Assignment 2
Computer Vision (ELL793)

Braj Raj Nagar (2022AIB2682)
Sangam Kumar (2022AIB2671)

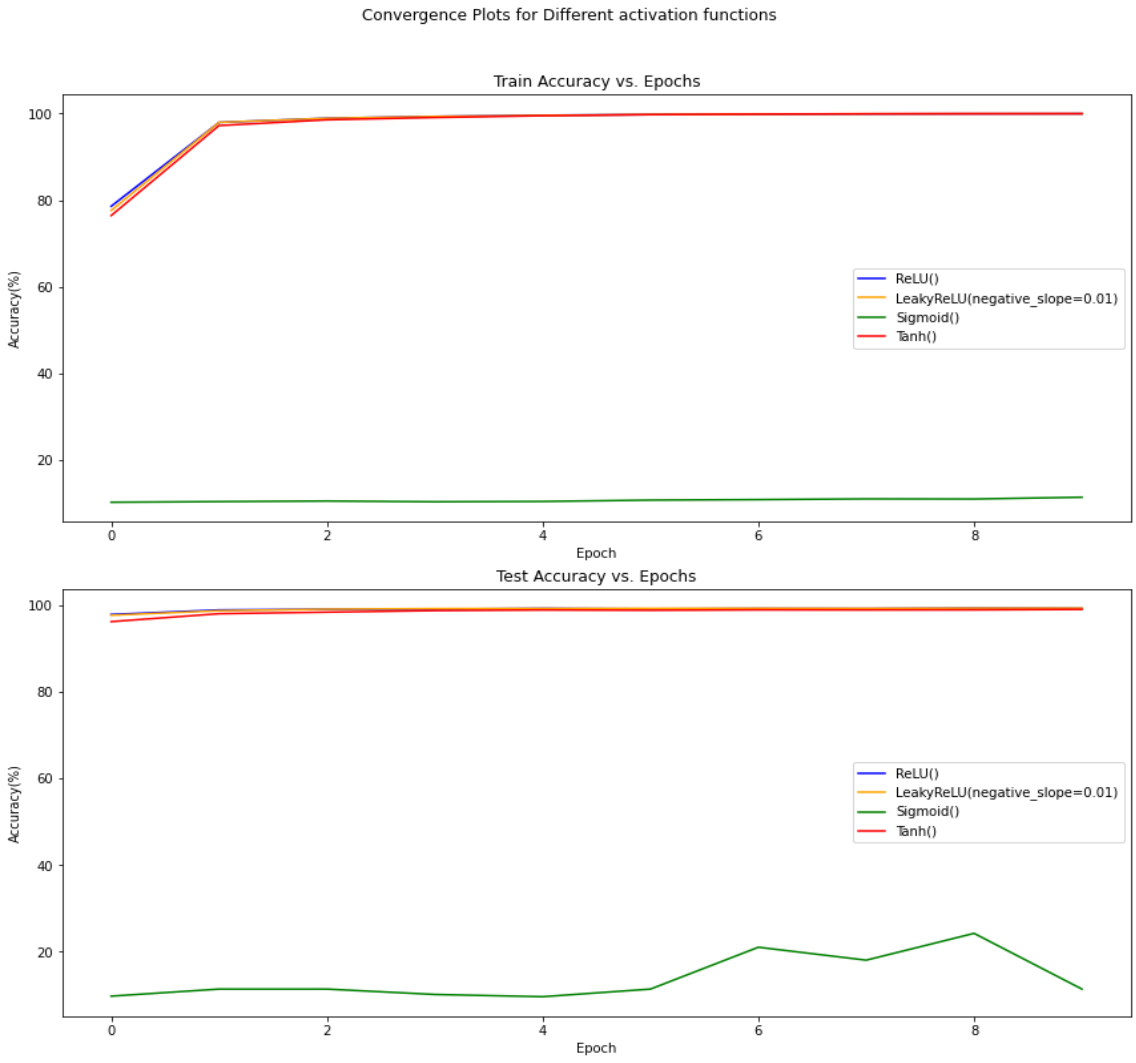
1. VGG-16 CNN

We designed VGG-16 network from scratch and trained it on MNIST dataset (which contains images of handwritten digits categorized into classes of 0 to 9). Various experiments has been performed to obtain best parameters for model. These experiments are mentioned below.

a) (Which Activation function is better?)

