COMPUTER VISION LAB 3 REPORT

QUESTION 1: IMAGE REPRESENTATION

I have loaded an image from the kaggle image dataset to test out image representation. From the image output in the notebook, you can see that my image has a shape of **(480, 640, 3)** which means 480 pixel rows on the Y axis, 640 pixel columns along the x axis, and 3 channels which means it is an RGB image.

I have plotted the images loaded using PIL library and openCV, the key difference between the two is that OpenCV's imread load images in **BGR** format as evident in the plotted sup-graphs.

To compare color image and grayscale image, I am going to use openCV's `cvtcolor` method and plot it also as a subplot. As you can see from the shape below, it is an image of **(480, 640)**, here you don't include the channel since it is one channel

In the final print statement of this segment, I have printed out the first 10 rows and columns of the image just to show how pixels are arranged in the array.

QUESTION 2: TRAINING THE NEURAL NETWORK

1. **Write a brief comment on the dataset class FMNISTDataset** - It has 60, 000 images that are 28 by 28 in size which can be categorized into 10 categories.

2. **Why do we shuffle data in the Dataloader** - We shuffle the training data in the dataloader so as to increase 'hypothetically' the size of the training dataset and avoid overfitting which occurs when the model sees the same type of data over and over again and thus improves generalization

3. **When we have batch size of 32, what is the shape of the input to the Neural network and**
the shape of the output? - **32** by **784** since the images are already flattened when we were loading the data

4. **How many parameters (weights & biases) does the given model have?** I run the `summary(model, (1,784))` below and got: **795,010** parameters

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1             [-1, 1, 1000]         785,000
              ReLU-2             [-1, 1, 1000]               0
            Linear-3               [-1, 1, 10]          10,010
================================================================
Total params: 795,010
Trainable params: 795,010
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 3.03
Estimated Total Size (MB): 3.05
----------------------------------------------------------------
```

5. **What is a loss function? Why do we use cross-entropy loss, and what other losses are there?** - A loss function is used to measure how well the model's predictions match the true values; it quantifies this difference. Here we use cross-entropy loss because we are tackling a multi-class classification problem . Other types of loss functions include

- Mean squared error
- Mean absolute error
- Huber loss
- Triplet loss

6. **What is the role of the optimizer? Give a list of optimizers you could use in Deep learning** - Optimizer is used to calculate and step the gradients, it propagates the errors back to the weights and biases, calculate their respective gradients and update the weights iteratively until a desired loss that can be tolerated is achieved.

- Adam
- SGD
- AdamW
- Nestrov Accelerated Gradient
- Adagrad
- RMSprop

7. **Why do we need the learning rate parameter?** - We need the learning rate to accelerate/step the convergence of our weights (if carefully chosen), it determines how large or small the weights updates will be during training.

8.**Why do we do batching? What is the difference between mini-batch and multiple-batches** - it speeds up training, as the model looks at the data multiple samples at a time which means we utilize computing resources wisely. Mini-Batch refers to the individual subset of the dataset used for one weight update. whereas multiple batches refers to the

collection of mini-batches used to complete an entire pass (epoch) through the dataset.

9. **How often do we do gradient updates if our dataset has 60,000 data points and uses a batch size of 32? What about using a batch size of 1,000?**

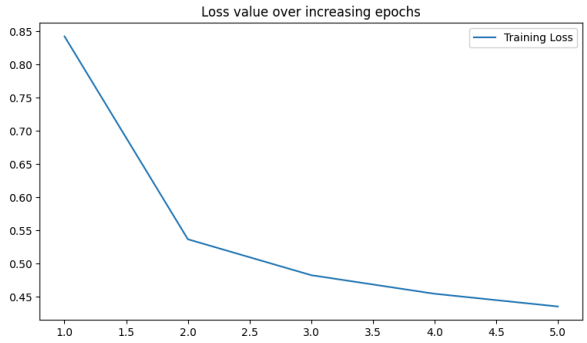
For batch size 32, the model will update its weights after every **32** samples, whereas for a batch size of 1,000 it will update after every **1000** samples.

