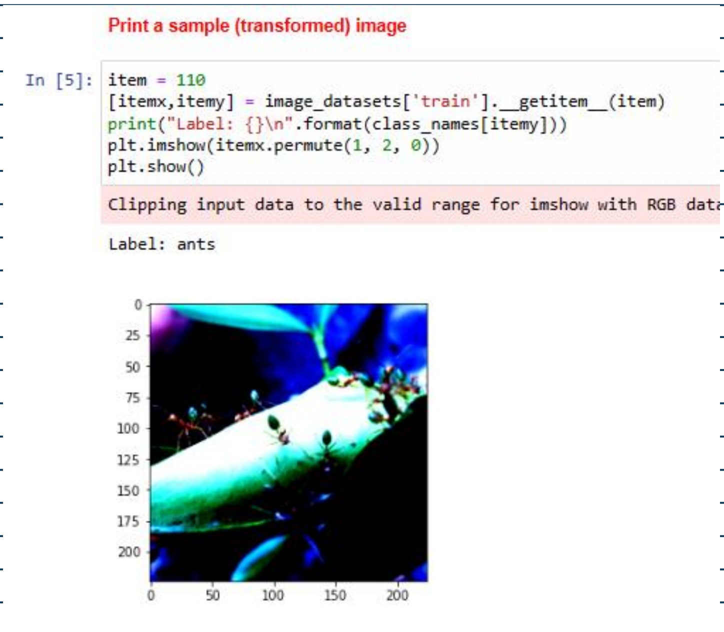
# Week 9 Programming Assignment

Sunday, March 12, 2023 6:14 PM

Question 1 - Part A

normalized vs original image

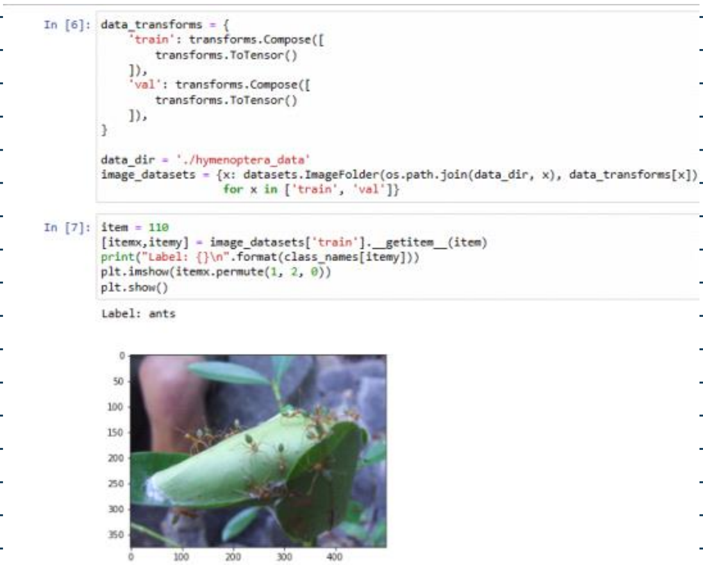


**Print a sample (transformed) image**

```
In [5]: item = 110
        [itemx,itemy] = image_datasets['train'].__getitem__(item)
        print("Label: {}\n".format(class_names[itemy]))
        plt.imshow(itemx.permute(1, 2, 0))
        plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data
```

```
Label: ants
```

