Skin Cancer Detection Using Convolution Neural Networks

Project Report

SE4072 - Deep Learning

IT16072848

S. M.R.B Saluwadana

Table of Contents

List of Figures

1. Introduction
In todays world skin cancers become most common cancers in the world. Skin cancers can be harm to human being skin in various ways. Specially because of the sunlight skin cancers can be occur. That is the most common way skin cancers can be harm to human beings skin. Approximately 9,500 people in the U.S are diagnosed with skin cancer every day. Because of that detection of skin cancers are needed for society.

2. Proposed Solution
In this deep learning assignment I have proposed a neural network to detect benign and malignant skin cancers. Specially i detect benign and malignant skin cancer from images. So that I have used deep learning technique call convolution neural networks.

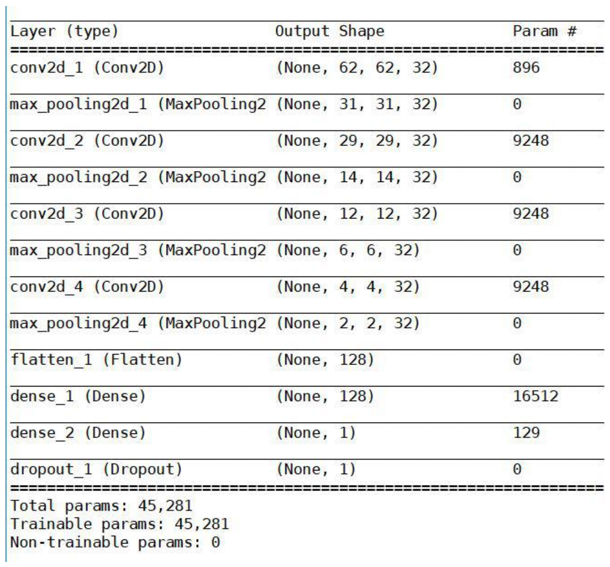
3. Dataset
3.1 Description of Dataset
Before starting to detect whether a skin cancer is benign or malignant we need to find out a dataset which is needed to this context. To train an algorithm I have used a dataset from a kaggle machine learning repository. Following link contains skin cancer dataset.

https://www.kaggle.com/fanconic/skin-cancer-malignant-vs-benign

This above dataset contains 3297 images of skin cancers which are malignant and benign.

3.2 Description of a Features
Basically above dataset has two directories which are train and test. These two directories can be used for training an algorithm as well as testing an algorithm. each of those directories contains another two sub directories. One is for benign and other is for malignant.

4. Algorithm
4.1 Convolution neural network
I used convolution neural network for classify whether given image is a benign or malignant. Convolution neural network is a deep learning algorithm which is most commonly used for image classification problems. It has several layers. Basically we are inputing some images to the convolution neural network and after that it perform some mathematical calculation and finally produce the output with classify class. Basically mathematical calculation performs in a hidden layer of the convolution neural network. So it is just like a black box operation.
In this network I have used several layers. Such as convolution layer , max pooling layer , Dense layer , Dropout layer as well as Fully connected layer. Following figure shows different layers that I applied with convolution neural network (Model summary).



```
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
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 32)        9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 32)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 4, 32)          9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 2, 2, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
_________________________________________________________________
dropout_1 (Dropout)          (None, 1)                 0
=================================================================
Total params: 45,281
Trainable params: 45,281
Non-trainable params: 0
```

Figure 1 : model summary

Their are some activation functions that are used for performing some mathematical calculations in this neural network. I used only two types of activation functions throughout this network.

1. Sigmoid Activation Function
2. Rectified Linear Unit Activation Function

Each of this activation functions will be discussed later in this report.

4.2 Convolution Layer

Convolution is a layer that we need to apply at a very begining of a neural network. We can use convolution layer to extracting special features from input images. Such as edges , curves and so on. Basically convolution is a mathematical process that is automatically calculated in convolution layer. In convolution operation we take a small matrix of numbers call filter. And then we pass it over our input image and transform it based on the values from the filter to obtain feature map or activation map. Following figure describes how convolution operation works in a convolution layer.

