```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
from glob import glob
import seaborn as sns
from PIL import Image
np.random.seed(123)
from sklearn.preprocessing import label_binarize
from sklearn.metrics import confusion_matrix
import itertools

import keras
from keras.utils.np_utils import to_categorical # used for converting labels to one-hot-encoding
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras import backend as K
import itertools
from keras.layers.normalization import BatchNormalization
from keras.utils.np_utils import to_categorical # convert to one-hot-encoding

from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ReduceLROnPlateau
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
import numpy as np
import sklearn.metrics as metrics


import pandas as pd
import numpy as np
from google.colab import drive
import shutil
import os
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
drive.flush_and_unmount()
```

```
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!mkdir Project
```

```
mkdir: cannot create directory ‘Project’: File exists
```

```
!rm -rf Project/benign
!rm -rf Project/malignant
```

```
!mkdir Project/benign
```

```
# import os, re, os.path
# mypath = "Project/malignant"
# for root, dirs, files in os.walk(mypath):
#     for file in files:
#         os.remove(os.path.join(root, file))
```

```
!mkdir Project/malignant
```

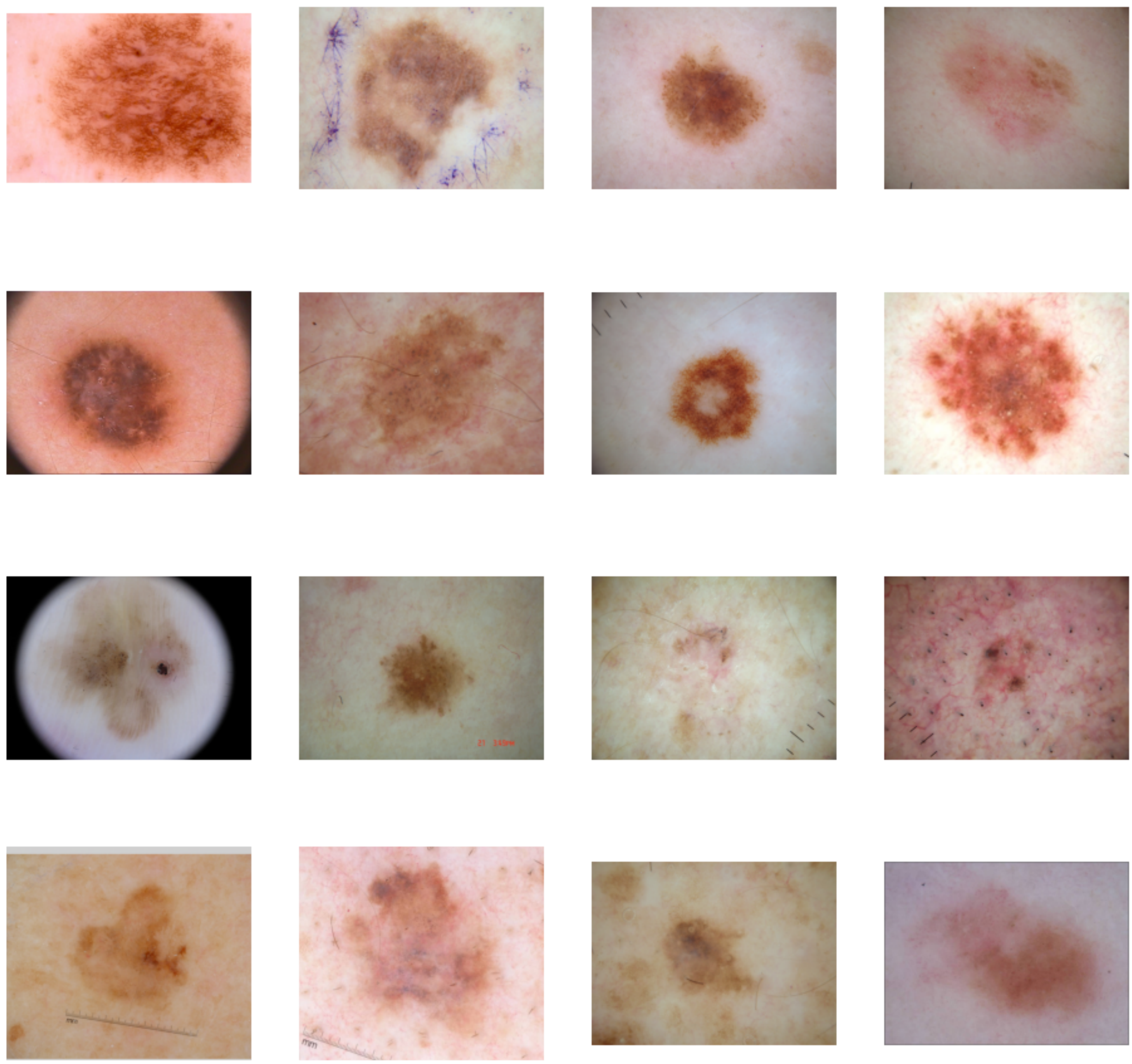
```
#code to split data in benign and malignant

df1=pd.read_csv("/content/drive/My Drive/ISBI2016_ISIC_Part3_Training_GroundTruth.csv",header=None)

source = '/content/drive/My Drive/ISBI2016_ISIC_Part3_Training_Data/'
dest1 = '/content/Project/benign'
dest2 ='/content/Project/malignant'

files = os.listdir(source)
```

```
files = os.listdir(source)

for i in range (df1.shape[0]):
    if df1.loc[i, 1]=="benign":
        shutil.copy(source+df1.loc[i,0]+'.jpg', dest1)
    else:
        shutil.copy(source+df1.loc[i,0]+'.jpg', dest2)



# Code to add more malignant data

df1=pd.read_csv("/content/drive/My Drive/HAM10000_metadata.csv")

source = '/content/drive/My Drive/HAM10000_images_part_1/'
dest2 ='/content/Project/malignant'

files = os.listdir(source)

counter = 727-173
for i in range (df1.shape[0]):
    if df1.loc[i, 'dx']=="mel":
        print (counter, source+df1.loc[i,'image_id']+'.jpg')
        try:
            shutil.copy(source+df1.loc[i,'image_id']+'.jpg', dest2)
            counter -= 1
            if counter == 0:
                break
