Homework 5: Neural Networks for Classifying Fashion-MNIST

Alyssa Jin

03/13/2020

Abstract

In this assignment, I need to work with an analogous data set called Fashion-MNIST. The Fashion-MNIST data has exactly the same structure as the MNIST data. I need to try several different neural network architectures with different hyperparameters to try to achieve the best accuracy I can on the validation data. There are two parts in this assignment, which are fully-connected neural network (part I) and convolutional neural network (part II). The accuracy for part I is 88%, and for part II is 91%.

I. Introduction and Overview

I use python to analyze the Fashion-MNIST data, which has exactly the same structure as the MNIST data.
For part I, I train a fully-connected neural network to classify the images in the Fashion-MNIST data set. I try several different neural network architectures with different hyperparameters to try to achieve 88% accuracy on the validation data.The hyperparameters I adjust are the depth of the network, the width of the layers, the learning rate, the regularization parameters, and the activation functions.
For part II, I train a convolutional neural network to classify the images in the Fashion-MNIST data set. I try several different neural network architectures with different hyperparameters to try to achieve 91% accuracy on the validation data.The hyperparameters I adjust are the number of filters for convolutional layers, the kernel sizes, the strides, the padding options, the pool sizes for your pooling layers.

II. Theoretical Background

II.i. Fashion-MNIST

The Fashion-MNIST is a dataset of Zalandos article images consisting $60,000$ training images in the array X_train_full and $10,000$ test images in the array X_test, each of which is 28×28 grayscale image, associated with a label from 10 classes. The labels are contained in the vectors y_train_full and y_test. The values in vectors y_train_full and y_test are numbers from 0 to 9, which correspond to the 10 classes in the following way:

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255. The training and test data sets have 785 columns.

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

II.ii. Neural Network

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.

II.iii. Convolutional neural network

In deep learning, a convolutional neural network (CNN) is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, and financial time series.

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.

II.iv. Fully-connected neural network

A fully-connected neural network consists of a series of fully connected layers. A fully connected layer is a function from $\mathbb{R}^m$ to $\mathbb{R}^n$.

For mathematical form, let $x \in \mathbb{R}^m$ represent the input to a fully connected layer. Let $y_i \in \mathbb{R}$ be the i-th output from the fully connected layer. Then $y_i \in \mathbb{R}$ is computed as

$y_i = \sigma(w_1 x_1 + ... + w_m x_m)$. Here, σ is a nonlinear function, and the w_i are learnable parameters in the network. The full output y is then $y = \begin{cases} \sigma\left(w_{1,1}x_1 + \ldots + w_{1,m}x_m\right) \\ \vdots \\ \sigma\left(w_{n,1}x_1\right) + \ldots + w_{n,m}x_m\right) \end{cases}$.

