2 Tasks

2.1 A Multi-class, Multi-layer perceptron

In this section, we will compare the common method in python, analyze the pros and cons, and further to select the better combination to establish the multi-class, multi-layer perceptron, with cross-entropy loss method to train the Fashion-MNIST dataset and observe the results.

2.1.1 Implement The Model

Compare common method in Python:

Activation Function Table

	Sigmoid	**Tanh**	**Relu**
Advantage	Simple to realize, suitable for small datasets	The output is at middle position(zero)	Prevent and revise vanishing gradient problems
Disadvantage	Vanishing gradient problems	Vanishing gradient problems, difficult to use for small datasets	Difficult to use for small datasets, need larger data for learning non-linear behavior
Function	$\sigma(x) = \frac{1}{1+e^{-x}}$	$\sigma(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$\sigma(x) = \max(0, x)$
Derivative	$\sigma'(x) = \sigma(x) * (1 - \sigma(x))$	$1 - \frac{(e^{x}-e^{-x})^2}{(e^{x}+e^{-x})^2}$	$\sigma'(x > 0) = 1,\ \sigma'(x <= 0) = 0$

Optimizer Table

	Adam	**SGD**	**RMSprop**
Advantage	Each iteration of the learning rate has certain range, let the update of the parameters more stable	Easy to converge to the local optimum, algorithm converge faster	Suitable for tackling complicated non-convex error surface
Disadvantage	In some situations and tasks, its performance not always well	Use a common learning rate for all parameters	Need to set global learning rate by ourselves

Choose the Optimal Combination

On the basis of the above comparison and the meaning of the question, we need to use the multi-class, multi-layer perceptron to adjust the relevant parameters and observe the difference. Therefore, in order to facilitate our analysis of the values and observations, in **Question 1**, we decide to use the keras method with appropriate parameters and manners to establish the perceptron model. Due to keras model possesses higher readability, easier for us to understand the discrepancy, and better to analyze the results. Whereas, we will also use the pytorch manner to create the model in **Question 3, Question 5** and learn more about other methods' principle.

Keras Model

In the end, we define our model as input 28×28 sizes, with one hidden layer 256 units and output 10 categories and choose the *Relu* function to be our activation function. In addition, the default optimization method used is ADAM with learning rate (η) being 0.001, batch-size being 256 and epoch being 50, and these parameters will be our optimal combination with cross-entropy loss method to do the following each sub-question. Actually, before we define the model and these optimal parameters, we have compared the results of different parameters, and then select this ideal vanilla model **(original model)**. For instances, when we adjust the learning rate, we find that the learning rate become larger, the results will skip some extreme value regions, become smaller, the results will fall into the wrong extreme value regions. Besides, the reason that batch-size is 256 as a result of the group effect that we can observe the best under this condition. Lastly, the epoch we define 50, because when the epoch value of training in this model exceeds 50, we will find that the loss values of the validation dataset arises quickly, and overfit in the early epoch. We also add the dropout function in the vanilla model to decrease the speed of the overfitting and have better results.

Fashion-MNIST Data:

The training dataset contains 50,000 images each with size 28×28, the validation dataset has 10,000 images, the test dataset has 10,000 images, the categories contain 10 types: t-shirt/top(0), trouser(1), pullover(2), dress(3), coat(4), sandal(5), shirt(6), sneaker(7), bag(8), ankle boot(9).

Original Model:

Some codes are break into two lines due to the limitation of page width to display in one line.

```
import numpy as np
import idx2numpy
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
import pandas as pd
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.utils import np_utils
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

# Due to the Keras will random occupy memory space, as a result, we need to
# control memory space
GPU = tf.compat.v1.GPUOptions(allow_growth = True)
session = tf.compat.v1.Session(config = tf.compat.v1.ConfigProto
                                        (gpu_options = GPU))

tf.compat.v1.keras.backend.set_session(session)

# Read the training data
data_x = idx2numpy.convert_from_file("train-images-idx3-ubyte")
```

```
data_y = idx2numpy.convert_from_file("train-labels-idx1-ubyte")

# Define training, validation set and split the dataset
trainxs, validxs, trainys, validys = train_test_split(data_x, data_y,
                                     test_size = 1/6, random_state = 0)

# Divide into 10 categories
# if category = 1 => [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
# One hot encode
trainys = np_utils.to_categorical(trainys)
validys = np_utils.to_categorical(validys)

# Reshape the trainxs, validxs => (50000, 784), (10000, 784),
# and do a normalization
trainxs = trainxs.reshape(len(trainxs), -1) / 255
validxs = validxs.reshape(len(validxs), -1) / 255

# Read the test data
testxs = idx2numpy.convert_from_file("t10k-images-idx3-ubyte")
testys = idx2numpy.convert_from_file("t10k-labels-idx1-ubyte")

# Divide into 10 categories
# if category = 1 => [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
# One hot encode
testys = np_utils.to_categorical(testys)

# Reshape the testxs => (10000, 784),
# and do a normalization
testxs = testxs.reshape(len(testxs), -1) / 255

# Define the layers and nodes of keras model
model = Sequential()
model.add(Dense(256, input_dim = 784, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation = 'softmax'))

# Define the loss method, optimizer method and compile the keras model
adam = keras.optimizers.Adam(lr = 0.001)
model.compile(loss = 'categorical_crossentropy', optimizer = adam,
                                                 metrics = ['accuracy'])

# Set the initial value
Initial_Train_Value = model.evaluate(trainxs, trainys)
Initial_Valid_Value = model.evaluate(validxs, validys)

# Fit the keras model on the dataset
# Set the model_process dictionary to obtain the loss value during each epoch
model_process = model.fit(trainxs, trainys, epochs = 50, batch_size = 256,
                validation_data = (validxs, validys), verbose = 0)
```

2.1.2 Explore different Layer Sizes and Depths

In this section, for the purpose of observing the difference in different layer sizes and depths, we will gradually increase the number of the hidden layer and gradually reduce the dimensionality **(decrease units by a factor of two)** at each layer at the same time. As a result of decreasing the dimensionality can let the output result be more reasonable and let the trend of the datasets be more credible. In addition, the parameters of the keras model will same as the above mentioned **(optimal combination)**, and we will create extra four models **(explain as below)**, offer two types of results, analyze the results, find which model works well on the validation set and explain the process how we arrive at our final model.

Define Other Four Models

The reason why we have other four models is as a result of the rules which we mention above. When we add one more layer, the dimension will decrease units by a factor of two at that layer simultaneously. So, for instance, our initial model input is 784 sizes, hidden layer is 256 units and the output are 10 categories, and the hidden layer of the model will conform the rules as follows: **128(+1)→64(+2)→32(+3)→16(+4)→etc**. However, when we add the 5th hidden layer, we can notice that the dimension is 8 and smaller than 10, it does not follow our rules, therefore, we will use these four models and original model to do the following experiments.

Five Models:

```
# Define the Layers and nodes of keras model
model = Sequential()
model.add(Dense(256, input_dim = 784, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation = 'softmax'))
```

(a) original model

```
# Define the Layers and nodes of keras model
model = Sequential()
model.add(Dense(256, input_dim = 784, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation = 'softmax'))
```

(b) one hidden layer

```
# Define the Layers and nodes of keras model
model = Sequential()
model.add(Dense(256, input_dim = 784, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(64, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation = 'softmax'))
```

(c) two hidden layer

```
# Define the Layers and nodes of keras model
model = Sequential()
model.add(Dense(256, input_dim = 784, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(64, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(32, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation = 'softmax'))
```

(d) three hidden layer

```
# Define the Layers and nodes of keras model
model = Sequential()
model.add(Dense(256, input_dim = 784, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(64, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(32, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(16, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation = 'softmax'))
```

(e) four hidden layer

Figure 1

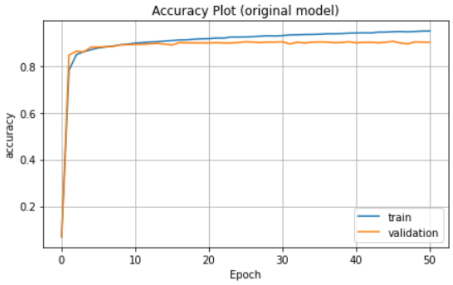
Type One Experiment:

One time training (Random Initial Value)

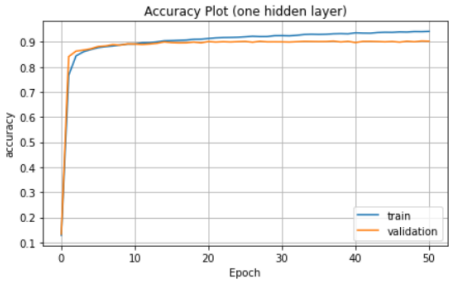
According to the one time training experiment, we simply train five models once, and obtain the five accuracy plots and loss plots as below. In these results, we can find that the accuracy of the five models are actually not much different. The accuracy curves of the training set and the validation set have tendency to overlap gradually, moreover, the time of the loss curves of the training set and the validation set cross paths at same point is postponed. Whereas, in this experiment, it is very formidable to determine which model is optimal, because the model execute only once and the weight

values of the keras model are randomly distributed, therefore, the results of each training will have deviation, which will affect the results and mislead our determination.

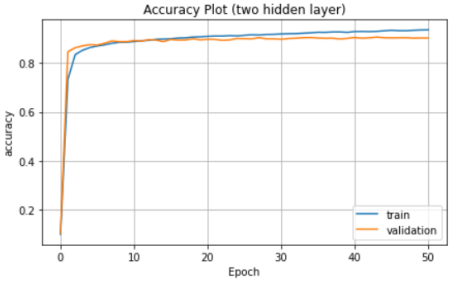
Accuracy Plot:



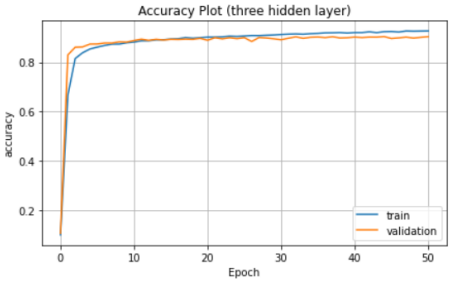
(a)



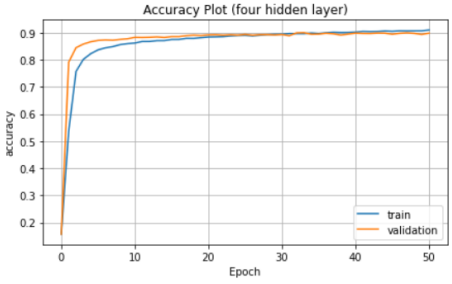
(b)



(c)

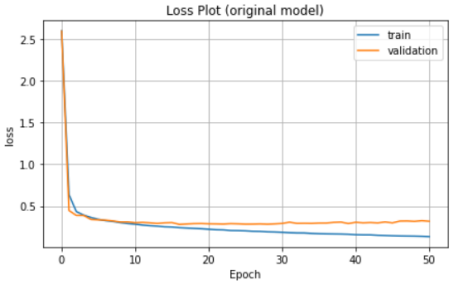


(d)

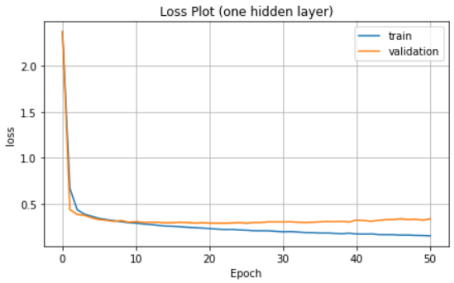


(e)

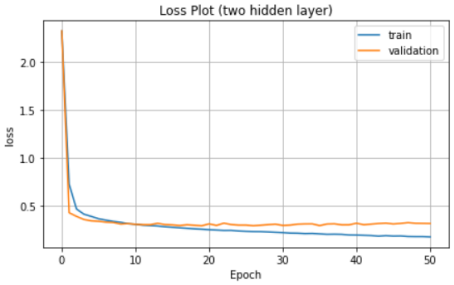
Loss Plot:



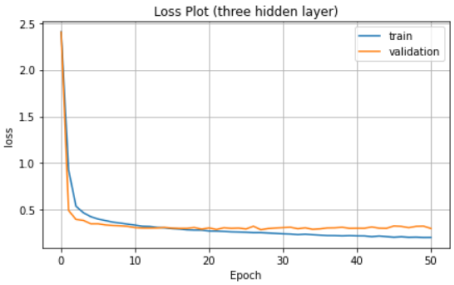
(a)



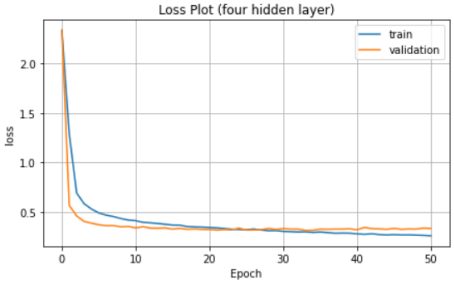
(b)



(c)



(d)



(e)

Type Two Experiment:

Use Standard deviation, Average Value

Owing to the weight values of the keras model are randomly distributed, so, when we are training the models, the output of each time will have some deviation. Therefore, it is very inappropriate to use this manner to determine which model is best. Hence, in order to handle this problem, in this experiment, the keras model will be written as a function and we will execute each model 100 times **(every time same 50 epochs, we will receive 100 accuracy and loss value at each model)** through importing the function, and calculate the standard deviation and average of accuracy and loss value, so that the obtained experimental results will be more convincing, and we will further to use these results to analyze and describe which model is our final selection.

Code:

```
import numpy as np
import idx2numpy
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
import pandas as pd
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.utils import np_utils
from sklearn.model_selection import train_test_split


def keras_model():

    # Due to the Keras will random occupy memory space, as a result, we need to
    # control memory space
    GPU = tf.compat.v1.GPUOptions(allow_growth = True)
    session = tf.compat.v1.Session(config = tf.compat.v1.ConfigProto
                                            (gpu_options = GPU))

    tf.compat.v1.keras.backend.set_session(session)

    # Read the training data
    data_x = idx2numpy.convert_from_file("train-images-idx3-ubyte")
    data_y = idx2numpy.convert_from_file("train-labels-idx1-ubyte")

    # Define training, validation set and split the dataset
    trainxs, validxs, trainys, validys = train_test_split(data_x, data_y,
                                         test_size = 1/6, random_state = 0)

    # Divide into 10 categories
    # if category = 1 => [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    # One hot encode
    trainys = np_utils.to_categorical(trainys)
    validys = np_utils.to_categorical(validys)

    # Reshape the trainxs, validxs => (50000, 784), (10000, 784),
    # and do a normalization
    trainxs = trainxs.reshape(len(trainxs), -1) / 255
    validxs = validxs.reshape(len(validxs), -1) / 255

    # Read the test data
    testxs = idx2numpy.convert_from_file("t10k-images-idx3-ubyte")
```

```
    testys = idx2numpy.convert_from_file("t10k-labels-idx1-ubyte")

    # Divide into 10 categories
    # if category = 1 => [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    # One hot encode
    testys = np_utils.to_categorical(testys)

    # Reshape the testxs => (10000, 784),
    # and do a normalization
    testxs = testxs.reshape(len(testxs), -1) / 255

    # Define the layers and nodes of keras model
    model = Sequential()
    model.add(Dense(256, input_dim = 784, activation = 'relu'))
    model.add(Dropout(0.2))
    # add one hidden layer
    model.add(Dense(128, activation = 'relu'))
    model.add(Dropout(0.2))