        except:
            pass

#Changing directories

base_dir = '/content/'

train_dir = os.path.join(base_dir, 'Project')

# Directory with our training picture
benign_dir= os.path.join(train_dir, 'benign')
malignant_dir = os.path.join(train_dir, 'malignant')


print('total training Benign images :', len(os.listdir(      benign_dir ) ))
print('total training Malignant images :', len(os.listdir(      malignant_dir ) ))
```

```
total training Benign images : 727
total training Malignant images : 173
```

```
trainb_fnames = os.listdir( benign_dir )
trainm_fnames = os.listdir( malignant_dir )




%matplotlib inline

# Parameters for our graph; we'll output images in a 4x4 configuration
nrows = 4
ncols = 4

pic_index = 0 # Index for iterating over images


# Set up matplotlib fig, and size it to fit 4x4 pics
fig = plt.gcf()
fig.set_size_inches(ncols*4, nrows*4)

pic_index+=8

next_benign_pix = [os.path.join(benign_dir, fname)
                for fname in trainb_fnames[ pic_index-8:pic_index]
               ]

next_m_pix = [os.path.join(malignant_dir, fname)
                for fname in trainm_fnames[ pic_index-8:pic_index]
               ]
for i, img_path in enumerate(next_benign_pix+next_m_pix):
  # Set up subplot; subplot indices start at 1
  sp = plt.subplot(nrows, ncols, i + 1)
  sp.axis('Off') # Don't show axes (or gridlines)

  img = mpimg.imread(img_path)
```

```
    plt.imshow(img)

plt.show()
```



```
#1. Function to plot model's validation loss and validation accuracy
def plot_model_history(model_history):
    fig, axs = plt.subplots(1,2,figsize=(15,5))
    # summarize history for accuracy
    axs[0].plot(range(1,len(model_history.history['accuracy'])+1),model_history.history['accuracy'])
    axs[0].plot(range(1,len(model_history.history['val_accuracy'])+1),model_history.history['val_accuracy'])
    axs[0].set_title('Model Accuracy')
    axs[0].set_ylabel('Accuracy')
    axs[0].set_xlabel('Epoch')
    axs[0].set_xticks(np.arange(1,len(model_history.history['accuracy'])+1),len(model_history.history['accuracy'])/10)
    axs[0].legend(['train', 'val'], loc='best')
    # summarize history for loss
    axs[1].plot(range(1,len(model_history.history['loss'])+1),model_history.history['loss'])
    axs[1].plot(range(1,len(model_history.history['val_loss'])+1),model_history.history['val_loss'])
    axs[1].set_title('Model Loss')
    axs[1].set_ylabel('Loss')
    axs[1].set_xlabel('Epoch')
