	Version 2.2		
I)			
1.	**readme.txt**	**User guide and Anaconda Installation guide.** **PLEASE USE Tensorflow 1.x or above with it's Keras instead ver 2.0 or above.**	
2.	**problem1-trainTestNall.py**	**Tasks 1: Scatter Plots (Paired)**	
3.	**problem2.py**	**Tasks 2: Recognition by the features**	
4.	**problem3.py**	**Tasks 3: For** **a) The [256, 6, 2, 1] version by setting NB_NEURON=6** **b) The [256, 3, 2, 1] version by setting NB_NEURON=3**	
5.	**problem4-1DFF.py** **problem4-2CNN.py**	**Tasks 4-1: Deep Feedforward with around 97% recognition accuracy** **Tasks 4-2: CNN – with a model successfully over 99.5%**	
6.	**table.xls**	**A table for mean and standard deviation calculations**	
II)	ADDITION Tools:	More details in the Appendix section at the last page of this article.	
a.	Problem4-3PredictTesting.py	Load Neural Net model and predict on Verify.txt	
b.	Verify.txt	A dataset for verification	

By Hoson LAM, Copyright 2024

Abstract

The experiments in this report explored and recognised a challenging image dataset provided. It starts with some scatter plots on the provided image features and recognition of the features. More experiments were done on recognizing raw pixels by deep feedforward neural network and convolution neural network in getting better results. The recognition success rates are gradually increased from less than 60% by simple methods to more sophistic methods around 97% and finally over 99%.

The best recognition success rate achieved is over 99.5% (<0.005 error rate) by one of the three folds in the final method of task 4.

Introduction

Image recognition is one of the biggest applications of machine learning. As images are two-dimensional data which have spatial correlation and can be generated by various kind of physical processes behind. It is difficult to be recognized well with recognition accuracy close to a human being. Historically it was done by algorithmic approach and recently big success is created by machine learning, in particular, deep convolution neural networks.

The basic building block is also a perceptron similar to the PLA in assignment 1 as below. Figure 1.1 is an example of a typical perceptron [9].

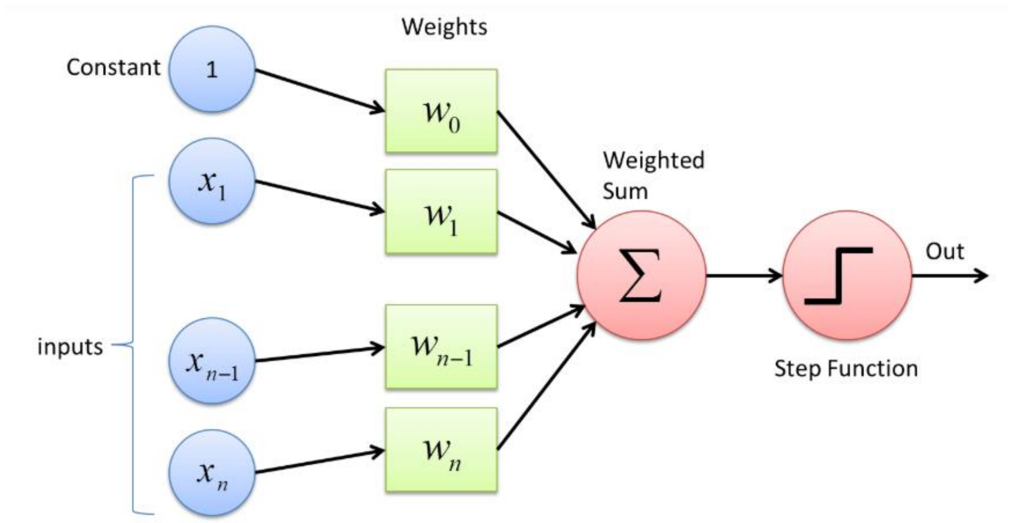


Figure 1.1

The perceptron is a modern but simplified version of biological inspired "neocognitron" [4] created by Kunihiko Fukushima back to the early ’80s. When cascading similar networks in parallel and series, shadow or deep network can be formed to solve different kinds of learning problems. The experiments used one to multi-layers neural network, figure 1 shows a generalized setting for multi-layer networks.

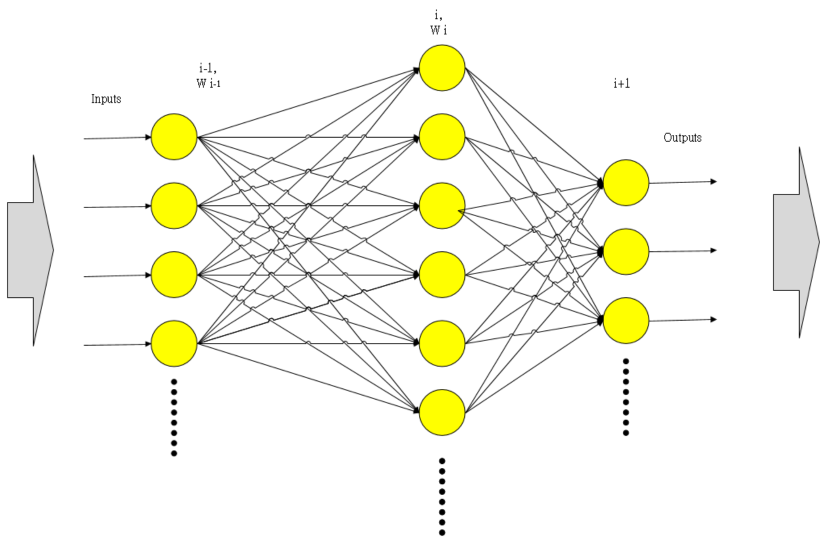


Figure 2. Hierarchical neural network build from PLA like units

A neural network is a hierarchical composite model where each layer (figure 2) applies a linear transformation followed by a non-linear function[3] to the preceding layer.

Let $\mathbf{X} \in \mathbb{R}^{N\ x\ D}$ be the input data, where each row of **X** is a D-dimensional data point and N is the number of training sample in the image space and $W^i \in \mathbb{R}^{d_{i-1}\ x\ d_i}$ be a matrix representing a linear transformation applied to the output of layer i-1, $X_{i-1} \in \mathbb{R}^{N\ x\ d_{i-1}}$, to obtain a d_i dimensional representation $X_{i-1}W^i \in \mathbb{R}^{N\ x\ d_i}$ at layer i. For example, each column of W^i could represent a convolution with some filter in convolutional neural networks or the application of a linear classifier in fully connected networks.

Let $\psi_i: \mathbb{R} \rightarrow \mathbb{R}$ be a non-linear activation function, e.g. a sigmoid $\psi_i(x) = 1/(1+e^{-x})$ or ReLu Rectified Linear unit $\psi_i(x) = \max\{0, x\}$ or Sign. This non-linearity is applied to each entry of $X_{i-1}W^i$ to generate the i-th layer of a neural network as $X_i = \psi_i(X_{i-1}W^i)$. The output X_I of the network is thus given by:

$$\boldsymbol{\mathcal{K}}(\mathbf{X}, \mathbf{W^1} \ldots\ldots \mathbf{W^I}) = \boldsymbol{\psi_I}(\boldsymbol{\psi_{I-1}}(\ldots\ldots..\boldsymbol{\psi_2}(\boldsymbol{\psi_1}(\boldsymbol{XW^1})\, \boldsymbol{W^2}) \ldots\ldots\ldots\ \boldsymbol{W^{I-1}})\boldsymbol{W^I}) \text{ --- Equation 1}$$

When the output dimension of the network is C that equal to d_I, $\boldsymbol{\mathcal{K}}$ will be an N x C matrix and C is the number of classes for a pattern recognition problem. Notice also that the mapping can be seen as a function of all the network weights $\mathbf{W} = \{W^i\}_{i=1}^{I}$ with a fixed input **X** and where the I layer is the last layer, output layer, with the indexed number I. For 0-9 ten classes, C=10.

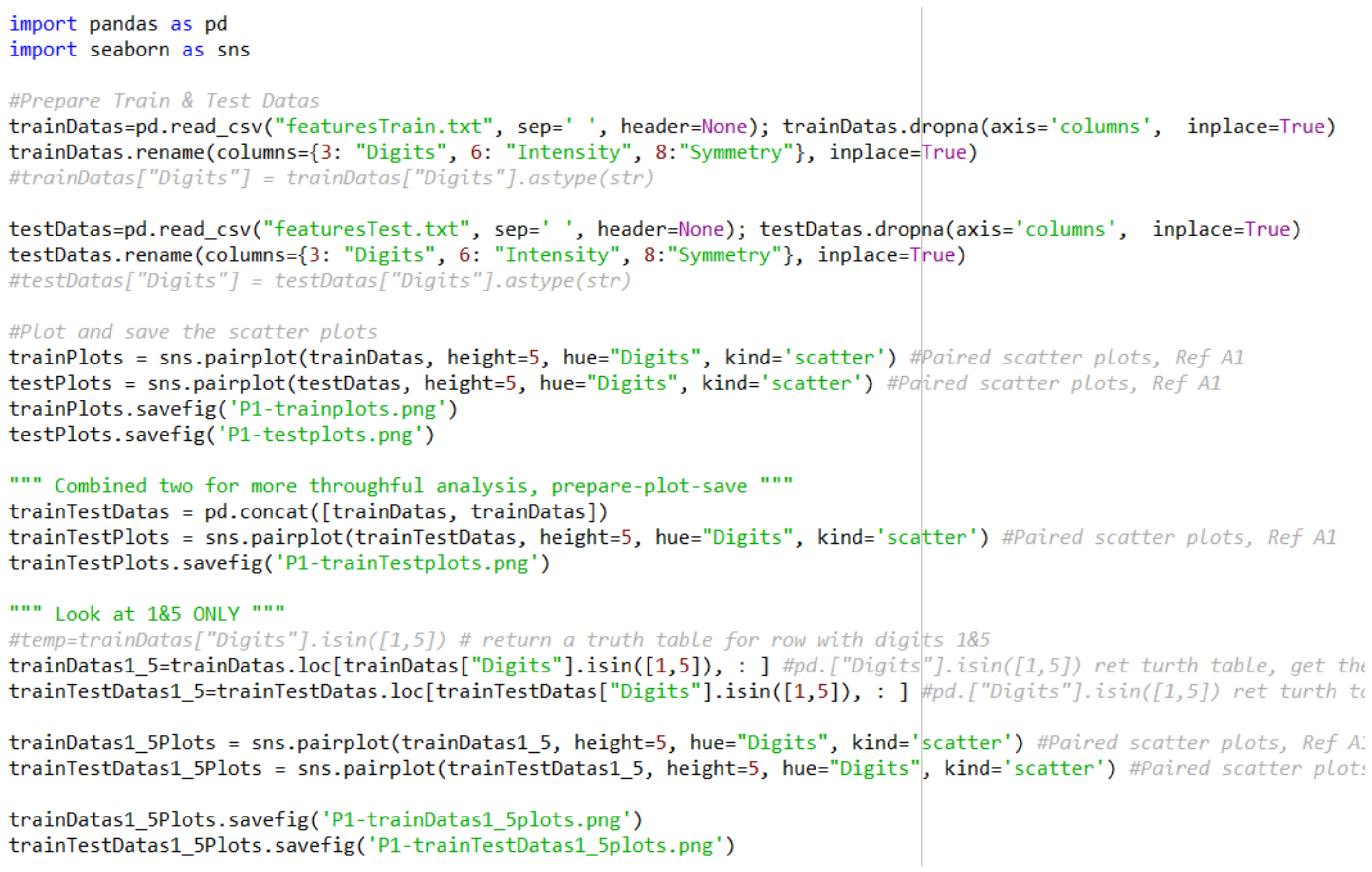
Tasks 1:

Details

Scatter plot of two features for the ten-digit images with labels (0-9). Feature 1 is the intensity and feature 2 is the symmetry.

