QUEENSLAND UNIVERSITY OF TECHNOLOGY

ADVANCED TOPICS IN ARTIFICIAL INTELLIGENCE

Siamese Network

IFN 680: Assignment 2

Student Name:

Jianwei Tang – N10057862

Yongrui Pan – N10296255

1 PROJECT OVERVIEW

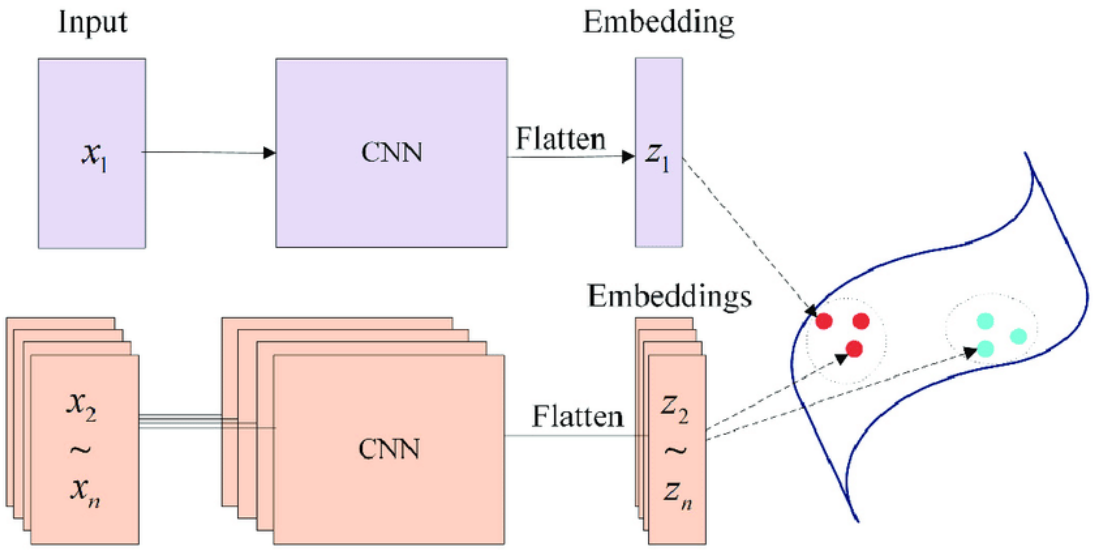
This project is an implementation of a deep neural network classifier to predict whether two images belong to the same class.

1.1 DATASET

The dataset in this project is a built-in dataset in keras.datasets.fashion_mnist.load_data, it is called Fashion-MNIST. The dataset contains images labelled with classes including ["top", "trouser", "pullover", "coat", "sandal", "ankle boot", "dress", "sneaker", "bag", "shirt"]. They are all small (28x28) grayscale images.

1.2 SIAMESE NETWORK ARCHITECTURE

A Siamese network consists of two identical subnetworks that share the same weights followed by a distance calculation layer. The graph below shows the structure.



1.3 IMPLEMENTATION ENVIRONMENT

This project is using python version 3.7 with keras library. keras is the core part in this project. It involves construction of the network and the classifier. NumPy and Matplotlib.pyplot are also used and they are for datasets slicing and figure plotting respectively.

Details are shown below.

```
1. from __future__ import absolute_import, division, print_function, unicode_li
   terals
2.
3. import random
4. #rom keras.layers import Input, Conv2D, Lambda, merge, Dense, Flatten,MaxPoo
   ling2D,GlobalAveragePooling2D,Activation
5. import tensorflow as tf
6. from tensorflow import keras
7. from keras.layers import Input, Flatten, Dense, Dropout, Lambda, MaxPooling2
   D
8. from keras.models import Model
```

```
from keras.optimizers import RMSprop
from keras import backend as K
from keras.layers.convolutional import Conv2D
from keras.layers import LeakyReLU
from keras.regularizers import l2
from keras.models import Model, Sequential

from tensorflow.keras import regularizers

import numpy as np
import matplotlib.pyplot as plt
```

1.4 PROJECT OBJECTIVES

Evaluate the generalization capability of your network by

1. testing it with pairs from the set of images with labels ["top", "trouser", "pullover", "coat", "sandal", "ankle boot"]
2. testing it with pairs from the set of images with labels ["top", "trouser", "pullover", "coat", "sandal", "ankle boot"] union ["dress", "sneaker", "bag", "shirt"]
3. testing it with pairsfrom the set of images with labels in ["dress", "sneaker", "bag", "shirt"]

2 CODING EXPLANATIONS

2.1 DATA LOADING AND OBSERVATION

Firstly, we load the data and have a glimpse on it.

