Deep Learning for Radiological Image Processing

...

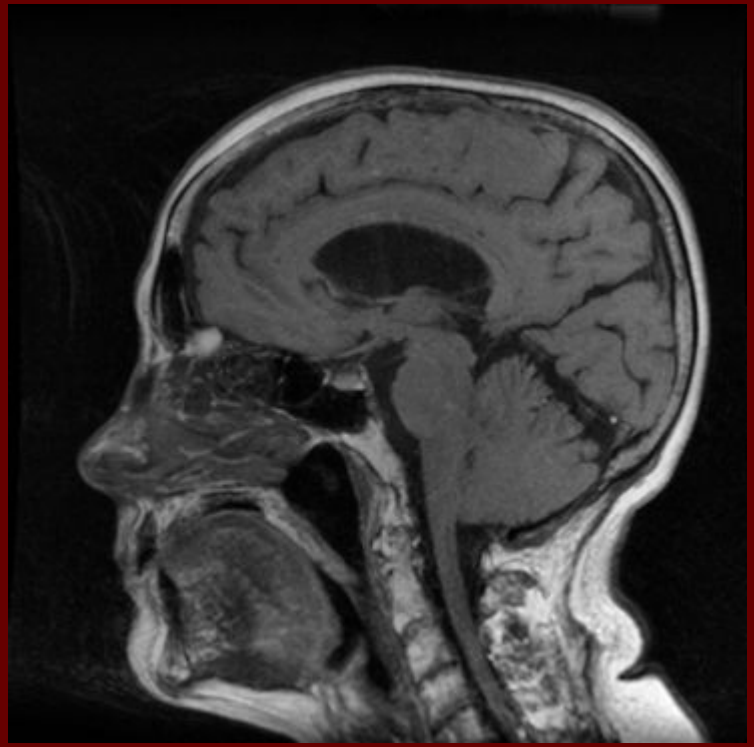
Authors: Mohannad Hussain & Marc Kohli
Society for Imaging Informatics in Medicine

Presented by: Allyson Warren & Leslie Cook

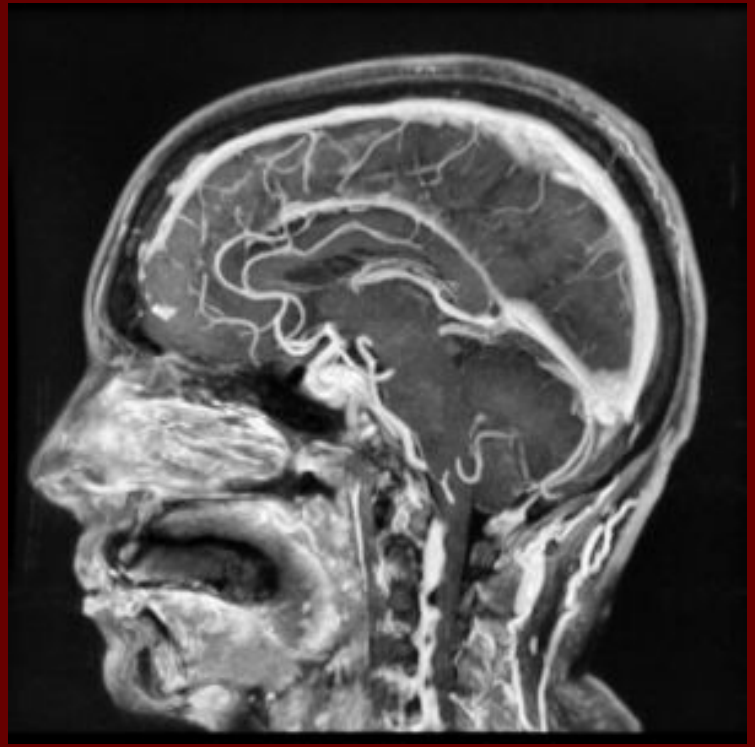
What is the Goal

- Deep Learning algorithms have the potential to give higher accuracy in identifying abnormalities in radiological images than the human eye
- In this project, the goal is to classify sagittal T1 MRI scans by pre- and post-contrast
- Sagittal describes how the body (brain) is divided into left and right halves
- TI is the time constant that determines the rate at which excited protons return to equilibrium
- Sagittal T1 MRI's show nerve connections of white matter to appear white, congregations of neurons of gray matter to appear gray, and the cerebrospinal fluid to appear dark

Pre-contrast



Post-contrast



Dataset and what it is

Given two folders: Train and Test

Training Dataset

- **Train/with_gad :** contains sequences with contrast
  - 209 items, 22.7 MB
- **Train/no_gad :** contains sequences without contrast
  - 695 items, 88.2 MB

Testing Dataset

- **Test/with_gad :** contains sequences with contrast
  - 118 items, 11.4 MB
- **Test/no_gad :** contains sequences without contrast
  - 445 items, 40.9 MB

Need to split images into training, validation, and test sets

Virtual Environment and System Requirements

In [3]:

```python
#What version of python do you have
import sys

import keras
import pandas as pd
import sklearn as sk
import tensorflow as tf

print(f"Tensor Flow Version: {tf.__version__}")
print(f"Keras Version: {keras.__version__}")
print()
print(f"Python Version: {sys.version}")
print(f"Pandas Version: {pd.__version__}")
print(f"Scikit-Learn Version: {sk.__version__}")
```

```
Tensor Flow Version: 2.9.1
Keras Version: 2.9.0

Python Version: 3.10.4 | packaged by conda-forge | (main, Mar 30 2022, 08:38:02) [MSC v.1916 64 bit (AMD64)]
Pandas Version: 1.4.2
Scikit-Learn Version: 1.1.1
```

Library Requirements

- **sys :** system specific parameters and functions
- **matplotlib :** data visualization library in Python (plots, graphs, stats)
- **pyplot :** matplotlib's plotting framework
- **numpy :** library for large scale mathematical operations
- **tensorflow :** library used in neural networks for dataflow programming across a range of tasks.
- **keras :** high-level API of tensorflow, highly-productive interface for solving DL problems
- **sklearn :** machine learning library of tools for predictive data analysis
- **os :** module to interact with underlying operating system
- **itertool :** collection of tools for handling iterators (data types used in for loops)

Importing the Libraries

- Setting the seed is starting point for generating random numbers
- Important for reproducible results
- Keras ustilizes NumPy random number generator with tensorflow backend

```
import sys
from matplotlib import pyplot
import numpy as np

from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Input
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.random import set_seed
from tensorflow.keras.activations import sigmoid
import os
from sklearn.metrics import confusion_matrix
from sklearn.metrics import auc
from sklearn.metrics import roc_curve
from sklearn.metrics import accuracy_score
import itertools

#This command makes only 1 GPU visible to our code, in case we have multiple GPUs.
os.environ["CUDA_VISIBLE_DEVICES"]="0"

# To have reproducible results, it is important to set the seed for all tensorflow functions
set_seed(1020)
```

Importing the Libraries

tensorflow.keras...

- **.models :** allows us to build and load the model into our neural network
- **.layers :** equips the model with a series of layers to process the images
- **.optimizers :** gradient descent algorithms to learn the best params
- **.preprocessing.image :** data augmentation of images in real time while the model is training
- **.activations** : classification function

sklearn.metrics...

- **import confusion_matrix :** technique to calculate the performance of classification
- **import auc :** computes area under the curve
- **import roc_curve :** plots the rate of true positives and false positives for a binary classifier
- **import accuracy_score :** computes the accuracy of the count of correct predictions

Variable Configuration

```
TRAIN_DATA_DIR = r'Train/Train'
TEST_DATA_DIR = r'Test/Test'
```

- Define the folders where the dataset is saved

```
IMG_HEIGHT = 331
IMG_WIDTH = 331
```

- Define the input size of our network

```
BATCH_SIZE = 64
```

- Define the size of the batch to leverage

DataGenerator functions will fetch the files, automatically resize the images, and set the capacity GPUs have to parallelize.

Function: Plot the Learning Curves

- Binary Cross Entropy Loss
- Binary Classification Accuracy

$$CE = -t_1 log(f(s_1)) - (1 - t_1) log(1 - f(s_1))$$

$$f(s_i) = \frac{1}{1 + e^{-s_i}}$$

```python
def plot_learning_curves(history):
    # plot loss
    pyplot.figure(figsize=(10, 5))
    pyplot.title('Binary Cross Entropy Loss')
    pyplot.plot(history.history['loss'], color='blue', label='Train')
    pyplot.plot(history.history['val_loss'], color='orange', label='Validation')
    pyplot.legend(loc='upper right', shadow=True, fontsize='x-large')
    pyplot.show()
    # plot accuracy
    pyplot.figure(figsize=(10, 5))
    pyplot.title('Binary Classification Accuracy')
    pyplot.plot(history.history['accuracy'], color='blue', label='Train')
    pyplot.plot(history.history['val_accuracy'], color='orange', label='Validation')
    pyplot.legend(loc='lower right', shadow=True, fontsize='x-large')
    pyplot.show()
```

Creating a DataGenerator to Read in Images

```
train_datagen = ImageDataGenerator(rescale=1. / 255,
                                   #shear_range=0.2,
                                   #zoom_range=0.2,
                                   #horizontal_flip=False,
                                   #vertical_flip=True,
                                   #rotation_range=0,
                                   #fill_mode='constant',
                                   #cval=0,
                                   #preprocessing_function=preprocess_input,
                                   validation_split=0.2)  # set validation split

train_it = train_datagen.flow_from_directory(
    TRAIN_DATA_DIR,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    color_mode='rgb',
    subset='training')  # set as training data

val_it = train_datagen.flow_from_directory(
    TRAIN_DATA_DIR,  # same directory as training data
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    color_mode='rgb',
    subset='validation')  # set as validation data
```

train_datagen : Keras ImageDataGenerator that will split ≈20% of the training data for the validation data set

train_it : array of training data to feed into the CNN model, stores 722 rgb training images

val_it : array validation data that is a subset from the TRAIN_DATA_DIR , holds 179 rgb validation images.

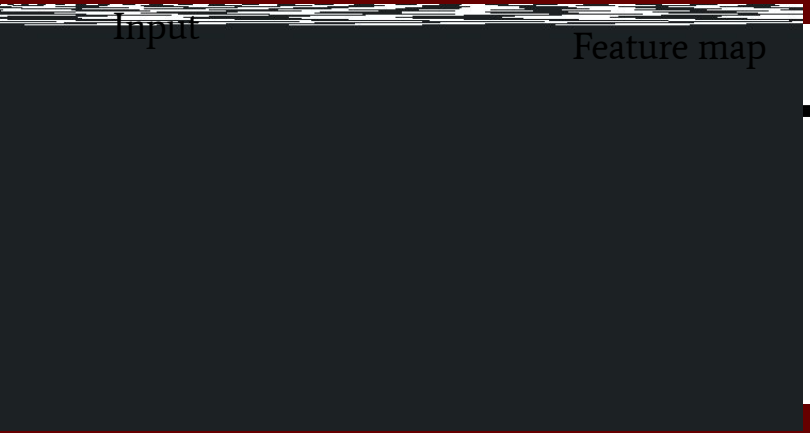
```
Found 722 images belonging to 2 classes.
Found 179 images belonging to 2 classes.
```

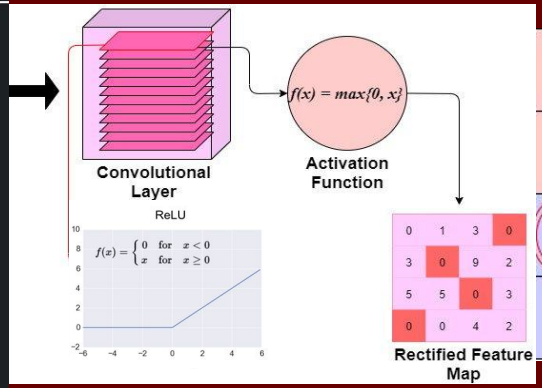
Convolution Neural Network Framework

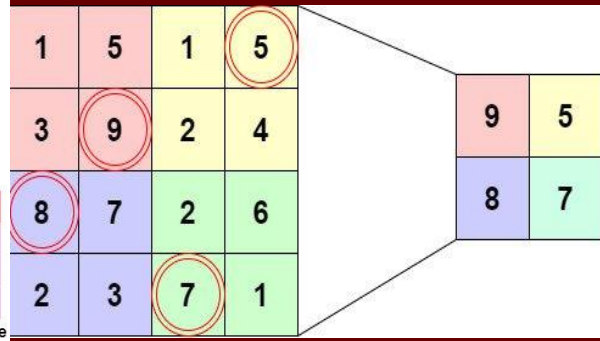
Convolution Layer

RELU activation

MaxPool Layer

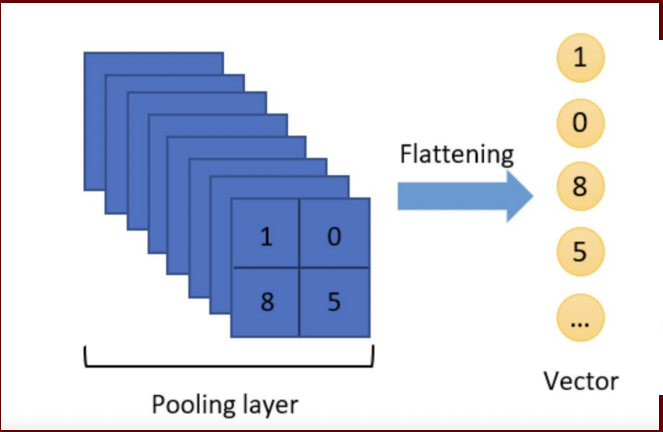




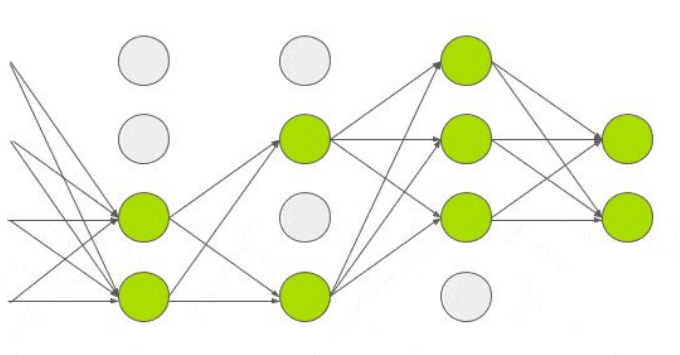


Convolution Neural Network Framework

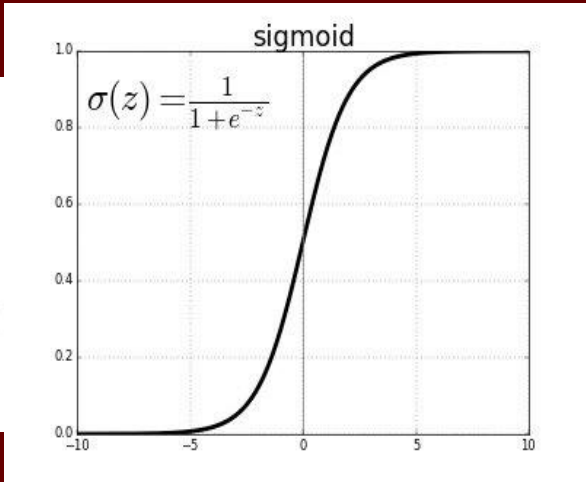
Flatten



Dense + Dropout



Binary Classification



Setting up the CNN

- **Input :** model input of shape (331x331x3) rgb images (x1)
- **Conv2D :** creates a convolution kernel that is convolved with the layer input to produce a tensor (multi-dimensional array) of outputs
- **Activation :** RELU, rectified linear unit, speeds up training with a simple gradient of 0 or 1, where all negative elements are set to 0 and computation is easy.
- **MaxPooling2D :** results in pooled feature maps that highlight the most prominent feature in the patch, reducing dimension volume by half.
- **Flatten :** flattens the multi-dimensional input tensors into a single dimension
- **Dropout :** used to avoid overfitting the data

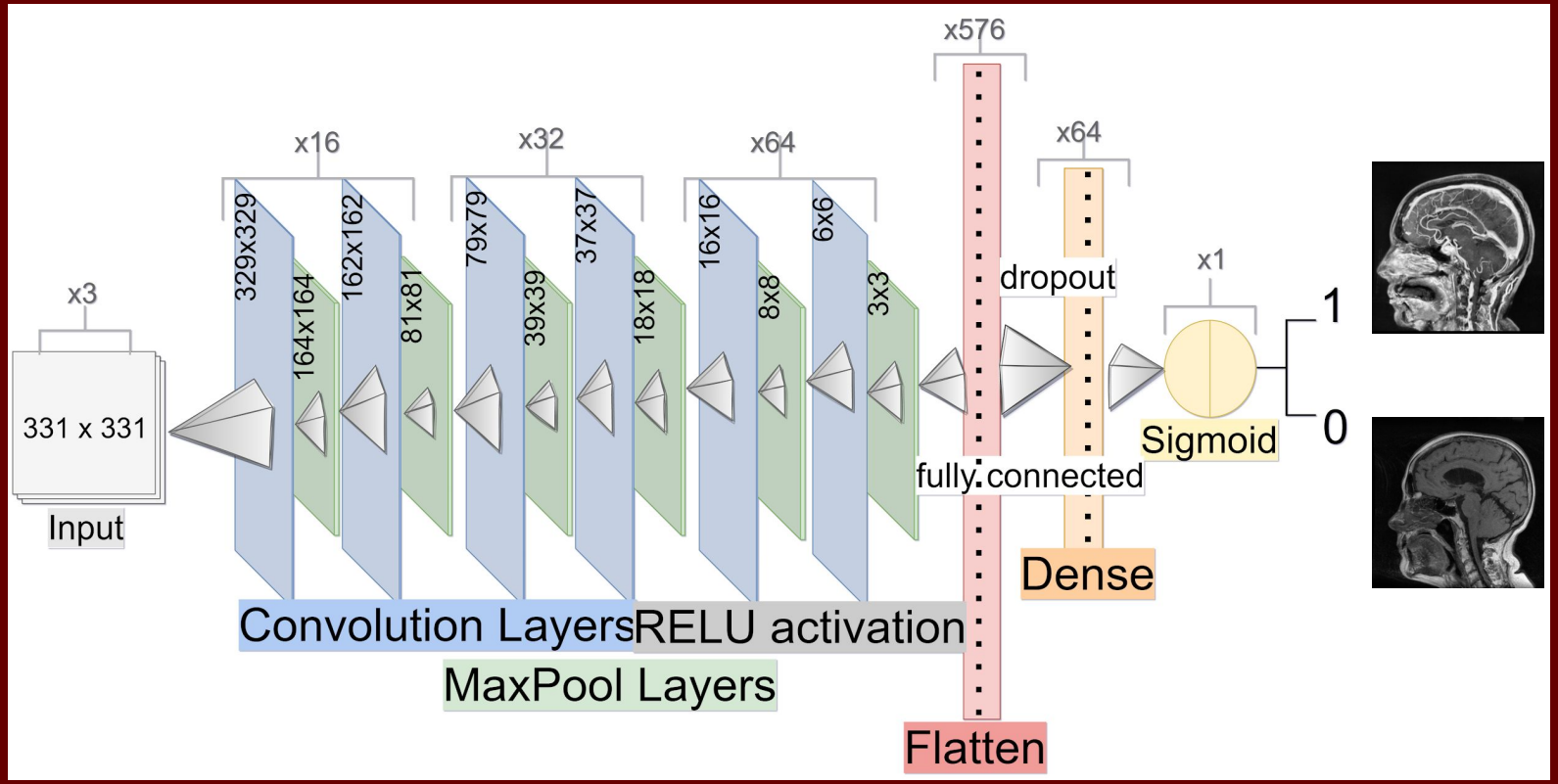
```
# Every model needs to have an input
x1 = Input(shape=(IMG_HEIGHT,IMG_WIDTH,3))

# Below, we add paired convolutional and maxpooling layers
x = Conv2D(16, (3,3), activation='relu')(x1)
x = MaxPooling2D()(x)
x = Conv2D(16, (3,3), activation='relu')(x)
x = MaxPooling2D()(x)
x = Conv2D(32, (3,3), activation='relu')(x)
x = MaxPooling2D()(x)
x = Conv2D(32, (3,3), activation='relu')(x)
x = MaxPooling2D()(x)
x = Conv2D(64, (3,3), activation='relu')(x)
x = MaxPooling2D()(x)
x = Conv2D(64, (3,3), activation='relu')(x)
x = MaxPooling2D()(x)

# Then we flatten the last vector
flat1 = Flatten()(x)
# Insert a dropout layer with 20% probability
flat2 = Dropout(0.2)(flat1)
```

Setting up the CNN

- **Dense :** used to create a fully connected model that receives all input from the previous layer
- **Kernel_initilaizer :** 'he_uniform' draws samples from a uniform distribution (-limit, limit)
- **Output :** sigmoid activation for binary classification
- **Model :** inputs the keras.Input model, outputs the final layer of sigmoid function
- **Adam :** extended stochastic gradient descent algorithm utilizing adaptive moment estimates
- **Model.compile :** configures the model for training with arguments (optimizer = Adam gradient descent, loss = binary cross entropy, metrics = accuracy dictionary from sklearn library)

```
# Then a dense layer
class1 = Dense(64, activation='relu', kernel_initializer='he_uniform')(flat2)
# And the output layer
class1b = Dense(1, activation='linear')(class1)
# The output needs to be binary, so we apply a sigmoid function
output = sigmoid(class1b)

# Here is where the model is created based on the input and output define above
model = Model(inputs=x1, outputs=output)

# We choose an optimizer
opt = Adam(learning_rate=0.0005)

# The last step is to compile the model
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])

# We can see the structure and number of parameter of our network
# by calling .summary()
model.summary()
```

CNN Model Outline



CNN Model Summary

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 331, 331, 3)]	0
conv2d (Conv2D)	(None, 329, 329, 16)	448
max_pooling2d (MaxPooling2D)	(None, 164, 164, 16)	0
conv2d_1 (Conv2D)	(None, 162, 162, 16)	2320
max_pooling2d_1 (MaxPooling2	(None, 81, 81, 16)	0
conv2d_2 (Conv2D)	(None, 79, 79, 32)	4640
max_pooling2d_2 (MaxPooling2	(None, 39, 39, 32)	0
conv2d_3 (Conv2D)	(None, 37, 37, 32)	9248
max_pooling2d_3 (MaxPooling2	(None, 18, 18, 32)	0
conv2d_4 (Conv2D)	(None, 16, 16, 64)	18496
max_pooling2d_4 (MaxPooling2	(None, 8, 8, 64)	0
conv2d_5 (Conv2D)	(None, 6, 6, 64)	36928
max_pooling2d_5 (MaxPooling2	(None, 3, 3, 64)	0
flatten (Flatten)	(None, 576)	0
dropout (Dropout)	(None, 576)	0
dense (Dense)	(None, 64)	36928
dense_1 (Dense)	(None, 1)	65
tf.math.sigmoid (TFOpLambda)	(None, 1)	0

Total params: 109,073
Trainable params: 109,073
Non-trainable params: 0

First Run Errors and how we fixed them

Original version

```
Tensor Flow Version: 1.15.0
Keras Version: 2.2.4

Python 3.7.13 (default, Mar 28 2022, 07:24:34)
[Clang 12.0.0 ]
Pandas 1.3.5
Scikit-Learn 1.0.2
```

```
TRAIN_DATA_DIR = r'Train/'
TEST_DATA_DIR = r'Test/'
```

```
opt = Adam(lr=0.0005)
```

Our version

```
Tensor Flow Version: 2.9.1
Keras Version: 2.9.0

Python Version: 3.10.4 | packaged by conda-forge |
Pandas Version: 1.4.2
Scikit-Learn Version: 1.1.1
```

```
TRAIN_DATA_DIR = r'Train/Train'
TEST_DATA_DIR = r'Test/Test'
```

```
opt = Adam(learning_rate=0.0005)
```

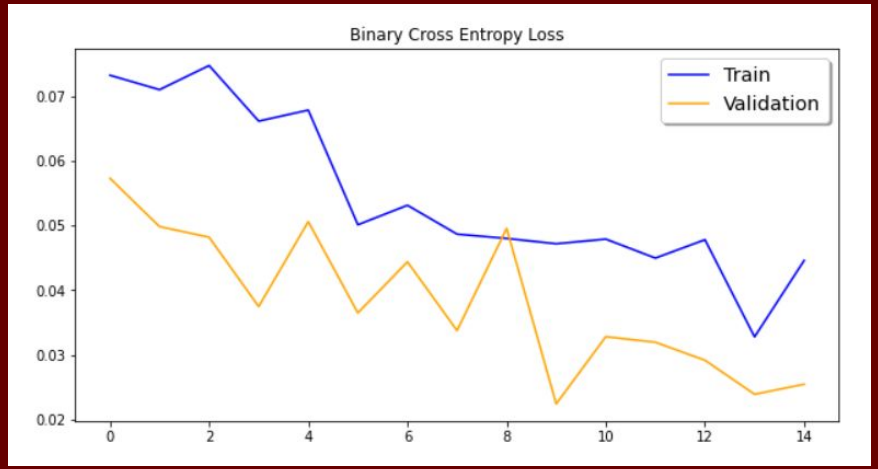
Training the Network

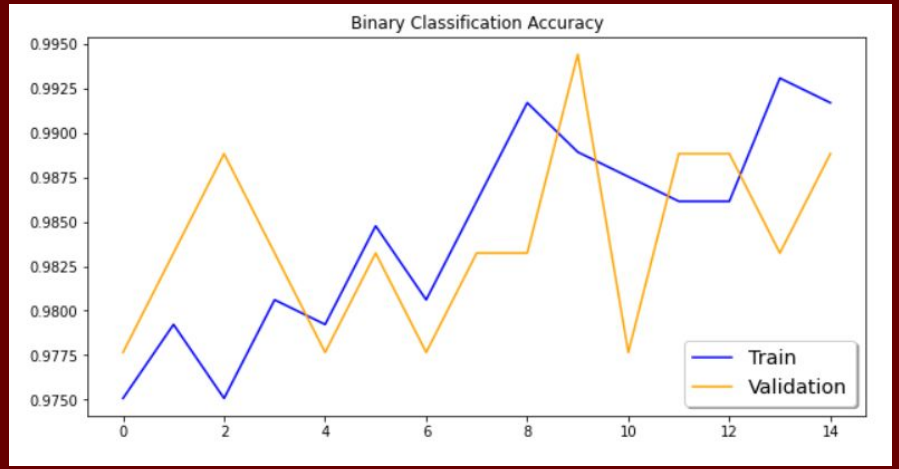
```
# The .fit() method is used to train our network
# You can specify here the number os epochs
history = model.fit(train_it, steps_per_epoch=len(train_it),
                              validation_data=val_it, validation_steps=len(val_it),
                              epochs=15, verbose=1)

plot_learning_curves(history)
```

- **train_it** : array of training data to feed our model
- **steps_per_epoch** : total number of steps (batches of samples) to yield from generator before declaring one epoch done.
- **validation_data** : array of data to evaluate the loss and any model metrics at the end of each epoch.
- **validation_steps** : total number of steps to validate before stopping.
- **epochs** : Number of epochs to train the model. (Go until this index is reached)
- **verbose** : Verbosity mode (1 means show progress bar while training)

Continued (Not a Great Training)

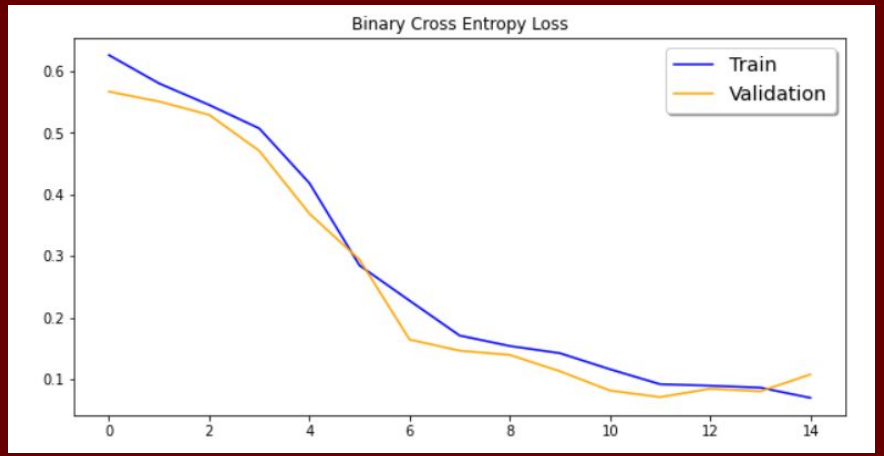
```
Epoch 12/15
12/12 [==============================] - 33s 3s/step - loss: 0.0449 - accuracy: 0.9861 - val_loss: 0.0319 - val_accuracy: 0.
9888
Epoch 13/15
12/12 [==============================] - 33s 3s/step - loss: 0.0478 - accuracy: 0.9861 - val_loss: 0.0291 - val_accuracy: 0.
9888
Epoch 14/15
12/12 [==============================] - 33s 3s/step - loss: 0.0328 - accuracy: 0.9931 - val_loss: 0.0239 - val_accuracy: 0.
9832
Epoch 15/15
 5/12 [===========>.................] - ETA: 18s - loss: 0.0656 - accuracy: 0.9875
```

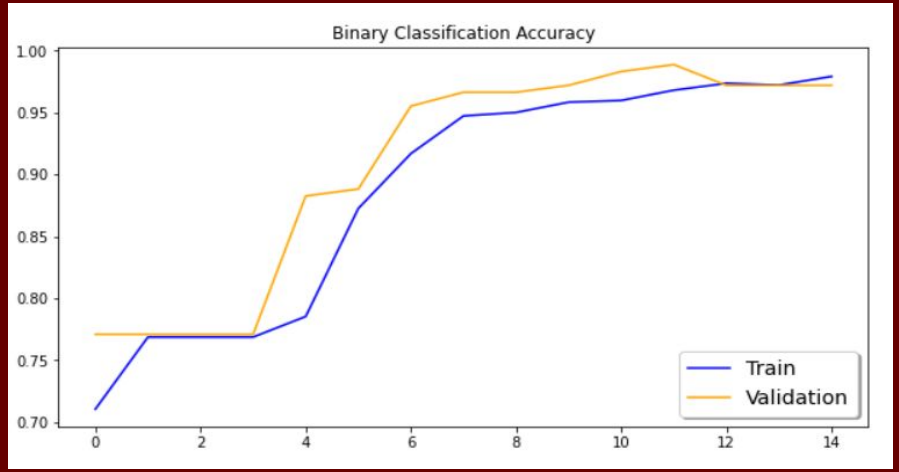




Continued (Better the Second Time)

```
Epoch 12/15
12/12 [==============================] - 34s 3s/step - loss: 0.0924 - accuracy: 0.9681 - val_loss: 0.0714 - val_accuracy: 0.
9888
Epoch 13/15
12/12 [==============================] - 34s 3s/step - loss: 0.0900 - accuracy: 0.9737 - val_loss: 0.0850 - val_accuracy: 0.
9721
Epoch 14/15
12/12 [==============================] - 33s 3s/step - loss: 0.0868 - accuracy: 0.9723 - val_loss: 0.0808 - val_accuracy: 0.
9721
Epoch 15/15
 3/12 [======>.....................] - ETA: 14s - loss: 0.0585 - accuracy: 0.9863
```





Predict the validation set/ performance metrics

Within the validation set:

- Get all of the true y values (Y)
- Get all of the predicted y values ($\hat{Y}$)
- Run the Network Model and get all of the x values used to make predictions to test the accuracy of the model.
- Continue until we reach the end of the validation set
- Store the Y and X values in numpy arrays

```
i=0
y_true = []
y_pred = []
x_test = []

for x, y in val_it:
    y_true.extend(y)
    y_pred.extend(model.predict(x))
    x_test.extend(x)
    i+=1
    if i==len(val_it):
        break

y_pred = np.asarray(y_pred)
x_test = np.asarray(x_test)
```

```
2/2 [==============================] - 1s 246ms/step
2/2 [==============================] - 0s 227ms/step
2/2 [==============================] - 0s 142ms/step
```

Plot the ROC Curve for Validation Data

```
fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_true, y_pred)

auc_keras = auc(fpr_keras, tpr_keras)
```

From the sklearn library

roc_curve : Compute Receiver Operating Characteristic (ROC).

– Parameters –

- **y_true** : True binary labels (Values are expected to be 0 or 1) (Expectation – Y)
- **y_pred** : Target scores (Values determined by the network) (Reality – $\hat{Y}$)

– Returns –

- **fpr_keras** : Array of increasing false positive rates
- **tpr_keras** : Array of increasing true positive rates
- **thresholds_keras** : Array of decreasing thresholds on the decision function used to compute fpr and tpr.

Plot the ROC Curve for Validation Data

```
fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_true, y_pred)

auc_keras = auc(fpr_keras, tpr_keras)
```

From the keras library

auc : Approximates the AUC (Area under the curve) of the ROC

– Parameters –

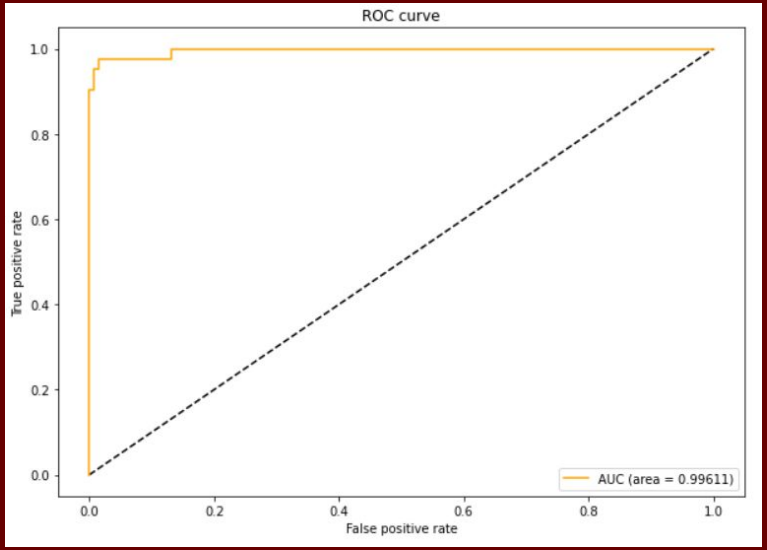
- **fpr_keras** : Array of increasing false positive rates
- **tpr_keras** : Array of increasing true positive rates

– Returns –

- **auc_keras** : an approximation of the area under the positive rates curve

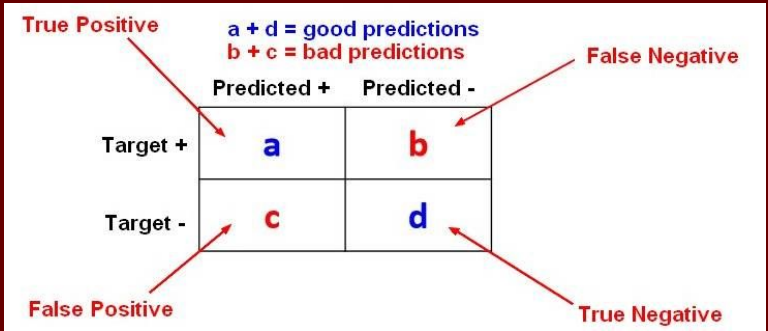
Plot the ROC Curve for Validation Data

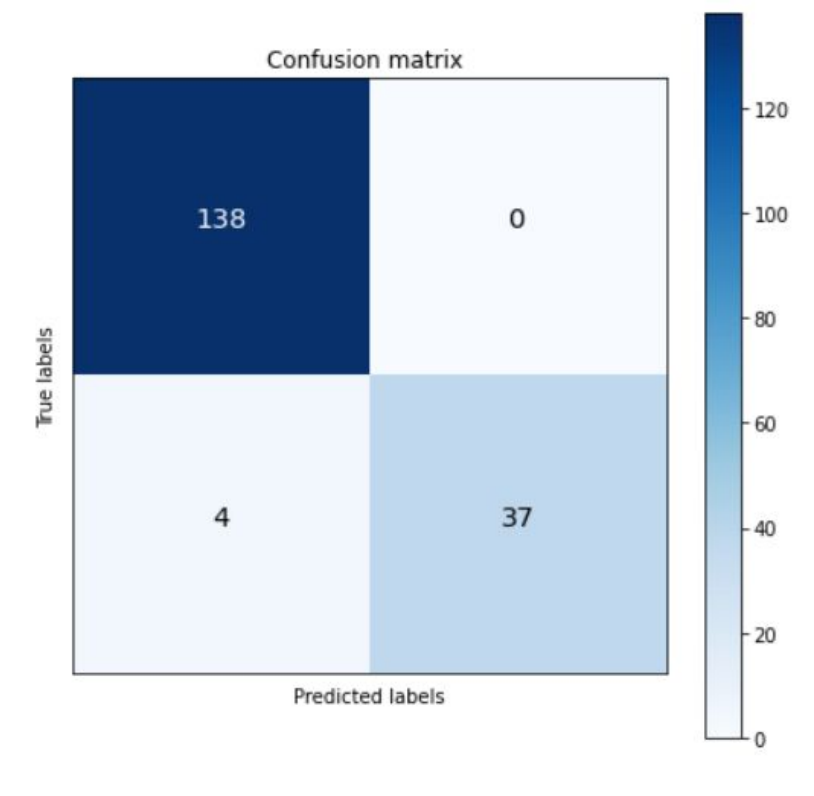
```
pyplot.figure(figsize=(10,7))
pyplot.plot([0, 1], [0, 1], 'k--')
pyplot.plot(fpr_keras, tpr_keras, label='AUC (area = {:.5f})'.format(auc_keras), color='orange')
pyplot.xlabel('False positive rate')
pyplot.ylabel('True positive rate')
pyplot.title('ROC curve')
pyplot.legend(loc='best')
pyplot.show()
```



Plot Confusion Matrix for Validation

```
thresh = 0.5
cm = confusion_matrix(y_true, y_pred > thresh)

pyplot.figure(figsize=(7,7))
pyplot.imshow(cm, cmap=pyplot.cm.Blues)
pyplot.xlabel("Predicted labels")
pyplot.ylabel("True labels")
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    pyplot.text(j, i, cm[i, j],
                horizontalalignment="center",
                color="white" if cm[i, j] > 120 else "black", size='x-large')
pyplot.xticks([], [])
pyplot.yticks([], [])
pyplot.title('Confusion matrix ')
pyplot.colorbar()
pyplot.show()
```





Predict the Test Set

```
ext_val_datagen = ImageDataGenerator(rescale=1. / 255,
                                     #shear_range=0.2,
                                     #zoom_range=0.2,
                                     #horizontal_flip=False,
                                     #vertical_flip=True,
                                     #rotation_range=0,
                                     #fill_mode='constant',
                                     #cval=0,
                                     #preprocessing_function=preprocess_input,
                                     )  # set validation split

ext_val_it = ext_val_datagen.flow_from_directory(
    TEST_DATA_DIR,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    color_mode='rgb')  # set as training data
```

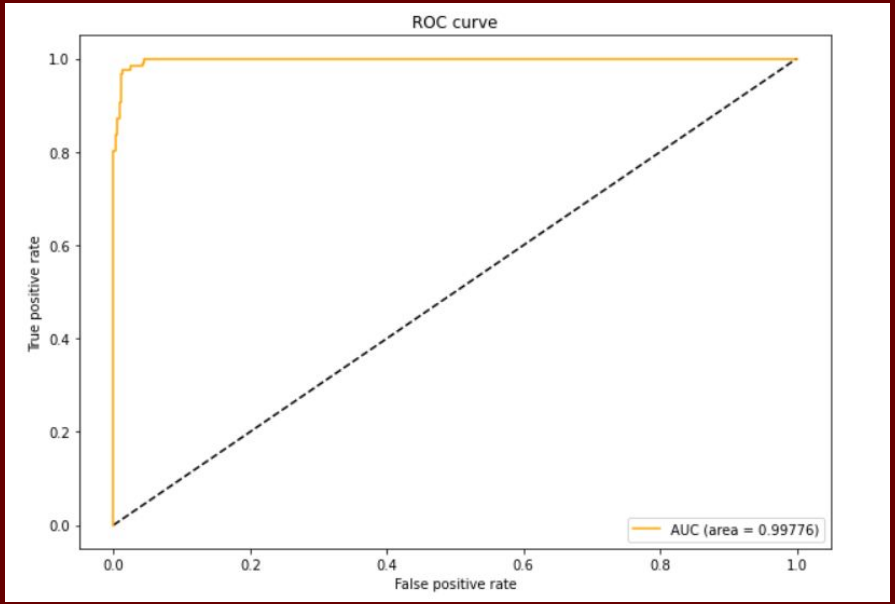
```
Found 571 images belonging to 2 classes.
```

```
i=0
y_true = []
y_pred = []
x_test = []

for x, y in ext_val_it:

    y_true.extend(y)
    y_pred.extend(model.predict(x))
    x_test.extend(x)
    i+=1
    if i==len(ext_val_it):
        break

y_pred = np.asarray(y_pred)
x_test = np.asarray(x_test)
```

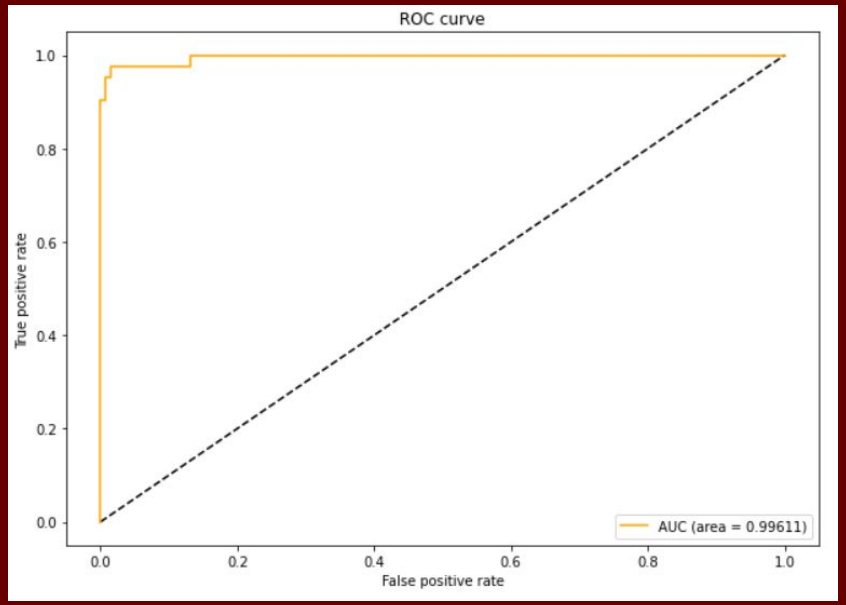
```
2/2 [==============================] - 1s 248ms/step
2/2 [==============================] - 0s 220ms/step
2/2 [==============================] - 1s 311ms/step
2/2 [==============================] - 0s 227ms/step
2/2 [==============================] - 1s 283ms/step
2/2 [==============================] - 0s 233ms/step
2/2 [==============================] - 0s 250ms/step
2/2 [==============================] - 1s 259ms/step
2/2 [==============================] - 0s 230ms/step
```

Plot ROC curve for Test Set

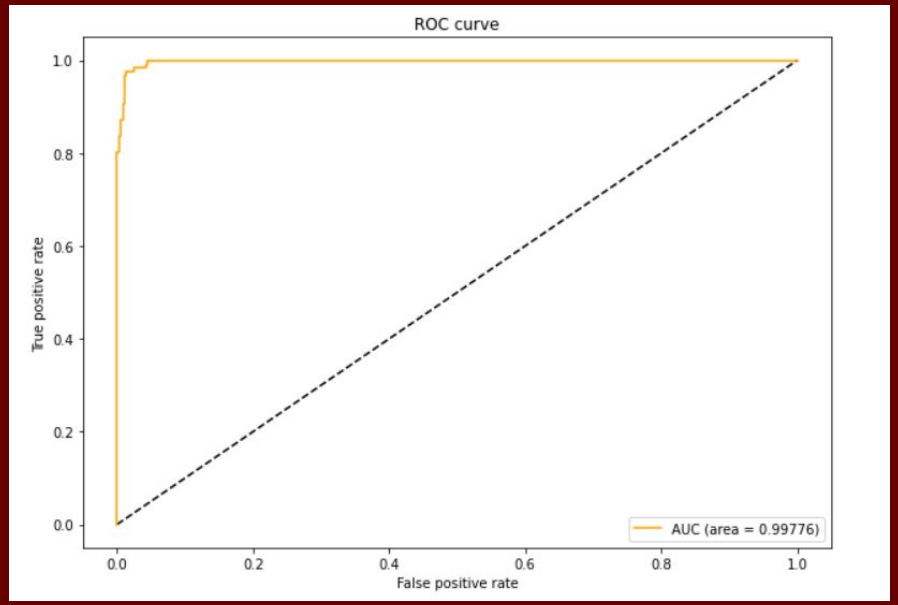
```
fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_true, y_pred)

auc_keras = auc(fpr_keras, tpr_keras)

pyplot.figure(figsize=(10,7))
pyplot.plot([0, 1], [0, 1], 'k--')
pyplot.plot(fpr_keras, tpr_keras, label='AUC (area = {:.5f})'.format(auc_keras), color='orange')
pyplot.xlabel('False positive rate')
pyplot.ylabel('True positive rate')
pyplot.title('ROC curve')
pyplot.legend(loc='best')
pyplot.show()
```



Comparison ROC of Validation With Test



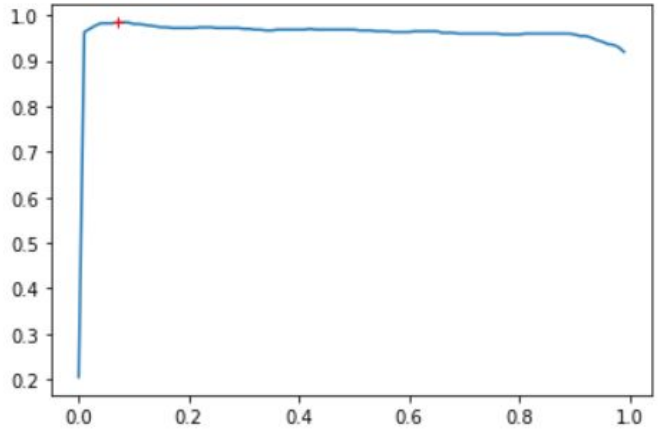
Validation Data Results



Test Data Results

Find threshold with the best accuracy

```
thr_list = []
acc_list = []
for _th in range(100):
    _th = _th / 100.
    thr_list.append(_th)
    acc_list.append(accuracy_score(y_true, y_pred > _th))

pyplot.figure()
pyplot.plot(thr_list,acc_list)
pyplot.plot(thr_list[acc_list.index(max(acc_list))], max(acc_list), 'r+')

pyplot.show()
```

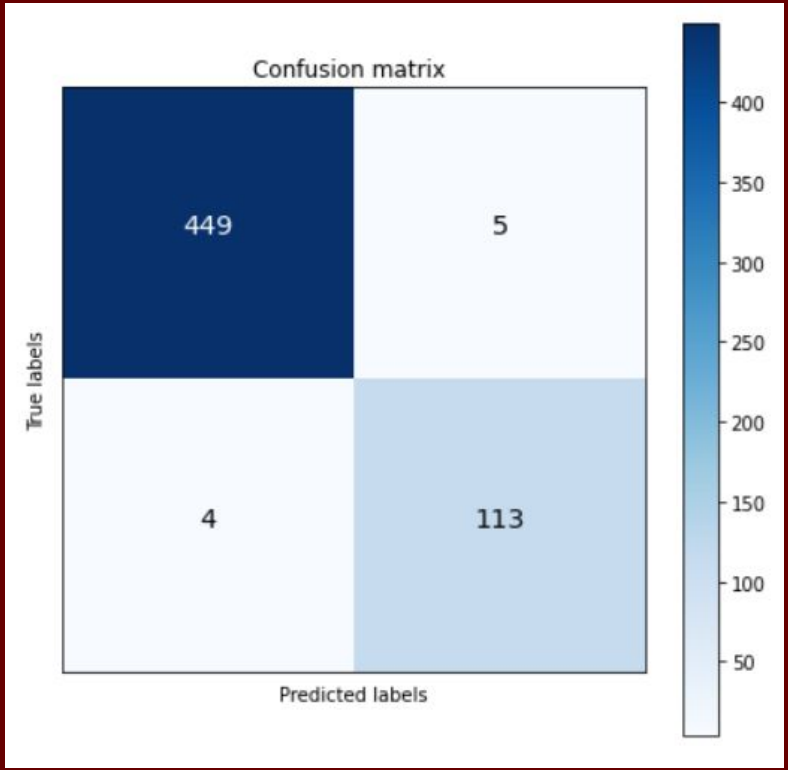


$$\boxed{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

What range was our network the most accurate when giving out its predictions

Plot the Confusion Matrix for the Test Set

```
thresh = thr_list[acc_list.index(max(acc_list))]
cm = confusion_matrix(y_true, y_pred > thresh)

pyplot.figure(figsize=(7,7))
pyplot.imshow(cm, cmap=pyplot.cm.Blues)
pyplot.xlabel("Predicted labels")
pyplot.ylabel("True labels")
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    pyplot.text(j, i, cm[i, j],
             horizontalalignment="center",
             color="white" if cm[i, j] > 120 else "black", size='x-large')
pyplot.xticks([], [])
pyplot.yticks([], [])
pyplot.title('Confusion matrix ')
pyplot.colorbar()
pyplot.show()
```

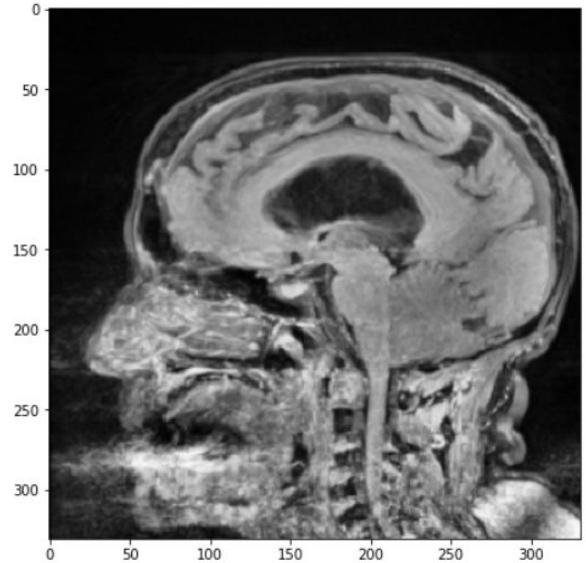


Displaying Cases Model Predicted Incorrectly

```
for i in range(len(y_true)):
    if y_true[i] != 1. * (y_pred[i, 0] > thresh):
        print('Truth:' + str(y_true[i]))
        print('Pred:' + str(y_pred[i, 0]))
        print(i)
        pyplot.figure(figsize=(7,7))
        pyplot.imshow(x_test[i])
        pyplot.show()
```

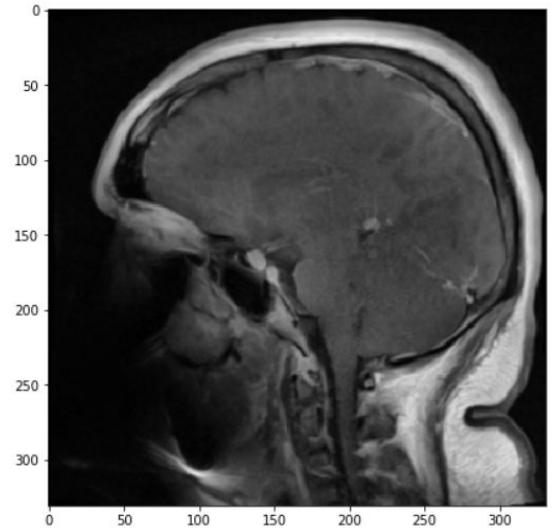
1.

```
Truth:0.0
Pred:0.41629934
64
```



2.

```
Truth:1.0
Pred:0.028921802
187
```

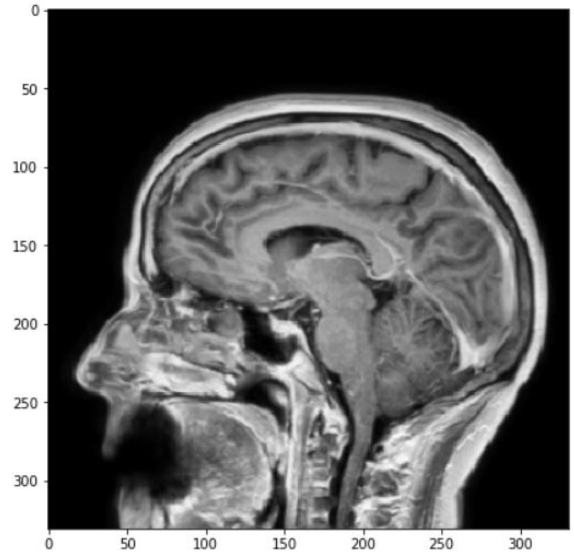


Display Cases Model Predicted Incorrectly

```
for i in range(len(y_true)):
    if y_true[i] != 1. * (y_pred[i, 0] > thresh):
        print('Truth:' + str(y_true[i]))
        print('Pred:' + str(y_pred[i, 0]))
        print(i)
        pyplot.figure(figsize=(7,7))
        pyplot.imshow(x_test[i])
        pyplot.show()
```

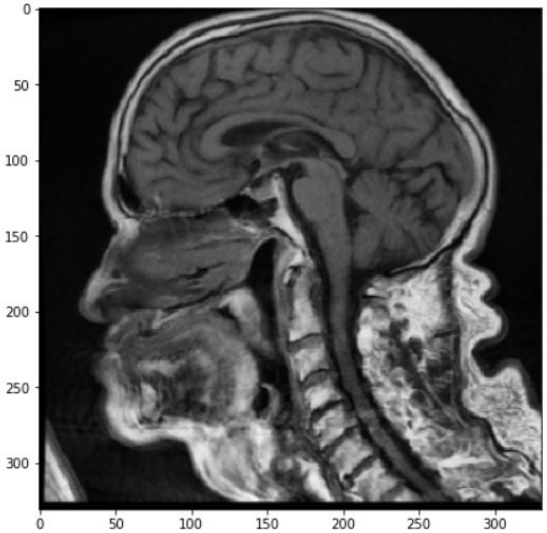
3.

```
Truth:1.0
Pred:0.012690382
211
```



4.

```
Truth:0.0
Pred:0.35805246
239
```

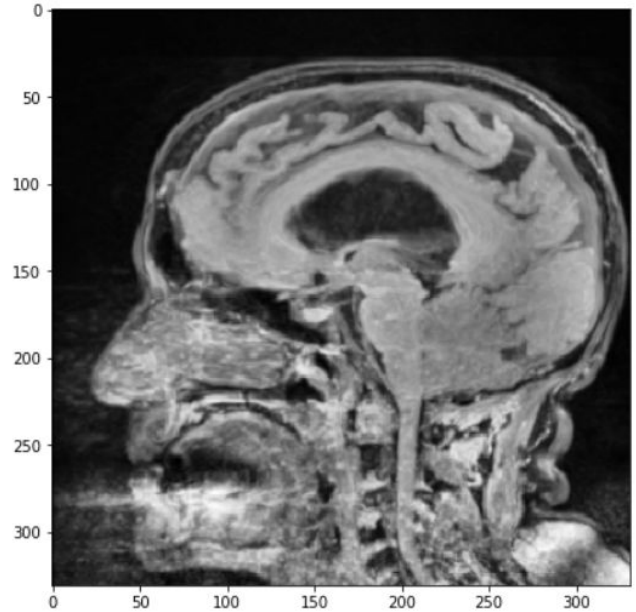


Display Cases Model Predicted Incorrectly

```
for i in range(len(y_true)):
    if y_true[i] != 1. * (y_pred[i, 0] > thresh):
        print('Truth:' + str(y_true[i]))
        print('Pred:' + str(y_pred[i, 0]))
        print(i)
        pyplot.figure(figsize=(7,7))
        pyplot.imshow(x_test[i])
        pyplot.show()
```

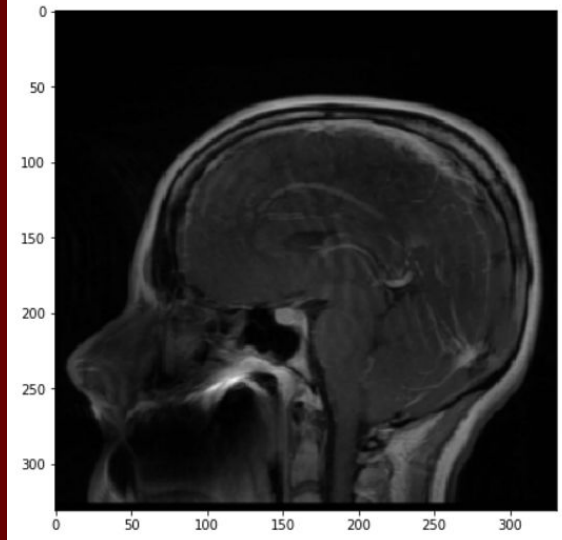
5.

```
Truth:0.0
Pred:0.21813016
309
```

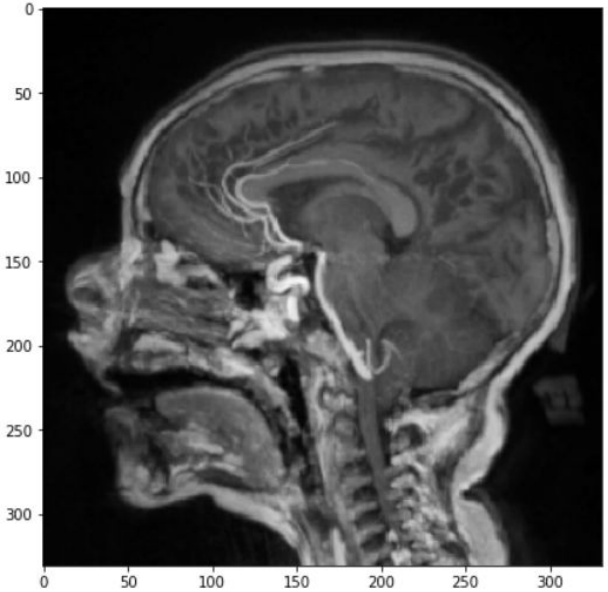


6.

```
Truth:1.0
Pred:0.056852665
330
```



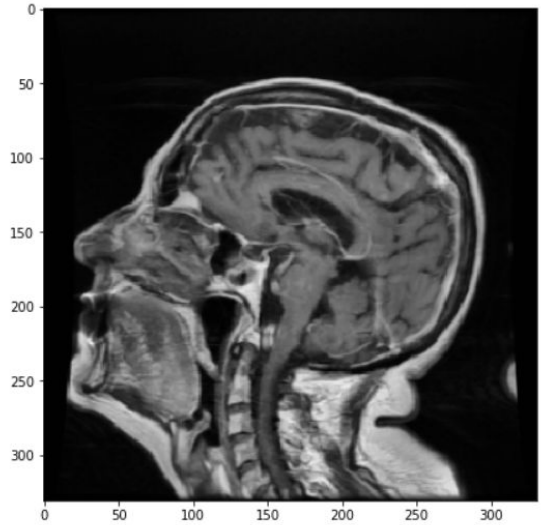
Truth:0.0
Pred:0.60006547
426



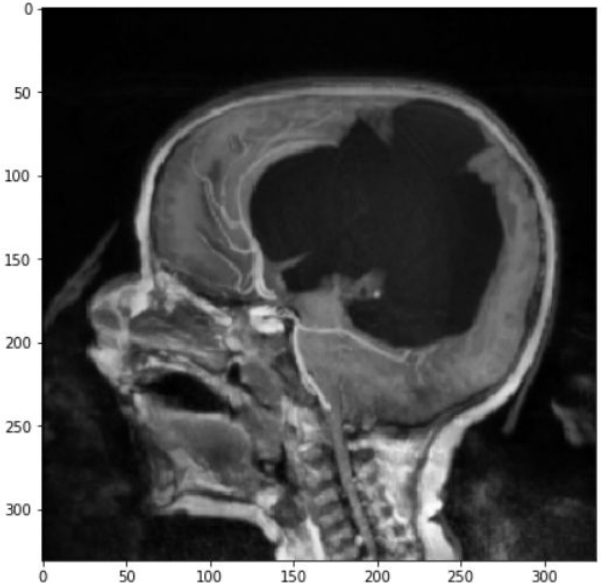
7.

8.

Truth:1.0
Pred:0.011222316
487



Truth:0.0
Pred:0.8044872
544



9.

Conclusion

- Medical Imaging is a productive field to be considered for application of deep learning techniques
- Binary Classification can be used to determine whether a T1 MRI scan consists of Pre- or Post-Contrast
- Each time the CNN is fitted and trained, its threshold and accuracy varies despite using the same training, validation, and testing data.
- The model that we fit to observe T1 MRI scans did contain perfect predictions; however, its errors were significantly lower than its correct guesses during the majority of running the notebook
- The model gives insight into how a CNN operates and the methods and libraries used in standard practices to produce statistical observations pertaining to medical imaging

Questions?

References

https://github.com/ImagingInformatics/machine-learning/blob/master/Education/KerasBinaryClassifier/SIIM_Keras_Binary_Classifier.ipynb

https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2698108/

https://machinelearningmastery.com/threshold-moving-for-imbalanced-classification/

https://www.geeksforgeeks.org/keras-conv2d-class/

https://data-flair.training/blogs/keras-convolution-neural-network/

https://towardsai.net/p/machine-learning/beginner-guides-to-convolutional-neural-network-from-scratch-kuzushiji-mnist-75f42c175b21