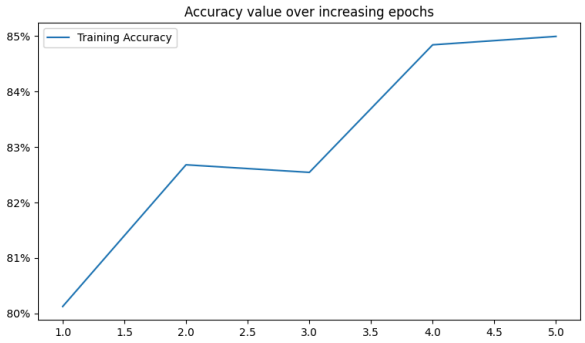
QUESTION 3: SCALING

Your dataset is initially made of tensors with numbers between 0 -255; you can scale them (as is done in the notebook) by dividing by 255 so that it's between 0 and 1.

- **What is the difference in the results you see?**
  - The scaled values training accuracy improve quickly in fewer epochs compared to unscaled values, its loss also continuously decreases unlike the raw pixels which plateaus after some point

This is the graph of scaled values



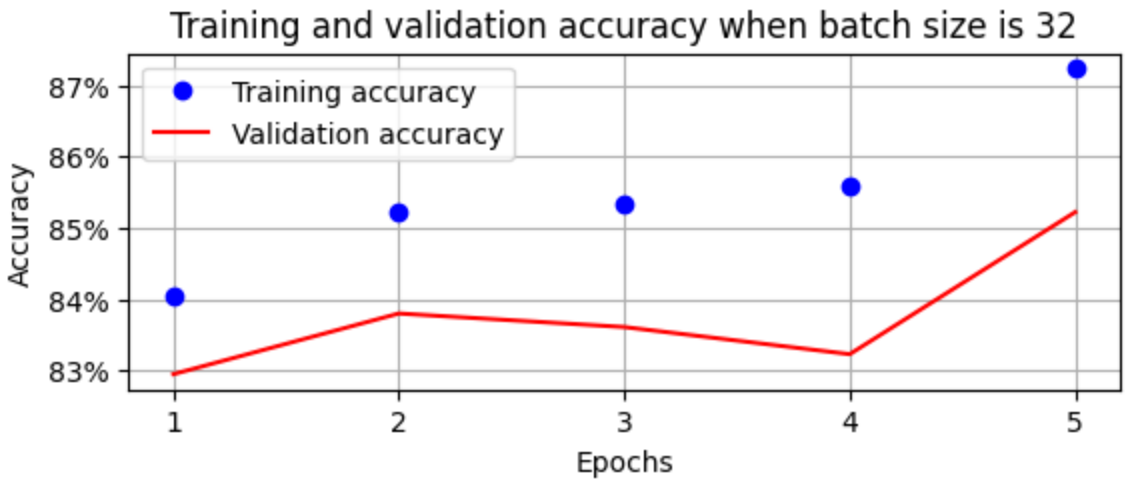


This is the graph of unscaled values

- **Can you explain why we see that difference?**
  - One could be the fact that smaller values of pixels generate smaller gradients hence easy to optimize them. Large weights make the updating and convergence slow because they are large even if the effect of learning rate on them will be minimal hence the training will be unstable.

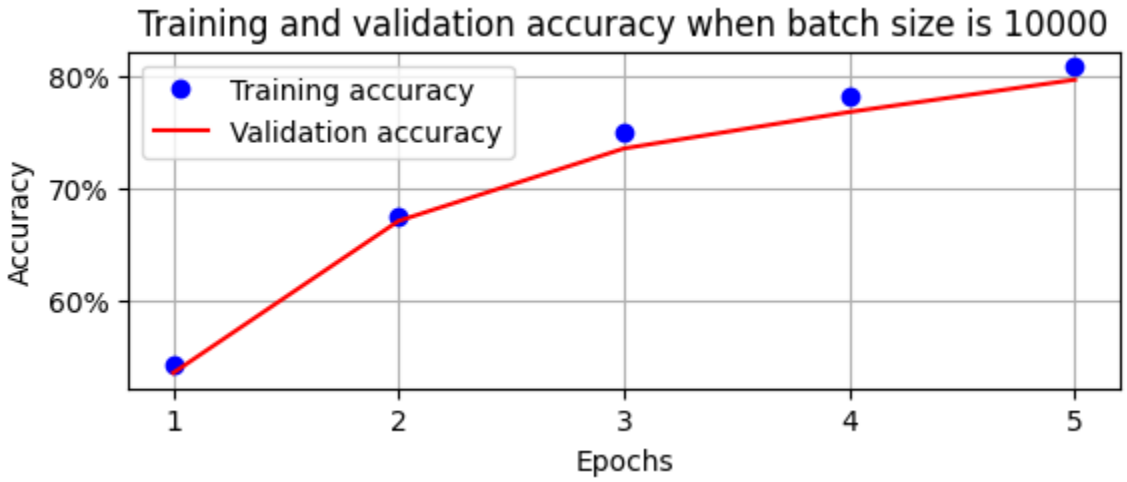
QUESTION 4: DIFFERENT BATCH SIZE

Batch size 32



From the plotted graph above, a batch size of 32 has an accuracy of 85% as the fifth epoch, while there is some gap between the training accuracy and validation accuracy where the training accuracy is still higher than the validation accuracy

Batch size of 1000



From the plotted graphs above, a batch size of 1000 has an accuracy of 80% as of the fifth epoch while the training accuracy and validation accuracy is close to each other

