DEEP LEARNING: CNN

COURSE PROJECT REPORT

In the past few decades, Deep Learning has proved to be a very powerful tool because of its ability to handle large amounts of data. The interest to use hidden layers has surpassed traditional techniques, especially in pattern recognition. One of the most popular deep neural networks is Convolutional Neural Networks.

ABOUT THE DATA

Context

Pneumonia is an infection that inflames the air sacs in one or both lungs. It kills more children younger than 5 years old each year than any other infectious disease, such as HIV infection, malaria, or tuberculosis. Diagnosis is often based on symptoms and physical examination. Chest X-rays may help confirm the diagnosis.

Content

This dataset contains 5,856 validated Chest X-Ray images. The images are split into a training set and a testing set of independent patients. Images are labeled as (*disease*:NORMAL/BACTERIA/VIRUS)-(*randomized patient I*D)-(*image number of a patient*). For details of the data collection and description, see the referenced paper below.

According to the paper, the 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.

A previous version (v2) of this dataset is available here: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia. Note that the files names are irregular in v2, but they are fixed in the new version (v3).

Inspiration

This data will be useful for developing/training/testing classification models with convolutional neural networks.

OBJECTIVES FROM THE ANALYSIS

To classify the color of wine as red or white

1. Data Generator class

2. Simple EDA

· Preprocessing

3. Applying CNN Algorithms tried 3 different models along with optimizers

4. VGG 16 and then applied transfer learning

5. Conclusion

METHODS USED

1. Import all the required libraries and data

2. Preprocessing the data

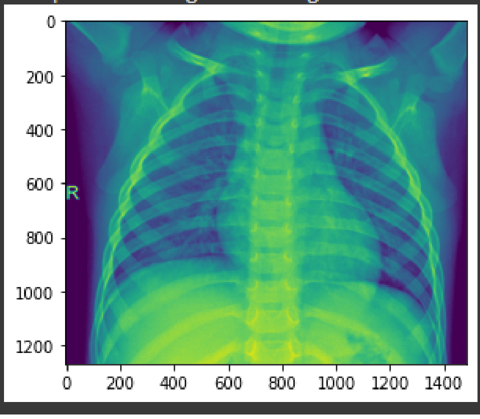
```
def process_data(img_dims, batch_size):
    # create three Data generation objects for train test and validation
    train_datagen = ImageDataGenerator(rescale=1./255)
    test_datagen = ImageDataGenerator(rescale=1./255)
    val_datagen = ImageDataGenerator(rescale=1./255)

    # Write the code for train,val,test batches which will be fed to the network in the specified batch sizes and image dimensions
     #solution for train_gen:(refer to this and similary write for test and validation batch)
    train_gen = train_datagen.flow_from_directory(
    directory=train_path,
    target_size=(img_dims, img_dims),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=True)

    test_gen = test_datagen.flow_from_directory(
    directory=test_path,
    target_size=(img_dims, img_dims),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=True)

    val_gen = val_datagen.flow_from_directory(
    directory=val_path,
    target_size=(img_dims, img_dims),
    batch_size=batch_size,
    class_mode='binary',
    shuffle=True)

    return train_gen, test_gen, val_gen
```

3. Visualize the data

```
normal_example = os.listdir('/content/drive/MyDrive/chest_xray/train/NORMAL')[0]
```

```
normal_img = plt.imread(f'/content/drive/MyDrive/chest_xray/train/NORMAL/{normal_example}')
plt.imshow(normal_img)
```

```
<matplotlib.image.AxesImage at 0x7f519f255e10>
```



