Prediction of Class of Tumor Through Radiology Image Processing

By Ankit Jain
Nishan Shetty

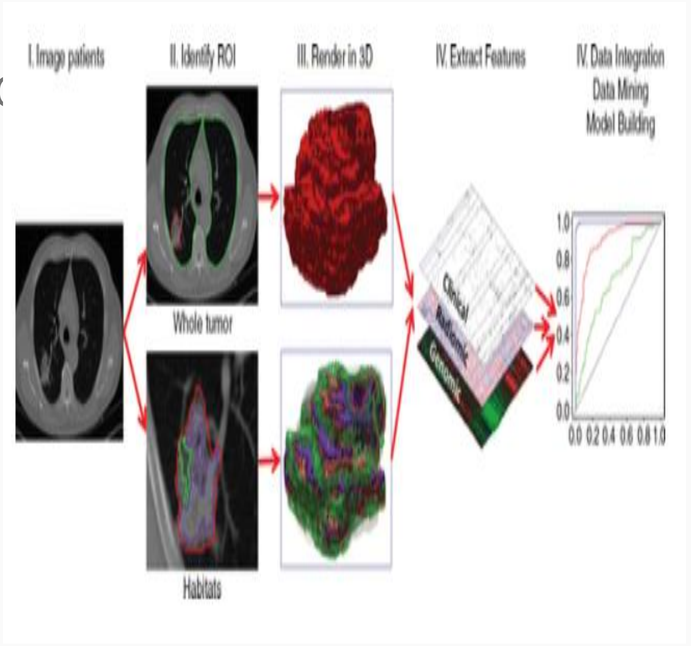
Introduction

- We built a Convolutional Neural Network (ConvNet) to classify a data set of 100 Colon Cancer CT images between two classes: Mutant or Wild.
- KRAS is the gene marker here, and patients that are found to have mutated it are also found to have lower rates of survival overall. Hence, preemptive detection at an early stage of this gene marker can often prove vital to patient mortality.
- Segmentation of images is done by physician of Mayo Clinic

Introduction

- Segmentation of the Region of Interest (ROI) of each image was done by physician at Mayo Clinic. Coordinates of the R patches were also provided, useful for feature extraction.

- The images are DICOM images, with metadata about the images and other radiology results encoded in them
- Before we build the network, we perform some data preprocessing and augmentation to improve computational efficiency and reduce overfitting

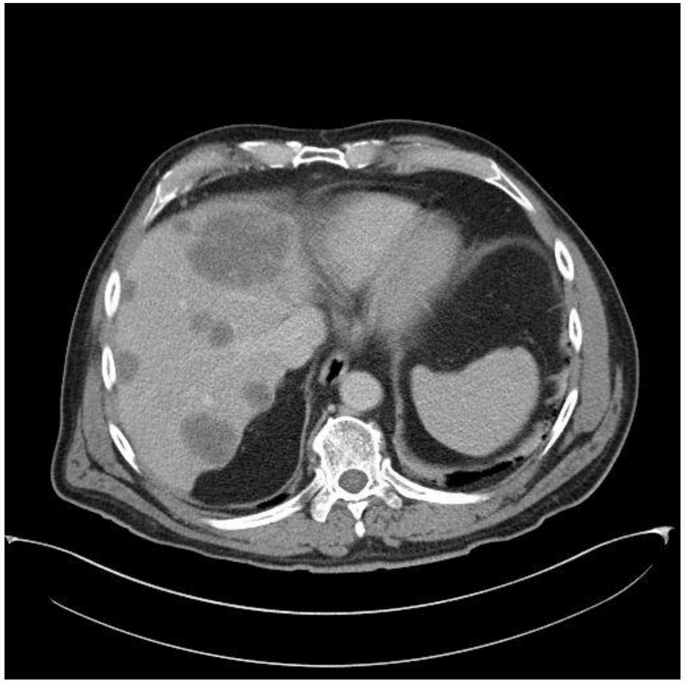


Data Preprocessing

Changing the format of the file : All the DICOM files were converted to jpeg format so that it becomes easier to apply classification model.