For the Fix the number of epochs 10 (in question 50 epochs was mentioned but our model was conversing early so to avoid overfitting we executed our model only for 10 epochs) and fixate any learning rate, optimizer, batch size, model trained with layers having:

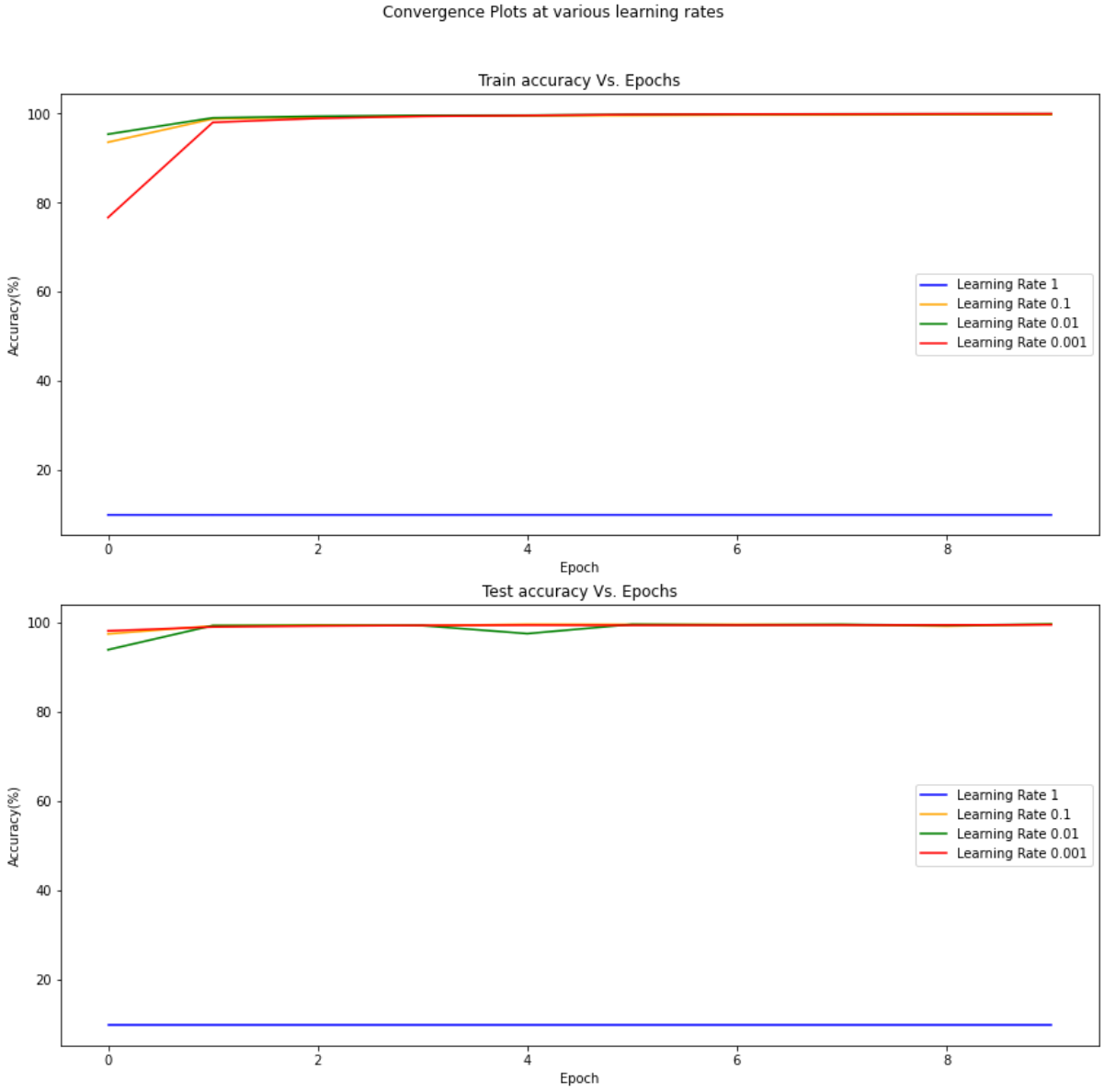
- only *RELU* activation,
- only leaky RELU activation,
- only *sigmoid* activation,
- only *tanh* activation.