III. Algorithm Implementation and Development

III.i. Fully-connected neural network

1. import *numpy*, *keras*, *argparse*, *matplotlib*, and *os*.
2. Use $path =$ "$data/fashion\_mnist.mat$" to load $fashion\_mnist.data$ to get array X_train_full, array X_test, vector y_train_full, and vector y_test.
3. Use *keras* to plot the accuracy of the data, with *epoch* in x-axis, *accuracy* in y-axis, blue curve shows the accuracy of data, and orange curve shows the cross-validation accuracy of data. The higher the accuracy is, the best the training is.
4. Use *keras* to plot the loss of the data, with *epoch* in x-axis, *loss* in y-axis, blue curve shows the loss of data, and orange curve shows the cross-validation loss of data. The lower the loss is, the best the training is.
5. Save the data of loss - *loss*, accuracy - *acc*, validation loss - val_loss, and cross-validation accuracy - val_acc by $save_history$, and save them in a txt file.
6. Use *model.add* to test different layers in the model, and return the best one.
7. import *scipy.io*, load data, and use *np.data* to return arrays of X_train, X_test, y_train, and y_test. Use $astype('float32')$ to convert the elements in arrays X_train and X_test to decimal in 32 dimensions. Use *reshape* to convert X_train and X_test to floating point numbers between 0 and 1 by dividing each by 255.0. Use $np_utils.to_categorical$ to converts the vector y_train, and y_test to binary class matrices $y\_label\_train\_OneHot$ and $y\_label\_test\_OneHot$, and return them.
8. Use $model.summary()$ to prints a summary representation of the model $(x\_train, y\_label\_train\_OneHot)$
9. Use *model.load_weights* to load the weights of "model/bp", "model.h5" to see if the model is successful.
10. Use *model.compile* to compile the multi-class classification of *accuracy*, and plot the model.
11. Use $H = model.fit$ to remove $5,000$ images from training data to use as validation data. Then, I have $55,000$ training examples in an array X_train and $5,000$ validation examples in an array X_valid.
12. Use *model.to_json* to return a representation of the model $(x\_train, y\_label\_train\_OneHot), (x\_test, y\_lab$ and save the weight of model by *model.save_weights*.
13. Use *model.evaluate* to returns the loss value & metrics values for the model.
14. Return the result by *pd.crosstab*.

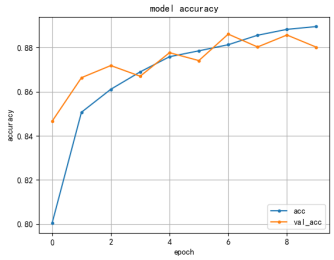

FIGURE 1. accuracy of model in part I

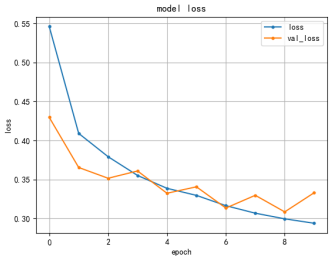

FIGURE 2. loss of model in part I

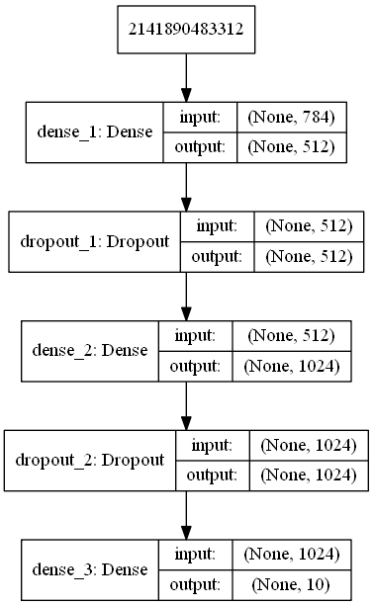

FIGURE 3. Process of finding the best hyper-parameters in part I

III.ii. Convolutional neural network

The process is almost the same as part I, except for building the model. I use $model.add(Conv2D)$ to try different number of filters (16, 32, 36, 64, 128) and kernel sizes. I use $model.add(Dropout)$ to add dropout layer to the model between existing layers, with rate 0.3 and 0.25. I use $model.add(MaxPooling2D)$ to max pooling operation for different pool sizes for pooling layers, e.g. (2,2). I use $model.add(Dence)$ to test different units (128, 1500, 10) and element-wise activation function $relu$ and $softmax$.

IV. Computational Results

IV.i. part I: fully-connected neural network

The best accuracy I find for the model is 88.9500% after I test 20 times with different hyperparameters, and I plot the number of accuracy and cross-validation in figure 1, loss and cross-validation loss in figure 2. The lowest accuracy is the first test, which is 80.0396%, and the highest is the last test, which is 88.9500%. I write the process of finding the best hyperparameters for model in figure 3. The hyperparameters i use for the final model are:

Hyperparameter	Best
The dimensionality of the output space	10
The drop out rate	30%
The activation functions	softmax

