	no	88.52%	32	87.58%	92.57%
CIFAR10	Resnet18	change input layer	40	0.001	64	no	Adam	Cross Entropy	0.9	1.00E-04	yes	no	no	no	87.64%	33	86.25%	96.18%
CIFAR10	Resnet18	change input layer	100	0.001	64	no	Adam	Cross Entropy	0.9	1.00E-04	yes	yes	yes	no	90.02%	82	89.18%	94.00%
CIFAR10	VGG16	change input layer	50	0.01	64	no	SGD	Cross Entropy	0.9	0.005	yes	no	no	no	82.10%	41	77.15%	84.54%
CIFAR10	VGG16	change input layer	50	0.01	64	no	SGD	Cross Entropy	0.9	0.005	no	no	no	no	78.16%	21	76.42%	88.75%
CIFAR10	VGG16	change input layer	69	0.02	64	no	SGD	Cross Entropy	0.9	0.005	yes	no	no	no	76.30%	28	73.42%	73.42%

CIFAR10	VGG16	change input layer	60	0.01	128	no	SGD	Cross Entropy		0.9	0.005	yes	no	yes	no	83.62%	48	82.42%	87.43%
CIFAR10	VGG16	change input layer	60	0.001	128	no	SGD	Cross Entropy		0.9	0.005	yes	no	yes	no	84.04%	60	82.85%	95.52%
CIFAR10	VGG16	change input layer	70	0.009	128	no	SGD	Cross Entropy		0.9	0.005	yes	no	yes	no	84.58%	60	82.59%	88.59%
CIFAR10	VGG16	change input layer	100	0.001	128	no	SGD	Cross Entropy		0.9	0.005	yes	yes	yes	no	85.58%	85	84.19%	95.08%
CIFAR100	GoogLeNet	change input layer	10	0.01	64	no	SGD	Cross Entropy		0.9	-	no	no	no	no	55.74%	8	54.79%	70.39%
CIFAR100	GoogLeNet	change input layer	30	0.001	64	no	Adam	Cross Entropy	-		-	yes	no	yes	no	65.98%	28	64.72%	88.50%
CIFAR100	GoogLeNet	change input layer	30	0.001	64	no	Adam	Cross Entropy	-		-	no	no	no	no	59.54%	12	58.69%	96.68%
CIFAR100	GoogLeNet	change input layer	40	0.001	128	no	Adam	Cross Entropy	-		-	yes	no	yes	no	67.56%	36	66.80%	94.47%
CIFAR100	GoogLeNet	change input layer	40	0.006	128	no	Adam	Cross Entropy	-		-	yes	no	yes	no	67.18%	36	66.96%	89.80%
CIFAR100	GoogLeNet	change input layer	200	0.01	128	yes	Adam	Cross Entropy	-		-	yes	yes	yes	no	71.54%	165	70.77%	99.82%
CIFAR100	GoogLeNet	change input layer	200	0.007	128	yes	Adam	Cross Entropy	-		-	yes	yes	yes	no	70.26%	198	70.80%	99.91%
CIFAR100	Resnet18	change input layer	20	0.01	32	no	SGD	Cross Entropy		0.9	1.00E-04	yes	no	no	no	62.36%	20	62.07%	96.50%
CIFAR100	Resnet18	change input layer	30	0.001	64	no	Adam	Cross Entropy	-		-	yes	no	yes	yes	64.14%	29	62.71%	81.10%
CIFAR100	Resnet18	change input layer	30	0.01	64	no	SGD	Cross Entropy		0.9	1.00E-04	yes	no	yes	yes	65.60%	29	65.83%	91.71%
CIFAR100	Resnet18	change input layer	30	0.001	64	no	SGD	Cross Entropy		0.9	1.00E-04	yes	no	yes	yes	53.90%	29	54.47%	60.79%
CIFAR100	Resnet18	change input layer	30	0.009	64	no	SGD	Cross Entropy		0.9	1.00E-04	yes	no	no	no	65.64%	28	64.37%	99.90%
CIFAR100	Resnet18	change input layer	40	0.06	64	no	SGD	Cross Entropy		0.9	1.00E-04	yes	no	yes	yes	64.88%	38	62.72%	90.38%
CIFAR100	Resnet18	change input layer	100	0.01	64	yes	SGD	Cross Entropy		0.9	1.00E-04	yes	yes	yes	yes	72.08%	95	71.82%	99.88%
CIFAR100	Resnet18	change input layer	200	0.007	64	yes	SGD	Cross Entropy		0.9	1.00E-04	yes	yes	yes	yes	72.84%	103	72.24%	99.98%
CIFAR100	VGG16	change input layer	50	0.006	64	no	SGD	Cross Entropy		0.9	0.005	yes	no	no	no	46.04%	20	42.68%	73.65%
CIFAR100	VGG16	change input layer	50	0.006	64	no	SGD	Cross Entropy		0.9	0.005	no	no	no	no	39.52%	10	31.80%	82.50%
CIFAR100	VGG16	change input layer	70	0.0001	64	no	SGD	Cross Entropy		0.9	0.005	yes	no	yes	no	44.24%	70	43.73%	46.25%
CIFAR100	VGG16	change input layer	70	0.004	64	no	SGD	Cross Entropy		0.9	0.005	yes	no	no	no	36.42%	16	33.88%	79.89%
CIFAR100	VGG16	change input layer	50	0.01	128	no	SGD	Cross Entropy		0.9	0.005	yes	no	yes	no	47.02%	47	46.36%	60.94%
CIFAR100	VGG16	change input layer	50	0.004	128	no	SGD	Cross Entropy		0.9	0.005	no	no	no	no	41.42%	12	35.66%	99.36%
CIFAR100	VGG16	change input layer	70	0.001	128	no	SGD	Cross Entropy		0.9	0.005	yes	no	yes	no	53.94%	67	53.73%	91.80%
CIFAR100	VGG16	change input layer	70	0.007	128	no	SGD	Cross Entropy		0.9	0.005	yes	no	no	no	45.24%	16	43.52%	84.37%
CIFAR100	VGG16	change input layer	200	0.007	128	yes	SGD	Cross Entropy		0.9	0.005	yes	yes	yes	no	66.98%	186	66.99%	99.25%