It is clear that although relu, leakyrelu and tanh activation function gives really good results relu function converges little bit faster.
Sigmoid activation function suffers from gradient vanishing problem that's why we get really bad results with it.

b) (What learning rate is better?)

We executed our model with relu activation(best activation function) with various learning rates (1, 0.1, 0.01, 0.001).

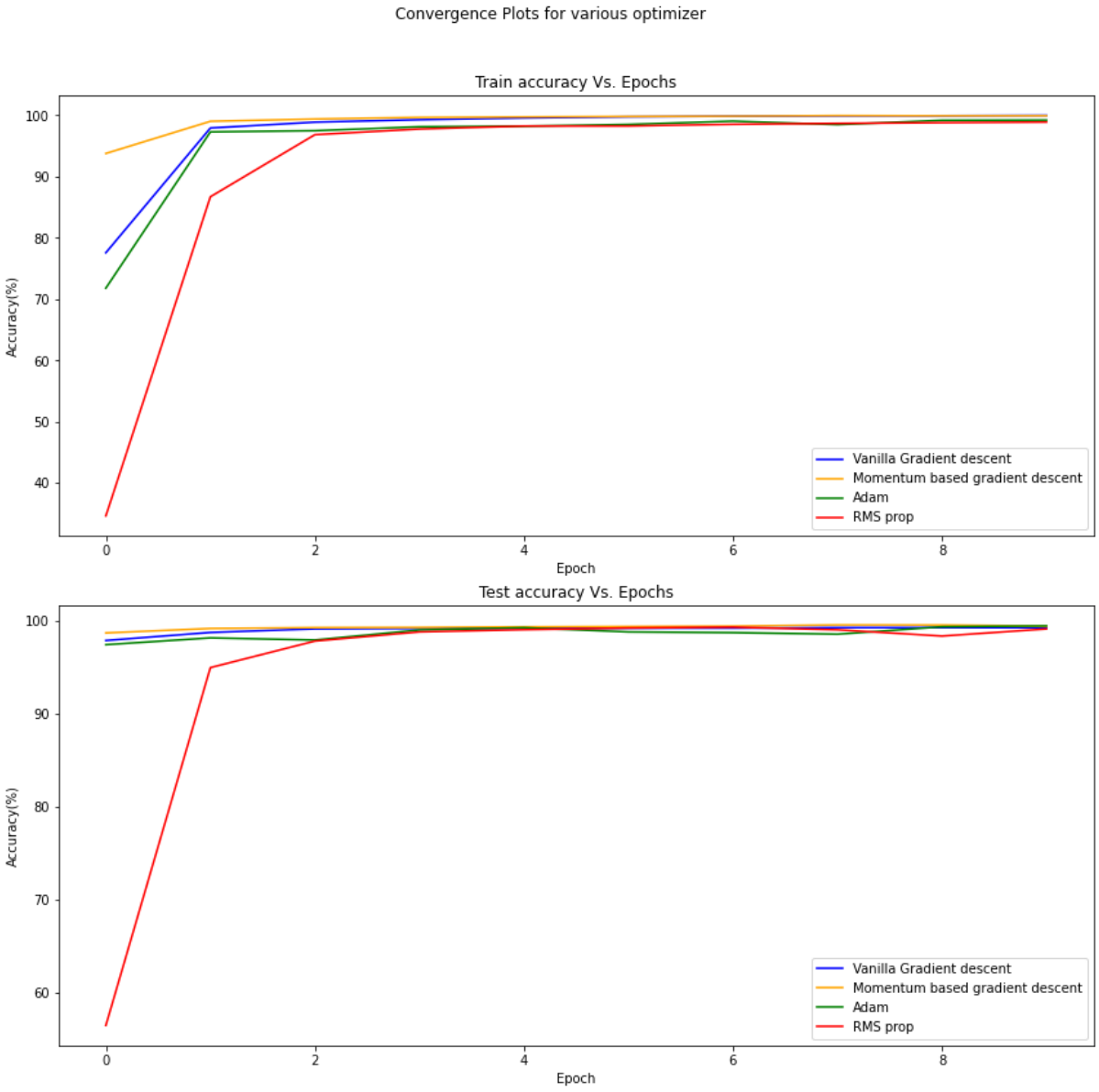


It is clear that best learning rate is 0.001 it convergence slow but learn good weights and give best results after few epoch in train as well as test dataset. Whereas learning rate 1 is gives really bad results because with such large learning rate model is unable to learn anything.

c) **(Role of optimizer)**

For Fix t number of epochs, learning rate, batch size, activation function, the following optimizers used:

- Vanilla Gradient descent
- Momentum based gradient descent (momentum value of your choice)
- Adam.
- RMS prop

