Project 4

# LUNG CANCER VS. COVID-19

Douglas Packard, Chris Rebello, Karina Alvarez

## TABLE OF CONTENTS

## INTRODUCTION

This project applies machine learning techniques to data from the Cancer Imaging Archive. The goal is to produce a script that creates, trains, and tests a machine learning model on medical imaging data from CT scans of the lung, *predicting* the classification of either lung cancer or COVID-19.

01

## DATA SOURCES

## DATA SOURCES

### The Cancer Imaging Archive (TCIA)

### COVID-19 Data

Dataset consists of CT images of patients who tested positive for COVID-19.

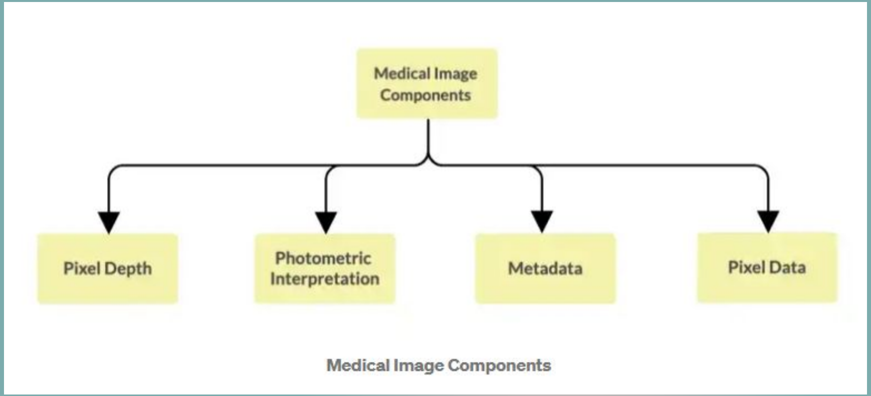
### Lung Cancer Data

Dataset consists of CT images of lung cancer subjects.

# DATA FORMAT

The CT scans/medical imaging was presented in DICOM (Digital Imaging and Communications in Medicine) format. DICOM format contains four components:

- Pixel Depth
- Photometric Interpretation
- Metadata
- Pixel Data



# Machine Learning Models

## VGG19

A  convolutional neural network that is 19 layers deep. This pretrained network has the capacity to classify images into 1000 object categories. Sizing 224 by 224.

## ResNet-50

A  convolutional neural network that is 50 layers deep. The pretrained network can classify images into 1000 object categories. Sizing 224 by 224.

## InceptionV3

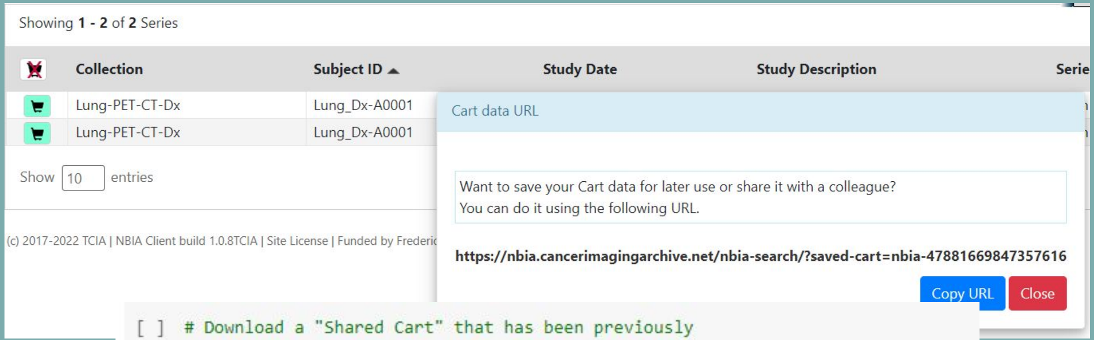
A  convolutional neural network that is 48 layers deep. The pretrained network can classify images into 1000 object categories. Sizing 299 by 299.

02

## Data Preparation and Preprocessing

# EXTRACTING THE DATA

From TCIA (The Cancer Imaging Archive), the Radiology Portal allows retrieval of the datasets by creating shared "carts" for identification. We then accessed the data by using the cart id in our script along with TCIA's Python utilities package.



