Model2_TL - Transfer learning using DenseNet201

by Kaushik Srivatsan - CDS - kaushik.s-25@scds.saiuniversity.edu.in

- Trained on Hutton Rock Dataset
- Model contains BatchNormalization and Dropout of 0.35% Drop Rate before output layer

```
#importing the file
import zipfile
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# zip_ref = zipfile.ZipFile("/content/drive/MyDrive/DL_Project/Hutton_Rock.zip", 'r')
# zip_ref.extractall("/content/drive/MyDrive/DL_Project/")
# zip_ref.close()
```

```
#Importing the libraries
import tensorflow as tf
print(tf.__version__)

from tensorflow import keras
tf.random.set_seed(42)

import numpy as np
np.random.seed(42)

import matplotlib.pyplot as plt
%matplotlib inline

import glob
import PIL
from PIL import Image
```

```
2.15.0
```

```
#Importing the images using glob as ImgFiles
imgFiles = glob.glob("/content/drive/MyDrive/DL_Main/DL_Project/Hutton_Rock/*/*.jpg")
for items in imgFiles[:8]:
  print(items)
```

```
/content/drive/MyDrive/DL_Main/DL_Project/Hutton_Rock/granite/1.jpg
/content/drive/MyDrive/DL_Main/DL_Project/Hutton_Rock/granite/125.jpg
/content/drive/MyDrive/DL_Main/DL_Project/Hutton_Rock/granite/106.jpg
/content/drive/MyDrive/DL_Main/DL_Project/Hutton_Rock/granite/117.jpg
/content/drive/MyDrive/DL_Main/DL_Project/Hutton_Rock/granite/127.jpg
/content/drive/MyDrive/DL_Main/DL_Project/Hutton_Rock/granite/116.jpg
/content/drive/MyDrive/DL_Main/DL_Project/Hutton_Rock/granite/10.jpg
/content/drive/MyDrive/DL_Main/DL_Project/Hutton_Rock/granite/12.jpg
```

Data Preprocessing and Labelling

```
X = []
y = []

for fName in imgFiles:

  # Prepare the dataset and populate X and y
  X_i = Image.open(fName)
  X_i = X_i.resize((299,299))
  X_i = (np.array(X_i).astype(np.float32) /127.5) - 1
        # X.append(X_i)

      # label = fName.split("/")
      # y_i = label[-2]

      # y.append(y_i)

  if X_i.shape == (299, 299, 3):
      X.append(X_i)

      label = fName.split("/")
      y_i = label[-2]

      y.append(y_i)
```

```
print(y)
```

```
['granite', 'granite', 'granite', 'granite', 'granite', 'granite', 'granite', 'granite', 'granite', 'granite', 'granite', 'granite',
```

```
class_counts = dict()

# Count images for each class
for file_path in imgFiles:
    class_name = file_path.split("/")[-2]
    if class_name not in class_counts:
        class_counts[class_name] = 1
    else:
        class_counts[class_name] += 1

for class_name, count in class_counts.items():
    print(f"Class: {class_name}, Count: {count}")
```

```
Class: granite, Count: 187
Class: basalt, Count: 130
Class: andesite, Count: 103
Class: coal, Count: 85
```

```
from sklearn.preprocessing import LabelEncoder
lEncoder = LabelEncoder()
y = lEncoder.fit_transform(y)

print(set(y))
print(lEncoder.classes_)
```

```
{0, 1, 2, 3}
['andesite' 'basalt' 'coal' 'granite']
```

```
X = np.array(X)
y = np.array(y)

print(X.shape)
print(y.shape)
```

```
(498, 299, 299, 3)
(498,)
```

Splitting the Data using sklearn

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
                                                    stratify=y, random_state=42)


print("X_train_shape: {}".format(X_train.shape))
print("X_test_shape: {}".format(X_test.shape))
```

```
X_train_shape: (373, 299, 299, 3)
X_test_shape: (125, 299, 299, 3)
```

Data Normalization

```
mu = X_train.mean()
std = X_train.std()

X_train_std = (X_train-mu)/std
X_test_std = (X_test-mu)/std
```

```
X_train_std.shape
```

```
(373, 299, 299, 3)
```

```
y_train.shape
```

```
(373,)
```

```
print(X_train)
```

```
[[[[ 0.27843142  0.2941177   0.254902  ]
   [ 0.27843142  0.2941177   0.254902  ]
   [ 0.27058828  0.28627455  0.24705887]
   ...
   [ 0.12941182  0.12941182  0.12941182]
   [ 0.04313731  0.04313731  0.04313731]
   [ 0.01176476  0.01176476  0.01176476]]