Segmentation: Image were cropped in a rectangular shape with the largest width and height provided with data set which contains ROI of the image.

Original images were 512* 512 in size and cropped images are of 72*72 in size.



$\longrightarrow$

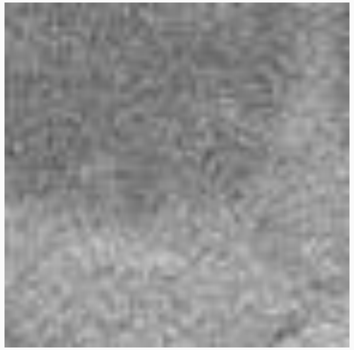
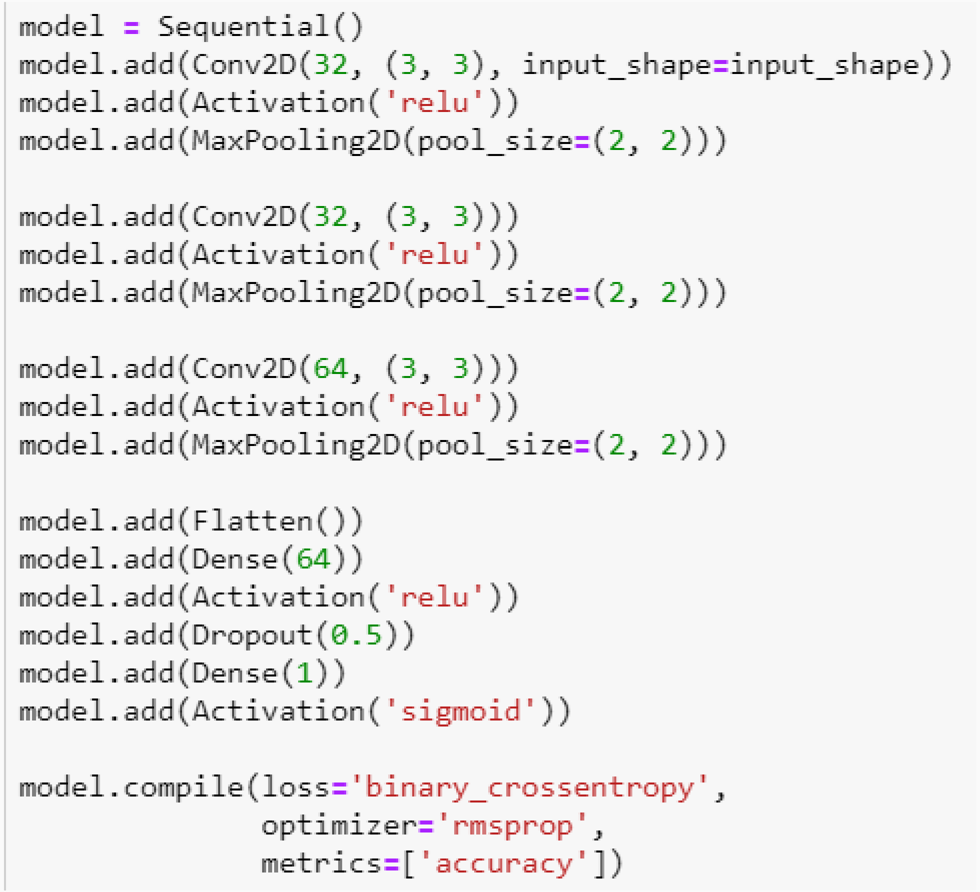