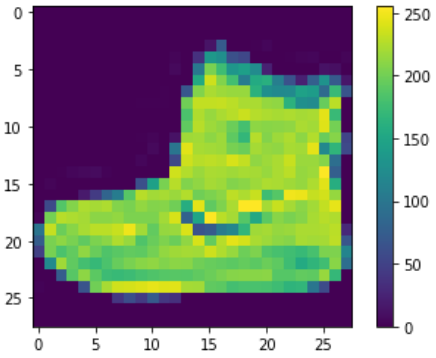
```
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()

print('train_images : ', x_train.shape, x_train.dtype)
print('train_labels : ', y_train.shape, y_train.dtype)
print('test_images : ', x_test.shape, x_test.dtype)
print('test_labels : ', y_test.shape, y_test.dtype)
```

```
train_images :  (60000, 28, 28) uint8
train_labels :  (60000,) uint8
test_images :  (10000, 28, 28) uint8
test_labels :  (10000,) uint8
```

It shows there are 60000 images of training images and 10000 images of test images.

Let us have a look at the first training image.

```
plt.figure()
plt.imshow(x_train[0, :])
plt.colorbar()
plt.grid(False)
plt.show()
```



2.2 DATA NORMALIZATION

We normalize the data between 0–1 by dividing train data and test data with 255, then we can take a pixel with value “1” for black color and a pixel with value “0” for white color.

```
1. x_train = x_train.astype('float32')
2. x_test = x_test.astype('float32')
3. x_train = x_train / 255.0
4. x_test = x_test / 255.0
```

2.3 DATA SLICING

We slice the data according their label for later usage.

```
1. # split labels ["top", "trouser", "pullover", "coat", "sandal", "ankle boot"
   ] to train set
2. digit_indices = [np.where(y_train == i)[0] for i in {0,1,2,4,5,9}]
3. digit_indices = np.array(digit_indices)
4.
5. # length of each column
6. n = min([len(digit_indices[d]) for d in range(6)])
7.
8. # Keep 80% of the images with labels ["top", "trouser", "pullover", "coat",
   "sandal", "ankleboot"] for training (and 20% for testing)
9. train_set_shape = n * 0.8
10. test_set_shape = n * 0.2
11. y_train_new = digit_indices[:, :int(train_set_shape)]
12. y_test_new = digit_indices[:, int(train_set_shape):]
13.
14. # Keep 100% of the images with labels in ["dress", "sneaker", "bag", "shirt"
   ] for testing
15. digit_indices_t = [np.where(y_train == i)[0] for i in {3,6,7,8}]
16. y_test_new_2 = np.array(digit_indices_t)
17.
18. print(y_train_new.shape)
19. print(y_test_new.shape)
20. print(y_test_new_2.shape)
```

```
(6, 4800)
(6, 1200)
(4, 6000)
```

2.4 CREATE PAIRS

In order to build a classifier which can identify whether two images belong to a same class, we need to create pairs of images throughout the datasets.