Codes and Results

Source code is Problem1-trainTestNall.py which shows paired scatter plots one by one. Following is a snapshot of the code.



```
import pandas as pd
import seaborn as sns

#Prepare Train & Test Datas
trainDatas=pd.read_csv("featuresTrain.txt", sep=' ', header=None); trainDatas.dropna(axis='columns',  inplace=True)
trainDatas.rename(columns={3: "Digits", 6: "Intensity", 8:"Symmetry"}, inplace=True)
#trainDatas["Digits"] = trainDatas["Digits"].astype(str)

testDatas=pd.read_csv("featuresTest.txt", sep=' ', header=None); testDatas.dropna(axis='columns',  inplace=True)
testDatas.rename(columns={3: "Digits", 6: "Intensity", 8:"Symmetry"}, inplace=True)
#testDatas["Digits"] = testDatas["Digits"].astype(str)

#Plot and save the scatter plots
trainPlots = sns.pairplot(trainDatas, height=5, hue="Digits", kind='scatter') #Paired scatter plots, Ref A1
testPlots = sns.pairplot(testDatas, height=5, hue="Digits", kind='scatter') #Paired scatter plots, Ref A1
trainPlots.savefig('P1-trainplots.png')
testPlots.savefig('P1-testplots.png')

""" Combined two for more throughful analysis, prepare-plot-save """
trainTestDatas = pd.concat([trainDatas, trainDatas])
trainTestPlots = sns.pairplot(trainTestDatas, height=5, hue="Digits", kind='scatter') #Paired scatter plots, Ref A1
trainTestPlots.savefig('P1-trainTestplots.png')

""" Look at 1&5 ONLY """
#temp=trainDatas["Digits"].isin([1,5]) # return a truth table for row with digits 1&5
trainDatas1_5=trainDatas.loc[trainDatas["Digits"].isin([1,5]), : ] #pd.["Digits"].isin([1,5]) ret turth table, get th
trainTestDatas1_5=trainTestDatas.loc[trainTestDatas["Digits"].isin([1,5]), : ] #pd.["Digits"].isin([1,5]) ret turth t

trainDatas1_5Plots = sns.pairplot(trainDatas1_5, height=5, hue="Digits", kind='scatter') #Paired scatter plots, Ref A
trainTestDatas1_5Plots = sns.pairplot(trainTestDatas1_5, height=5, hue="Digits", kind='scatter') #Paired scatter plot

trainDatas1_5Plots.savefig('P1-trainDatas1_5plots.png')
trainTestDatas1_5Plots.savefig('P1-trainTestDatas1_5plots.png')
```

Results

All plots are done by comparing “digits”, “intensity”, “symmetry” one by one against each other as below. The intensity against symmetry is the most important comparison which shows that it is not easy to separate all ten digits by a line or just a few straight lines. It is the same for the test set, train set and even more worst when mixed the two sets.

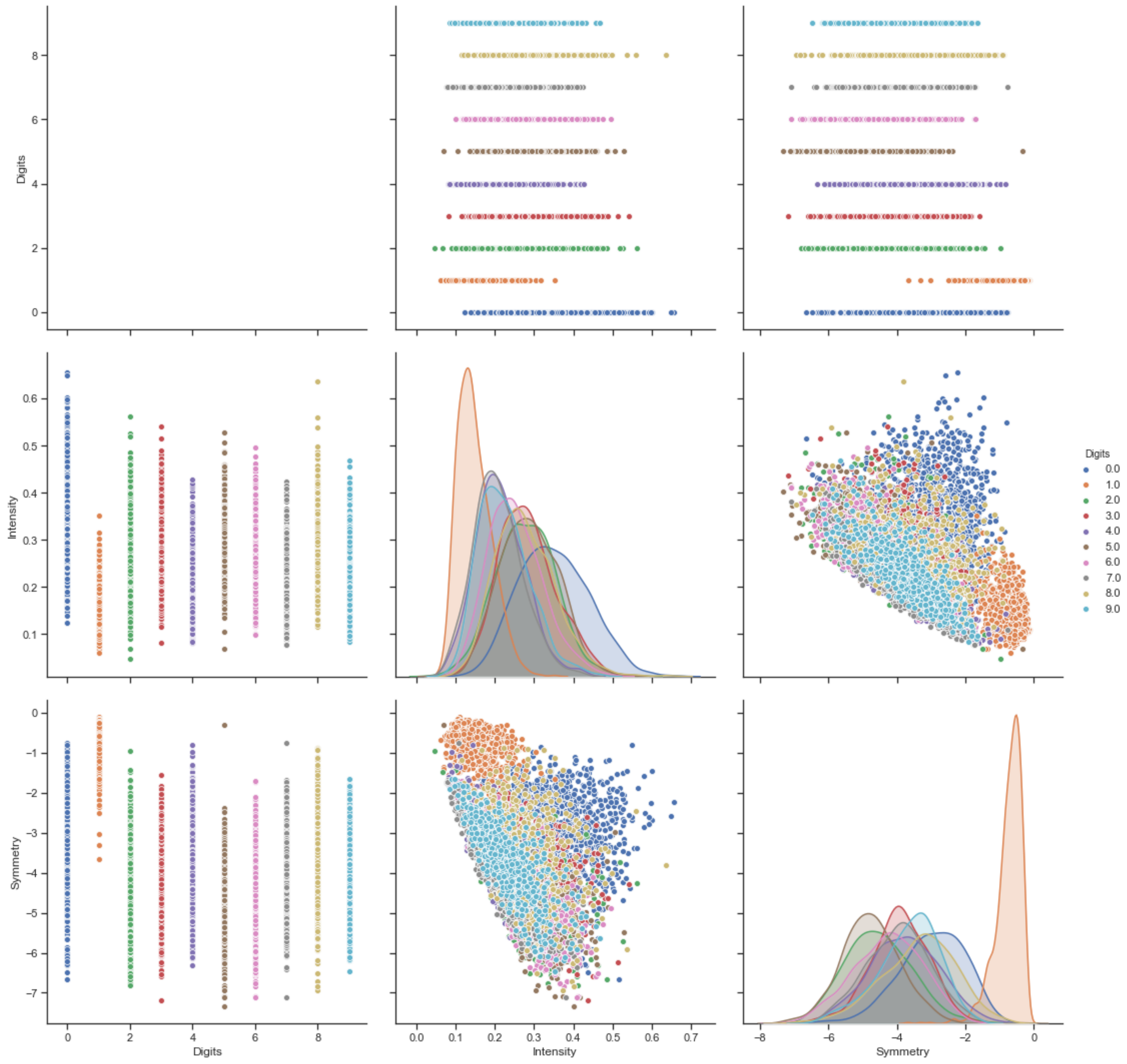


Figure Task1-1, train set only

By Hoson LAM, Copyright 2024

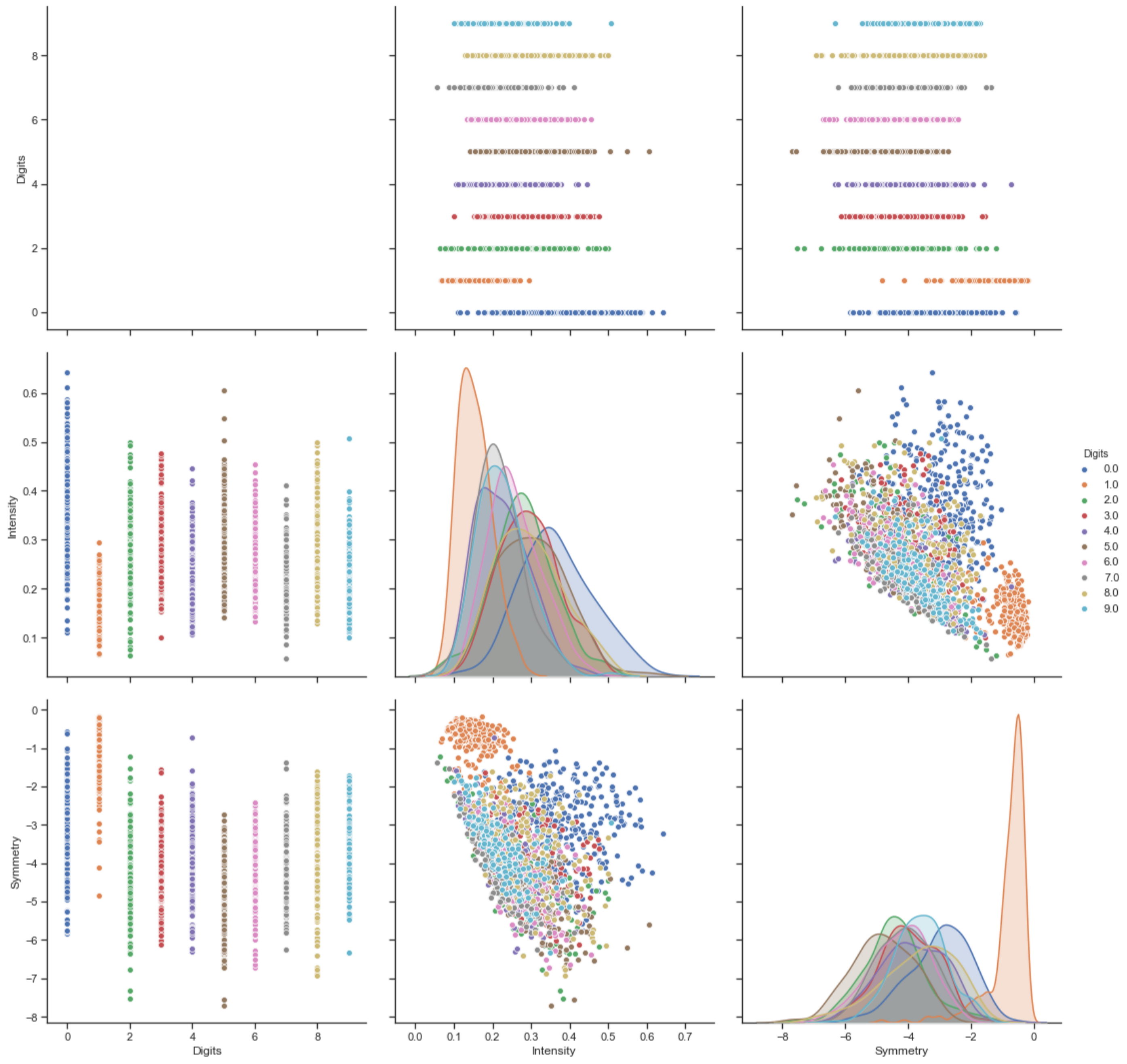


Figure Task1-2, the test set only

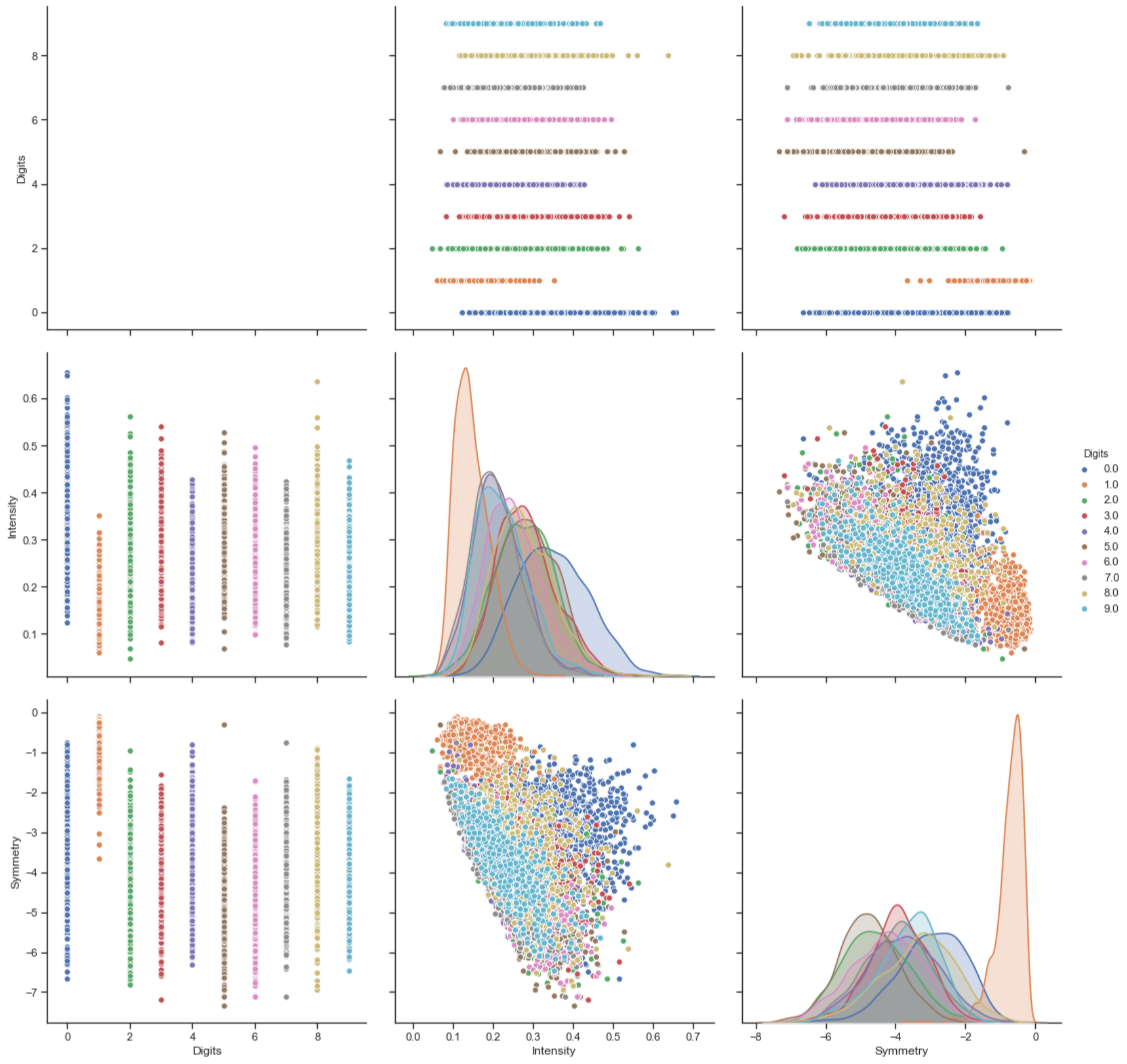


Figure Task1-3, train and test together

However, it is better if classification is only on ‘1’ and ‘5’ as shown below by the specific intensity against symmetry plots.

