HOMEWORK 1 (CPSC 8430 – DEEP LEARNING)

SHRAVANI KONDA

HW1 1-1 Simulate a Function:

Describe the models you use, including the number of parameters (at least two models) and the function you use.

https://github.com/shravanik31/Deep-Learning/blob/main/Shravani_Konda_HW1_1_1_Simulate_Function.ipynb

Target Function 1:

The single input, single output non-linear function that is used is **sin(x)+cos(x)**. Training data (x_train) is generated with uniform distribution between -2 and 2, while validation data (x_val) is linearly spaced.

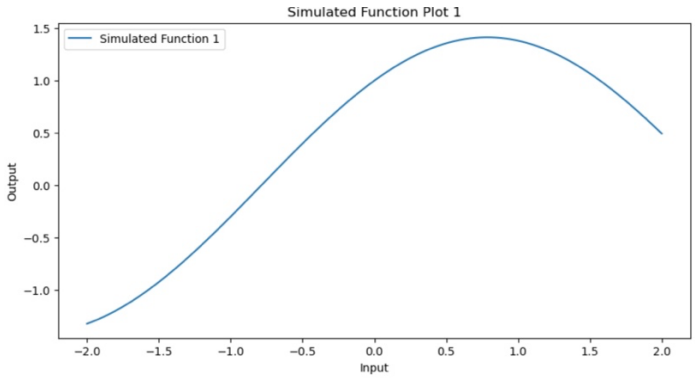
```
def simulate_function_1(x):
    return np.sin(x)+np.cos(x)

x_train = np.linspace(-2, 2, 5000).reshape(5000,1)
y_train = simulate_function_1(x_train)

x_val = np.linspace(-2, 2, 1000).reshape(1000,1)
y_val = simulate_function_1(x_val)

x_train = torch.tensor(x_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32)
x_val = torch.tensor(x_val, dtype=torch.float32)
y_val = torch.tensor(y_val, dtype=torch.float32)
```

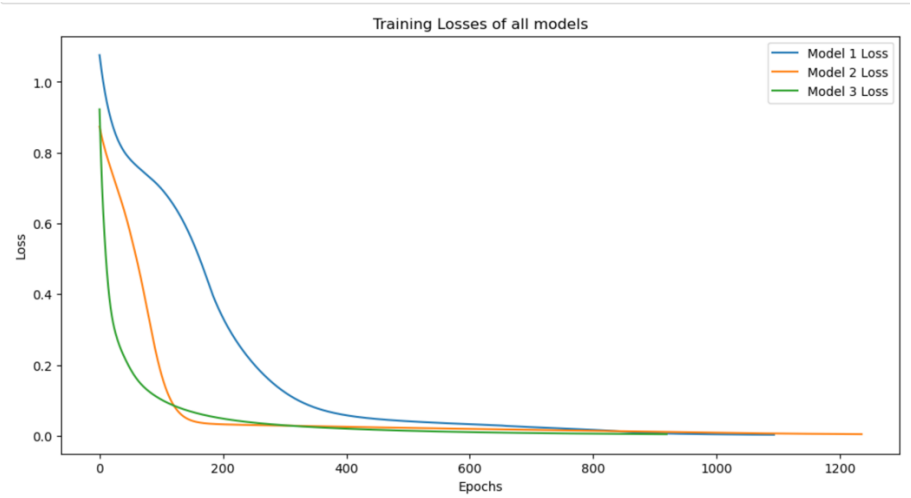
The plot for the above function is:



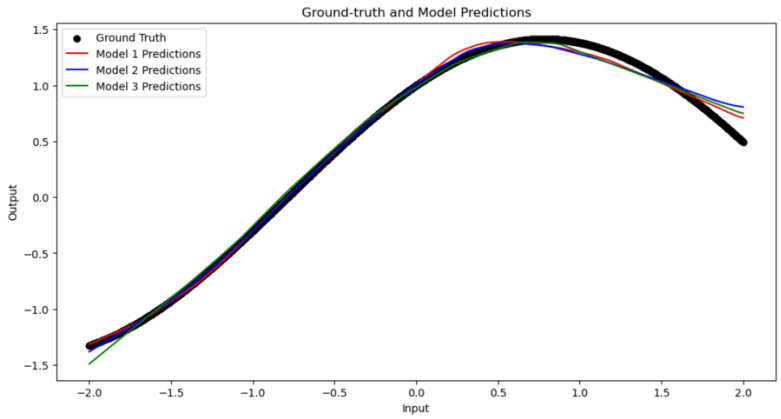
Three distinct DNN models were created to estimate the target function, differing in the number of hidden layers and neurons to investigate their impact on learning and generalization.

Model	No. of Hidden layers	No. of parameters	Loss function	Optimizer	Learning rate	Convergence at epoch	Loss at convergence
Model 1	4	5831	MSELoss	SGD	1e-2	1093	0.00284
Model 2	3	4011	MSELoss	SGD	1e-2	1235	0.00444
Model 3	1	1141	MSELoss	SGD	1e-2	919	0.00434

Simulation of the function 1:
The models converged after reaching the epoch limit or when the model learns at a slow rate. Below is the graph for training loss of all the models:



Below is the plot for Ground Truth v/s Predictions of all models:



Observation:
The convergence data for models approximating sin(x)+cos(x) reveal that simpler models, like Model 3, efficiently learn with fewer resources, converging fastest with minimal loss. Model 1, although the most complex, offers only slightly better accuracy at a higher computational cost,

questioning the benefit of increased complexity. Model 2 presents a balanced approach but does not significantly outperform the simpler model.

Target Function 2:

The single input, single output non-linear function that is used is **sin(5*pi*x) / 5*pi*x.** Training data (x_train) is generated with uniform distribution between -1.5 and 1.5 with 0.01 step value, while validation data (x_val) is linearly spaced.

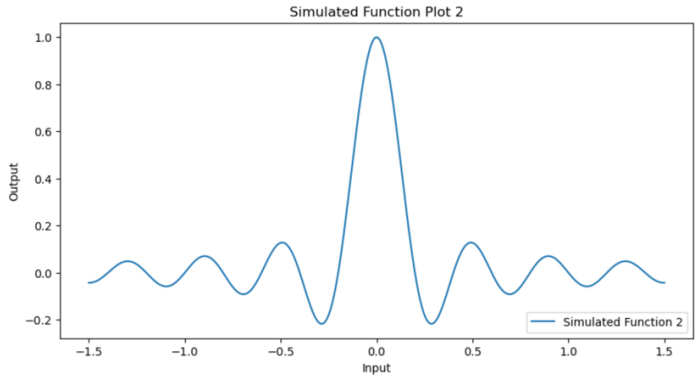
```
def simulate_function_2(x):
    return np.sinc(5*x)

x_train = np.linspace(-1.5, 1.5, int((3.0 / 0.01) + 1)).reshape(-1, 1)
y_train = simulate_function_2(x_train)

x_val = x_train
y_val = simulate_function_2(x_val)

x_train = torch.tensor(x_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32)
x_val = torch.tensor(x_val, dtype=torch.float32)
y_val = torch.tensor(y_val, dtype=torch.float32)
```

