Report

Cameron Mattson

February 7, 2022

1 Neural Network Hyperparameters

1.1 Code

[1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```
# Dataset: https://archive.ics.uci.edu/ml/datasets/banknote+authentication
data = np.genfromtxt ('data_banknote_authentication.csv', delimiter=",")
X = data[:,:-1]
y = data[:,-1].astype(np.int64)
X.shape
```

[2]:
```
(1372, 4)
```

[3]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=1,train_size =␣
↪0.8, shuffle = True)
```

[4]:
```
stdsc = StandardScaler()
stdsc.fit(X_train)
X_train_ss = stdsc.transform(X_train)
X_test_ss = stdsc.transform(X_test)
```

[5]:
```
hidden_layers = [1,4]
num_neurons = [5,50]
activation_funcs = ['tanh','relu']
batchs = 200
maxiter = 50
models = []
hyperparams = []
for hidden in hidden_layers:
```

```
    for neurons in num_neurons:
        layer_list = [neurons] * hidden
        for func in activation_funcs:
            hyperparams.append([layer_list,func])
            models.append(MLPClassifier(activation=func,random_state=1, \
            hidden_layer_sizes=layer_list,max_iter=maxiter,batch_size=batchs))

model_conf = []
for ind,model in enumerate(models):
    model.fit(X_train_ss, y_train)
    conf_mat = confusion_matrix(y_test,model.predict(X_test_ss))
    model_conf.append(conf_mat)
    tit_str1 = f'Hidden Layers =␣
 ↪{hyperparams[ind][0][0]}(x{len(hyperparams[ind][0])})\n'
    tit_str2 = f'Activation Function = {hyperparams[ind][1]}\n'
    tit_str3 = f'Accuracy = {model.score(X_test_ss, y_test)}'
    tit_str = tit_str1 + tit_str2 + tit_str3
    myax = sns.heatmap(conf_mat.T, square = True, annot = True, fmt = 'd', \
    cbar = False).set(title=tit_str,xlabel='True Label', ylabel='Predicted␣
 ↪Label')
    plt.show()
```

1.2 Write-up

To implement this experiment, $2^3 = 8$ models were trained and tested using a combination of 3 hyperparameters, where each hyperparameter could be one of two possible values. The two possible values for each hyperparameter are given below:

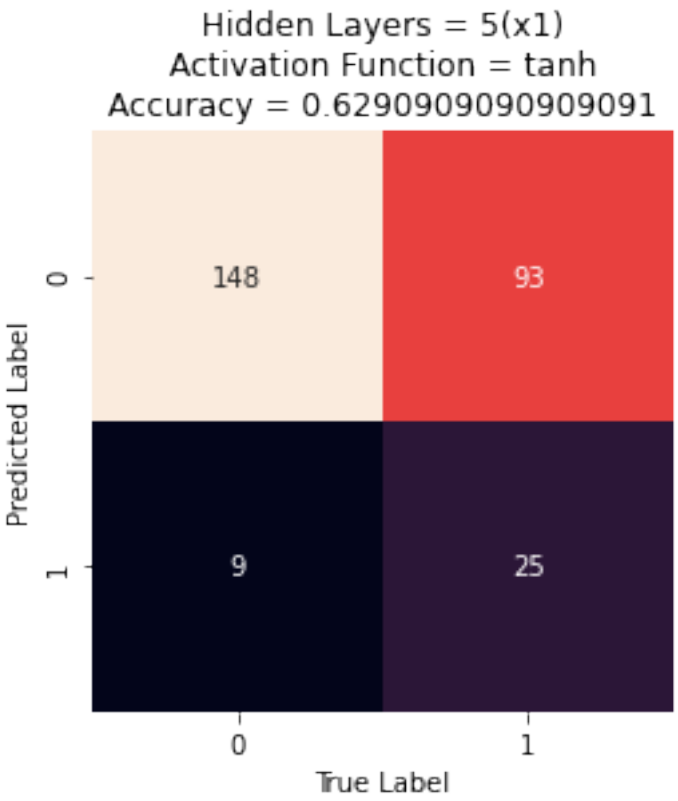
Number of hidden layers $\in \{1, 4\}$
Number of neurons per hidden layer $\in \{5, 50\}$
Types of activation functions $\in \{\text{tanh, relu}\}$