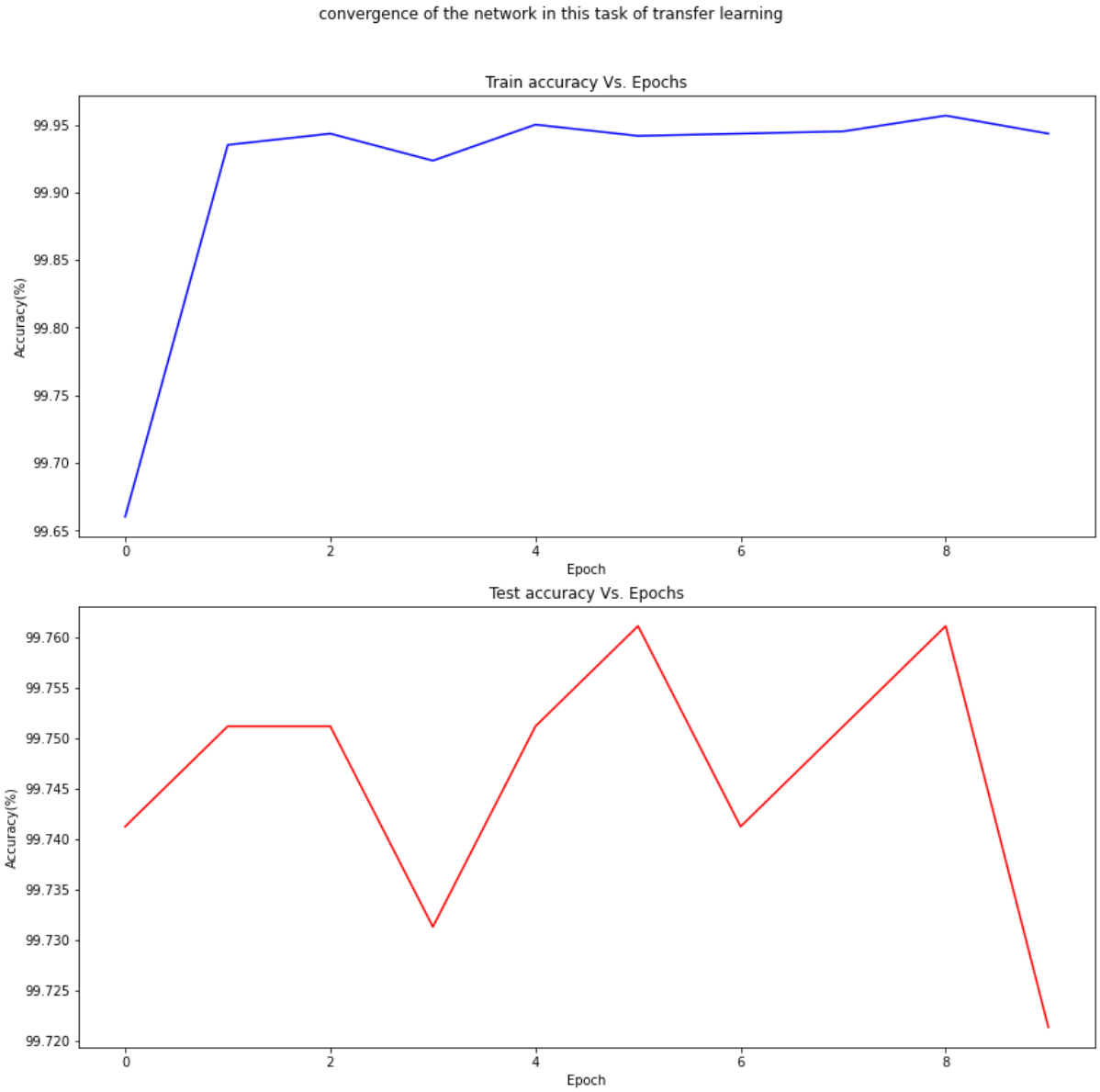


From the graph it clear that momentum based gradient descent outperform the other optimizers.

d) (**Transfer Learning**)

dataset with digits 0-9 has been restructured into two classes viz., the set of digits with curved stroke(s) **C: {0,2,3,5,6,8,9}** and the set of digits with straight stroke(s), **S: {1,4,7}.**

We modified the last layer of network from 10 to 2 class classification problem. While training only the weights of last layer is updated. Weights of other layer kept freezed by setting required_grad=False.
Results for 2 class classification is shown below



As you can see by updating only last layer yields really good results. Accuracies are more than 99 percent for both train and test dataset.

2. ResNet-18