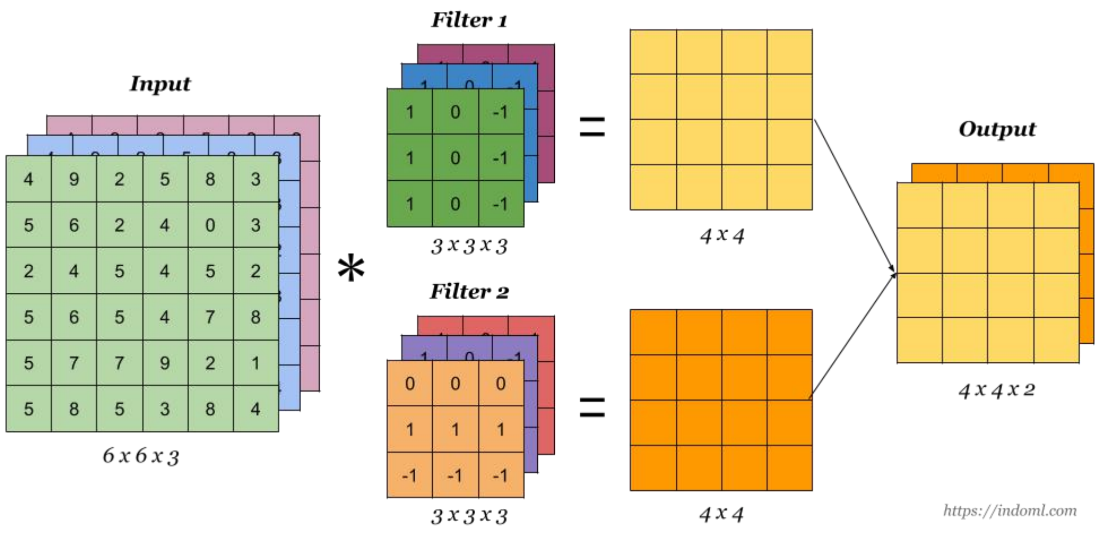


Figure 2 : Convolution Operation

4.3 Pooling

Pooling is the second step which is done after the convolution operation. Basically convolution layer provide a feature map as an output. This output used in a pooling step. What pooling step does is it reduces a size of a feature map. This pooling step is available for controlling overfitting as well. Their are two main types available in pooling layer.

1. Max Pooling
2. Average Pooling

For this algorithm I used max pooling in the pooling step. It returns the maximum value from the portion of the image covered by the kernel. Following figure shows how pooling layer works in different pooling techniques.

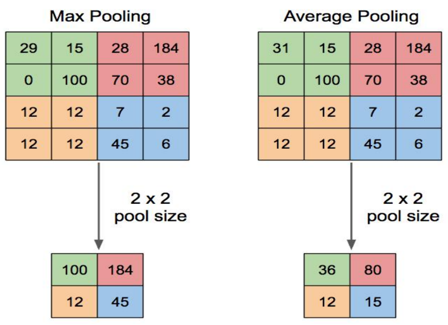


Figure 3 : Pooling

4.4 Flattening

Flattening layer takes all pooled feature maps and maps them into 1D vector. Following figure shows how flattening operation has been done in flattening layer.

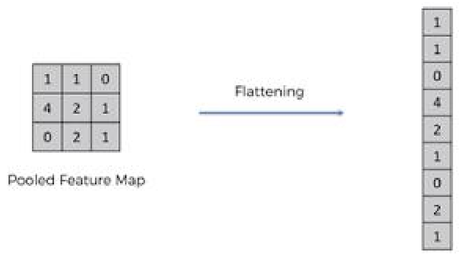


Figure 4 : Flattening

4.5 Fully Connected Layer

In fully connected layer we are combined whole artificial neural network to the convolution neural network. Basically fully connected layers input will be the output provided from the flattening layer. That means a single 1D vector will be a input. This layer determines which feature belongs to which class. That is going to be an output of the fully connected layer. In this scenario their are two possible class that are provided from fully connected layer. Following figure shows how fully connected layer works.