IV.i. part II: convolutional neural network

The best accuracy I find for the model is 90.6000% after I test 20 times with different hyperparameters, and I plot the number of accuracy and cross-validation in figure 4, loss and cross-validation loss in figure 5. The lowest accuracy is the first test, which is 68.2417%, and the highest is the last test, which is 90.6000%. I write the process of finding the best hyperparameters for model in figure 6. The hyperparameters i use for the final model are:

Hyperparameter	Best
The number of lters for your convolutional layers	128
The kernel sizes	$(3, 3)$
The activation functions	softmax
The dropout rate	30%
The dimensionality of the output space	10
The padding options	same

V. Summary and Conclusions

In this assignment, I use python to find the best hyperparameters get the highest accuracy. For part I, the lowest accuracy is the first test, which is 80.0396%, and the highest is the last test, which is 88.9500%. For part II, the lowest accuracy is the first test, which is 68.2417%, and the highest is the last test, which is 90.6000%. The difference between the lowest accuracy and highest accuracy shows that a convolutional layer is much more specialized, and efficient, than a fully-connected layer.In a fully-connected layer each neuron is connected to every neuron in the previous layer, and each connection has it's own weight. In contrast, in a convolutional layer each neuron is only connected to a few nearby (aka local) neurons in the previous layer, and the same set of weights and local connection layout) is used for every neuron.

Reference

1. A Beginner's Guide to Neural Networks and Deep Learning. (n.d.). Retrieved March 13, 2020, from https://pathmind.com/wiki/neural-network
2. Convolutional neural network. (2020, March 10). Retrieved March 13, 2020, from https://en.wikipedia.org/wiki/Convolutional_neural_network
3. Research, Z. (2017, December 7). Fashion MNIST. Retrieved March 13, 2020, from https://www.kaggle.com/zalando-research/fashionmnist
4. Zadeh, R. B., & Ramsundar, B. (n.d.). TensorFlow for Deep Learning. Retrieved March 13, 2020, from https://learning.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html

Appendix A. Python functions used and brief implementation explanation

1. $tf.keras.layers.Conv2D$: This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If use_bias is True, a bias vector is created