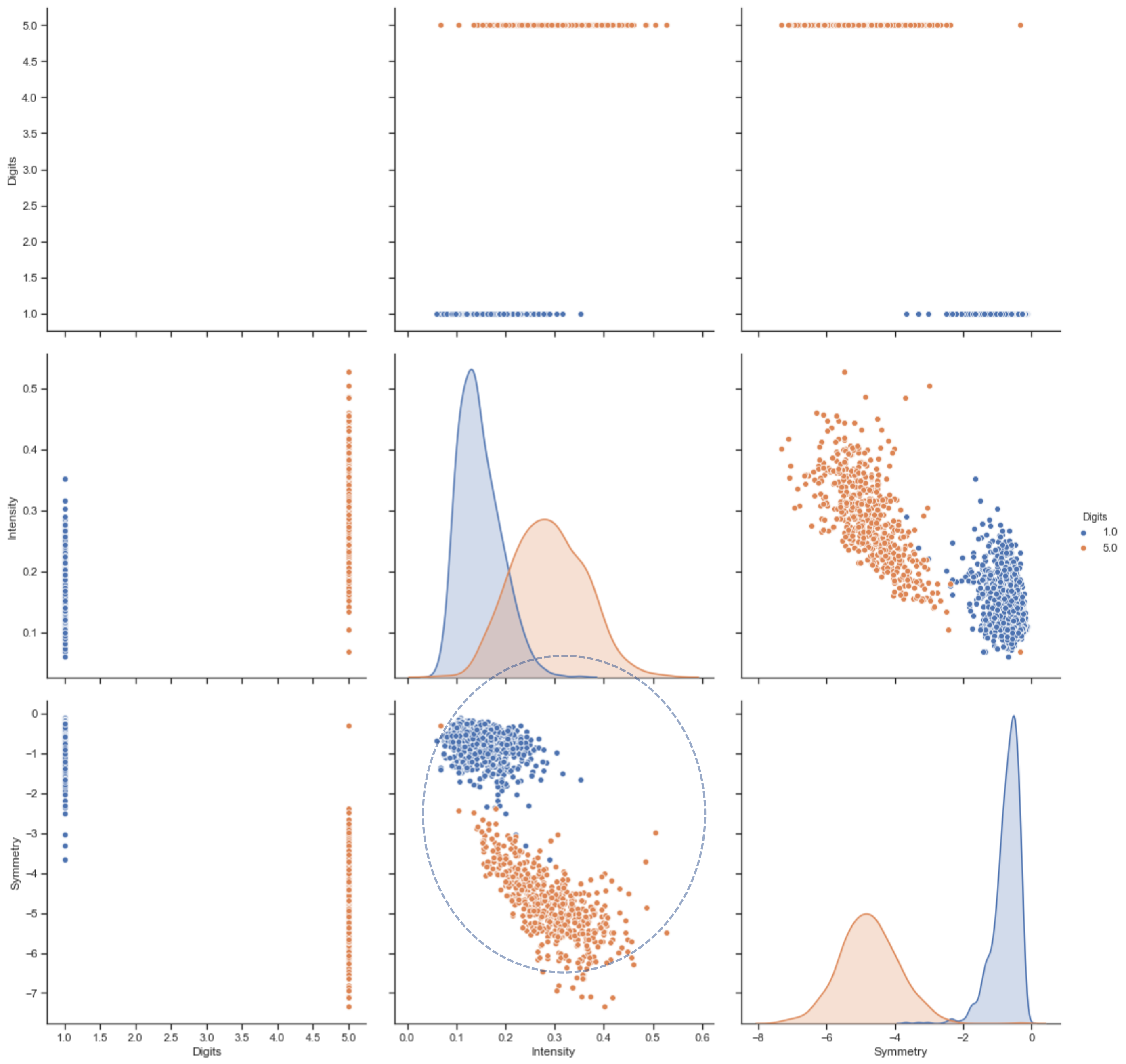


Figure Task1-4, digits 1 & 5 only for the train set

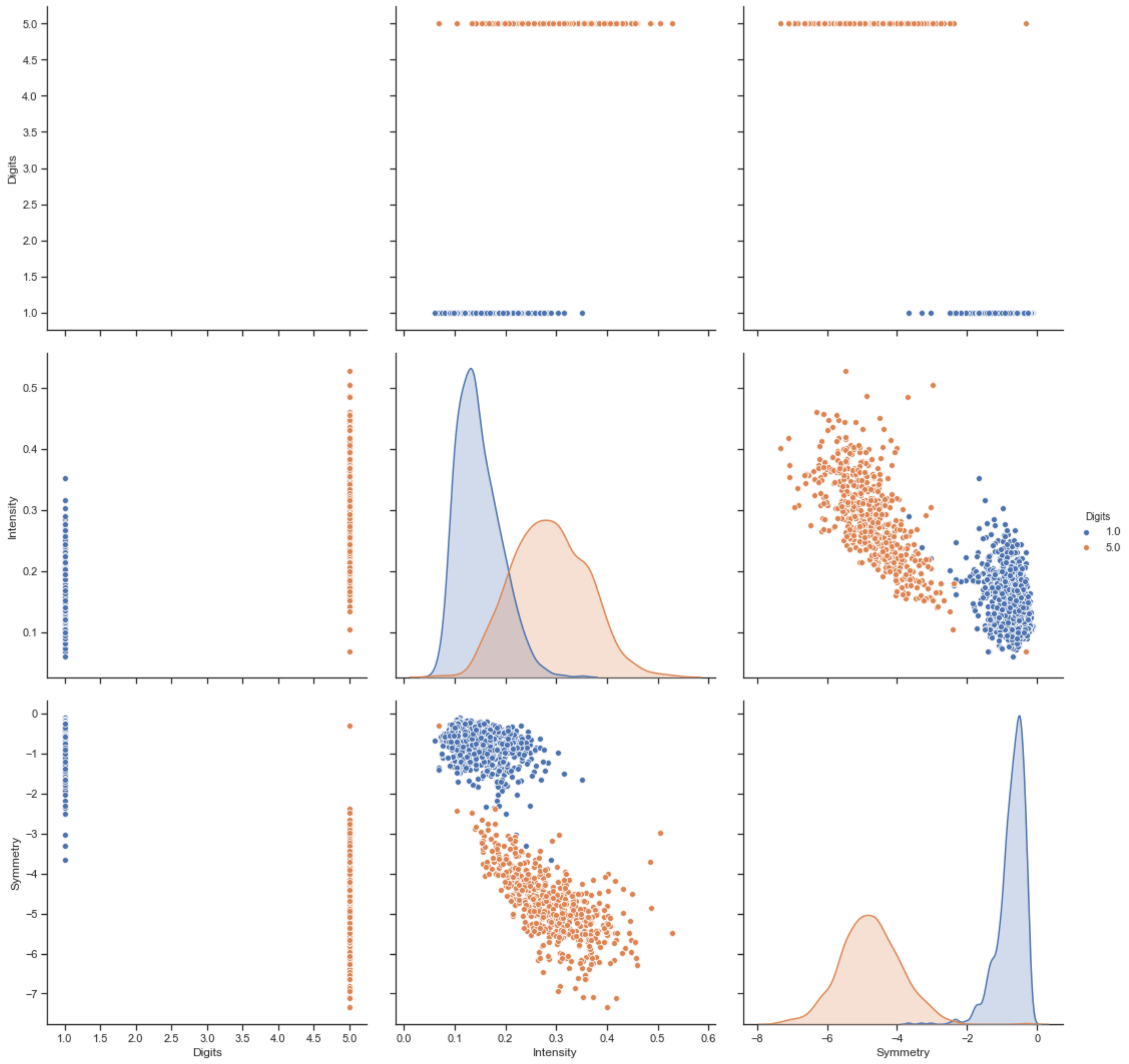


Figure Task1-5, digits 1 & 5 only for the train and test sets altogether

Tasks 2:

Details

Apply a neural network of 1 hidden layer to classify 1 and 5. The features are symmetry and average intensity. 3-fold cross-validation is used by Sklearn's KFold API (from sklearn.model_selection import KFold).

Following showing the network structure in detail and corresponding hyperparameters. A dropout layer with a rate of 0.5 is added to make some sensible recognition results, otherwise, the success rate is too low for practical recognition and the three folds look very different from each other for making sensible interpretations.

Inputs are the two given features, the first layer has two neurons for each feature, output has two neurons for two classes in matching Kera's API standard.

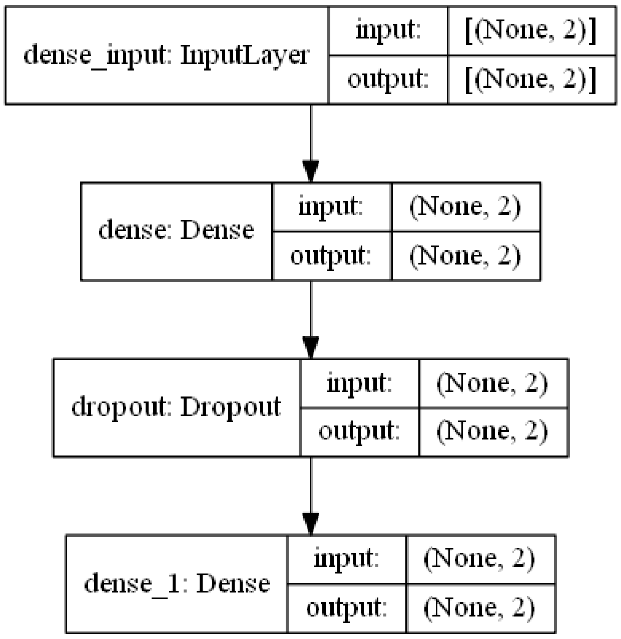
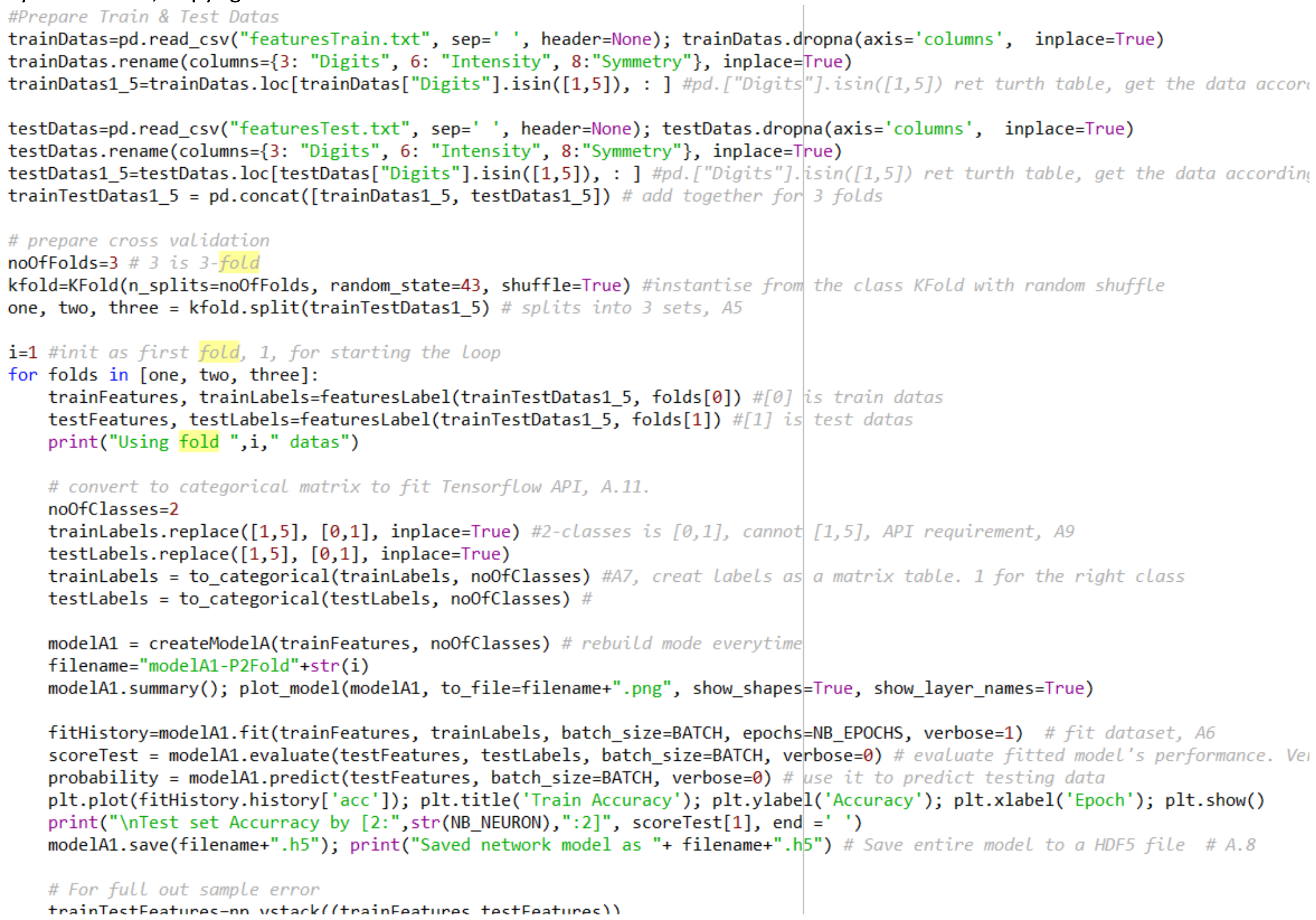


Figure Task2-1, Neural Network Dense Layer's structure and parameters.

```
_________________________________________________________________
Layer (type)              Output Shape             Param #
=================================================================
dense (Dense)              (None, 2)              6
_________________________________________________________________
dropout (Dropout)           (None, 2)              0
_________________________________________________________________
dense (Dense)              (None, 2)              6
=================================================================
Total params: 12
Trainable params: 12
Non-trainable params: 0
```

Figure Task2-2, Tensorflow reported parameters.

