DEEP LEARNING ASSIGNMENT

Image Classification Using Convolutional Neural Networks

Akash.E
2018103005
CSE - ‘P’ Batch
19/04/2021

PROBLEM STATEMENT :

Pneumonia is an inflammatory condition of the lung affecting primarily the small air sacs known as alveoli. We build a CNN model to predict if a given Chest X-Ray image is 'Normal' or affected with 'Pneumonia'. The disease may be classified by where it was acquired, such as community- or hospital-acquired or healthcare-associated pneumonia.

DATASET DETAILS :

The dataset is organized into 3 folders (train, test, val) and contains subfolders for each image category (Pneumonia/Normal).

There are 5,863 X-Ray images (JPEG) and 2 categories (Pneumonia/Normal). Chest X-ray images (anterior-posterior) were selected from retrospective cohorts of pediatric patients of one to five years old.

For the analysis of chest x-ray images, all chest radiographs were initially screened for quality control by removing all low quality or unreadable scans.

MODULE LIST :

1. Fetching the Data/Visualizing the Data.
2. Data Pre-processing.
3. Building the Model
4. Training and Validation.
5. Results and Visualization

1. FETCHING THE DATA /VISUALIZING THE DATA :

A function is defined to fetch the dataset and store it in three separate arrays for train,test and validation.The corresponding path is specified to load the dataset into the arrays.

cv2.imread function is used to read the dataset and convert it to grayscale.

```python
#Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D , MaxPool2D , Flatten , Dropout , BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
from keras.callbacks import ReduceLROnPlateau
import cv2
import os
```

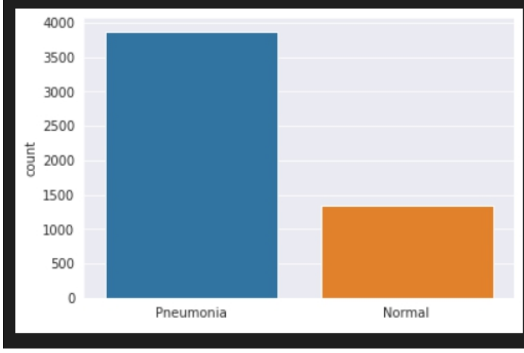
```python
#Function the fetch the data
labels = ['PNEUMONIA', 'NORMAL']
img_size = 150
def get_training_data(data_dir):
    data = []
    for label in labels:
        path = os.path.join(data_dir, label)
        class_num = labels.index(label)
        for img in os.listdir(path):
            try:
                img_arr = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
                resized_arr = cv2.resize(img_arr, (img_size, img_size)) # Reshaping images to preferred size
                data.append([resized_arr, class_num])
            except Exception as e:
                print(e)
    return np.array(data)
```

```python
#Separate Arrays to store the data
train = get_training_data('../input/chest-xray-pneumonia/chest_xray/chest_xray/train')
test = get_training_data('../input/chest-xray-pneumonia/chest_xray/chest_xray/test')
val = get_training_data('../input/chest-xray-pneumonia/chest_xray/chest_xray/val')
```

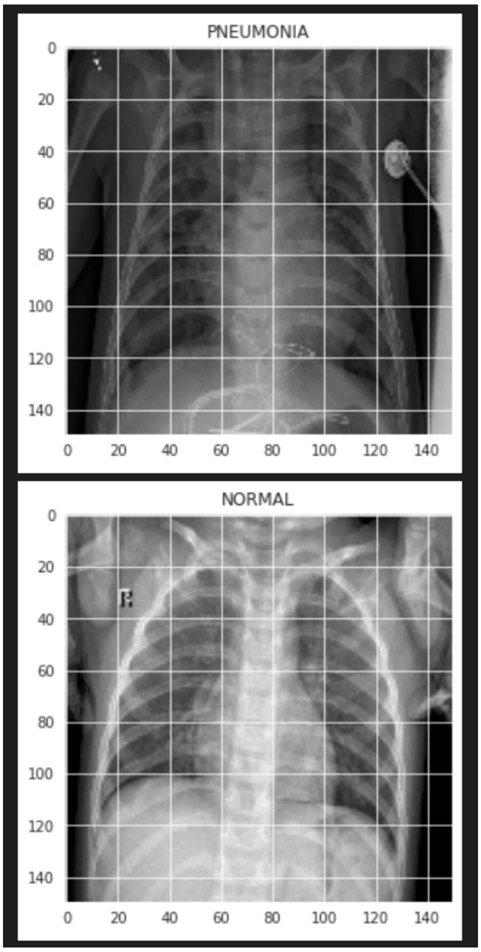
Data population and Visualizing the data

```python
l = []
for i in train:
    if(i[1] == 0):
        l.append("Pneumonia")
    else:
        l.append("Normal")
sns.set_style('darkgrid')
sns.countplot(l)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6e49049320>
```



```python
plt.figure(figsize = (5,5))
plt.imshow(train[0][0], cmap='gray')
plt.title(labels[train[0][1]])

plt.figure(figsize = (5,5))
plt.imshow(train[-1][0], cmap='gray')
plt.title(labels[train[-1][1]])
```



2. DATA PRE-PROCESSING :

Now the data is separated as x and y in the three arrays and preprocessing steps such as normalization and scaling occurs.

