Project Part 3: Classification Using Neural Networks and Deep Learning (SVHN Dataset)
Name:Wenzhe Zheng
Date:4/19/2021

Convolutional Neural Network (ConvNet / CNN) is a deep learning algorithm that can input images, assign importance information to each object in the image, and assign its importance level, and finally can distinguish them from each other. In this project, I used the following steps to complete the CNN. Finally achieved good results.

1) We import the needed libraries

```
import numpy as np
import keras
from matplotlib import pyplot as plt
from scipy.io import loadmat
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from keras.preprocessing.image import ImageDataGenerator
%matplotlib inline
np.random.seed(20)
```

2) Load the data

```
# Step (1): Load the data

train_raw = loadmat('/content/drive/MyDrive/Study-Gate/CNN/train_32x32 (1).mat')
test_raw = loadmat('/content/drive/MyDrive/Study-Gate/CNN/test_32x32 (2).mat')
```

3) Get the images and labels from the data

```
# from data get images and thier labels

train_images = np.array(train_raw['X'])
test_images = np.array(test_raw['X'])
train_labels = train_raw['y']
test_labels = test_raw['y']
```

4) Data Preprocessing

```
# do some data preprocessing to prepare them for the model

# imageshas DataType uint8 not in list of allowed values so we need to convert the images into float
train_images = train_images.astype('float64')
test_images = test_images.astype('float64')

#The labels must be of type int so Convert the labels into integers
train_labels = train_labels.astype('int64')
test_labels = test_labels.astype('int64')
```

5) Data Normalization

```
# Step (2_1): Normalize the images data to bring it in the range of 0-1.

print('Image Data before Normalization: Min: {}, Max: {}'.format(train_images.min(), train_images.max()))
train_images /= 255.0
test_images /= 255.0
print('Image Data after Normalization: Min: {}, Max: {}'.format(train_images.min(), train_images.max()))
```

```
Image Data before Normalization: Min: 0.0, Max: 255.0
Image Data after Normalization: Min: 0.0, Max: 1.0
```

6) One-hot Encoding

```
# Step (2_2): Encode the labels using one-hot vector encoding.

one_hot_encoding = LabelBinarizer()
train_labels = one_hot_encoding.fit_transform(train_labels)
test_labels = one_hot_encoding.fit_transform(test_labels)
```

7) Train_Test_Split

```
# Split train data into train and validation sets

X_train, X_val, y_train, y_val = train_test_split(train_images, train_labels,
                                                  test_size=0.15, random_state=22)
```

8) Data Augmentation

```
# Data augmentation to enhance the size and quality of the training data

datagen = ImageDataGenerator(rotation_range=8,
                             zoom_range=[0.95, 1.05],
                             height_shift_range=0.10,
                             shear_range=0.15)
```

9) Build the Model
Note that: We use batch normalization, which can speed up the training process of neural networks and improve the performance of the model. Then, batch normalization is a separate layer, which allows each layer of the network to learn independently.

```
# Step (3): Define the architecture of the model

keras.backend.clear_session()

model = keras.Sequential([
    keras.layers.Conv2D(64, (5, 5), padding='same',
                           activation='relu',
                           input_shape=(32, 32, 3)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(64, (5, 5), padding='same',
                           activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(128, (5, 5), padding='same',
                           activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Flatten(),
    keras.layers.Dense(3072, activation='relu'),
    keras.layers.Dense(2048, activation='relu'),
    keras.layers.Dense(10,  activation='softmax')
])

optimizer = keras.optimizers.SGD(lr=1e-1)
model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

10) Model summary

```
# The Model architecture details

model.summary()
```

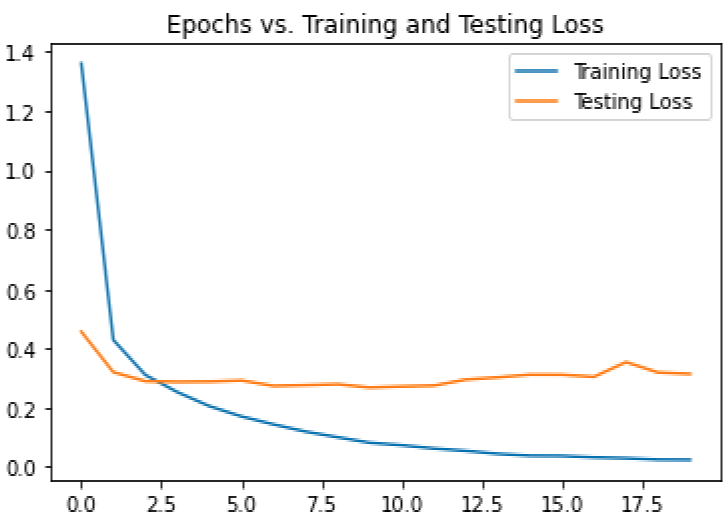
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 32, 32, 64)        4864      
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 64)        256       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 64)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 64)        102464    
_________________________________________________________________
batch_normalization_1 (Batch (None, 16, 16, 64)        256       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 128)         204928    
_________________________________________________________________
batch_normalization_2 (Batch (None, 8, 8, 128)         512       
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0         
_________________________________________________________________
dense (Dense)                (None, 3072)              25168896  
_________________________________________________________________
dense_1 (Dense)              (None, 2048)              6293504   
_________________________________________________________________
dense_2 (Dense)              (None, 10)                20490     
=================================================================
Total params: 31,796,170
Trainable params: 31,795,658
Non-trainable params: 512
_________________________________________________________________
```