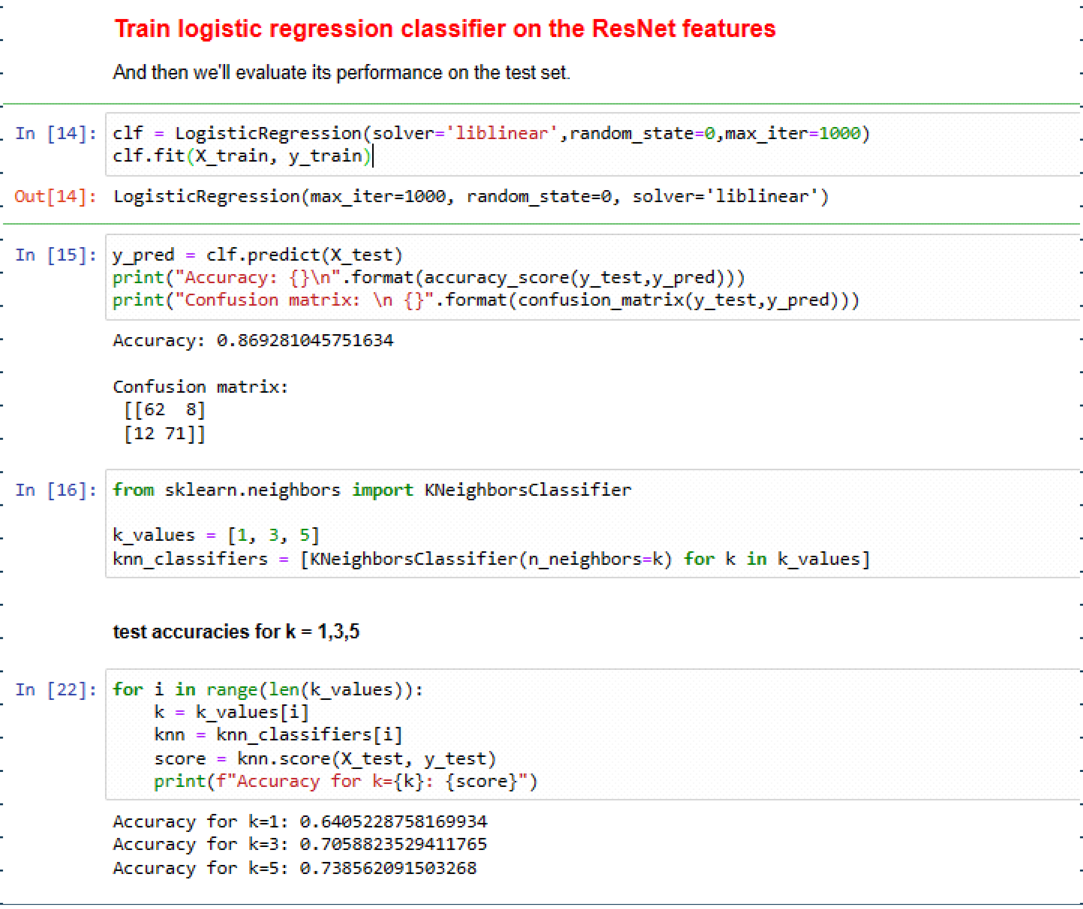


```
In [6]: data_transforms = {
            'train': transforms.Compose([
                transforms.ToTensor()
            ]),
            'val': transforms.Compose([
                transforms.ToTensor()
            ]),
        }

        data_dir = './hymenoptera_data'
        image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x), data_transforms[x])
                          for x in ['train', 'val']}
```

```
In [7]: item = 110
        [itemx,itemy] = image_datasets['train'].__getitem__(item)
        print("Label: {}\n".format(class_names[itemy]))
        plt.imshow(itemx.permute(1, 2, 0))
        plt.show()
```

```
Label: ants
```

Part B - k-nearest neighbor classifier



### Train logistic regression classifier on the ResNet features

And then we'll evaluate its performance on the test set.

