UM-SJTU Joint Institute
Problem Solving with AI Techniques
(Ve593)

Project Three
Convolutional Neural Networks for Visual Recognition

Ming Xingyu 517370910224

Date: 2020/11/21

Note: The discreption of the functions I designed for each part, please refer to the *README.md*.

1 Classification on Fashion-MNIST

Following the requirement of the project manual, I explored the data and then design the model.

1.1 Explore the data

For the data set, it consists of 10 different categories, as mentioned in the manual. Then, I find the shape of the data in each category is (28,28,1).

And the size of the train set and test set is 60000 and 10000 respectively.

Also, I plotted the sample picture for each category. The following are the four sample pictures.

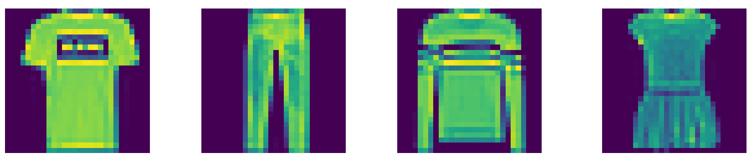

Figure 1: Sample train data in Fashion-MNIST. From left to right, category 0 to 3.

The other figures are generated and inside the zip file.

1.2 ANN

I designed and trained the model with 1 input layer, 1 hidden layer, 1 output layer as follows,

```
model=keras.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(28,28)))
model.add(tf.keras.layers.Dense(128,activation='relu'))
model.add(tf.keras.layers.Dense(10,activation='softmax'))
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
                                                metrics=['accuracy'])
model.fit(x_train,y_train,batch_size=200,epochs=10)
```

Then I evaluate the above ANN with the test data set.

```
model.evaluate(x_test,y_test)
```

The result is

```
313/313 [==============================] - 0s 578us/step - loss: 0.5833 - accuracy: 0.8175
```

Obviously, the result is not quite good, according to the evaluation.

1.3 CNN

Now, I built and trained the CNN, which involves convolution and pooling layer compared with the previous method.

```
model2=keras.Sequential()
model2.add(tf.keras.layers.Conv2D(filters=10,kernel_size=2,activation='tanh',
                                input_shape=(28,28,1)))
model2.add(tf.keras.layers.MaxPool2D(pool_size=(6,3)))
model2.add(tf.keras.layers.Flatten())#make the data into array
model2.add(tf.keras.layers.Dense(128,activation='relu'))
model2.add(tf.keras.layers.Dense(10,activation='softmax'))
model2.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
                                        metrics=['accuracy'])
model2.fit(x_train1,y_train,batch_size=200,epochs=10)
```

And the evaluation step, provided me with the accuracy as

```
313/313 [==============================] - 0s 1ms/step - loss: 0.3526 - accuracy: 0.8689
```

This is slightly higher than the ANN model, hence I will improve it in the following part.

1.4 Improvement

Since the CNN can be improved to be more accurate. I designed and trained the following model.

```
model3=keras.Sequential()
model3.add(tf.keras.layers.Conv2D(filters=50,kernel_size=3,
    activation='relu',input_shape=(28,28,1),padding='SAME'))
model3.add(tf.keras.layers.MaxPool2D(pool_size=(2,2)))
model3.add(tf.keras.layers.Conv2D(filters=50,kernel_size=2,
                        activation='relu',padding='SAME'))
model3.add(tf.keras.layers.MaxPool2D(pool_size=(2,2)))
model3.add(tf.keras.layers.Dropout(rate=0.25))
model3.add(tf.keras.layers.Flatten())
model3.add(tf.keras.layers.Dense(128,activation='relu'))
model3.add(tf.keras.layers.Dense(64,activation='tanh'))
model3.add(tf.keras.layers.Dense(10,activation='softmax'))
model3.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
                                                metrics=['accuracy'])
model3.fit(x_train1,y_train,batch_size=500,epochs=10,validation_split=0.25)
```

Explanations for the modifications on CNN:

- The first layer is to identify some initial patterns for the input pictures, for example, the curve of shoes and shapes of T-shirts.
- Also, I make the size of kernel smaller to speed up the training process.
- After pooling, I would like to find the high-dimensional features of the input figure, so I added another convolution layer.
- Moreover, since there are 10 categories, I used 50 filters to identify their patterns, which I think is more than enough.
- The following dense layers add some non-linear functions into the network.
- I also added the validation part for this model.