Explanation

A batch size of 32 has a higher accuracy than a batch size of 1000, because weights are updated more frequently than a batch size of 1000 per epoch, this means that weights have enough time to update and converge to the optimal solutions. This though might lead to overfitting as shown in the graph where the training accuracy keeps improving whereas the validation accuracy is always lower than the training accuracy. This means that our model keeps getting better/ fitting better into training data but not improving well in unseen data. Poor

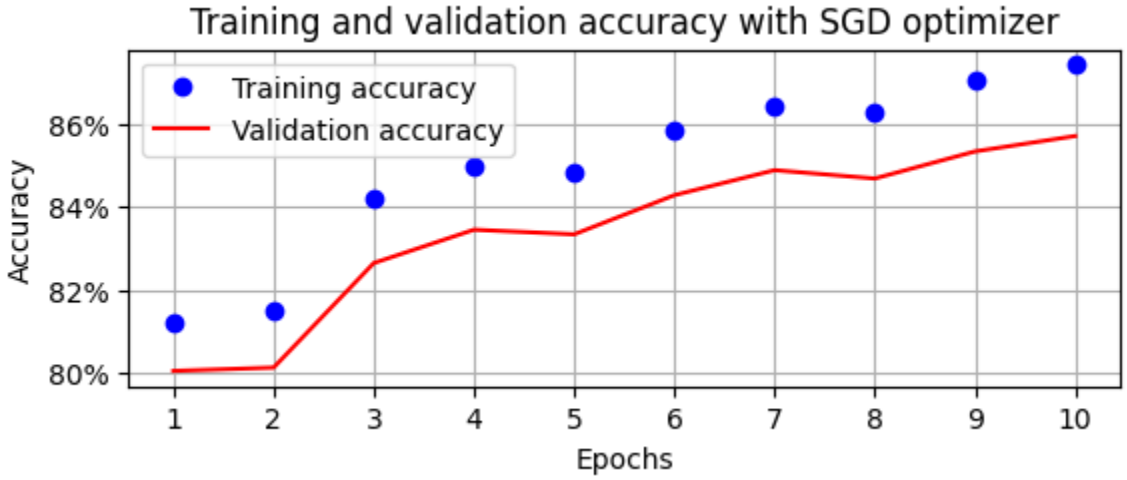
generalization.

The inference from the losses graph is that the loss graph for a batch size of 32 maybe be aggressive and high in the first few epochs because we are rapidly updating weights whereas in a batch size of 1000 it is smoother because we do not update the weights more rapidly as is the case on batch size of 32

QUESTION 5: OPTIMIZERS

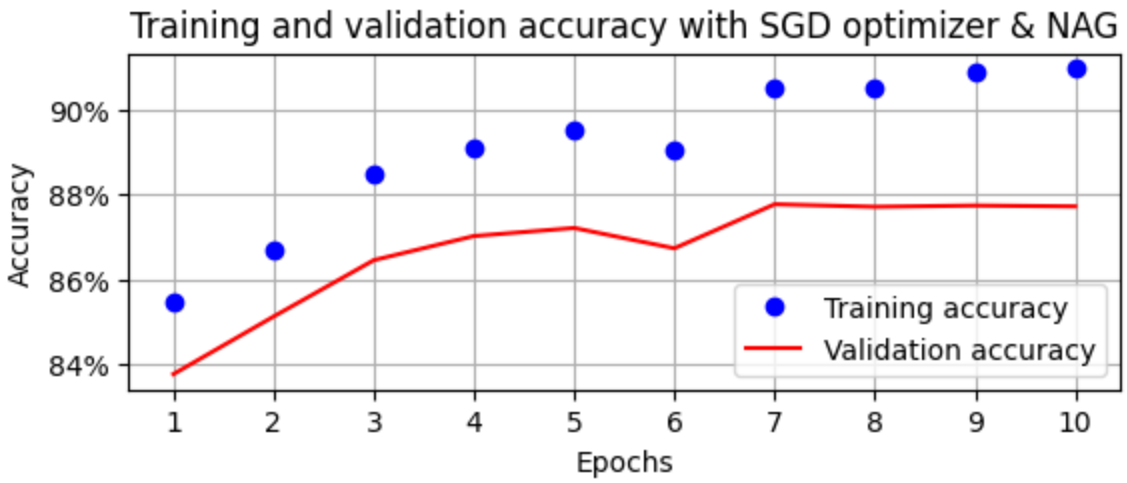
From the optimizers you listed above. Describe 3 of them; how do they work? How do you expect them to affect your model's performance in training? Try those optimizers & show the results, and explain them

SGD
It is a variant of Gradient Descent that updates model parameters using a single data sample (or a small batch) per iteration rather than computing the gradient over the entire dataset.