By Hoson LAM, Copyright 2024

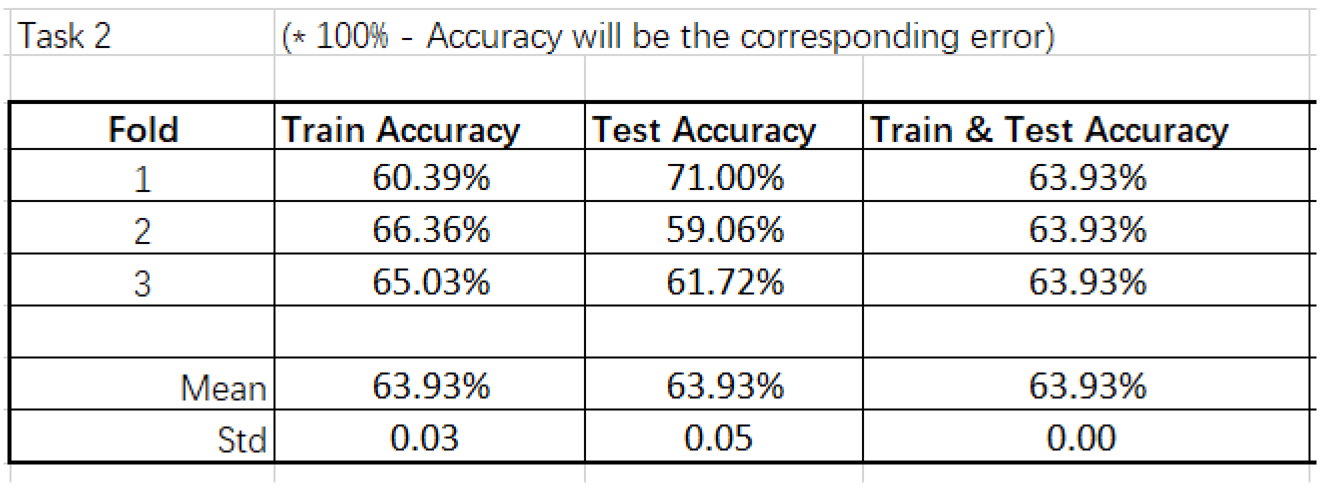


```
#Prepare Train & Test Datas
trainDatas=pd.read_csv("featuresTrain.txt", sep=' ', header=None); trainDatas.dropna(axis='columns',  inplace=True)
trainDatas.rename(columns={3: "Digits", 6: "Intensity", 8:"Symmetry"}, inplace=True)
trainDatas1_5=trainDatas.loc[trainDatas["Digits"].isin([1,5]), : ] #pd.["Digits"].isin([1,5]) ret turth table, get the data accor

testDatas=pd.read_csv("featuresTest.txt", sep=' ', header=None); testDatas.dropna(axis='columns',  inplace=True)
testDatas.rename(columns={3: "Digits", 6: "Intensity", 8:"Symmetry"}, inplace=True)
testDatas1_5=testDatas.loc[testDatas["Digits"].isin([1,5]), : ] #pd.["Digits"].isin([1,5]) ret turth table, get the data accordin
trainTestDatas1_5 = pd.concat([trainDatas1_5, testDatas1_5]) # add together for 3 folds

# prepare cross validation
noOfFolds=3 # 3 is 3-fold
kfold=KFold(n_splits=noOfFolds, random_state=43, shuffle=True) #instantise from the class KFold with random shuffle
one, two, three = kfold.split(trainTestDatas1_5) # splits into 3 sets, A5

i=1 #init as first fold, 1, for starting the loop
for folds in [one, two, three]:
    trainFeatures, trainLabels=featuresLabel(trainTestDatas1_5, folds[0]) #[0] is train datas
    testFeatures, testLabels=featuresLabel(trainTestDatas1_5, folds[1]) #[1] is test datas
    print("Using fold ",i," datas")

    # convert to categorical matrix to fit Tensorflow API, A.11.
    noOfClasses=2
    trainLabels.replace([1,5], [0,1], inplace=True) #2-classes is [0,1], cannot [1,5], API requirement, A9
    testLabels.replace([1,5], [0,1], inplace=True)
    trainLabels = to_categorical(trainLabels, noOfClasses) #A7, creat labels as a matrix table. 1 for the right class
    testLabels = to_categorical(testLabels, noOfClasses) #

    modelA1 = createModelA(trainFeatures, noOfClasses) # rebuild mode everytime
    filename="modelA1-P2Fold"+str(i)
    modelA1.summary(); plot_model(modelA1, to_file=filename+".png", show_shapes=True, show_layer_names=True)

    fitHistory=modelA1.fit(trainFeatures, trainLabels, batch_size=BATCH, epochs=NB_EPOCHS, verbose=1)  # fit dataset, A6
    scoreTest = modelA1.evaluate(testFeatures, testLabels, batch_size=BATCH, verbose=0) # evaluate fitted model's performance. Ve
    probability = modelA1.predict(testFeatures, batch_size=BATCH, verbose=0) # use it to predict testing data
    plt.plot(fitHistory.history['acc']); plt.title('Train Accuracy'); plt.ylabel('Accuracy'); plt.xlabel('Epoch'); plt.show()
    print("\nTest set Accurracy by [2:",str(NB_NEURON),":2]", scoreTest[1], end =' ')
    modelA1.save(filename+".h5"); print("Saved network model as "+ filename+".h5") # Save entire model to a HDF5 file  # A.8

    # For full out sample error
    trainTestFeatures=np.vstack((trainFeatures,testFeatures))
```

Figure Task2-3, Codes.

Results

Table Task2-3 summarised the 3 fold results and performance.



Task 2 (* 100% - Accuracy will be the corresponding error)

Fold	Train Accuracy	Test Accuracy	Train & Test Accuracy
1	60.39%	71.00%	63.93%
2	66.36%	59.06%	63.93%
3	65.03%	61.72%	63.93%
Mean	63.93%	63.93%	63.93%
Std	0.03	0.05	0.00

Table Task2-3, results from the 3 folds Training and Prediction

With such simple image features and simple neural networks, the result is not too bad and repeatable for verifications. However, a dropout layer is added to make this happen.

Tasks 3:

Details

Apply a two-layer neural network for classification of 1 and 5 by using the raw features as input. Applied 3-fold cross-validation.

a) The [256, 6, 2, 1] version

Following showing the network structure in detail and corresponding hyperparameters. A dropout layer with a rate of 0.5 is added for better results.

Input is pixels, the first layer has 6 neurons with Sign activation function and dropout is added for better repeatable results. The output layer has 2 neurons (match Keras standard of 2 instead of 1) with Sign activation function for classifying 2 classes. Softsign is used to approximate Sign from Keras' API[5] for quicker implementation and testing under limited time.

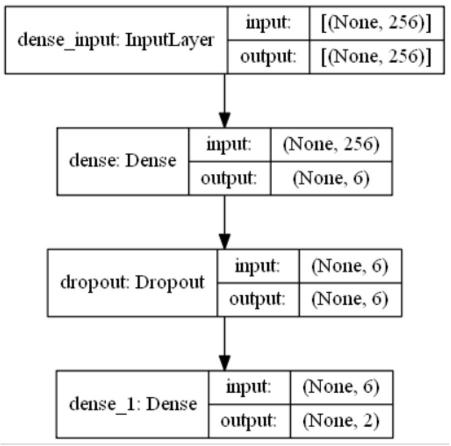
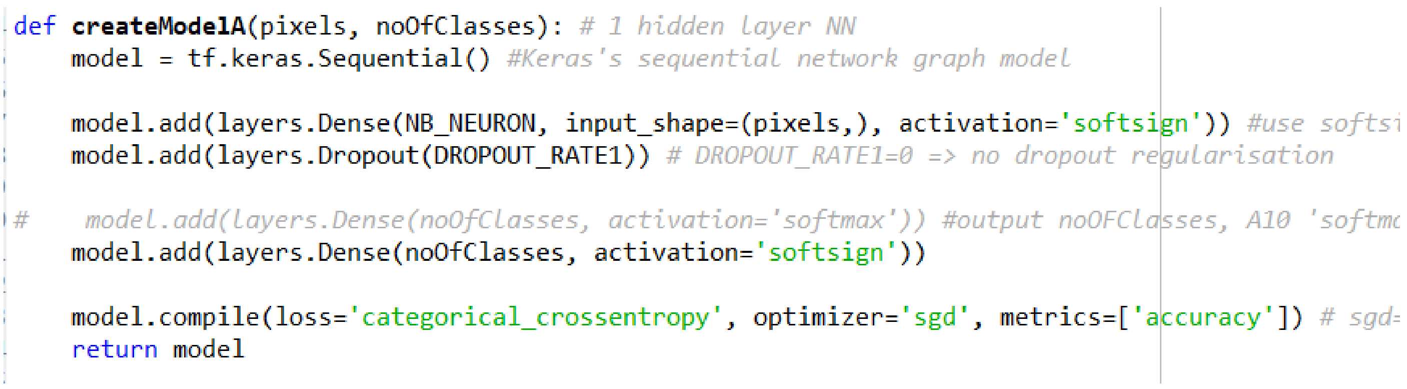


Table Task3-3, The neural network's structure and parameter



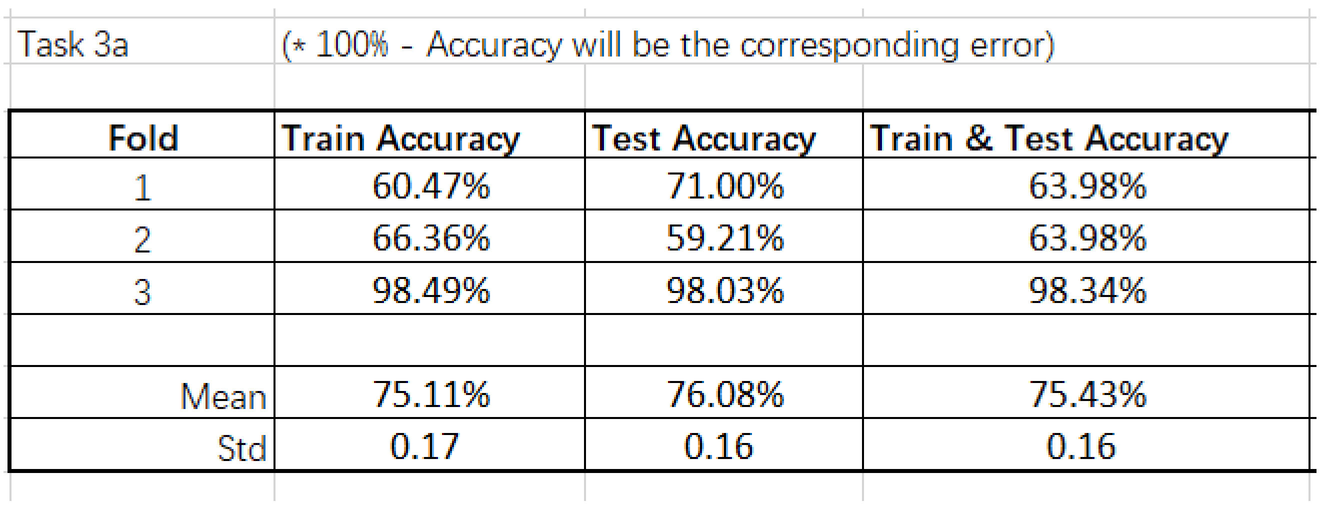
```
def createModelA(pixels, noOfClasses): # 1 hidden layer NN
    model = tf.keras.Sequential() #Keras's sequential network graph model

    model.add(layers.Dense(NB_NEURON, input_shape=(pixels,), activation='softsign')) #use softsi
    model.add(layers.Dropout(DROPOUT_RATE1)) # DROPOUT_RATE1=0 => no dropout regularisation

#    model.add(layers.Dense(noOfClasses, activation='softmax')) #output noOFClasses, A10 'softmc
    model.add(layers.Dense(noOfClasses, activation='softsign'))

    model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy']) # sgd=
    return model
```

Table Task3-2, the network model's Python code

Results

a) The [256, 6, 2, 1] version

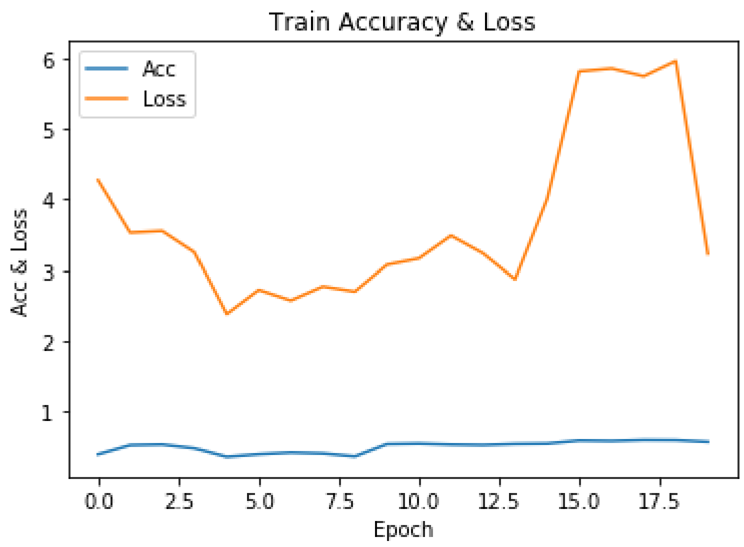
Table Task3a-1 summarised the 3 fold results and performance. It is not very good when compared to the two features recognition method as this all pixel recognition used much more computational resources relatively.