Showing **1 - 2** of **2** Series

| | Collection | Subject ID ▲ | Study Date | Study Description | Serie |
|---|---|---|---|---|---|
| 🛒 | Lung-PET-CT-Dx | Lung_Dx-A0001 | | | |
| 🛒 | Lung-PET-CT-Dx | Lung_Dx-A0001 | | | |

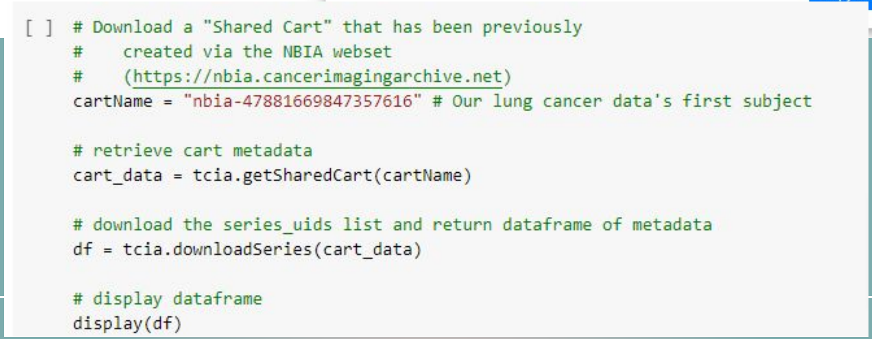
Show 10 entries

(c) 2017-2022 TCIA | NBIA Client build 1.0.8TCIA | Site License | Funded by Frederic

Cart data URL

Want to save your Cart data for later use or share it with a colleague?
You can do it using the following URL.

**https://nbia.cancerimagingarchive.net/nbia-search/?saved-cart=nbia-47881669847357616**

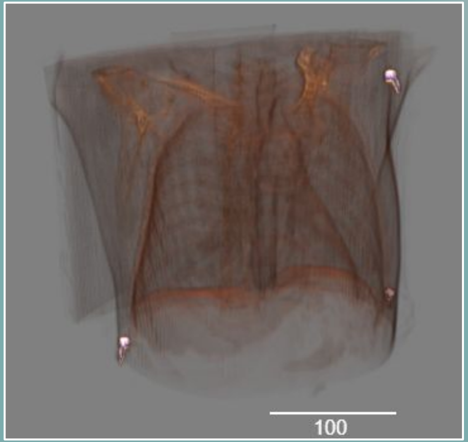


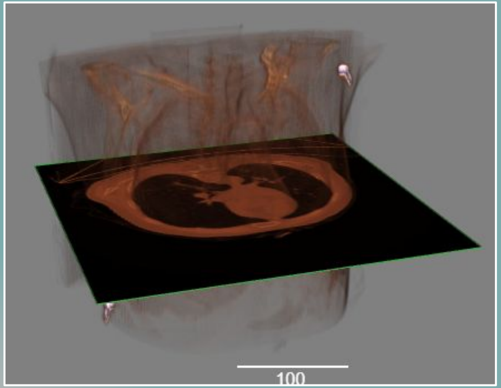
```
[ ]  # Download a "Shared Cart" that has been previously
     #    created via the NBIA webset
     #    (https://nbia.cancerimagingarchive.net)
     cartName = "nbia-47881669847357616" # Our lung cancer data's first subject

     # retrieve cart metadata
     cart_data = tcia.getSharedCart(cartName)

     # download the series_uids list and return dataframe of metadata
     df = tcia.downloadSeries(cart_data)

     # display dataframe
     display(df)
```

## EXTRACTING THE DATA

To retrieve, view, and visualize the data we used:

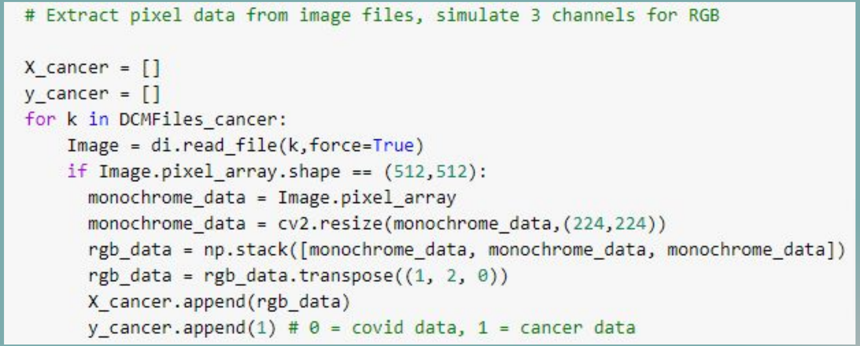
- Insight Toolkit (ITK) – an extensive suite of software tools for image visualization and analysis
- Pydicom – python package to read complex files into natural pythonic structures for easy manipulation



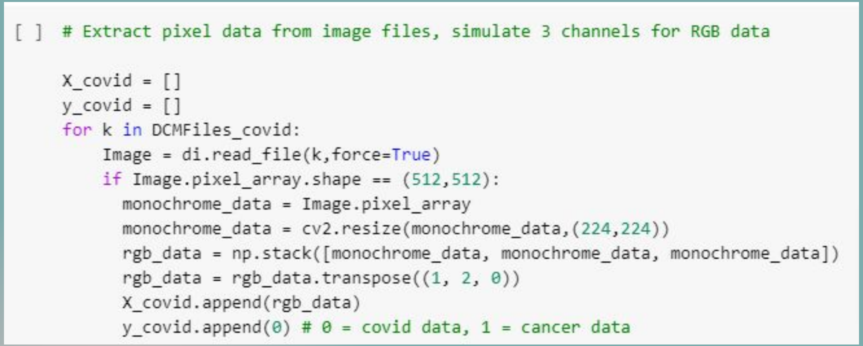


# TRANSFORMING THE DATA

- Resizing for the cancer and COVID-19 data was necessary to fit each model
- Troubleshoot: The CT scans were monochrome grayscale and the pretrained models expected three channel RGB scale
  - Used np.stack to duplicate three times as if it were three channel RGB
- We appended each 3d pixel array to a created list of target labels for the data (0 = covid data, 1 = cancer data)



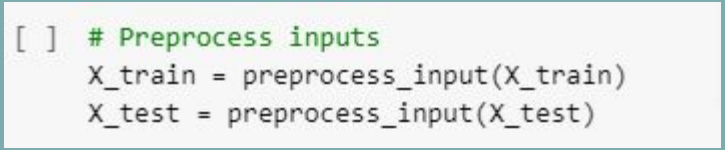
```
# Extract pixel data from image files, simulate 3 channels for RGB

X_cancer = []
y_cancer = []
for k in DCMFiles_cancer:
    Image = di.read_file(k,force=True)
    if Image.pixel_array.shape == (512,512):
      monochrome_data = Image.pixel_array
      monochrome_data = cv2.resize(monochrome_data,(224,224))
      rgb_data = np.stack([monochrome_data, monochrome_data, monochrome_data])
      rgb_data = rgb_data.transpose((1, 2, 0))
      X_cancer.append(rgb_data)
      y_cancer.append(1) # 0 = covid data, 1 = cancer data
```



```
[ ]  # Extract pixel data from image files, simulate 3 channels for RGB data

     X_covid = []
     y_covid = []
     for k in DCMFiles_covid:
         Image = di.read_file(k,force=True)
         if Image.pixel_array.shape == (512,512):
           monochrome_data = Image.pixel_array
           monochrome_data = cv2.resize(monochrome_data,(224,224))
           rgb_data = np.stack([monochrome_data, monochrome_data, monochrome_data])
           rgb_data = rgb_data.transpose((1, 2, 0))
           X_covid.append(rgb_data)
           y_covid.append(0) # 0 = covid data, 1 = cancer data
```

## PREPROCESSING

- Necessary to run a preprocess_inputs algorithm specific to each pretrained model (VGG19, ResNet-50, InceptionV3)
- Doing so removes the mean from each color channel that was created previously



```
[ ]  # Preprocess inputs
     X_train = preprocess_input(X_train)
     X_test = preprocess_input(X_test)
```