    # add two hidden layer
    model.add(Dense(64, activation = 'relu'))
    model.add(Dropout(0.2))
    # add three hidden layer
    model.add(Dense(32, activation = 'relu'))
    model.add(Dropout(0.2))
    # add four hidden layer
    model.add(Dense(16, activation = 'relu'))
    model.add(Dropout(0.2))
    model.add(Dense(10, activation = 'softmax'))

    # Define the loss method, optimizer method and compile the keras model
    adam = keras.optimizers.Adam(lr = 0.001)
    model.compile(loss = 'categorical_crossentropy', optimizer = adam,
                                                     metrics = ['accuracy'])

    # Set the initial value
    Initial_Train_Value = model.evaluate(trainxs, trainys)
    Initial_Valid_Value = model.evaluate(validxs, validys)

    # Fit the keras model on the dataset
    # Set the model_process dictionary to obtain the loss value
    # during each epoch
    model_process = model.fit(trainxs, trainys, epochs = 50, batch_size = 256,
                    validation_data = (validxs, validys), verbose = 0)

    # Evaluate the keras model, calculate loss and accuracy of
    # training and validation dataset
    train_loss, train_accuracy = model.evaluate(trainxs, trainys)
    print('Accuracy: %.2f' % (train_accuracy * 100))

    valid_loss, valid_accuracy = model.evaluate(validxs, validys)
    print('Accuracy: %.2f' % (valid_accuracy * 100))

    return train_loss, train_accuracy, valid_loss, valid_accuracy

# Call function
from Keras_Model import keras_model
import numpy as np

# Define four list to store the value
```

```
train_loss, train_accuracy, valid_loss, valid_accuracy = [], [], [], []

# Define hidden_layer original, 1, 2, 3, 4
hidden_layer = 4

# Train 100 times to observe the standard deviation and average result
for i in range(1, 101):

    # Call the function, and append the value into each list
    trainloss, trainacc, validloss, validacc = keras_model()
    train_loss.append(trainloss)
    train_accuracy.append(trainacc)
    valid_loss.append(validloss)
    valid_accuracy.append(validacc)

    # Every 10 steps store the result, avoid the server break down
    if i % 10 == 0:
        np.savez("layer = {}.npz".format(hidden_layer), train_loss = train_loss,
                                        train_accuracy = train_accuracy,
                                        valid_loss = valid_loss,
                                        valid_accuracy = valid_accuracy)

# Store file into the npz file, f is a dictionary
f = np.load("layer = {}.npz".format(hidden_layer))

# Calculate average results
np.mean(f["train_accuracy"])
round(np.mean(f["train_accuracy"]), 4)

np.mean(f["valid_accuracy"])
round(np.mean(f["valid_accuracy"]), 4)

np.mean(f["train_loss"])
round(np.mean(f["train_loss"]), 4)

np.mean(f["valid_loss"])
round(np.mean(f["valid_loss"]), 4)

# Calculate standard deviation results
np.std(f["train_accuracy"])
round(np.std(f["train_accuracy"]), 6)

np.std(f["valid_accuracy"])
round(np.std(f["valid_accuracy"]), 6)

np.std(f["train_loss"])
round(np.std(f["train_loss"]), 6)

np.std(f["valid_loss"])
round(np.std(f["valid_loss"]), 6)
```