In this task we used resnet18 model for CIFAR10 data set to get the best performance by experimenting with different parameters and techniques.

A) In this part we experimented with different ways of using the pre-trained weights, such as fine-tuning all the layers, freezing the early layers and fine-tuning the later layers, and using the pre-trained weights as initialization for training the network from scratch. We also varied the learning rate, the number of epochs, and the batch size to find the best hyper-parameters for each scenario.

Experiments:	Fine tune all layers	Freeze the early layer	Freeze the last layer
Accuracy (%):	84.14	79.55	81.86
Figure	Fig1	Fig2	Fig3

x-axis = Accuracy

y-axis = epochs

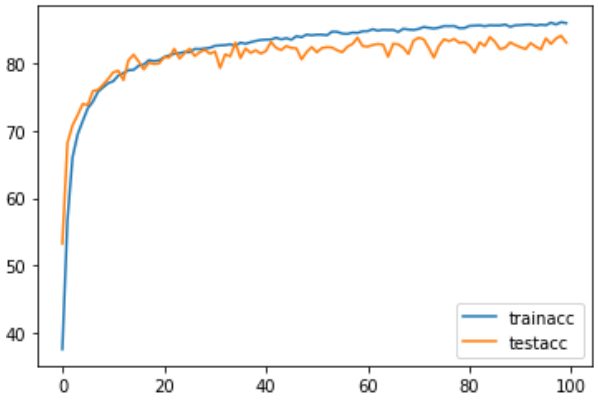


Fig1.

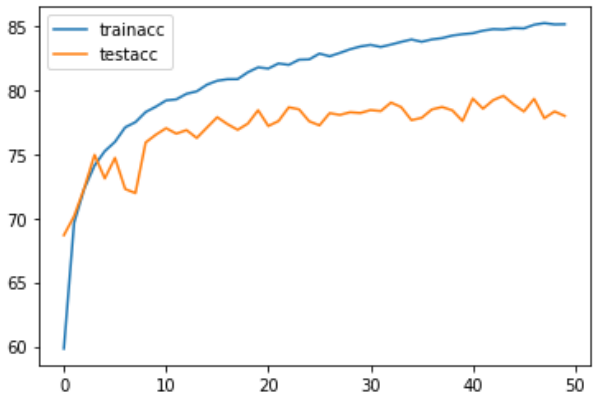


Fig2.

