X-RAY Bone Classification

With MURA database and DenseNet169 pre-trained on imagenet

By: Aileen Dugan

Setup of Data with pandas dataframes

Extract only the shoulder x-ray images for both training images and then testing images using the .csv files provided by MURA

```
[ ]
#A function to generate the dataframe for a csv file
def generate_df(dataset_root, csv_name, BODYPART):
    df = pd.read_csv(dataset_root/csv_name, header=None, names=['filename'])
    df['class'] = (df.filename
                   .str.extract('study.*_(positive|negative)'))
    df['BodyPart'] = (df.filename
                   .str.extract('XR_(SHOULDER|ELBOW|FINGER|FOREARM|HAND|HUMERUS|WRIST)'))
    bodypart = df[(df["BodyPart"]==BODYPART)]
    df_onlyOne = bodypart[["filename","class"]]
    return df_onlyOne
```

```
df_train = generate_df(dataset_root, 'train_image_paths.csv',"SHOULDER")
df_train.head()
```

	filename	class
0	MURA-v1.1/train/XR_SHOULDER/patient00001/study...	positive
1	MURA-v1.1/train/XR_SHOULDER/patient00001/study...	positive
2	MURA-v1.1/train/XR_SHOULDER/patient00001/study...	positive
3	MURA-v1.1/train/XR_SHOULDER/patient00002/study...	positive
4	MURA-v1.1/train/XR_SHOULDER/patient00002/study...	positive

Turn DataFrames into ndarrays of images and labels

Get images from dataframe filepaths and size them properly to stack into an ndarray. Also create an array of labels using the dataframe’s class of positive or negative for each image

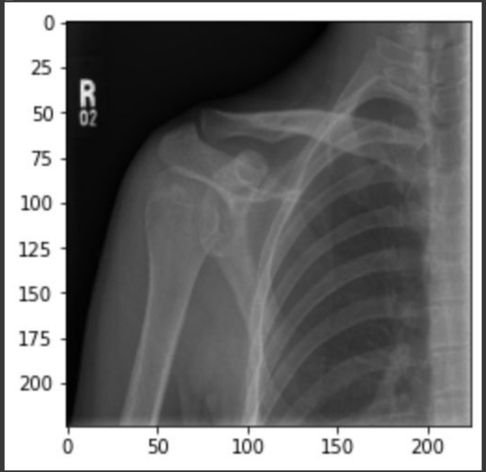
```python
img_path = '/content/drive/MyDrive/MURA/'
```

```python
#take dataframe and make ndarray items for training and testing - fit to densenet keep 3 channels wit
x_train = []
y_train = []
for index,row in df_train.iterrows():
  input_arr = cv2.imread(os.path.join(img_path,row["filename"]))
  #, cv2.IMREAD_GRAYSCALE)
  input_arr = cv2.resize(input_arr, (224,224))
  label = row["class"]
  x_train.append(input_arr)
  y_train.append(label)
x_train = np.stack(x_train, axis=0)
y_train = np.array(y_train)
```

Show an example image to make sure this was done properly - shows image and it’s label

```
# Let's show one example from the dataset
image_index = 688 # you may select anything from 0 to 8378 (we have 8379 training samples for shoulde
print(y_train[image_index])
plt.imshow(x_train[image_index], cmap='Greys')
plt.show()
```

```
positive
```



Reshape ndarrays to fit for imagenet as 4-dimensional with 3 layers, float32 type, and 8-bit grayscale intensities

```
# Reshaping the array to 4-dims so that it can work with the Keras API
x_train = x_train.reshape(x_train.shape[0], 224, 224, 3)
x_test = x_test.reshape(x_test.shape[0], 224, 224, 3)
input_shape = (224, 224, 3)

# Making sure that the values are float so that we can get decimal points after division
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# Normalizing the 8-bit grayscale intensities by dividing them by the max intensity value
x_train /= 255
x_test /= 255
```

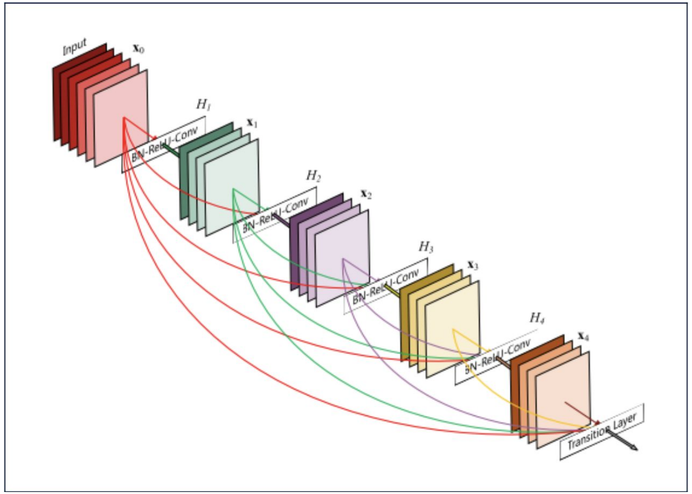
Download and build the DenseNet169 Architecture from tf.keras

```
[ ]  1 #Downloading the densenet model pretrained on the imagenet dataset
     2 densenet = tf.keras.applications.DenseNet169(weights='imagenet', include_top = False, input_shape=(224

    Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet169_w
    51877672/51877672 [==============================] - 0s 0us/step
```

```
[ ]  1 #Freezing the weights of the pretrained model
     2 densenet.trainable = False
```

```
[ ]  1 densenet.summary()

     conv5_block20_1_relu (Activati  (None, 7, 7, 128)   0         ['conv5_block20_1_bn[0][0]']
     on)

     conv5_block20_2_conv (Conv2D)  (None, 7, 7, 32)    36864     ['conv5_block20_1_relu[0][0]']

     conv5_block20_concat (Concaten  (None, 7, 7, 1280)  0         ['conv5_block19_concat[0][0]',
     ate)                                                           'conv5_block20_2_conv[0][0]']

     conv5_block21_0_bn (BatchNorma  (None, 7, 7, 1280)  5120      ['conv5_block20_concat[0][0]']
     lization)

     conv5_block21_0_relu (Activati  (None, 7, 7, 1280)  0         ['conv5_block21_0_bn[0][0]']
     on)
```