In order to avoid overfitting problem, we need to expand artificially our dataset. We can make your existing dataset even larger. The idea is to alter the training data with small transformations to reproduce the variations.

Approaches that alter the training data in ways that change the array representation while keeping the label the same are known as data augmentation techniques.

Some popular augmentations people use are grayscales, horizontal flips, vertical flips, random crops, color jitters, translations, rotations, and much more.

```python
x_train = []
y_train = []

x_val = []
y_val = []

x_test = []
y_test = []

for feature, label in train:
    x_train.append(feature)
    y_train.append(label)

for feature, label in test:
    x_test.append(feature)
    y_test.append(label)

for feature, label in val:
    x_val.append(feature)
    y_val.append(label)
```

```python
# Normalize the data
x_train = np.array(x_train) / 255
x_val = np.array(x_val) / 255
x_test = np.array(x_test) / 255
```

```python
# resize data for deep learning
x_train = x_train.reshape(-1, img_size, img_size, 1)
y_train = np.array(y_train)

x_val = x_val.reshape(-1, img_size, img_size, 1)
y_val = np.array(y_val)

x_test = x_test.reshape(-1, img_size, img_size, 1)
y_test = np.array(y_test)
```

By applying just a couple of these transformations to our training data, we can easily double or triple the number of training examples and create a very robust model.

For data augmentation :

- Randomly rotate some training images by 30 degrees
- Randomly Zoom by 20% some training images
- Randomly shift images horizontally by 10% of the width
- Randomly shift images vertically by 10% of the height
- Randomly flip images horizontally. Once our model is ready, we fit the training dataset.

```python
# With data augmentation to prevent overfitting and handling the imbalance in dataset

datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range = 30,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.2, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip = True,  # randomly flip images
        vertical_flip=False)  # randomly flip images


datagen.fit(x_train)
```

3. BUILDING THE MODEL :

Now we build our CNN model with the following specifications :

```python
model = Sequential()
model.add(Conv2D(32 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu' , input_shape = (150,150,1)))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(64 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(Dropout(0.1))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(64 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(128 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Conv2D(256 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))
model.add(Flatten())
model.add(Dense(units = 128 , activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(units = 1 , activation = 'sigmoid'))
model.compile(optimizer = "rmsprop" , loss = 'binary_crossentropy' , metrics = ['accuracy'])
model.summary()
```

Model summary :

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 150, 150, 32)      320
_________________________________________________________________
batch_normalization_1 (Batch (None, 150, 150, 32)      128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 75, 75, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 75, 75, 64)        18496
_________________________________________________________________
dropout_1 (Dropout)          (None, 75, 75, 64)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 75, 75, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 38, 38, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 38, 38, 64)        36928
_________________________________________________________________
batch_normalization_3 (Batch (None, 38, 38, 64)        256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 19, 19, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 19, 19, 128)       73856
dropout_2 (Dropout)          (None, 19, 19, 128)       0
_________________________________________________________________
batch_normalization_4 (Batch (None, 19, 19, 128)       512
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 10, 10, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 10, 10, 256)       295168
_________________________________________________________________
dropout_3 (Dropout)          (None, 10, 10, 256)       0
_________________________________________________________________
batch_normalization_5 (Batch (None, 10, 10, 256)       1024
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 5, 5, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6400)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               819328
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,246,401
Trainable params: 1,245,313
Non-trainable params: 1,088
```

4. TRAINING AND VALIDATION :

Now we train our data with the CNN we created. Reduce learning rate when a metric has stopped improving. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates.

This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.For this we use **ReduceLROnPlateau** class.

Parameters :

Learning Rate	Default LR = 0.01 , reduced by a factor 0.3.
Epochs	5
Batch Size	32
Optimizer	Root Mean Square Prop - ‘rmsprop’
Loss	Binary Cross Entropy

```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy', patience = 2, verbose=1,factor=0.3, min_lr=0.000001)
```

```
history = model.fit(datagen.flow(x_train,y_train, batch_size = 32) ,epochs = 5 , validation_data = datagen.flow(x_val, y_val) ,callbacks = [learning_rate_reduction])
```