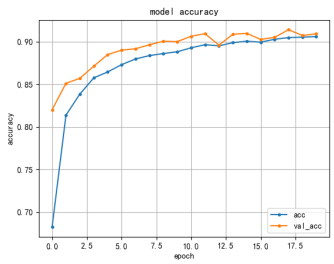

FIGURE 4. accuracy of model in part II

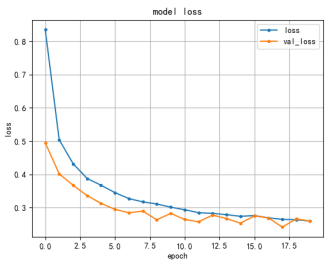

FIGURE 5. loss of model in part II

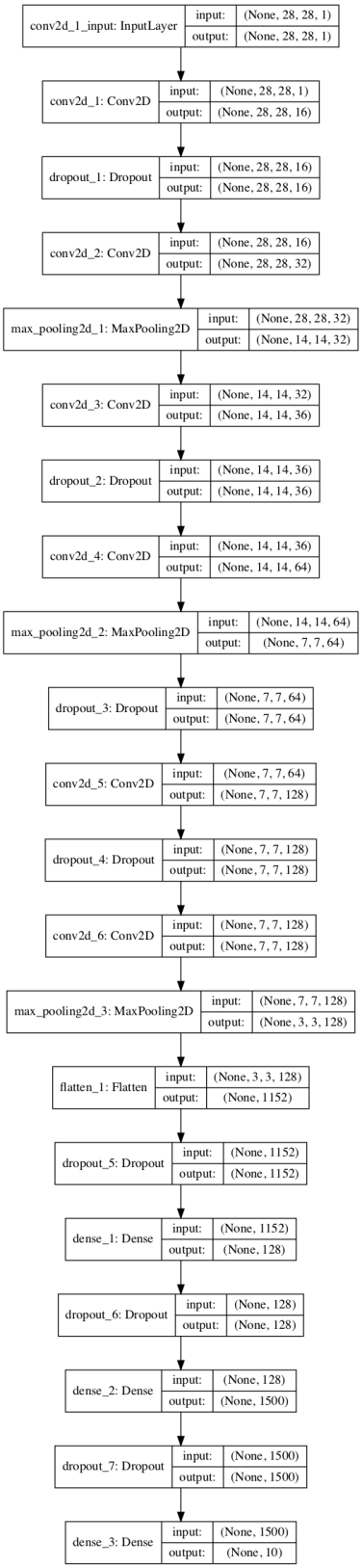

FIGURE 6. Process of finding the best hyper-parameters in part II

and added to the outputs. Finally, if *activation* is not *None*, it is applied to the outputs as well.

2. *pd.DataFrame*: Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns).
3. *model.add*: add layers in model.
4. *np.data*: return an array object satisfying the specified requirements.
5. *reshape*: Gives a new shape to an array without changing its data.Gives a new shape to an array without changing its data.
6. *np_utils*: Numpy-related utilities.
7. *to_categorical*: Converts a class vector (integers) to binary class matrix.
8. *model.summary*(): prints a summary representation of your model. For layers with multiple outputs, multiple is displayed instead of each individual output shape due to size limitations.
9. $model.load\_weights(filepath, by\_name = False)$: loads the weights of the model from a HDF5 file (created by *save_weights*). By default, the architecture is expected to be unchanged. To load weights into a different architecture (with some layers in common), use $by\_name = True$ to load only those layers with the same name.
10. *model.compile*: compile a multi-class classification problem.
11. $model.fit$: trains the model for a fixed number of epochs.
12. *model.to_json*(): returns a representation of the model as a JSON string. Note that the representation does not include the weights, only the architecture.
13. $model.save\_weights(filepath)$: saves the weights of the model as a HDF5 file.
14. *model.evaluate*: returns the loss value & metrics values for the model in test mode.
15. *pd.crosstab*: compute a simple cross tabulation of two (or more) factors. By default computes a frequency table of the factors unless an array of values and an aggregation function are passed.
16. *Dropout*: The Dropout layer is added to a model between existing layers and applies to outputs of the prior layer that are fed to the subsequent layer.
17. $MaxPooling2D$: max pooling operation for spatial data.
18. *Dense*: just your regular densely-connected NN layer.
19. *Flatten*: Flattens the input. Does not affect the batch size.

