BMI260 2019 Problem Set 3

May 27, 2020

1 BIOMEDIN 260/RAD260: Problem Set 3 - Mammogram Project

1.1 Spring 2020

1.2 Name 1:

Kristen Anderson

1.3 Name 2:

Ryan Crowley

1.4 Introduction

Breast cancer has the highest incidence and second highest mortality rate for women in the US.

Your task is to utilize machine learning to study mammograms in any way you want (e.g. classification, segmentation) as long as you justify why it is useful to do whatever it is you want to do. Turning in a deep dream assignment using mammograms might be amusing, for example, but not so useful to patients. That being said, choose something that interests you. As the adage goes, "do what you love, and you'll never have to work another day in your life, at least in BMI 260."

Treat this as a mini-project. We highly encourage working with 1 other person, possibly someone in your main project team.

In addition to the mammograms themselves, the dataset includes "ground-truth" segmentations and `mass_case_description_train_set.csv`, which contains metadata information about mass shapes, mass margins, assessment numbers, pathology diagnoses, and subtlety in the data. Take some time to research what all of these different fields mean and how you might utilize them in your work. You dont need to use all of what is provided to you.

Some ideas:

1. Use the ROI's or segmentations to extract features, and then train a classifier based on those features using the algorithms presented to you in the machine learning lectures (doesn't need to use deep learning).

2. Use convolutional neural networks. Feel free to use any of the code we went over in class or use your own (custom code, sklearn, keras, Tensorflow etc.). If you dont want to place helper functions and classes into this notebook, place them in a `.py` file in the same folder called `helperfunctions.py` and import them into this notebook.

1.5 Data

The data is here:

https://wiki.cancerimagingarchive.net/display/Public/CBIS-DDSM

1.6 Grading and Submission

This assignment has 3 components: code, figures (outputs/analyses of your code), and a write-up detailing your mini-project. You will be graded on these categories.

If you're OK with Python or R, please place all three parts into this notebook/.Rmd file that we have provided where indicated. We have written template sections for you to follow for simplicity/completeness. When you're done, save as a `.pdf` (please knit to `.pdf` if you are using `.Rmd`, or knit to `.html` and use a browser's "Print" function to convert to `.pdf`).

If you don't like Python OR R, we will allow you to use a different language, but please turn your assignment in with: 1) a folder with all your code, 2) a folder with all your figures, and 3) a `.tex`/`.doc`/`.pdf` file with a write-up.

1.7 Differentiating between Malignant and Benign Masses in Mammograms using Transfer Learning

1. Describe what you are doing and why it matters to patients using at least one citation.

We will use the VGG-16 CNN to diagnose benign and malignant tumors in mammograms. In doing so, we could help doctors diagnose cancer at earlier stages through discovering malignant tumors that may otherwise be missed. CNNs have been shown to classify mammogram images with an overall sensitivity of 96% and a 0.99 AUC [1]. In contrast, doctors acurately diagnose breast cancer through mammography in about 78% of their patients, with diagnostic accuracy rising to about 83% when female patients are older than 50 [2].

The difference between these scores suggests both a significant need for technological aid as well as the CNN's capacity to effectively improve diagnosis. The chance that a woman will die from breast cancer is about 1 in 38, so reducing this risk could save many women's lives [3]. Early diagnosis gives women the opportunity to receive proper medical care when fighting cancer.

1. https://pubmed.ncbi.nlm.nih.gov/31838610/
2. https://www.uchealth.org/today/how-accurate-are-mammograms/
3. https://www.cancer.org/cancer/breast-cancer/about/how-common-is-breast-cancer.html

2. Describe the relevant statistics of the data. How were the images taken? How were they labeled? What is the class balance and majority classifier accuracy? How will you divide the data into testing, training and validation sets?

The images are taken using low-dose X-ray mammography as the imgaging technique. These images were collected from a variety of sources including Massachusetts General Hopsital and Washington University of St. Louis School of Medicine. The images are labeled with diverse information including ROIs for calcifications and masses as well as BI-RADS information about the mass shape, mass margin, breast density, and other data. There also exists other metadata that include the patient age, date of study, scanner used to digitize, and resolution of the image along other data. Cases with abnormalities also contain extra information including the type of abnormality. We will be focusing upon the masses. There is a training set of 653 benign abnormalities and 621 malignant abnormalities. There also exists a testing set with 189 benign abnormalities and 156 malignant abnormalities. As such, the majority classifer accuracy on the test set is .5478. We will keep the groups sorted as given and will not perform any hyperparameter tuning so a validation set will not be neccessary.

