Six_Classify

August 30, 2020

[1]:
```
# Load the TensorBoard notebook extension
%load_ext tensorboard
```

[2]:
```
import tensorflow as tf
from keras import layers
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
import keras.backend as K
K.set_image_data_format('channels_last')

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
import datetime
from alexnet import AlexNet
```

[3]:
```
# Set up the GPU in the condition of allocation exceeds system memory with the␣
 ↪reminding message: Could not
# create cuDNN handle... The following lines of code can avoids the sudden stop␣
 ↪of the runtime.
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
```

[4]:
```
# Give the global constants.Please notify BATCH_SIZE for model.fit() and␣
 ↪Batch_Size for
# model.evaluate() and model.predict()
EPOCHS = 64
BATCH_SIZE = 32
Batch_Size = 1
image_height = 227
image_width = 227
channels = 3
num_classes = 6
```

[5]:
```
# It calls the alexnet model in alexnet.py. It is equivalent to the following␣
 ↪function.
```

```python
# model = AlexNet(train[0][0].shape[1:])
model = AlexNet((image_width,image_height,channels), num_classes)
```

[6]:
```python
# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

[7]:
```python
# It will output the AlexNet model after executing the command
model.summary()
```

```
Model: "alex_net"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 55, 55, 96)        34944
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 27, 27, 96)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 27, 27, 256)       614656
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 256)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 13, 384)       885120
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 13, 384)       1327488
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 256)       884992
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0
_________________________________________________________________
dense (Dense)                (None, 4096)              37752832
_________________________________________________________________
dropout (Dropout)            (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 6006
=================================================================
Total params: 62,384,350
Trainable params: 62,384,350
```

```
Non-trainable params: 0
_________________________________________________________________
```

[8]:
```
train_dir = '/home/mic/Documents/Six_Classification_AlexNet/seg_train/seg_train'
test_dir = '/home/mic/Documents/Six_Classification_AlexNet/seg_test/seg_test'
predict_dir = '/home/mic/Documents/Six_Classification_AlexNet/seg_pred/'
```

[9]:
```
# keras.preprocessing.image.ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1.0/255)

# keras.preprocessing.image.DirectoryIterator
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size=(227,227),
                                                    class_mode='categorical')

train_num = train_generator.samples
```

```
Found 14034 images belonging to 6 classes.
```

[10]:
```
# Start Tensorboard --logdir logs/fit
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir)
callback_list = [tensorboard_callback]
```

[11]:
```
# Set verbose=1 (or verbose=0) for visibale (or invisible) training procedure.
model.fit(train_generator,
          epochs=EPOCHS,
          steps_per_epoch=train_num//BATCH_SIZE,
          callbacks=callback_list,
          verbose=1)
```

```
Epoch 1/64
438/438 [==============================] - 19s 43ms/step - loss: 1.2548 -
accuracy: 0.5315
Epoch 2/64
438/438 [==============================] - 19s 43ms/step - loss: 0.9041 -
accuracy: 0.6513
Epoch 3/64
438/438 [==============================] - 19s 43ms/step - loss: 0.8070 -
accuracy: 0.6973
Epoch 4/64
438/438 [==============================] - 19s 44ms/step - loss: 0.7215 -
accuracy: 0.7360
Epoch 5/64
438/438 [==============================] - 19s 44ms/step - loss: 0.6502 -
accuracy: 0.7682
Epoch 6/64
438/438 [==============================] - 19s 43ms/step - loss: 0.6381 -
```