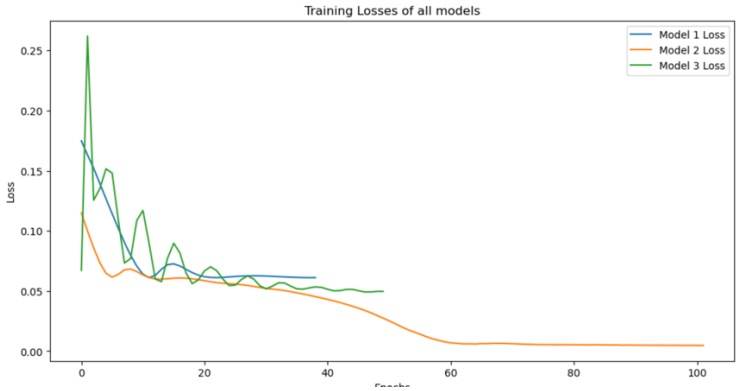
The plot for the above function is:



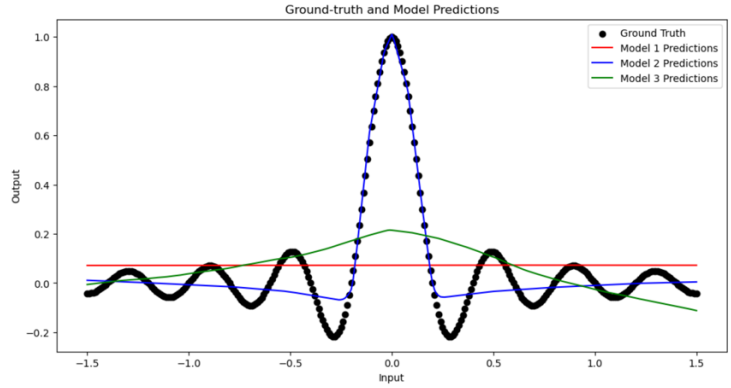
Three distinct DNN models were created to estimate the target function, differing in the number of hidden layers and neurons to investigate their impact on learning and generalization.

Model	No. of Hidden layers	No. of parameters	Loss function	Optimizer	Learning rate	Convergence at epoch	Loss at convergence
Model 1	6	571	MSELoss	Adam	1e-2	38	0.06112
Model 2	3	556	MSELoss	Adam	1e-2	101	0.00474
Model 3	0	547	MSELoss	Adam	1e-2	49	0.04967

Simulation of the function 2:
The models converged after reaching the epoch limit or when the model learns at a slow rate. Below is the graph for training loss of all the models:



Below is the plot for Ground Truth v/s Predictions of all models:



Observation:
Model 1 converges at epoch 38 with a loss of 0.0611, Model 2 shows superior fitting by converging at epoch 101 with a lower loss of 0.0047, and Model 3, with a simpler design, converges at epoch 49 with a loss of 0.0497. This indicates that Model 2, despite taking longer, best approximates the sinc(5x) function, while the complexity and structure of the models significantly influence their efficiency and accuracy in learning.

HW1 1-2 Train on Actual Tasks:

https://github.com/shravanik31/Deep-Learning/blob/main/Shravani_Konda_HW1_1_2_Train_On_Actual_Task.ipynb

The three CNN models created for classifying CIFAR-10 images with train batch size of 100 and test batch size of 64 are distinguished by their unique features and hyperparameters are below:

CNN Model 1:

- Convolutional Layers: 3 layers with 16, 32, and 64 filters
- Kernel size of 3x3 and padding=1.
- Batch Normalization: Applied after each convolutional layer for 16, 32, and 64 filters.
- Max-Pooling: Applied after each convolutional layer with a 2x2 window.
- Fully Connected Layers: Two layers with 120 and 10 neurons, respectively.
- Activation: ReLU for intermediate layers; log softmax for the output layer.

CNN Model 2:

- Convolutional Layers: 3 layers with 32, 64, and 128 filters.
- Kernel size of 3x3 and padding=1.
- Batch Normalization: Applied after each convolutional layer for 32, 64, and 128 filters.
- Max-Pooling: Applied after each convolutional layer with a 2x2 window.
- Fully Connected Layers: Two layers with 256 and 10 neurons, respectively.
- Activation: ReLU for intermediate layers; log softmax for the output layer.

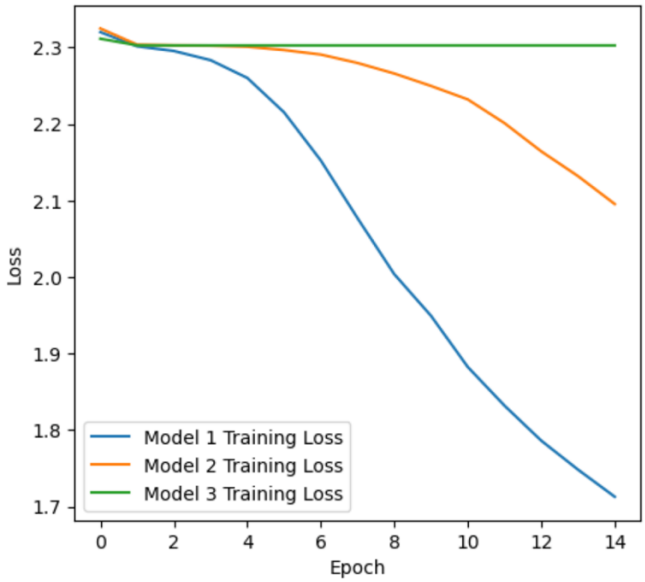
CNN Model 3:

- Convolutional Layers: 4 layers with 64, 128, 256, and 256 filters.
- Kernel size of 3x3 and padding=1.
- Batch Normalization: Applied after each convolutional layer for 64, 128, 256, and 256 filters.
- Max-Pooling: Applied after the first, second, and fourth convolutional layers with a 2x2 window.
- Dropout: Applied before the first fully connected layer with a dropout rate of 0.25 and before the final layer with a rate of 0.5.
- Fully Connected Layers: Two layers with 512 and 10 neurons, respectively.
- Activation: ReLU for intermediate layers; log softmax for the output layer.