One hundred times training (Random Initial Value)

Average Results (Accuracy):

	Original model	One hidden layer	Two hidden layer	Three hidden layer	Four hidden layer
Training Dataset	0.9613	0.9574	0.9501	0.9429	0.9304
Validation Dataset	0.9025	0.9032	0.9026	0.9012	0.8982

Average Results (Loss):

	Original model	One hidden layer	Two hidden layer	Three hidden layer	Four hidden layer
Training Dataset	0.1085	0.1135	0.1310	0.1499	0.1856
Validation Dataset	0.3177	0.3238	0.3129	0.3135	0.3340

Explanation:

On the basis of the average values of the accuracy and loss calculated above, we can notice that the deviations of five models are not too large in terms of the accuracy of the validation dataset. Their validation datasets are all close to 90% accuracy. However, there is a little difference in average values of the loss. We can find that the average loss values of the original model, two hidden layer model and three hidden layer model are relatively little lower. They possess a little better performance than the one hidden layer model and four hidden layer model. But, if we want to make any decision in this section is not suitable, because, their average results are too similar, and if we just use above results to determine the final model is lack of credibility, so, we need to coordinate with standard deviation to justify our selection. Hence, we continue to see and analyze the experimental results of the standard deviation as below.

Standard Deviation Results (Accuracy):

	Original model	One hidden layer	Two hidden layer	Three hidden layer	Four hidden layer
Training Dataset	0.002507	0.002415	0.002410	0.002359	0.002894
Validation Dataset	0.002545	0.002301	0.002075	0.002052	0.002292

Standard Deviation Results (Loss):

	Original model	One hidden layer	Two hidden layer	Three hidden layer	Four hidden layer
Training Dataset	0.005113	0.005091	0.005578	0.005028	0.006498
Validation Dataset	0.007317	0.009034	0.007756	0.007519	0.009441

Explanation:

According to the standard deviation values of the accuracy and loss calculated above, we can notice that the five standard deviation values of the accuracy of the validation dataset are still very close **(estimate to the fourth decimal places)**, in other words, there is almost no any difference. In addition, in the standard deviation values of loss, we can find that the original model, two hidden layer model and three hidden layer model are still having a little better efficacy compared with the one hidden layer model and four hidden layer model. Therefore, combining average and standard deviation results, we conclude in the following.

Final Model

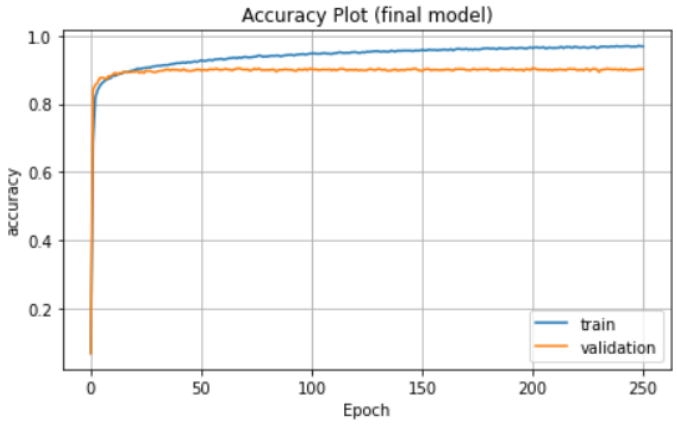
In fact, combining the experimental results of the above 100 times training results, we can find that there is not much difference between the five models. To put it another way, we can get a conclusion that the Fashion-MNIST dataset does not have a large impact on these five models, as long as we adjust the parameters within a reasonable range, we can gain the similar experimental results as above. Accordingly, on my personal note, I think that it is reasonable to choose random one of these five models. Whereas, if we want to use above experimental results to select our final model by a rigorous method, I have some viewpoints. Some reasons illustrating my perspectives will be shed light on as follows. First of all, we can use the slight difference in the above experimental results to know that the results of the original model, two hidden layer model, and three hidden layer model are better compared with other two models. Furthermore, we utilize the standard deviation results to justify the viewpoints **(the average results of accuracy and loss are too similar)**. In line with the definition of the standard deviation, the larger of the standard deviation value, the distribution of the data more dispersed in the dataset, that is to say, the model is more unstable. On the contrary, the smaller of the standard deviation value, the distribution of the data more concentrate in the dataset, and the model more stable. Hence, we can realize that the **three hidden layer model** (figure 1d) may be a better option compared with the other four models in a rigorous way, and we will also use this model to complete our other sub-questions below.

2.1.3 Training Final Model and Convergence

In this section, we use the three hidden layer model with optimal combination of the parameters that we mentioned above to do the task, besides, we increase the times of the training to **250 epochs** to observe the accuracy and loss values of the training dataset. According to the Accuracy Plot as below, we can understand that the accuracy results incline to steady at epoch 200 **(convergence)**, on top of that, even though it is difficult to determine whether the training set is convergence through **Loss Plot at next sub-question**, however, on the basis of the each iteration result showing on our computer, we can conclude that the loss value of the training dataset start have a little bumpy at epoch 25, and tend toward stability at epoch 200 **(convergence)**.

Code:

```
# Plot the accuracy value at each epoch
plt.figure(figsize = (7, 4))
plt.title('Accuracy Plot (final model)')
plt.plot(range(250 + 1), [Initial_Train_Value[1]] +
                         model_process.history['accuracy'], "-",
                         label = 'train')
plt.plot(range(250 + 1), [Initial_Valid_Value[1]] +
                         model_process.history['val_accuracy'], "-",
                         label = 'validation')
plt.xlabel('Epoch')
plt.ylabel('accuracy')
plt.grid()
plt.legend()
```