Image Segmentation

Network Architecture

- We used a very small ConvNet with a few layers and few filters per layer and dropout. Dropout helps reduce overfitting by preventing a layer from seeing twice the exact same pattern.
- The architecture consists of a simple stack of 3 convolution layers with a Rectified Linear Unit (ReLu) activation function and filter stride 3x3, followed by 2x2 size max pooling layers.
- On top of it we use two fully connected layers. Finally we end the model with a sigmoid activation since it’s a binary classification problem.



```python
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

Network Architecture

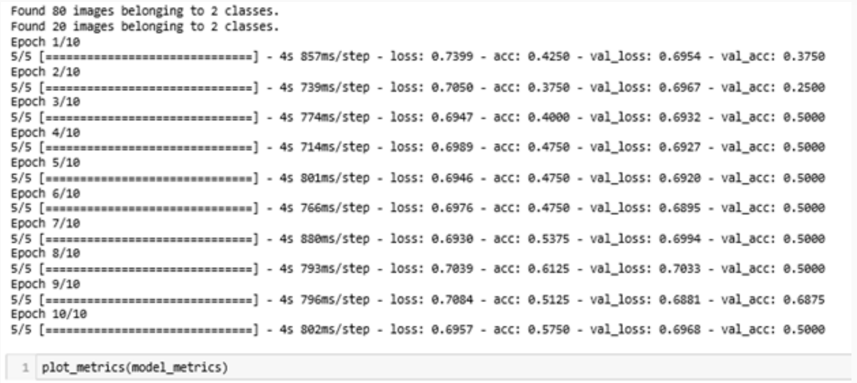
Model 1

ConvNet on Original Data Set:

- We partitioned the data set into 80 images for our training set and the remaining 20 images as our validation set.
- With so few samples to train on, we suspect overfitting will occur which can be shown by the large variance displayed by the accuracy learning curve for the model along each epoch

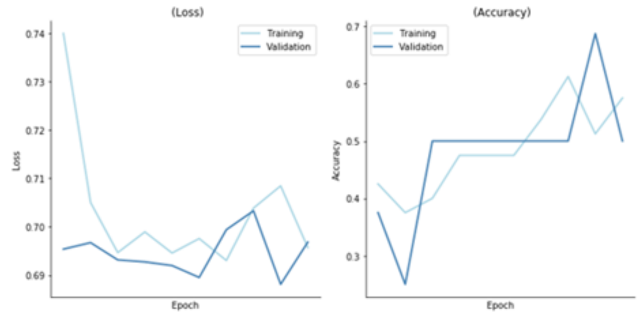
Model 1

- Other hyperparameters used are: Optimizer = RMSProp, Epochs =20, Batch Size = 16
- As suspected, the validation accuracy varies between 25% to 68.75% through all the epochs, indicating serious overfitting which leads to the large variance
- To solve for this, we decided to augment the data set



```
Found 80 images belonging to 2 classes.
Found 20 images belonging to 2 classes.
Epoch 1/10
5/5 [==============================] - 4s 857ms/step - loss: 0.7399 - acc: 0.4250 - val_loss: 0.6954 - val_acc: 0.3750
Epoch 2/10
5/5 [==============================] - 4s 739ms/step - loss: 0.7050 - acc: 0.3750 - val_loss: 0.6967 - val_acc: 0.2500
Epoch 3/10
5/5 [==============================] - 4s 774ms/step - loss: 0.6947 - acc: 0.4000 - val_loss: 0.6932 - val_acc: 0.5000
Epoch 4/10
5/5 [==============================] - 4s 714ms/step - loss: 0.6989 - acc: 0.4750 - val_loss: 0.6927 - val_acc: 0.5000
Epoch 5/10
5/5 [==============================] - 4s 801ms/step - loss: 0.6946 - acc: 0.4750 - val_loss: 0.6920 - val_acc: 0.5000
Epoch 6/10
5/5 [==============================] - 4s 766ms/step - loss: 0.6976 - acc: 0.4750 - val_loss: 0.6895 - val_acc: 0.5000
Epoch 7/10
5/5 [==============================] - 4s 880ms/step - loss: 0.6930 - acc: 0.5375 - val_loss: 0.6994 - val_acc: 0.5000
Epoch 8/10
5/5 [==============================] - 4s 793ms/step - loss: 0.7039 - acc: 0.6125 - val_loss: 0.7033 - val_acc: 0.5000
Epoch 9/10
5/5 [==============================] - 4s 796ms/step - loss: 0.7084 - acc: 0.5125 - val_loss: 0.6881 - val_acc: 0.6875
Epoch 10/10
5/5 [==============================] - 4s 802ms/step - loss: 0.6957 - acc: 0.5750 - val_loss: 0.6968 - val_acc: 0.5000
```