11) Train Model using SGD optimizer

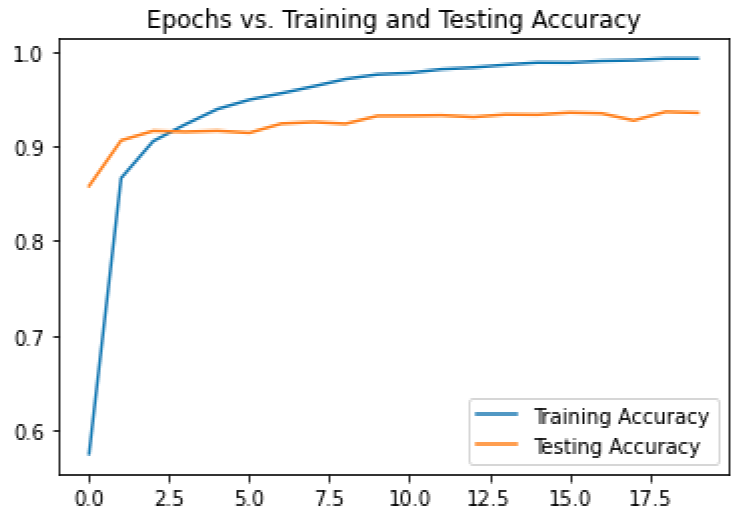
```
# Step (4): Train the model using SGD optimizer

optimizer = keras.optimizers.SGD(lr=1e-1)
model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit_generator(datagen.flow(X_train, y_train, batch_size=128),
                              epochs=20, validation_data=(X_val, y_val))
```

```
Epoch 1/20
487/487 [==============================] - 66s 68ms/step - loss: 2.4147 - accuracy: 0.3542 - val_loss: 0.4571 - val_accuracy: 0.8580
Epoch 2/20
487/487 [==============================] - 33s 68ms/step - loss: 0.4511 - accuracy: 0.8597 - val_loss: 0.3204 - val_accuracy: 0.9062
Epoch 3/20
487/487 [==============================] - 33s 67ms/step - loss: 0.3116 - accuracy: 0.9050 - val_loss: 0.2896 - val_accuracy: 0.9163
Epoch 4/20
487/487 [==============================] - 33s 67ms/step - loss: 0.2576 - accuracy: 0.9227 - val_loss: 0.2875 - val_accuracy: 0.9153
Epoch 5/20
487/487 [==============================] - 33s 67ms/step - loss: 0.2003 - accuracy: 0.9412 - val_loss: 0.2883 - val_accuracy: 0.9164
Epoch 6/20
487/487 [==============================] - 33s 67ms/step - loss: 0.1679 - accuracy: 0.9505 - val_loss: 0.2923 - val_accuracy: 0.9142
Epoch 7/20
487/487 [==============================] - 32s 67ms/step - loss: 0.1367 - accuracy: 0.9574 - val_loss: 0.2738 - val_accuracy: 0.9239
Epoch 8/20
487/487 [==============================] - 33s 67ms/step - loss: 0.1112 - accuracy: 0.9656 - val_loss: 0.2763 - val_accuracy: 0.9257
Epoch 9/20
487/487 [==============================] - 33s 67ms/step - loss: 0.0942 - accuracy: 0.9724 - val_loss: 0.2796 - val_accuracy: 0.9237
Epoch 10/20
487/487 [==============================] - 33s 67ms/step - loss: 0.0776 - accuracy: 0.9780 - val_loss: 0.2683 - val_accuracy: 0.9321
Epoch 11/20
487/487 [==============================] - 33s 67ms/step - loss: 0.0687 - accuracy: 0.9793 - val_loss: 0.2721 - val_accuracy: 0.9323
Epoch 12/20
487/487 [==============================] - 32s 67ms/step - loss: 0.0588 - accuracy: 0.9828 - val_loss: 0.2748 - val_accuracy: 0.9328
Epoch 13/20
487/487 [==============================] - 32s 66ms/step - loss: 0.0487 - accuracy: 0.9849 - val_loss: 0.2952 - val_accuracy: 0.9309
Epoch 14/20
487/487 [==============================] - 32s 66ms/step - loss: 0.0424 - accuracy: 0.9868 - val_loss: 0.3027 - val_accuracy: 0.9337
Epoch 15/20
487/487 [==============================] - 33s 67ms/step - loss: 0.0341 - accuracy: 0.9896 - val_loss: 0.3119 - val_accuracy: 0.9334
Epoch 16/20
487/487 [==============================] - 32s 66ms/step - loss: 0.0351 - accuracy: 0.9888 - val_loss: 0.3116 - val_accuracy: 0.9357
Epoch 17/20
487/487 [==============================] - 32s 66ms/step - loss: 0.0299 - accuracy: 0.9908 - val_loss: 0.3046 - val_accuracy: 0.9348
Epoch 18/20
487/487 [==============================] - 32s 66ms/step - loss: 0.0273 - accuracy: 0.9918 - val_loss: 0.3543 - val_accuracy: 0.9273
Epoch 19/20
487/487 [==============================] - 32s 66ms/step - loss: 0.0242 - accuracy: 0.9933 - val_loss: 0.3192 - val_accuracy: 0.9365
Epoch 20/20
487/487 [==============================] - 32s 66ms/step - loss: 0.0219 - accuracy: 0.9937 - val_loss: 0.3143 - val_accuracy: 0.9354
```

12) Training and Testing(Validation) Loss vs. Number of Epochs



13) Training and Testing(Validation) Accuracy vs. Number of Epochs



14) Final Classification error on the testing set

```
# Test model on the test set
# the final classification accuracy on testing set.

test_loss, test_acc = model.evaluate(x=test_images, y=test_labels, verbose=0)

print('Test accuracy is ', test_acc)
```

```
Test accuracy is  0.9304317831993103
```

Finally , we got accuracy is 93%.I think that result is good.