DenseNet169 Architecture specs



Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Binary Cross entropy loss and Adam optimizer for DenseNet169

```
#Compiling the model using adam optimizer
model.compile(loss=tf.keras.losses.binary_crossentropy,
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])
```

Encode Categorical labels (Positive/Negative) as numerical values 1 and 0

```python
from sklearn import preprocessing
# prepare labels so that positive and negative map to 0 and 1
def prepare_targets(y_train, y_test):
 le = preprocessing.LabelEncoder()
 le.fit(y_train)
 y_train_enc = le.transform(y_train)
 y_test_enc = le.transform(y_test)
 return y_train_enc, y_test_enc
```

```python
def decode_targets(y_train, prep_label):
 le = preprocessing.LabelEncoder()
 le.fit(y_train)
 label = le.inverse_transform(prep_label)
 #y_test_enc = le.transform(y_test)
 return label
```

```python
y_train_enc,y_test_enc = prepare_targets(y_train,y_test)
```

Train denseNet169 on 8379 training images with 5 epochs - accuracy of ~83%

+ Code + Text

```
[ ]
epochs = 5
print('-TRAINING----------------------------')
print('Input shape:', x_train.shape)
print('Number of training images: ', x_train.shape[0])

model.fit(x=x_train, y=y_train_enc, epochs=epochs)
```

```
-TRAINING----------------------------
Input shape: (8379, 224, 224, 3)
Number of training images:  8379
Epoch 1/5
262/262 [==============================] - 1574s 6s/step - loss: 1.1167 - accuracy: 0.6484
Epoch 2/5
262/262 [==============================] - 1555s 6s/step - loss: 0.8418 - accuracy: 0.7334
Epoch 3/5
262/262 [==============================] - 1550s 6s/step - loss: 0.6444 - accuracy: 0.7894
Epoch 4/5
262/262 [==============================] - 1502s 6s/step - loss: 0.4345 - accuracy: 0.8417
Epoch 5/5
262/262 [==============================] - 1518s 6s/step - loss: 0.4868 - accuracy: 0.8385
<keras.callbacks.History at 0x7f38226362e0>
```

Evaluate on testing images accuracy ~67%

```
print('-TESTING-----------------------------')
print('Number of test images:', x_test.shape[0])
score = model.evaluate(x_test, y_test_enc)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

# Print 10 example test digits with their true and predicted labels
fig, axes = plt.subplots(2, 5)
fig.tight_layout(rect=(0,0,3,3))

image_idx = np.random.randint(1,562,(2,5))

for i, j in it.product(range(2), range(5)):
    test_image = x_test[image_idx[i,j]].reshape(1, 224, 224, 3)
    test_label = y_test[image_idx[i,j]]
    sigmoid_outputs = model.predict(test_image)
    pred_label = sigmoid_outputs.argmax()
    pred_label = decode_targets(y_train,[pred_label])

    axes[i, j].imshow(test_image.reshape(224, 224,3),cmap='Greys')
    axes[i, j].set_aspect('equal', 'box')
    axes[i, j].set_title("actual: {} predicted: {}".format(test_label,pred_label[0]))

plt.show()
```

Evaluate on testing images accuracy ~67% 10 example images

```
-TESTING------------------------------
Number of test images: 563
18/18 [==============================] - 107s 6s/step - loss: 1.3024 - accuracy: 0.6785
Test loss: 1.3024436235427856
Test accuracy: 0.6785079836845398
1/1 [==============================] - 3s 3s/step
1/1 [==============================] - 0s 289ms/step
1/1 [==============================] - 0s 272ms/step
1/1 [==============================] - 0s 280ms/step
1/1 [==============================] - 0s 279ms/step
1/1 [==============================] - 0s 265ms/step
1/1 [==============================] - 0s 268ms/step
1/1 [==============================] - 0s 259ms/step
1/1 [==============================] - 0s 259ms/step
1/1 [==============================] - 0s 257ms/step
```