Admittedly, the result is quite good compared with the above two model.

```
313/313 [==============================] - 2s 6ms/step - loss: 0.2855 -
↪ accuracy: 0.9006
```

2 Recognition of Traffic Signs

2.1 Explore the data

Firstly, I downloaded and extract the three data sets. Then, I check the shape of the data and the number of the sets.

The shape of the data is (32,32,3), which is an RGB picture. The size of the sets are 34799 pictures for training, 12630 pictures for testing and 4410 pictures for validation.

Also, I found out the number of categories is 43, and plot samples for each of them. The following is the sample plots.

Figure 2: Sample train data in German traffic signs. From left to right, category 16, 17, 18, 21.

2.2 CNN1

***Note:** The model in the code file has been rerun.

2.2.1 Model

Without any data pre-processing, I designed and trained several models. After tuning the parameter and modify the optimizer, I found this model that has the validation accuracy larger than 95%.

```
adam=tf.keras.optimizers.Adam(
    learning_rate=0.001,
    beta_1=0.8,
    beta_2=0.999,
    epsilon=1e-8,
    amsgrad=False,
    name="Adam"
)
model1=keras.Sequential()
model1.add(tf.keras.layers.Conv2D(filters=150,kernel_size=4,activation='tanh',
                        input_shape=(32,32,3),padding='SAME'))
model1.add(tf.keras.layers.MaxPool2D(pool_size=(2,2)))
model1.add(tf.keras.layers.Conv2D(filters=100,kernel_size=4,
                        activation='sigmoid',padding='SAME'))
model1.add(tf.keras.layers.MaxPool2D(pool_size=(2,2)))
model1.add(tf.keras.layers.Conv2D(filters=100,kernel_size=2,
                            activation='relu',padding='SAME'))
model1.add(tf.keras.layers.MaxPool2D(pool_size=(2,2)))
model1.add(tf.keras.layers.Conv2D(filters=100,kernel_size=2,
                            activation='relu',padding='SAME'))
model1.add(tf.keras.layers.MaxPool2D(pool_size=(2,2)))
model1.add(tf.keras.layers.Dropout(rate=0.25))
model1.add(tf.keras.layers.Flatten())
model1.add(tf.keras.layers.Dense(128,activation='relu'))
model1.add(tf.keras.layers.Dense(128,activation='tanh'))
model1.add(tf.keras.layers.Dense(43,activation='softmax'))
model1.compile(optimizer=adam,loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])
model1.fit(x_train,y_train,batch_size=500,epochs=10,
validation_data=(x_valid,y_valid),shuffle=True,validation_freq=1)
```

Also, I can have the structure of the model being printed as follows.

```
Model: "sequential_23"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_70 (Conv2D)           (None, 32, 32, 150)       7350
_________________________________________________________________
max_pooling2d_68 (MaxPooling2D) (None, 16, 16, 150)    0
_________________________________________________________________
conv2d_71 (Conv2D)           (None, 16, 16, 100)       240100
_________________________________________________________________
max_pooling2d_69 (MaxPooling2D) (None, 8, 8, 100)      0
_________________________________________________________________
conv2d_72 (Conv2D)           (None, 8, 8, 100)         40100
_________________________________________________________________
max_pooling2d_70 (MaxPooling2D) (None, 4, 4, 100)      0
_________________________________________________________________
conv2d_73 (Conv2D)           (None, 4, 4, 100)         40100
_________________________________________________________________
max_pooling2d_71 (MaxPooling2D) (None, 2, 2, 100)      0
_________________________________________________________________
dropout_23 (Dropout)         (None, 2, 2, 100)         0
_________________________________________________________________
flatten_23 (Flatten)         (None, 400)               0
_________________________________________________________________
dense_69 (Dense)             (None, 128)               51328
_________________________________________________________________
dense_70 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_71 (Dense)             (None, 43)                5547
=================================================================
Total params: 401,037
Trainable params: 401,037
Non-trainable params: 0
```

Figure 3: The structure of CNN1.

With the loss function and the accuracy matrice printed on the result, I can have the following figure.