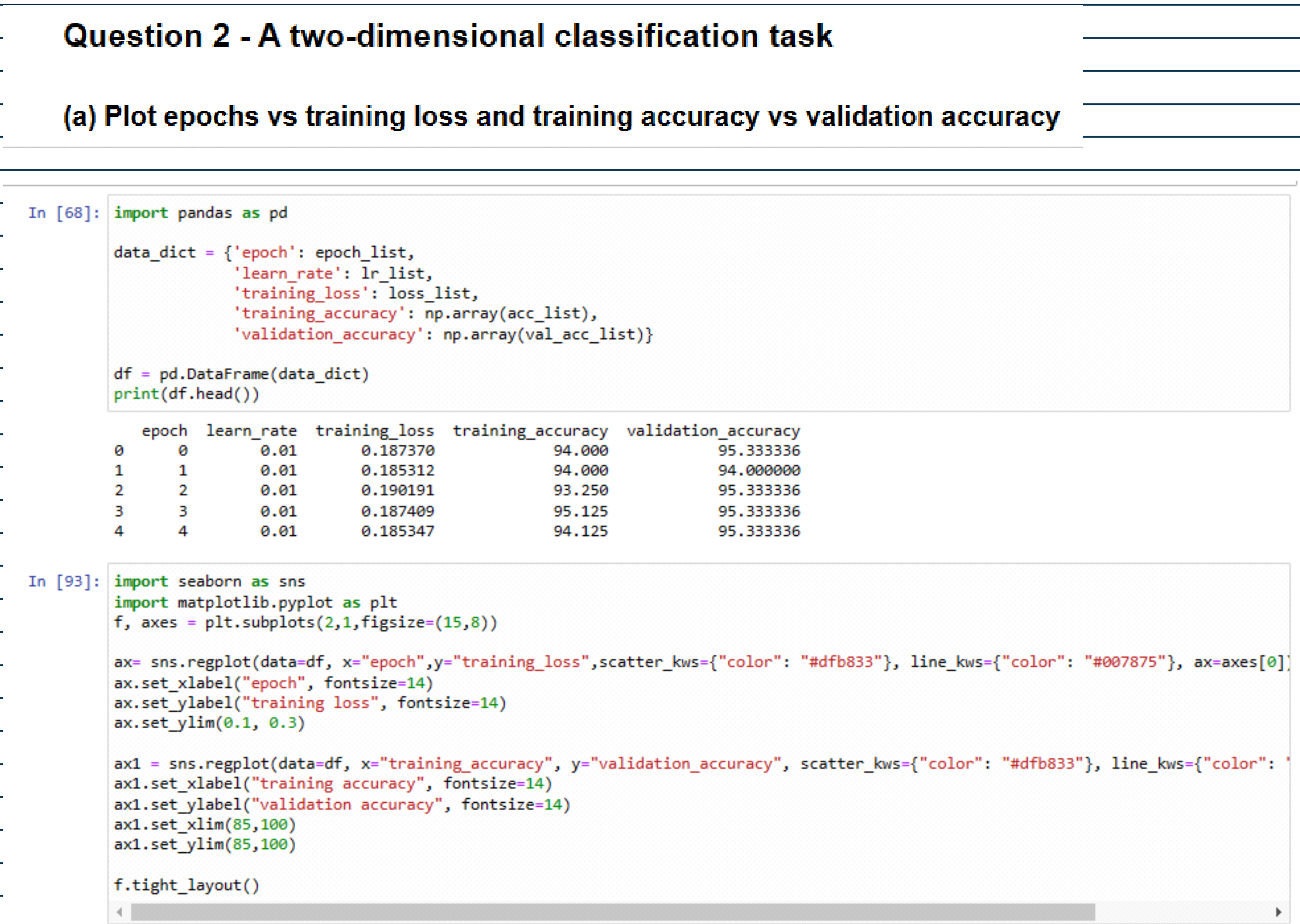
```
In [14]: clf = LogisticRegression(solver='liblinear',random_state=0,max_iter=1000)
         clf.fit(X_train, y_train)

Out[14]: LogisticRegression(max_iter=1000, random_state=0, solver='liblinear')
```

```
In [15]: y_pred = clf.predict(X_test)
         print("Accuracy: {}\n".format(accuracy_score(y_test,y_pred)))
         print("Confusion matrix: \n {}".format(confusion_matrix(y_test,y_pred)))

         Accuracy: 0.869281045751634

         Confusion matrix:
          [[62  8]
          [12 71]]
```

```
In [16]: from sklearn.neighbors import KNeighborsClassifier

         k_values = [1, 3, 5]
         knn_classifiers = [KNeighborsClassifier(n_neighbors=k) for k in k_values]
```