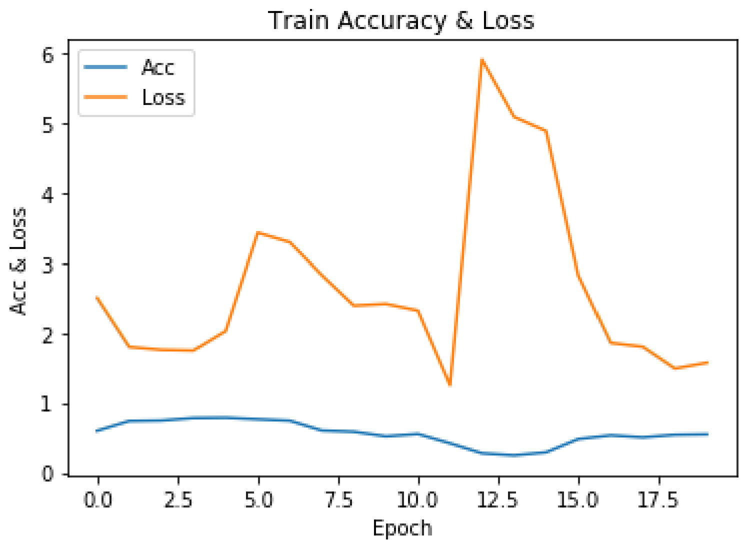
Task 3a (* 100% - Accuracy will be the corresponding error)

Fold	Train Accuracy	Test Accuracy	Train & Test Accuracy
1	60.47%	71.00%	63.98%
2	66.36%	59.21%	63.98%
3	98.49%	98.03%	98.34%
Mean	75.11%	76.08%	75.43%
Std	0.17	0.16	0.16

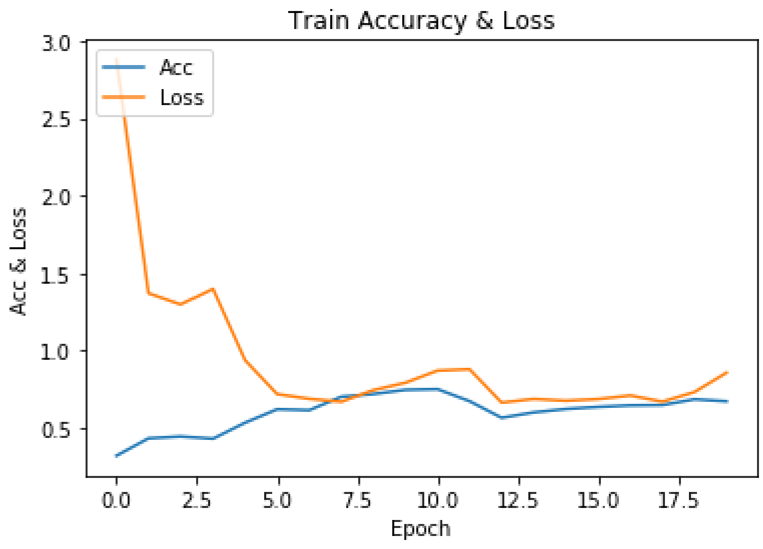
Table Task3a-1, results from the 3 folds Training and Prediction



Using fold 1 datas
Test set Accurracy by [256: 6 :2]= 0.70996976
Train sets Accurracy = 0.6046863
Test&Train sets Accurracy by [256: 6 :2]= 0.63979846



Using fold 2 datas
Test set Accurracy by [256: 6 :2]= 0.592145
Train sets Accurracy = 0.66364324
Test&Train sets Accurracy by [256: 6 :2]= 0.63979846



Using fold 3 datas
Test set Accurracy by [256: 6 :2]= 0.98033285
Train sets Accurracy = 0.9848943
Test&Train sets Accurracy by [256: 6 :2]= 0.9833753

Fold 3 is interesting and outperform fold 1 and fold 2 in a big amount. In fact, several more runs are done, most of the folds are in the range of 60-70% and over 90% is rare. This configuration is not very stable.

b) The [256, 3, 2, 1] version

Following showing the network structure in detail and corresponding hyperparameters. A dropout layer with a rate of 0.5 is added for better results.

Input is pixels, the first layer has 3 neurons with Sign activation function and dropout is added for better repeatable results. The output layer has 2 neurons (match Keras standard of 2 instead of 1) with Sign activation function for classifying 2 classes. Softsign is used to approximate Sign from Keras' API[5] for quicker implementation and testing under limited time.

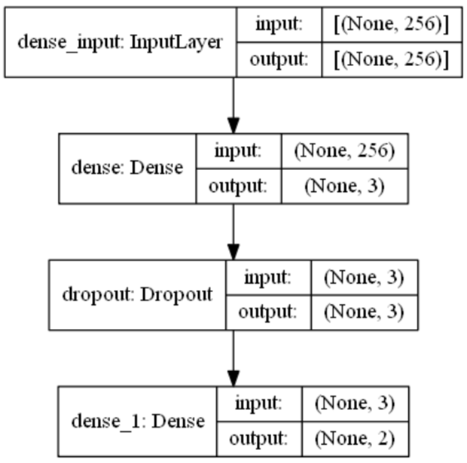
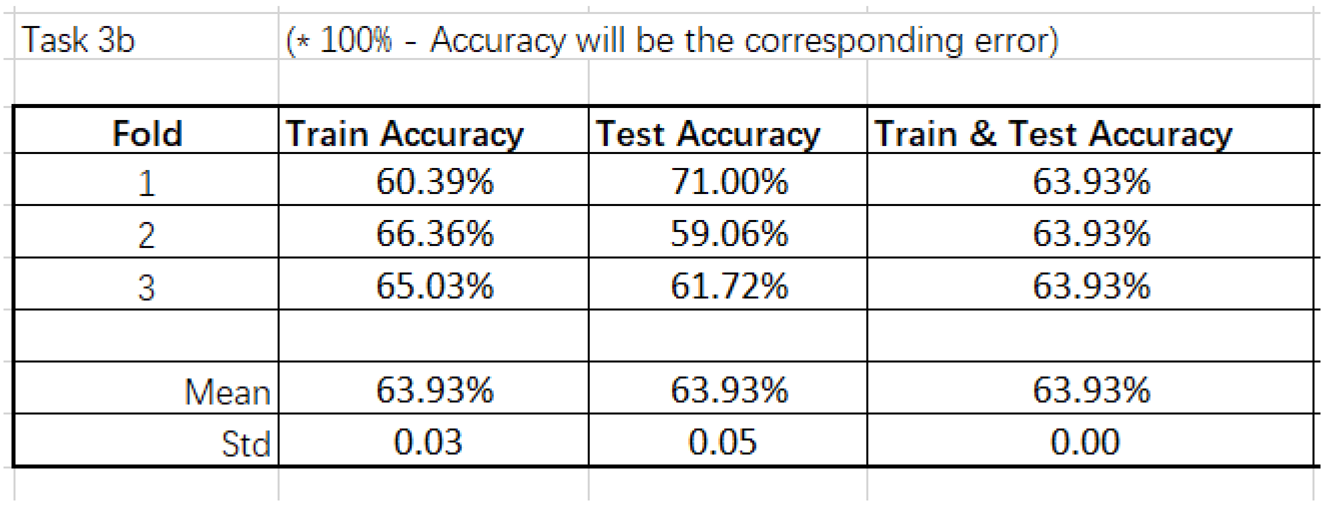


Figure Task3b-1, The neural network's structure and parameter

Results b)

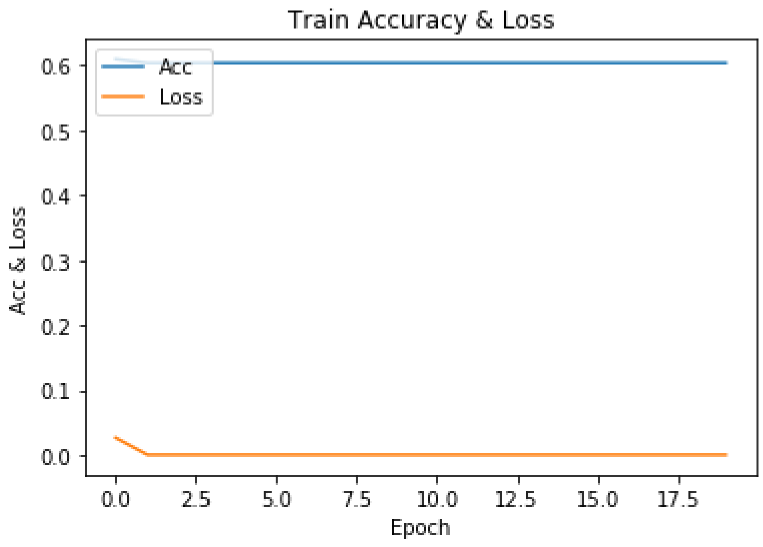
Table Task3b-1 summarised the 3 fold results and performance. It is more worst than a) and also using more resources than task 2. However, variation is less than task 3a.



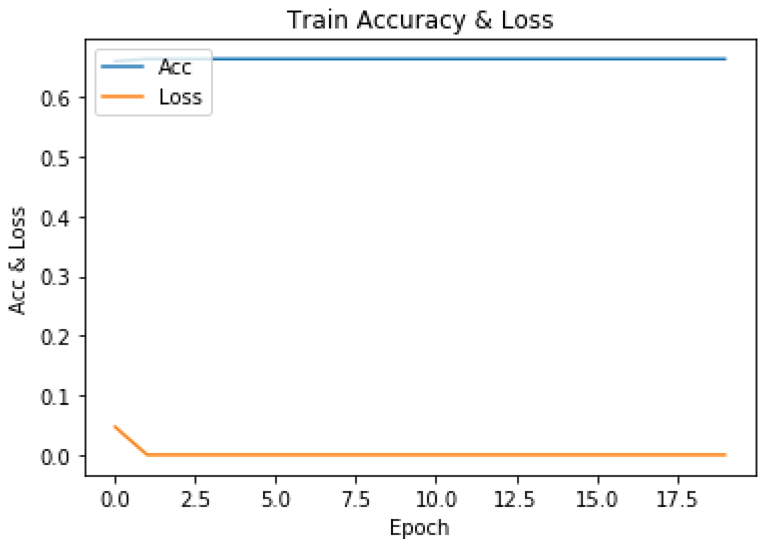
Task 3b (* 100% - Accuracy will be the corresponding error)

Fold	Train Accuracy	Test Accuracy	Train & Test Accuracy
1	60.39%	71.00%	63.93%
2	66.36%	59.06%	63.93%
3	65.03%	61.72%	63.93%
Mean	63.93%	63.93%	63.93%
Std	0.03	0.05	0.00

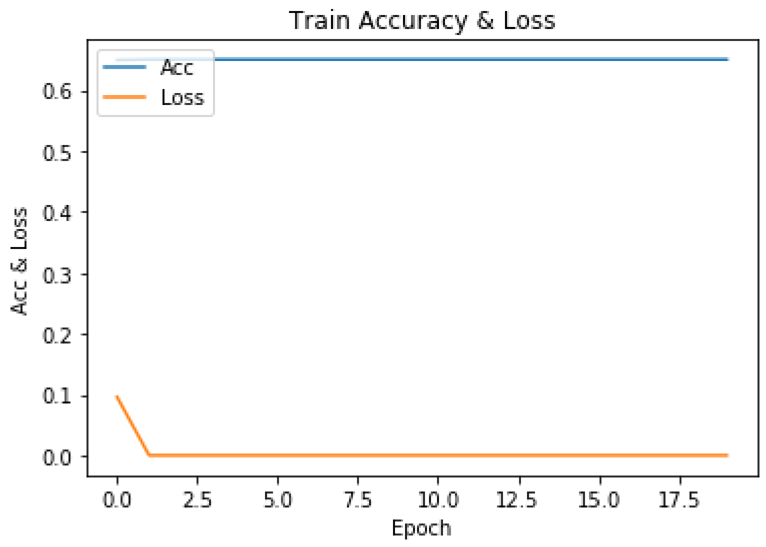
Table Task3b-1, results from the 3 folds Training and Prediction



Using fold 1 datas
Test set Accurracy by [256: 3 :2]= 0.70996976
Train sets Accurracy = 0.6039305
Test&Train sets Accurracy by [256: 3 :2]= 0.6392947



Using fold 2 datas
Test set Accurracy by [256: 3 :2]= 0.59063447
Train sets Accurracy = 0.66364324
Test&Train sets Accurracy by [256: 3 :2]= 0.6392947



Using fold 3 datas
Test set Accurracy by [256: 3 :2]= 0.61724657
Train sets Accurracy = 0.6503021
Test&Train sets Accurracy by [256: 3 :2]= 0.6392947

This version is more stable and uses fewer resources.