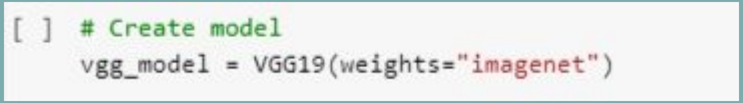
03

## PRETRAINED MODELS & TRANSFER LEARNING

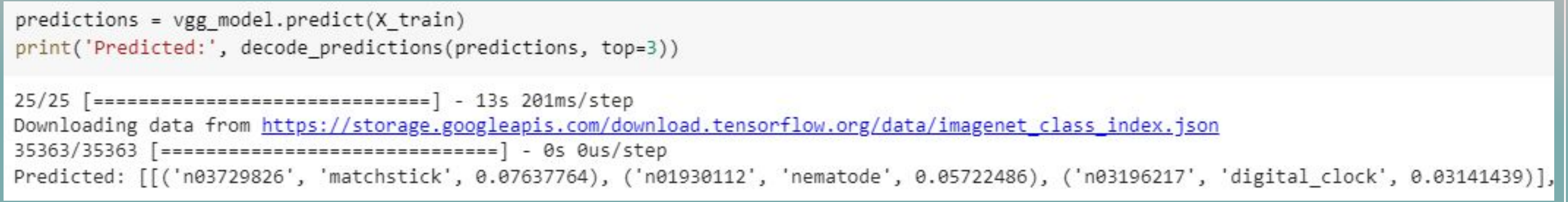
## PRETRAINED MODEL PREDICTIONS

After creating each model, we decided to run it as it is.

The top three predictions for the VGG19 model from the imaging are matchstick, nematode, and digital clock.



```
[ ]  # Create model
     vgg_model = VGG19(weights="imagenet")
```

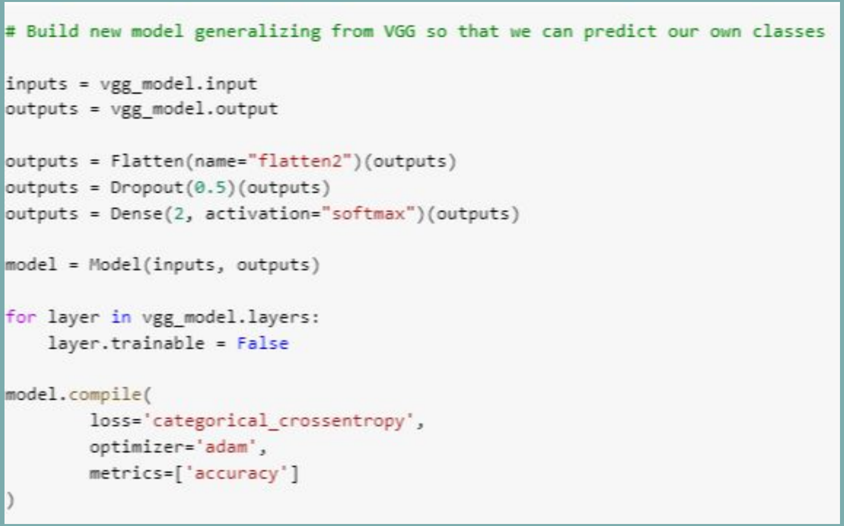


```
predictions = vgg_model.predict(X_train)
print('Predicted:', decode_predictions(predictions, top=3))

25/25 [==============================] - 13s 201ms/step
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
35363/35363 [==============================] - 0s 0us/step
Predicted: [[('n03729826', 'matchstick', 0.07637764), ('n01930112', 'nematode', 0.05722486), ('n03196217', 'digital_clock', 0.03141439)],
```