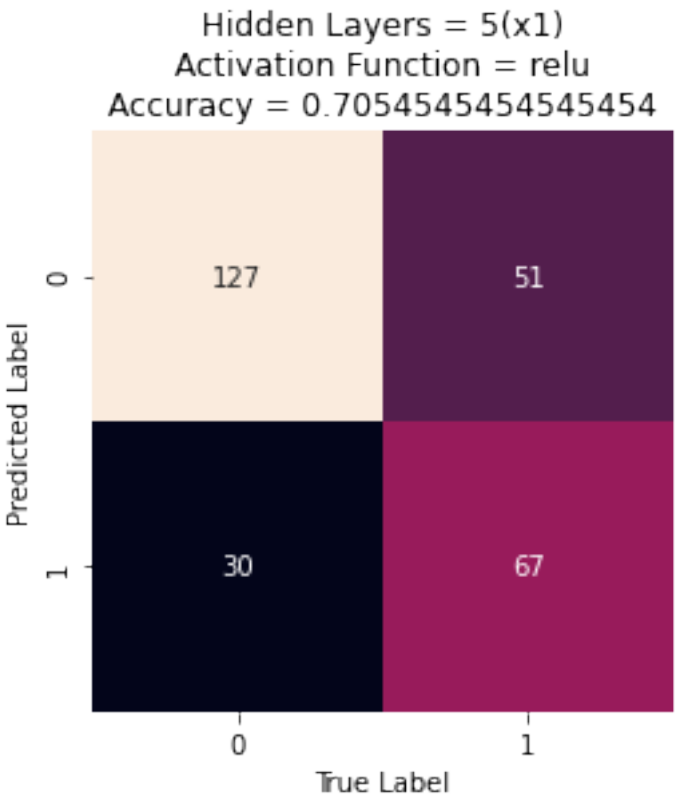
In other words, for the number of hidden layers the two possible values were 1 or 4. For the number of neurons per hidden layers the two possible values were 5 or 50. The 2 types of activation functions used were tanh or relu. The dataset for this experiment came from the University of California, Irvine's Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/banknote+authentication). In total, there were 1372 samples available for this dataset. The image attributes were taken from genuine and forged banknotes to predict wheather a given banknote was genuine or not. The attributes are listed below:

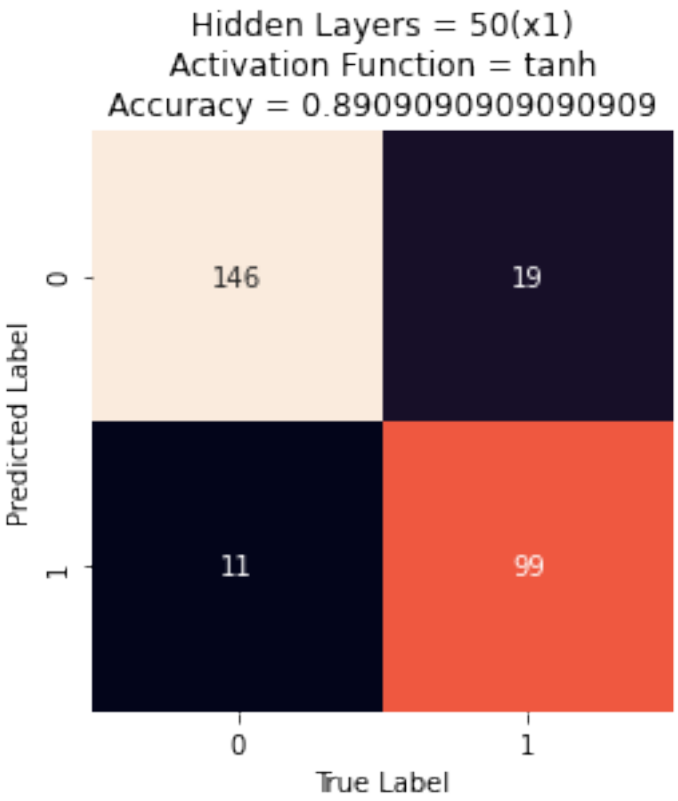
1. Variance of Wavelet Transformed image (continuous)
2. Skewness of Wavelet Transformed image (continuous)
3. Curtosis of Wavelet Transformed image (continuous)
4. Entropy of image (continuous)

