M3 Group Assignment - X-rays

By: Jonas Røge Jepsen, Michael Bering Olesen and Tobias Maltha Christensen

We recommend that you read this notebook on collab and not in the PDF. Furthermore the computation time for the colab is ~ 1 hour.

Links:

- **Colab:** https://colab.research.google.com/drive/1UzyrNF8QVWeDDKjYY3kd33yaGmq4qf7a
- **Kaggle:** https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia
- **Github:** https://github.com/TobiasMaltha/m3

We are working with X-ray images of childrens lungs, and we want to predict whether the children have pneumonia (inflammation of the lung parenchyma) or have "normal" lungs.

Getting the data

In [7]:

```
!pip install kaggle # Install kaggle module
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.6/dist-packages (1.5.6)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.6/dist-packages (
from kaggle) (4.0.0)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/dist-pac
kages (from kaggle) (1.24.3)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.6/dist-packages (from
kaggle) (1.12.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.6/dist-packages
(from kaggle) (2.6.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.6/dist-packages (from ka
ggle) (2019.9.11)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from k
aggle) (2.21.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages (from kaggl
e) (4.28.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.6/dist-packa
ges (from python-slugify->kaggle) (1.3)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-packages (
from requests->kaggle) (2.8)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-pac
kages (from requests->kaggle) (3.0.4)
```

Here you have to import a token API, which you collect from your account at kaggle.com If you need further help or explaination. Go to this help PDF:
https://github.com/TobiasMaltha/m3/blob/master/token%20api%20guide%20pdf.pdf

In [8]:

```
from google.colab import files
files.upload() # Upload your kaggle.json here
```

Choose File **No file selected**

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

Out[8]:

```
{'kaggle.json': b'{"username":"roede9","key":"7584213262d255cf00cccd891ba65eb8"}'}
```

In [0]:

```
# Before importing the dataset we want to use this code
# The Kaggle API client expects this file to be in ~/.kaggle,
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/

# This permissions change avoids a warning on Kaggle tool startup.
!chmod 600 ~/.kaggle/kaggle.json
```

In [10]:

```
!kaggle datasets download -d paultimothymooney/chest-xray-pneumonia # Downloads the data set
```

```
Downloading chest-xray-pneumonia.zip to /content
100% 2.29G/2.29G [00:28<00:00, 70.5MB/s]
100% 2.29G/2.29G [00:28<00:00, 86.0MB/s]
```

In [11]:

```
!ls # Shows the files we have in our directorary
```

```
chest-xray-pneumonia.zip  kaggle.json  sample_data
```

In [0]:

```
!unzip -qq chest-xray-pneumonia.zip # Unzips the zipfile with the dataset. OBS if this is the second or more times your have unzipped, you have to write: 'N' and press enter
```

In [13]:

```
# Import packages
import pandas as pd
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
from keras.preprocessing.image import ImageDataGenerator
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info.

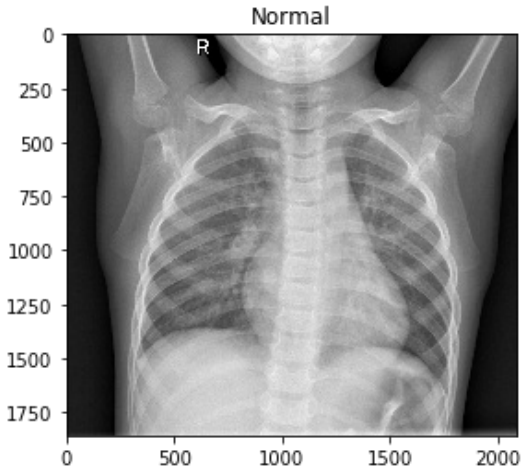
EDA

The data: The dataset used in this assignment contains 5863 X-ray images of childrens lungs selected from a women and children's medical center in Guangzhou, China. The data is available from Kaggle and is organized into to three folders: Training, Test and Validation. Each of these folders contain subfolders which categorizes the images into the patient having "normal" lungs or having pneumonia. All the images have been screened for quality where low quality and unreadable scans have been removed. Furthermore, in order to secure the quality several experts have gone through the images to make sure they were labeled correctly.

In [14]:

```
picturenm = '/content/chest_xray/train/NORMAL/IM-0115-0001.jpeg' # Creating a varible that is a path to an image
image = Image.open(picturenm).convert("L") # Use the Image function to go to the path an open the image. The "L" converts it to grayscale.
arr = np.asarray(image) # Convert to an array
```

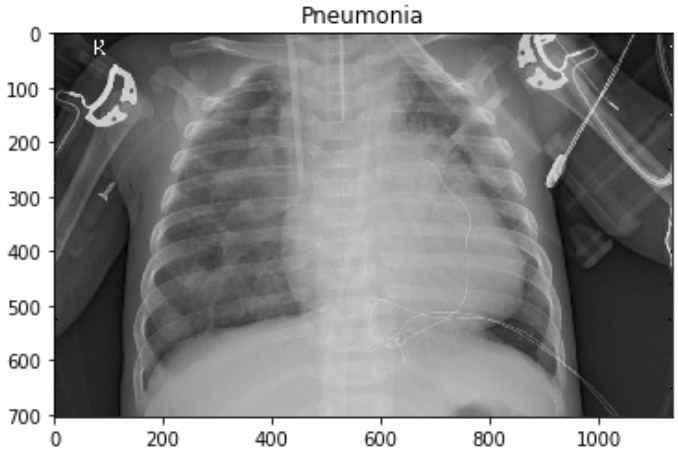
```
plt.imshow(arr, cmap='gray', vmin=0, vmax=255) # Display image
plt.title('Normal') # Give the image a title
plt.show() # Shows the image
```



This is a x-ray of a "normal" set of lungs.

In [15]:

```
picturepm = '/content/chest_xray/train/PNEUMONIA/person1024_bacteria_2955.jpeg' # Creatin
g a varible that is a path to an image
image = Image.open(picturepm).convert("L") # Use the Image function to go to the path an
open the image. The "L" converts it to grayscale.
arr = np.asarray(image) # Convert to an array
plt.imshow(arr, cmap='gray', vmin=0, vmax=255) # Display image
plt.title('Pneumonia') # Give the image a title
plt.show() # Shows the image
```



This is a x-ray of lungs with pneumonia. If you look at both pictures, it is difficult to see the difference of the pictures, other than the small details. (Except if you are a doctor of course).

In [0]:

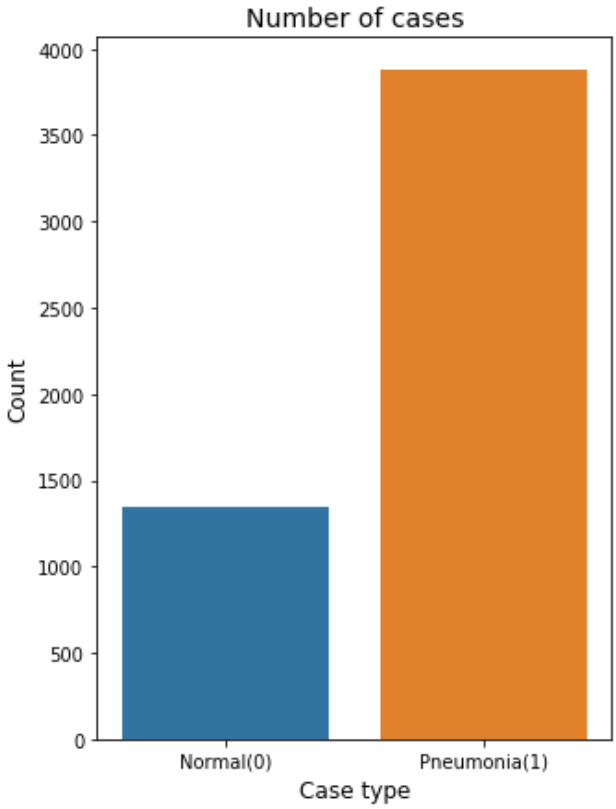
```
datagen = ImageDataGenerator()

def show_cases (path): #Creating a method that makes it possible to plot the images for e
ach of
#the 3 parts of the dataset, and shows the amount of each label
  graphdata = datagen.flow_from_directory(path) #Saves the images from the "path"
  cases_count_s = pd.Series(graphdata.labels) #Converts to a series which allows value c
ounts
  cases_count= cases_count_s.value_counts() #Value counts the cases
  # Plot the results
  plt.figure(figsize=(5,7))
  sns.barplot(x=cases_count.index, y= cases_count.values)
  plt.title('Number of cases', fontsize=14)
  plt.xlabel('Case type', fontsize=12)
  plt.ylabel('Count', fontsize=12)
```

```
  plt.xticks(range(len(cases_count.index)), ['Normal(0)', 'Pneumonia(1)'])
  plt.show()
```

In [17]:

```
show_cases('/content/chest_xray/chest_xray/train') # Plotting the amount of cases from th
e train-set
```

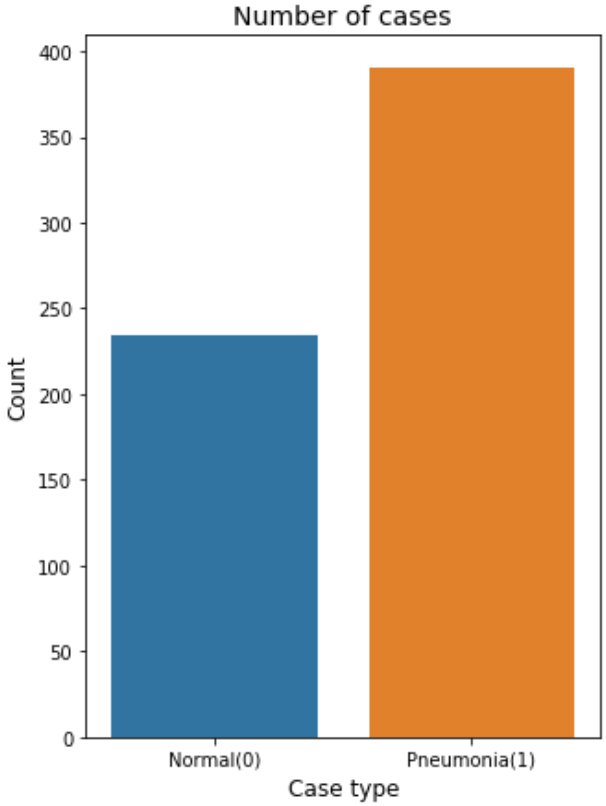
```
Found 5216 images belonging to 2 classes.
```



In [18]:

```
show_cases('/content/chest_xray/chest_xray/test') #Plotting the amount of cases from the
test-set
```

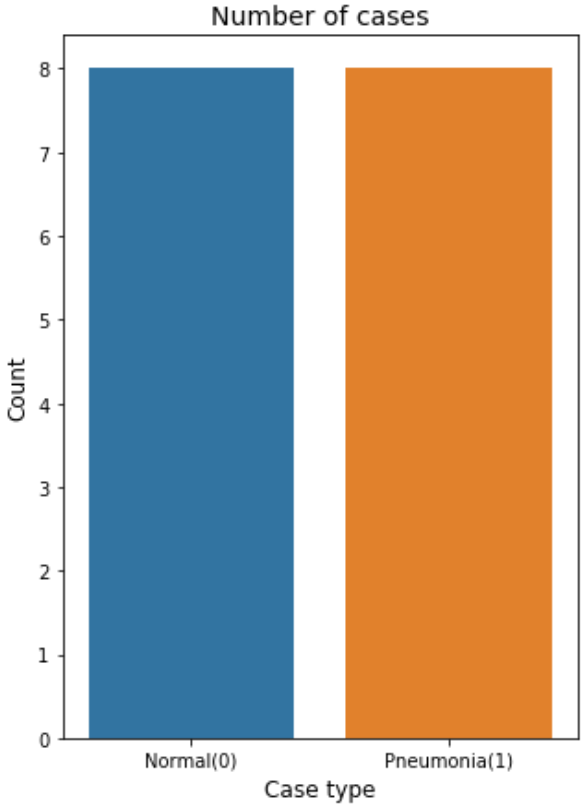
```
Found 624 images belonging to 2 classes.
```



In [19]:

```
show_cases('/content/chest_xray/chest_xray/val') #Plotting the amount of cases from the val-set
```

```
Found 16 images belonging to 2 classes.
```



As can be seen above the training part of dataset consists of 1341 images of ”normal” lungs and 3875 images of lungs with Pneumonia. The distribution between the two is a little unequal, but there should be enough data for it to work. The same seems to be the case with the test part, here there is 234 images of ”normal” lungs and 390 images of lungs with Pneumonia. The validation part of the dataset only consist of 16 images 8 of each class, and this seems to be too small to be useful, and for that reason we will not use this part of the dataset, instead we use a small part of the test part of the dataset as validation set, but more on this later.

KNN (Non-neural)

We use KNN (k-Nearest Neighbor) because it is one of the most simple machine learning classification algorithms and because it can be used in relation to image classification.

In [0]:

```
# Import packages
from sklearn.neighbors import KNeighborsClassifier
import imutils
from imutils import paths
import cv2
import os
```

Since we are working with non-neural-network, we have to preprocess the data in a way so the KNN can interpret it and work with it

In [0]:

```
# This is a minor helper method that will resize the input image, and
# afterwards flatten the image into a single list of pixel intensities.
# In this case we reseize the image to 32x32 pixels, if nothing else is
# specified.
def image_to_feature_vector(image, size=(32, 32)):
 return cv2.resize(image, size).flatten()
```

In [0]:

```
# Gets the paths to all images in both training and test as a list
train_imagePaths = list(paths.list_images("/content/chest_xray/train"))
test_imagePaths = list(paths.list_images("/content/chest_xray/test"))

# Here we initialize lists for the rawImages(pixel intensities) and labels for
# both train and test sets, that we will use later
train_rawImages = []
train_labels = []

test_rawImages = []
test_labels = []
```

In [23]:

```
# Here we loop over the input images of train to extract the labels
# and pixel intensities
for (i, imagePath) in enumerate(train_imagePaths):
 image = cv2.imread(imagePath) # load the image
 label = imagePath.split(os.path.sep)[-2].split("/")[0] # extract class label
  # The path looks like this: /content/chest_xray/train/NORMAL/IM-0119-0001.jpeg
 # and the above code vil then extract the label between the last two backslahes

 pixels = image_to_feature_vector(image) # Extract the raw pixel intensity using the earl
ier defined helper method

 # Add the extracted labels and pixel intensities to the two lists
 train_rawImages.append(pixels)
 train_labels.append(label)

 # The above will take some time to compute and this will give an indication of
 # the progress by showing an update every 500 images
 if i > 0 and i % 500 == 0:
  print("[INFO] processed {}/{}".format(i, len(train_imagePaths)))
```

```
[INFO] processed 500/5216
[INFO] processed 1000/5216
[INFO] processed 1500/5216
[INFO] processed 2000/5216
[INFO] processed 2500/5216
[INFO] processed 3000/5216
[INFO] processed 3500/5216
[INFO] processed 4000/5216
[INFO] processed 4500/5216
[INFO] processed 5000/5216
```

In [24]:

```
# This is the same as the above for loop, but here we loop over the input images
# of test instead, to extract the labels and pixel intensities of these images
for (i, imagePath) in enumerate(test_imagePaths):
 image = cv2.imread(imagePath) # load the image
 label = imagePath.split(os.path.sep)[-2].split("/")[0] #extract class label
  # The path looks like this: /content/chest_xray/test/NORMAL/IM-0007-0001.jpeg
 # and the above code vil then extract the label between the last two backslahes

 pixels = image_to_feature_vector(image) # Extract the raw pixel intensity using the earl
ier defined helper method

 # Add the extracted labels and pixel intensities to the two lists
 test_rawImages.append(pixels)
 test_labels.append(label)

 # The above will take some time to compute and this will give an indication of
 # the progress by showing an update every 200 images
 if i > 0 and i % 200 == 0:
  print("[INFO] processed {}/{}".format(i, len(test_imagePaths)))
```

```
[INFO] processed 200/624
[INFO] processed 400/624
```

```
[INFO] processed 600/624
```

With the above preprossing in place we are now ready to train and evaluate the KNN model

In [25]:

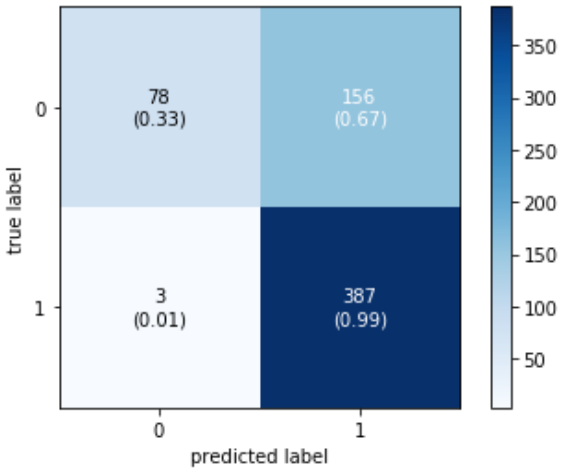
```
# Creating a KNN model with N_neighbors set to 3 and N_jobs to 1
knn = KNeighborsClassifier(n_neighbors=3, n_jobs=1)

# Fit the model on the pixel intensities (as X) and labels (as y)
knn.fit(train_rawImages, train_labels)

# Evaluating the model
acc = knn.score(test_rawImages, test_labels)
print("the accuracy is: {:.2f}%".format(acc * 100))
```

```
the accuracy is: 74.52%
```

In [26]:

```
y_pred = knn.predict(test_rawImages) # Use predict on the model with the test_set
confmatrix = confusion_matrix(test_labels, y_pred) # Create a confusion matrix
plot_confusion_matrix(conf_mat=confmatrix, colorbar=True, show_absolute=True, show_normed
=True) # Plotting the confusions matrix
print(classification_report(test_labels, y_pred)) # Printing the classification report
```

```
              precision    recall  f1-score   support

      NORMAL       0.96      0.33      0.50       234
   PNEUMONIA       0.71      0.99      0.83       390

    accuracy                           0.75       624
   macro avg       0.84      0.66      0.66       624
weighted avg       0.81      0.75      0.70       624
```



As you can see the KNN give as surprising good result. It has a low number of false negatives and high number of true positives, however it takes some extra guesses when it comes to false positives, which isn't great. What we want is both a high number of true negatives and true positives for us to use it in a real situation in a hospital. Hopefully our deep learn model will perform better than the accuracy of KNN, which is 0.75.

To improve the KNN model one could:

- **Change the parameters of the model and data**
- **Use hyperparameter tuning to improve the model**
- **Perhaps changing the datasets image pixels size**

Image augmentation

We are using augmentations to increase variations of our training dataset by manipulating the image by flipping,

shearing and zooming them. Which have proofed to give better results with our models

In [0]:

```
image_size = 64 # Setting the pixels size
batch_size = 16 # Setting the batch size
```

In [0]:

```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255, # Normalizing the images
                                   shear_range = 0.2, # Shear the images in differnet di
rections
                                   zoom_range = 0.2, # Zoom in on different spots on the
images
                                   horizontal_flip = True) # flips the images horizontal
ly

test_datagen = ImageDataGenerator(rescale = 1./255, validation_split=0.15) # Normalizing
the images and we take 15 % to use for validation
```

As mentioned earlier the validation set that is included, only consist of 16 pictures and, we find this to be to small to be usefull, so instead we create a validation split above and turn 15 % of test data into validation data. Below we make use of this split by the subset argument of flow_from_directory the create on the val_set and test_set.

In [65]:

```
training_set = train_datagen.flow_from_directory('chest_xray/train', # Choose the path to
the images
                                                 target_size = (image_size, image_size)
, # Sets the pixelsize from the varible
                                                 batch_size = batch_size, # Setting the
batchsize from the varible
                                                 class_mode = 'binary') # Setting class
mode to binary, since we have true or false (1/0)

val_set = test_datagen.flow_from_directory('chest_xray/test', # Choose the path to the im
ages
                                           target_size = (image_size, image_size), # S
ets the pixelsize from the varible
                                           batch_size = batch_size, # Setting the batch
size from the varible
                                           class_mode = 'binary', # Setting class mode
to binary, since we have true or false (1/0)
                                           subset='validation')  # Note that we use a
subset of the test dataset as validation

test_set = test_datagen.flow_from_directory('chest_xray/test', # Choose the path to the i
mages
                                            target_size = (image_size, image_size), # S
ets the pixelsize from the varible
                                            batch_size = batch_size, # Setting the batch
size from the varible
                                            class_mode = 'binary', # Setting class mode
to binary, since we have true or false (1/0)
                                            subset='training', # This is a little unort
hodox naming wise, but to make the split such that most
                                            shuffle=False)     # of the test data is kep
t as test data and not validation this argument is the most fitting.
```

```
Found 5216 images belonging to 2 classes.
Found 93 images belonging to 2 classes.
Found 531 images belonging to 2 classes.
```

As you can see the validation set have increased, while the test set have decreased.

CNN without transfer learning

Convolutional Neural Network (CNN) is well known for its abilities in relation to image recognition and therefore it seems obvious to go with this as our main approach to solving this image classification problem.

In [0]:

```
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```

In [31]:

```
cnn = Sequential() # Creating the model
cnn.add(Conv2D(32, (3, 3), input_shape = (image_size, image_size, 3), activation = 'relu
')) # Creates a convolution kernel, with two dimension and RGB
cnn.add(MaxPooling2D(pool_size = (2, 2))) # Max pooling for preserving features and preve
nt over fitting
cnn.add(Conv2D(32, (3, 3), activation = 'relu')) # Creates a convolution kernel, with tw
o dimension and RBG
cnn.add(MaxPooling2D(pool_size = (2, 2))) # Max pooling for preserving features and preve
nt over fitting
cnn.add(Flatten()) # Flattens the data to one dimension
cnn.add(Dense(units = 128, activation = 'relu')) # A Dense layer with 128 neurons
cnn.add(Dense(units = 1, activation = 'sigmoid')) # An output dense layer with 1 neuron
and the activation set to "Sigmoid", because we are working with a binary classification
problem

cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy']) #C
ompiling the model with Loss set to a binary_crossentropy and optimizer to adam


cnn.summary() # Gives a summary of the model setup
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:66: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_de
fault_graph instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:541: The name tf.placeholder is deprecated. Please use tf.compat.v1.placeholder
instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:4432: The name tf.random_uniform is deprecated. Please use tf.random.uniform in
stead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:4267: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instea
d.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:793: T
he name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead
.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_b
ackend.py:3657: The name tf.log is deprecated. Please use tf.math.log instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops
/nn_impl.py:183: where (from tensorflow.python.ops.array_ops) is deprecated and will be r
emoved in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 62, 62, 32)        896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 29, 29, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 32)        0
```

```
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               802944
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 813,217
Trainable params: 813,217
Non-trainable params: 0
_________________________________________________________________
```

In [32]:

```
!cd chest_xray/train; ls # Show the two types of labels in train
```

```
NORMAL PNEUMONIA
```

In [33]:

```
cnnhistory = cnn.fit_generator(training_set, # Using fit_generator to use the pictures used in datagen
                               epochs = 5, # We use 5 epochs
                               validation_data = val_set, # Validate the results by the validation dataset
                               validation_steps = val_set.samples / batch_size) # Setting validation steps
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1033: The name tf.assign_add is deprecated. Please use tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1020: The name tf.assign is deprecated. Please use tf.compat.v1.assign instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3005: The name tf.Session is deprecated. Please use tf.compat.v1.Session instead.

Epoch 1/5
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:190: The name tf.get_default_session is deprecated. Please use tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:197: The name tf.ConfigProto is deprecated. Please use tf.compat.v1.ConfigProto instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:207: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:216: The name tf.is_variable_initialized is deprecated. Please use tf.compat.v1.is_variable_initialized instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:223: The name tf.variables_initializer is deprecated. Please use tf.compat.v1.variables_initializer instead.

326/326 [==============================] - 71s 217ms/step - loss: 0.3533 - acc: 0.8491 - val_loss: 0.2982 - val_acc: 0.8710
Epoch 2/5
326/326 [==============================] - 63s 193ms/step - loss: 0.2353 - acc: 0.8995 - val_loss: 0.2199 - val_acc: 0.9032
Epoch 3/5
326/326 [==============================] - 64s 196ms/step - loss: 0.2202 - acc: 0.9091 - val_loss: 0.4493 - val_acc: 0.7312
Epoch 4/5
326/326 [==============================] - 64s 197ms/step - loss: 0.1927 - acc: 0.9237 - val_loss: 0.5798 - val_acc: 0.7204
Epoch 5/5
```

```
326/326 [==============================] - 64s 197ms/step - loss: 0.1827 - acc: 0.9333 -
val_loss: 0.2055 - val_acc: 0.9032
```

In [34]:

```
cnn.evaluate_generator(test_set) # Evaluting our trained model with the test_set
```

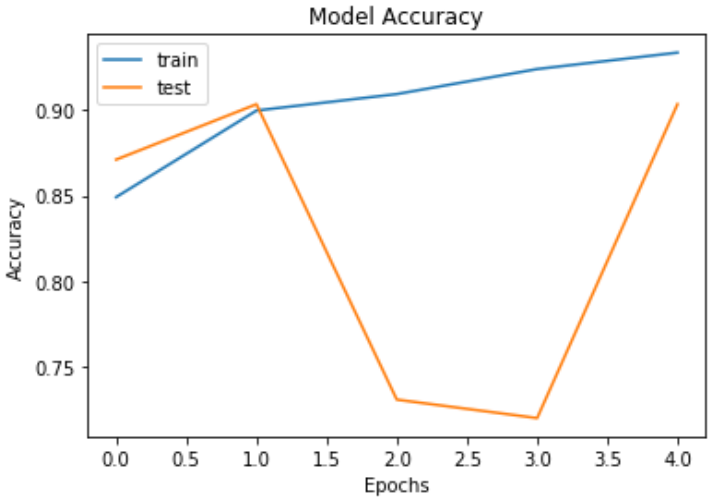
Out[34]:

```
[0.3792536874036892, 0.8531073446327684]
```

The evalute calculates the cnn models to 85.31% accuracy

In [35]:

```
# Creating a plot over each epoch and accuracy
plt.plot(cnnhistory.history['acc']) # Plotting accuracy
plt.plot(cnnhistory.history['val_acc']) # Plotting Value accuracy
plt.title('Model Accuracy') # Give the plot a title
plt.ylabel('Accuracy') # labling the y-axis
plt.xlabel('Epochs') # labeling the x-ais
plt.legend(['train', 'test']) # Name each line in the plot
plt.show() # Show the plot
```

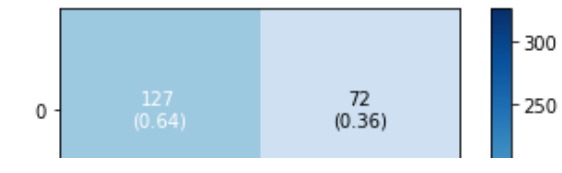


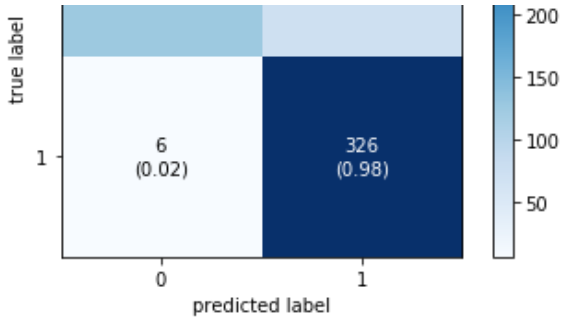
As you can see it is difficult to set the right amount of epochs. However, our expericence through many different tries concluded the epochs amount of 5 to be the best.

In [36]:

```
y_pred = cnn.predict_generator(test_set) # Use predict on the model with the test_set
y_pred = np.where(y_pred >= 0.5, 1, 0) # Setting everything under 0.5 to 0 and over and
equal 0.5 to 1
confmatrix = confusion_matrix(test_set.labels, y_pred) # Create a confusion matrix
plot_confusion_matrix(conf_mat=confmatrix, colorbar=True, show_absolute=True, show_normed
=True) # Plotting the confusions matrix
print(classification_report(test_set.labels, y_pred)) # Printing the classification repor
t
```

```
              precision    recall  f1-score   support

           0       0.95      0.64      0.77       199
           1       0.82      0.98      0.89       332

    accuracy                           0.85       531
   macro avg       0.89      0.81      0.83       531
weighted avg       0.87      0.85      0.85       531
```





The confusion matrix for the CNN shows fine results. However there is still som False negatives and false positives, which needs to be reduced. However, it gets 98% of those with pneumonia correct, but only 64% with True negatives aka "normal" lungs.

VGG16 model

Now we try with a pre-trained model, which is build and trained in advance on 14 million images. VGG16 is also in top 5 in accuracy test, which makes it one of the best and this is why we are using it on the dataset, to maximize the results. We could have used other pre-trained models such as the ResNet-50

In [0]:

```
from keras.applications import VGG16 # Importing the pre-trained model

vgg_conv = VGG16(weights='imagenet', include_top=False, input_shape=(image_size, image_s
ize, 3)) # Creating the model with the pre-trained weights.
```

In [58]:

```
# We try to unfreeze some of the layers so the model can learn from the images
for layer in vgg_conv.layers[:-3]: # After many different tries we found that allowing th
e last 3 layers to learn gave the best result
    layer.trainable = False

# Showing the VGG16 model stucture and if the layer is trainable or not
for layer in vgg_conv.layers:
    print(layer, layer.trainable)
```

```
<keras.engine.input_layer.InputLayer object at 0x7f98bdb626d8> False
<keras.layers.convolutional.Conv2D object at 0x7f98bdb627b8> False
<keras.layers.convolutional.Conv2D object at 0x7f98bdb62c88> False
<keras.layers.pooling.MaxPooling2D object at 0x7f98bdb11cc0> False
<keras.layers.convolutional.Conv2D object at 0x7f98bd9e8b70> False
<keras.layers.convolutional.Conv2D object at 0x7f98bd9ef3c8> False
<keras.layers.pooling.MaxPooling2D object at 0x7f98bd9f1240> False
<keras.layers.convolutional.Conv2D object at 0x7f98bd9f5860> False
<keras.layers.convolutional.Conv2D object at 0x7f98bd9fa978> False
<keras.layers.convolutional.Conv2D object at 0x7f98bda00898> False
<keras.layers.pooling.MaxPooling2D object at 0x7f98bda0e160> False
<keras.layers.convolutional.Conv2D object at 0x7f98bda10710> False
<keras.layers.convolutional.Conv2D object at 0x7f98bd99a898> False
<keras.layers.convolutional.Conv2D object at 0x7f98bd9a25f8> False
<keras.layers.pooling.MaxPooling2D object at 0x7f98bd9ab5f8> False
<keras.layers.convolutional.Conv2D object at 0x7f98bd9b34a8> False
<keras.layers.convolutional.Conv2D object at 0x7f98bd9bc7b8> True
<keras.layers.convolutional.Conv2D object at 0x7f98bd9c2518> True
<keras.layers.pooling.MaxPooling2D object at 0x7f98bd9ce518> True
```

In [59]:

```
vgg = Sequential() # Now we make the model

vgg.add(vgg_conv) # Add the vgg pre-trained model
vgg.add(Flatten()) # Flattens the input
vgg.add(Dense(1024, activation='relu')) # A normal denselayer with 1024 neurons and with
ReLU as activation
```

```
vgg.add(Dropout(0.1)) # Helps better genralisation of the data and prevents overfitting
vgg.add(Dense(1, activation='sigmoid')) # The output as a single neuron dense and sigmoid as activations (0/1)

vgg.summary() # Show a summary of the model. Check the number of trainable parameters
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 2, 2, 512)         14714688
_________________________________________________________________
flatten_12 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_23 (Dense)             (None, 1024)              2098176
_________________________________________________________________
dropout_3 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_24 (Dense)             (None, 1)                 1025
=================================================================
Total params: 16,813,889
Trainable params: 6,818,817
Non-trainable params: 9,995,072
_________________________________________________________________
```

In [0]:

```
vgg.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc']) # Compiling the model with binary_crosstropy, adam and accuracy as parameters
```

In [61]:

```
vgghistory = vgg.fit_generator(training_set, # Fitting with generator and the training dataset
                    epochs = 5, # Using 5 epochs
                    validation_data = val_set, # Validate the results by the validation dataset
                    validation_steps = val_set.samples / batch_size) # Setting validation steps
```

```
Epoch 1/5
326/326 [==============================] - 69s 212ms/step - loss: 0.2533 - acc: 0.9034 - val_loss: 0.1523 - val_acc: 0.9570
Epoch 2/5
326/326 [==============================] - 67s 205ms/step - loss: 0.1514 - acc: 0.9429 - val_loss: 0.1573 - val_acc: 0.9462
Epoch 3/5
326/326 [==============================] - 67s 206ms/step - loss: 0.1606 - acc: 0.9369 - val_loss: 0.1288 - val_acc: 0.9462
Epoch 4/5
326/326 [==============================] - 68s 209ms/step - loss: 0.1363 - acc: 0.9509 - val_loss: 0.1291 - val_acc: 0.9570
Epoch 5/5
326/326 [==============================] - 68s 208ms/step - loss: 0.1274 - acc: 0.9526 - val_loss: 0.1583 - val_acc: 0.9785
```

In [62]:

```
results = vgg.evaluate_generator(test_set)
print('This is the loss and the accuracy score of the VGG16 model:', results) #Showing the loss and accuracy of the model
```

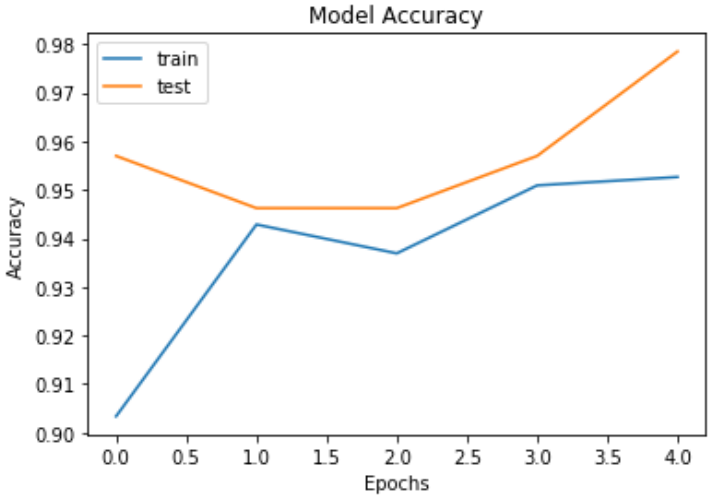
```
This is the loss and the accuracy score of the VGG16 model: [0.49182643721725533, 0.8531073446327684]
```

The evaluate function have calculated the VGG model to 85.31% in accuracy

In [63]:

```
# Creating a plot over each epoch and accuracy
plt.plot(vgghistory.history['acc']) # Plotting accuracy
```

```
plt.plot(vgghistory.history['val_acc']) # Plotting Value accuracy
plt.title('Model Accuracy') # Give the plot a title
plt.ylabel('Accuracy') # labling the y-axis
plt.xlabel('Epochs') # labeling the x-ais
plt.legend(['train', 'test']) # Name each line in the plot
plt.show() # Show the plot
```

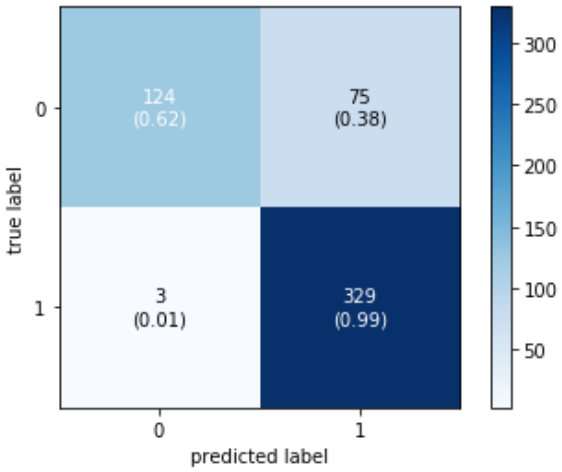


Looking at the above model of how the model progressed through the five epochs, it looks like this model could have benefited from running more epochs, because the test accuracy is rising at the end, but other runs of the exact same code has indicated that the model was overfitting and this number of epochs seems to give the best result in average.

In [64]:

```
y_pred = vgg.predict_generator(test_set) # Use predict on the model with the test_set
y_pred = np.where(y_pred >= 0.5, 1, 0) # Setting everything under 0.5 to 0 and over and
equal 0.5 to 1
confmatrix = confusion_matrix(test_set.labels, y_pred) # Create a confusion matrix
plot_confusion_matrix(conf_mat=confmatrix, colorbar=True, show_absolute=True, show_normed
=True) # Plotting the confusions matrix
print(classification_report(test_set.labels, y_pred)) # Printing the classification repor
t
```

```
              precision    recall  f1-score   support

           0       0.98      0.62      0.76       199
           1       0.81      0.99      0.89       332

    accuracy                           0.85       531
   macro avg       0.90      0.81      0.83       531
weighted avg       0.88      0.85      0.84       531
```



The VGG confusion matrix is a little better than the CNN. Earlier results of the VGG had an accuracy on 90%, were this run only gave 85%. Only 3 x-rays got predicted as False negatives which is great and 75 as false positives, which could be better.

Conclusion

Off the three aproches the VGG model gives the best result.

Tuning of CNN without tranfer learning

In this section we will show an example of how we have tried to tune the model, we have tried changing pixel size, batch size, adding dropout layers, and as can be seen we have created a for loop approach where we try two different optimizers and a different number of neurons in the second to last dense layer, in this case we used 128 pixel and this seemed to give the best results besides of the above model architectures. Besides the things we have tried, it could be possible to try to tune the model based on learning rate, activation functions, number of layers etc. Changing the learning rate and adding more epochs could perhaps also prevent the model from "getting stock" meaning that it stops learning and keeps showing the same accuracy epoch after epoch, which happened a couple of times. Further, we could have used the callbacks such as EarlyStopping to stop the model from overfitting. However, it would only have made sense with a slower learning rate.

For convenience during tuning the split of the data, augmentation, etc. is repeated here, to make easier to make changes to these.

In [45]:

```
Image_size = 128 # Note that we have changed image size to 128 pixels down here.
Batch_size = 16
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255, validation_split=0.15)

training_set = train_datagen.flow_from_directory('chest_xray/train',
                                                 target_size = (Image_size, Image_size)
,
                                                 batch_size = Batch_size,
                                                 class_mode = 'binary')

val_set = test_datagen.flow_from_directory('chest_xray/test',
                                           target_size = (Image_size, Image_size),
                                           batch_size = Batch_size,
                                           class_mode = 'binary',
                                           subset='validation')  # note that we use a
subset of the test dataset as validation

test_set = test_datagen.flow_from_directory('chest_xray/test',
                                           target_size = (Image_size, Image_size),
                                           batch_size = Batch_size,
                                           class_mode = 'binary',
                                           subset='training')  # this is a little unor
thodox naming wise, but to make the split such that most
                                           # of the test data is kept as test data and
not validation this argument is the most fitting.
```

```
Found 5216 images belonging to 2 classes.
Found 93 images belonging to 2 classes.
Found 531 images belonging to 2 classes.
```

We create a function for the creation of our model, such that it becomes easier to perform the tuning of the model. We give it two parameters, which is optimizer and number of neurons, such that it becomes possible to try different optimizers and number of neurons in the second to last denselayer.

In [0]:

```
def create_model(optimizer='adam', nn=128):
  classifier = Sequential()
  classifier.add(Conv2D(32, (3, 3), input_shape = (Image_size, Image_size, 3), activatio
n = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2, 2)))
  classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
  classifier.add(MaxPooling2D(pool_size = (2, 2)))
  classifier.add(Flatten())
  classifier.add(Dense(units = nn, activation = 'relu'))
  classifier.add(Dense(units = 1, activation = 'sigmoid'))
  classifier.compile(optimizer = optimizer, loss = 'binary_crossentropy', metrics = ['ac
curacy'])
  return classifier
```

Normally when performing hyperparametertuning you would use GridSearchCV or RandomizedSearchCV, but because we are using ImageDataGenerator, this is not possible, so we have to create our own for loop, to do something simlar to GridSearchCV, and this can be seen below.

In [47]:

```
results = [] # Creating a list to save the accuracy of each model on the testset
optimizer = ['adam', 'sgd'] # Setting the optimizers we want to try out
numberOfNeurons = [128, 256, 512, 1024] # Setting the different number of neurons
for i in optimizer: # First for loop that will loop through the different optimizers.
  for j in numberOfNeurons: # second (nested) for loop that will loop through each number
of neuron for each optimizer
    model = create_model(i, j) # create the model with the given optimizer and number of
neurons
    model.summary()
    model.fit_generator(training_set,
                        epochs = 5,
                        validation_data = val_set,
                        validation_steps = val_set.samples / Batch_size)
    results.append(model.evaluate_generator(test_set)) # Save the results of the evaulate
on the test of each model.

print(results) # print the results list such that we can compare the results of each mode
l.
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 61, 61, 32)        9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 30, 30, 32)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 28800)             0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               3686528
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 129
=================================================================
Total params: 3,696,801
Trainable params: 3,696,801
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
326/326 [==============================] - 81s 248ms/step - loss: 0.4234 - acc: 0.8046 -
val_loss: 0.5843 - val_acc: 0.6452
Epoch 2/5
326/326 [==============================] - 79s 244ms/step - loss: 0.2835 - acc: 0.8936 -
val_loss: 0.5278 - val_acc: 0.7419
Epoch 3/5
326/326 [==============================] - 80s 245ms/step - loss: 0.2109 - acc: 0.9166 -
val_loss: 1.0031 - val_acc: 0.6667
```

```
Epoch 4/5
326/326 [==============================] - 81s 248ms/step - loss: 0.1993 - acc: 0.9212 -
val_loss: 0.2300 - val_acc: 0.9032
Epoch 5/5
326/326 [==============================] - 80s 245ms/step - loss: 0.1736 - acc: 0.9317 -
val_loss: 0.2637 - val_acc: 0.9032
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 61, 61, 32)        9248
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 30, 30, 32)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 28800)             0
_________________________________________________________________
dense_7 (Dense)              (None, 256)               7373056
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 257
=================================================================
Total params: 7,383,457
Trainable params: 7,383,457
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
326/326 [==============================] - 80s 244ms/step - loss: 0.3337 - acc: 0.8643 -
val_loss: 0.8492 - val_acc: 0.6774
Epoch 2/5
326/326 [==============================] - 79s 243ms/step - loss: 0.2279 - acc: 0.9051 -
val_loss: 0.4464 - val_acc: 0.7742
Epoch 3/5
326/326 [==============================] - 79s 243ms/step - loss: 0.2135 - acc: 0.9185 -
val_loss: 0.4086 - val_acc: 0.7527
Epoch 4/5
326/326 [==============================] - 79s 243ms/step - loss: 0.1884 - acc: 0.9248 -
val_loss: 0.7523 - val_acc: 0.6667
Epoch 5/5
326/326 [==============================] - 79s 242ms/step - loss: 0.1970 - acc: 0.9206 -
val_loss: 0.4115 - val_acc: 0.7419
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 61, 61, 32)        9248
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 30, 30, 32)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 28800)             0
_________________________________________________________________
dense_9 (Dense)              (None, 512)               14746112
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 513
=================================================================
Total params: 14,756,769
Trainable params: 14,756,769
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
326/326 [==============================] - 81s 249ms/step - loss: 0.3446 - acc: 0.8562 -
val_loss: 0.3358 - val_acc: 0.8495
Epoch 2/5
326/326 [==============================] - 81s 248ms/step - loss: 0.2133 - acc: 0.9156 -
val_loss: 0.4032 - val_acc: 0.8065
Epoch 3/5
```

```
326/326 [==============================] - 81s 249ms/step - loss: 0.1978 - acc: 0.9245 -
val_loss: 0.5405 - val_acc: 0.7742
Epoch 4/5
326/326 [==============================] - 81s 249ms/step - loss: 0.1764 - acc: 0.9314 -
val_loss: 0.3964 - val_acc: 0.7742
Epoch 5/5
326/326 [==============================] - 82s 250ms/step - loss: 0.1638 - acc: 0.9329 -
val_loss: 0.1443 - val_acc: 0.9570
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 61, 61, 32)        9248
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 30, 30, 32)        0
_________________________________________________________________
flatten_6 (Flatten)          (None, 28800)             0
_________________________________________________________________
dense_11 (Dense)             (None, 1024)              29492224
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 1025
=================================================================
Total params: 29,503,393
Trainable params: 29,503,393
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
326/326 [==============================] - 85s 262ms/step - loss: 0.3136 - acc: 0.8758 -
val_loss: 0.9437 - val_acc: 0.6559
Epoch 2/5
326/326 [==============================] - 88s 269ms/step - loss: 0.2167 - acc: 0.9107 -
val_loss: 1.2217 - val_acc: 0.6452
Epoch 3/5
326/326 [==============================] - 88s 270ms/step - loss: 0.1901 - acc: 0.9252 -
val_loss: 0.7181 - val_acc: 0.6989
Epoch 4/5
326/326 [==============================] - 87s 268ms/step - loss: 0.1878 - acc: 0.9258 -
val_loss: 0.2470 - val_acc: 0.8710
Epoch 5/5
326/326 [==============================] - 88s 270ms/step - loss: 0.1666 - acc: 0.9339 -
val_loss: 0.4194 - val_acc: 0.8065
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_11 (Conv2D)           (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 61, 61, 32)        9248
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 30, 30, 32)        0
_________________________________________________________________
flatten_7 (Flatten)          (None, 28800)             0
_________________________________________________________________
dense_13 (Dense)             (None, 128)               3686528
_________________________________________________________________
dense_14 (Dense)             (None, 1)                 129
=================================================================
Total params: 3,696,801
Trainable params: 3,696,801
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
326/326 [==============================] - 82s 253ms/step - loss: 0.4767 - acc: 0.7801 -
val_loss: 0.3374 - val_acc: 0.8602
Epoch 2/5
326/326 [==============================] - 83s 256ms/step - loss: 0.3098 - acc: 0.8679 -
```

```
val_loss: 0.3358 - val_acc: 0.8495
Epoch 3/5
326/326 [==============================] - 82s 253ms/step - loss: 0.2690 - acc: 0.8857 -
val_loss: 0.3239 - val_acc: 0.8387
Epoch 4/5
326/326 [==============================] - 82s 253ms/step - loss: 0.2575 - acc: 0.8938 -
val_loss: 0.3727 - val_acc: 0.8172
Epoch 5/5
326/326 [==============================] - 82s 252ms/step - loss: 0.2350 - acc: 0.9062 -
val_loss: 0.1883 - val_acc: 0.9140
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 61, 61, 32)        9248
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 30, 30, 32)        0
_________________________________________________________________
flatten_8 (Flatten)          (None, 28800)             0
_________________________________________________________________
dense_15 (Dense)             (None, 256)               7373056
_________________________________________________________________
dense_16 (Dense)             (None, 1)                 257
=================================================================
Total params: 7,383,457
Trainable params: 7,383,457
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
326/326 [==============================] - 83s 253ms/step - loss: 0.4752 - acc: 0.7895 -
val_loss: 0.3779 - val_acc: 0.8172
Epoch 2/5
326/326 [==============================] - 82s 253ms/step - loss: 0.3182 - acc: 0.8646 -
val_loss: 0.2090 - val_acc: 0.9140
Epoch 3/5
326/326 [==============================] - 82s 253ms/step - loss: 0.2560 - acc: 0.8957 -
val_loss: 0.1733 - val_acc: 0.9247
Epoch 4/5
326/326 [==============================] - 82s 252ms/step - loss: 0.2409 - acc: 0.9020 -
val_loss: 0.1421 - val_acc: 0.9355
Epoch 5/5
326/326 [==============================] - 82s 252ms/step - loss: 0.2309 - acc: 0.9080 -
val_loss: 0.1721 - val_acc: 0.9462
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_15 (Conv2D)           (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 61, 61, 32)        9248
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 30, 30, 32)        0
_________________________________________________________________
flatten_9 (Flatten)          (None, 28800)             0
_________________________________________________________________
dense_17 (Dense)             (None, 512)               14746112
_________________________________________________________________
dense_18 (Dense)             (None, 1)                 513
=================================================================
Total params: 14,756,769
Trainable params: 14,756,769
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
326/326 [==============================] - 80s 246ms/step - loss: 0.4843 - acc: 0.7745 -
val_loss: 0.2899 - val_acc: 0.9247
```

```
Epoch 2/5
326/326 [==============================] - 83s 254ms/step - loss: 0.3132 - acc: 0.8687 -
val_loss: 0.1995 - val_acc: 0.9247
Epoch 3/5
326/326 [==============================] - 83s 253ms/step - loss: 0.2625 - acc: 0.8934 -
val_loss: 0.1769 - val_acc: 0.9355
Epoch 4/5
326/326 [==============================] - 83s 254ms/step - loss: 0.2470 - acc: 0.9032 -
val_loss: 0.1868 - val_acc: 0.9247
Epoch 5/5
326/326 [==============================] - 82s 251ms/step - loss: 0.2299 - acc: 0.9043 -
val_loss: 0.1633 - val_acc: 0.9355
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_17 (Conv2D)           (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 61, 61, 32)        9248
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 30, 30, 32)        0
_________________________________________________________________
flatten_10 (Flatten)         (None, 28800)             0
_________________________________________________________________
dense_19 (Dense)             (None, 1024)              29492224
_________________________________________________________________
dense_20 (Dense)             (None, 1)                 1025
=================================================================
Total params: 29,503,393
Trainable params: 29,503,393
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
326/326 [==============================] - 85s 260ms/step - loss: 0.4809 - acc: 0.7789 -
val_loss: 0.5023 - val_acc: 0.6452
Epoch 2/5
326/326 [==============================] - 83s 256ms/step - loss: 0.3179 - acc: 0.8623 -
val_loss: 0.2898 - val_acc: 0.8817
Epoch 3/5
326/326 [==============================] - 83s 256ms/step - loss: 0.2659 - acc: 0.8921 -
val_loss: 0.2006 - val_acc: 0.9032
Epoch 4/5
326/326 [==============================] - 84s 257ms/step - loss: 0.2479 - acc: 0.8944 -
val_loss: 0.2276 - val_acc: 0.9032
Epoch 5/5
326/326 [==============================] - 84s 258ms/step - loss: 0.2331 - acc: 0.9011 -
val_loss: 0.1478 - val_acc: 0.9247
[[0.32637181300003426, 0.8794726931442649], [0.4399992368989072, 0.8418079097167696], [0.
2833473677913572, 0.8964218456305129], [0.5603517309645026, 0.8267419963457715], [0.44098
477500531186, 0.8173258004888977], [0.3505535425439393, 0.8361581921465204], [0.406650778
48847525, 0.8418079097167696], [0.41321574608933903, 0.8436911488881443]]
```

The result of the “for loop tuning” above is shown as the last thing in the printout. The first 4 results are with the ”adam” optimizer and the last four are runs on the ”sgd” optimizer, and as it can be seen the adam optimizer seems to give the best results in regard to accuracy. Further, the combination of the “adam” optimizer and 512 neurons in the second to last denselayer seems to be the best combination with an accuracy of 89.64 % - thereby making it better than what was achieved earlier with 64 pixels, 128 neurons and the adam optimizers.