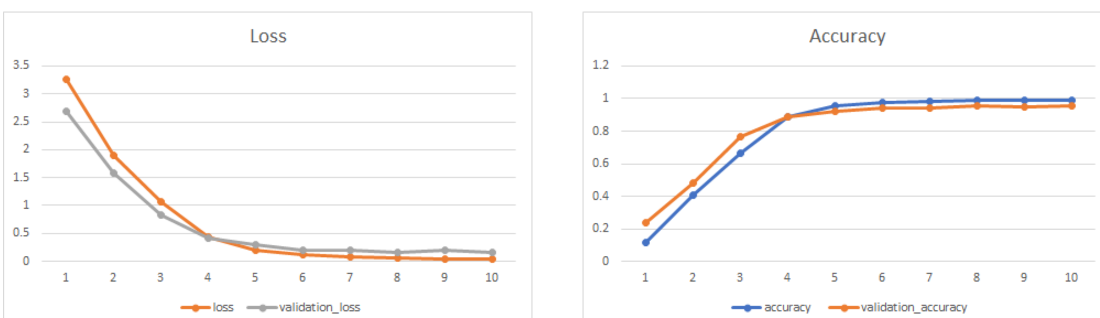

Figure 4: The loss, val_loss and acc, val_acc for model 1 at each epoch.

For the final epoch, I have

```
Epoch 10/10
70/70 [==============================] - 112s 2s/step - loss: 0.0486 -
↪ accuracy: 0.9886 - val_loss: 0.1658 - val_accuracy: 0.9537
```

Then, I tested the model with the given test set. The result is given by,

```
395/395 [==============================] - 12s 31ms/step - loss: 0.2297 -
↪  accuracy: 0.9413
```

So far, the model is quite satisfying.

2.2.2 Testing

For the testing, I have the following pictures which were found on the internet and my living environment. These are some samples.

Figure 5: Sample testing data of traffic signs. From left to right, category 0, 0, 25, 2.

These figures had been cut in advance by hand. However, they still need some modification before testing. Since the channel number of some of these figures is 4 instead of 3 and also the size of the them are not (32,32). So I implemented the following function to modify and prediction.

```python
def imgPr(name):
    im=Image.open(name)
    if len(im.split())==4:
        r, g, b, a = im.split()
        I = Image.merge("RGB", (r, g, b))
        I.save(name)
    im=tf.io.gfile.GFile(name,'rb').read()
    I=tf.image.decode_jpeg(im)
    I=tf.image.convert_image_dtype(I, dtype = tf.float64)
    I=tf.image.resize(I, (32,32))
    plt.imshow(I)
    I=np.expand_dims(I,axis=0)
    print(I.shape)
    print(model1.predict(I))
    print(model1.predict(I).argmax())
```

The results of this testing is relatively unpleasing, only one has been correctly categorized.

filename	t0	t1	t02	t2	t17	t22	t25	t32	t36	t252
category	0	1	0	2	17	2	25	32	36	25
prediction	20	32	32	11	38	32	2	32	13	39

Table 1: Testing results for model 1.

2.3 CNN2

Since the performance of CNN1 is really bad on my own testing data. I want to improve the performance of the CNN.

2.3.1 Data pre-process

By looking at the picture data, I found the training data was generated through data enhancement. However, some data generated is not good enough for example

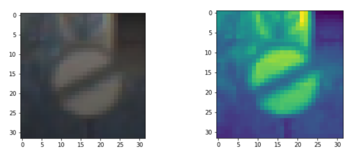

Figure 6: Left: Original figure; Right: Changed into 1 channel.

There is also another intuition to do this. Since the color may not be quite important, it is economic and reasonable to change the picture into 1 channel.

2.3.2 Model

Now, I designed and trained the model as following,

```
model2=keras.Sequential()
model2.add(tf.keras.layers.Conv2D(filters=150,kernel_size=5,
                activation='tanh',input_shape=(32,32,1)))
model2.add(tf.keras.layers.MaxPool2D(pool_size=(2,2)))
model2.add(tf.keras.layers.Conv2D(filters=100,kernel_size=2,
                activation='relu',padding='SAME'))
model2.add(tf.keras.layers.MaxPool2D(pool_size=(2,2)))
model2.add(tf.keras.layers.Conv2D(filters=100,kernel_size=2,
                activation='sigmoid',padding='SAME'))
model2.add(tf.keras.layers.MaxPool2D(pool_size=(2,2)))
model2.add(tf.keras.layers.Dropout(rate=0.25))
model2.add(tf.keras.layers.Flatten())
model2.add(tf.keras.layers.Dense(128,activation='relu'))
model2.add(tf.keras.layers.Dense(128,activation='tanh'))
```

```python
model2.add(tf.keras.layers.Dense(43,activation='softmax'))
model2.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
                    metrics=['accuracy'])
model2.fit(x_train2,y_train,batch_size=200,epochs=20,validation_data=
                    (x_valid2,y_valid),shuffle=True,validation_freq=1)
```