```
accuracy: 0.7762
Epoch 7/64
438/438 [==============================] - 19s 44ms/step - loss: 0.5727 -
accuracy: 0.8019
Epoch 8/64
438/438 [==============================] - 19s 44ms/step - loss: 0.5331 -
accuracy: 0.8142
Epoch 9/64
438/438 [==============================] - 19s 44ms/step - loss: 0.5012 -
accuracy: 0.8260
Epoch 10/64
438/438 [==============================] - 19s 44ms/step - loss: 0.5093 -
accuracy: 0.8260
Epoch 11/64
438/438 [==============================] - 19s 42ms/step - loss: 0.4858 -
accuracy: 0.8315
Epoch 12/64
438/438 [==============================] - 19s 43ms/step - loss: 0.4448 -
accuracy: 0.8503
Epoch 13/64
438/438 [==============================] - 20s 45ms/step - loss: 0.4275 -
accuracy: 0.8525
Epoch 14/64
438/438 [==============================] - 20s 46ms/step - loss: 0.4205 -
accuracy: 0.8569
Epoch 15/64
438/438 [==============================] - 19s 44ms/step - loss: 0.4006 -
accuracy: 0.8637
Epoch 16/64
438/438 [==============================] - 19s 44ms/step - loss: 0.3759 -
accuracy: 0.8758
Epoch 17/64
438/438 [==============================] - 19s 44ms/step - loss: 0.3724 -
accuracy: 0.8749
Epoch 18/64
438/438 [==============================] - 19s 44ms/step - loss: 0.3724 -
accuracy: 0.8798
Epoch 19/64
438/438 [==============================] - 19s 44ms/step - loss: 0.3181 -
accuracy: 0.8949
Epoch 20/64
438/438 [==============================] - 19s 43ms/step - loss: 0.3144 -
accuracy: 0.8969
Epoch 21/64
438/438 [==============================] - 19s 43ms/step - loss: 0.3071 -
accuracy: 0.8979
Epoch 22/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2923 -
```

```
accuracy: 0.9062
Epoch 23/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2903 -
accuracy: 0.9077
Epoch 24/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2703 -
accuracy: 0.9150
Epoch 25/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2716 -
accuracy: 0.9154
Epoch 26/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2789 -
accuracy: 0.9106
Epoch 27/64
438/438 [==============================] - 18s 42ms/step - loss: 0.2672 -
accuracy: 0.9177
Epoch 28/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2227 -
accuracy: 0.9312
Epoch 29/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2542 -
accuracy: 0.9199
Epoch 30/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2474 -
accuracy: 0.9259
Epoch 31/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2580 -
accuracy: 0.9224
Epoch 32/64
438/438 [==============================] - 19s 43ms/step - loss: 0.2048 -
accuracy: 0.9399
Epoch 33/64
438/438 [==============================] - 19s 43ms/step - loss: 0.2322 -
accuracy: 0.9322
Epoch 34/64
438/438 [==============================] - 19s 43ms/step - loss: 0.2532 -
accuracy: 0.9280
Epoch 35/64
438/438 [==============================] - 19s 43ms/step - loss: 0.2251 -
accuracy: 0.9351
Epoch 36/64
438/438 [==============================] - 19s 44ms/step - loss: 0.1975 -
accuracy: 0.9442
Epoch 37/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2347 -
accuracy: 0.9318
Epoch 38/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2106 -
```

```
accuracy: 0.9397
Epoch 39/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2464 -
accuracy: 0.9334
Epoch 40/64
438/438 [==============================] - 20s 45ms/step - loss: 0.2071 -
accuracy: 0.9399
Epoch 41/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2317 -
accuracy: 0.9356
Epoch 42/64
438/438 [==============================] - 19s 43ms/step - loss: 0.2184 -
accuracy: 0.9397
Epoch 43/64
438/438 [==============================] - 19s 43ms/step - loss: 0.2757 -
accuracy: 0.9247
Epoch 44/64
438/438 [==============================] - 19s 43ms/step - loss: 0.1963 -
accuracy: 0.9453
Epoch 45/64
438/438 [==============================] - 19s 43ms/step - loss: 0.1813 -
accuracy: 0.9528
Epoch 46/64
438/438 [==============================] - 19s 42ms/step - loss: 0.2237 -
accuracy: 0.9401
Epoch 47/64
438/438 [==============================] - 19s 43ms/step - loss: 0.2508 -
accuracy: 0.9324
Epoch 48/64
438/438 [==============================] - 19s 43ms/step - loss: 0.1941 -
accuracy: 0.9499
Epoch 49/64
438/438 [==============================] - 19s 43ms/step - loss: 0.1858 -
accuracy: 0.9506
Epoch 50/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2127 -
accuracy: 0.9437
Epoch 51/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2931 -
accuracy: 0.9252
Epoch 52/64
438/438 [==============================] - 19s 43ms/step - loss: 0.2108 -
accuracy: 0.9462
Epoch 53/64
438/438 [==============================] - 19s 43ms/step - loss: 0.2378 -
accuracy: 0.9390
Epoch 54/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2141 -
```