From the above graph, you can see that plain SGD has a maximum accuracy of about 85%

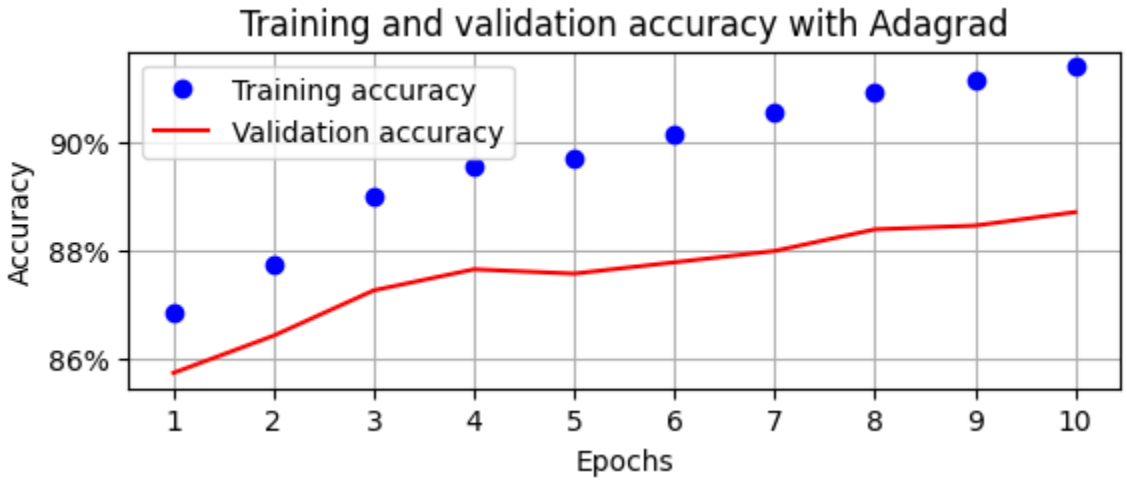
Nesterov Accelerated Gradient
- Nesterov Accelerated Gradient (NAG) improves momentum by looking ahead at the predicted next step before computing the gradient, leading to faster convergence and smoother updates. It helps avoid overshooting and accelerates training.



When we introduce Nesterov to SGD we see an improvement in the validation accuracy to about 88%, this is because NAG looks ahead before computing gradients therefore quicker convergence is achieved.