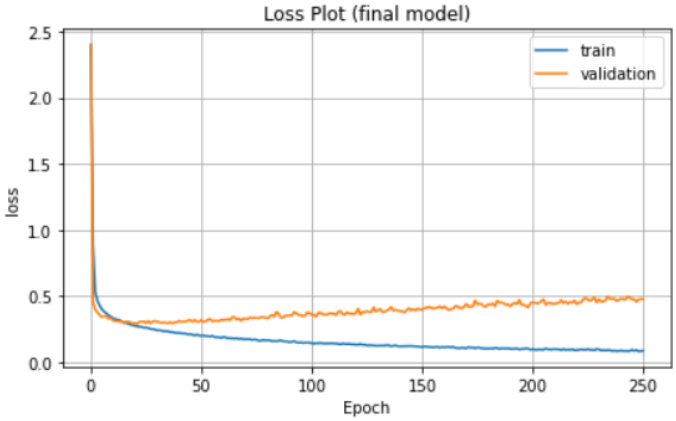
(a) accuracyc plot

2.1.4 Final Model Loss Plot

Code:

```
# Plot the loss value at each epoch
plt.figure(figsize = (7, 4))
plt.title('Loss Plot (final model)')
plt.plot(range(250 + 1), [Initial_Train_Value[0]] +
                         model_process.history['loss'], "-",
                         label = 'train')
plt.plot(range(250 + 1), [Initial_Valid_Value[0]] +
                         model_process.history['val_loss'], "-",
                         label = 'validation')
plt.xlabel('Epoch')
plt.ylabel('loss')
```

```
plt.grid()
plt.legend()
```

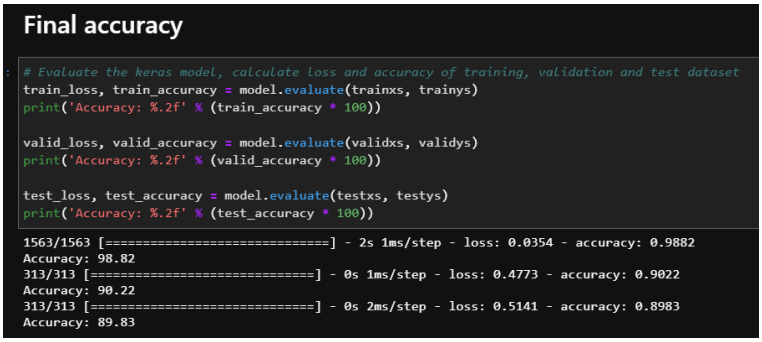


(a) loss plot

2.1.5 Final Accuracy

Code:

```
# Evaluate the keras model, calculate loss and accuracy of training,
# validation and test dataset
train_loss, train_accuracy = model.evaluate(trainxs, trainys)
print('Accuracy: %.2f' % (train_accuracy * 100))

valid_loss, valid_accuracy = model.evaluate(validxs, validys)
print('Accuracy: %.2f' % (valid_accuracy * 100))

test_loss, test_accuracy = model.evaluate(testxs, testys)
print('Accuracy: %.2f' % (test_accuracy * 100))
```



(a) final accuracy

Results:

Training Accuracy: 0.9882 (98.82%)

Validation Accuracy: 0.9022 (90.22%)

Test Accuracy: 0.8983 (89.83%)

2.1.6 Confusion Matrix

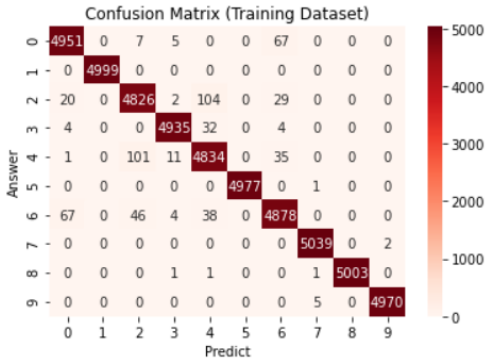
In this section, we will calculate the confusion matrix of the training, validation and test datasets, and through the results to determine which categories (labels) are easily confused and describe the reasons.