    axs[1].set_xticks(np.arange(1,len(model_history.history['loss'])+1),len(model_history.history['loss'])/10)
    axs[1].legend(['train', 'val'], loc='best')
    plt.show()

def show_confusion_matrix(model):
    prob=model.predict_generator(valid_generator)
    y_pred= [list(x).index(True) for x in prob > 0.5]
    cm= metrics.confusion_matrix(valid_generator.classes, y_pred)
    return cm


# Function to plot confusion matrix
```

```
# Function to plot confusion matrix
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
# With data augmentation to prevent overfitting

from tensorflow.keras.preprocessing.image import ImageDataGenerator
base_dir = '/content/Project'
batch_size = 32
# All images will be rescaled by 1./255.
# Flow training images in batches of 20 using generator

train_datagen = ImageDataGenerator(
        rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.2, # Randomly zoom image
        width_shift_range=0.2,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.2,  # randomly shift images vertically (fraction of total height)
        rescale=1./255,
        shear_range=0.2,
        validation_split=0.2
        )

valid_datagen = ImageDataGenerator(
        rescale=1./255,
        validation_split=0.2
        )

train_generator = train_datagen.flow_from_directory(
        train_dir,  # this is the target directory
        target_size=(150, 150),  # all images will be resized to 150x150
        batch_size=batch_size,
        subset = "training",
        class_mode='binary')

valid_generator = valid_datagen.flow_from_directory(
        train_dir,  # this is the target directory
        target_size=(150, 150),  # all images will be resized to 150x150
        batch_size=batch_size,
        subset = "validation",
        class_mode='binary')
```

```
Found 721 images belonging to 2 classes.
Found 179 images belonging to 2 classes.
```

```
#Basic model
model_basic = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)
])
```

```
model_basic.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 72, 72, 32)        9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 36, 36, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 41472)             0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               5308544
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 129
=================================================================
Total params: 5,318,817
Trainable params: 5,318,817
Non-trainable params: 0
_________________________________________________________________
```