3. Describe your data pipeline (how is the data scrubbed, normalized, stored, and fed to the model for training?).

The data pipeline that we developed first used the csv files to link individual patient ID's with the associated diagnosis of whether the mass was benign or malignant. We then looped through all of the individual ROI images in the "data_fixed_crop_w_mask" folder to assign them into either the training malignant folder or the training benign folder. We also cut the images down in size to 224 x 224 pixels as this is the required input for our version of VGG-16. We chose to cut the images down by taking the center 224 x 224 pixels and cutting off the border around each image. We then performed the same data preprocessing with all of the individual ROI images in the "test" folder to assign them either to the test malignant folder or the test benign folder and cut them down in size as well to 224 x 224 pixels.

4. Explain how the model you chose works alongside the code for it. Add at least one technical citation to give credit where credit is due.

[21]:
```
import os
import numpy as np
import pydicom as dicom
import matplotlib.pyplot as plt
import os
import skimage
import h5py
import csv
%matplotlib inline
import keras
import PIL
from PIL import Image
from keras.models import Model
from keras.layers import Dense
from keras import optimizers
from keras.preprocessing.image import ImageDataGenerator
```

```
from keras.preprocessing import image
from keras.applications.vgg16 import VGG16
from keras.callbacks import ModelCheckpoint, EarlyStopping
```

[22]:
```
#Create a dictionary that links patient id to outcome
dictionary = {};
with open('mass_case_description_train_set-1.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        key = row[0]

        dictionary[key] = row[9]
with open('mass_case_description_test_set-1.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        key = row[0]
        dictionary[key] = row[9]
```

[26]:
```
#Create the training data set
directory = "data_fixed_crop_w_mask"
for folder in os.listdir(directory):
    path = directory + "/" + folder;
    sum = 0
    for filename in os.listdir(path):
        sum += 1
        fullFilePath = directory + "/" + folder + "/" + filename;
        f = h5py.File(fullFilePath, "r")
        datasetNames = [n for n in f.keys()]
        a_group = f['data']
        second = a_group[:,:,1]
        finalImage = second[16:240,16:240]

        patient_ID =  "P_" + folder;
        if patient_ID in dictionary:
            for row in range(0,224):
                for col in range(0,224):
                    if finalImage[row,col] < .5:
                        finalImage[row,col] = 0
                    else:
                        finalImage[row,col] = 255
            im = Image.fromarray(finalImage)
            im = im.convert("L")

            if dictionary[patient_ID] == "MALIGNANT":
                im.save('trainSorted/malignant/'+patient_ID+'_'+str(sum)+'.
↪png', 'png')
            else:
```

```
                im.save('trainSorted/benign/'+patient_ID+'_'+str(sum)+'.png',␣
↪'png')
```

[27]:
```
#Create the testing dataset
directory = "test"
for folder in os.listdir(directory):
    path = directory + "/" + folder;
    if path != 'test/.DS_Store':
        sum = 0
        for filename in os.listdir(path):
            sum += 1
            fullFilePath = directory + "/" + folder + "/" + filename;
            f = h5py.File(fullFilePath, "r")
            datasetNames = [n for n in f.keys()]
            a_group = f['data']
            second = a_group[:,:,1]
            finalImage = second[16:240,16:240]

            patient_ID =  "P_" + folder;
            if patient_ID in dictionary:
                for row in range(0,224):
                    for col in range(0,224):
                        if finalImage[row,col] < .5:
                            finalImage[row,col] = 0
                        else:
                            finalImage[row,col] = 255
                im = Image.fromarray(finalImage)
                im = im.convert("L")

                if dictionary[patient_ID] == "MALIGNANT":
                    im.save('testSorted/malignant/'+patient_ID+'_'+str(sum)+'.
↪png', 'png')
                else:
                    im.save('testSorted/benign/'+patient_ID+'_'+str(sum)+'.
↪png', 'png')
```

[28]:
```
#Load the data so that it can be used in the model
trdata = ImageDataGenerator()
traindata = trdata.flow_from_directory(directory="/Users/ryancrowley/Desktop/
↪BMI 260/trainSorted/",target_size=(224,224))
tsdata = ImageDataGenerator()
testdata = tsdata.flow_from_directory(directory="/Users/ryancrowley/Desktop/BMI␣
↪260/testSorted/", target_size=(224,224))
```

```
Found 1274 images belonging to 2 classes.
Found 345 images belonging to 2 classes.
```