Similarly, I can plot the loss and accuracy below. For the final epoch, I have

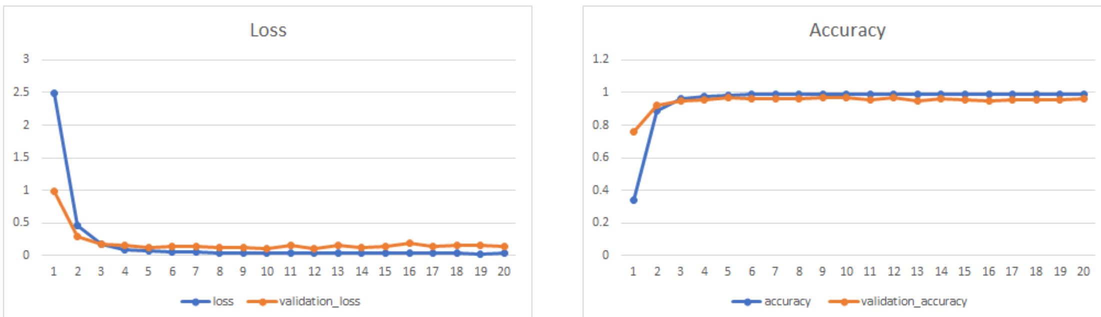

Figure 7: The loss, val_loss and acc, val_acc for model 2 at each epoch.

```
Epoch 20/20
174/174 [==============================] - 54s 308ms/step - loss: 0.0288 -
    accuracy: 0.9914 - val_loss: 0.1354 - val_accuracy: 0.9639
```

Then, I tested the model with the given test set. The result is given by,

```
395/395 [==============================] - 7s 18ms/step - loss: 0.1800 -
    accuracy: 0.9505
```

Also the structure of the model is given by,

```
Model: "sequential_35"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_104 (Conv2D)          (None, 28, 28, 150)       3900
_________________________________________________________________
max_pooling2d_102 (MaxPooling2D) (None, 14, 14, 150)   0
_________________________________________________________________
conv2d_105 (Conv2D)          (None, 14, 14, 100)       60100
_________________________________________________________________
max_pooling2d_103 (MaxPooling2D) (None, 7, 7, 100)     0
_________________________________________________________________
conv2d_106 (Conv2D)          (None, 7, 7, 100)         40100
_________________________________________________________________
max_pooling2d_104 (MaxPooling2D) (None, 3, 3, 100)     0
_________________________________________________________________
dropout_35 (Dropout)         (None, 3, 3, 100)         0
_________________________________________________________________
flatten_35 (Flatten)         (None, 900)               0
_________________________________________________________________
dense_105 (Dense)            (None, 128)               115328
_________________________________________________________________
dense_106 (Dense)            (None, 128)               16512
_________________________________________________________________
dense_107 (Dense)            (None, 43)                5547
=================================================================
Total params: 241,487
Trainable params: 241,487
Non-trainable params: 0
```

Figure 8: The structure of CNN2.

Comparing with CNN1, CNN2 is improved.

2.3.3 Testing

Now let's do testing on the my test data. The function is given by,

```python
def gimgPr(name):
    im=Image.open(name)
    I = im.convert('L')
    I.save("g"+name)
    im=tf.io.gfile.GFile("g"+name,'rb').read()
    I=tf.image.decode_jpeg(im)
    I=tf.image.convert_image_dtype(I, dtype = tf.float64)
    I=tf.image.resize(I, (32,32))
    #plt.imshow(I)
    I=np.expand_dims(I,axis=0)
    print(I.shape)
    print(model2.predict(I))
    print(model2.predict(I).argmax())
```

After testing 10 different pictures, I have *t32.jpg* and *t17.jpg* being correctly categorized.

Note that the sign for speed limit signs are wrongly categorized because of the ambiguity of the number on the plate, which may because of the quality of the picture.

filename	t0	t1	t02	t2	t17	t22	t25	t32	t36	t252
category	0	1	0	2	17	2	25	32	36	25
prediction	8	8	8	3	17	3	3	32	13	3

Table 2: Test results for CNN2.