```
plot_metrics(model_metrics)
```

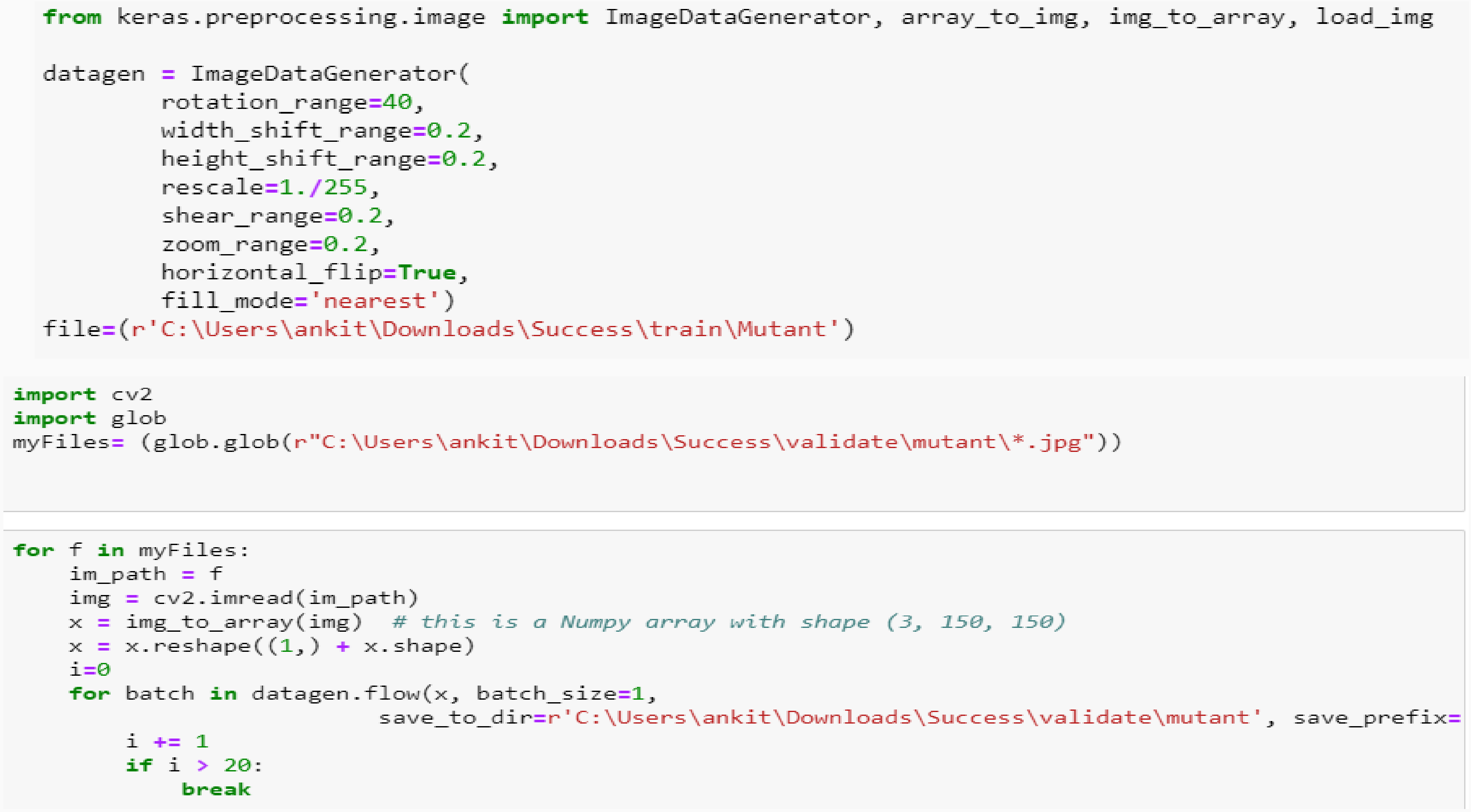


Data Augmentation

- To augment the data set, we defined an ImageDataGenerator() function to randomly configure several transformations and normalization operations on the image data
- Parameters given were a rotation range, width and height translation limits, rescaling by a factor of 1/255, flipping, shearing and zooming ranges

Data Augmentation

- These operations resulted in an augmented training data set of 1685 images and a validation data set of 437 images
- To see the effect of these operations and transformations, we fit the ConvNet on the now augmented data set