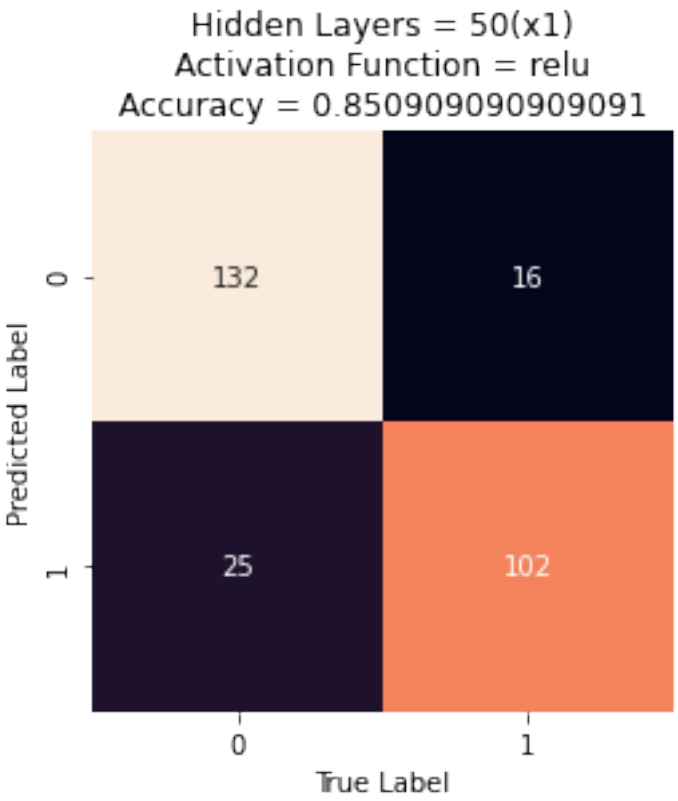
Each model used the same batch size (200), and number of iterations (10). The data was also standardized by calculating the standard score of each sample.

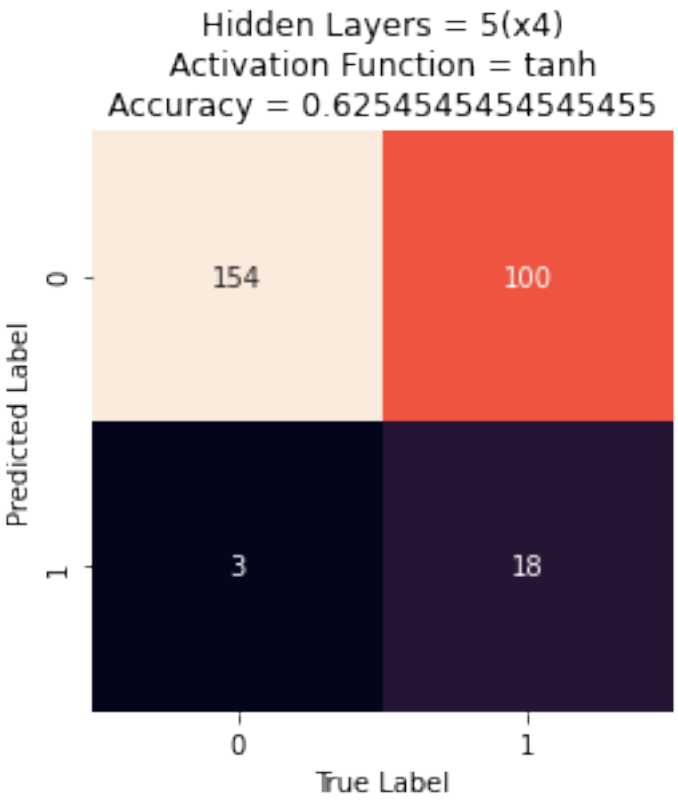
Nomenclature used to disern models will look similar to this in structure: 5(x1), where the number 5 outside the brackets indicates the number of neurons per layer, and the number inside the brackets indicates the number of hidden layers.