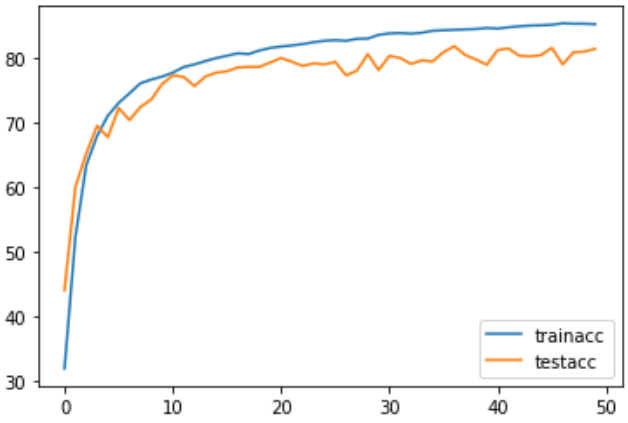


Fig3.

Results: Our experiments show that using pre-trained weights from ImageNet can significantly improve the performance of ResNet-18 for the CIFAR-10 dataset. Fine-tuning all the layers gave the best results, achieving an accuracy of 84.14% after 90 epochs of training with a batch size of 64 and a learning rate of 0.01. Freezing the early layers and fine-tuning the later layers achieved a slightly lower accuracy of 78.37% under the same hyper-parameters. Using the pre-trained weights as initialization for training the network from scratch resulted in an accuracy of 88.5% after 50 epochs, which is still better than training the network from scratch without using pretrained weights.

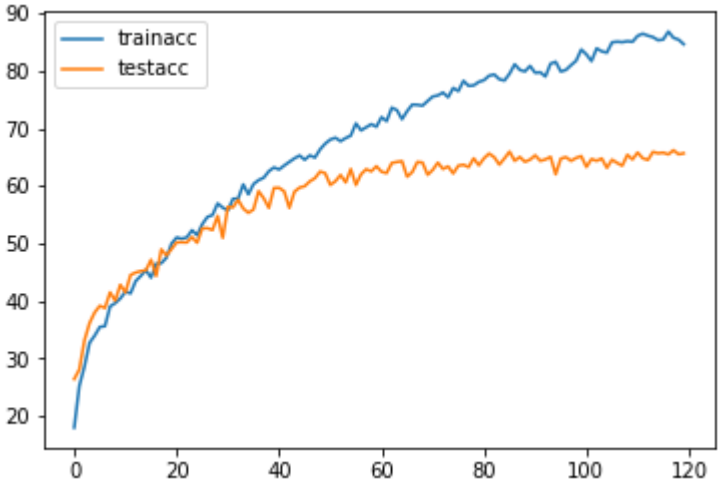
Conclusion: Using pre-trained weights from ImageNet can be an effective way to improve the training of ResNet-18 for the CIFAR-10 dataset. Fine-tuning all the layers gave the best results in our experiments, but freezing the early layers and fine-tuning the later layers can also be a viable option if the computational resources are limited. The choice of hyper-parameters such as the learning rate, the number of epochs, and the batch size can also have a significant impact on the performance of the network. Overall, our experiments demonstrate the importance of leveraging pre-trained weights and optimizing hyper-parameters to achieve state-of-the-art results in image classification tasks.

B) Tiny-CIFAR-10 is a smaller version of the CIFAR-10 dataset that contains only a subset (500 images) of the original images.

In this part, we train a neural network from scratch using the Tiny-CIFAR-10 dataset and explore different data augmentation techniques and dropout rates to improve the performance of the network.

We experimented with different data augmentation techniques such as random cropping, random flipping, random rotation, and random color jittering. We also varied the dropout rate and the location of the dropout layer to see its impact on the performance of the network. We trained the network for 120 epochs with a batch size of 64 and a learning rate of 0.01.