```
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img

datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
file=(r'C:\Users\ankit\Downloads\Success\train\Mutant')
```

```
import cv2
import glob
myFiles= (glob.glob(r"C:\Users\ankit\Downloads\Success\validate\mutant\*.jpg"))
```

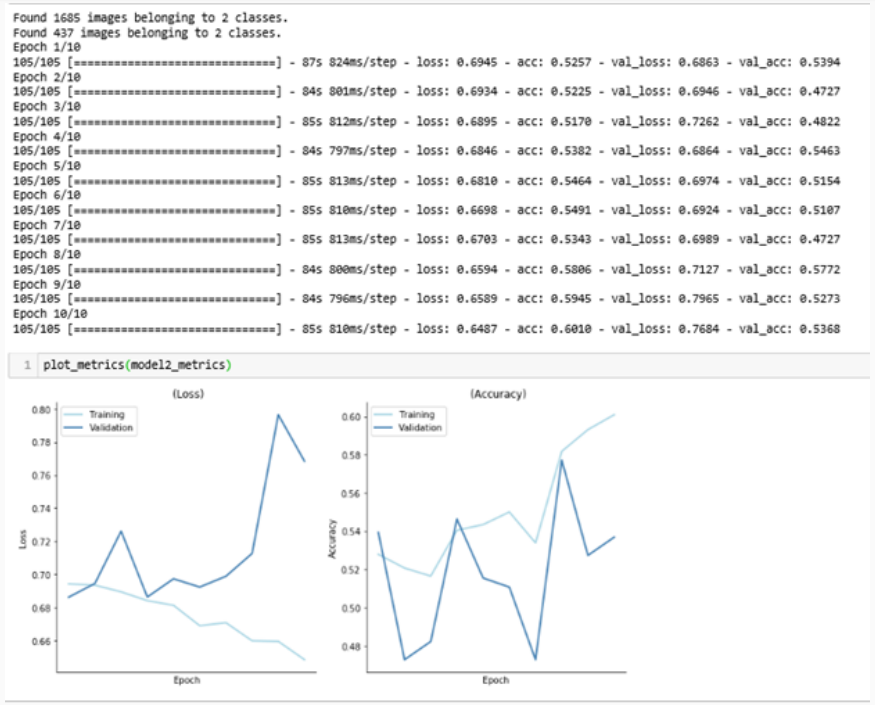
```
for f in myFiles:
    im_path = f
    img = cv2.imread(im_path)
    x = img_to_array(img)  # this is a Numpy array with shape (3, 150, 150)
    x = x.reshape((1,) + x.shape)
    i=0
    for batch in datagen.flow(x, batch_size=1,
                              save_to_dir=r'C:\Users\ankit\Downloads\Success\validate\mutant', save_prefix=
        i += 1
        if i > 20:
            break
```

Data Augmentation

Model 2

ConvNet on Augmented Data Set:

- This time, we used the Adaptive Moment Estimation (Adam) optimizer, but kept the rest of the hyperparameters the same
- As we can see from the Accuracy and Loss learning curves, the variation is much smaller indicating a much better fit and more reliable predictions compared to the previous model



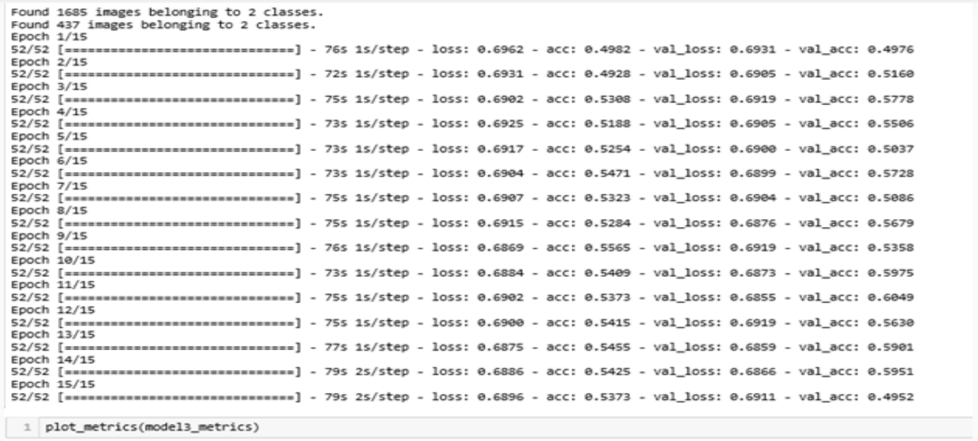
```
Found 1685 images belonging to 2 classes.
Found 437 images belonging to 2 classes.
Epoch 1/10
105/105 [==============================] - 87s 824ms/step - loss: 0.6945 - acc: 0.5257 - val_loss: 0.6863 - val_acc: 0.5394
Epoch 2/10
105/105 [==============================] - 84s 801ms/step - loss: 0.6934 - acc: 0.5225 - val_loss: 0.6946 - val_acc: 0.4727
Epoch 3/10
105/105 [==============================] - 85s 812ms/step - loss: 0.6895 - acc: 0.5170 - val_loss: 0.7262 - val_acc: 0.4822