```
accuracy: 0.9432
Epoch 55/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2250 -
accuracy: 0.9401
Epoch 56/64
438/438 [==============================] - 19s 43ms/step - loss: 0.2050 -
accuracy: 0.9481
Epoch 57/64
438/438 [==============================] - 19s 42ms/step - loss: 0.1747 -
accuracy: 0.9536
Epoch 58/64
438/438 [==============================] - 19s 43ms/step - loss: 0.2707 -
accuracy: 0.9328
Epoch 59/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2190 -
accuracy: 0.9447
Epoch 60/64
438/438 [==============================] - 19s 44ms/step - loss: 0.1817 -
accuracy: 0.9497
Epoch 61/64
438/438 [==============================] - 18s 42ms/step - loss: 0.2455 -
accuracy: 0.9414
Epoch 62/64
438/438 [==============================] - 19s 42ms/step - loss: 0.1908 -
accuracy: 0.9539
Epoch 63/64
438/438 [==============================] - 19s 44ms/step - loss: 0.2951 -
accuracy: 0.9285
Epoch 64/64
438/438 [==============================] - 19s 42ms/step - loss: 0.2317 -
accuracy: 0.9402
```

[11]:
```
<tensorflow.python.keras.callbacks.History at 0x7f8fcd7f5510>
```

[12]:
```
%tensorboard --logdir logs/fit
```

```
Reusing TensorBoard on port 6006 (pid 6318), started 0:43:01 ago. (Use '!kill 6318' to kill it


<IPython.core.display.HTML object>
```

[13]:
```
# It is the test generator as similar as the above.
test_datagen = ImageDataGenerator(rescale=1.0/255)

test_generator = test_datagen.flow_from_directory(test_dir,
                                                                    ␣
 ↪target_size=(image_height,image_width),
                                                class_mode='categorical')

test_num = test_generator.samples
```

```
Found 3000 images belonging to 6 classes.
```

[14]:
```
# Evalute the trained model and return both the loss and the test accuracy.
test_num = test_generator.samples
preds = model.evaluate(test_generator,
                       verbose=1,
                       batch_size=Batch_Size,
                       steps=test_num//Batch_Size)

print("Loss = " + str(preds[0]))
print("Test Accuracy = " + str(preds[1]))
```

```
3000/3000 [==============================] - 131s 44ms/step - loss: 1.0992 -
accuracy: 0.8041
Loss = 1.0991582870483398
Test Accuracy = 0.8040562868118286
```

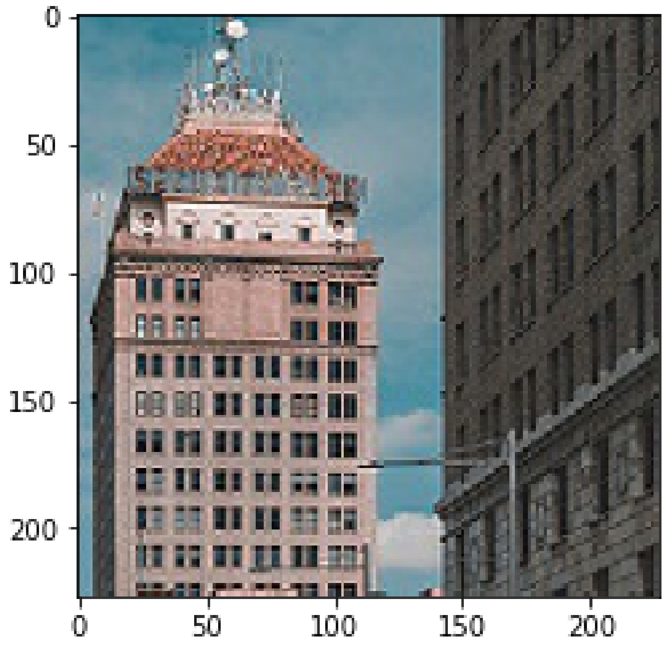
[15]:
```
# Give the implicit steps=7301 for selecting the specific image number.
predict_datagen = ImageDataGenerator(rescale=1.0/255)

predict_generator = predict_datagen.flow_from_directory(predict_dir,
                                                                  ␣
 ↪target_size=(image_height,image_width),
                                                                   batch_size=1,
                                                                  ␣
 ↪class_mode='categorical')

predict_num = predict_generator.samples
```

```
Found 7301 images belonging to 1 classes.
```

[16]:
```
# Make the prediction for any one of the predicted images
predictions = model.predict(predict_generator,
                            verbose=1,
                            batch_size=Batch_Size,
                            steps=predict_num//Batch_Size)
```