4. Type 1 model:

```
def model_2():
    model = Sequential([
    Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding = 'same', input_shape=(150,150,3)),
    BatchNormalization(),  #--------------batch normalization--------#
    MaxPool2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation="relu", padding='same'),
    BatchNormalization(),
    MaxPool2D(pool_size=(2, 2)),
    Conv2D(128, (3, 3), activation="relu", padding='same'),
     BatchNormalization(),
    MaxPool2D(pool_size=(3, 3)),
    Flatten(),
    Dense(120, activation='relu'),
    Dense(60, activation='relu'),
    Dropout(rate=0.2),    #---------dropout----------------------#
    Dense(1, activation='sigmoid') ,

    ])



    return model
```

```
es = EarlyStopping(patience=15, monitor='val_accuracy', restore_best_weights=True)

hist= model_2.fit(train_gen, steps_per_epoch=train_gen.samples // batch_size,
          epochs=epochs, validation_data=test_gen,
          validation_steps=test_gen.samples // batch_size,
          verbose=2,callbacks=[es])
```

```
Epoch 1/10
163/163 - 365s - loss: 0.0273 - accuracy: 0.9906 - val_loss: 1.1971 - val_accuracy: 0.8059 - 365s/epoch - 2s/step
Epoch 2/10
163/163 - 365s - loss: 0.0110 - accuracy: 0.9960 - val_loss: 0.9346 - val_accuracy: 0.8158 - 365s/epoch - 2s/step
Epoch 3/10
163/163 - 364s - loss: 0.0058 - accuracy: 0.9981 - val_loss: 2.8561 - val_accuracy: 0.7336 - 364s/epoch - 2s/step
Epoch 4/10
163/163 - 365s - loss: 0.0116 - accuracy: 0.9969 - val_loss: 1.2335 - val_accuracy: 0.8273 - 365s/epoch - 2s/step
Epoch 5/10
163/163 - 364s - loss: 0.0292 - accuracy: 0.9893 - val_loss: 1.7446 - val_accuracy: 0.7895 - 364s/epoch - 2s/step
Epoch 6/10
163/163 - 368s - loss: 0.0132 - accuracy: 0.9954 - val_loss: 1.6134 - val_accuracy: 0.7829 - 368s/epoch - 2s/step
Epoch 7/10
163/163 - 365s - loss: 0.0202 - accuracy: 0.9937 - val_loss: 1.8053 - val_accuracy: 0.7895 - 365s/epoch - 2s/step
Epoch 8/10
163/163 - 363s - loss: 0.0073 - accuracy: 0.9967 - val_loss: 1.0350 - val_accuracy: 0.8388 - 363s/epoch - 2s/step
Epoch 9/10
163/163 - 364s - loss: 0.0015 - accuracy: 0.9998 - val_loss: 3.8232 - val_accuracy: 0.7270 - 364s/epoch - 2s/step
Epoch 10/10
163/163 - 364s - loss: 5.8120e-04 - accuracy: 1.0000 - val_loss: 5.3907 - val_accuracy: 0.7105 - 364s/epoch - 2s/step
```

5. Type 2 model:

```
model_4.compile(optimizer='adam',
                loss='binary_crossentropy',
                metrics="accuracy")
es = EarlyStopping(patience=15, monitor='accuracy', restore_best_weights=True)

hist= model_4.fit(train_gen, steps_per_epoch=train_gen.samples // batch_size,
          epochs=epochs, validation_data=test_gen,
          validation_steps=test_gen.samples // batch_size,
          verbose=2,callbacks=[es])
```

```
Epoch 1/10
163/163 - 403s - loss: 0.1367 - accuracy: 0.7422 - 403s/epoch - 2s/step
Epoch 2/10
163/163 - 399s - loss: 0.0983 - accuracy: 0.7422 - 399s/epoch - 2s/step
Epoch 3/10
163/163 - 399s - loss: 0.0807 - accuracy: 0.7425 - 399s/epoch - 2s/step
Epoch 4/10
163/163 - 400s - loss: 0.0715 - accuracy: 0.7425 - 400s/epoch - 2s/step
Epoch 5/10
163/163 - 400s - loss: 0.0711 - accuracy: 0.7439 - 400s/epoch - 2s/step
Epoch 6/10
163/163 - 400s - loss: 0.0495 - accuracy: 0.7425 - 400s/epoch - 2s/step
Epoch 7/10
163/163 - 399s - loss: 0.0508 - accuracy: 0.7429 - 399s/epoch - 2s/step
Epoch 8/10
163/163 - 399s - loss: 0.0381 - accuracy: 0.7427 - 399s/epoch - 2s/step
Epoch 9/10
163/163 - 398s - loss: 0.0349 - accuracy: 0.7418 - 398s/epoch - 2s/step
Epoch 10/10
163/163 - 399s - loss: 0.0433 - accuracy: 0.7431 - 399s/epoch - 2s/step
```