**test accuracies for k = 1,3,5**

```
In [22]: for i in range(len(k_values)):
             k = k_values[i]
             knn = knn_classifiers[i]
             score = knn.score(X_test, y_test)
             print(f"Accuracy for k={k}: {score}")

         Accuracy for k=1: 0.6405228758169934
         Accuracy for k=3: 0.7058823529411765
         Accuracy for k=5: 0.738562091503268
```

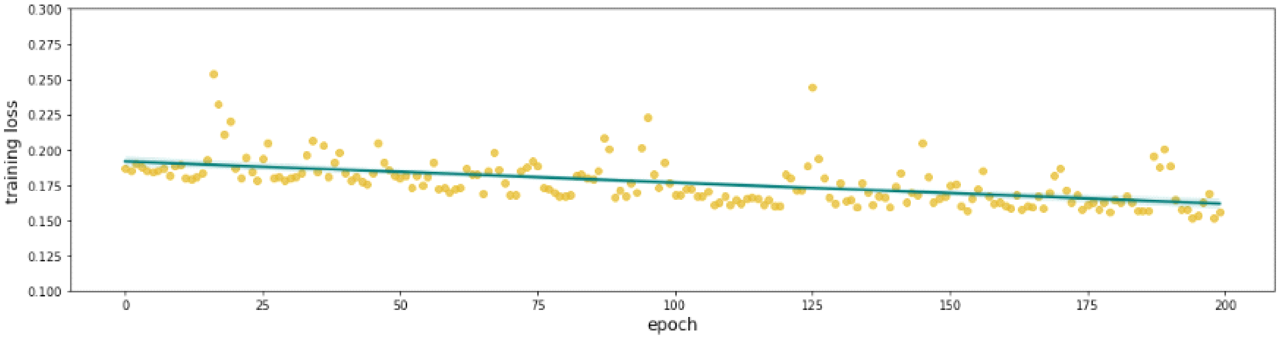


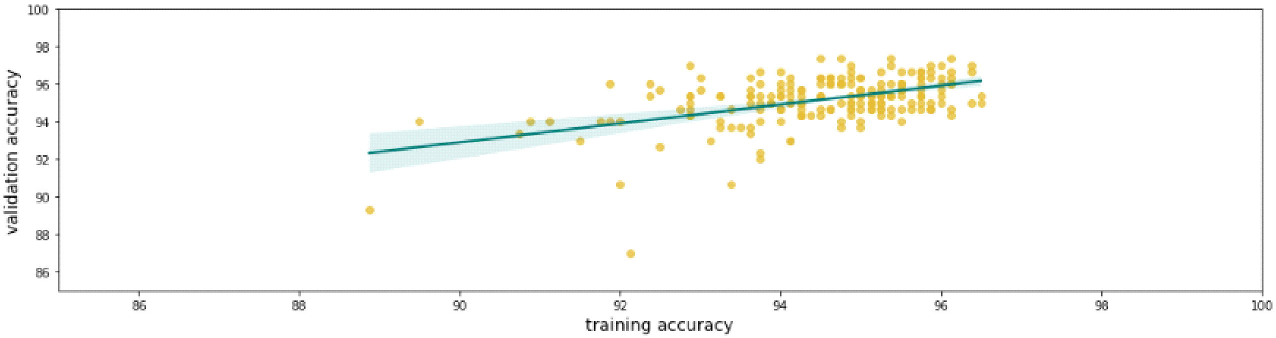
## Question 2 - A two-dimensional classification task