[29]:
```
#Edit the model so that it works for this application
vggmodel = VGG16(weights='imagenet', include_top=True)
for layers in (vggmodel.layers)[:19]:
    layers.trainable = False
X= vggmodel.layers[-2].output
predictions = Dense(2, activation="softmax")(X)
model_final = Model(input = vggmodel.input, output = predictions)
model_final.compile(loss = "categorical_crossentropy", optimizer = optimizers.
 ↪SGD(lr=0.0001, momentum=0.9), metrics=["accuracy"])
model_final.summary()
```

```
WARNING:tensorflow:From
/Users/ryancrowley/opt/anaconda3/envs/bmi2602/lib/python3.6/site-
packages/tensorflow/python/ops/resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
Model: "model_1"
_________________________________________________________________
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
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 8194
=================================================================
Total params: 134,268,738
Trainable params: 119,554,050
Non-trainable params: 14,714,688
_________________________________________________________________
/Users/ryancrowley/opt/anaconda3/envs/bmi2602/lib/python3.6/site-
packages/ipykernel_launcher.py:6: UserWarning: Update your `Model` call to the
Keras 2 API: `Model(inputs=Tensor("in…, outputs=Tensor("de…)`
```

[30]:
```
#Run the training and validation of the model
checkpoint = ModelCheckpoint("vgg16_1.h5", monitor='val_acc', verbose=1,␣
 ↪save_best_only=True, save_weights_only=False, mode='auto', period=1)
early = EarlyStopping(monitor='val_acc', min_delta=0, patience=40, verbose=1,␣
 ↪mode='auto')
model_final.fit_generator(generator= traindata, steps_per_epoch= 2, epochs=␣
 ↪100, validation_data= testdata, validation_steps=1,␣
 ↪callbacks=[checkpoint,early])
model_final.save_weights("vgg16_1.h5")
```

```
WARNING:tensorflow:From
/Users/ryancrowley/opt/anaconda3/envs/bmi2602/lib/python3.6/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
```

```
Epoch 1/100
2/2 [==============================] - 58s 29s/step - loss: 0.9766 - accuracy:
0.5469 - val_loss: 1.1089 - val_accuracy: 0.3750
Epoch 2/100

/Users/ryancrowley/opt/anaconda3/envs/bmi2602/lib/python3.6/site-
packages/keras/callbacks/callbacks.py:707: RuntimeWarning: Can save best model
only with val_acc available, skipping.
  'skipping.' % (self.monitor), RuntimeWarning)
/Users/ryancrowley/opt/anaconda3/envs/bmi2602/lib/python3.6/site-
packages/keras/callbacks/callbacks.py:846: RuntimeWarning: Early stopping
conditioned on metric `val_acc` which is not available. Available metrics are:
val_loss,val_accuracy,loss,accuracy
  (self.monitor, ','.join(list(logs.keys()))), RuntimeWarning

2/2 [==============================] - 46s 23s/step - loss: 1.1332 - accuracy:
0.5156 - val_loss: 0.6396 - val_accuracy: 0.5625
Epoch 3/100
2/2 [==============================] - 45s 23s/step - loss: 1.0413 - accuracy:
0.5312 - val_loss: 0.9555 - val_accuracy: 0.6250
Epoch 4/100
2/2 [==============================] - 48s 24s/step - loss: 1.0534 - accuracy:
0.5000 - val_loss: 1.0931 - val_accuracy: 0.4375
Epoch 5/100
2/2 [==============================] - 48s 24s/step - loss: 1.1376 - accuracy:
0.5156 - val_loss: 0.8782 - val_accuracy: 0.6250
Epoch 6/100
2/2 [==============================] - 57s 29s/step - loss: 1.0228 - accuracy:
0.5625 - val_loss: 1.3707 - val_accuracy: 0.4688