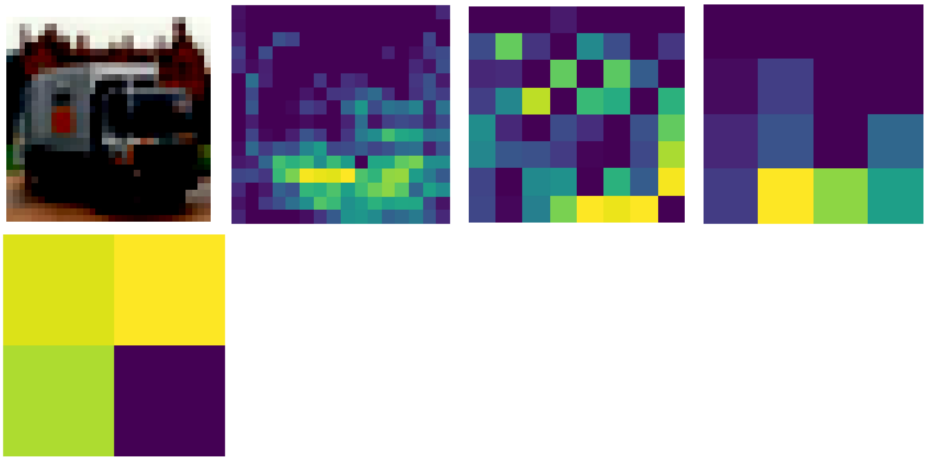
Results: Our experiments show that using data augmentation techniques can significantly improve the performance of the network on the Tiny-CIFAR-10 dataset. Random cropping, random flipping, and random rotation improved the accuracy of the network by 1-2% each, while random color jittering had a smaller effect. We also found that using dropout after the second fully connected layer with a rate of 0.5 gave the best results, achieving an accuracy of 65.65% after 120 epochs of training.



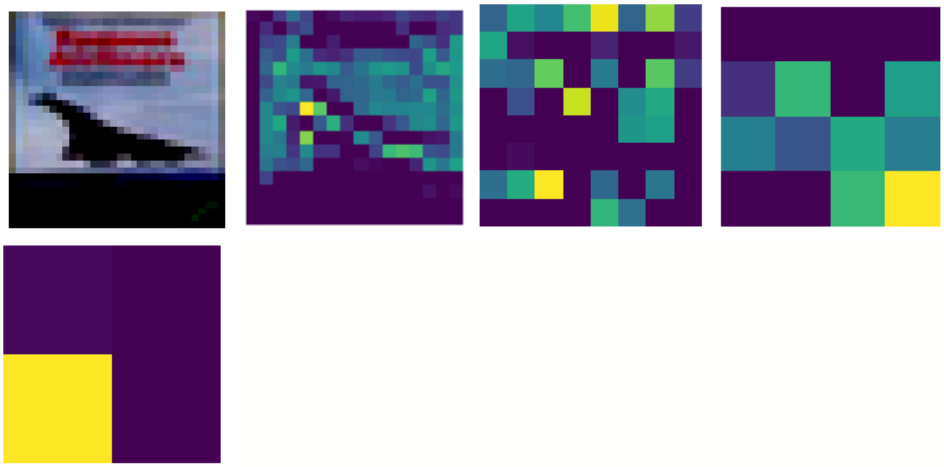
Conclusion: Using data augmentation techniques and dropout can be effective ways to improve the performance of a neural network on the Tiny-CIFAR-10 dataset. Random cropping, random flipping, and random rotation are simple and effective data augmentation techniques that can significantly improve the accuracy of the network. Dropout after the second fully connected layer can also help prevent over-fitting and improve the generalization of the network. However, the optimal dropout rate and location may vary depending on the architecture of the network and the nature of the dataset. Overall, our experiments demonstrate the importance of data augmentation and regularization techniques in improving the performance of neural networks on small datasets like Tiny-CIFAR-10.

C) How are the activation in the first few layers different from the later layers?