### (a) Plot epochs vs training loss and training accuracy vs validation accuracy

```
In [68]: import pandas as pd

         data_dict = {'epoch': epoch_list,
                      'learn_rate': lr_list,
                      'training_loss': loss_list,
                      'training_accuracy': np.array(acc_list),
                      'validation_accuracy': np.array(val_acc_list)}

         df = pd.DataFrame(data_dict)
         print(df.head())

            epoch  learn_rate  training_loss  training_accuracy  validation_accuracy
         0      0        0.01       0.187370             94.000            95.333336
         1      1        0.01       0.185312             94.000            94.000000
         2      2        0.01       0.190191             93.250            95.333336
         3      3        0.01       0.187409             95.125            95.333336
         4      4        0.01       0.185347             94.125            95.333336
```

```
In [93]: import seaborn as sns
         import matplotlib.pyplot as plt
         f, axes = plt.subplots(2,1,figsize=(15,8))

         ax= sns.regplot(data=df, x="epoch",y="training_loss",scatter_kws={"color": "#dfb833"}, line_kws={"color": "#007875"}, ax=axes[0]
         ax.set_xlabel("epoch", fontsize=14)
         ax.set_ylabel("training loss", fontsize=14)
         ax.set_ylim(0.1, 0.3)

         ax1 = sns.regplot(data=df, x="training_accuracy", y="validation_accuracy", scatter_kws={"color": "#dfb833"}, line_kws={"color": '
         ax1.set_xlabel("training accuracy", fontsize=14)
         ax1.set_ylabel("validation accuracy", fontsize=14)
         ax1.set_xlim(85,100)
         ax1.set_ylim(85,100)

         f.tight_layout()
```





### change in training loss with epochs

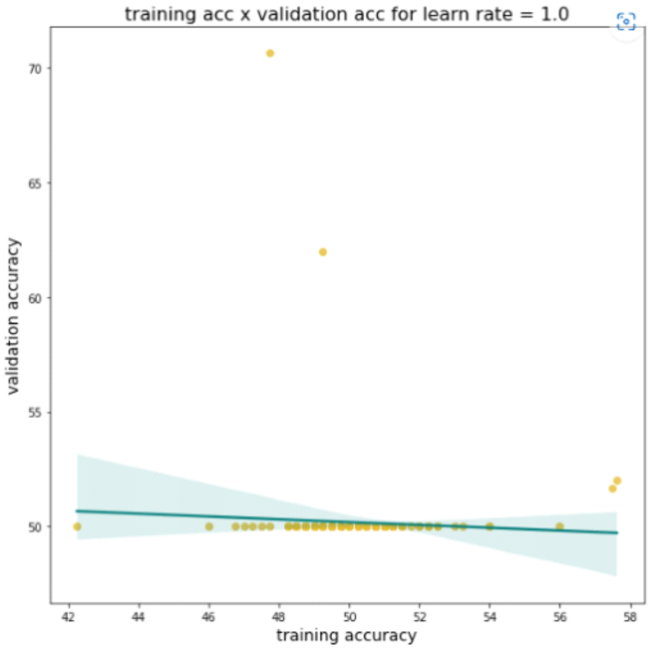
There is a gradual negative trend for training loss with epochs. It makes sense that the training loss would decrease as the model learns to make better predictions and therefore improves with each iteration.

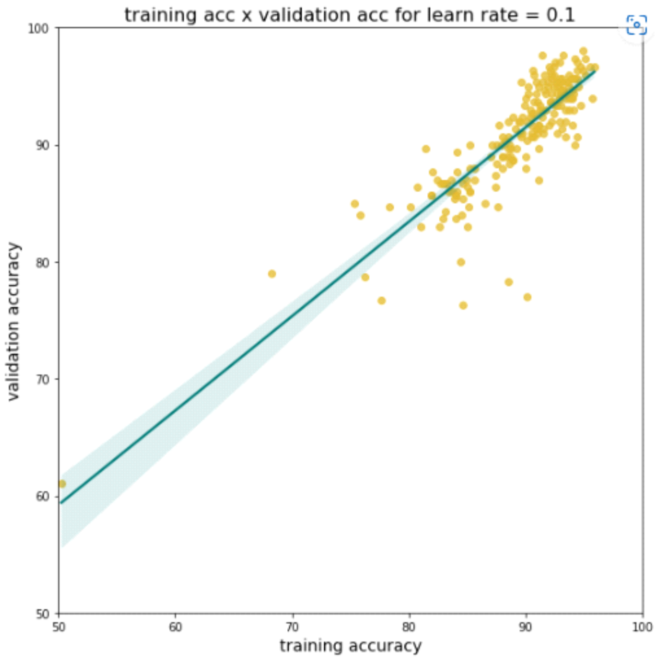
### training accuracy vs validation accuracy

There is a positive trend between validation accuracy and training accuracy. As the model improves its training accuracy, it is tweaking weights and biases to more accurately capture the training dataset. Therefore, it makes sense that as accuracy improves on the training data, it would also improve on a similar validation set.