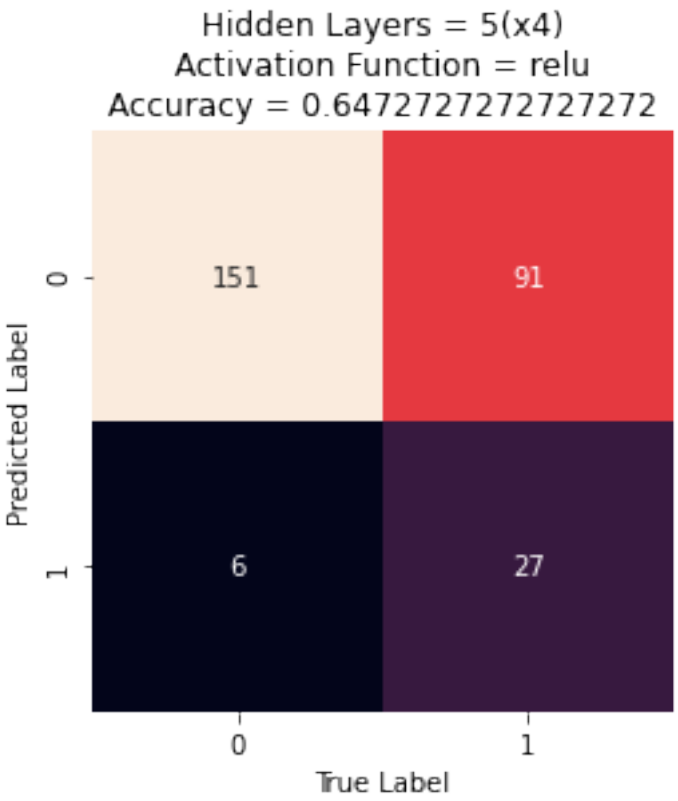

Hidden Layers = 5(x1)
Activation Function = tanh
Accuracy = 0.6290909090909091
148
93
9
25
0
1
Predicted Label
0
1
True Label

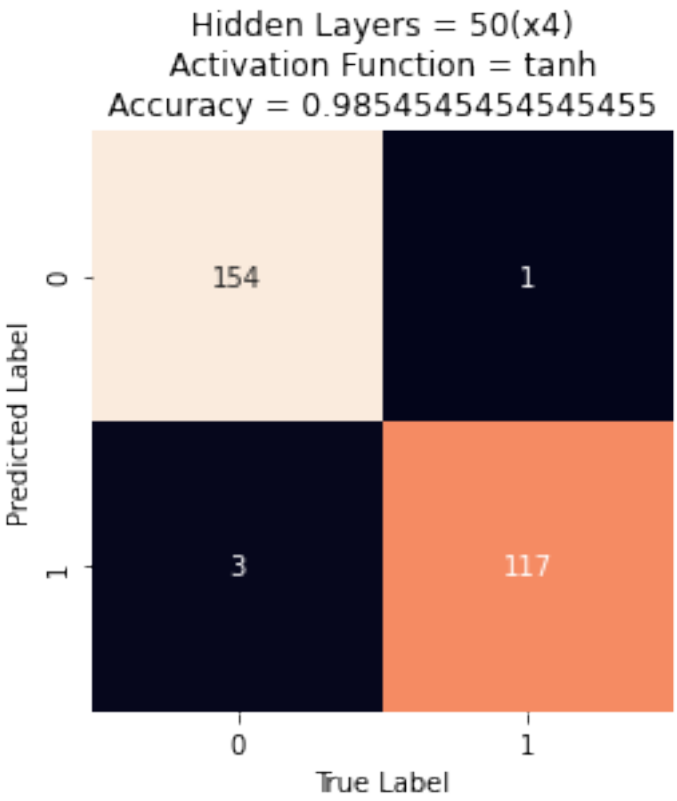

Hidden Layers = 5(x1)
Activation Function = relu
Accuracy = 0.7054545454545454
127
51
30
67
0
1
Predicted Label
0
1
True Label