  [[ 0.27843142  0.2941177   0.254902  ]
   [ 0.27843142  0.2941177   0.254902  ]
   [ 0.27843142  0.2941177   0.254902  ]
   ...
   [ 0.16078436  0.16078436  0.16078436]
   [ 0.082353    0.082353    0.082353  ]
   [ 0.05098045  0.05098045  0.05098045]]

  [[ 0.27843142  0.2941177   0.254902  ]
   [ 0.28627455  0.30196083  0.26274514]
   [ 0.2941177   0.30980396  0.27058828]
   ...
   [ 0.23921573  0.2313726   0.23921573]
   [ 0.17647064  0.17647064  0.17647064]
   [ 0.15294123  0.15294123  0.15294123]]

  ...

  [[ 0.07450986  0.02745104  0.02745104]
   [ 0.082353    0.03529418  0.03529418]
   [ 0.09019613  0.04313731  0.04313731]
   ...
   [-0.04313725 -0.08235294 -0.05098039]
   [-0.09019607 -0.12941176 -0.09803921]
   [-0.10588235 -0.14509803 -0.11372548]]

  [[ 0.082353    0.03529418  0.03529418]
   [ 0.09019613  0.04313731  0.04313731]
   [ 0.12156868  0.07450986  0.07450986]
   ...
   [-0.05882353 -0.09803921 -0.06666666]
   [-0.10588235 -0.14509803 -0.11372548]
   [-0.12156862 -0.1607843  -0.12941176]]

  [[ 0.082353    0.03529418  0.03529418]
   [ 0.09803927  0.05098045  0.05098045]
   [ 0.13725495  0.09019613  0.09019613]
   ...
   [-0.06666666 -0.10588235 -0.0745098 ]
   [-0.11372548 -0.15294117 -0.12156862]
   [-0.12941176 -0.16862744 -0.1372549 ]]]


 [[[-0.09019607 -0.02745098 -0.02745098]
   [ 0.11372554  0.18431377  0.21568632]
```

```
[ 0.06666672  0.12156868  0.18431377]
...
[-0.1372549  -0.1372549  -0.0745098 ]
[-0.05882353 -0.0745098  -0.0745098 ]
[-0.11372548 -0.1607843  -0.17647058]]
```

Transfer Learning - Using DenseNet201

```
base_desnet = keras.applications.DenseNet201(weights='imagenet', input_shape = (299,299,3), include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet201_weights_tf_dim_ordering_tf_k
74836368/74836368 [==============================] - 1s 0us/step
```

```
base_desnet.trainable = False
```

```
for layer in base_desnet.layers:
  layer.trainabe = False
```

Building the Classifier

```
global_pool = keras.layers.GlobalAveragePooling2D()(base_desnet.output)
x = keras.layers.BatchNormalization()(global_pool)
x = keras.layers.Dropout(0.35)(x)

output_ = keras.layers.Dense(units=4, activation='softmax')(x)
model_TL2 = keras.models.Model(inputs=[base_desnet.input], outputs=[output_])
```

Training the Model

```
model_TL2.compile(loss='sparse_categorical_crossentropy', optimizer='adam',metrics=['accuracy'])

callbacks_TL = [keras.callbacks.ModelCheckpoint("bestM2TL.h5",monitor='val_accuracy',save_weights_only=True,save_best_only=True)]

history_TL = model_TL2.fit(x = X_train, y = y_train, epochs=10, batch_size = 16, validation_split=0.1, callbacks = callbacks_TL)
```