Tasks 4:

Apply a neural network for classification of all 10 digits, using the raw features as input. Two systems are tried:

4-1)

It is a simple deep feedforward DFF network with Relu and Softmax activation functions instead of the crispy sign function. Furthermore, dropout regularisation layers are added and several dropout rates are tried before settling the values in the latest version of the Python code.

Deep Feed Forward (DFF)

Figure Task4-1-1, A typical DFF [1]

4-2)

It is a Convolution Neural Network CNN with frontend learnable feature map filters.

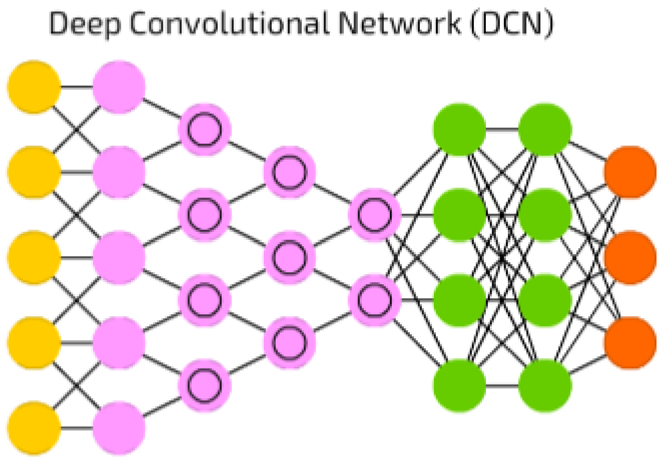


Figure Task4-1-2, A typical CNN [1]

Tasks 4-1: DFF with around 97% recognition accuracy

Details

Following showing the network structure in detail and corresponding hyperparameters. Input are pixels, the first layer has 32 neurons with ReLu output activation and dropout, the second layer has 32*2 (64) neurons with Relu output activation and dropout regularization, output layer use Softmax activation and having 10 neurons for classifying 10 classes.

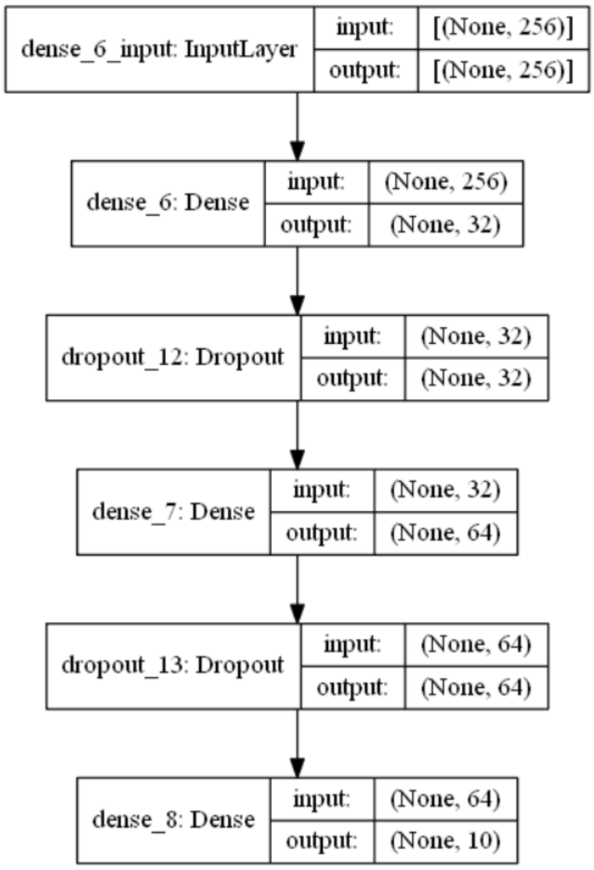
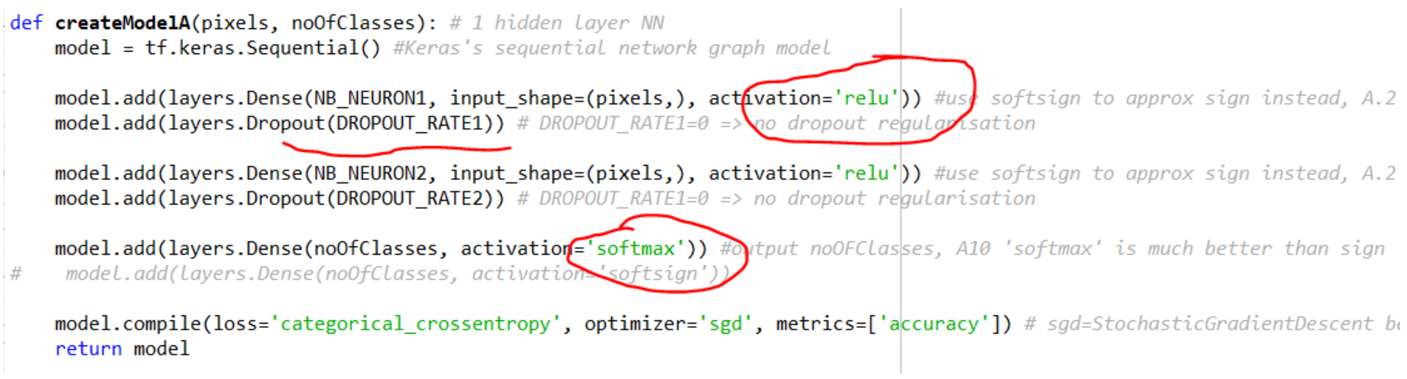
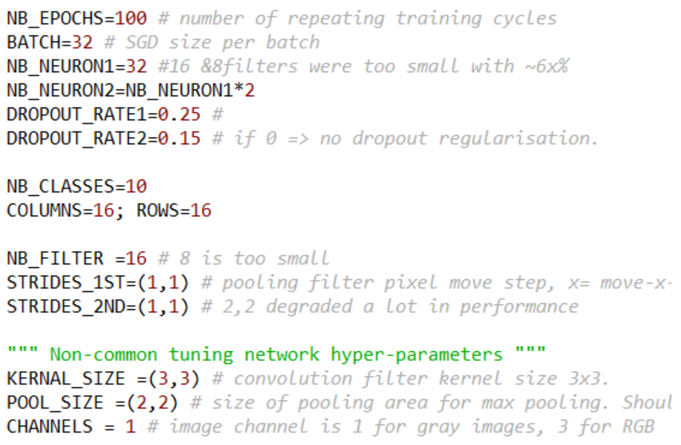


Figure Task4DFF-1, The neural network's structure and parameters.



```
def createModelA(pixels, noOfClasses): # 1 hidden layer NN
    model = tf.keras.Sequential() #Keras's sequential network graph model

    model.add(layers.Dense(NB_NEURON1, input_shape=(pixels,), activation='relu')) #use softsign to approx sign instead, A.2
    model.add(layers.Dropout(DROPOUT_RATE1)) # DROPOUT_RATE1=0 => no dropout regularisation

    model.add(layers.Dense(NB_NEURON2, input_shape=(pixels,), activation='relu')) #use softsign to approx sign instead, A.2
    model.add(layers.Dropout(DROPOUT_RATE2)) # DROPOUT_RATE1=0 => no dropout regularisation

    model.add(layers.Dense(noOfClasses, activation='softmax')) #output noOFClasses, A10 'softmax' is much better than sign
#     model.add(layers.Dense(noOfClasses, activation='softsign'))

    model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy']) # sgd=StochasticGradientDescent b
    return model
```



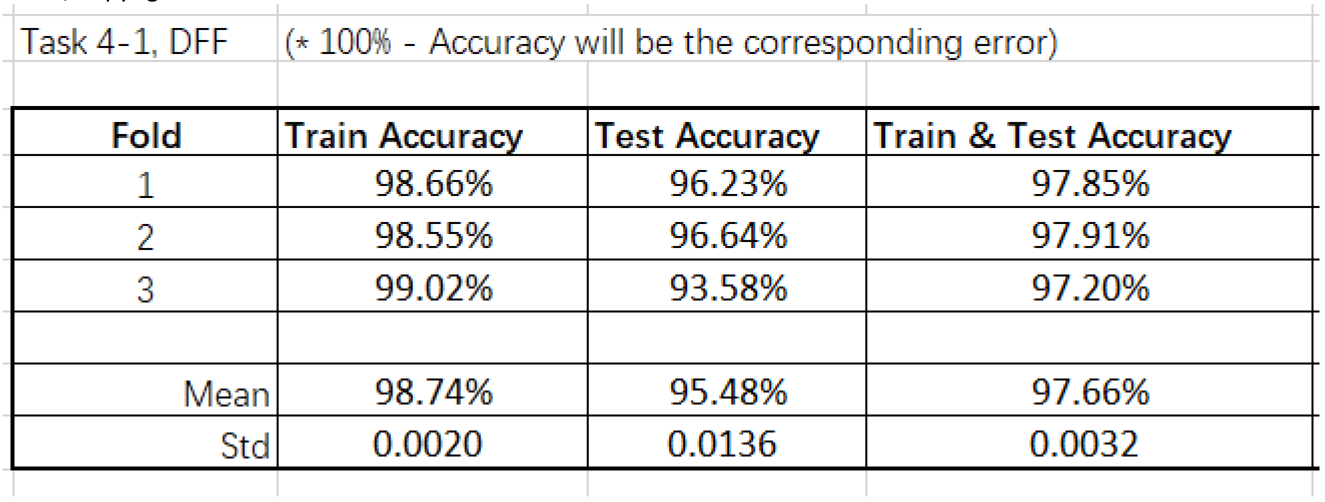
```
NB_EPOCHS=100 # number of repeating training cycles
BATCH=32 # SGD size per batch
NB_NEURON1=32 #16 &8filters were too small with ~6x%
NB_NEURON2=NB_NEURON1*2
DROPOUT_RATE1=0.25 #
DROPOUT_RATE2=0.15 # if 0 => no dropout regularisation.

NB_CLASSES=10
COLUMNS=16; ROWS=16

NB_FILTER =16 # 8 is too small
STRIDES_1ST=(1,1) # pooling filter pixel move step, x= move-x-
STRIDES_2ND=(1,1) # 2,2 degraded a lot in performance

""" Non-common tuning network hyper-parameters """
KERNAL_SIZE =(3,3) # convolution filter kernel size 3x3.
POOL_SIZE =(2,2) # size of pooling area for max pooling. Shoul
CHANNELS = 1 # image channel is 1 for gray images, 3 for RGB
```

```
_________________________________________________________________
Layer (type)              Output Shape           Param #
=================================================================
dense (Dense)              (None, 32)           8224
_________________________________________________________________
dropout (Dropout)           (None, 32)            0
_________________________________________________________________
dense (Dense)              (None, 64)           2112
_________________________________________________________________
dropout (Dropout)           (None, 64)            0
_________________________________________________________________
dense (Dense)              (None, 10)           650
=================================================================
Total params: 10,986
Trainable params: 10,986
Non-trainable params: 0
```

Figure Task4DFF-1-2, Tensorflow reported parameters.