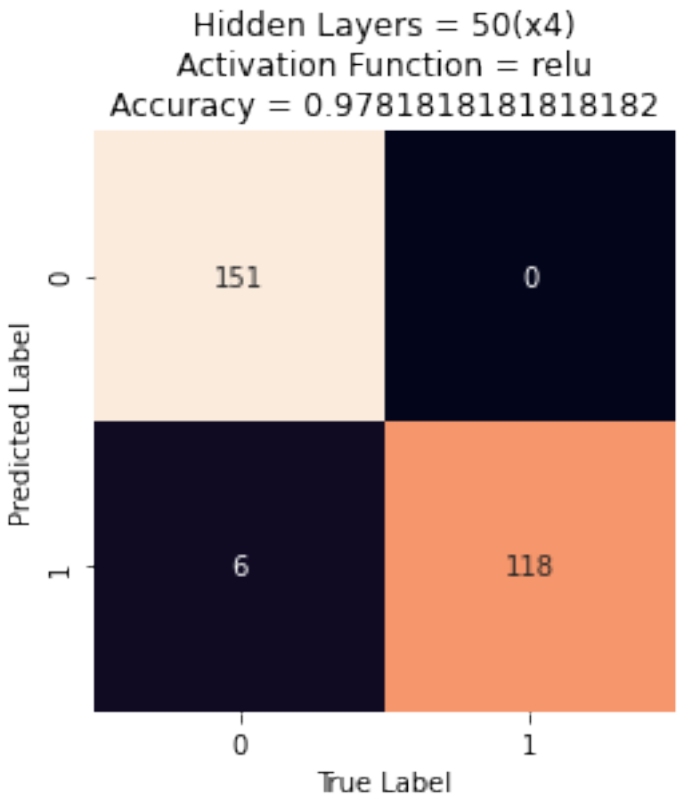

Hidden Layers = 50(x1)
Activation Function = tanh
Accuracy = 0.8909090909090909
146
19
11
99
0
1
Predicted Label
0
1
True Label


Hidden Layers = 50(x1)
Activation Function = relu
Accuracy = 0.850909090909091
132
16
25
102
0
1
Predicted Label
0
1
True Label


Hidden Layers = 5(x4)
Activation Function = tanh
Accuracy = 0.6254545454545455
154
100
3
18
0
1
Predicted Label
0
1
True Label


Hidden Layers = 5(x4)
Activation Function = relu
Accuracy = 0.6472727272727272
151
91
6
27
0
1
Predicted Label
0
1
True Label


Hidden Layers = 50(x4)
Activation Function = tanh
Accuracy = 0.9854545454545455
154
1
3
117
0
1
Predicted Label
0
1
True Label



When comparing both the relu and the tanh activation functions accross these models they seem to achieve similar accuracies with one exception. In the models with 5(x1) hidden layers, the tanh function performs almost 8% better than the relu function using the accuracy metric. I don't think this is because of overfitting, because other models had the same number of layers and more neurons per layer, which would make these model more likely to overfit. One reason this model's accuracy is smaller may be due to the fact that the tanh function's parameters are not changing as quickly for positive values. This means that the model may not have learned to generalize as well from the training data compared to the relu model with the limited number of epochs. This same phenomenon didn't seem to occur with the other models that have more neurons per layer, or more layers, because they are able to compensate for the limited number of epochs by training more neurons per layer or by traing with more layers. The maximum number of layers allowed in this expirement is much smaller than the maximum number of neurons per layer, which may be the reason that the tanh model with 5(x4) hidden layers has a slightly smaller accuracy than the relu model with 5(x4) hidden layers.

Another pattern that I noticed is that the tanh 50(x1) model has a slightly better accuracy than relu 50(x1) model. This may be the case because relu model prameters change quicker with positive values. Therefore if many of these parameter change too quickly than this error may compound leading to a reduced accuracy. Alternatively, in tanh models, the gradient decreases as values become more positive, which would lead to smaller changes per iteration for positive values. Similarly, I think this is the reason that the 5(x4) relu model performs worse than the 5(x1) relu model in terms of accuracy. This is because the error propogates across multiple layers and there are not as many neurons per layer to "pad" or reduce this error such as with the 50(x4) relu

model. The accuracy could also be better in the 50(x4) model as compared to the 5(x4) model, because there are more bias neurons.

In general, the models with more neurons per layer tend to perform better than models with less neurons per layer when using accuracy as a metric. The same relationship is apparent when controlling for everything except the number of layers, where accuracy either increases or remains relatively constant with the number of layers. The exception to this trend was mentioned above, which compared the 5(x4) and the 5(x1) relu models.