*Code is from VGG19 model notebook

# TRANSFER LEARNING

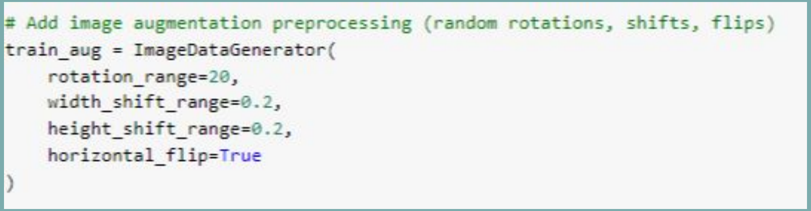
We had to perform a transfer learning process with our own target labels ("cancer" and "covid") in order to generalize the pretrained models.



```
# Build new model generalizing from VGG so that we can predict our own classes

inputs = vgg_model.input
outputs = vgg_model.output

outputs = Flatten(name="flatten2")(outputs)
outputs = Dropout(0.5)(outputs)
outputs = Dense(2, activation="softmax")(outputs)

model = Model(inputs, outputs)

for layer in vgg_model.layers:
    layer.trainable = False

model.compile(
        loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy']
)
```

## Transfer Learning:

A machine learning technique where the application of knowledge gained from completing one task is used to help solve a different, but related, problem.



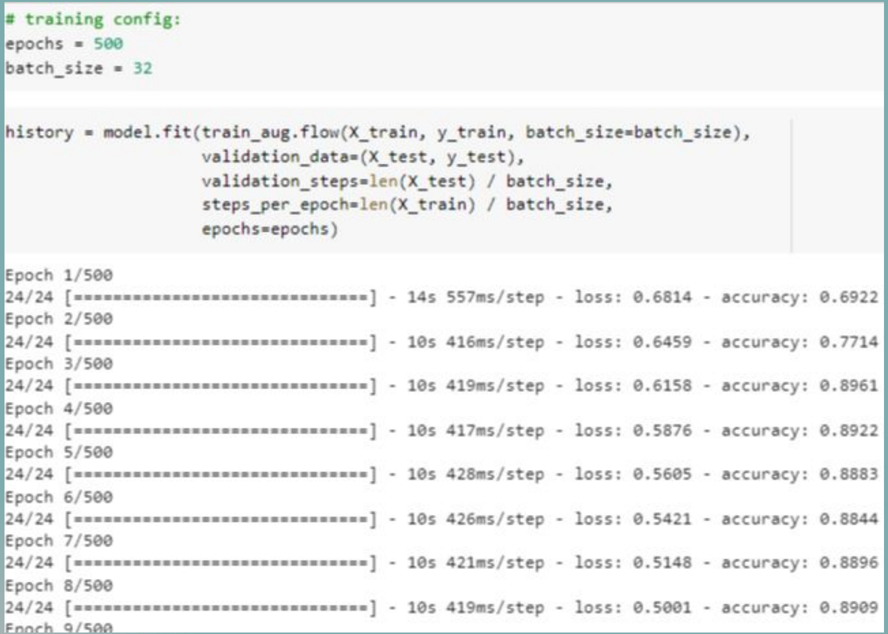
```
# Add image augmentation preprocessing (random rotations, shifts, flips)
train_aug = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True
)
```