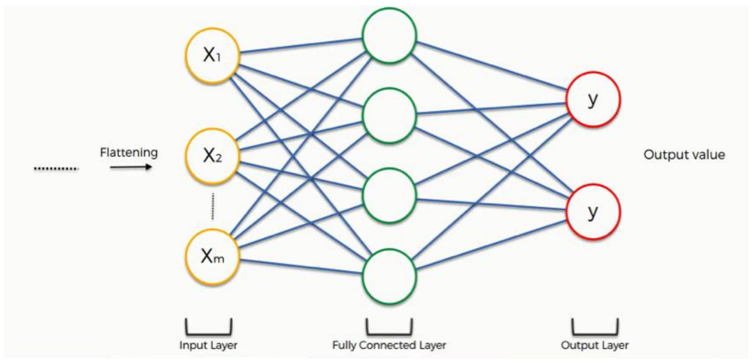


Figure 5 : Fully connected layer

4.6 Dropout Layer

This layer can be used to reduce overfitting of an algorithm. Basically this also acts as a regularizer. When we apply to dropout layer to some algorithm it will reduce some neurons in random manner from a neural network. Following figure explains how dropout layer works.

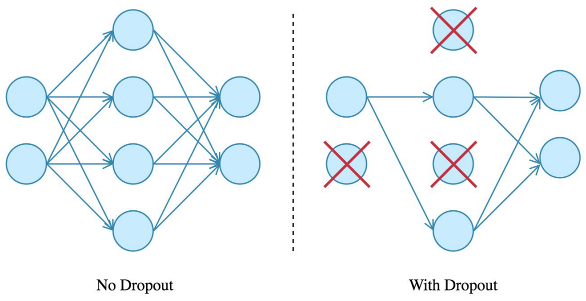
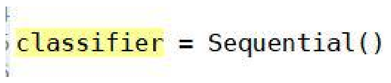


Figure 6 : Dropout

5 Implementation

5.1 Initialize a classifier

Before making a convolution neural network we need to initialize our classifier or else model. So to initialize the classifier I used a class called Sequential which is available in a keras.models package. Following code shows the initialization of model.

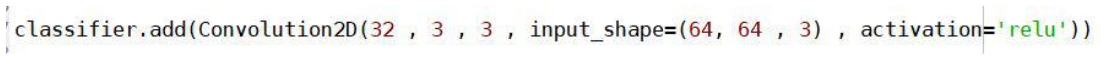


```
classifier = Sequential()
```

Figure 7 : Build a classifier

5.2 Add Convolution Layer

After initialize the model we need perform convolution operation in a convolution layer. In python there is a class call Convolution2D which is available in keras.layers package. I used this class for adding convolution layer. In here we need to define size of feature detectors , input shape and activation function as parameters.
As mentioned in the following code I have used 32 feature detectors with 3×3 dimensions. Next input shape parameter is for converting all input images for same format. Since we are dealing with RGB images we need to convert all input images to 3D array to represent 3 channels. And I used 64×64 as a dimension of each input images. And finally I have used rectified linear unit activation function to perform mathematical operation.

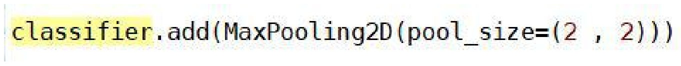


```
classifier.add(Convolution2D(32 , 3 , 3 , input_shape=(64, 64 , 3) , activation='relu'))
```

Figure 8 : Add convolution layer

5.3 Add Max Pooling layer

By performing max pooling operation we can reduce the size of a feature map. In order to perform Max pooling operation I have used MaxPooling2D in keras.layers package. As mention in the following code I have used pool size of 2×2.