Epoch 7/100
2/2 [==============================] - 52s 26s/step - loss: 1.1867 - accuracy:
0.5156 - val_loss: 0.7215 - val_accuracy: 0.5000
Epoch 8/100
2/2 [==============================] - 54s 27s/step - loss: 0.9830 - accuracy:
0.4531 - val_loss: 1.1456 - val_accuracy: 0.3125
Epoch 9/100
2/2 [==============================] - 52s 26s/step - loss: 0.8931 - accuracy:
0.5000 - val_loss: 0.8169 - val_accuracy: 0.4688
Epoch 10/100
2/2 [==============================] - 50s 25s/step - loss: 0.9900 - accuracy:
0.5312 - val_loss: 0.6293 - val_accuracy: 0.7188
Epoch 11/100
2/2 [==============================] - 57s 28s/step - loss: 0.7348 - accuracy:
0.5312 - val_loss: 0.9188 - val_accuracy: 0.3200
Epoch 12/100
2/2 [==============================] - 56s 28s/step - loss: 0.8073 - accuracy:
0.5172 - val_loss: 0.6479 - val_accuracy: 0.6562
Epoch 13/100
2/2 [==============================] - 63s 31s/step - loss: 0.8000 - accuracy:
```

```
0.5469 - val_loss: 0.6431 - val_accuracy: 0.7188
Epoch 14/100
2/2 [==============================] - 62s 31s/step - loss: 0.7490 - accuracy:
0.4844 - val_loss: 0.5815 - val_accuracy: 0.6250
Epoch 15/100
2/2 [==============================] - 62s 31s/step - loss: 0.7521 - accuracy:
0.5781 - val_loss: 0.6333 - val_accuracy: 0.6875
Epoch 16/100
2/2 [==============================] - 62s 31s/step - loss: 0.8650 - accuracy:
0.5000 - val_loss: 0.7076 - val_accuracy: 0.5938
Epoch 17/100
2/2 [==============================] - 67s 33s/step - loss: 0.6864 - accuracy:
0.6094 - val_loss: 0.7003 - val_accuracy: 0.6250
Epoch 18/100
2/2 [==============================] - 65s 32s/step - loss: 0.8665 - accuracy:
0.4844 - val_loss: 0.7134 - val_accuracy: 0.5312
Epoch 19/100
2/2 [==============================] - 70s 35s/step - loss: 0.8577 - accuracy:
0.4688 - val_loss: 0.7529 - val_accuracy: 0.6250
Epoch 20/100
2/2 [==============================] - 66s 33s/step - loss: 0.8835 - accuracy:
0.4531 - val_loss: 0.6320 - val_accuracy: 0.7812
Epoch 21/100
2/2 [==============================] - 64s 32s/step - loss: 0.6822 - accuracy:
0.5781 - val_loss: 0.7100 - val_accuracy: 0.5938
Epoch 22/100
2/2 [==============================] - 56s 28s/step - loss: 0.7655 - accuracy:
0.5156 - val_loss: 1.0915 - val_accuracy: 0.4400
Epoch 23/100
2/2 [==============================] - 66s 33s/step - loss: 0.8529 - accuracy:
0.5781 - val_loss: 1.1303 - val_accuracy: 0.5312
Epoch 24/100
2/2 [==============================] - 63s 32s/step - loss: 0.7588 - accuracy:
0.5938 - val_loss: 0.8501 - val_accuracy: 0.5938
Epoch 25/100
2/2 [==============================] - 64s 32s/step - loss: 0.8278 - accuracy:
0.5156 - val_loss: 0.7272 - val_accuracy: 0.5312
Epoch 26/100
2/2 [==============================] - 63s 32s/step - loss: 0.6497 - accuracy:
0.6562 - val_loss: 0.9167 - val_accuracy: 0.5625
Epoch 27/100
2/2 [==============================] - 63s 32s/step - loss: 0.8112 - accuracy:
0.4688 - val_loss: 0.6698 - val_accuracy: 0.6250
Epoch 28/100
2/2 [==============================] - 59s 29s/step - loss: 0.7218 - accuracy:
0.5469 - val_loss: 0.7890 - val_accuracy: 0.4688
Epoch 29/100
2/2 [==============================] - 62s 31s/step - loss: 0.9085 - accuracy:
```

```
0.4688 - val_loss: 0.6969 - val_accuracy: 0.5938
Epoch 30/100
2/2 [==============================] - 61s 31s/step - loss: 0.8420 - accuracy:
0.5625 - val_loss: 0.6588 - val_accuracy: 0.6250
Epoch 31/100
2/2 [==============================] - 67s 34s/step - loss: 0.8283 - accuracy:
0.5469 - val_loss: 0.7104 - val_accuracy: 0.4688
Epoch 32/100
2/2 [==============================] - 83s 42s/step - loss: 0.7923 - accuracy:
0.5312 - val_loss: 0.7863 - val_accuracy: 0.5625
Epoch 33/100
2/2 [==============================] - 58s 29s/step - loss: 0.7008 - accuracy:
0.6406 - val_loss: 0.6828 - val_accuracy: 0.5600
Epoch 34/100
2/2 [==============================] - 63s 32s/step - loss: 0.8157 - accuracy:
0.5781 - val_loss: 0.5340 - val_accuracy: 0.7812
Epoch 35/100
2/2 [==============================] - 60s 30s/step - loss: 0.9292 - accuracy:
0.5625 - val_loss: 0.6491 - val_accuracy: 0.5938
Epoch 36/100
2/2 [==============================] - 64s 32s/step - loss: 0.7527 - accuracy:
0.4844 - val_loss: 1.0024 - val_accuracy: 0.4062
Epoch 37/100
2/2 [==============================] - 64s 32s/step - loss: 0.7818 - accuracy:
0.5312 - val_loss: 0.6312 - val_accuracy: 0.6562
Epoch 38/100
2/2 [==============================] - 62s 31s/step - loss: 0.7321 - accuracy:
0.5312 - val_loss: 0.6390 - val_accuracy: 0.6562
Epoch 39/100
2/2 [==============================] - 57s 29s/step - loss: 0.6417 - accuracy:
0.5862 - val_loss: 0.6466 - val_accuracy: 0.5938
Epoch 40/100
2/2 [==============================] - 57s 28s/step - loss: 0.6580 - accuracy:
0.6406 - val_loss: 0.4824 - val_accuracy: 0.8125
Epoch 41/100
2/2 [==============================] - 65s 32s/step - loss: 0.6681 - accuracy:
0.5312 - val_loss: 0.4995 - val_accuracy: 0.6875
Epoch 42/100
2/2 [==============================] - 63s 31s/step - loss: 0.6274 - accuracy:
0.6719 - val_loss: 0.8575 - val_accuracy: 0.6250
Epoch 43/100
2/2 [==============================] - 59s 30s/step - loss: 0.6581 - accuracy:
0.6562 - val_loss: 1.0343 - val_accuracy: 0.5625
Epoch 44/100
2/2 [==============================] - 55s 28s/step - loss: 0.6803 - accuracy:
0.6406 - val_loss: 0.6494 - val_accuracy: 0.7200
Epoch 45/100
2/2 [==============================] - 60s 30s/step - loss: 0.6053 - accuracy:
```

```
0.6406 - val_loss: 0.7640 - val_accuracy: 0.5938
Epoch 46/100
2/2 [==============================] - 70s 35s/step - loss: 0.7167 - accuracy:
0.5781 - val_loss: 0.6595 - val_accuracy: 0.5938
Epoch 47/100
2/2 [==============================] - 62s 31s/step - loss: 0.6118 - accuracy:
0.6875 - val_loss: 0.9666 - val_accuracy: 0.5625
Epoch 48/100
2/2 [==============================] - 52s 26s/step - loss: 0.8231 - accuracy:
0.5312 - val_loss: 0.7211 - val_accuracy: 0.5625
Epoch 49/100
2/2 [==============================] - 51s 25s/step - loss: 0.6488 - accuracy:
0.6250 - val_loss: 0.6212 - val_accuracy: 0.5938
Epoch 50/100
2/2 [==============================] - 54s 27s/step - loss: 0.6494 - accuracy:
0.6406 - val_loss: 0.5563 - val_accuracy: 0.7500
Epoch 51/100
2/2 [==============================] - 55s 27s/step - loss: 0.5626 - accuracy:
0.7031 - val_loss: 0.4763 - val_accuracy: 0.8750
Epoch 52/100
2/2 [==============================] - 51s 25s/step - loss: 0.6735 - accuracy:
0.5690 - val_loss: 0.6543 - val_accuracy: 0.6250
Epoch 53/100
2/2 [==============================] - 61s 31s/step - loss: 0.6192 - accuracy:
0.6250 - val_loss: 0.6946 - val_accuracy: 0.5625