*Code is from VGG19 model notebook

04

## RESULTS

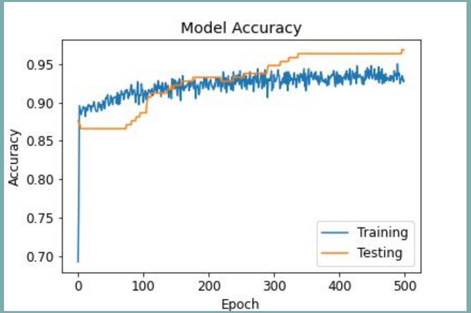
# TESTING



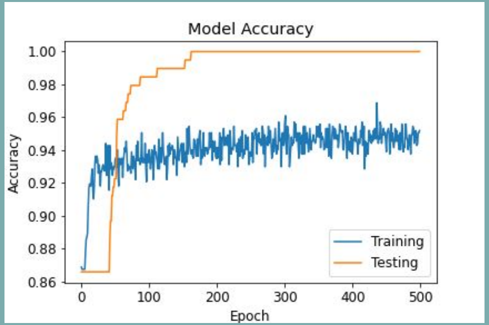
```
# training config:
epochs = 500
batch_size = 32

history = model.fit(train_aug.flow(X_train, y_train, batch_size=batch_size),
                    validation_data=(X_test, y_test),
                    validation_steps=len(X_test) / batch_size,
                    steps_per_epoch=len(X_train) / batch_size,
                    epochs=epochs)

Epoch 1/500
24/24 [==============================] - 14s 557ms/step - loss: 0.6814 - accuracy: 0.6922
Epoch 2/500
24/24 [==============================] - 10s 416ms/step - loss: 0.6459 - accuracy: 0.7714
Epoch 3/500
24/24 [==============================] - 10s 419ms/step - loss: 0.6158 - accuracy: 0.8961
Epoch 4/500
24/24 [==============================] - 10s 417ms/step - loss: 0.5876 - accuracy: 0.8922
Epoch 5/500
24/24 [==============================] - 10s 428ms/step - loss: 0.5605 - accuracy: 0.8883
Epoch 6/500
24/24 [==============================] - 10s 426ms/step - loss: 0.5421 - accuracy: 0.8844
Epoch 7/500
24/24 [==============================] - 10s 421ms/step - loss: 0.5148 - accuracy: 0.8896
Epoch 8/500
24/24 [==============================] - 10s 419ms/step - loss: 0.5001 - accuracy: 0.8909
Epoch 9/500
```

- Our initial runs were between 5 and 100 epochs
- Despite initially high accuracy scores, the predictions on our testing data were only resulting in one label being predicted for all the test data
- Increasing the epoch number to 500 resulted in predictions that applied both target labels