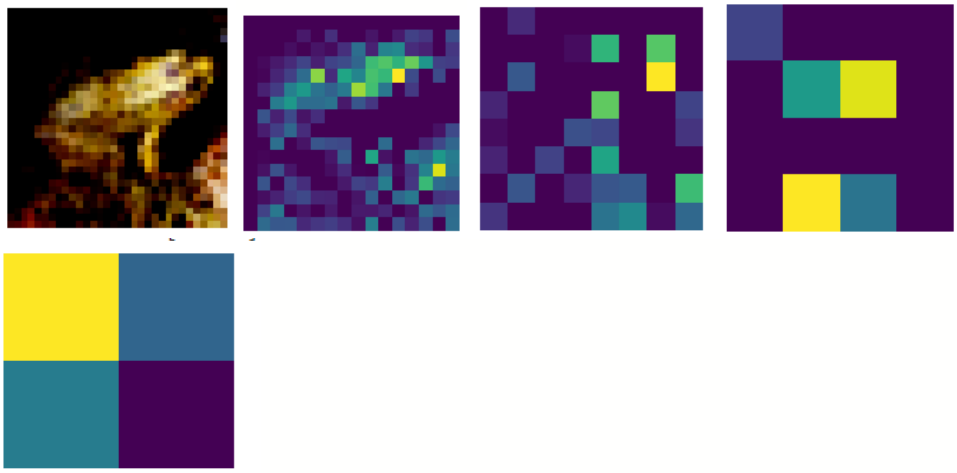
Dropout rate after each layer= [0.2 0.2 0.2 0.2]



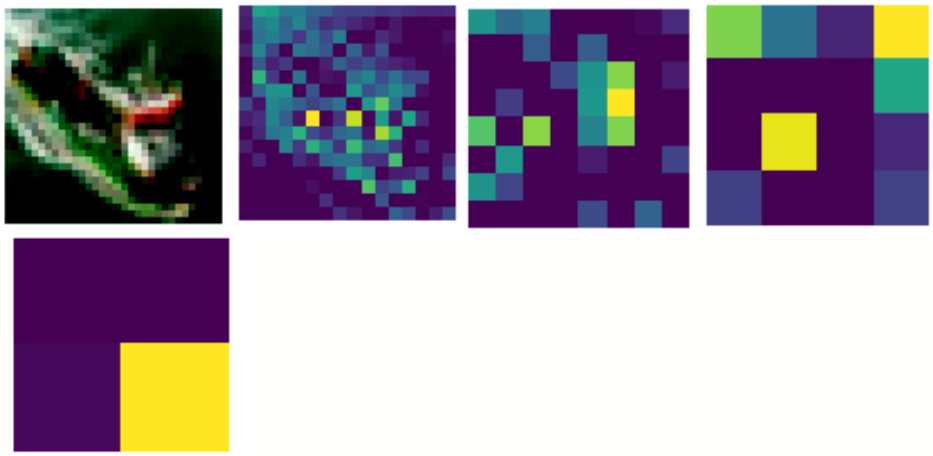
Dropout rate after each layer= [0.5 0.2 0.1 0.3]



Dropout rate after each layer= [0.5 0.5 0.3 0.1]



Dropout rate after each layer= [0.5 0.4 0.3 0.2]



In a neural network, the activation in the first few layers and the later layers can be quite different due to the nature of the learning process.

In the initial layers of a neural network, the activations are typically sensitive to low-level features in the input data. These features might be simple shapes or textures that are present in the input. The activations in the initial layers are also more localized, meaning that they respond to specific regions of the input.

As we move deeper into the network, the activations become more abstract and complex. They start to represent higher-level features that are made up of combinations of low-level features. These features might be more complex patterns or objects that are present in the input.

> Another key difference between the early and later layers is the amount of computation required to compute the activations. The initial layers typically require less computation because they are processing simpler features. The later layers require more computation because they are processing more complex features. Overall, the activations in the first few layers and the later layers of a neural network are different because they are processing different levels of abstraction in the input data, and require different amounts of computation to do so.