Results

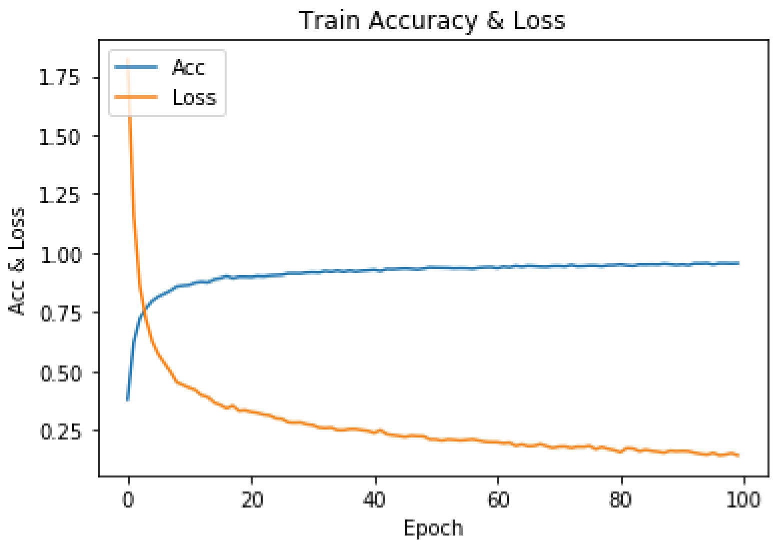
Table task4DFF-1 summarized the 3 fold results and performance. It is encouraging but not up to state of art standard yet.



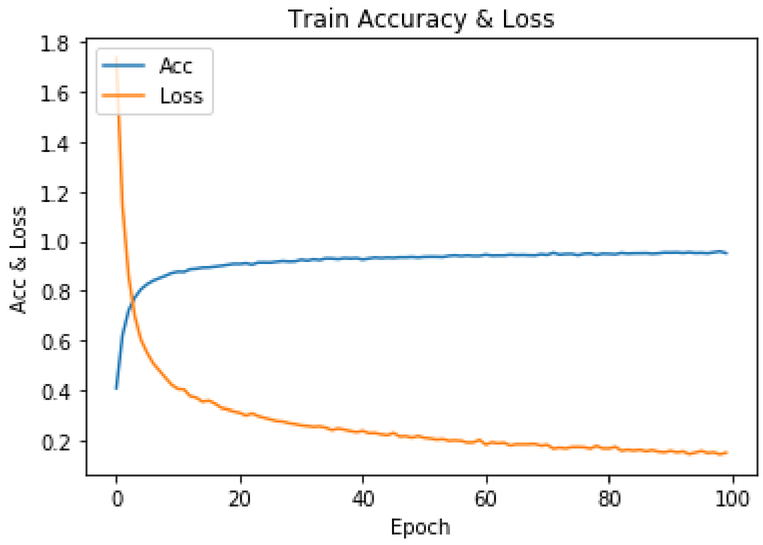
Task 4-1, DFF (* 100% - Accuracy will be the corresponding error)

Fold	Train Accuracy	Test Accuracy	Train & Test Accuracy
1	98.66%	96.23%	97.85%
2	98.55%	96.64%	97.91%
3	99.02%	93.58%	97.20%
Mean	98.74%	95.48%	97.66%
Std	0.0020	0.0136	0.0032

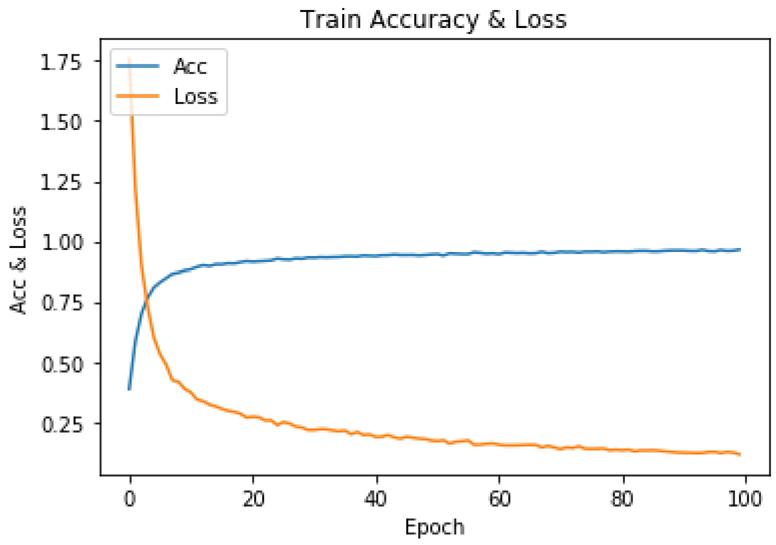
Table Task4DFF-1, results from the 3 folds Training and Prediction



Above is using fold 1 datas



Above is using fold 2 datas



Above is using fold 3 datas

Tasks 4-2: CNN – with a model successfully over 99.5%

Details

CNN with two convolution filter layers for further exploring the spatial relationship of the pixels and some higher-level shapes. Together with the regularization dropout layer and pooling layer for complexity reduction. Start from the CNN example in reference [2] chapter 3, a new version of the neural network model was tried to boost up performance as close to 100% as possible, within the limited time available. Following showing the network structure in detail and corresponding hyperparameters. Filters sizes are 64 and 128, the hidden flatten layer has 128 neurons, strides sizes are (1, 1), kernel size is (3, 3) and pooling size is (2, 2). Training time is less than 73s

By Hoson LAM, Copyright 2024

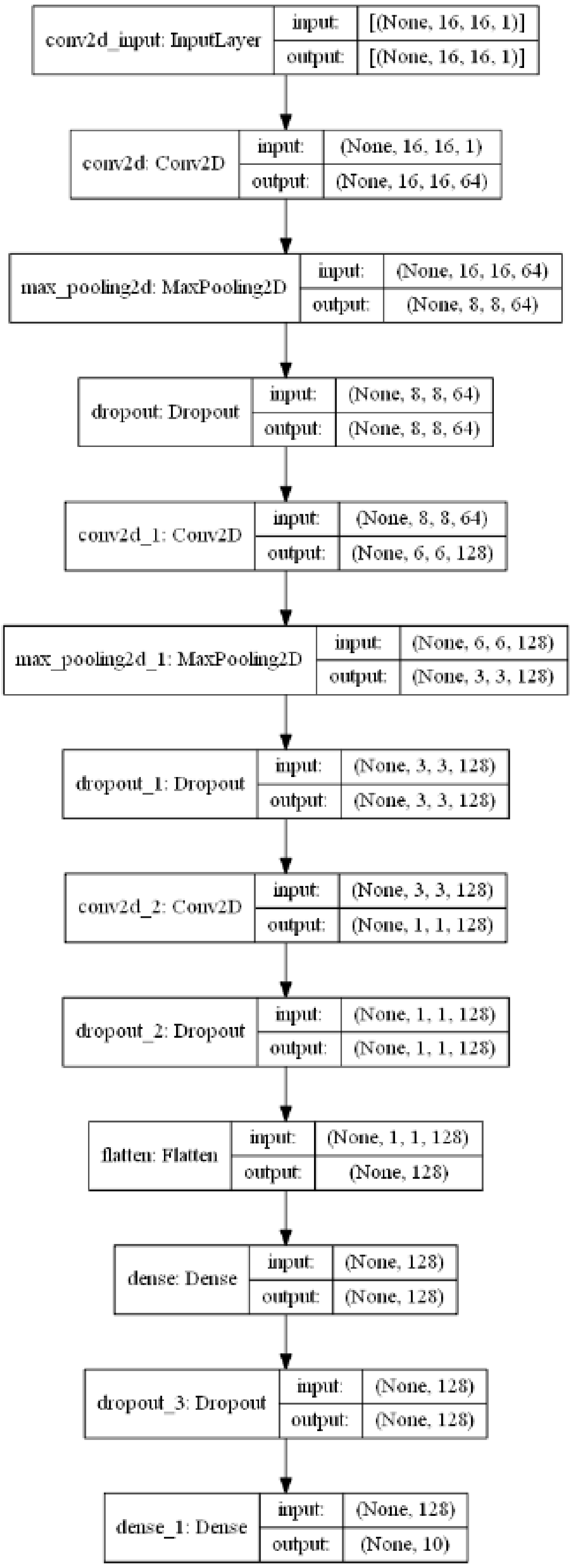
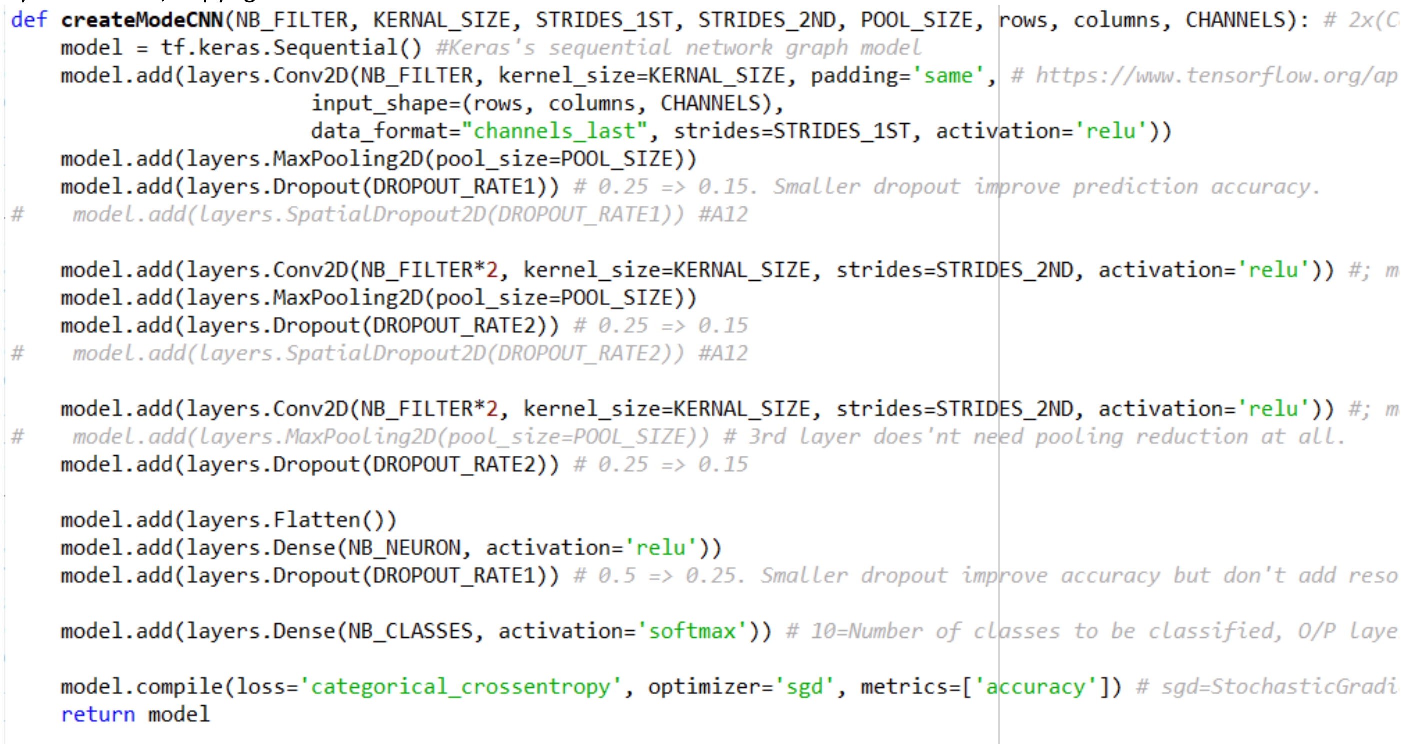


Figure Task4CNN-2, The neural network’s structure and parameters.