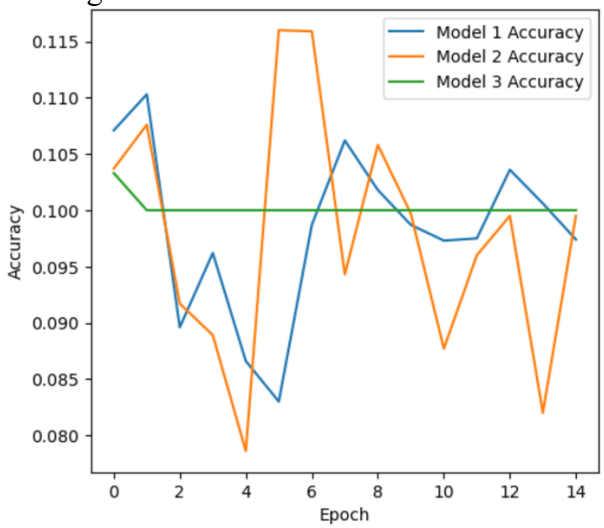
The hyperparameters used are below:

- Learning Rate (LR): 0.001
- Loss Function: CrossEntropyLoss
- Optimizer: Adam with the model parameters as its argument.
- Number of Epochs: 20

Below is the plot for the training losses for all models:



Below is the plot for the training accuracies for all models:



Observation:
Across the epochs, Model 1 and Model 2 display a pattern of decreasing training loss, suggesting learning from the training data, but this does not translate into improved performance on the test set, as indicated by the increasing test loss and low accuracy, hovering around 10%. Conversely, Model 3 exhibits no significant change in loss or accuracy throughout the epochs, indicating a lack of learning and poor generalization ability. These observations suggest that while Models 1 and 2 may be overfitting to the training data, Model 3 fails to learn from it altogether.

HW1 2 Optimization:

HW1 2-1 Visualize the optimization process:

https://github.com/shravanik31/Deep-Learning/blob/main/Shravani_Konda_HW1_2_1.ipynb

DNN Model:

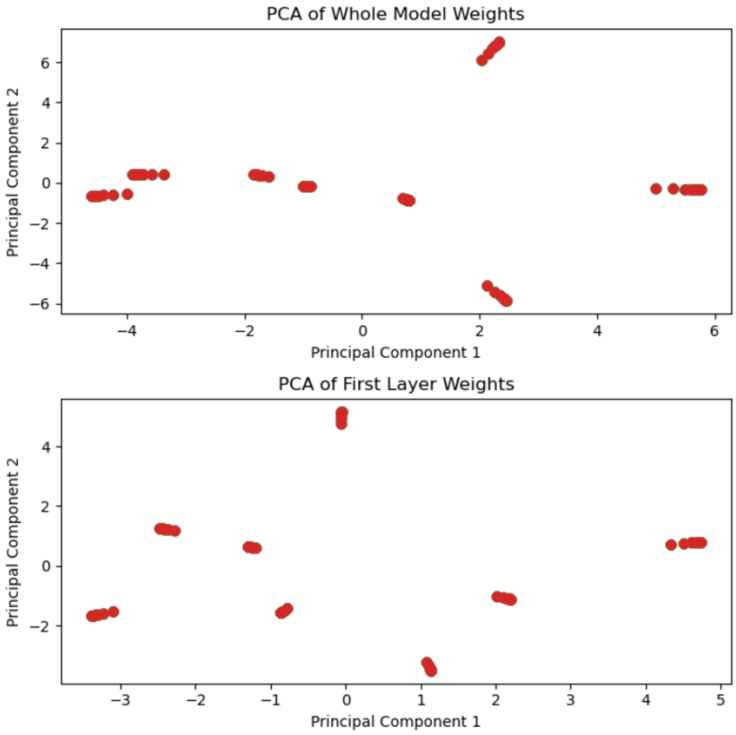
- An input layer with 784 units to match the flattened input dimensions of 28x28 pixel MNIST images.
- A hidden layer with 128 neurons.
- A second hidden layer with 64 neurons.
- An output layer with 10 neurons corresponding to the 10 classes of digits (0-9).
- ReLU activation functions are applied.

The model utilizes the MNIST dataset, normalized to a mean and standard deviation of 0.5. The DataLoader is configured with a batch size of 64 for both the training and testing sets.

The training process involves 8 separate training sessions, each spanning 24 epochs. However, the weights are recorded at intervals of every 3 epochs, totaling 8 recordings per session. The optimization is performed using the Adam optimizer with a learning rate set at 0.001, and the CrossEntropyLoss function computes the discrepancy between predicted outputs and actual labels. During each training iteration, the network's first-layer weights (fc1) and the collective weights from all layers are extracted and stored.

After training, Principal Component Analysis (PCA) is conducted on these weight sets to distill the information down to two principal components for both the complete model weights and the first-layer weights.

Below are the graphs for PCA reduced weights from full model and PCA reduced weights from the first layer of model:

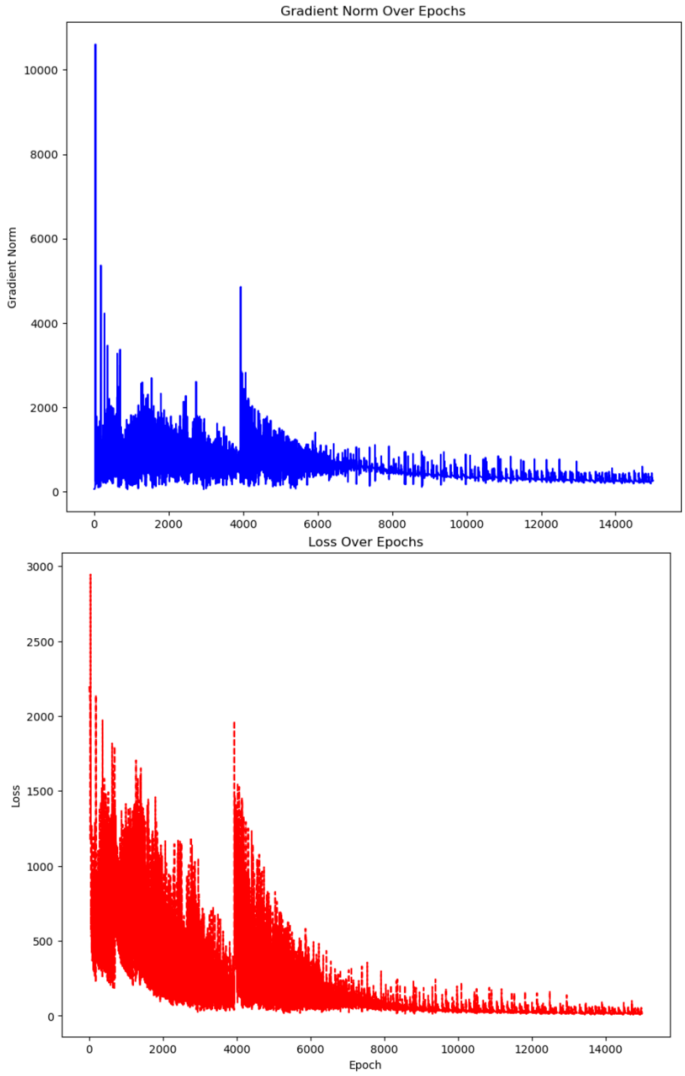


HW1 2-2 Observe gradient norm during training:

https://github.com/shravanik31/Deep-Learning/blob/main/Shravani_Konda_HW1_2_2.ipynb

The single input, single output non-linear function that is used is **exp(x)+exp(-x) i.e, hyperbolic cosine function.** Training data (x_train) is generated with uniform distribution between -5 and 5 with 0.001 step value, while validation data (x_val) is linearly spaced.

Below is a plot for gradient norm, loss across the epochs:



Observation:

The training of the model exhibits an initial period of high volatility in both loss and gradient norms, likely due to large parameter updates. As training progresses, both metrics stabilize, indicating convergence towards a solution. Sporadic spikes in later epochs suggest sensitivity to certain data patterns or noise, but the overall trend of decreasing loss signifies improving model performance over time.

HW1 2-3 What happens when gradient is almost zero?

https://github.com/shravanik31/Deep-Learning/blob/main/Shravani_Konda_HW1_2_3.ipynb

The single input, single output non-linear function that is used is **exp(x)+exp(-x) i.e, hyperbolic cosine function.** Training data (x_train) is generated with uniform distribution between -3 and 3 with 0.001 step value, while validation data (x_val) is linearly spaced.

DNN Model: The model architecture comprises an input layer with one neuron, followed by five hidden layers decreasing in size, each applying a linear transformation and Leaky ReLU activation function. Finally, there's an output layer with one neuron. This setup aims to approximate a given function by learning the relationship between input and output data.

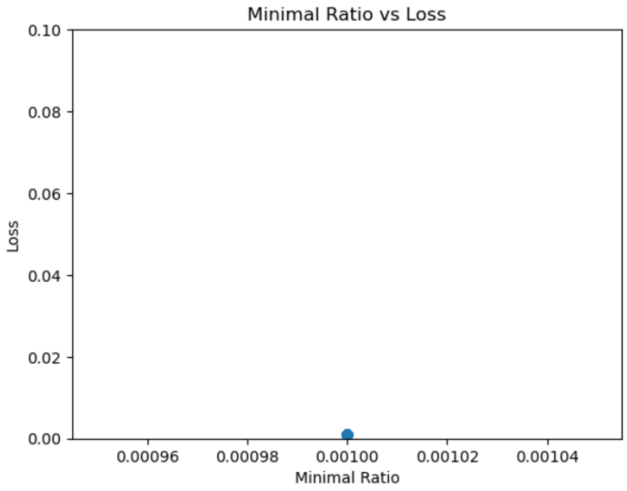
The gradient norms are monitored during training. If the norm of gradients for all parameters falls below a certain threshold (0.001 in this case), the model is considered to have converged.

The minimal ratio is calculated based on the eigenvalues of the Hessian matrix.

The model is trained for 100 iterations, with each iteration involving the following steps:

- Initialize the model's parameters and optimizer.
- Train the model, which includes backpropagation, gradient calculation, and parameter updates until convergence or a maximum number of epochs.
- Monitor the gradient norms during training.
- Calculate the minimal ratio based on the eigenvalues of the Hessian matrix.
- Record the loss value and minimal ratio obtained during training for analysis.

Below is the plot for minimal ratio v/s loss:



Observation:

The experiment involved training the model 100 times, each time for 100 epochs. Throughout these iterations, the loss function approached nearly zero, indicating that the model was effectively learning to minimize the difference between predicted and actual values. Additionally, the minimal ratio of 0.001 suggests that the ratio of positive eigenvalues to the total number of parameters in the Hessian matrix was extremely small, implying that the optimization process was stable and that the model likely reached a good solution.

HW1 3 Generalization:

HW1 3-1 Can network fit random labels?

https://github.com/shravanik31/Deep-Learning/blob/main/Shravani_Konda_HW1_3_1.ipynb

Below model is trained on the CIFAR-10 dataset

CNN Model:

- Convolutional Layers: Three convolutional layers with 16, 32 and 64 filters
- Kernel size of 3x3 and padding of 1.
- Pooling Layers: Max pooling with a 2x2 window is applied after each convolutional layer.
- Fully Connected Layers: The data is flattened and passed through two fully connected layers. The first fully connected layer comprises 120 neurons, and the output layer has 10 neurons.
- Activation Functions: ReLU activation is applied.
- The output layer utilizes log softmax activation.

CIFAR-10 Dataset:

- The dataset consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class.
- Transformations: Each image is converted to a tensor and normalized using the mean (0.4914, 0.4822, 0.4465) and standard deviation (0.2023, 0.1994, 0.2010) of the dataset.

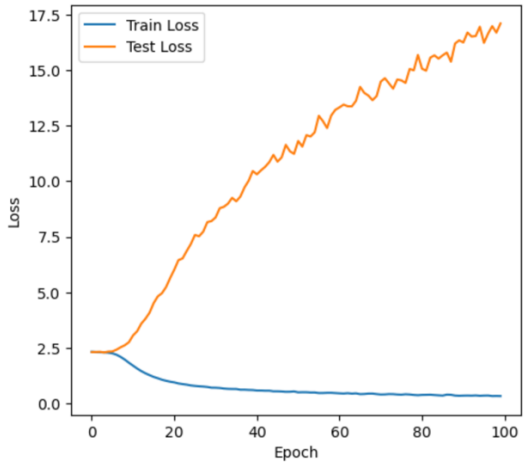
The training set labels are randomized by replacing them with random integers from 0 to 9. This randomization complicates the classification task, as the labels no longer correspond to the actual content of the images.

Training Process:

- Batch Sizes: The training batch size is set to 100, while the test batch size is 64.
- Training Loop: The model is trained for 100 epochs and the training loss is recorded at each epoch.
- Loss Function: CrossEntropyLoss
- Optimizer: The Adam optimizer is utilized with a learning rate of 0.001.

  At the end of each training epoch, the model's performance is evaluated on the test set to compute the test loss.

Below is the plot for training and testing losses v/s epochs.



Observation:

The model's training loss steadily decreases over the epochs, indicating effective learning from the training data. However, the test loss initially fluctuates before gradually increasing, suggesting overfitting as the model becomes too tailored to the training set. Despite fluctuations in accuracy, it maintains a relatively stable performance, indicating a balance between underfitting and overfitting.

HW1 3-2 Number of parameters vs Generalization:

https://github.com/shravanik31/Deep-Learning/blob/main/Shravani_Konda_HW1_3_2.ipynb

CNN Models (Net_1 to Net_10) vary in the number of channels in their convolutional layers and the number of neurons in their fully connected layers:

- Net_1: 16 channels, 120 neurons
- Net_2: 32 channels, 240 neurons
- Net_3: 48 channels, 360 neurons
- Net_4: 64 channels, 480 neurons
- Net_5: 80 channels, 600 neurons
- Net_6: 96 channels, 720 neurons
- Net_7: 112 channels, 840 neurons
- Net_8: 128 channels, 960 neurons
- Net_9: 144 channels, 1080 neurons
- Net_10: 160 channels, 1200 neurons

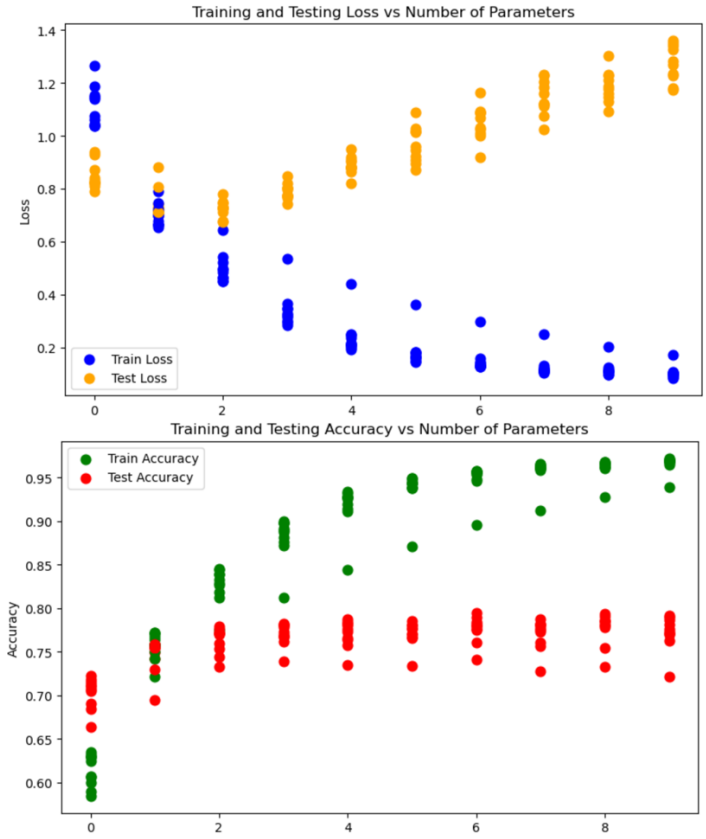
CIFAR-10 dataset is used. CIFAR-10 images are transformed to tensors and normalized using the specified mean (0.4914, 0.4822, 0.4465) and standard deviation (0.2023, 0.1994, 0.2010).

Training Settings:

- Optimizer: Adam with a learning rate of 0.001.

- Loss Function: CrossEntropyLoss
- Epochs: Each model is trained for 10 epochs.
- Batch Size: 100 for training, 64 for testing.

Below is a figure of training and testing, loss and accuracy to number of parameters.



Observation:

Across all models, the training loss consistently decreases, and the training accuracy increases with successive epochs, indicating a learning progression. However, the test loss tends to rise after initial improvements, and the test accuracy plateaus or even slightly decreases. This pattern suggests that while the models are gaining proficiency on the training data, they are becoming overfitted and losing generalization ability on the test data. The increasing complexity of models, as indicated by the number of channels and FC size, does not necessarily lead to better generalization, shown by the higher test losses and plateauing test accuracies in later epochs.

HW1 3-3 Flatness vs Generalization part-1:

https://github.com/shravanik31/Deep-Learning/blob/main/Shravani_Konda_HW1_3_3_PartA.ipynb

The model is trained on the CIFAR-10 dataset. CIFAR-10 images are converted to tensors and normalized using mean (0.4914, 0.4822, 0.4465) and standard deviation (0.2023, 0.1994, 0.2010).

CNN_Model:

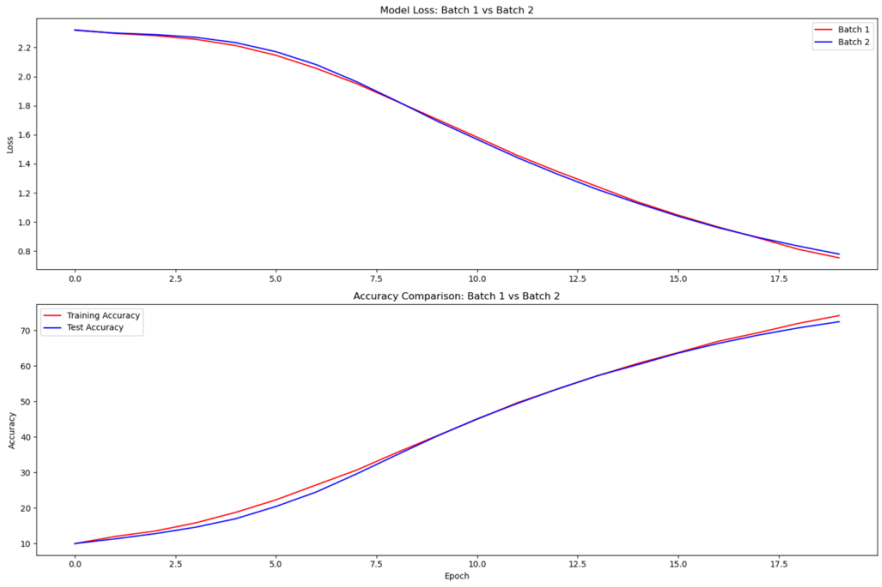
- Convolutional Layers: Three layers with 16, 32, and 64 filters respectively.
- Kernel size of 3x3 and padding of 1.
- Fully Connected Layers: Two layers, with the first having 120 neurons and the second having 10 neurons.
- Activation and Pooling: ReLU activation is used after each convolutional layer, followed by max-pooling with a window of size 2.

Training Approach:

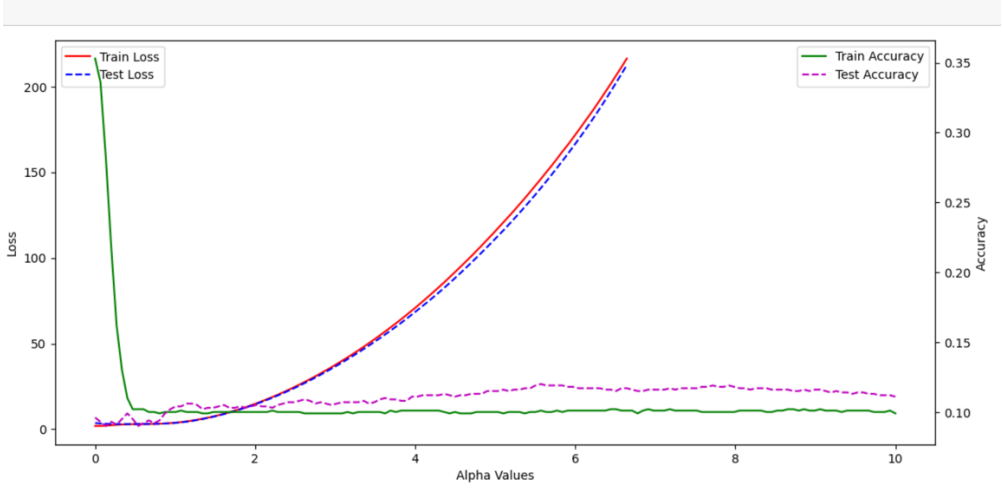
- Batch Sizes: Two separate trainings are conducted with batch sizes of 500 and 300 respectively.
- Loss Function: CrossEntropyLoss is used.
- Optimizer: Adam with a learning rate of 0.001.
- Training Duration: Each model is trained for an equal number of 15 epochs.

Parameters from both trained models are linearly interpolated to create a continuum of models, parameterized by an alpha value ranging from 0 to 1 in 100 steps. For each interpolated set of parameters, the model's loss and accuracy on the training and test sets are evaluated.

Below plot is the comparison of training losses and test accuracies for two models with two different batch sizes:



Below is the plot for the training and test loss and accuracy across the range of alpha values.