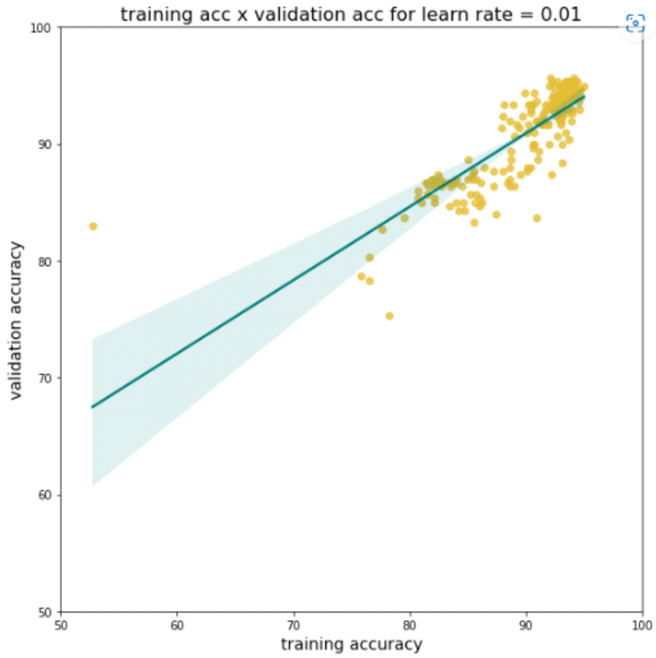
That being said, as the training data approaches 100% accuracy we would want to avoid overfitting. If the model begins overfitting the training data, we may see accuracy on the validation data begin to decrease slightly.

