*# This Python 3 environment comes with many helpful analytics libraries installed*

*# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python*

*# For example, here's several helpful packages to load in*

​

**import** numpy **as** np *# linear algebra*

**import** pandas **as** pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*

​

*# Input data files are available in the "../input/" directory.*

*# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory*

​

**import** os

**for** dirname, \_, filenames **in** os.walk('/kaggle/input'):

**for** filename **in** filenames:

print(os.path.join(dirname, filename))

​

*# Any results you write to the current directory are saved as output.*

**Expand**

add Codeadd Markdown

**Importing the necessary libraries**

add Codeadd Markdown

[ ]:



**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

add Codeadd Markdown

**Description of the Pneumonia Dataset**

**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 from Guangzhou Women and Children’s Medical Center, Guangzhou. All chest X-ray imaging was performed as part of patients’ routine clinical care. 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. The diagnoses for the images were then graded by two expert physicians before being cleared for training the AI system. In order to account for any grading errors, the evaluation set was also checked by a third expert.**

add Codeadd Markdown

[ ]:



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)

add Codeadd Markdown

**Loading the Dataset**

add Codeadd Markdown

[ ]:



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')

**Expand**

add Codeadd Markdown

**Data Visualization & Preprocessing**

add Codeadd Markdown

[ ]:



l **=** []

**for** i **in** train:

**if**(i[1] **==** 0):

l.append("Pneumonia")

**else**:

l.append("Normal")

sns.set\_style('darkgrid')

sns.countplot(l)

add Codeadd Markdown

**The data seems imbalanced . To increase the no. of training examples, we will use data augmentation**

add Codeadd Markdown

**Previewing the images of both the classes**

add Codeadd Markdown

[ ]:



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]])

add Codeadd Markdown

[ ]:



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)

add Codeadd Markdown

**We perform a grayscale normalization to reduce the effect of illumination's differences.Moreover the CNN converges faster on [0..1] data than on [0..255].**

add Codeadd Markdown

[ ]:



*# Normalize the data*

x\_train **=** np.array(x\_train) **/** 255

x\_val **=** np.array(x\_val) **/** 255

x\_test **=** np.array(x\_test) **/** 255

add Codeadd Markdown

[ ]:



*# 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)

add Codeadd Markdown

**Data Augmentation**

**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. 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.**

add Codeadd Markdown

[ ]:



*# 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)

add Codeadd Markdown

For the data augmentation, i choosed to :

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

add Codeadd Markdown

**Training the Model**

add Codeadd Markdown

[ ]:



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()

add Codeadd Markdown

[ ]:



learning\_rate\_reduction **=** ReduceLROnPlateau(monitor**=**'val\_accuracy', patience **=** 2, verbose**=**1,factor**=**0.3, min\_lr**=**0.000001)

add Codeadd Markdown

[ ]:



history **=** model.fit(datagen.flow(x\_train,y\_train, batch\_size **=** 32) ,epochs **=** 12 , validation\_data **=** datagen.flow(x\_val, y\_val) ,callbacks **=** [learning\_rate\_reduction])

add Codeadd Markdown

[ ]:



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 , "%")

add Codeadd Markdown

**Analysis after Model Training**

add Codeadd Markdown

[ ]:



epochs **=** [i **for** i **in** range(12)]

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()

add Codeadd Markdown

[ ]:



predictions **=** model.predict\_classes(x\_test)

predictions **=** predictions.reshape(1,**-**1)[0]

predictions[:15]

add Codeadd Markdown

[ ]:



print(classification\_report(y\_test, predictions, target\_names **=** ['Pneumonia (Class 0)','Normal (Class 1)']))

add Codeadd Markdown

[ ]:



cm **=** confusion\_matrix(y\_test,predictions)

cm

add Codeadd Markdown

[ ]:



cm **=** pd.DataFrame(cm , index **=** ['0','1'] , columns **=** ['0','1'])

add Codeadd Markdown

[ ]:



plt.figure(figsize **=** (10,10))

sns.heatmap(cm,cmap**=** "Blues", linecolor **=** 'black' , linewidth **=** 1 , annot **=** **True**, fmt**=**'',xticklabels **=** labels,yticklabels **=** labels)

add Codeadd Markdown

[ ]:



correct **=** np.nonzero(predictions **==** y\_test)[0]

incorrect **=** np.nonzero(predictions **!=** y\_test)[0]

add Codeadd Markdown

**Some of the Correctly Predicted Classes**

add Codeadd Markdown

[ ]:



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

add Codeadd Markdown

**Some of the Incorrectly Predicted Classes**

add Codeadd Markdown

[ ]:



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

add Codeadd Markdown

[ ]:



​