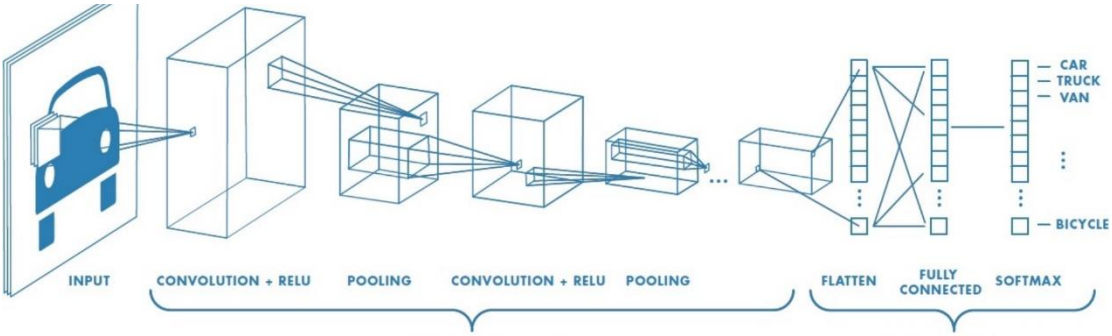
The method we doing it is: 1) For every image belongs to each given class, we pick the image next to it and form a pair. For example, in the class of “top”, the first image and the second image will form a pair, and the second image will form a pair with the third image… Those pairs will be positive pairs. 2) Meanwhile, we pick an image belongs to another class and form a pair. For example, the first image in the class of “top” will form a pair with the first image in the class of “pullover”. Those pairs will be negative pairs. 3) We assign labels to each combination of positive pair and negative pair as [1,0].

```
def create_pairs(x, digit_indices):
    '''''Positive and negative pair creation.
    Alternates between positive and negative pairs.
    '''
    pairs = []
    labels = []

    class_num = digit_indices.shape[0]
    for d in range(class_num):
        for i in range(int(digit_indices.shape[1])-1):
            z1, z2 = digit_indices[d][i], digit_indices[d][i + 1]
            pairs += [[x[z1], x[z2]]]
            inc = random.randrange(1, class_num)
            dn = (d + inc) % class_num
            z1, z2 = digit_indices[d][i], digit_indices[dn][i]
            pairs += [[x[z1], x[z2]]]
            labels += [1, 0]
    return np.array(pairs), np.array(labels)
# two image
tr_pairs, tr_y = create_pairs(x_train, y_train_new)
tr_pairs = tr_pairs.reshape(tr_pairs.shape[0], 2, 28, 28, 1)
print(tr_pairs.shape)

te_pairs_1, te_y_1 = create_pairs(x_train, y_test_new)
te_pairs_1 = te_pairs_1.reshape(te_pairs_1.shape[0], 2, 28, 28, 1)
print(te_pairs_1.shape)

te_pairs_2, te_y_2 = create_pairs(x_train, y_test_new_2)
te_pairs_2 = te_pairs_2.reshape(te_pairs_2.shape[0], 2, 28, 28, 1)
print(te_pairs_2.shape)
```

```
(57588, 2, 28, 28, 1)
(14388, 2, 28, 28, 1)
(47992, 2, 28, 28, 1)
```

2.5 BASE NETWORK CREATION

For the base network, we get the insight from the structure below.

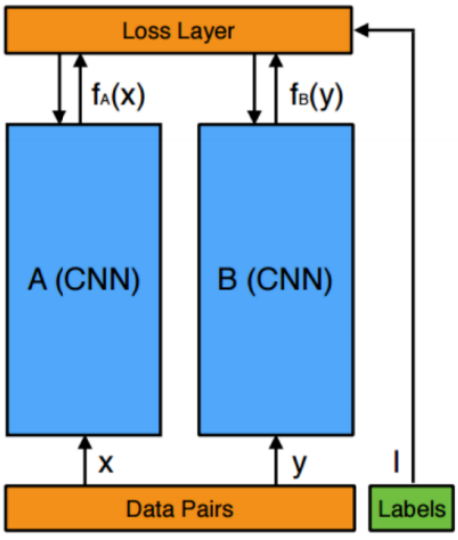


At first, we have a convolution+relu layer with a larger size filter 7*7 and followed by a maxpooling layer which reduce parameters to reduce computation and overfitting. Then, there is another convolution+relu layer with a smaller size filter 3*3. Then the flatten layer flatten the multi-dimensions to one dimension for the followed fully connect layer. Also, in several layers, regularizers are utilized for reducing overfitting.

```
1.  def create_base_network(input_shape):
2.      '''''Base network to be shared (eq. to feature extraction).
3.      '''
4.      input = Input(shape=input_shape)
5.      print(input)
6.      x = Conv2D(32, (7, 7), activation='relu', input_shape=input_shape, kerne
    l_regularizer=regularizers.l2(0.01),
7.                 bias_regularizer=regularizers.l1(0.01))(input)
8.      x = MaxPooling2D()(x)
9.      x = Conv2D(64, (3 ,3),activation='relu', kernel_regularizer=regularizers
    .l2(0.01),
10.                bias_regularizer=regularizers.l1(0.01))(x)
11.
12.     x = Flatten()(x)
13.     x = Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.0
    1),
14.                bias_regularizer=regularizers.l1(0.01))(x)
15.
16.     return Model(input, x)
17.
18. input_shape = (28,28,1)
19.
20. base_network = create_base_network(input_shape)
21.
22. input_a = Input(shape=input_shape)
23. input_b = Input(shape=input_shape)
24.
25. # because we re-use the same instance `base_network`,
26. # the weights of the network
27. # will be shared across the two branches
28. processed_a = base_network(input_a)
29. processed_b = base_network(input_b)
30. print(base_network.summary())
```

2.6 LOSS LAYER

A loss layer is needed in the siamese network.



```
# add a lambda layer
distance = Lambda(euclidean_distance,
                  output_shape=eucl_dist_output_shape)([processed_a, processed_b])

model = Model([input_a, input_b], distance)
```

2.7 MODEL TRAINING

Now we have completed the Siamese network structure. Model can be trained with the training dataset.

```
# train
rms = RMSprop()
model.compile(loss=contrastive_loss, optimizer=rms, metrics=[accuracy])
history = model.fit([tr_pairs[:, 0], tr_pairs[:, 1]], tr_y,
          batch_size=128,
          epochs=epochs,
          validation_data=([te_pairs_1[:, 0], te_pairs_1[:, 1]], te_y_1))
```

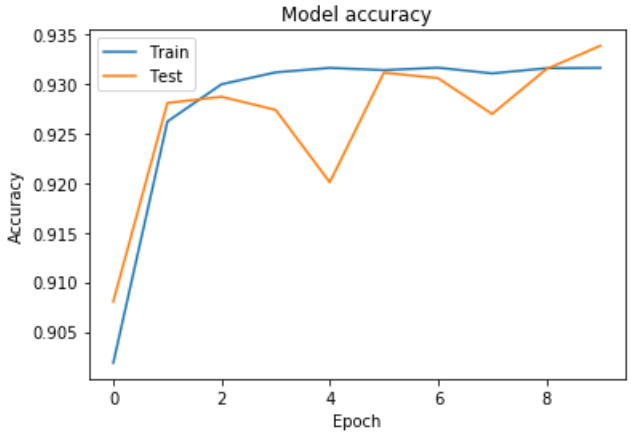
2.8 MAKING PREDICTION

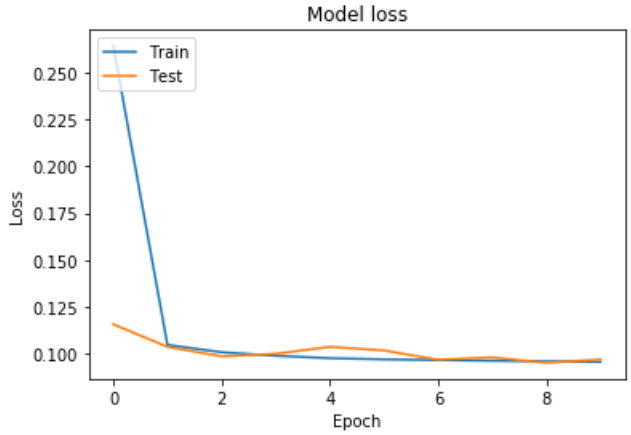
The model is able to make predictions since now. It takes test data as input. The accuracy can be evaluated based its prediction.

```
y_pred = model.predict([tr_pairs[:, 0], tr_pairs[:, 1]])
tr_acc = compute_accuracy(tr_y, y_pred)
y_pred = model.predict([te_pairs_1[:, 0], te_pairs_1[:, 1]])
te_acc = compute_accuracy(te_y_1, y_pred)
```

3 MODEL EVALUATION

OBJECTIVE 1: testing it with pairs from the set of images with labels ["top", "trouser", "pullover", "coat", "sandal", "ankle boot"]






```
* Accuracy on training set: 93.66%
* Accuracy on test set: 93.38%
```

OBJECTIVE 2: testing it with pairs from the set of images with labels ["top", "trouser", "pullover", "coat", "sandal", "ankle boot"] union ["dress", "sneaker", "bag", "shirt"]

```
* Accuracy on training set: 93.36%
* Accuracy on test set: 83.94%
```

OBJECTIVE 3: testing it with pairsfrom the set of images with labels in ["dress", "sneaker", "bag", "shirt"]

```
* Accuracy on training set: 93.30%
* Accuracy on test set: 74.85%
```