6. Transfer learning:
VGG model:

```python
#Defining input image shape
input = Input((224,224,1))
```

```python
#1st set of convo layer
conv1  = Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu")(input)
conv2  = Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu")(conv1)
#layer of pooling
pool1  = MaxPool2D((2, 2))(conv2)
#fill the following code for the second set of convo layer
conv3  = Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu")(pool1)
conv4  = Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu")(conv3)
pool2  = MaxPool2D((2, 2))(conv4)
#Notice that the third set of convo layer contains 3 consecutive
conv5  = Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(pool2)
conv6  = Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(conv5)
conv7  = Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")(conv6)
#layer of pooling
pool3  = MaxPool2D((2, 2))(conv7)

#Complete the fourth set of convo layers
conv8  = Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(pool3)
conv9  = Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(conv8)
conv10 = Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(conv9)
#layer of pooling
pool4  = MaxPool2D((2, 2))(conv10)

#fifth set of convo

conv11  = Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(pool4)
conv12  = Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(conv11)
conv13 = Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")(conv12)

#layer of pooling
pool5=pool3  = MaxPool2D((2, 2))(conv13)

#flatten the layers
flat   = Flatten()(pool5)
dense1 = Dense(4096, activation="relu")(flat)
dense2 = Dense(4096,activation="relu")(dense1)
output = Dense(1000,activation="softmax")(dense2)

vgg16_model  = Model(inputs=input, outputs=output)
```

Model parameters

```python
vgg = VGG16(input_shape=[224,224]+[3], weights='imagenet', include_top=False)

# don't train existing weights
for layer in vgg.layers:
    layer.trainable = False

# our layers
x = Flatten()(vgg.output)
prediction = Dense(1, activation='softmax')(x)


# create a model object
model = Model(inputs=vgg.input, outputs=prediction)
```

Compile and fit the model

```
#compile the model
model.compile(optimizer='adam',
                 loss='categorical_crossentropy',
                 metrics="accuracy")
```

```
#fit the model
es = EarlyStopping(patience=15, monitor='accuracy', restore_best_weights=True)

hist= model.fit(train_gen, steps_per_epoch=train_gen.samples // batch_size,
          epochs=epochs, validation_data=test_gen,
          validation_steps=test_gen.samples // batch_size,
          verbose=2,callbacks=[es])
```

```
Epoch 1/10
163/163 - 978s - loss: 0.0000e+00 - accuracy: 0.7422 - val_loss: 0.0000e+00 - val_accuracy: 0.6217 - 978s/epoch - 6s/step
Epoch 2/10
163/163 - 80s - loss: 0.0000e+00 - accuracy: 0.7431 - val_loss: 0.0000e+00 - val_accuracy: 0.6283 - 80s/epoch - 492ms/step
Epoch 3/10
163/163 - 80s - loss: 0.0000e+00 - accuracy: 0.7431 - val_loss: 0.0000e+00 - val_accuracy: 0.6250 - 80s/epoch - 490ms/step
Epoch 4/10
163/163 - 82s - loss: 0.0000e+00 - accuracy: 0.7429 - val_loss: 0.0000e+00 - val_accuracy: 0.6316 - 82s/epoch - 502ms/step
Epoch 5/10
163/163 - 82s - loss: 0.0000e+00 - accuracy: 0.7425 - val_loss: 0.0000e+00 - val_accuracy: 0.6266 - 82s/epoch - 500ms/step
Epoch 6/10
163/163 - 81s - loss: 0.0000e+00 - accuracy: 0.7431 - val_loss: 0.0000e+00 - val_accuracy: 0.6201 - 81s/epoch - 498ms/step
Epoch 7/10
163/163 - 81s - loss: 0.0000e+00 - accuracy: 0.7429 - val_loss: 0.0000e+00 - val_accuracy: 0.6217 - 81s/epoch - 499ms/step
Epoch 8/10
163/163 - 83s - loss: 0.0000e+00 - accuracy: 0.7435 - val_loss: 0.0000e+00 - val_accuracy: 0.6283 - 83s/epoch - 509ms/step
Epoch 9/10
163/163 - 82s - loss: 0.0000e+00 - accuracy: 0.7425 - val_loss: 0.0000e+00 - val_accuracy: 0.6299 - 82s/epoch - 506ms/step
Epoch 10/10
163/163 - 81s - loss: 0.0000e+00 - accuracy: 0.7437 - val_loss: 0.0000e+00 - val_accuracy: 0.6184 - 81s/epoch - 499ms/step
```

Brief of the 3 models:
We suggest that out of the three models we get the best accuracy in type 2 model but further improvements can be made. This can be done by varying the parameters of the model for getting a better accuracy.

CONCLUSION
Concepts of deep learning in cnn were understood and executed very well.Optimizers and early stopping were used to help with the computational cost. Vgg 16 was used and so was transfer learning.