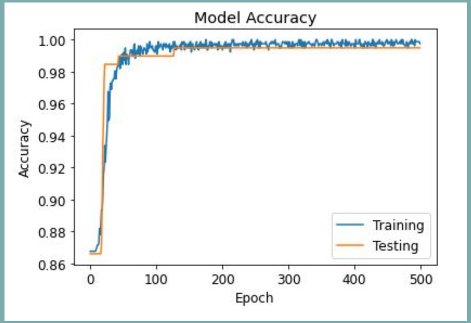
# ACCURACY



VGG19

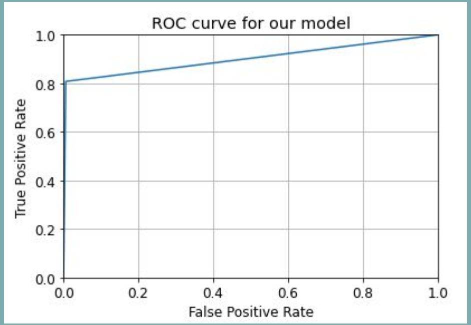


ResNet50



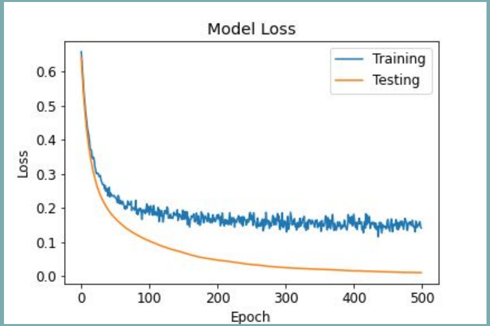
InceptionV3

# OTHER RESULTS



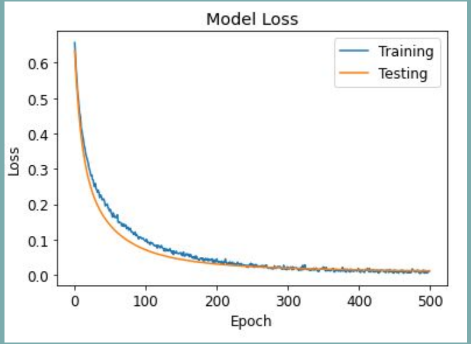
## VGG19

ROC Curve



## ResNet50

Losses over epoches



## InceptionV3

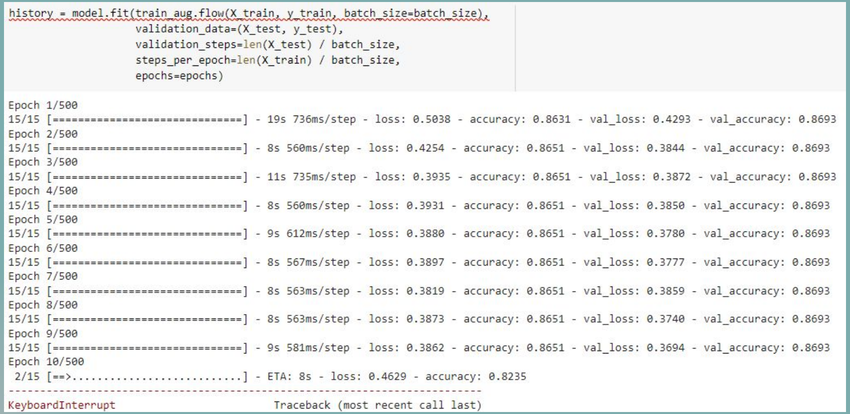
Losses over epoches

## FINDINGS

- All three models performed well in terms of accuracy on the datasets
- After our first successful run, we had some suspicion due to our COVID data initially coming from signed integer data and the cancer data coming from unsigned integers
  - The models could potentially be detecting this trivial difference, and not granting us any sort of medically useful classification abilities

# TROUBLESHOOTING

- Implementing rescaling of both datasets so their numerical values were in [0, 1]
- Varying the Adam optimizer learning rate between 0.001, 0.01, and 0.1
- Changing to an SGD optimizer with a learning rate of 0.1 and momentum of 0.9
- Varying the batch size
- Rescaling with StandardScaler function
- Rescaling by hand



```
history = model.fit(train_aug.flow(X_train, y_train, batch_size=batch_size),
                    validation_data=(X_test, y_test),
                    validation_steps=len(X_test) / batch_size,
                    steps_per_epoch=len(X_train) / batch_size,
                    epochs=epochs)
```

```
Epoch 1/500
15/15 [==============================] - 19s 736ms/step - loss: 0.5038 - accuracy: 0.8631 - val_loss: 0.4293 - val_accuracy: 0.8693
Epoch 2/500
15/15 [==============================] - 8s 560ms/step - loss: 0.4254 - accuracy: 0.8651 - val_loss: 0.3844 - val_accuracy: 0.8693
Epoch 3/500
15/15 [==============================] - 11s 735ms/step - loss: 0.3935 - accuracy: 0.8651 - val_loss: 0.3872 - val_accuracy: 0.8693
Epoch 4/500
15/15 [==============================] - 8s 560ms/step - loss: 0.3931 - accuracy: 0.8651 - val_loss: 0.3850 - val_accuracy: 0.8693
Epoch 5/500
15/15 [==============================] - 9s 612ms/step - loss: 0.3880 - accuracy: 0.8651 - val_loss: 0.3780 - val_accuracy: 0.8693
Epoch 6/500
15/15 [==============================] - 8s 567ms/step - loss: 0.3897 - accuracy: 0.8651 - val_loss: 0.3777 - val_accuracy: 0.8693
Epoch 7/500
15/15 [==============================] - 8s 563ms/step - loss: 0.3819 - accuracy: 0.8651 - val_loss: 0.3859 - val_accuracy: 0.8693
Epoch 8/500
15/15 [==============================] - 8s 563ms/step - loss: 0.3873 - accuracy: 0.8651 - val_loss: 0.3740 - val_accuracy: 0.8693
Epoch 9/500
15/15 [==============================] - 9s 581ms/step - loss: 0.3862 - accuracy: 0.8651 - val_loss: 0.3694 - val_accuracy: 0.8693
Epoch 10/500
 2/15 [==>...........................] - ETA: 8s - loss: 0.4629 - accuracy: 0.8235
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
```

After running our models again, the accuracy scores stopped improving after one or two epochs. The losses varied slightly but did not improve.

All three models suffered this problem.

## CONCLUSIONS

The ResNet50 model showed the most and earliest improvement of the testing accuracy relative to the training. Due to this, we recommend proceeding with that model.

However, we would not recommend any of the models for use in medical applications at this time.

# THANK YOU