2 Impact of Training Duration and Training Data

2.1 Code

[9]:
```
stdsc = StandardScaler()
stdsc.fit(X_train)
X_train_ss = stdsc.transform(X_train)
X_test_ss = stdsc.transform(X_test)
```

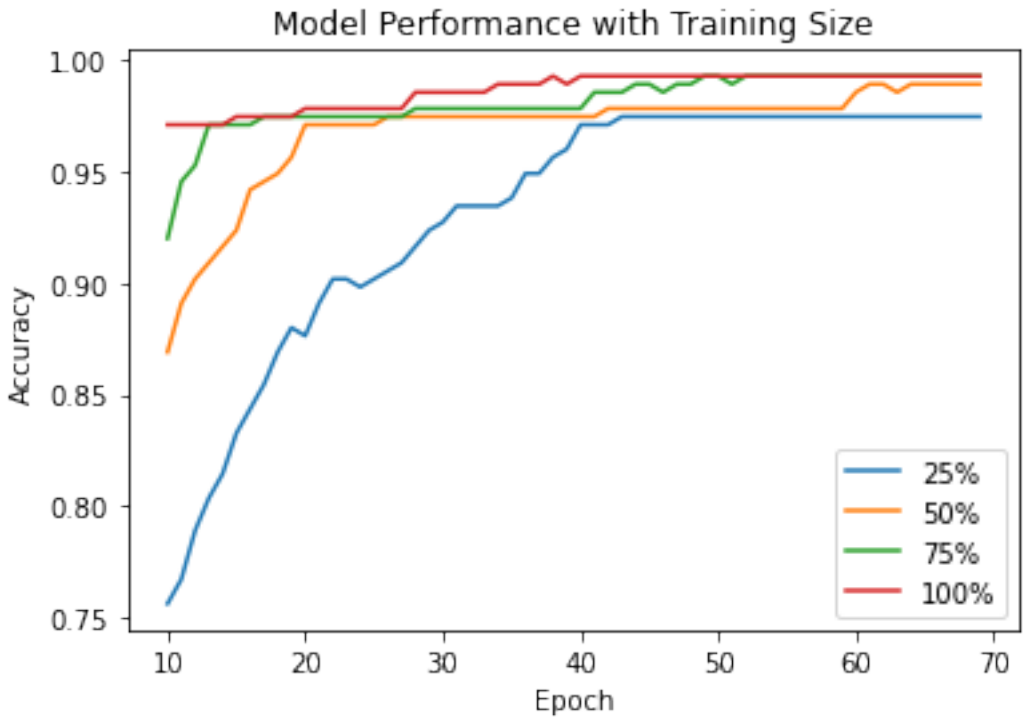
[10]:
```
layer_size = 10
num_layers = 2
hidden_layers = [layer_size] * num_layers
activation_func = 'tanh'
batchs = 50
start_epoch = 10
inc_epoch = 1
fin_epoch = 70
ran_epoch = np.arange(start_epoch,fin_epoch,inc_epoch)

train_per = [1/4,1/2,3/4,1]
trainnum = []

for percent in train_per:
    acc_epoch = []
    X_trainmod,X_dum,y_trainmod,y_dum =␣
 ↪train_test_split(X_train_ss,y_train,random_state=1,train_size = percent)
    if (percent == 1):
        X_trainmod = X_train_ss
        y_trainmod = y_train
    for cepoch in range(start_epoch,fin_epoch,inc_epoch):
        mlpclass = MLPClassifier(activation=activation_func,random_state=1,␣
 ↪hidden_layer_sizes=hidden_layers,max_iter=cepoch,batch_size=batchs)
        mlpclass.fit(X_trainmod, y_trainmod)
        #print(X_test_ss.shape,y_test.shape)
        acc_epoch.append(mlpclass.score(X_test_ss, y_test))
    trainnum.append(acc_epoch)
```

2.2 Write-up

To implement this experiment, 4 models train with 25%, 50%, 75%, and 100% of the training data by incrementing the number epochs for each of these models by 1, from 10 epochs to 70 epochs. The accuracy for each of these models was updated after each epoch. The dataset for this experiment came from the University of California, Irvine's Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/banknote+authentication). In total, there were 1372 samples available for this dataset. The image attributes were taken from genuine and forged banknotes to predict wheather a given banknote was genuine or not. The attributes are listed below:

1. Variance of Wavelet Transformed image (continuous)
2. Skewness of Wavelet Transformed image (continuous)
3. Curtosis of Wavelet Transformed image (continuous)
4. Entropy of image (continuous)

Each model used the same batch size (50), the same number of layers (2), the same number of neurons per layer (10), and the same activation function (tanh). The data was standardized by calculating the standard score of each sample.