Adagrad

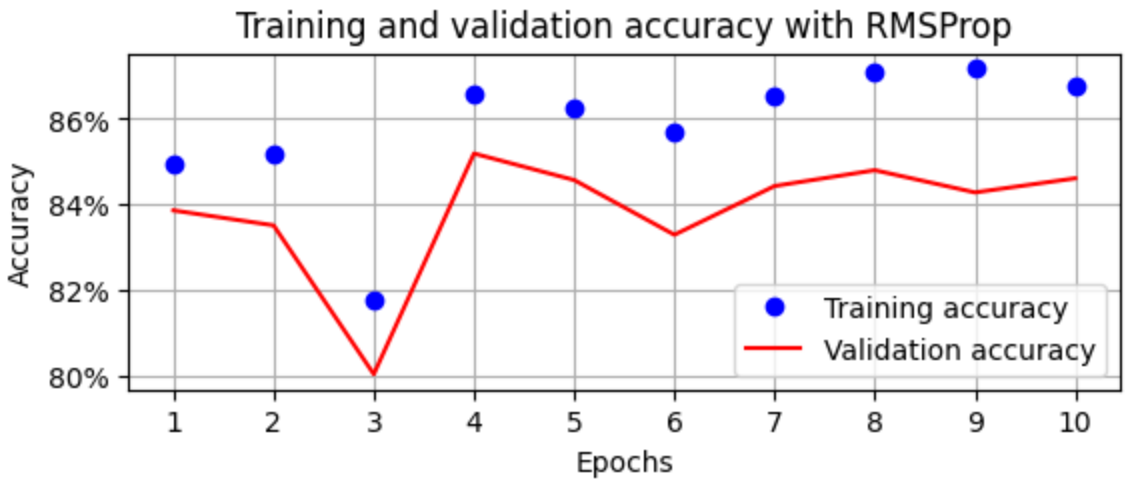
- **Adagrad** adjusts the learning rate for each parameter based on its past gradients, using larger updates for infrequent features and smaller ones for frequent features. It helps with sparse data but can slow down over time as the learning rate decreases.



The accuracy improved to 89% when using Adagrad as our optimizer. This is because Adagrad adapts learning rate to each parameter - the weights and biases, and this enables our weights to converge quickly and thus there is an improvement in accuracy.

RMSprop

- RMSprop (Root Mean Square Propagation) is an adaptive learning rate optimization algorithm that modifies Adagrad to reduce its drastically decreasing learning rate. It scales the learning rate based on a moving average of squared gradients, helping stabilize training.



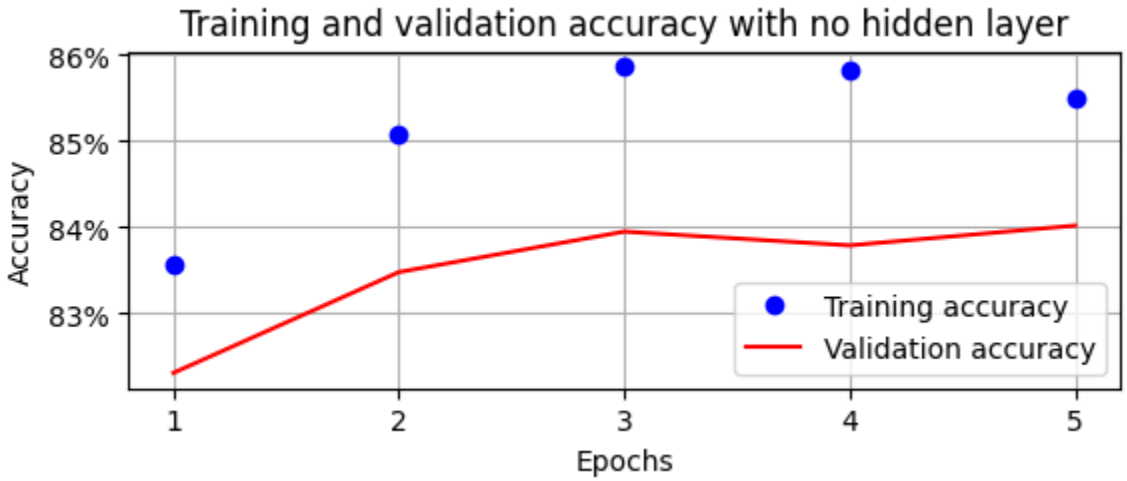
The accuracy of RMS is really low with a maximum of 85% at epoch 4 and then stochastically changes over other epochs. RMSprop scales learning rates based on recent gradients, which helps in some cases but can cause instability if the learning rate is too high.

QUESTION 6: MODEL DEPTH & SIZE

Explore models of different depths that change the number of layers and the number of hidden layers. Run at least four different architectures. What is the impact on the training time and performance of the model?

- I tested models with 0 to 3 hidden layers and found that adding layers improved performance but increased training time. The only significant change I noticed was the difference between the models with zero hidden layers and the others which had a relatively low accuracy of 84% cap. This ideally shows that having hidden layers could improve performance, maybe with more epochs I could achieve more accuracy.

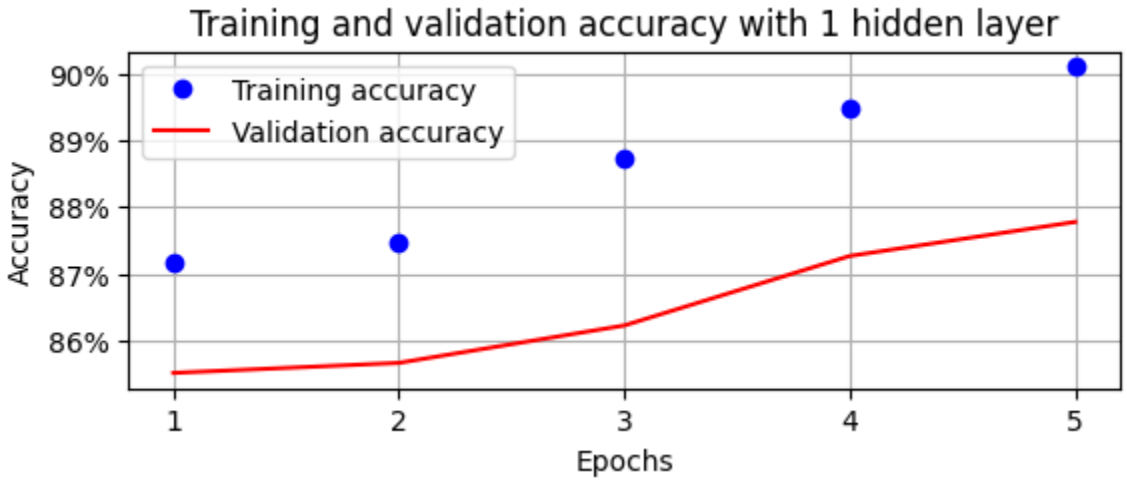
0 HIDDEN LAYERS



The maximum accuracy in a neural network with 0 hidden layers achieved a maximum

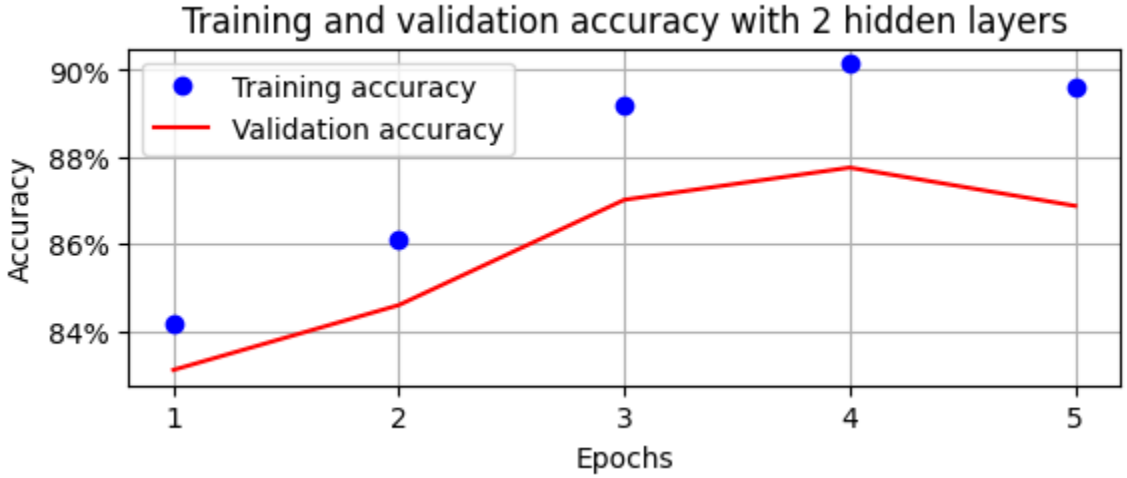
accuracy of 84%

1 hidden layer



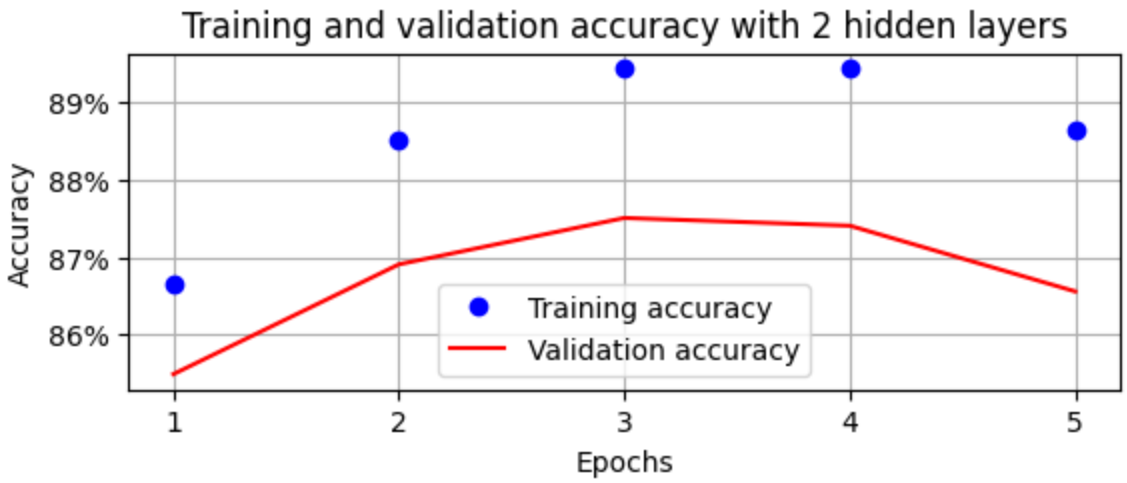
Here, there is an improvement in the validation Accuracy where we achieve an accuracy of around 88%

2 Hidden layers



We achieve an accuracy close to the one with one hidden layer at epoch 4 but slightly decreased at epoch 5.

3 Hidden layers



The validation accuracy gets worse as the number of hidden layers increases, this is usually not the case when you have a huge amount of data. This might be happening because our model is overfitting to data. Also the training accuracy is also reducing, that means we might be having exploding gradients or vanishing gradients as our network keeps getting deeper.

QUESTION 7: BATCH NORMALIZATION

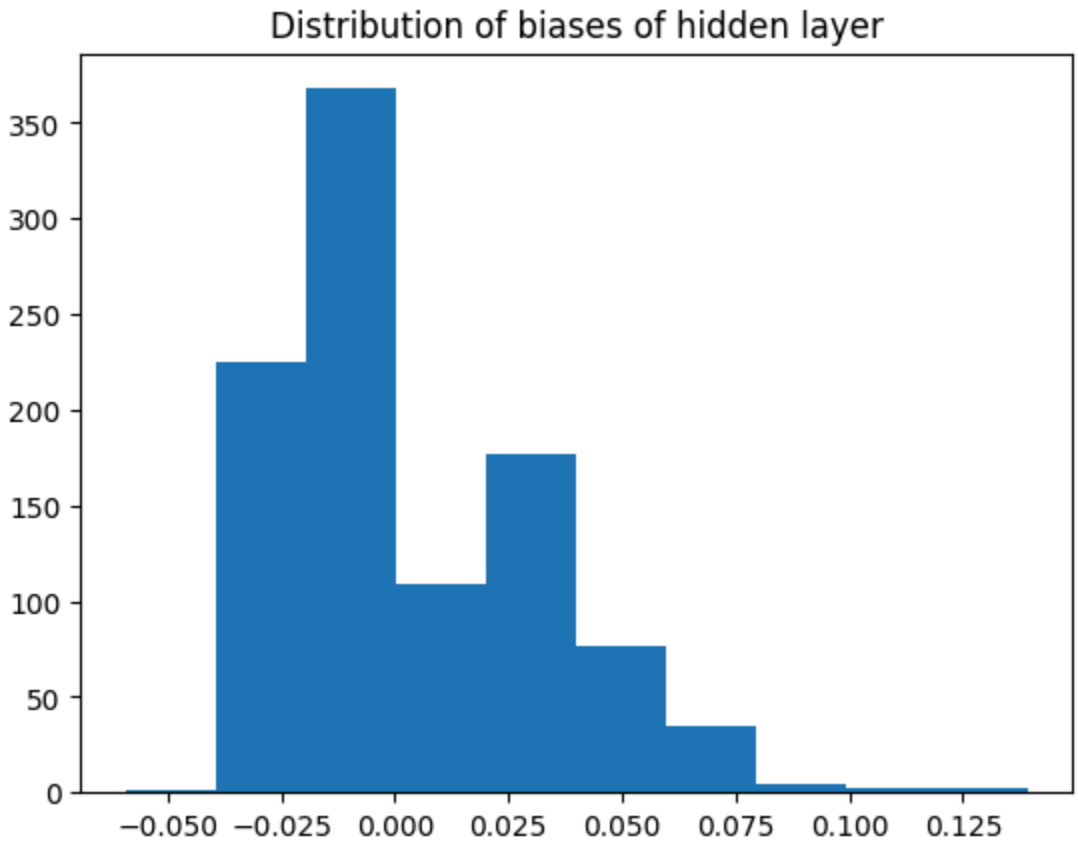
1. **What is batch normalization**

- Batch Normalization (BatchNorm) normalizes layer inputs in mini-batches during training by subtracting the batch mean and dividing by the batch variance, then scaling and shifting with learnable parameters. It speeds up training, improves stability, and acts as a regularizer, but performance can degrade with very small batch sizes.

3. Using this tutorial, explore the impact of batch normalization on your training, plot and explain the histograms of the weight values at different layers of your model.

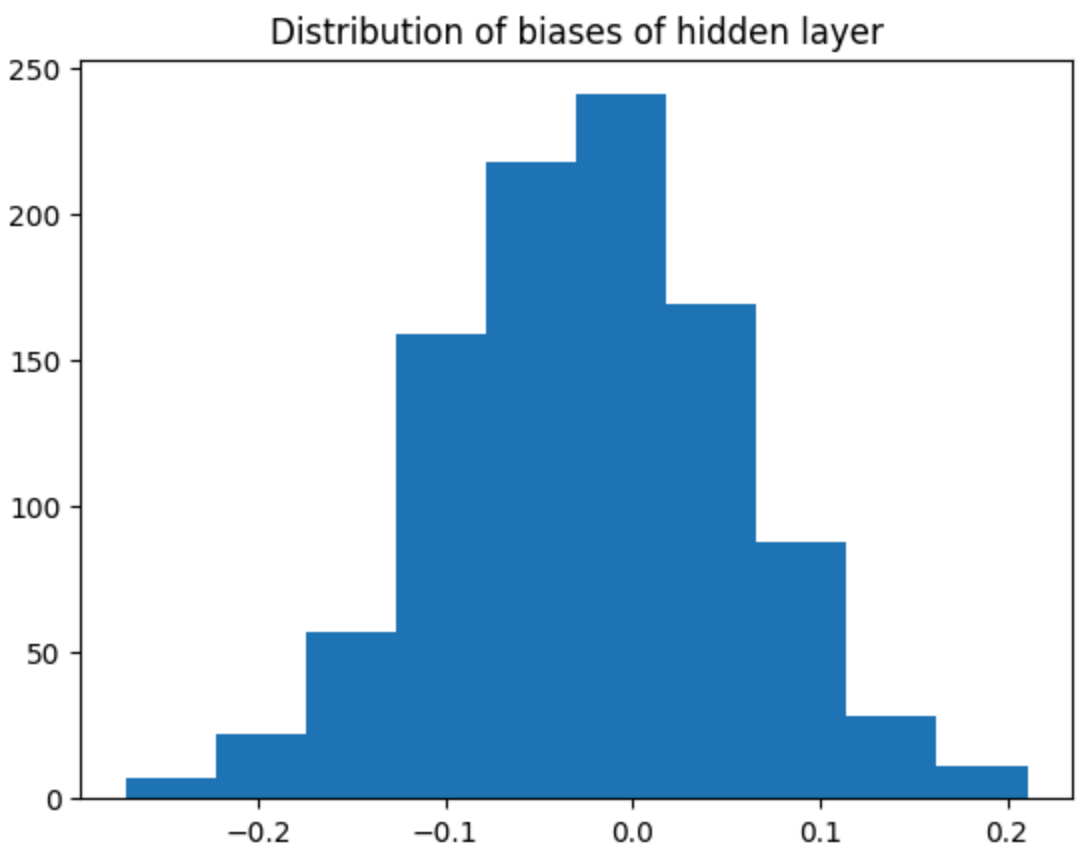
Histogram without Batch Normalization

Without BatchNorm, the histograms show that the weights have a much wider distribution, with some layers exhibiting skewed or collapsed distributions, especially as training progresses. This suggests issues like exploding or vanishing gradients, where the model struggles with unstable weight updates.



Histogram after Batch Normalization

After applying BatchNorm, the histograms become more centered around zero and display a more controlled variance. This indicates that BatchNorm is stabilizing the learning process by normalizing the inputs to each layer. The histograms also remain more consistent across epochs, demonstrating improved weight distribution, which aids in faster and more stable convergence. Early layers show better normalization, middle layers exhibit less variance, and final layers are more balanced, reflecting BatchNorm's effectiveness in controlling weight values and promoting smoother updates throughout training.


Distribution of biases of hidden layer
250
200
150
100
50
0
−0.2
−0.1
0.0
0.1
0.2