```
Epoch 1/10
21/21 [==============================] - 44s 1s/step - loss: 1.3671 - accuracy: 0.4358 - val_loss: 1.0089 - val_accuracy: 0.6053
Epoch 2/10
21/21 [==============================] - 4s 195ms/step - loss: 0.8145 - accuracy: 0.6716 - val_loss: 0.8794 - val_accuracy: 0.6316
Epoch 3/10
21/21 [==============================] - 3s 156ms/step - loss: 0.6406 - accuracy: 0.7642 - val_loss: 0.8100 - val_accuracy: 0.5789
Epoch 4/10
21/21 [==============================] - 3s 163ms/step - loss: 0.5096 - accuracy: 0.7791 - val_loss: 0.7510 - val_accuracy: 0.6053
Epoch 5/10
21/21 [==============================] - 4s 190ms/step - loss: 0.4734 - accuracy: 0.8030 - val_loss: 0.6898 - val_accuracy: 0.6579
Epoch 6/10
21/21 [==============================] - 3s 161ms/step - loss: 0.3962 - accuracy: 0.8507 - val_loss: 0.6639 - val_accuracy: 0.6579
Epoch 7/10
21/21 [==============================] - 4s 215ms/step - loss: 0.4239 - accuracy: 0.8299 - val_loss: 0.6281 - val_accuracy: 0.6842
Epoch 8/10
21/21 [==============================] - 4s 201ms/step - loss: 0.3266 - accuracy: 0.8746 - val_loss: 0.6314 - val_accuracy: 0.7105
Epoch 9/10
21/21 [==============================] - 3s 145ms/step - loss: 0.3227 - accuracy: 0.8537 - val_loss: 0.6105 - val_accuracy: 0.6579
Epoch 10/10
21/21 [==============================] - 3s 144ms/step - loss: 0.3120 - accuracy: 0.8597 - val_loss: 0.6394 - val_accuracy: 0.6842
```

Testing the Model using Best Weights

```
model_TL2.load_weights("bestM2TL.h5")

testLoss_TL, testAccuracy_TL = model_TL2.evaluate(x = X_test, y = y_test)

print("Test-loss: %f, Test-accuracy: %f" % (testLoss_TL, testAccuracy_TL))
```

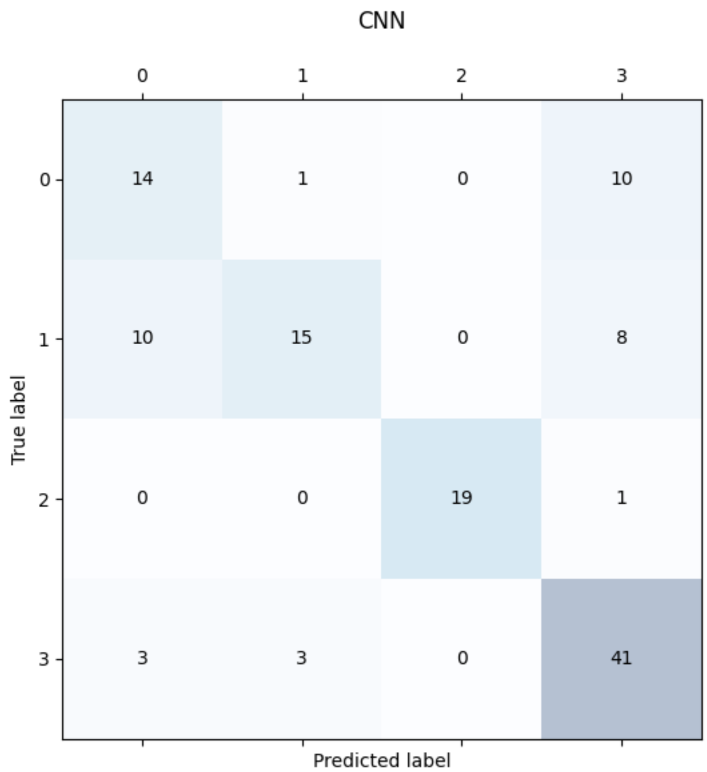
```
4/4 [==============================] - 21s 4s/step - loss: 0.6179 - accuracy: 0.7520
Test-loss: 0.617881, Test-accuracy: 0.752000
```

Checking the performance

```
y_proba = model_TL2.predict(X_test_std)
y_predict = np.argmax(y_proba, axis=-1)
```

```
4/4 [==============================] - 5s 276ms/step
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true = y_test, y_pred = y_predict)

fig, ax = plt.subplots(figsize=(6, 6))
ax.matshow(cm, cmap=plt.cm.Blues, alpha=0.3)

for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(x=j, y=i, s=cm[i, j], va='center', ha='center')

ax.title.set_text('CNN\n')
plt.xlabel('Predicted label')
plt.ylabel('True label')

plt.tight_layout()
plt.savefig("ConfusionMatrix.png", dpi=300, format='png', pad_inches=0.3)
plt.show()

print(set(y))
print(lEncoder.classes_)
```




```
{0, 1, 2, 3}
['andesite' 'basalt' 'coal' 'granite']
```

Saving the Model

```
model_TL2.save('/content/drive/MyDrive/Models/model2_TL.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file vi
  saving_api.save_model(
```