Code:

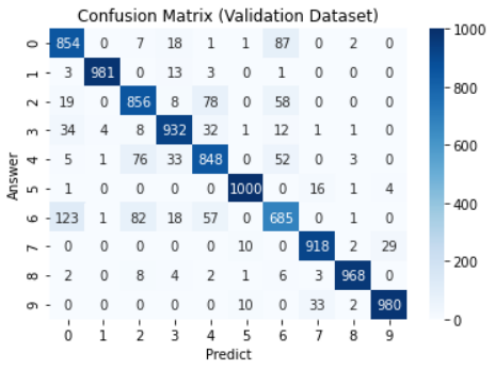
```
def Confusion_Matrix(datax, datay):
    # Define the 10 * 10 dimension matrix (10 categories)
    confusion_matrix = np.zeros((10, 10), int)
    predict_value = model.predict(datax)

    # Compare the predict and answer, and add the result into matrix
    for number in range (len(datax)):
        predict = np.argmax(predict_value[number])
        answer = np.argmax(datay[number])
        confusion_matrix[answer][predict] = confusion_matrix[answer][predict]
                                            + 1

    return confusion_matrix

import seaborn as sn

# Training Dataset confusion matrix
train_matrix = Confusion_Matrix(trainxs, trainys)
tmatrix = pd.DataFrame(train_matrix)
print(tmatrix)

# Plot the heatmap confusion matrix
ax = plt.axes()
sn.heatmap(tmatrix, ax = ax, annot = True, cmap = 'Reds', fmt = 'g')
ax.set_title('Confusion Matrix (Training Dataset)')
plt.xlabel('Predict')
plt.ylabel('Answer')
plt.show()

# Validation Dataset confusion matrix
valid_matrix = Confusion_Matrix(validxs, validys)
vmatrix = pd.DataFrame(valid_matrix)
print(vmatrix)

# Plot the heatmap confusion matrix
ax = plt.axes()
sn.heatmap(vmatrix, ax = ax, annot = True, cmap = 'Blues', fmt = 'g')
ax.set_title('Confusion Matrix (Validation Dataset)')
plt.xlabel('Predict')
plt.ylabel('Answer')
plt.show()

# Test Dataset confusion matrix
test_matrix = Confusion_Matrix(testxs, testys)
testmatrix = pd.DataFrame(test_matrix)
print(testmatrix)

# Plot the heatmap confusion matrix
ax = plt.axes()
sn.heatmap(testmatrix, ax = ax, annot = True, cmap = 'Greens', fmt = 'g')
ax.set_title('Confusion Matrix (Test Dataset)')
plt.xlabel('Predict')
```

```
plt.ylabel('Answer')
plt.show()
```

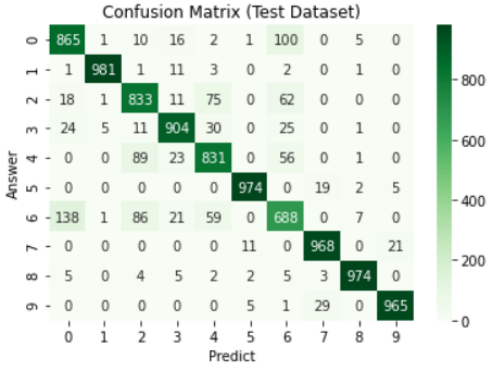
Confusion Matrix Plot:



(a) training dataset



(b) validation dataset



(c) test dataset

Conclusion:

To sum up, according to the above three datasets confusion matrix plots, we can notice that the error predictions are higher in categories (0,6), (2,4), (2,6) pairs **(0: top/ t-shirt, 2: pullover, 4: coat, 6: shirt)**, and these categories are all clothes-related pictures. Therefore, we can understand that the most reasons for the error predictions are as a result of the objects too similar with each other. It depends on many reasons, such as the picture is too blurry, the objects' shape is too similar and so on, which would have probability to lead to the model to make a wrong decision. To describe it thoroughly, because of the feature points, feature values captured by the keras model are same when the keras model analyzes the pictures, which would further cause the analysis of pictures have some deviation, and predict the pictures into the wrong positions. Even though we list the confusion matrix of the three datasets above, whereas, based on the problem-solving process, we can realize that the result of the test dataset is more meaningful, because it does not attend in the process of parameters adjustment. Hence, the confusion matrix result is more representative and significant.