## (b) Effect of learning rate: lr = 1, 0.1, 0.01, 0.001, 0.0001

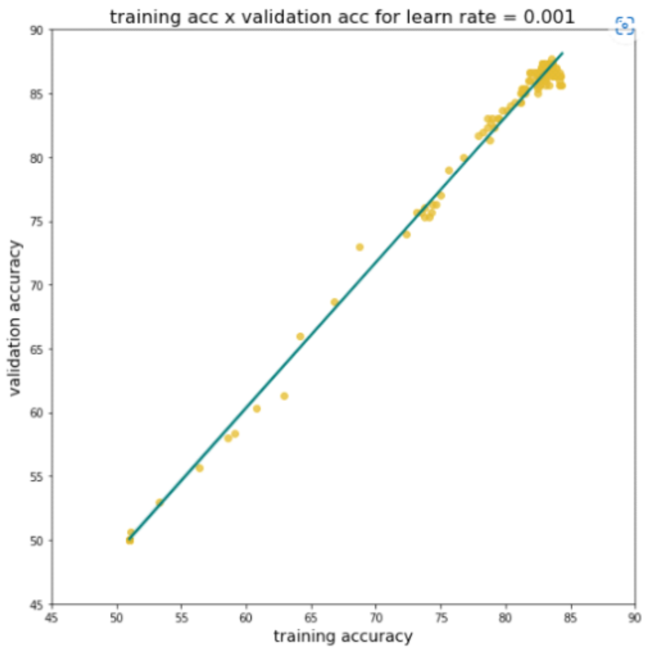


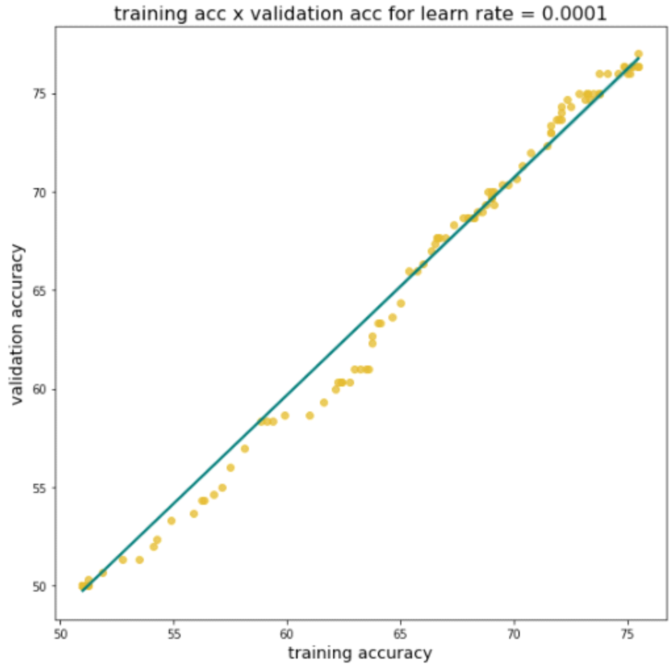


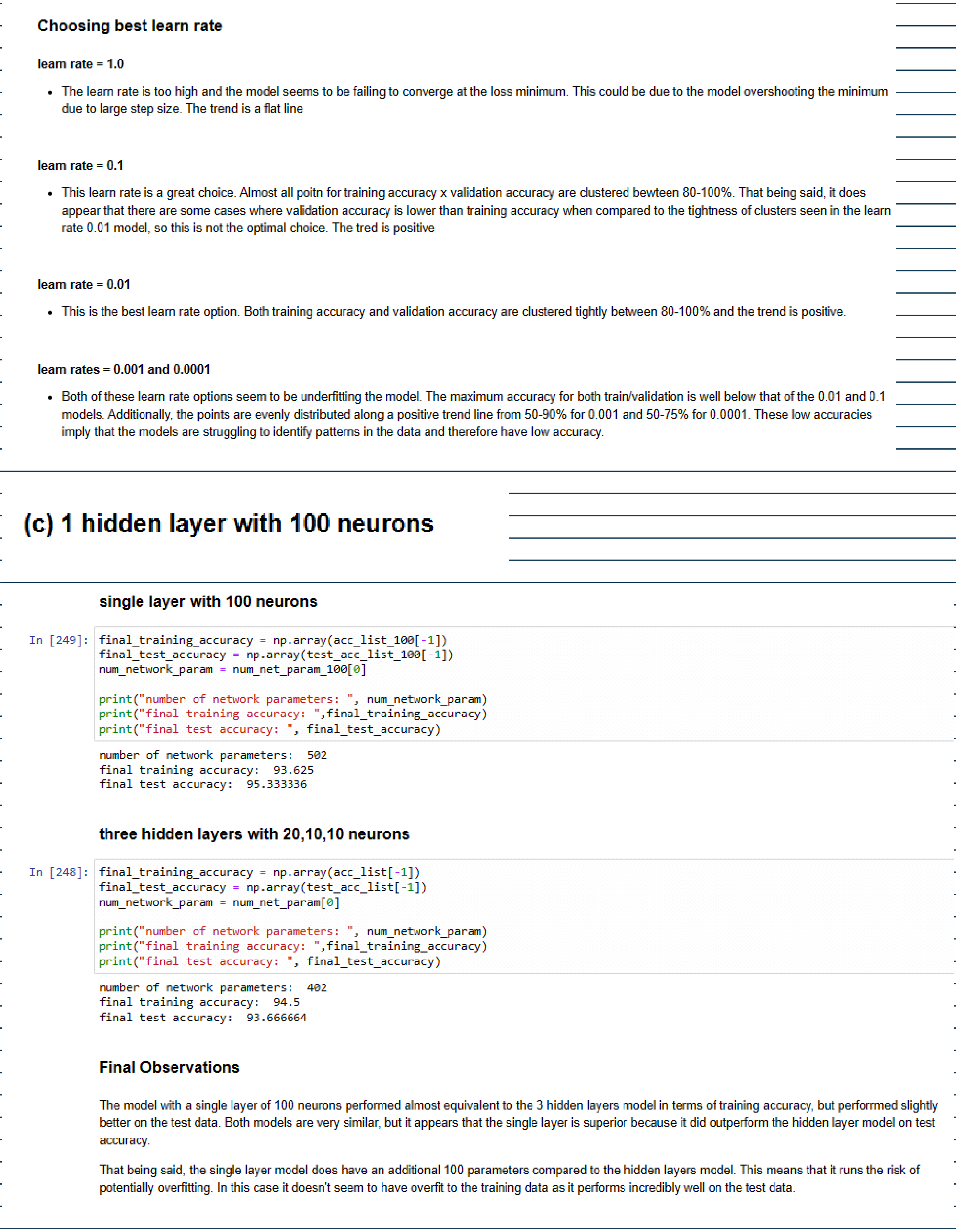
training acc x validation acc for learn rate = 0.1



training acc x validation acc for learn rate = 0.01


training acc x validation acc for learn rate = 0.001
validation accuracy
training accuracy
90
85
80
75
70
65
60
55
50
45
45
50
55
60
65
70
75
80
85
90


training acc x validation acc for learn rate = 0.0001
validation accuracy
training accuracy
75
70
65
60
55
50
50
55
60
65
70
75



### Choosing best learn rate

**learn rate = 1.0**

- The learn rate is too high and the model seems to be failing to converge at the loss minimum. This could be due to the model overshooting the minimum due to large step size. The trend is a flat line

**learn rate = 0.1**

- This learn rate is a great choice. Almost all poitn for training accuracy x validation accuracy are clustered bewteen 80-100%. That being said, it does appear that there are some cases where validation accuracy is lower than training accuracy when compared to the tightness of clusters seen in the learn rate 0.01 model, so this is not the optimal choice. The tred is positive

**learn rate = 0.01**

- This is the best learn rate option. Both training accuracy and validation accuracy are clustered tightly between 80-100% and the trend is positive.

**learn rates = 0.001 and 0.0001**

- Both of these learn rate options seem to be underfitting the model. The maximum accuracy for both train/validation is well below that of the 0.01 and 0.1 models. Additionally, the points are evenly distributed along a positive trend line from 50-90% for 0.001 and 50-75% for 0.0001. These low accuracies imply that the models are struggling to identify patterns in the data and therefore have low accuracy.

# (c) 1 hidden layer with 100 neurons

### single layer with 100 neurons

```
In [249]: final_training_accuracy = np.array(acc_list_100[-1])
          final_test_accuracy = np.array(test_acc_list_100[-1])
          num_network_param = num_net_param_100[0]

          print("number of network parameters: ", num_network_param)
          print("final training accuracy: ",final_training_accuracy)
          print("final test accuracy: ", final_test_accuracy)
```

```
number of network parameters:  502
final training accuracy:  93.625
final test accuracy:  95.333336
```

### three hidden layers with 20,10,10 neurons

```
In [248]: final_training_accuracy = np.array(acc_list[-1])
          final_test_accuracy = np.array(test_acc_list[-1])
          num_network_param = num_net_param[0]

          print("number of network parameters: ", num_network_param)
          print("final training accuracy: ",final_training_accuracy)
          print("final test accuracy: ", final_test_accuracy)
```

```
number of network parameters:  402
final training accuracy:  94.5
final test accuracy:  93.666664
```

### Final Observations

The model with a single layer of 100 neurons performed almost equivalent to the 3 hidden layers model in terms of training accuracy, but performed slightly better on the test data. Both models are very similar, but it appears that the single layer is superior because it did outperform the hidden layer model on test accuracy.

That being said, the single layer model does have an additional 100 parameters compared to the hidden layers model. This means that it runs the risk of potentially overfitting. In this case it doesn't seem to have overfit to the training data as it performs incredibly well on the test data.