Epoch 4/10
105/105 [==============================] - 84s 797ms/step - loss: 0.6846 - acc: 0.5382 - val_loss: 0.6864 - val_acc: 0.5463
Epoch 5/10
105/105 [==============================] - 85s 813ms/step - loss: 0.6810 - acc: 0.5464 - val_loss: 0.6974 - val_acc: 0.5154
Epoch 6/10
105/105 [==============================] - 85s 810ms/step - loss: 0.6698 - acc: 0.5491 - val_loss: 0.6924 - val_acc: 0.5107
Epoch 7/10
105/105 [==============================] - 85s 813ms/step - loss: 0.6703 - acc: 0.5343 - val_loss: 0.6989 - val_acc: 0.4727
Epoch 8/10
105/105 [==============================] - 84s 800ms/step - loss: 0.6594 - acc: 0.5806 - val_loss: 0.7127 - val_acc: 0.5772
Epoch 9/10
105/105 [==============================] - 84s 796ms/step - loss: 0.6589 - acc: 0.5945 - val_loss: 0.7965 - val_acc: 0.5273
Epoch 10/10
105/105 [==============================] - 85s 810ms/step - loss: 0.6487 - acc: 0.6010 - val_loss: 0.7684 - val_acc: 0.5368
```

```
plot_metrics(model2_metrics)
```

Model 3

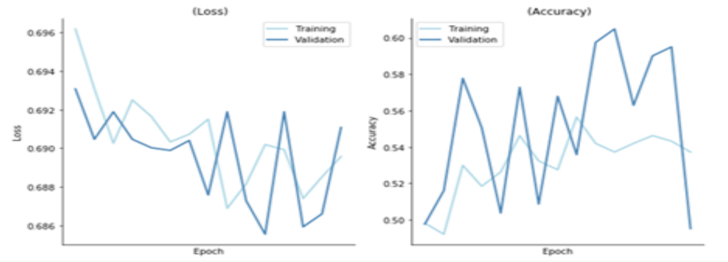
Hyperparameter Tuning of ConvNet:

- In an effort to compare different models, we tuned some of the hyperparameters for this model. We used the Stochastic Gradient Descent (SGD) optimizer, 15 epochs, batch size of 32 and a filter stride of 5x5 at the last convolution layer
- We observed that the learning curve for accuracy is similar to the previous model's, but the loss curve indicates lower loss in the validation step along each epoch compared to the same in the previous model



