KNNcnn

March 6, 2022

[ ]:
```python
import os
import cv2

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sklearn.metrics as metrics

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report


from sklearn.model_selection import GridSearchCV
from sklearn import svm
from sklearn.multiclass import OneVsRestClassifier
from sklearn.neighbors import KNeighborsClassifier

from tensorflow.python.keras.utils import np_utils
from tensorflow.keras.models import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16, ResNet101, Xception
from tensorflow.keras.layers import Input, Dense, Flatten, MaxPooling2D,␣
↪GlobalAveragePooling2D, Dropout, BatchNormalization, Conv2D, InputLayer
```

[ ]:
```python
SIZE = 224  #Resize images
```

[ ]:
```python
imagePaths = []

for dirname, _, filenames in os.walk(r'Dataset'):
    for filename in filenames:
        if (filename[-3:] == 'png'):
            imagePaths.append(os.path.join(dirname, filename))
```

[ ]:
```python
X = []
y = []

for img_path in imagePaths:
    label = img_path.split(os.path.sep)[-2]

    img = cv2.imread(img_path, cv2.IMREAD_COLOR)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (SIZE,SIZE))


    X.append(img)
    y.append(label)

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

print(type(X), type(y), '\n')
print(X.shape, y.shape)
```

```
<class 'numpy.ndarray'> <class 'numpy.ndarray'>

(8149, 224, 224, 3) (8149,)
```

[ ]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,␣
↪random_state=3)
```

[ ]:
```python
#Encode labels as integers
le = LabelEncoder()
le.fit(y_test)
y_test_labels_encoded = le.transform(y_test)
le.fit(y_train)
y_train_labels_encoded = le.transform(y_train)
```

[ ]:
```python
# Normalize pixel values to between 0 and 1
X_train, X_test = X_train / 255.0, X_test / 255.0
```

[ ]:
```python
y_train_one_hot = np_utils.to_categorical(y_train_labels_encoded)
y_test_one_hot = np_utils.to_categorical(y_test_labels_encoded)
```

[ ]:
```python
VGG_model = VGG16(weights='imagenet', include_top=False, input_shape=(SIZE,␣
↪SIZE, 3))

#Make loaded layers as non-trainable. This is important as we want to work with␣
↪pre-trained weights
for layer in VGG_model.layers:
        layer.trainable = False
```

```python
VGG_model.summary()  #Trainable parameters will be 0
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0         
                                                                 
 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080    
                                                                 
 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0         
                                                                 
 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160   
                                                                 
 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808   
                                                                 
 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808   
                                                                 
 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0         
                                                                 
 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808   
                                                                 
 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0         
                                                                 
=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
```

[ ]:
```
#Now, let us use features from convolutional network for RF
feature_extractor=VGG_model.predict(X_train)
```

[ ]:
```
features = feature_extractor.reshape(feature_extractor.shape[0], -1)

X_for_KNN = features #This is our X input to RF
```

[ ]:
```

knn = OneVsRestClassifier(KNeighborsClassifier())
knn.fit(X_for_KNN, y_train_labels_encoded)
```

[ ]:
```
OneVsRestClassifier(estimator=KNeighborsClassifier())
```

[ ]:
```
X_test_feature = VGG_model.predict(X_test)
X_test_features = X_test_feature.reshape(X_test_feature.shape[0], -1)
```

[ ]:
```
prediction_svm = knn.predict(X_test_features)
#Inverse le transform to get original label back.
prediction_svm = le.inverse_transform(prediction_svm)
```

[ ]:
```
print ("Accuracy = ", metrics.accuracy_score(y_test, prediction_svm)*100)



cm = confusion_matrix(y_test, prediction_svm)
print(cm)
cm_df = pd.DataFrame(cm, index=le.classes_, columns=le.classes_)
cm_df.head()
```

```
Accuracy =  93.61963190184049
[[1051   34    9]
 [  79  844   10]
 [   2   22  394]]
```

[ ]:

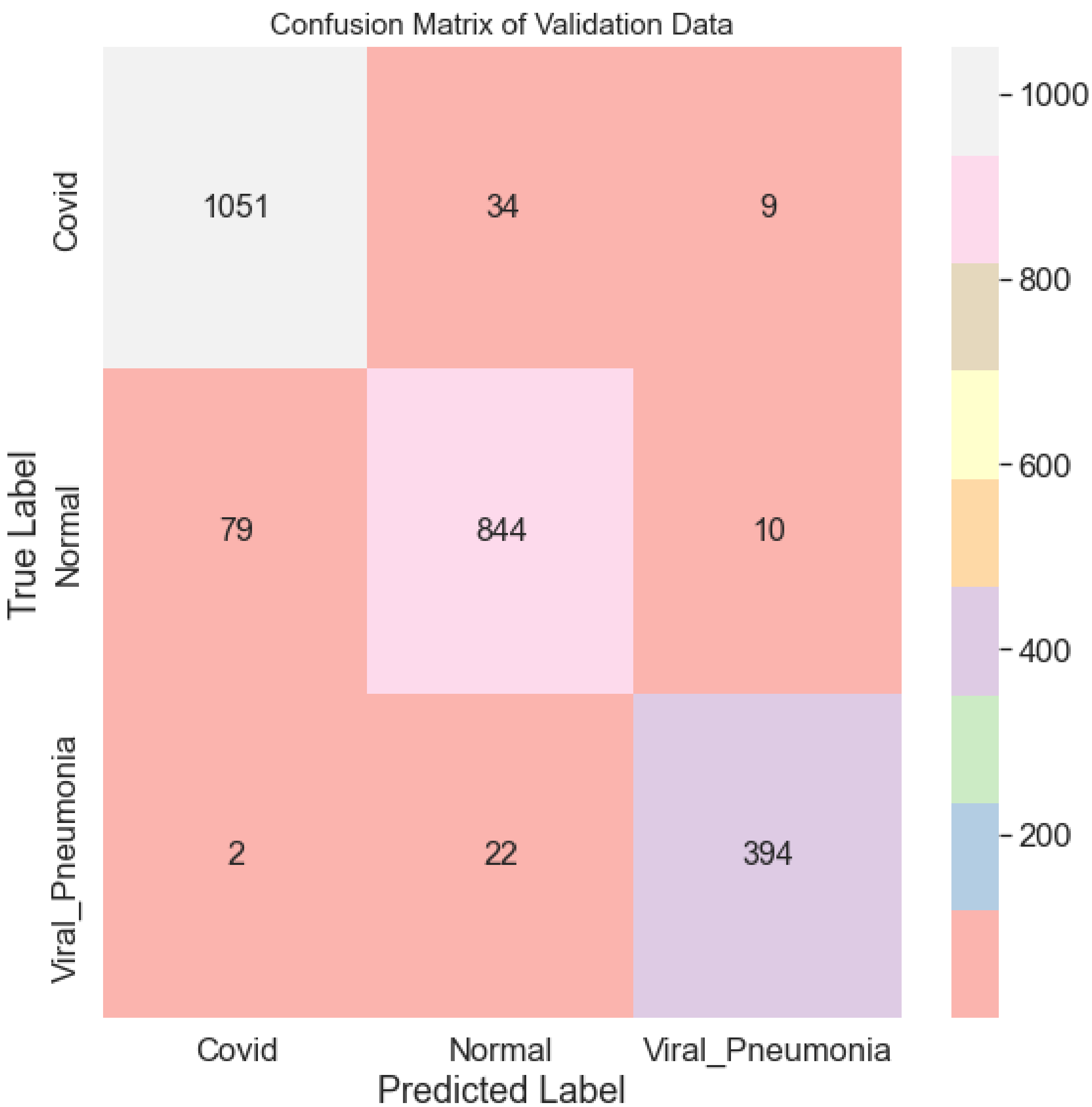
	Covid	Normal	Viral_Pneumonia
Covid	1051	34	9
Normal	79	844	10
Viral_Pneumonia	2	22	394

[ ]:
```
plt.figure(figsize=(9,9))

sns.set(font_scale=1.5, color_codes=True, palette='deep')
sns.heatmap(cm_df, annot=True, annot_kws={'size':16}, fmt='d', cmap='Pastel1')

plt.ylabel("True Label")
plt.xlabel("Predicted Label")
plt.title('Confusion Matrix of Validation Data',size=15)
```

```python
plt.show()
```



[ ]:
```python
print(classification_report(y_test, prediction_svm))
```

```
                 precision    recall  f1-score   support

          Covid       0.93      0.96      0.94      1094
         Normal       0.94      0.90      0.92       933
Viral_Pneumonia       0.95      0.94      0.95       418

       accuracy                           0.94      2445
      macro avg       0.94      0.94      0.94      2445
   weighted avg       0.94      0.94      0.94      2445
```

```
[ ]:
```