Homework 1 Deep Learning

Dan Plotkin, Nicole Birova, & Jack Hanke

Question 1.

Call the data in the table x_1 through x_5, going top to bottom.

Perceptron (zero-one) loss:

a. The perceptron loss function assigns a value of 1 to misclassified points, and 0 to correctly classified points. In this case, x_1 and x_2 have a perceptron loss of 1, as the prediction is the different sign as the ground truth, which makes them misclassified.

b. x_3, x_4, x_5 all have a perceptron loss of 0, as the prediction is the same sign as the ground truth, so they are correctly classified.

c. We certainly can choose weights that yield a lower total perceptron loss. We would make our decision boundary look like the equation x=3, which would improve classification accuracy and lower the total loss.

d. No, because the loss function's gradient is 0 everywhere except the decision boundary, so training a multilayer perceptron with gradient descent would result in little to no learning.

Squared error loss:

a. Squared error loss is computed as (g(x) - y)^2 which penalizes large deviations between prediction and ground truth values. x_5 has the highest loss value, as it has the greatest distance from its prediction to the ground truth.

b. Similarly, x_3 has the lowest loss value, as it has the smallest distance from its prediction to the ground truth.

c. Maybe, as the scale of g affects the loss, but the decision boundary only plots the sign of g. Moving the decision boundary closer to x_5 might yield a better total loss by noticeably decreasing the large x_5 loss at the expense of slightly increasing the other losses.

d. No, as this loss function for a classification task penalizes properly classified data points that just so happen to be far away from the decision boundary. For example, x_5 is classified correctly, and yet contributes to over 70% of the total loss. This is because the squared function is symmetric, and so significant deviations in either direction are penalized. This would lead to either bad performance or slow learning in a perceptron.

Binary cross-entropy loss:

a. Binary cross-entropy loss is effective in classification because it penalizes misclassified points more heavily while rewarding correct classifications with high confidence. x_2 has the highest loss, as it is most misclassified because it is large and positive while the ground truth is negative.

b. x_5 has the lowest loss, as it is very strongly positive when the ground truth is positive.

c. Yes, a similar boundary to the one described for the perceptron loss would almost certainly result in a smaller BCE.

d. Yes, BCE would be a good choice for training an MLP for classification, because it heavily penalizes misclassification while not penalizing proper classification as much.

Hinge loss:
a. Hinge loss penalizes misclassified points and those within the decision margin, encouraging the model to maximize the classification margin. x_2 has the highest loss, as it is most misclassified, as it is large and positive while the ground truth is negative.

b. x_3, x_5 are both classified correctly and greater in absolute value than 1, and so receive a loss of 0.

c. Similar to BCE and perceptron loss, adjusting the decision boundary would improve classification and lower hinge loss. So a similar boundary to the one described for the perceptron loss would almost certainly result in a smaller BCE.

d. Yes, Hinge would be a good choice for training an MLP for classification, because it heavily penalizes misclassification while not penalizing proper "confident" classification at all.

Question 2.

We create a single-layer perceptron with two weights and a bias. The equation is as follows

$$g(X) = Xw + b.$$

As we did not have any hidden neurons, we had no architectural hyperparameters. As for training hyperparameters, we chose to train over 1000 epochs at a learning rate of 0.01. These training hyperparameters were found via trial and error, viewing the associated loss curves for each attempt.

We cannot use the perceptron algorithm on this learning task, as the data is not linearly separable. This can be clearly seen from the plots of the original and augmented data.

We split the dataset into 64% for training, 20% for validating, 16% for testing.

Question 3.

a. The `run_one_epoch` function trains or evaluates a model for a single epoch and returns the accuracy and loss of that epoch. In the `run_one_epoch` function, we run the forward pass of the model and output logits shaped (N, 2), where N is the batch size, and 2 represents how many classes. This is because in PyTorch, nn.Module forward passes expect input data to be in batch-first form, which is (N, n_features). During the forward pass, we propagate forward with matrix multiplication, which preserves the batch size until the final output. Since our final layer has two neurons, our output will be (N, n_classes).

In the function, we convert our label tensor (shaped (N,) into binary by using a `torch.where` conditional, where we input the condition y > 0 to be 1, else 0. We then use `torch.nn.CrossEntropyLoss()` to compute our loss, which expects in labels (N, and raw logits (N, 2) as input. It takes the raw logits, applies the softmax, and then calculates the log loss between the probabilities and the labels. We calculate the accuracy by first converting our raw logits into predicted classes by using `torch.argmax`, which returns the index of the highest value logit in a tensor. We then are able to compute accuracy using real and predicted class labels.

b. The `run_experiment` function runs the entire training and evaluation for the model by running `pretrain_and_train`, which pretrains our model on an easier task, and then fine tunes it on the main dataset. The `plot_results` function creates six plotting panels. The first panel shows a visualization that shows our decision boundary of our model after pertraining on our pretrain dataset. The second panel shows the pre-trained model's decision boundary on the training set. The third panel shows the decision boundary of our trained model on our train dataset. The fourth panel shows the decision boundary of our trained model on our test set. The fifth panel shows the training and test loss per epoch during training. The last panel shows the accuracy per epoch on the train and test set for each epoch during training.

Question 4.

a. An indication of overfitting is that the test loss consistently increases as the training loss decreases. This is exhibited in Experiments #1 and #3, which demonstrate complicated decision boundaries that often capture a single point in these "spoke-like" patterns. It appears that the number of neurons has the greatest effect on whether or not the model overfits. Too many neurons lead to overfitting, as shown in the high-neuron Experiment #1, while the low-neuron Experiment #3 performs better.

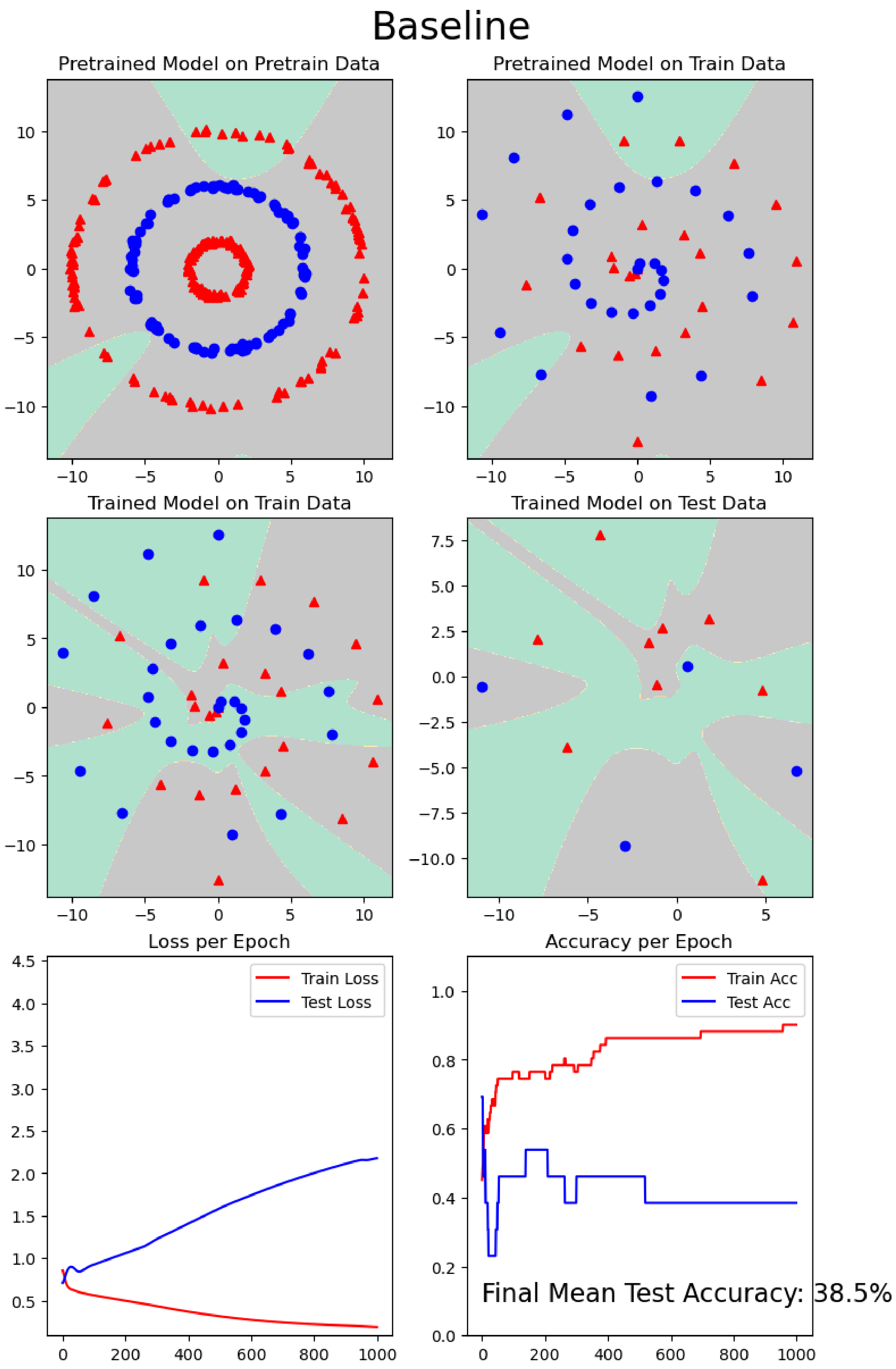
b. We first decrease, increase, then *really* increase the number of data points used for pretraining in Experiments #4, #5, and #6. We saw improvements in test performance when more data was provided for pretraining. In general, all experiments still exhibit the "spoke-like" decision boundaries, but gain 6% higher test accuracy than baseline with significantly more data in each ring for pretraining. We saw a decrease in performance from baseline when decreasing the number of datapoints used for pre-training.

In Experiment #7, we modified experiment #6 so that the time spent on training on pre-training data and real data was switched, with 10 to 1 time on pretraining. This did not affect performance, as both experiments were in the 46% range. This indicates that in some aspect, the number of epochs in total are more important than the split between the two.

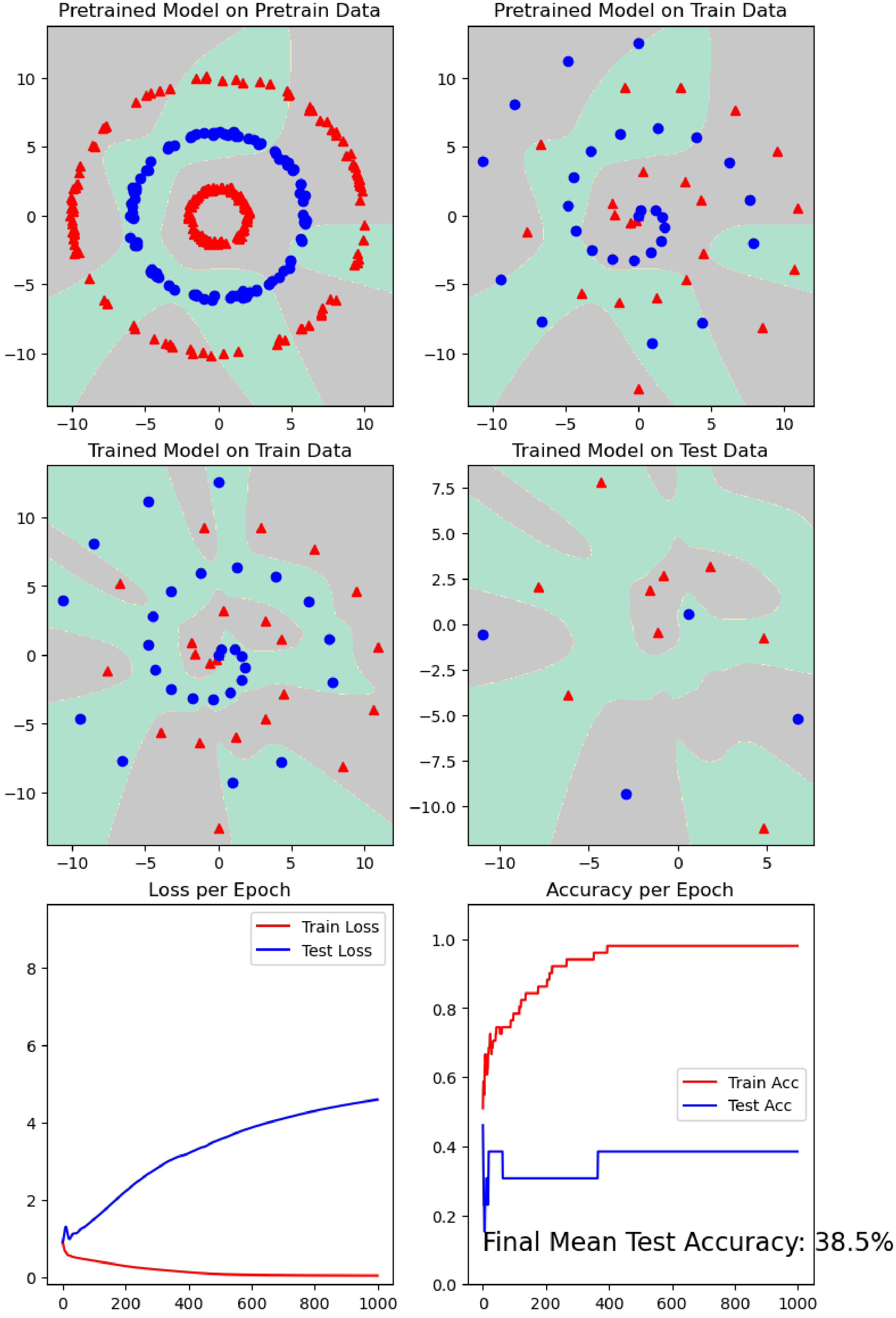
c. From the trends we found in the baseline and Experiment #1, we chose to design a low-neuron model. With Experiments #4, #5, and #6, we chose a training regiment with a high number of pre-training data points. We also chose the ReLu activation function, which showed a small performance gain in Experiment #2. Experiments #6 and #7 led us to choose to decrease both the time spent pretraining and training, with more time spent training. This resulted in our best test accuracy of around 54%. This was conducted in Experiment #3.

Overall, the trends we identified seemed to be more heuristic than set-in-stone. We just-as-often expected the performance change as we were surprised by it. We found it difficult to predict exactly what hyperparameters and changing to the training regiment would affect if anything. Overall, test performance appears equally affected by changes to network architecture and training regiment, and we see the best performance when both trends are followed in their implied directions.

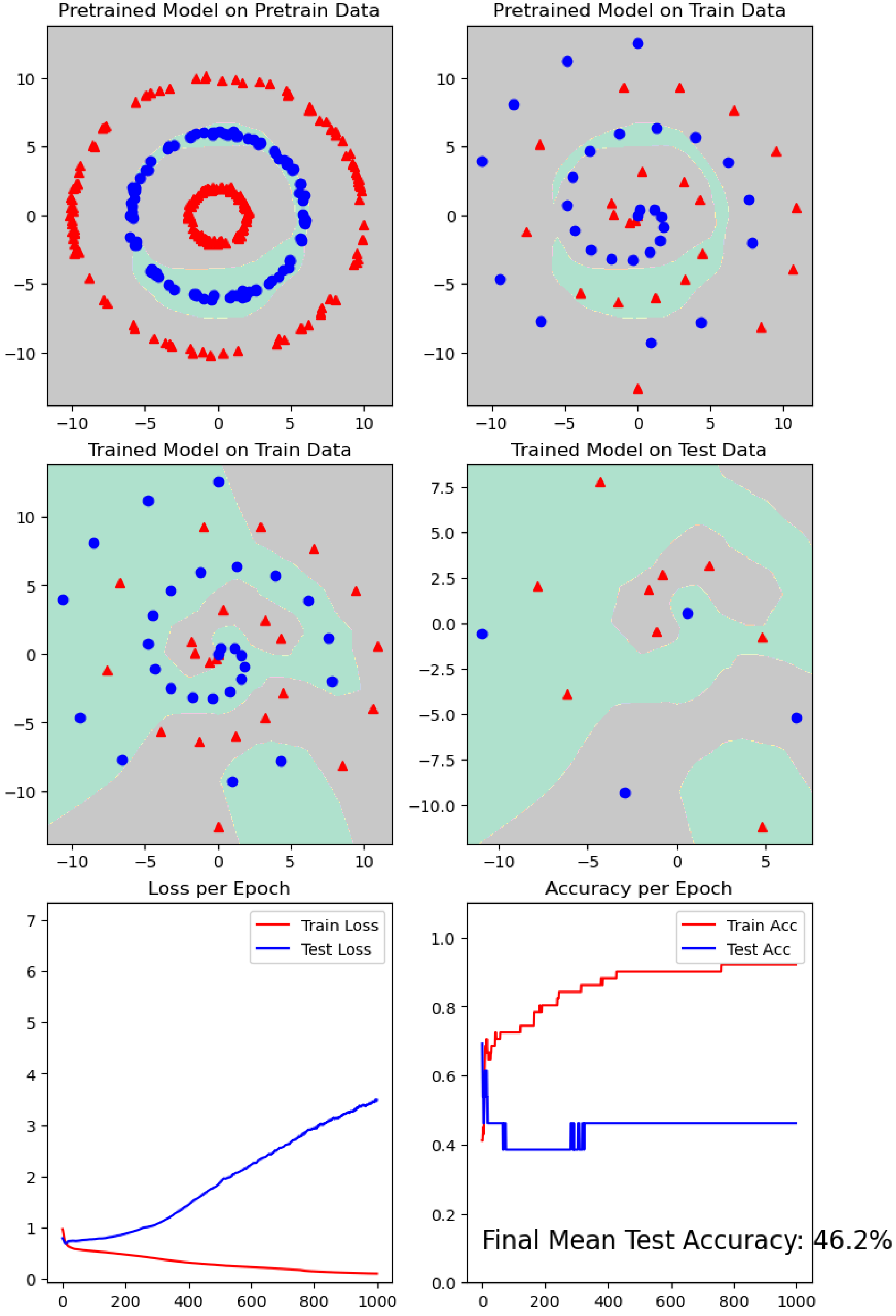
**See below for referenced images.


Baseline
Pretrained Model on Pretrain Data
Pretrained Model on Train Data
Trained Model on Train Data
Trained Model on Test Data
10
5
0
−5
−10
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
Loss per Epoch
Train Loss
Test Loss
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
200
400
600
800
1000
Accuracy per Epoch
Train Acc
Test Acc
1.0
0.8
0.6
0.4
0.2
0.0
Final Mean Test Accuracy: 38.5%

Experiment #1



Experiment #2



Pretrained Model on Pretrain Data

Pretrained Model on Train Data

Trained Model on Train Data

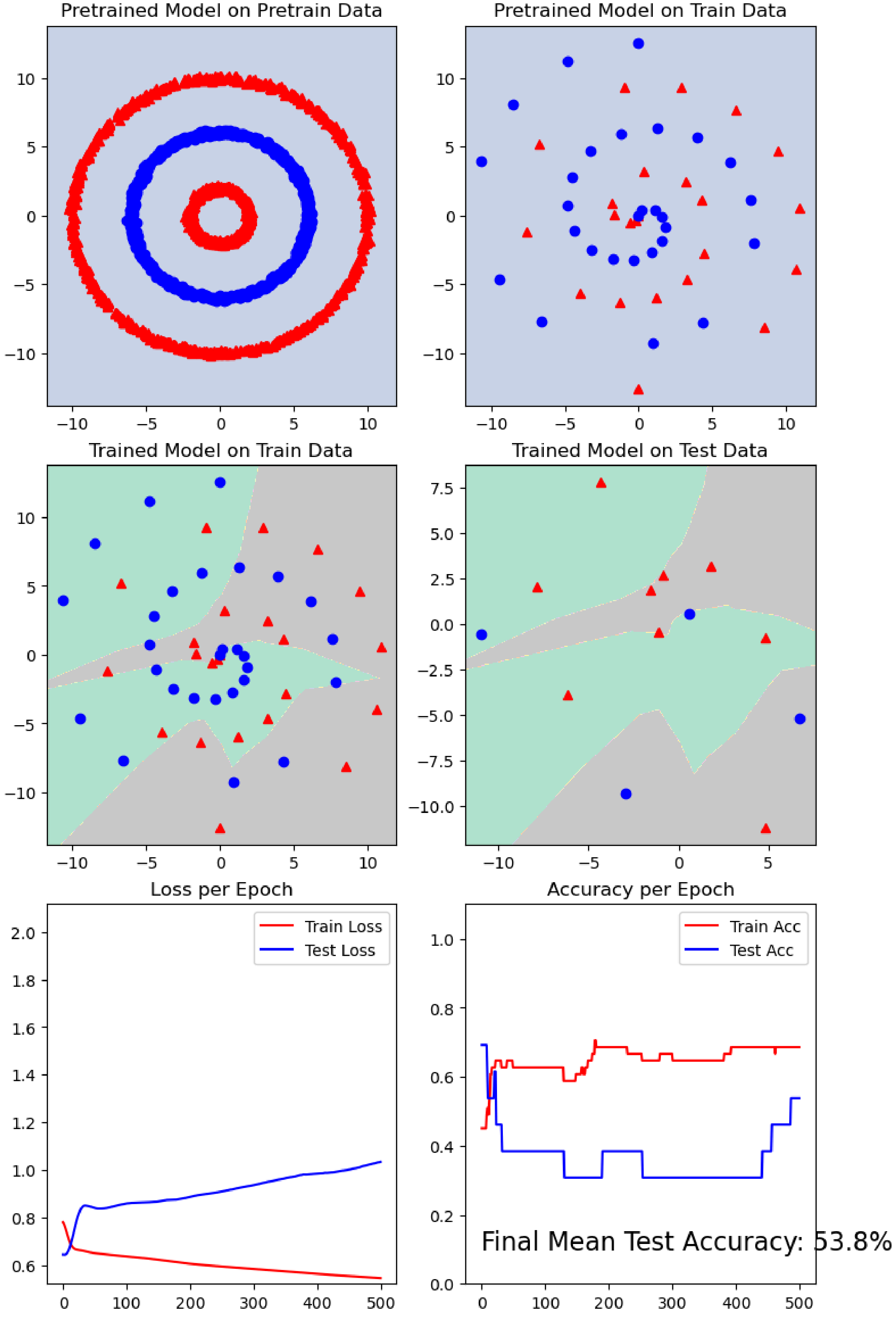
Trained Model on Test Data

Loss per Epoch

Accuracy per Epoch

Final Mean Test Accuracy: 46.2%

Experiment #3



Pretrained Model on Pretrain Data

Pretrained Model on Train Data

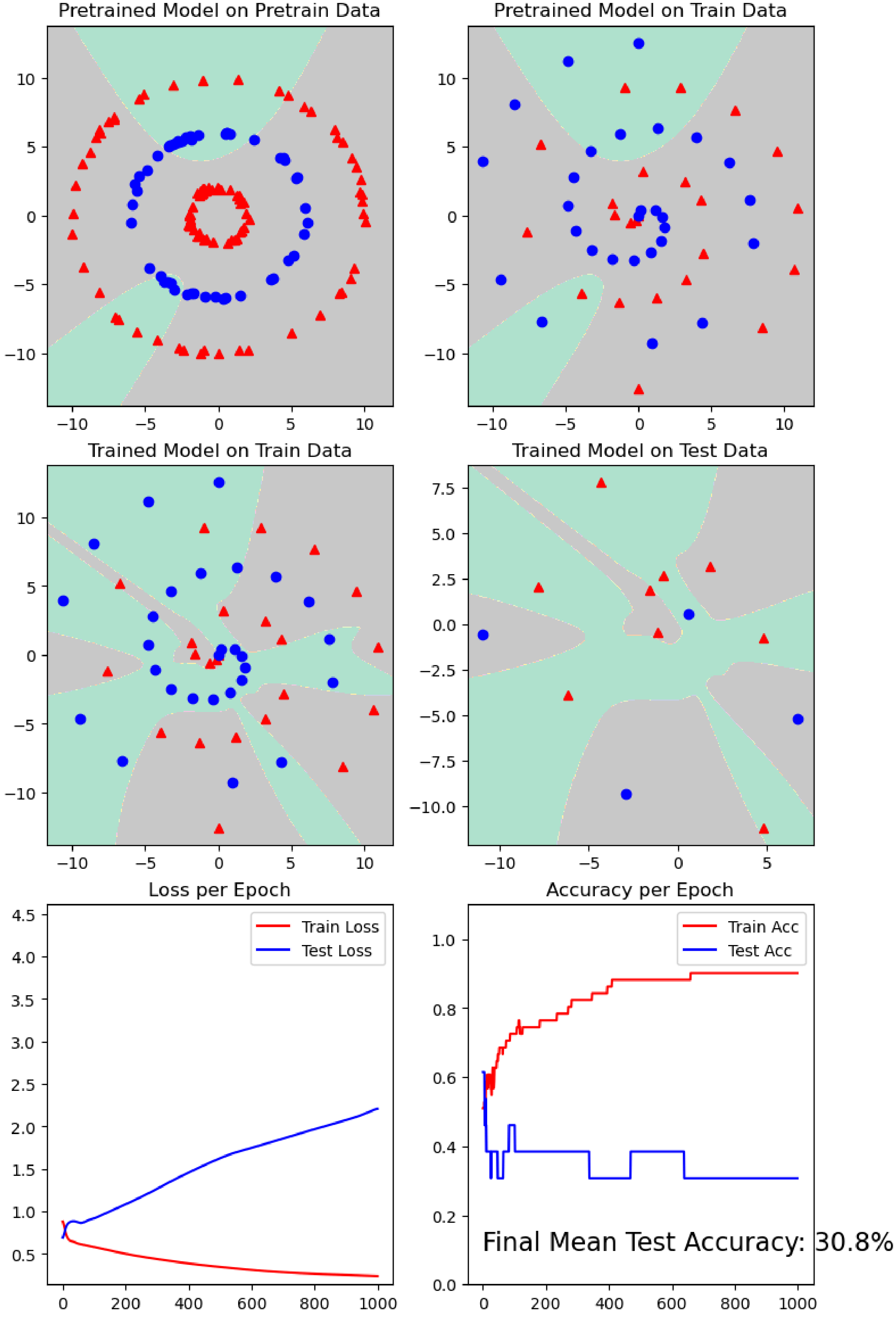
Trained Model on Train Data

Trained Model on Test Data

Loss per Epoch

Accuracy per Epoch

Experment #4



Pretrained Model on Pretrain Data

Pretrained Model on Train Data

Trained Model on Train Data

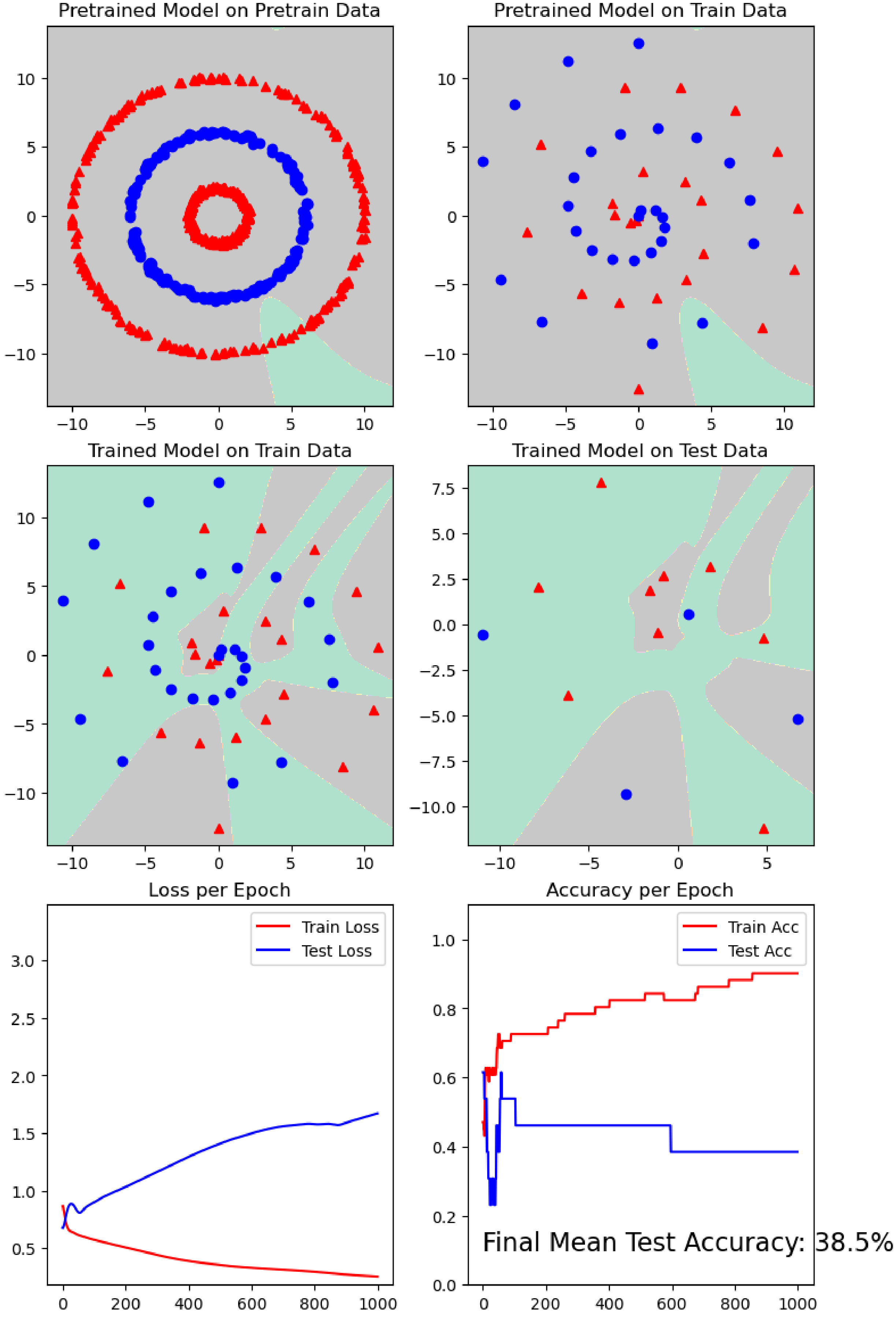
Trained Model on Test Data

Loss per Epoch

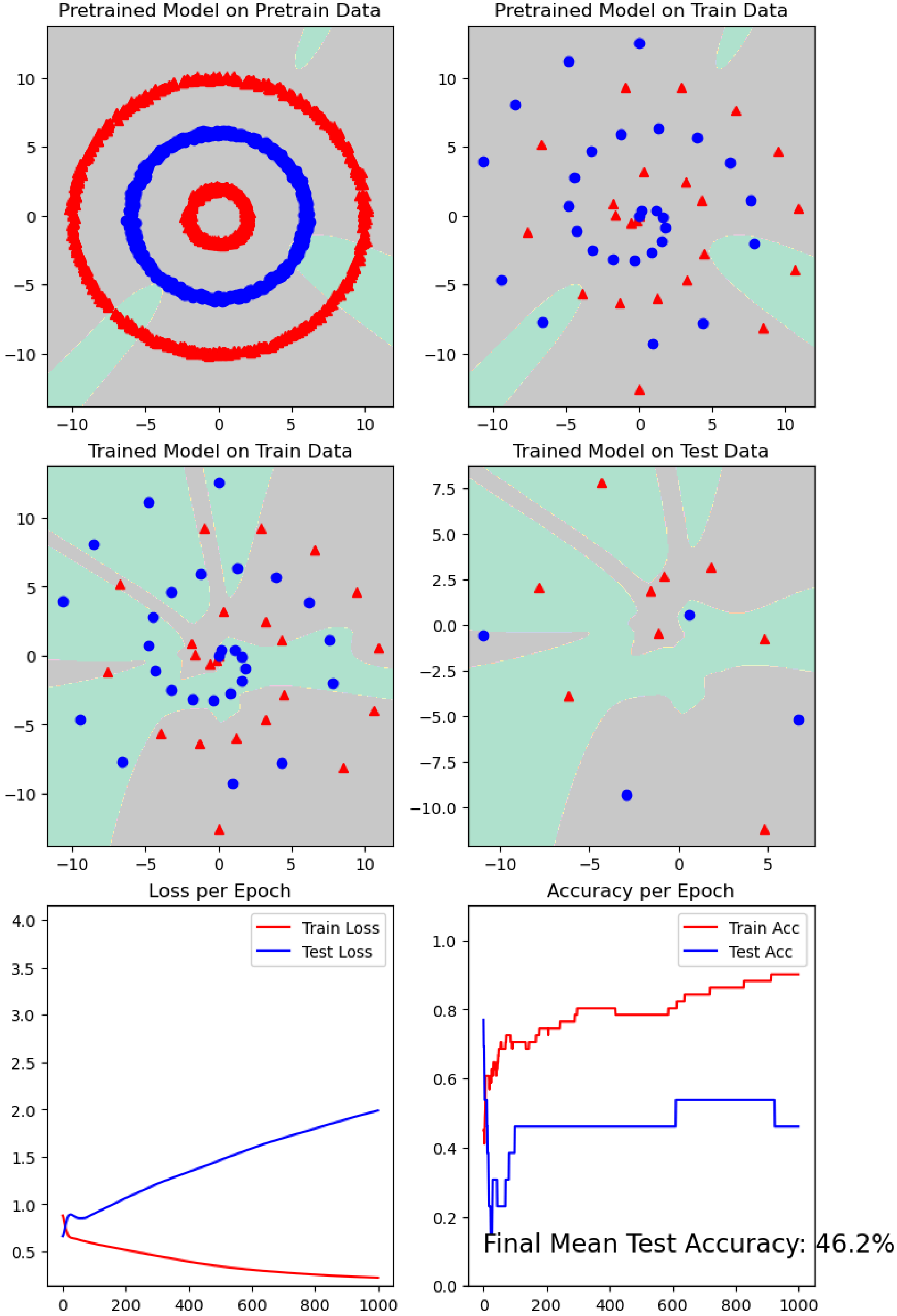
Accuracy per Epoch

Final Mean Test Accuracy: 30.8%

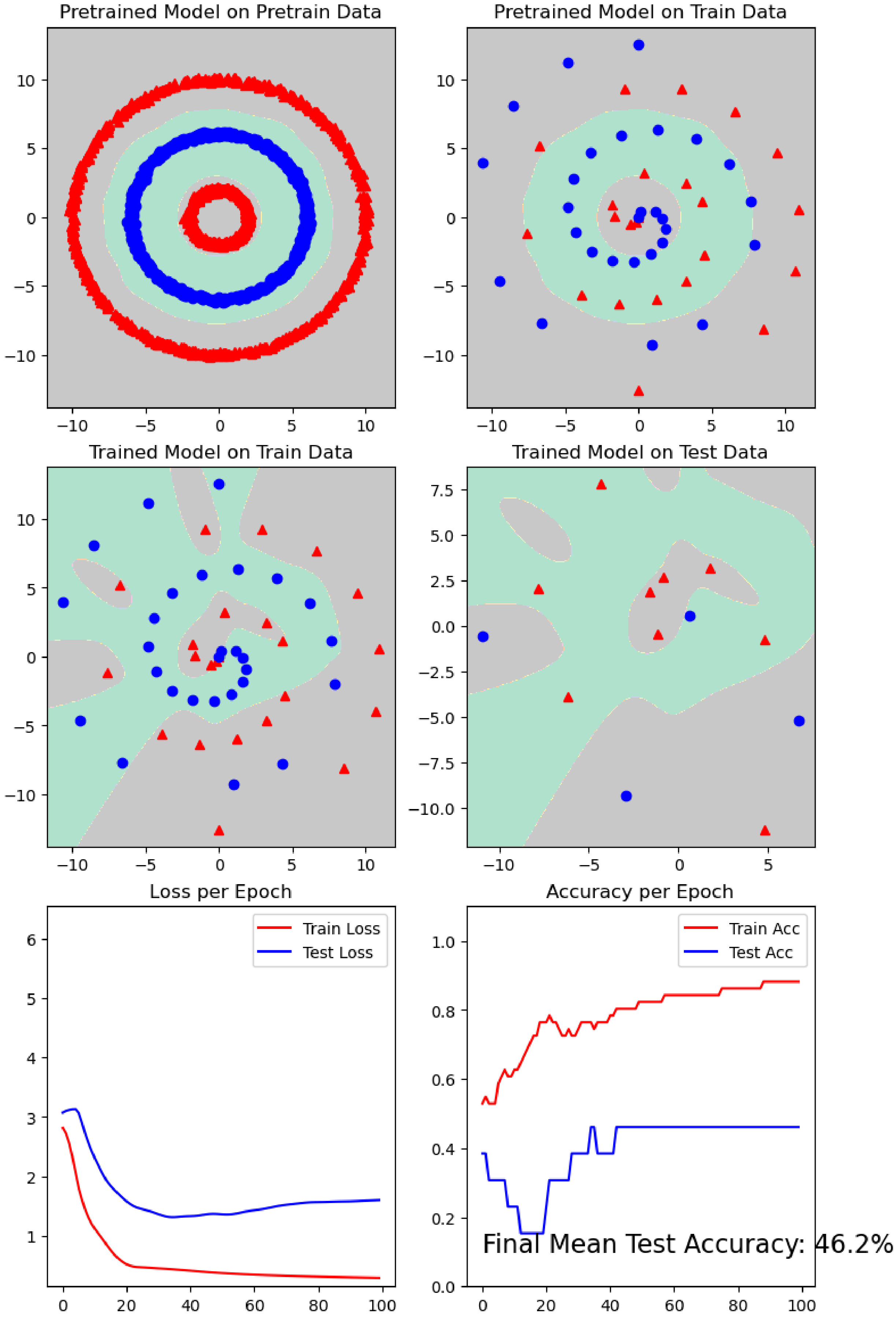
Experment #5


Pretrained Model on Pretrain Data
Pretrained Model on Train Data
Trained Model on Train Data
Trained Model on Test Data
Loss per Epoch
Train Loss
Test Loss
Accuracy per Epoch
Train Acc
Test Acc
Final Mean Test Accuracy: 38.5%
−10
−5
0
5
10
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.2
0.4
0.6
0.8
200
400
600
800
1000

Experment #6


Pretrained Model on Pretrain Data
Pretrained Model on Train Data
Trained Model on Train Data
Trained Model on Test Data
Loss per Epoch
Train Loss
Test Loss
Accuracy per Epoch
Train Acc
Test Acc
−10
−5
0
5
10
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.8
0.6
0.4
0.2
200
400
600
800
1000
Final Mean Test Accuracy: 46.2%

Experment #7


Pretrained Model on Pretrain Data
Pretrained Model on Train Data
Trained Model on Train Data
Trained Model on Test Data
Loss per Epoch
Train Loss
Test Loss
Accuracy per Epoch
Train Acc
Test Acc
Final Mean Test Accuracy: 46.2%
−10
−5
0
5
10
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
1
2
3
4
5
6
20
40
60
80
100
0.0
0.2
0.4
0.6
0.8
1.0