```
Found 1685 images belonging to 2 classes.
Found 437 images belonging to 2 classes.
Epoch 1/15
52/52 [==============================] - 76s 1s/step - loss: 0.6962 - acc: 0.4982 - val_loss: 0.6931 - val_acc: 0.4976
Epoch 2/15
52/52 [==============================] - 72s 1s/step - loss: 0.6931 - acc: 0.4928 - val_loss: 0.6905 - val_acc: 0.5160
Epoch 3/15
52/52 [==============================] - 75s 1s/step - loss: 0.6902 - acc: 0.5308 - val_loss: 0.6919 - val_acc: 0.5778
Epoch 4/15
52/52 [==============================] - 73s 1s/step - loss: 0.6925 - acc: 0.5188 - val_loss: 0.6905 - val_acc: 0.5506
Epoch 5/15
52/52 [==============================] - 73s 1s/step - loss: 0.6917 - acc: 0.5254 - val_loss: 0.6900 - val_acc: 0.5037
Epoch 6/15
52/52 [==============================] - 73s 1s/step - loss: 0.6904 - acc: 0.5471 - val_loss: 0.6899 - val_acc: 0.5728
Epoch 7/15
52/52 [==============================] - 75s 1s/step - loss: 0.6907 - acc: 0.5323 - val_loss: 0.6904 - val_acc: 0.5086
Epoch 8/15
52/52 [==============================] - 75s 1s/step - loss: 0.6915 - acc: 0.5284 - val_loss: 0.6876 - val_acc: 0.5679
Epoch 9/15
52/52 [==============================] - 76s 1s/step - loss: 0.6869 - acc: 0.5565 - val_loss: 0.6919 - val_acc: 0.5358
Epoch 10/15
52/52 [==============================] - 73s 1s/step - loss: 0.6884 - acc: 0.5409 - val_loss: 0.6873 - val_acc: 0.5975
Epoch 11/15
52/52 [==============================] - 75s 1s/step - loss: 0.6902 - acc: 0.5373 - val_loss: 0.6855 - val_acc: 0.6049
Epoch 12/15
52/52 [==============================] - 75s 1s/step - loss: 0.6900 - acc: 0.5415 - val_loss: 0.6919 - val_acc: 0.5630
Epoch 13/15
52/52 [==============================] - 77s 1s/step - loss: 0.6875 - acc: 0.5455 - val_loss: 0.6859 - val_acc: 0.5901
Epoch 14/15
52/52 [==============================] - 79s 2s/step - loss: 0.6886 - acc: 0.5425 - val_loss: 0.6866 - val_acc: 0.5951
Epoch 15/15
52/52 [==============================] - 79s 2s/step - loss: 0.6896 - acc: 0.5373 - val_loss: 0.6911 - val_acc: 0.4952
```

```
plot_metrics(model3_metrics)
```



Discussion

- Overfitting was a concern from the start. More unique CT images would have improved the predictive ability of the ConvNet. Dropout Regularization at the fully connected layers and data augmentation mitigated the effects of this, but they are not enough as the augmented samples are still highly correlated.
- Access to more powerful hardware would have allowed us to train a more extensive architecture for longer. Training our network for 50 epochs was computationally expensive, something which our laptops could not handle quickly and efficiently.
- Using a pre-trained model such as VGG16 or ImageNet and performed Transfer Learning with those architectures would certainly lead to a better performance.
- Incorporation of spatial characteristics (such as unilateral patterns of growth and localization) would have further improved model performance.

THANK YOU.....