Epoch 54/100
2/2 [==============================] - 61s 30s/step - loss: 0.7069 - accuracy:
0.5469 - val_loss: 0.6040 - val_accuracy: 0.6875
Epoch 55/100
2/2 [==============================] - 63s 32s/step - loss: 0.5583 - accuracy:
0.7656 - val_loss: 0.4797 - val_accuracy: 0.7200
Epoch 56/100
2/2 [==============================] - 56s 28s/step - loss: 0.9033 - accuracy:
0.4688 - val_loss: 0.6656 - val_accuracy: 0.5938
Epoch 57/100
2/2 [==============================] - 54s 27s/step - loss: 0.6271 - accuracy:
0.6562 - val_loss: 0.7704 - val_accuracy: 0.5312
Epoch 58/100
2/2 [==============================] - 56s 28s/step - loss: 0.6380 - accuracy:
0.6094 - val_loss: 0.5655 - val_accuracy: 0.6875
Epoch 59/100
2/2 [==============================] - 57s 28s/step - loss: 0.6500 - accuracy:
0.5781 - val_loss: 0.7270 - val_accuracy: 0.5625
Epoch 60/100
2/2 [==============================] - 54s 27s/step - loss: 0.6955 - accuracy:
0.5781 - val_loss: 0.6462 - val_accuracy: 0.6562
Epoch 61/100
2/2 [==============================] - 57s 28s/step - loss: 0.5963 - accuracy:
```

```
0.6562 - val_loss: 0.5686 - val_accuracy: 0.8438
Epoch 62/100
2/2 [==============================] - 57s 29s/step - loss: 0.6022 - accuracy:
0.6406 - val_loss: 0.6336 - val_accuracy: 0.5312
Epoch 63/100
2/2 [==============================] - 52s 26s/step - loss: 0.6829 - accuracy:
0.5469 - val_loss: 0.5114 - val_accuracy: 0.7500
Epoch 64/100
2/2 [==============================] - 50s 25s/step - loss: 0.5477 - accuracy:
0.7188 - val_loss: 0.6305 - val_accuracy: 0.6875
Epoch 65/100
2/2 [==============================] - 42s 21s/step - loss: 0.6170 - accuracy:
0.6250 - val_loss: 0.8560 - val_accuracy: 0.5938
Epoch 66/100
2/2 [==============================] - 43s 22s/step - loss: 0.5890 - accuracy:
0.6562 - val_loss: 0.6577 - val_accuracy: 0.6400
Epoch 67/100
2/2 [==============================] - 44s 22s/step - loss: 0.6554 - accuracy:
0.6094 - val_loss: 0.5147 - val_accuracy: 0.6562
Epoch 68/100
2/2 [==============================] - 43s 21s/step - loss: 0.6523 - accuracy:
0.7031 - val_loss: 0.5319 - val_accuracy: 0.6562
Epoch 69/100
2/2 [==============================] - 44s 22s/step - loss: 0.6854 - accuracy:
0.5938 - val_loss: 0.6693 - val_accuracy: 0.6562
Epoch 70/100
2/2 [==============================] - 46s 23s/step - loss: 0.7482 - accuracy:
0.4844 - val_loss: 0.6290 - val_accuracy: 0.6250
Epoch 71/100
2/2 [==============================] - 42s 21s/step - loss: 0.6402 - accuracy:
0.5938 - val_loss: 0.6544 - val_accuracy: 0.5938
Epoch 72/100
2/2 [==============================] - 44s 22s/step - loss: 0.7904 - accuracy:
0.5000 - val_loss: 0.6496 - val_accuracy: 0.5938
Epoch 73/100
2/2 [==============================] - 44s 22s/step - loss: 0.6489 - accuracy:
0.6875 - val_loss: 0.8400 - val_accuracy: 0.6250
Epoch 74/100
2/2 [==============================] - 41s 20s/step - loss: 0.7659 - accuracy:
0.5862 - val_loss: 0.6970 - val_accuracy: 0.6250
Epoch 75/100
2/2 [==============================] - 43s 21s/step - loss: 0.7145 - accuracy:
0.6094 - val_loss: 0.5657 - val_accuracy: 0.6875
Epoch 76/100
2/2 [==============================] - 41s 20s/step - loss: 0.6240 - accuracy:
0.6562 - val_loss: 0.6535 - val_accuracy: 0.7188
Epoch 77/100
2/2 [==============================] - 38s 19s/step - loss: 0.6581 - accuracy:
```

```
0.5938 - val_loss: 0.8955 - val_accuracy: 0.5200
Epoch 78/100
2/2 [==============================] - 40s 20s/step - loss: 0.5118 - accuracy:
0.7656 - val_loss: 0.5952 - val_accuracy: 0.6875
Epoch 79/100
2/2 [==============================] - 45s 22s/step - loss: 0.6832 - accuracy:
0.5938 - val_loss: 0.7398 - val_accuracy: 0.5312
Epoch 80/100
2/2 [==============================] - 45s 23s/step - loss: 0.5510 - accuracy:
0.8281 - val_loss: 0.4768 - val_accuracy: 0.7812
Epoch 81/100
2/2 [==============================] - 39s 19s/step - loss: 0.6016 - accuracy:
0.6250 - val_loss: 0.5220 - val_accuracy: 0.7188
Epoch 82/100
2/2 [==============================] - 50s 25s/step - loss: 0.4890 - accuracy:
0.8281 - val_loss: 0.4631 - val_accuracy: 0.7188
Epoch 83/100
2/2 [==============================] - 45s 23s/step - loss: 0.4926 - accuracy:
0.8125 - val_loss: 0.8964 - val_accuracy: 0.5625
Epoch 84/100
2/2 [==============================] - 47s 23s/step - loss: 0.4770 - accuracy:
0.7969 - val_loss: 0.9050 - val_accuracy: 0.5938
Epoch 85/100
2/2 [==============================] - 44s 22s/step - loss: 0.5302 - accuracy:
0.7031 - val_loss: 0.6994 - val_accuracy: 0.7188
Epoch 86/100
2/2 [==============================] - 46s 23s/step - loss: 0.5929 - accuracy:
0.6562 - val_loss: 0.5831 - val_accuracy: 0.7812
Epoch 87/100
2/2 [==============================] - 51s 26s/step - loss: 0.6179 - accuracy:
0.5938 - val_loss: 0.5834 - val_accuracy: 0.6875
Epoch 88/100
2/2 [==============================] - 44s 22s/step - loss: 0.5781 - accuracy:
0.6875 - val_loss: 0.5542 - val_accuracy: 0.7200
Epoch 89/100
2/2 [==============================] - 46s 23s/step - loss: 0.6098 - accuracy:
0.6562 - val_loss: 0.6192 - val_accuracy: 0.6875
Epoch 90/100
2/2 [==============================] - 45s 23s/step - loss: 0.5260 - accuracy:
0.7344 - val_loss: 0.5900 - val_accuracy: 0.6875
Epoch 91/100
2/2 [==============================] - 39s 19s/step - loss: 0.5307 - accuracy:
0.7241 - val_loss: 0.5826 - val_accuracy: 0.6875
Epoch 92/100
2/2 [==============================] - 42s 21s/step - loss: 0.5755 - accuracy:
0.6719 - val_loss: 0.4501 - val_accuracy: 0.8125
Epoch 93/100
2/2 [==============================] - 46s 23s/step - loss: 0.5424 - accuracy:
```

```
0.7188 - val_loss: 0.4050 - val_accuracy: 0.8438
Epoch 94/100
2/2 [==============================] - 48s 24s/step - loss: 0.5647 - accuracy:
0.7188 - val_loss: 0.7907 - val_accuracy: 0.6875
Epoch 95/100
2/2 [==============================] - 50s 25s/step - loss: 0.6025 - accuracy:
0.6250 - val_loss: 0.6001 - val_accuracy: 0.6875
Epoch 96/100
2/2 [==============================] - 48s 24s/step - loss: 0.5679 - accuracy:
0.7031 - val_loss: 0.6046 - val_accuracy: 0.6562
Epoch 97/100
2/2 [==============================] - 44s 22s/step - loss: 0.5328 - accuracy:
0.7031 - val_loss: 0.5363 - val_accuracy: 0.7188
Epoch 98/100
2/2 [==============================] - 47s 23s/step - loss: 0.6346 - accuracy:
0.6406 - val_loss: 0.6785 - val_accuracy: 0.5625
Epoch 99/100
2/2 [==============================] - 43s 22s/step - loss: 0.6390 - accuracy:
0.6562 - val_loss: 0.6503 - val_accuracy: 0.7200
Epoch 100/100
2/2 [==============================] - 46s 23s/step - loss: 0.5998 - accuracy:
0.6406 - val_loss: 0.5149 - val_accuracy: 0.7500
```