Looking at each model, the accuracy is more widely distributed for less epochs and tends to be

closer with more epochs. Models that have less training data tend to be less accurate for any given epoch. Similarly, the accuracies of models with more training data tend to change less with each epoch compared to models with less training data. I think this is because the models with more training data have more information to generalize from, and therefore, can make better predictions without training as often, whereas models with less training data need to iterate more often to try to learn more from the information available by iterating more times.

The convergenece of each model also seems to depend on the amount of training data. If less data is available for a model to train from, the model may not represent our population well enough to achieve a desired accuracy, regardless of the number of epochs used in training. This explains the reason that the models with less training data seem to coverge to smaller accuracies, in general, compared to models with more training data.

3 All Code

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
# Dataset: https://archive.ics.uci.edu/ml/datasets/banknote+authentication
data = np.genfromtxt ('data_banknote_authentication.csv', delimiter=",")
X = data[:,:-1]
y = data[:,-1].astype(np.int64)
X.shape
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=1,train_size = 0.8, shuffle =
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=1,train_size = 0.8, shuffle =
stdsc = StandardScaler()
stdsc.fit(X_train)
X_train_ss = stdsc.transform(X_train)
X_test_ss = stdsc.transform(X_test)
hidden_layers = [1,4]
num_neurons = [5,50]
activation_funcs = ['tanh','relu']
batchs = 200
maxiter = 10
models = []
hyperparams = []
for hidden in hidden_layers:
    for neurons in num_neurons:
        layer_list = [neurons] * hidden
        for func in activation_funcs:
            hyperparams.append([layer_list,func])
            models.append(MLPClassifier(activation=func,random_state=1, \
            hidden_layer_sizes=layer_list,max_iter=maxiter,batch_size=batchs))

model_conf = []
for ind,model in enumerate(models):
    model.fit(X_train_ss, y_train)
    conf_mat = confusion_matrix(y_test,model.predict(X_test_ss))
    model_conf.append(conf_mat)
    tit_str1 = f'Hidden Layers = {hyperparams[ind][0][0]}(x{len(hyperparams[ind][0])})\n'
    tit_str2 = f'Activation Function = {hyperparams[ind][1]}\n'
    tit_str3 = f'Accuracy = {model.score(X_test_ss, y_test)}'
    tit_str = tit_str1 + tit_str2 + tit_str3
    myax = sns.heatmap(conf_mat.T, square = True, annot = True, fmt = 'd', \
    cbar = False).set(title=tit_str,xlabel='True Label', ylabel='Predicted Label')
    plt.show()
```

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
# Dataset: https://archive.ics.uci.edu/ml/datasets/banknote+authentication
data = np.genfromtxt ('data_banknote_authentication.csv', delimiter=",")
X = data[:,:-1]
y = data[:,-1].astype(np.int64)
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=1,train_size = 0.8, shuffle =
stdsc = StandardScaler()
stdsc.fit(X_train)
X_train_ss = stdsc.transform(X_train)
X_test_ss = stdsc.transform(X_test)
layer_size = 10
num_layers = 2
hidden_layers = [layer_size] * num_layers
activation_func = 'tanh'
batchs = 50
start_epoch = 10
inc_epoch = 1
fin_epoch = 70
ran_epoch = np.arange(start_epoch,fin_epoch,inc_epoch)

train_per = [1/4,1/2,3/4,1]
trainnum = []

for percent in train_per:
    acc_epoch = []
    X_trainmod,X_dum,y_trainmod,y_dum = train_test_split(X_train_ss,y_train,random_state=1,train
    if (percent == 1):
        X_trainmod = X_train_ss
        y_trainmod = y_train
    for cepoch in range(start_epoch,fin_epoch,inc_epoch):
        mlpclass = MLPClassifier(activation=activation_func,random_state=1, hidden_layer_sizes=h
        mlpclass.fit(X_trainmod, y_trainmod)
        #print(X_test_ss.shape,y_test.shape)
        acc_epoch.append(mlpclass.score(X_test_ss, y_test))
    trainnum.append(acc_epoch)
plt.figure()
plt.plot(ran_epoch,trainnum[0],label='25%')
plt.plot(ran_epoch,trainnum[1],label='50%')
plt.plot(ran_epoch,trainnum[2],label='75%')
plt.plot(ran_epoch,trainnum[3],label='100%')
plt.xlabel('Epoch')
```

```
plt.ylabel('Accuracy')
plt.title('Model Performance with Training Size')
plt.legend()
plt.show()
```