Observation:

Both models demonstrate a decrease in training loss and an increase in training accuracy over epochs, indicative of learning from the training data. However, the test loss for both models initially decreases but subsequently begins to rise, while the test accuracy plateaus, signaling that the models are overfitting to the training data and not generalizing well to new, unseen data. The accuracy peaks at less than 35%, suggesting the models might require architectural improvements, more representative training data, or additional hyperparameter optimization to enhance their predictive performance.

HW1 3-3 Flatness vs Generalization part-2:

https://github.com/shravanik31/Deep-Learning/blob/main/Shravani_Konda_HW1_3_3_PartB.ipynb

The CIFAR-10 dataset is used for both training and testing. It consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. The dataset is divided into 50,000 training images and 10,000 testing images.

CNN_Model: The CNN model architecture comprises one convolutional layers followed by two fully connected layers. Each convolutional layer is followed by a ReLU activation function and batch normalization. The output layer utilizes a log-softmax activation function to output class probabilities.

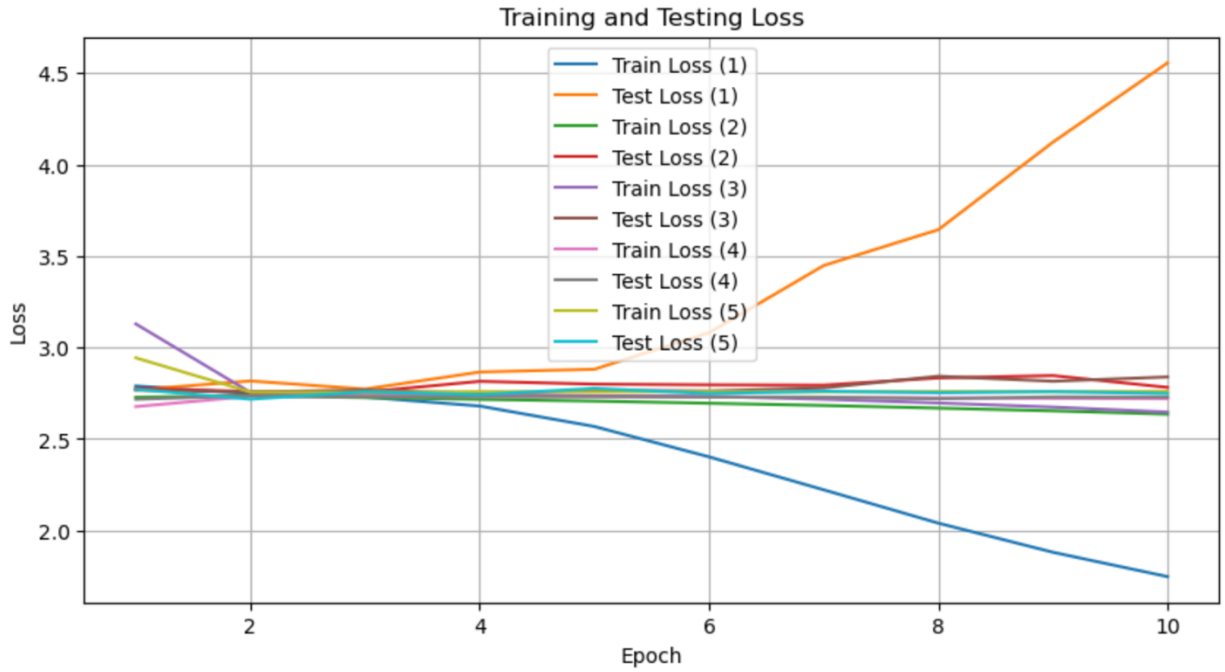
Training Approach:

- Optimizer: The experiments use different optimizers (Adam and SGD) with varying learning rates.
- Loss Function: Cross-entropy loss is employed as the loss function.
- Batch Size: The batch size varies across experiments (64, 128, or 256).
- Epochs: The models are trained for a fixed number of epochs (10 epochs in this case).

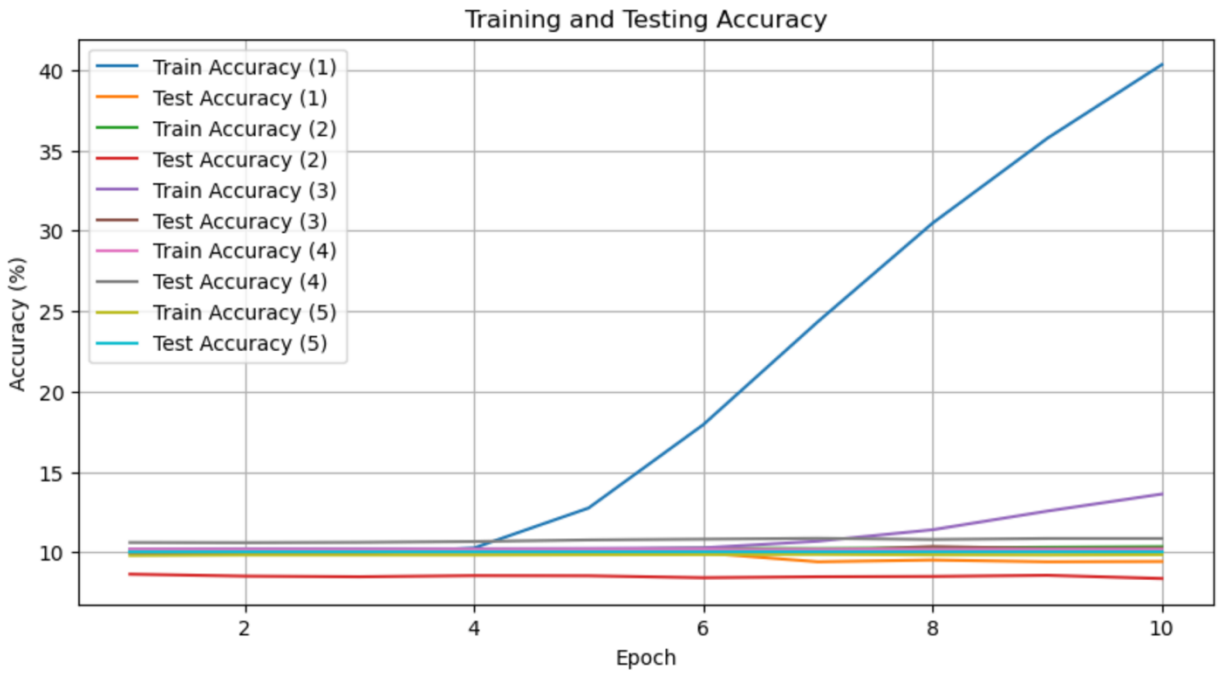
- Training and Evaluation: The training loop iterates over the dataset for multiple epochs, computing loss, accuracy, and sensitivity metrics at each epoch.
- Sensitivity Calculation: Sensitivity, a measure of how much the model's output changes concerning small changes in input, is computed and tracked during training.