Appendix B. MATLAB codes

Code I: fully-connected neural network

```
from keras.models import Sequential
import numpy as np
from keras.utils import np_utils
from keras.layers.convolutional import Conv2D
from keras.layers.core import  Activation
from keras.layers.core import Dense,Dropout
from keras.layers import MaxPooling2D
from keras.layers.core import Flatten
from keras import backend as K
```

```
from keras import regularizers
from keras.layers.normalization import BatchNormalization
import argparse
import matplotlib
import numpy as np
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import os
from keras.callbacks import TensorBoard
from sklearn.model_selection import train_test_split
import keras

from keras.utils.vis_utils import plot_model
def plot_history(history, result_dir):
    plt.plot(history.history['acc'], marker='.')
    plt.plot(history.history['val_acc'], marker='.')
    plt.title('model accuracy')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
    plt.grid()
    plt.legend(['acc', 'val_acc'], loc='lower right')
    plt.savefig(os.path.join(result_dir, 'model_accuracy.png'))
    plt.close()

    plt.plot(history.history['loss'], marker='.')
    plt.plot(history.history['val_loss'], marker='.')
    plt.title('model loss')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.grid()
    plt.legend(['loss', 'val_loss'], loc='upper right')
    plt.savefig(os.path.join(result_dir, 'model_loss.png'))
    plt.close()
def save_history(history, result_dir):
    loss = history.history['loss']
    acc = history.history['acc']
    val_loss = history.history['val_loss']
    val_acc = history.history['val_acc']
    nb_epoch = len(acc)
    with open(os.path.join(result_dir, 'result.txt'), 'w') as fp:
        fp.write('epoch\tloss\tacc\tval_loss\tval_acc\n')
        for i in range(nb_epoch):
            fp.write('{}\t{}\t{}\t{}\t{}\n'.format(i, loss[i], acc[i], val_loss[i], val_
def build():
    model = Sequential()
    model.add(Dense(512, activation='relu',input_shape=(28*28,)))
```

```
    model.add(Dropout(rate=0.3))
    model.add(Dense(1024,activation='relu'))
    model.add(Dropout(rate=0.3))
    model.add(Dense(10, activation='softmax'))
    return model

def get_data(path):
    import scipy.io as scio
    data = scio.loadmat(path)
    x_train = np.array(data["X_train"]).astype('float32').reshape(60000, 28 * 28) / 255.
    y_train = np.array(data["y_train"]).T
    x_test = np.array(data["X_test"]).astype('float32').reshape(10000, 28 * 28) / 255.0
    y_test = np.array(data["y_test"]).T
    y_label_train_OneHot = np_utils.to_categorical(y_train)
    y_label_test_OneHot = np_utils.to_categorical(y_test)
    return (x_train,y_label_train_OneHot),(x_test,y_label_test_OneHot)


def main(path):
    (x_train,y_label_train_OneHot),(x_test,y_label_test_OneHot) = get_data(path)
    model = build()
    print(model.summary())
    try:
        model.load_weights("model/bp", "model.h5")
        print("File load successfully! continue on training model!")
    except:
        print("File load failed! Let's start a new model!")

    model.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.adam(),
                  metrics=['accuracy'])  # multi-classification for categorical_crossent
    plot_model(model, show_shapes=True, to_file=os.path.join("model/bp", 'model.png'))

    print('[INFO] training network...')
    H = model.fit(x_train, y_label_train_OneHot, validation_split=0.2, batch_size=100, e
    model_json = model.to_json()
    with open(os.path.join("model/bp", 'model.json'), 'w') as json_file:
        json_file.write(model_json)
    model.save_weights(os.path.join("model/bp", 'model.h5'))

    loss, acc = model.evaluate(x_test, y_label_test_OneHot, verbose=0)
    print('Test loss:', loss)
    print('Test accuracy:', acc)
    plot_history(H, "model/bp")
    save_history(H, "model/bp")
    prediction = model.predict_classes(x_test)  # predict category
    print(prediction.shape)
```

```
    y_label = np.array([np.argmax(label) for label in y_label_test_OneHot])
    print(y_label.shape)

    import pandas as pd
    result = pd.crosstab(y_label, prediction, rownames=['label'], colnames=['prediction'
    print(result)


if __name__ == '__main__':
    path = "data/fashion_mnist.mat"
    main(path)
```