```
Epoch 1/5
163/163 [==============================] - 281s 2s/step - loss: 0.3494 - accuracy: 0.8620 - val_loss: 9.9030 - val_accuracy: 0.5000
Epoch 2/5
163/163 [==============================] - 250s 2s/step - loss: 0.2613 - accuracy: 0.9011 - val_loss: 14.1557 - val_accuracy: 0.5000
Epoch 3/5
163/163 [==============================] - 250s 2s/step - loss: 0.2093 - accuracy: 0.9285 - val_loss: 25.8908 - val_accuracy: 0.5000

Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.0003000000142492354.
Epoch 4/5
163/163 [==============================] - 273s 2s/step - loss: 0.1464 - accuracy: 0.9519 - val_loss: 2.1655 - val_accuracy: 0.5625
Epoch 5/5
163/163 [==============================] - 253s 2s/step - loss: 0.1305 - accuracy: 0.9536 - val_loss: 3.7105 - val_accuracy: 0.5000
```

Accuracy and Loss :

```
print("Loss of the model is - " , model.evaluate(x_test,y_test)[0])
print("Accuracy of the model is - " , model.evaluate(x_test,y_test)[1]*100 , "%")
```

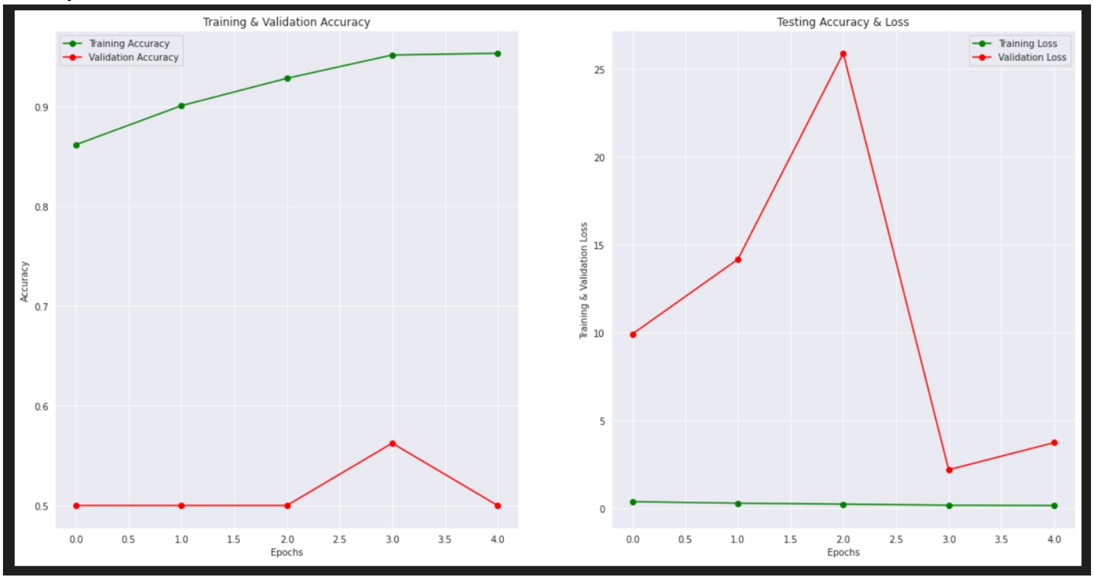
```
624/624 [==============================] - 9s 14ms/step
Loss of the model is -  0.3520233004521101
624/624 [==============================] - 8s 13ms/step
Accuracy of the model is -  87.66025900840759 %
```

5. RESULTS AND VISUALIZATION :

Analysis after training the model :

```
epochs = [i for i in range(5)]
fig , ax = plt.subplots(1,2)
train_acc = history.history['accuracy']
train_loss = history.history['loss']
val_acc = history.history['val_accuracy']
val_loss = history.history['val_loss']
fig.set_size_inches(20,10)

ax[0].plot(epochs , train_acc , 'go-' , label = 'Training Accuracy')
ax[0].plot(epochs , val_acc , 'ro-' , label = 'Validation Accuracy')
ax[0].set_title('Training & Validation Accuracy')
ax[0].legend()
ax[0].set_xlabel("Epochs")
ax[0].set_ylabel("Accuracy")

ax[1].plot(epochs , train_loss , 'g-o' , label = 'Training Loss')
ax[1].plot(epochs , val_loss , 'r-o' , label = 'Validation Loss')
ax[1].set_title('Testing Accuracy & Loss')
ax[1].legend()
ax[1].set_xlabel("Epochs")
ax[1].set_ylabel("Training & Validation Loss")
plt.show()
```

Graph Plot :



Classification Report and Confusion Matrix :