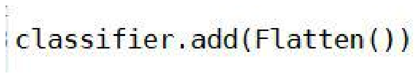


```
classifier.add(MaxPooling2D(pool_size=(2 , 2)))
```

Figure 9 : Add max pooling

5.4 Add flattening

After performing pooling operation I have added flatten layer to perform flattening operation. Here I have used Flatten class which is available in a keras.layers package. By the way for flattening we no need to input any parameters to Flatten function. Since we are using classifier object keras can identify it by own.

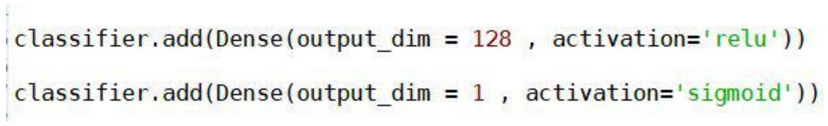


```
classifier.add(Flatten())
```

Figure 10 : Add flattening

5.5 Add fully connected layer

In order to add fully connected layer in python I have used Dense class available in a keras.layers package. As I mention in the following code I have used 128 nodes in a hidden layer. That is mentioned in an output_dim parameter.And second code for output layer. I have mentioned only 1 node in an output layer. That is mentioned in a output_dim parameter in second code. And for output layer I have used sigmoid activation function. I have used that sigmoid activation function because of binary outcome.

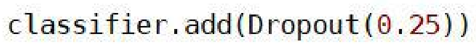


```
classifier.add(Dense(output_dim = 128 , activation='relu'))

classifier.add(Dense(output_dim = 1 , activation='sigmoid'))
```

Figure 11 : Add fully connected layer

5.6 Add Dropout layer

Dropout layer is used for reducing some nodes to reducing an overfitting and improve the accuracy of an algorithm. Here in order to add dropout layer I have used Dropout class which is available in a keras.layers.core package. And here I have removed some nodes with 0.25 probability.



