Image Recognition by CNN and VGG (Intel Image Recognition Competition)

by Johnson Wei

In [ ]:
```
import os
import glob
import json
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import  applications
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D,GlobalAveragePooling2D, Fl
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLRO
from tensorflow.keras.optimizers import Adam
```

In [ ]:
```
import warnings
warnings.filterwarnings('ignore')
import random
import logging
logger = tf.get_logger()
logger.setLevel(logging.ERROR)
import glob
import shutil
import os
import itertools
from sklearn.metrics import confusion_matrix ,classification_report
from keras.layers import Input,Flatten
from keras.models import Model
from random import randint
```

In [ ]:
```
# import the dataset
train_path = '/Users/johnsonwei/Dropbox/Mac/Desktop/NLP+ML/Timeseries/archive/se
test_path = '/Users/johnsonwei/Dropbox/Mac/Desktop/NLP+ML/Timeseries/archive/seg
```

In [ ]:
```
# Image rescaling and read the data
IMG_SIZE = (150,150)
train_images=ImageDataGenerator(rescale=1.0/255).flow_from_directory(directory=t
```

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

In [ ]:
```
IMG_SIZE = (150,150)
test_images=ImageDataGenerator(rescale=1.0/255).flow_from_directory(directory=te
```

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

In [ ]:
```
# construct the CNN model, specify the layers
model1 = Sequential([
```

```
    Conv2D(filters=32,kernel_size=(3,3),activation='relu',padding='same',input_s

    MaxPool2D(pool_size=(2,2),strides=2),

    Conv2D(filters=32,kernel_size=(3,3),activation='relu',padding='same'),
     Dropout(0.5),

    MaxPool2D(pool_size=(2,2),strides=2),

    Flatten(),
    Dense(units=6,activation='softmax'),
])
```

In [ ]:
```
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 150, 150, 32)      896

 max_pooling2d (MaxPooling2D  (None, 75, 75, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 75, 75, 32)        9248

 dropout (Dropout)           (None, 75, 75, 32)        0

 max_pooling2d_1 (MaxPooling  (None, 37, 37, 32)       0
 2D)

 flatten (Flatten)           (None, 43808)             0

 dense (Dense)               (None, 6)                 262854

=================================================================
Total params: 272,998
Trainable params: 272,998
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
# Using Adam optimizer with learning rate of 0.0001
model1.compile(optimizer=Adam(learning_rate=0.0001),loss='categorical_crossentro
```

In [ ]:
```
results=model1.fit(train_images,epochs=10)
```

```
Epoch 1/10
1404/1404 [==============================] - 138s 98ms/step - loss: 1.1727 - acc
uracy: 0.5507
Epoch 2/10
1404/1404 [==============================] - 144s 102ms/step - loss: 0.9000 - ac
curacy: 0.6606
Epoch 3/10
1404/1404 [==============================] - 149s 106ms/step - loss: 0.7770 - ac
curacy: 0.7175
Epoch 4/10
1404/1404 [==============================] - 138s 98ms/step - loss: 0.6998 - acc
uracy: 0.7488
```

```
Epoch 5/10
1404/1404 [==============================] - 165s 117ms/step - loss: 0.6421 - ac
curacy: 0.7717
Epoch 6/10
1404/1404 [==============================] - 154s 110ms/step - loss: 0.5818 - ac
curacy: 0.7950
Epoch 7/10
1404/1404 [==============================] - 137s 98ms/step - loss: 0.5330 - acc
uracy: 0.8132
Epoch 8/10
1404/1404 [==============================] - 151s 107ms/step - loss: 0.5002 - ac
curacy: 0.8254
Epoch 9/10
1404/1404 [==============================] - 168s 120ms/step - loss: 0.4550 - ac
curacy: 0.8438
Epoch 10/10
1404/1404 [==============================] - 165s 118ms/step - loss: 0.4193 - ac
curacy: 0.8573
```

In [ ]:
```
model1.evaluate(test_images)
```

```
300/300 [==============================] - 9s 26ms/step - loss: 0.6839 - accurac
y: 0.7743
```

Out[ ]:
```
[0.6839448809623718, 0.7743333578109741]
```

The overall accuracy has room for improvement

In [ ]:
```
train_images_p = ImageDataGenerator(preprocessing_function=tf.keras.applications
```

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

In [ ]:
```
test_images_p=ImageDataGenerator(preprocessing_function=tf.keras.applications.vg
```

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

In [ ]:
```
# Construct VGG16 model to optmize the result
from tensorflow.keras.applications import VGG16

pretrained_model=VGG16(input_shape = (150, 150, 3),
                       include_top = False,
                       weights = 'imagenet')

for layer in pretrained_model.layers:
     layer.trainable = False


last_layer = pretrained_model.get_layer('block5_pool')
print('last layer of vgg : output shape: ', last_layer.output_shape)
last_output= last_layer.output

x = Flatten()(last_output)
x = Dense(64, activation='relu')(x)
x = Dropout(0.2)(x)
x = Dense(6, activation='softmax')(x)

model_vgg = Model(pretrained_model.input, x)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
ns/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
```

```
58889256/58889256 [==============================] - 3s 0us/step
last layer of vgg : output shape:  (None, 4, 4, 512)
```

In [ ]:
```
model_vgg.compile(optimizer=Adam(learning_rate=0.0001),loss='categorical_crossen
```

In [ ]:
```
results=model_vgg.fit(train_images_p,epochs=10)
```

```
Epoch 1/10
1404/1404 [==============================] - 913s 649ms/step - loss: 1.0198 - ac
curacy: 0.8035
Epoch 2/10
1404/1404 [==============================] - 857s 610ms/step - loss: 0.2956 - ac
curacy: 0.8997
Epoch 3/10
1404/1404 [==============================] - 797s 568ms/step - loss: 0.1972 - ac
curacy: 0.9280
Epoch 4/10
1404/1404 [==============================] - 857s 610ms/step - loss: 0.1511 - ac
curacy: 0.9465
Epoch 5/10
1404/1404 [==============================] - 886s 631ms/step - loss: 0.1212 - ac
curacy: 0.9543
Epoch 6/10
1404/1404 [==============================] - 931s 663ms/step - loss: 0.0929 - ac
curacy: 0.9643
Epoch 7/10
1404/1404 [==============================] - 1396s 995ms/step - loss: 0.0782 - a
ccuracy: 0.9723
Epoch 8/10
1404/1404 [==============================] - 911s 649ms/step - loss: 0.0683 - ac
curacy: 0.9758
Epoch 9/10
1404/1404 [==============================] - 988s 704ms/step - loss: 0.0580 - ac
curacy: 0.9785
Epoch 10/10
1404/1404 [==============================] - 916s 652ms/step - loss: 0.0519 - ac
curacy: 0.9809
```

In [ ]:
```
 model_vgg.evaluate(test_images_p)
```

```
300/300 [==============================] - 212s 704ms/step - loss: 0.4859 - accu
racy: 0.9110
```

Out[ ]:
```
[0.48593783378601074, 0.9110000133514404]
```

The result has shown better accuracy