```
print(classification_report(y_test, predictions, target_names = ['Pneumonia (Class 0)','Normal (Class 1)']))
```

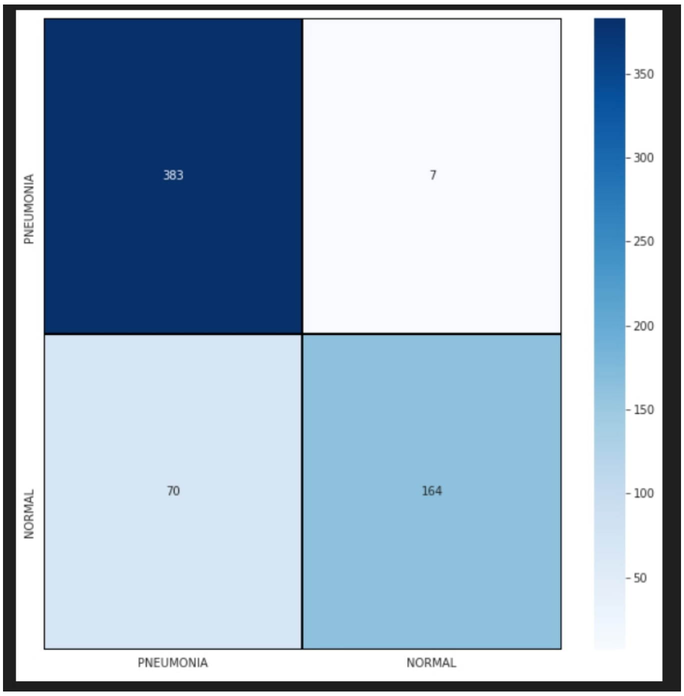
```
                     precision    recall  f1-score   support

Pneumonia (Class 0)       0.85      0.98      0.91       390
   Normal (Class 1)       0.96      0.70      0.81       234

           accuracy                           0.88       624
          macro avg       0.90      0.84      0.86       624
       weighted avg       0.89      0.88      0.87       624
```

```
cm = confusion_matrix(y_test,predictions)
cm
```

```
array([[383,   7],
       [ 70, 164]])
```

Heat map :

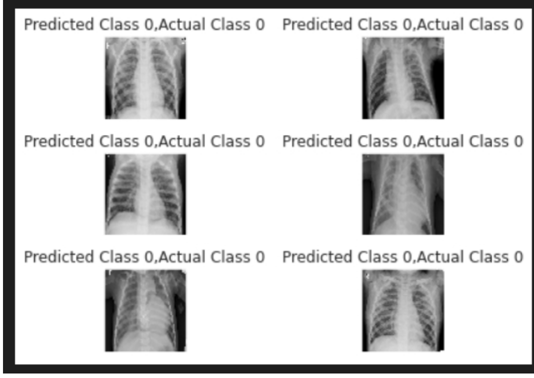
```
plt.figure(figsize = (10,10))
sns.heatmap(cm,cmap= "Blues", linecolor = 'black' , linewidth = 1 , annot = True, fmt='',xticklabels = labels,yticklabels = labels)
```



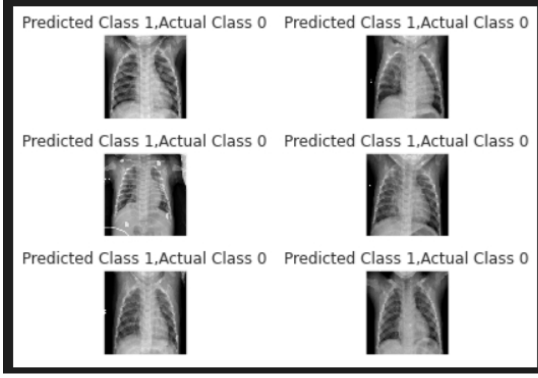
Some correctly and incorrectly predicted images :

```
correct = np.nonzero(predictions == y_test)[0]
incorrect = np.nonzero(predictions != y_test)[0]
```

```
#Correct Classifications
i = 0
for c in correct[:6]:
    plt.subplot(3,2,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_test[c].reshape(150,150), cmap="gray", interpolation='none')
    plt.title("Predicted Class {},Actual Class {}".format(predictions[c], y_test[c]))
    plt.tight_layout()
    i += 1
```



```python
#Incorrect Classifications
i = 0
for c in incorrect[:6]:
    plt.subplot(3,2,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_test[c].reshape(150,150), cmap="gray", interpolation='none')
    plt.title("Predicted Class {},Actual Class {}".format(predictions[c], y_test[c]))
    plt.tight_layout()
    i += 1
```



CONCLUSION :

Thus we have created a CNN model which can classify CXR images as normal or pneumonia with high accuracy.

The world of medical imaging is ripe for a revolution in terms of deploying CNN based technologies. There is no need for a doctor or a health care provider to ponder these images to gauge things. The task of reading these is incredibly menial and repititive. Those are two things that AI technologies are great at.

REFERENCES :

Dataset : https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

Conv Nets : https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Python Implementation : https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python