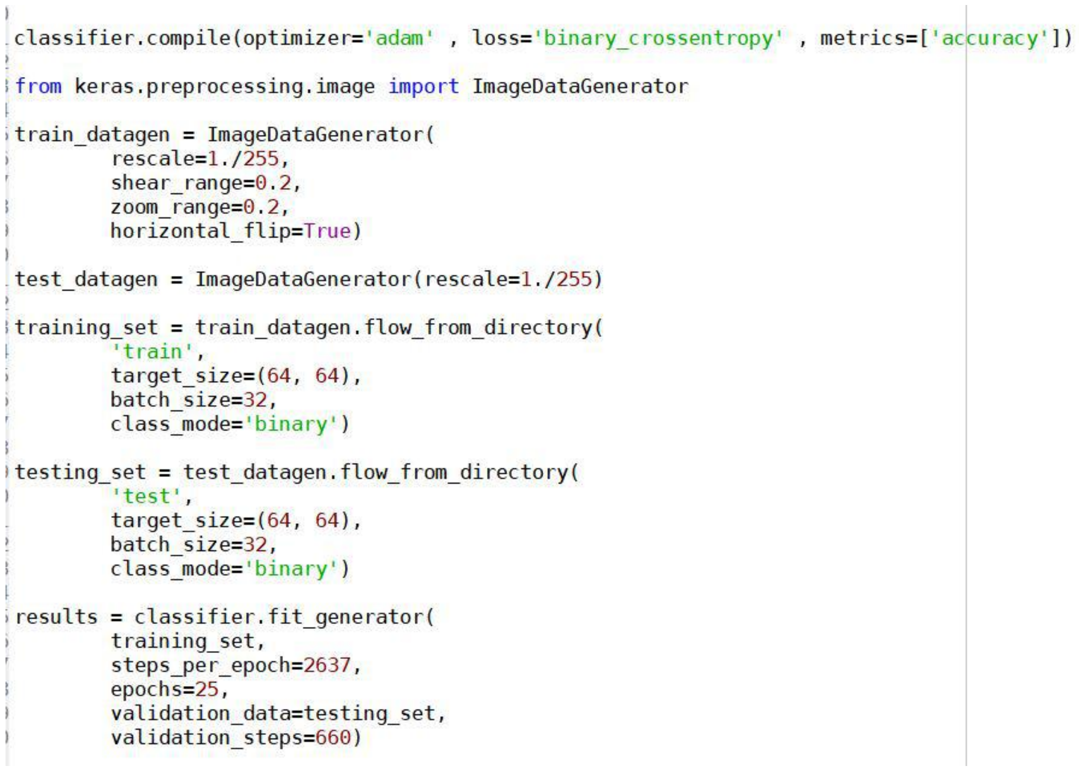
```
classifier.add(Dropout(0.25))
```

Figure 12 : Add dropout layer

5.7 Compile the model

Finally I have started training process of a neural network. In here I have used binary cross entropy as a loss function. because their are only two class in the dataset which are benign and malignant. I used ImageDataGenerator class available in keras.preprocessing.image package to fit the training set images and testing set images to the classifier. Since we are dealing with images that contains in a directory I have used flow_from_directory method.

Our training image dataset contains 2637 images. Because of that I have used 2637 steps for epoch. And their are 25 epochs available. For testing set their are 660 images. Because of that I have used 660 steps in validation set.



```
classifier.compile(optimizer='adam' , loss='binary_crossentropy' , metrics=['accuracy'])

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)

training_set = train_datagen.flow_from_directory(
        'train',
        target_size=(64, 64),
        batch_size=32,
        class_mode='binary')

testing_set = test_datagen.flow_from_directory(
        'test',
        target_size=(64, 64),
        batch_size=32,
        class_mode='binary')

results = classifier.fit_generator(
        training_set,
        steps_per_epoch=2637,
        epochs=25,
        validation_data=testing_set,
        validation_steps=660)
```

Figure 13 : compile the model

6 Test Results

We then used previously created model for doing some prediction. As a result of an above neural network their are several metrics which came as outputs.

6.1 Training and validation accuracy

The first and most important metric is an accuracy of an algorithm. Accuracy tells us probability to do prediction by an algorithm correctly. Basically i tested an accuracy for test set as well as training set separately. If the accuracy is more than 75% then we can trust the algorithm for prediction. Above explain neural network has more than 80% accuracy for training set as well as validation set.

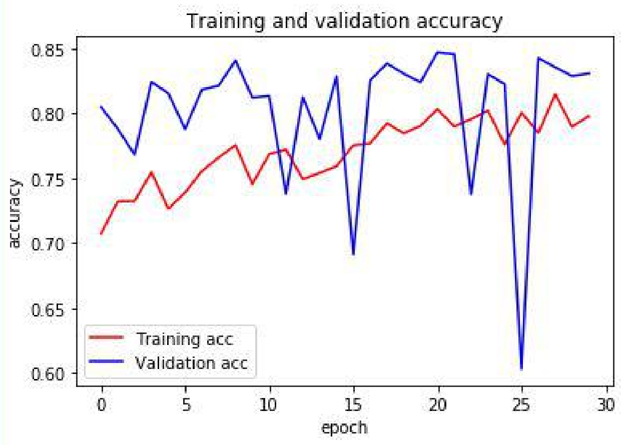


Figure 14 : training and validation accuracy

6.2 Training and validation loss

Loss function is also a main outcome of an algorithm. Specially loss function is used for training a neural network. Loss function also can be also harm to the algorithm accuracy as well. Because in order to get higher accuracy from our algorithm we need to reduce thee loss of an algorithm. If loss value is much higher then accuracy of an algorithm going to be decrease. We have a two different loss values for training set and validation set.

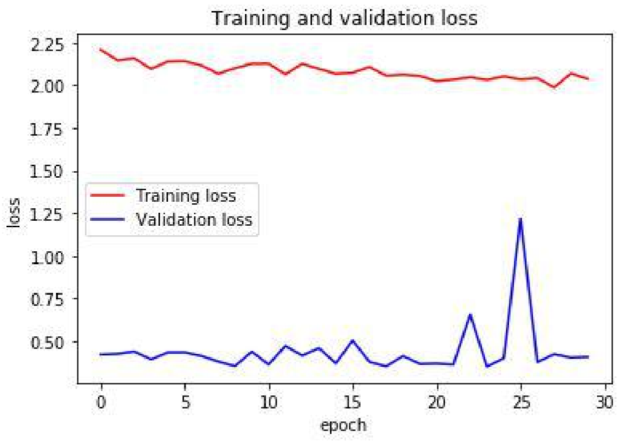


Figure 15 : training and validation loss

7 Evaluation

7.1 Future Works

1 Repeating training process of this convolution neural network in again and again until increase the accuracy of an algorithm

2 Apply same dataset to several neural networks for training and compare the accuracy of all the trained neural networks to find out a most accurate neural network.

3. Train this skin cancer dataset with pre-trained model to increase the accuracy.

8 Discussions

8.1 Logarithmic Loss Function

Basically logarithmic loss function is used for algorithms which produce binary outcomes. Sometimes logarithmic loss function is called as binary cross entropy. So in above explained neural network also produce a binary outcome which means two classes that are malignant and benign. So that I used binary cross entropy as a loss function. Following equation shows how to calculate the loss function.

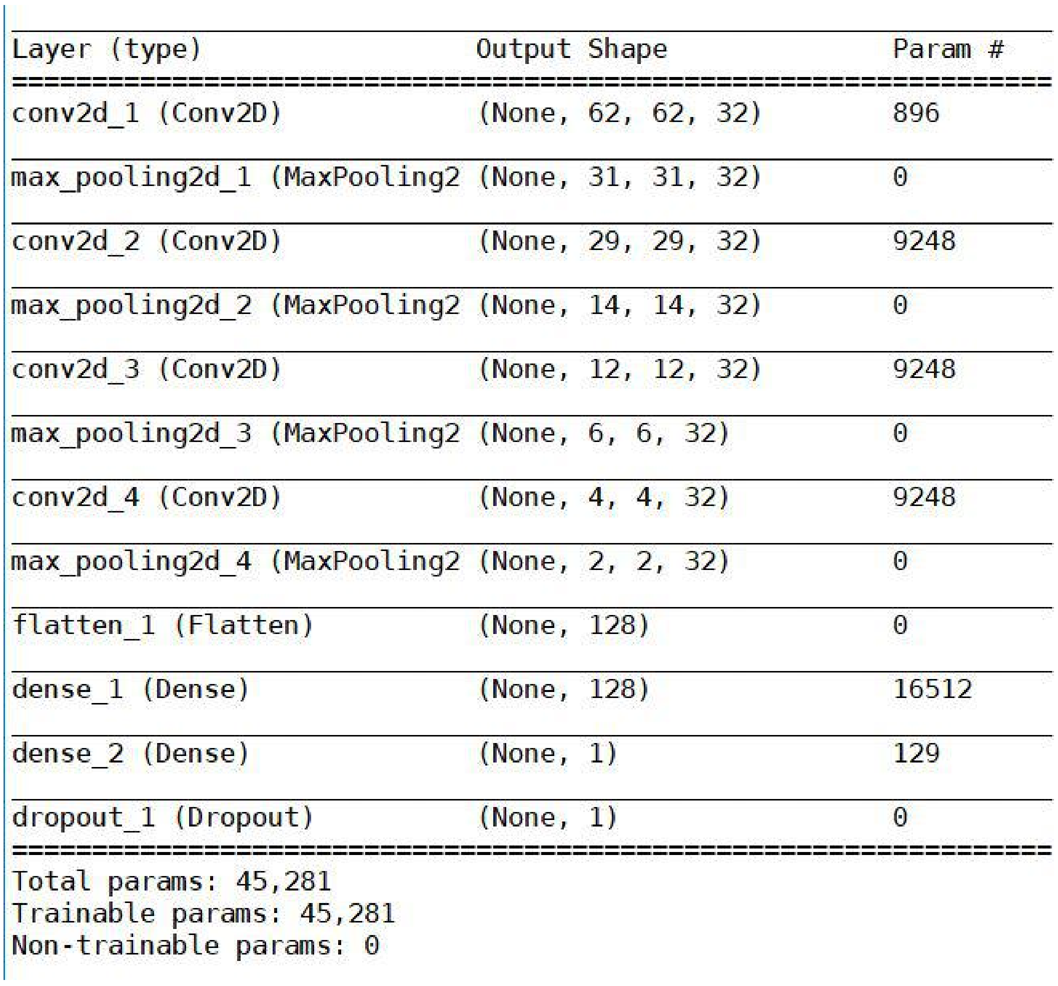
$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Figure 16 : Logarithmic loss function

9 Appendix
9.1 Code
```
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras import regularizers
from keras.layers.core import Dropout

#Build a classifier

classifier = Sequential()

classifier.add(Convolution2D(32 , 3 , 3 , input_shape=(64, 64 , 3) , activation='relu'))

classifier.add(MaxPooling2D(pool_size=(2 , 2)))

classifier.add(Convolution2D(32 , 3 , 3 , activation='relu'))

classifier.add(MaxPooling2D(pool_size=(2 , 2)))

classifier.add(Convolution2D(32 , 3 , 3 , activation='relu'))

classifier.add(MaxPooling2D(pool_size=(2 , 2)))

classifier.add(Convolution2D(32 , 3 , 3 , activation='relu'))

classifier.add(MaxPooling2D(pool_size=(2 , 2)))

classifier.add(Flatten())

classifier.add(Dense(output_dim = 128 , activation='relu'))

classifier.add(Dense(output_dim = 1 , activation='sigmoid'))

classifier.add(Dropout(0.25))

classifier.compile(optimizer='adam',loss='binary_crossentropy' , metrics=['accuracy'])

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)
```

```
training_set = train_datagen.flow_from_directory(
            'train',
            target_size=(64, 64),
            batch_size=32,
            class_mode='binary')

testing_set = test_datagen.flow_from_directory(
            'test',
            target_size=(64, 64),
            batch_size=32,
            class_mode='binary')

results = classifier.fit_generator(
            training_set,
            steps_per_epoch=2637,
            epochs=30,
            validation_data=testing_set,
            validation_steps=660)

classifier.summary()

pip install Pillow

pip install scipy==1.1.0

import scipy.misc
imgs = scipy.misc.imread('test/benign/1.jpg')

imgs = scipy.misc.imresize(imgs,(64 , 64))

classifier.predict_classes(imgs.reshape(1,64,64,3))

true_classes = testing_set.classes
class_labels = list(testing_set.class_indices.keys())

import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report

#Graphing our training and validation
acc = results.history['acc']
val_acc = results.history['val_acc']
loss = results.history['loss']
val_loss = results.history['val_loss']
epochs = range(len(acc))
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend()
plt.show()

#Confution Matrix and Classification Report (To check an accuracy)
from sklearn.metrics import confusion_matrix
batch_size = 32
num_of_test_samples = 660
Y_pred = classifier.predict_generator(testing_set,num_of_test_samples //
batch_size+1)
y_pred = np.argmax(Y_pred, axis=1)
cm = confusion_matrix(testing_set.classes, y_pred)
print('Confusion Matrix')
print(cm)

print('Classification Report')
target_names = ['Benign', 'Maliginent']
print(classification_report(testing_set.classes, y_pred, target_names=target_names))

import seaborn as sns

#print the confusion metrix based on the test values
sns.set()
get_ipython().run_line_magic('matplotlib','inline')
sns.heatmap(cm.T,square=True,annot=True,fmt='d',cbar=False)
plt.xlabel('True value')
plt.ylabel('Predicted value')
```

9.2 Model summary



```
_________________________________________________________________
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
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 32)        9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 32)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 4, 32)          9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 2, 2, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
_________________________________________________________________
dropout_1 (Dropout)          (None, 1)                 0
=================================================================
Total params: 45,281
Trainable params: 45,281
Non-trainable params: 0
```