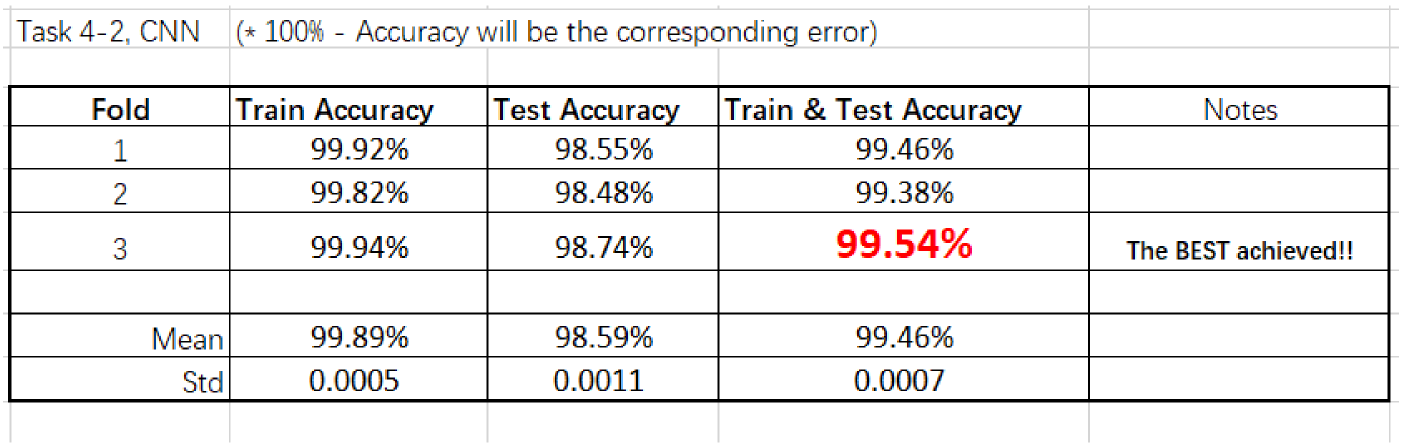
```
def createModeCNN(NB_FILTER, KERNAL_SIZE, STRIDES_1ST, STRIDES_2ND, POOL_SIZE, rows, columns, CHANNELS): # 2x(C
    model = tf.keras.Sequential() #Keras's sequential network graph model
    model.add(layers.Conv2D(NB_FILTER, kernel_size=KERNAL_SIZE, padding='same', # https://www.tensorflow.org/ap
                      input_shape=(rows, columns, CHANNELS),
                      data_format="channels_last", strides=STRIDES_1ST, activation='relu'))
    model.add(layers.MaxPooling2D(pool_size=POOL_SIZE))
    model.add(layers.Dropout(DROPOUT_RATE1)) # 0.25 => 0.15. Smaller dropout improve prediction accuracy.
#     model.add(layers.SpatialDropout2D(DROPOUT_RATE1)) #A12

    model.add(layers.Conv2D(NB_FILTER*2, kernel_size=KERNAL_SIZE, strides=STRIDES_2ND, activation='relu')) #; m
    model.add(layers.MaxPooling2D(pool_size=POOL_SIZE))
    model.add(layers.Dropout(DROPOUT_RATE2)) # 0.25 => 0.15
#     model.add(layers.SpatialDropout2D(DROPOUT_RATE2)) #A12

    model.add(layers.Conv2D(NB_FILTER*2, kernel_size=KERNAL_SIZE, strides=STRIDES_2ND, activation='relu')) #; m
#     model.add(layers.MaxPooling2D(pool_size=POOL_SIZE)) # 3rd layer does'nt need pooling reduction at all.
    model.add(layers.Dropout(DROPOUT_RATE2)) # 0.25 => 0.15

    model.add(layers.Flatten())
    model.add(layers.Dense(NB_NEURON, activation='relu'))
    model.add(layers.Dropout(DROPOUT_RATE1)) # 0.5 => 0.25. Smaller dropout improve accuracy but don't add reso

    model.add(layers.Dense(NB_CLASSES, activation='softmax')) # 10=Number of classes to be classified, O/P laye

    model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy']) # sgd=StochasticGradi
    return model
```

```
Layer (type)              Output Shape           Param #
=================================================================
conv2d (Conv2D)           (None, 16, 16, 64)      640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 8, 8, 64)        0
_________________________________________________________________
dropout (Dropout)         (None, 8, 8, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)         (None, 6, 6, 128)       73856
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 3, 3, 128)       0
_________________________________________________________________
dropout_1 (Dropout)       (None, 3, 3, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)         (None, 1, 1, 128)       147584
_________________________________________________________________
dropout_2 (Dropout)       (None, 1, 1, 128)       0
_________________________________________________________________
flatten (Flatten)         (None, 128)             0
_________________________________________________________________
dense (Dense)             (None, 128)             16512
_________________________________________________________________
dropout_3 (Dropout)       (None, 128)             0
_________________________________________________________________
dense_1 (Dense)           (None, 10)              1290
=================================================================
Total params: 239,882
Trainable params: 239,882
Non-trainable params:
_________________________________________________________________
Training time: 72.79258489608765s
```

Figure Task4CNN-2-2, Tensorflow reported parameters.

Results

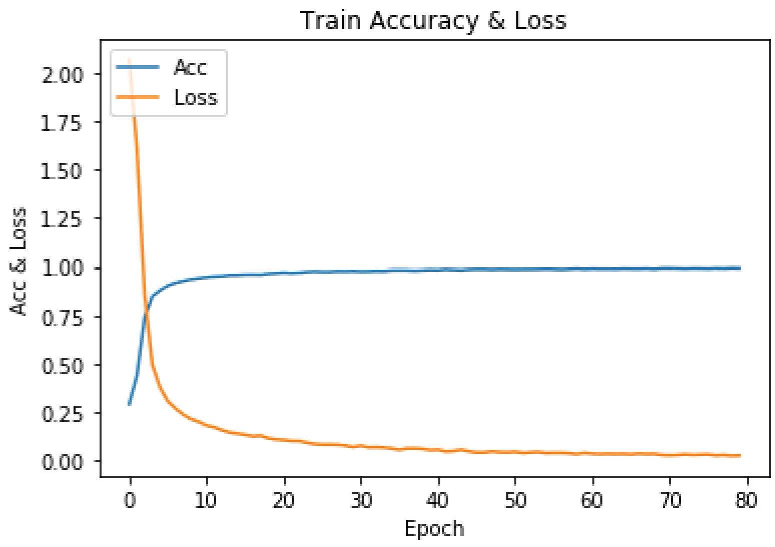
Table task4CNN-1 summarized the 3 fold results and performance. It is very encouraging.



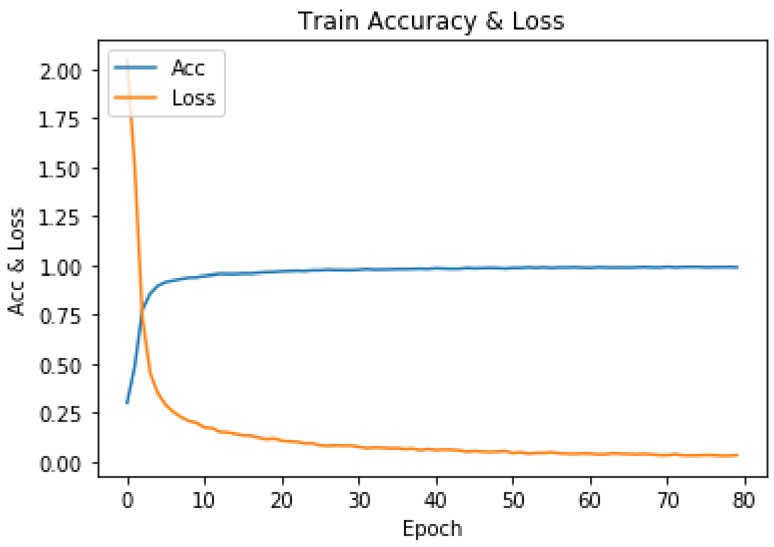
Task 4-2, CNN (* 100% - Accuracy will be the corresponding error)

Fold	Train Accuracy	Test Accuracy	Train & Test Accuracy	Notes
1	99.92%	98.55%	99.46%	
2	99.82%	98.48%	99.38%	
3	99.94%	98.74%	**99.54%**	**The BEST achieved!!**
Mean	99.89%	98.59%	99.46%	
Std	0.0005	0.0011	0.0007	

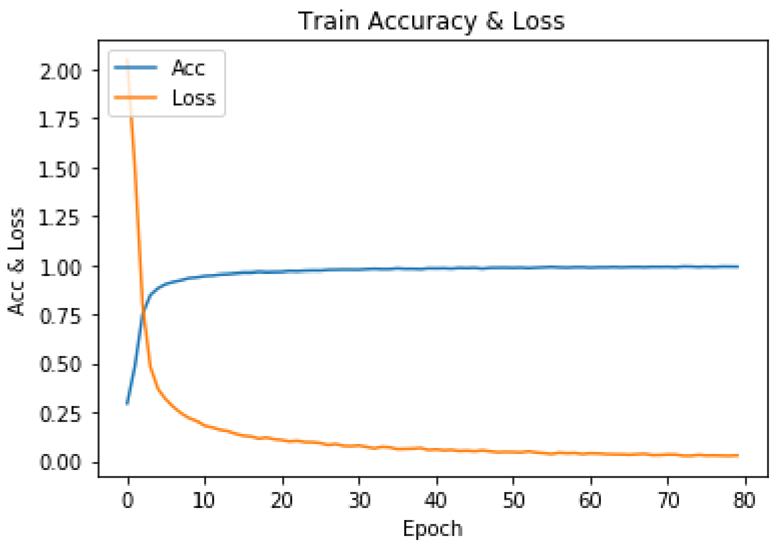
Table task4CNN-1, results from the 3 folds Training and Prediction



Above is using fold 1 datas
Test set Accurracy = 0.9854839
Train sets Accurracy = 0.9991933
Test&Train sets Accurracy = 0.9946225



Above is using fold 2 datas
Test set Accurracy = 0.98483384
Train sets Accurracy = 0.9982255
Test&Train sets Accurracy = 0.9937621



Above is using fold 3 datas
Test set Accurracy = 0.9874153
Train sets Accurracy = 0.9993547
Test&Train sets Accurracy = 0.99537534

Conclusion

The best one for 10-digits is the last one used in task 4-2. It is a modified one with two extra layers of features learning filters. This best one was called deep convolution neural network CNN with better “Relu” activations in the hidden layers and Softmax activation for multi-classes recognition at the output layers. Furthermore, regularizations were done by adding dropout which improves the recognition rate and reduces over-fitting. However, the CNN version consumes more computational resources and require longer training time but looking justified by the great improvements.

In task 3, setting for the 6-neurons version is not stable enough. These may due to heuristic variations from the back-error propagation method that is unable to sort out global optimum point, random sampling nature of the 3-fold methods and higher than enough complexity when compared to the 3-neurons version setting.

For further research, raw pixel’s image conversions like Fourier transform, discrete cosine transform can be considered or using more complicated networks like VGG16 [6]. It is still very hard to increase the recognition success rate much further in Task 4-2’s CNN by just increase its feature map filters or increase the number of neurons without increasing non-justifiable computational resources. Within the limited time frame, I did look at the image pictures directly and found that some contradicting images at raw-pixel level representation which make recognition more difficult.

Without additional tricks or transformation, 99.5% may be very close to the current CNN structure’s limit already. One more thing can be tried is to ensemble [7, 8] more than one neural network results. A possible ensemble method is to figure out wrong predictions during first training and evaluation, sort out the wrongly predicted images to train a second neural network for the difficult images only. During the recognition stage, sort out a final result from the highest probability scored result from the two neural networks which trained differently.

References

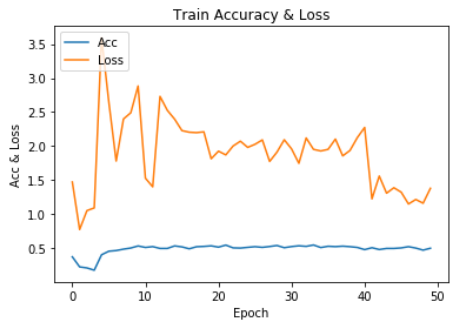
[1] Neural Cheat Sheet: https://twitter.com/darnocks/status/779365025295327232
[2] Antonio Gulli, Sujit Pal, Deep Learning with Keras, 2017
[3] Rene Vidal, Joan Bruna, Raja Giryes, Stefano Soatto. Mathematics of Deep Learning, December 2017.
[4] Kunihiko Fukushima, Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition, 1980.
[5] Activation functions available: https://keras.io/activations/
[6] VGG16 - https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c
[7] Ensemble Deep Learning - https://machinelearningmastery.com/model-averaging-ensemble-for-deep-learning-neural-networks/
[8] Zhi-Hua Zhou , Ensemble Methods: Foundations and Algorithms, CRC Press 2012.
[9] https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

Appendix

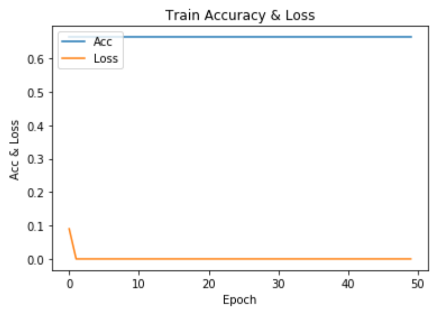
I)

Problem4-3PredictTesting.py	- Python to load Neural Net model and predict classes - Default file for verification is “Verify.txt”
Verify.txt	A dataset for verification with data extracted from the given test and train data files.
genVerifyDatasSet.py	A Python help to sample image data from original data sets for Verify.txt

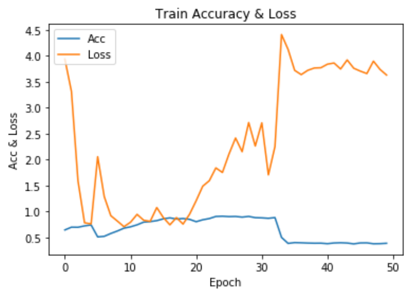
II) Screenshots of some extra runs from task 3’s 6-neurons version:



```
Saved network model as modelA1-N6-P3Fold1.h5

Test set Accurracy by [256: 6 :2]= 0.71601206
Train sets Accurracy = 0.60922146
Test&Train sets Accurracy by [256: 6 :2]= 0.64483625
```



```
Saved network model as modelA1-N6-P3Fold2.h5

Test set Accurracy by [256: 6 :2]= 0.59063447
Train sets Accurracy = 0.66364324
Test&Train sets Accurracy by [256: 6 :2]= 0.6392947
```



```
Saved network model as modelA1-N6-P3Fold3.h5

Test set Accurracy by [256: 6 :2]= 0.3827534
Train sets Accurracy = 0.3496979
Test&Train sets Accurracy by [256: 6 :2]= 0.3607053
```

For the third case, early stopping may help a bit but no time to try and this is also not a stable model !