Code II: convolutional neural network

```
from keras.models import Sequential
import numpy as np
from keras.utils import np_utils
from keras.layers.convolutional import Conv2D
from keras.layers.core import  Activation
from keras.layers.core import Dense,Dropout
from keras.layers import MaxPooling2D
from keras.layers.core import Flatten
from keras import backend as K
from keras import regularizers
from keras.layers.normalization import BatchNormalization
import argparse
import matplotlib
import numpy as np
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import os
from keras.callbacks import TensorBoard
from sklearn.model_selection import train_test_split
import keras

from keras.utils.vis_utils import plot_model
def plot_history(history, result_dir):
    plt.plot(history.history['acc'], marker='.')
    plt.plot(history.history['val_acc'], marker='.')
    plt.title('model accuracy')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
    plt.grid()
    plt.legend(['acc', 'val_acc'], loc='lower right')
    plt.savefig(os.path.join(result_dir, 'model_accuracy.png'))
    plt.close()
```

```
    plt.plot(history.history['loss'], marker='.')
    plt.plot(history.history['val_loss'], marker='.')
    plt.title('model loss')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.grid()
    plt.legend(['loss', 'val_loss'], loc='upper right')
    plt.savefig(os.path.join(result_dir, 'model_loss.png'))
    plt.close()
def save_history(history, result_dir):
    loss = history.history['loss']
    acc = history.history['acc']
    val_loss = history.history['val_loss']
    val_acc = history.history['val_acc']
    nb_epoch = len(acc)
    with open(os.path.join(result_dir, 'result.txt'), 'w') as fp:
        fp.write('epoch\tloss\tacc\tval_loss\tval_acc\n')
        for i in range(nb_epoch):
            fp.write('{}\t{}\t{}\t{}\t{}\n'.format(i, loss[i], acc[i], val_loss[i], val_
def build():
    model = Sequential()
    model.add(Conv2D(filters=16, kernel_size=(5, 5), input_shape=(28, 28, 1), activation
    model.add(Dropout(rate=0.3))
    model.add(Conv2D(filters=32,kernel_size=(3,3),activation='relu',padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(filters=36, kernel_size=(5, 5), activation='relu', padding='same'))
    model.add(Dropout(0.3))
    model.add(Conv2D(filters=64,kernel_size=(3,3),activation='relu',padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(filters=128,kernel_size=(3,3),activation='relu',padding='same'))
    model.add(Dropout(rate=0.3))
    model.add(Conv2D(filters=128,kernel_size=(3,3),activation='relu',padding='same'))
    model.add(MaxPooling2D(pool_size=(2,2)))

    model.add(Flatten())
    model.add(Dropout(rate=0.3))
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(rate=0.5))
    model.add(Dense(1500,activation='relu'))
    model.add(Dropout(rate=0.3))
    model.add(Dense(10, activation='softmax'))
    return model

def get_data(path):
```

```
    import scipy.io as scio
    data = scio.loadmat(path)
    x_train = np.array(data["X_train"]).astype('float32').reshape(60000,28,28,1)/255.0
    y_train = np.array(data["y_train"]).T
    x_test = np.array(data["X_test"]).astype('float32').reshape(10000,28,28,1)/255.0
    y_test = np.array(data["y_test"]).T
    y_label_train_OneHot = np_utils.to_categorical(y_train)
    y_label_test_OneHot = np_utils.to_categorical(y_test)
    return (x_train,y_label_train_OneHot),(x_test,y_label_test_OneHot)


def main(path):
    (x_train,y_label_train_OneHot),(x_test,y_label_test_OneHot) = get_data(path)
    model = build()
    print(model.summary())
    try:
        model.load_weights("model/cnn", "model.h5")
        print("File load successfully! continue on training model!")
    except:
        print("File load failed! Let's start a new model!")

    model.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.adam(),
                  metrics=['accuracy'])  # categorical_crossentropy
    plot_model(model, show_shapes=True, to_file=os.path.join("model/cnn", 'model.png'))

    print('[INFO] training network...')
    H = model.fit(x_train, y_label_train_OneHot, validation_split=0.2, batch_size=100, e
    model_json = model.to_json()
    with open(os.path.join("model/cnn", 'model.json'), 'w') as json_file:
        json_file.write(model_json)
    model.save_weights(os.path.join("model/cnn", 'model.h5'))

    loss, acc = model.evaluate(x_test, y_label_test_OneHot, verbose=0)
    print('Test loss:', loss)
    print('Test accuracy:', acc)
    plot_history(H, "model/cnn")
    save_history(H, "model/cnn")
    prediction = model.predict_classes(x_test)  # predict category
    print(prediction.shape)
    y_label = np.array([np.argmax(label) for label in y_label_test_OneHot])
    print(y_label.shape)

    import pandas as pd
    result = pd.crosstab(y_label, prediction, rownames=['label'], colnames=['prediction'
    print(result)
```

```
if __name__ == '__main__':
    path = "data/fashion_mnist.mat"
    main(path)
```