Different combinations of batch sizes, optimizers, and learning rates are experimented with to observe their impact on training and testing metrics. 10 epochs are performed for each configuration, capturing the variability in model performance.

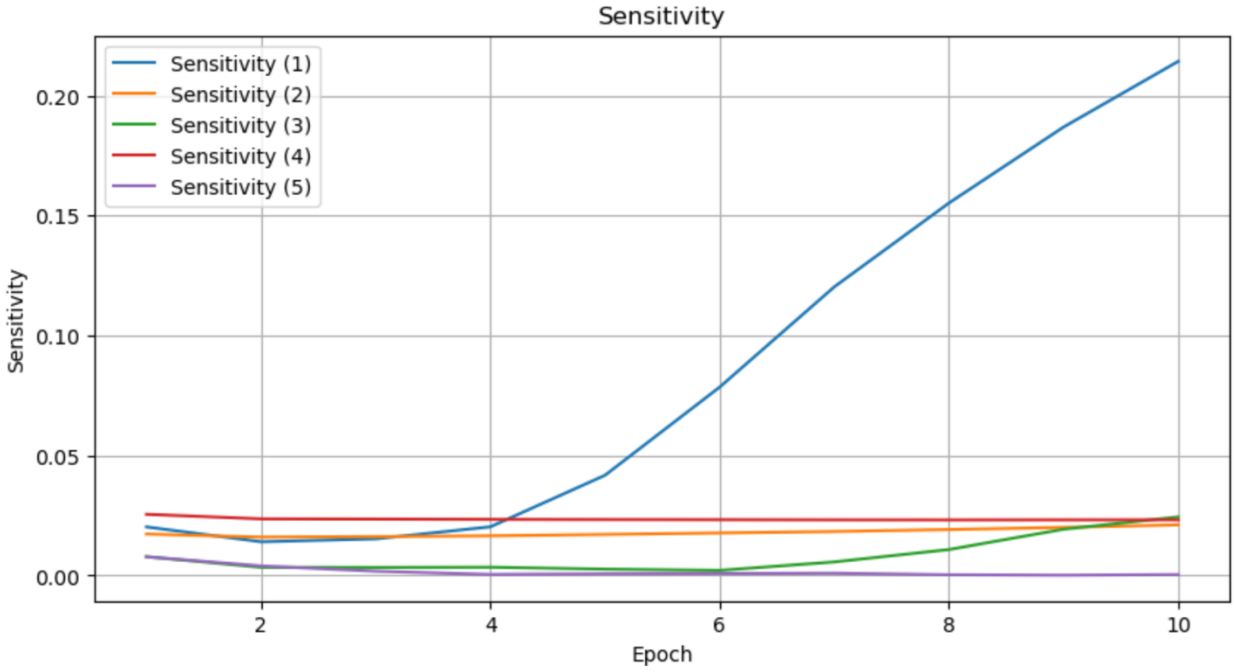
Below is the plot for training and test losses for all 5 models against epochs:



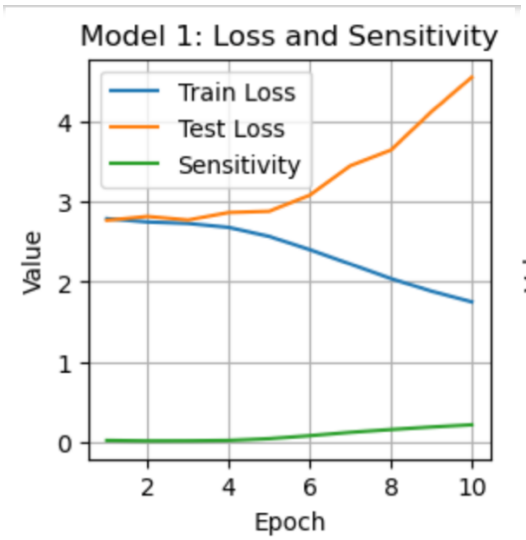
Below is the plot for training and test accuracies for all 5 models against epochs:

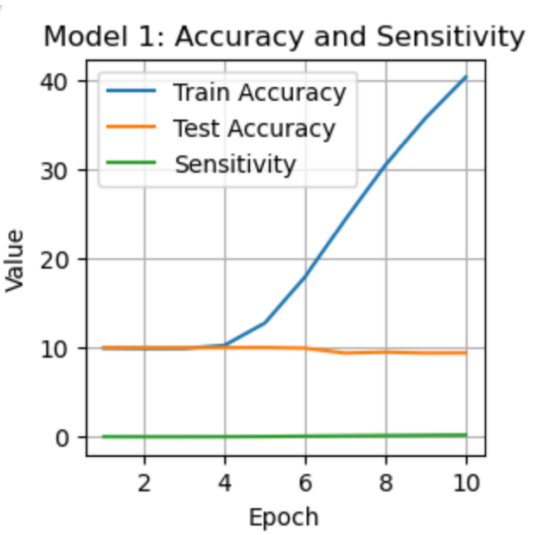


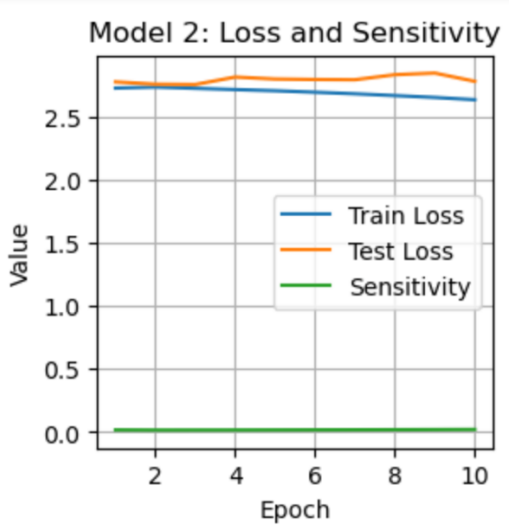
Below is the plot for sensitivities for all 5 models against epochs:

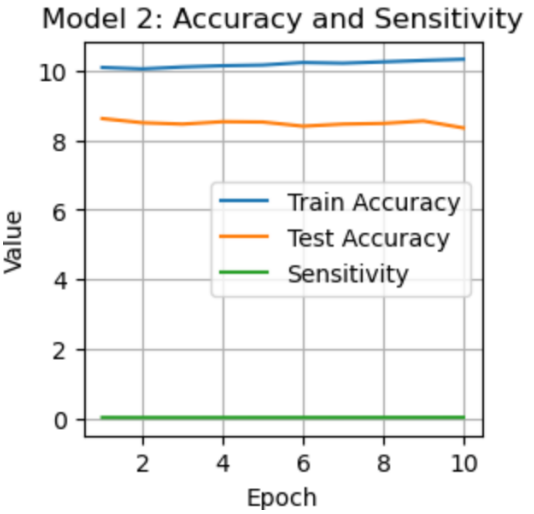


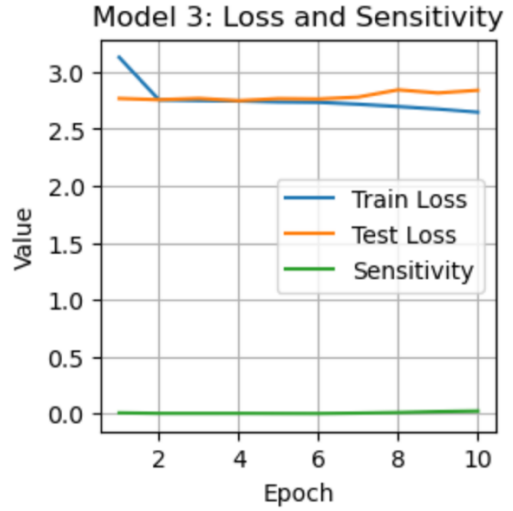
Below are the plots for all combinations: train loss, test loss, sensitivity against epochs train accuracy, test accuracy, sensitivity against epochs:

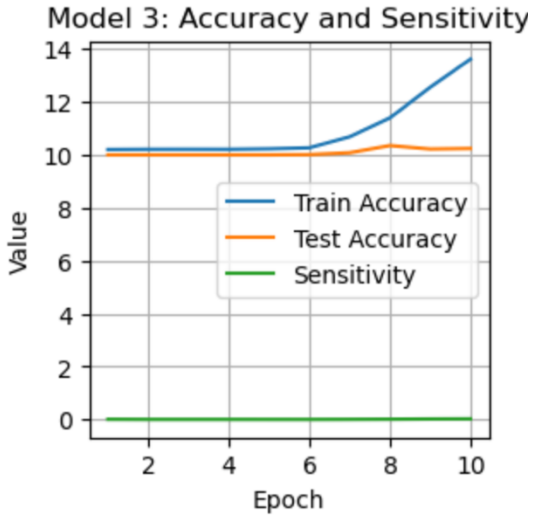

Model 1: Loss and Sensitivity
Train Loss
Test Loss
Sensitivity
Value
Epoch

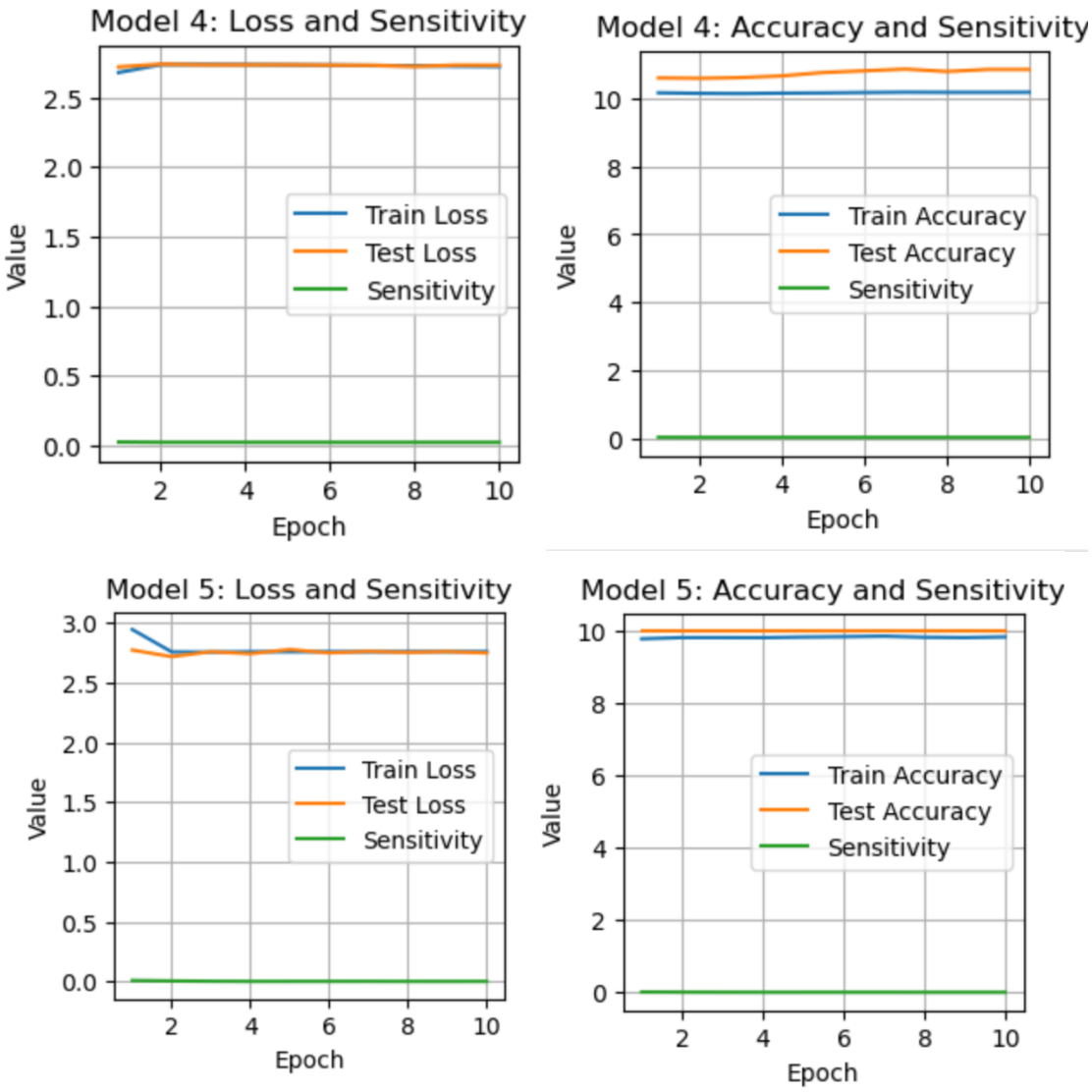

Model 1: Accuracy and Sensitivity
Train Accuracy
Test Accuracy
Sensitivity
Value
Epoch


Model 2: Loss and Sensitivity
Train Loss
Test Loss
Sensitivity
Value
Epoch


Model 2: Accuracy and Sensitivity
Train Accuracy
Test Accuracy
Sensitivity
Value
Epoch


Model 3: Loss and Sensitivity
Train Loss
Test Loss
Sensitivity
Value
Epoch


Model 3: Accuracy and Sensitivity
Train Accuracy
Test Accuracy
Sensitivity
Value
Epoch



Observation:

Batch Sizes: Experimenting with different batch sizes (e.g., 64, 128, 256) influences the convergence speed and stability of the training process. Larger batch sizes tend to result in faster convergence initially but may suffer from decreased generalization performance, as indicated by fluctuating or stagnating test accuracies and sensitivities over epochs.

Optimizers: The choice of optimizers (e.g., Adam, SGD) impacts the optimization process, affecting how quickly the model learns and adapts to the training data. Adam tends to provide more stable and consistent performance across experiments, exhibiting smoother decreases in both training and testing losses compared to SGD.

Learning Rates: Varying learning rates affect the step size of parameter updates during optimization. Higher learning rates might lead to faster initial progress in reducing training loss,

but they may also cause instability or overshooting, resulting in erratic behavior in both training and testing metrics.