```
7301/7301 [==============================] - 24s 3ms/step
```

[17]:
```
# Plot the discriptive diagram
imshow(predict_generator[5800][0][0])
plt.imsave("predicted1.png",predict_generator[5800][0][0])
```



```
[18]: predictions[5800]
```

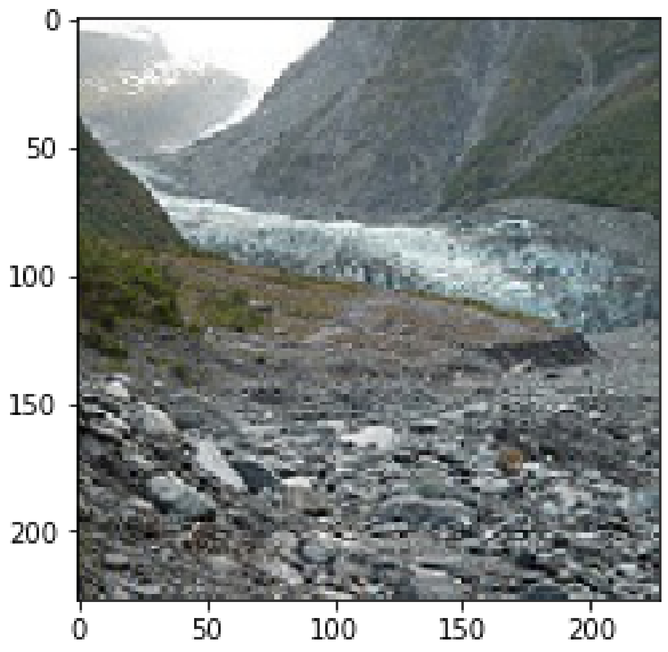
```
[18]: array([3.9492771e-02, 3.6630628e-03, 4.2655575e-03, 6.4428593e-04,
              4.2253063e-04, 9.5151180e-01], dtype=float32)
```

```
[19]: print(predictions[5800])
```

```
[3.9492771e-02 3.6630628e-03 4.2655575e-03 6.4428593e-04 4.2253063e-04
 9.5151180e-01]
```

```
[20]: imshow(predict_generator[4800][0][0])
```

```
[20]: <matplotlib.image.AxesImage at 0x7f8f3c2bb1d0>
```



```
[21]: predictions[4800]
```

```
[21]: array([1.3618610e-17, 1.0000000e+00, 9.8087005e-34, 1.1027436e-30,
             9.5559834e-38, 8.2592264e-17], dtype=float32)
```

```
[22]: import os

      def get_category(predicted_output):
          return os.listdir(train_dir)[np.argmax(predicted_output)]
```

```
[23]: print(get_category(predictions[512]))
```

```
buildings
```

```
[24]: fig, axs = plt.subplots(2, 3, figsize=(10,10))

      axs[0][0].imshow(predict_generator[1002][0][0])
      axs[0][0].set_title(get_category(predictions[1002]))

      axs[0][1].imshow(predict_generator[22][0][0])
      axs[0][1].set_title(get_category(predictions[22]))

      axs[0][2].imshow(predict_generator[1300][0][0])
      axs[0][2].set_title(get_category(predictions[1300]))
```

```
axs[1][0].imshow(predict_generator[3300][0][0])
axs[1][0].set_title(get_category(predictions[3300]))

axs[1][1].imshow(predict_generator[7002][0][0])
axs[1][1].set_title(get_category(predictions[7002]))

axs[1][2].imshow(predict_generator[512][0][0])
axs[1][2].set_title(get_category(predictions[512]))
```

```
[24]: Text(0.5, 1.0, 'buildings')
```