After the data preprocessing steps, we then loaded the data into the correct format by using ImageDataGenerator. From here, we then imported the VGG-16 model that was pretrained on the ImageNet dataset. We then altered the structure of the network such that it only outputs malignant or benign instead of the 1000 different categories it would output in the past. We also altered the network such that the first 19 layers were frozen as part of the transfer learning approach. This approach is overall similar to a transfer learning approach using previously to differentiate between COVID-19 and viral pneumonia with a great deal of success [1].

1.https://arxiv.org/abs/2003.13145

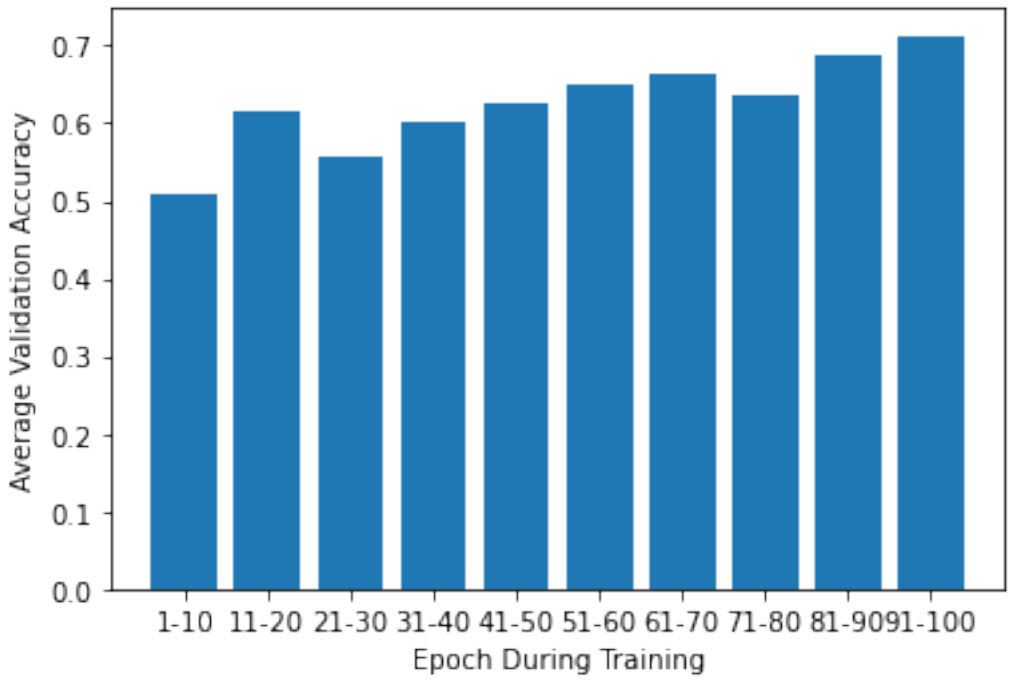
5. There are many ways to do training. Take us through how you do it (e.g. "We used early stopping and stopped when validation loss increased twice in a row.").

We used an early stopping criteria where we would stop if the validation accuracy did not change over the course of two consecutive epochs. Additionally, we trained using a total of 100 epochs with 2 steps per epoch. Our learning rate was eta = 0.0001, and we used Stochastic Gradient Descent with momentum = .9

6. Make a figure displaying your results.

[32]:
```
height = [.50939,.61637,.55651,.60192,.62513,.65012,.66319,.63638,.68764,.71263]
bars = ('1-10', '11-20', '21-30', '31-40',␣
 ↪'41-50','51-60','61-70','71-80','81-90','91-100')
y_pos = np.arange(len(bars))

# Create bars
plt.bar(y_pos, height)
```

```
# Create names on the x-axis
plt.xticks(y_pos, bars)

plt.xlabel("Epoch During Training")

plt.ylabel("Average Validation Accuracy")

# Show graphic
plt.show()
```



Here, we asssessed the average validation accuracy over 10 different intervals during the training process. Overall, we found that the validation accuracy appeared to increase over the course of the training indicating that the model was working as intended. Of note, the model outperforms the majority classifier accuracy of .5478

7. Discuss pros and cons of your method and what you might have done differently now that you've tried or would try if you had more time.

Of note, this method achieved a moderate level of success with our final 10 iterations of the model averaging a validation accuracy of .71263 Our model appeared to show some success in differentiating between benign and malignant tumors. This is likely largely due to the fact that CNNs are powerful for image analysis and transfer learning can be an effective method for quickly and efficiently training a model. Some possible limitations of our approach included our relatively small amount of data, the fact that no textural features could be calculated due to the fact that our input images were segmentations, and a relatively short training period. Additionally, accuracy

appeared to fluctuate significantly indicating that the model was quite variable. Finally, the model that we developed may not easily fit into clinical workflows due to the need to determine the boundaries of the ROI. If we had had more time and more computational power, we could have tried to use more images to train the dataset and trained the model over the course of more epochs. Of course, there would then exist the possibility of overfitting so if we chose to train for more epochs we would also include additional methods to regularize the model. Overall, our approach shows some promise but also leaves significant room for improvement.

You will not be graded on the performance of your model. You'll only be graded on the scientific soundness of your claims, methodology, evaluation (i.e. fair but insightful statistics), and discussion of the strengths and shortcomings of what you tried. Feel free to reuse some of the code you are/will be using for your projects. The write-up doesn't need to be long (~1 page will suffice), but please cite at least one clinical paper and one technical paper (1 each in questions 1 and 4 at least, and more if needed).

[ ]: