IFN 680 Advanced Topics in Artificial Intelligence

Assignment 2-Siamese Network

Student name: JIN ZHAO (n9800174)
Yuchen Jiang (N9573950) (Submitter)
Jinning Guo (N9858598)

1. Load the fashion_mnist data

In this project, we use `tensorflow.keras.datasets.fashion_mnist.load_data` to load the fashion_mnist dataset.

```
from tensorflow.keras.datasets import fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

Concat train and test data:

```
train_images = np.concatenate((train_images, test_images))

train_labels = np.concatenate((train_labels, test_labels))
```

Count of each class:

```
class_count = [len(np.where(train_labels == i)[0]) for i in range(10)]
```

The statistics of the fashion_mnist data set are shown in the following table.

In the table, there are 70,000 handwritten digital pictures. There is the same number of 7000 images in each class.

Label	0	1	2	3	4	5	6	7	8	9	Total
Count	7000	7000	7000	7000	7000	7000	7000	7000	7000	7000	70000

2. Split the dataset

```
train_classes = [0, 1, 2, 4, 5, 9]

test_classes = [3, 7, 8, 6]
```

Based on the project request, we divided the 10 classes into two parts: train_classes and test_classes.

train_classes represents ["top", "trouser", "pullover", "coat", "sandal", "ankle boot"];

test_classes represents ["dress", "sneaker", "bag", "shirt"];

First of all, we get indices of each classes in train classes:

```
class_indices = [np.where(train_labels == i)[0] for i in train_classes]
```

get indices of each classes in test sets:

```
class_indices = [np.where(train_labels == i)[0] for i in test_classes]
```

In this project, we used the function which is shown below to create positive and negative pairs:

```
def create_pairs(x, class_indices):
    '''Positive and negative pair creation.
    Alternates between positive and negative pairs.
    '''

    pairs = []
```

```
    labels = []
    n = min([len(class_indices[d]) for d in range(len(class_indices))]) -
1
    for d in range(len(class_indices)):
        for i in range(n):
            z1, z2 = class_indices[d][i], class_indices[d][i + 1]
            pairs += [[x[z1], x[z2]]]
            inc = random.randrange(1, len(class_indices))
            dn = (d + inc) % len(class_indices)
            z1, z2 = class_indices[d][i], class_indices[dn][i]
            pairs += [[x[z1], x[z2]]]
            labels += [1, 0]
    return np.array(pairs), np.array(labels)
```

After that, we create pairs for train tests and test sets:

```
pairs, y = create_pairs(train_images, class_indices)
# using 80% of date for train others for testing
train_y_len = round(len(y)*0.8)
# round shuffle the train date sets
shuffled_pairs, shuffled_y = shuffle(pairs, y)
train_pairs, train_y = shuffled_pairs[0:train_y_len], shuffled_y[0:train_y
_len]
test_pairs, test_y = shuffled_pairs[train_y_len:], shuffled_y[train_y_len:
]
# create pairs for ["dress", "sneaker", "bag", "shirt"]
ftest_pairs, ftest_y = create_pairs(train_images, class_indices)
```

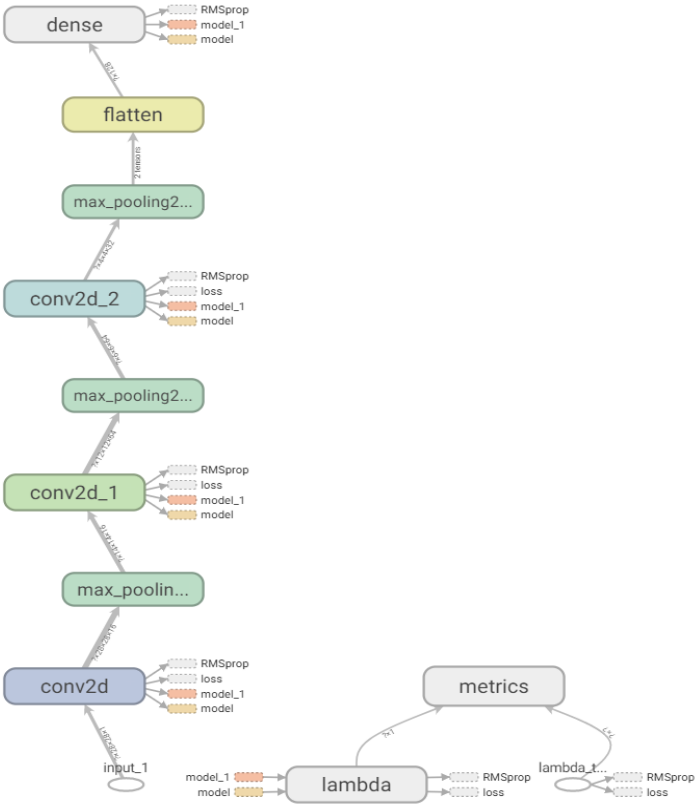
3. Implementation and testing of the contrastive loss function

In order to achieve the contrast loss function, our group adopted the following design. And we set the margin equal to 1. (Retrieved from https://github.com/keras-team/keras/blob/master/examples/mnist_siamese.py)

```
def contrastive_loss(y_true, y_pred):
margin = 1
sqaure_pred = K.square(y_pred)
margin_square = K.square(K.maximum(margin - y_pred, 0))
return K.mean((1 - y_true) * sqaure_pred + y_true * margin_square)
```

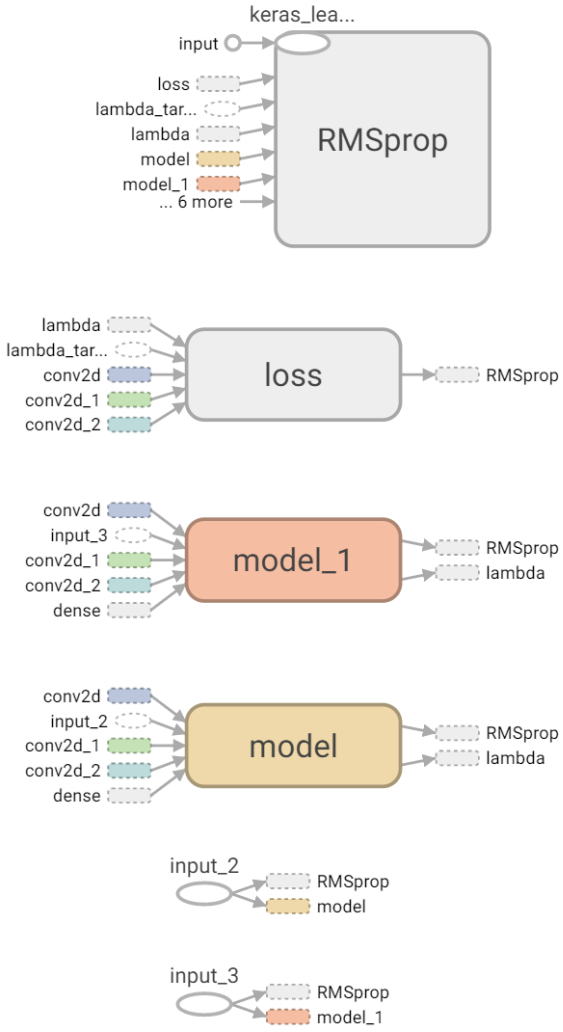
4. Create a Siamese network

Our team adopt the CNN network architecture used in the solution of Week 08 practical. Shown in the following figure, the CNN network has three convolutional layers, three max_pooling layers, and one dense layer. For each convolutional layer, we use different generalization models.



5. The training of Siamese network

```
history = model.fit([train_pairs[:, 0], train_pairs[:, 1]], train_y, batch
_size=128,
epochs=epochs, validation_data=([ftest_pairs[:, 0], ftest_pairs[:, 1]], ft
est_y), callbacks=[tensorboard_callback])
```

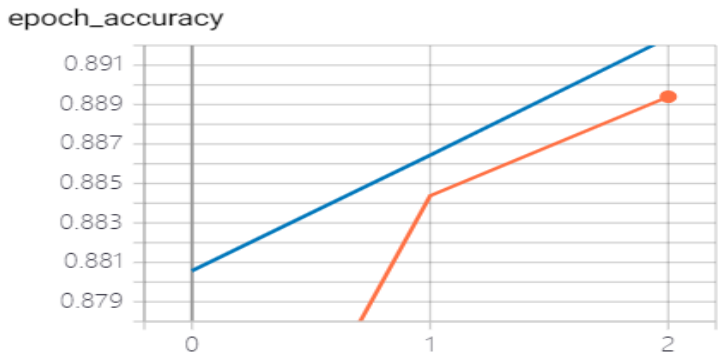


Verify the data

Train/test accuracy in data set ["top", "trouser", "pullover", "coat", "sandal", "ankle boot"]

Shown in the following figure, the orange line represents the accuracy of training and the blue line represents the accuracy of testing.

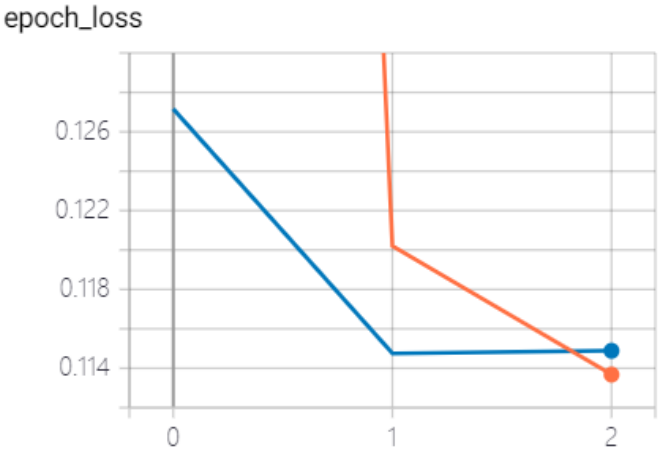
We can conclude from the figure that the accuracy of training after learning twice increased linearly from 0.881 to 0.891, while the accuracy of the test increased to 0.889.



Train/test loss in data set ["top", "trouser", "pullover", "coat", "sandal", "ankle boot"]

Shown in the following figure, the orange line represents the accuracy of training and the blue line represents the accuracy of testing.

We can conclude from the figure that the loss of training after learning twice decreased from 0.126 to 0.114, while the loss of the test decreased to 114.

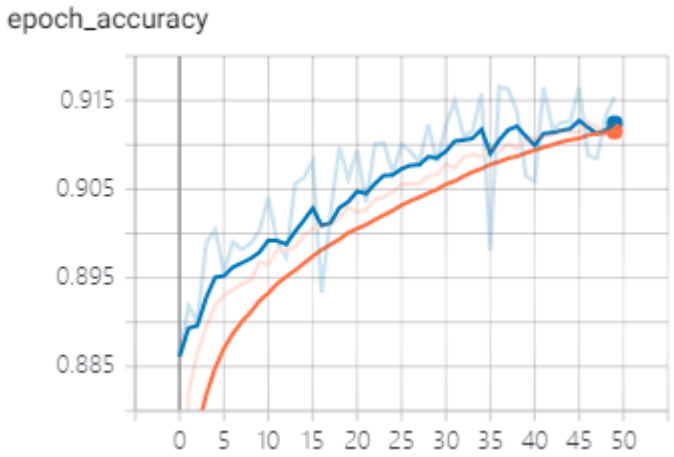


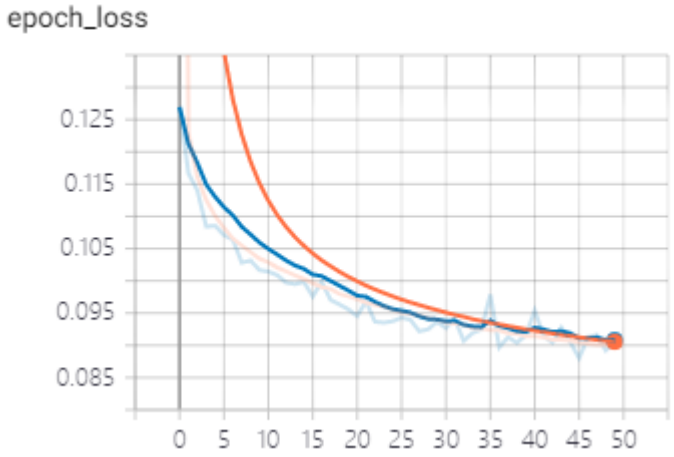
For further work, we changed the number of epoches：

```
epochs = 50
```

Compared with learning 3 times, we increased the learning times to 50. Shown in the following figure, the orange line represents the accuracy of training and the blue line represents the accuracy of testing.

We can conclude that when the number of learning increases indefinitely, the accuracy is infinitely close to 1, which means the process of training and testing is relative stable.





6. Evaluate the capability of generalization

Based on the project request,

(1) Firstly, we need test it with pairs from the class ["top", "trouser", "pullover", "coat", "sandal", "ankle boot"] which is [0,1,2,4,5,9].

```
y_pred = model.predict([pairs_for_012459[:, 0], pairs_for_012459[:, 1]])

tr_acc = compute_accuracy(label_for_012459, y_pred)

print('* Accuracy on 012459 set: %0.2f%%' % (100 * tr_acc))
```

(2) Secondly, we need to test the class ["top", "trouser", "pullover", "coat", "sandal", "ankle boot"] union ["dress", "sneaker", "bag", "shirt"] which is [0,1,2,3,4,5,6,7,8,9]

```
class_indices = [np.where(train_labels == i)[0] for i in range(10)]

all_pairs, all_label = create_pairs(train_images, class_indices)

y_pred = model.predict([all_pairs[:, 0], all_pairs[:, 1]])

tr_acc = compute_accuracy(all_label, y_pred)

print('* Accuracy on all set: %0.2f%%' % (100 * tr_acc))
```

(3) Finally, we need to test the class ["dress", "sneaker", "bag", "shirt"] which is [0,1,8,9]

```
y_pred = model.predict([ftest_pairs[:, 0], ftest_pairs[:, 1]])

tx_acc = compute_accuracy(ftest_y, y_pred)

print('* Accuracy on final_test set: %0.2f%%' % (100 * tx_acc))
```

(4) Outcomes in epochs = 3 and epochs = 50

Accuracy on 012459	88.85%	91.45%
Accuracy on 0123456789	81.95%	82.74%
Accuracy on 0189	80.42%	77.74%

Comparing the results of epochs equal to 3 and 50, we found that the accuracy was very close. 50 times of training is more accurate, but it takes longer. In this project, we decided to use epochs = 3 to training and test the accuracy, because 3 times of training is enough to get the high accuracy.

7. Visualize Tool (Tensorboard)

TensorBoard is a Tensorflow visualization tool that can be used to present TensorFlow images, plot quantitative indicators generated by images, and attach data.

TensorBoard summarizes the data through the summary statement.(Retrieved from https://www.tensorflow.org/api_guides/python/summary)

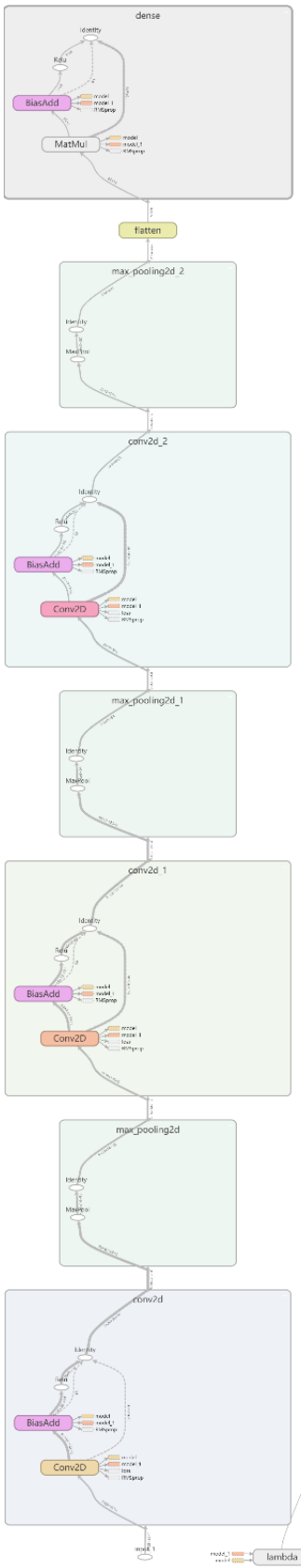
First, make sure the version number is correct. In this project, the version of the tensorboard that we used is 3.7.3.

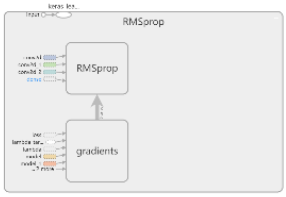
```
log_dir = "logs/fit/"
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=log_dir, histogram_freq=1)
```

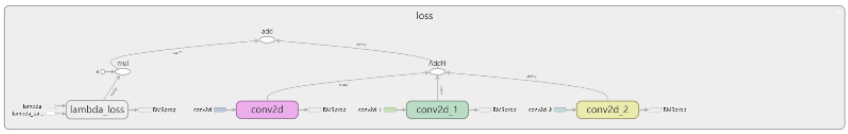
After the command is executed, a new file is generated in the specified directory.

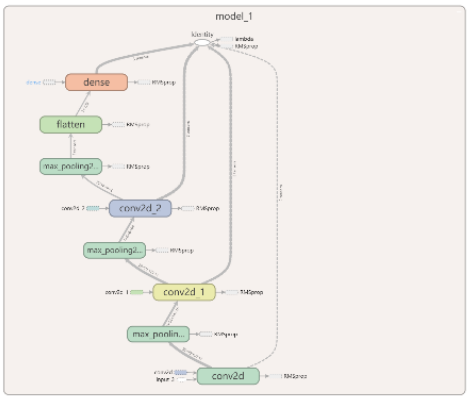
Open console command, input command tensorboard --logdir="logs/fit"

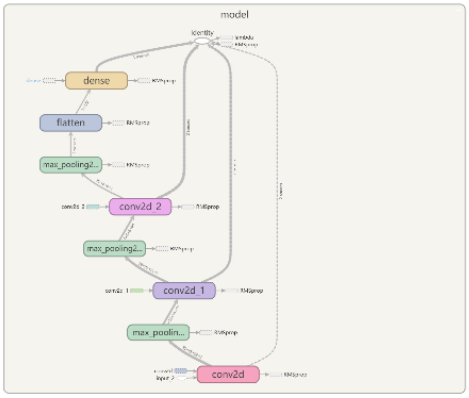
Open a browser and enter the host address http://localhost:6006,


dense
Identity
Relu
BiasAdd
MatMul
flatten
max_pooling2d_2
MaxPool
conv2d_2
Identity
Relu
BiasAdd
Conv2D
max_pooling2d_1
MaxPool
conv2d_1
Identity
Relu
BiasAdd
Conv2D
max_pooling2d
MaxPool
conv2d
Identity
Relu
BiasAdd
Conv2D
metrics
lambda


RMSprop
RMSprop
gradients


loss
add
mul
AddN
lambda_loss
conv2d
conv2d_1
conv2d_2


model_1
dense
flatten
max_pooling2...
conv2d_2
max_pooling2...
conv2d_1
max_poolin...
conv2d


model
dense
flatten
max_pooling2...
conv2d_2
max_pooling2...
conv2d_1
max_poolin...
conv2d