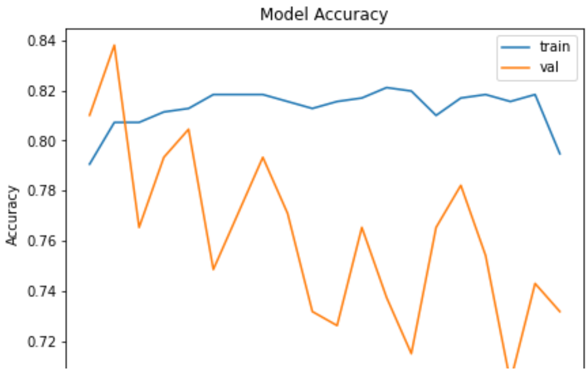
```
model_basic.compile(optimizer = 'adam', loss = tf.keras.metrics.binary_crossentropy, metrics=[ 'accuracy' ])
```

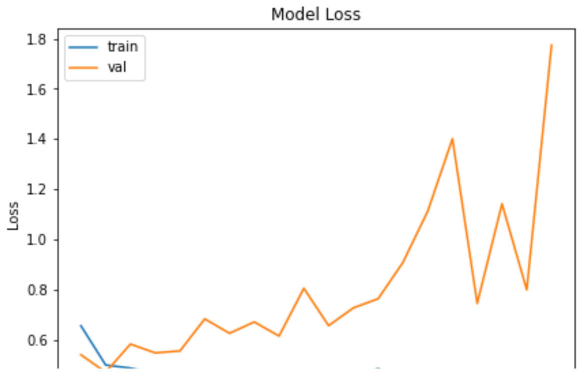
```
# with softmax, last layer = 6
```

```
history_basic = model_basic.fit(train_generator,
                    steps_per_epoch = np.ceil(721/batch_size),
                    epochs=20,
                    validation_data=valid_generator,
                    validation_steps=np.ceil(179/batch_size))
```

```
Epoch 1/20
23/23 [==============================] - 42s 2s/step - loss: 0.6550 - accuracy: 0.7906 - val_loss: 0.5391 - val_accuracy: 0.8101
Epoch 2/20
23/23 [==============================] - 42s 2s/step - loss: 0.4984 - accuracy: 0.8072 - val_loss: 0.4699 - val_accuracy: 0.8380
Epoch 3/20
23/23 [==============================] - 42s 2s/step - loss: 0.4860 - accuracy: 0.8072 - val_loss: 0.5819 - val_accuracy: 0.7654
Epoch 4/20
23/23 [==============================] - 42s 2s/step - loss: 0.4633 - accuracy: 0.8114 - val_loss: 0.5467 - val_accuracy: 0.7933
Epoch 5/20
23/23 [==============================] - 41s 2s/step - loss: 0.4553 - accuracy: 0.8128 - val_loss: 0.5545 - val_accuracy: 0.8045
Epoch 6/20
23/23 [==============================] - 42s 2s/step - loss: 0.4679 - accuracy: 0.8183 - val_loss: 0.6824 - val_accuracy: 0.7486
Epoch 7/20
23/23 [==============================] - 42s 2s/step - loss: 0.4615 - accuracy: 0.8183 - val_loss: 0.6249 - val_accuracy: 0.7709
Epoch 8/20
23/23 [==============================] - 42s 2s/step - loss: 0.4642 - accuracy: 0.8183 - val_loss: 0.6702 - val_accuracy: 0.7933
Epoch 9/20
23/23 [==============================] - 41s 2s/step - loss: 0.4500 - accuracy: 0.8155 - val_loss: 0.6139 - val_accuracy: 0.7709
Epoch 10/20
23/23 [==============================] - 42s 2s/step - loss: 0.4582 - accuracy: 0.8128 - val_loss: 0.8043 - val_accuracy: 0.7318
Epoch 11/20
23/23 [==============================] - 42s 2s/step - loss: 0.4515 - accuracy: 0.8155 - val_loss: 0.6555 - val_accuracy: 0.7263
Epoch 12/20
23/23 [==============================] - 41s 2s/step - loss: 0.4499 - accuracy: 0.8169 - val_loss: 0.7262 - val_accuracy: 0.7654
Epoch 13/20
23/23 [==============================] - 42s 2s/step - loss: 0.4834 - accuracy: 0.8211 - val_loss: 0.7627 - val_accuracy: 0.7374
Epoch 14/20
23/23 [==============================] - 41s 2s/step - loss: 0.4375 - accuracy: 0.8197 - val_loss: 0.9073 - val_accuracy: 0.7151
Epoch 15/20
23/23 [==============================] - 42s 2s/step - loss: 0.4341 - accuracy: 0.8100 - val_loss: 1.1114 - val_accuracy: 0.7654
Epoch 16/20
23/23 [==============================] - 42s 2s/step - loss: 0.4361 - accuracy: 0.8169 - val_loss: 1.4006 - val_accuracy: 0.7821
Epoch 17/20
23/23 [==============================] - 42s 2s/step - loss: 0.4325 - accuracy: 0.8183 - val_loss: 0.7442 - val_accuracy: 0.7542
Epoch 18/20
23/23 [==============================] - 41s 2s/step - loss: 0.4364 - accuracy: 0.8155 - val_loss: 1.1410 - val_accuracy: 0.7039
Epoch 19/20
23/23 [==============================] - 41s 2s/step - loss: 0.4322 - accuracy: 0.8183 - val_loss: 0.7982 - val_accuracy: 0.7430
Epoch 20/20
23/23 [==============================] - 42s 2s/step - loss: 0.4697 - accuracy: 0.7947 - val_loss: 1.7737 - val_accuracy: 0.7318
```

```
plot_model_history(history_basic)
```






```
print(show_confusion_matrix(model_basic,history_basic))
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-52-3df22697c8fe> in <module>()
----> 1 print(show_confusion_matrix(model_basic,history_basic))

11 frames
/usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/execute.py in quick_execute(op_name, num_outputs, inputs, attrs, c
     58     ctx.ensure_initialized()
     59     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
---> 60                                         inputs, attrs, num_outputs)
     61   except core._NotOkStatusException as e:
     62     if name is not None:

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

```
model_basic.save("BasicModel.h5")
```

```
# Set a learning rate annealer
learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

```
#Basic model with dropout layer
model_basic_dropout = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)
])
model_basic_dropout.summary()
```

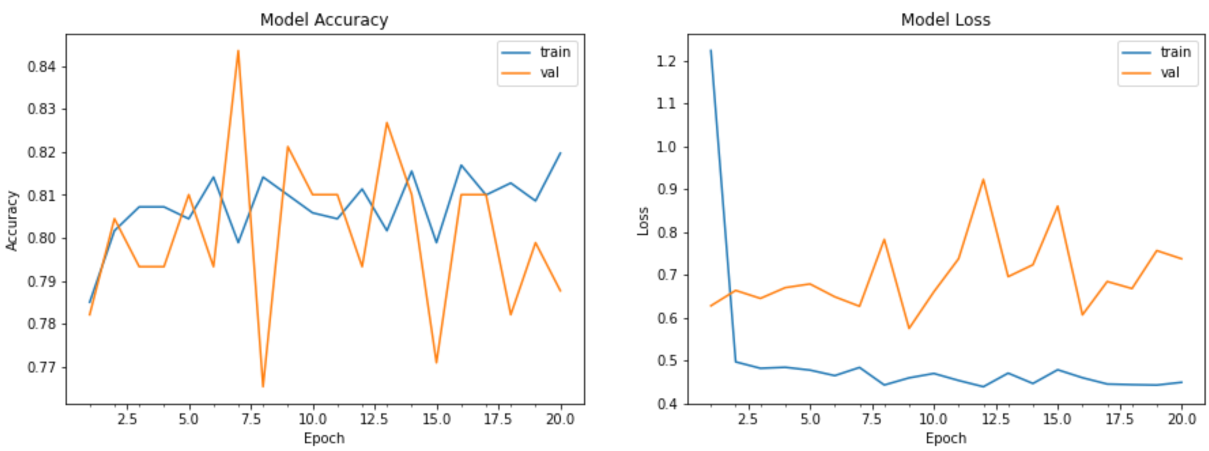
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 74, 74, 32)        0
```

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

```
history_basic_dropout = model_basic_dropout.fit(train_generator,
                    steps_per_epoch = np.ceil(721/batch_size),
                    epochs=20,
                    validation_data=valid_generator,
                    validation_steps=np.ceil(179/batch_size))
```

```
Epoch 1/20
23/23 [==============================] - 40s 2s/step - loss: 1.2227 - accuracy: 0.7850 - val_loss: 0.6274 - val_accuracy: 0.7821
Epoch 2/20
23/23 [==============================] - 41s 2s/step - loss: 0.4966 - accuracy: 0.8017 - val_loss: 0.6633 - val_accuracy: 0.8045
Epoch 3/20
23/23 [==============================] - 41s 2s/step - loss: 0.4816 - accuracy: 0.8072 - val_loss: 0.6446 - val_accuracy: 0.7933
Epoch 4/20
23/23 [==============================] - 41s 2s/step - loss: 0.4839 - accuracy: 0.8072 - val_loss: 0.6697 - val_accuracy: 0.7933
Epoch 5/20
23/23 [==============================] - 40s 2s/step - loss: 0.4774 - accuracy: 0.8044 - val_loss: 0.6783 - val_accuracy: 0.8101
Epoch 6/20
23/23 [==============================] - 40s 2s/step - loss: 0.4645 - accuracy: 0.8141 - val_loss: 0.6484 - val_accuracy: 0.7933
Epoch 7/20
23/23 [==============================] - 42s 2s/step - loss: 0.4835 - accuracy: 0.7989 - val_loss: 0.6263 - val_accuracy: 0.8436
Epoch 8/20
23/23 [==============================] - 41s 2s/step - loss: 0.4426 - accuracy: 0.8141 - val_loss: 0.7822 - val_accuracy: 0.7654
Epoch 9/20
23/23 [==============================] - 42s 2s/step - loss: 0.4593 - accuracy: 0.8100 - val_loss: 0.5745 - val_accuracy: 0.8212
Epoch 10/20
23/23 [==============================] - 40s 2s/step - loss: 0.4695 - accuracy: 0.8058 - val_loss: 0.6599 - val_accuracy: 0.8101
Epoch 11/20
23/23 [==============================] - 41s 2s/step - loss: 0.4532 - accuracy: 0.8044 - val_loss: 0.7372 - val_accuracy: 0.8101
Epoch 12/20
23/23 [==============================] - 40s 2s/step - loss: 0.4386 - accuracy: 0.8114 - val_loss: 0.9222 - val_accuracy: 0.7933
Epoch 13/20
23/23 [==============================] - 42s 2s/step - loss: 0.4703 - accuracy: 0.8017 - val_loss: 0.6953 - val_accuracy: 0.8268
Epoch 14/20
23/23 [==============================] - 40s 2s/step - loss: 0.4461 - accuracy: 0.8155 - val_loss: 0.7231 - val_accuracy: 0.8101
Epoch 15/20
23/23 [==============================] - 41s 2s/step - loss: 0.4782 - accuracy: 0.7989 - val_loss: 0.8599 - val_accuracy: 0.7709
Epoch 16/20
23/23 [==============================] - 40s 2s/step - loss: 0.4596 - accuracy: 0.8169 - val_loss: 0.6062 - val_accuracy: 0.8101
Epoch 17/20
23/23 [==============================] - 40s 2s/step - loss: 0.4449 - accuracy: 0.8100 - val_loss: 0.6840 - val_accuracy: 0.8101
Epoch 18/20
23/23 [==============================] - 40s 2s/step - loss: 0.4434 - accuracy: 0.8128 - val_loss: 0.6676 - val_accuracy: 0.7821
Epoch 19/20
23/23 [==============================] - 41s 2s/step - loss: 0.4424 - accuracy: 0.8086 - val_loss: 0.7561 - val_accuracy: 0.7989
Epoch 20/20
23/23 [==============================] - 41s 2s/step - loss: 0.4489 - accuracy: 0.8197 - val_loss: 0.7371 - val_accuracy: 0.7877
```

```
plot_model_history(history_basic_dropout)
```



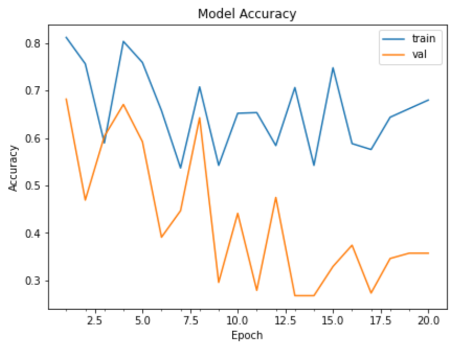

```
model_basic_dropout.save("BasicWithDropout.h5")
```

```
#basic model with oversampling using class weights
model_basic_CW = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150, 150, 3)),
```

```
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)
])

class_weight = {train_generator.class_indices['benign']: 173/900,
                 train_generator.class_indices['malignant']: 727/900}
history_basic_CW = model_basic_dropout.fit(train_generator,
                    steps_per_epoch = np.ceil(721/batch_size),
                    epochs=20,
                    validation_data=valid_generator,
                    validation_steps=np.ceil(179/batch_size),class_weight=class_weight)
```

```
Epoch 1/20
23/23 [==============================] - 42s 2s/step - loss: 0.2114 - accuracy: 0.8114 - val_loss: 0.7184 - val_accuracy: 0.6816
Epoch 2/20
23/23 [==============================] - 40s 2s/step - loss: 0.1934 - accuracy: 0.7559 - val_loss: 0.7711 - val_accuracy: 0.4693
Epoch 3/20
23/23 [==============================] - 42s 2s/step - loss: 0.2015 - accuracy: 0.5895 - val_loss: 0.7861 - val_accuracy: 0.6034
Epoch 4/20
23/23 [==============================] - 40s 2s/step - loss: 0.1931 - accuracy: 0.8031 - val_loss: 0.9443 - val_accuracy: 0.6704
Epoch 5/20
23/23 [==============================] - 41s 2s/step - loss: 0.1940 - accuracy: 0.7587 - val_loss: 0.9651 - val_accuracy: 0.5922
Epoch 6/20
23/23 [==============================] - 42s 2s/step - loss: 0.1885 - accuracy: 0.6574 - val_loss: 0.8197 - val_accuracy: 0.3911
Epoch 7/20
23/23 [==============================] - 39s 2s/step - loss: 0.1972 - accuracy: 0.5368 - val_loss: 0.7619 - val_accuracy: 0.4469
Epoch 8/20
23/23 [==============================] - 42s 2s/step - loss: 0.1871 - accuracy: 0.7074 - val_loss: 0.9731 - val_accuracy: 0.6425
Epoch 9/20
23/23 [==============================] - 41s 2s/step - loss: 0.1967 - accuracy: 0.5423 - val_loss: 0.8000 - val_accuracy: 0.2961
Epoch 10/20
23/23 [==============================] - 41s 2s/step - loss: 0.1854 - accuracy: 0.6519 - val_loss: 0.9636 - val_accuracy: 0.4413
Epoch 11/20
23/23 [==============================] - 41s 2s/step - loss: 0.1990 - accuracy: 0.6533 - val_loss: 0.8912 - val_accuracy: 0.2793
Epoch 12/20
23/23 [==============================] - 41s 2s/step - loss: 0.1934 - accuracy: 0.5839 - val_loss: 0.7811 - val_accuracy: 0.4749
Epoch 13/20
23/23 [==============================] - 40s 2s/step - loss: 0.1951 - accuracy: 0.7060 - val_loss: 0.8209 - val_accuracy: 0.2682
Epoch 14/20
23/23 [==============================] - 42s 2s/step - loss: 0.1963 - accuracy: 0.5423 - val_loss: 0.9572 - val_accuracy: 0.2682
Epoch 15/20
23/23 [==============================] - 40s 2s/step - loss: 0.1878 - accuracy: 0.7476 - val_loss: 1.1236 - val_accuracy: 0.3296
Epoch 16/20
23/23 [==============================] - 39s 2s/step - loss: 0.1885 - accuracy: 0.5881 - val_loss: 1.2524 - val_accuracy: 0.3743
Epoch 17/20
23/23 [==============================] - 43s 2s/step - loss: 0.1911 - accuracy: 0.5756 - val_loss: 0.7974 - val_accuracy: 0.2737
Epoch 18/20
23/23 [==============================] - 40s 2s/step - loss: 0.1945 - accuracy: 0.6436 - val_loss: 0.9665 - val_accuracy: 0.3464
Epoch 19/20
23/23 [==============================] - 43s 2s/step - loss: 0.1838 - accuracy: 0.6616 - val_loss: 1.3017 - val_accuracy: 0.3575
Epoch 20/20
23/23 [==============================] - 41s 2s/step - loss: 0.1957 - accuracy: 0.6796 - val_loss: 1.2979 - val_accuracy: 0.3575
```

```
plot_model_history(history_basic_CW)
model_basic_CW.save("ClassWeights.h5")
```




```
#with learning rate reduction
model_LR = tf.keras.Sequential([
```

```
model_LR = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)
])

class_weight = {train_generator.class_indices['benign']: 173/900,
                 train_generator.class_indices['malignant']: 727/900}
history_LR = model_basic_dropout.fit(train_generator,
                    steps_per_epoch = np.ceil(721/batch_size),
                    epochs=20,
                    validation_data=valid_generator,
                    validation_steps=np.ceil(179/batch_size),class_weight=class_weight,callbacks=[learning_rate_reduction])
plot_model_history(history_LR)
model_LR.save("ReducedLearning.h5")
```

```
Epoch 1/20
23/23 [==============================] - 43s 2s/step - loss: 0.1903 - accuracy: 0.6491 - val_loss: 1.6662 - val_accuracy: 0.3017
Epoch 2/20
23/23 [==============================] - 42s 2s/step - loss: 0.1844 - accuracy: 0.6838 - val_loss: 1.1428 - val_accuracy: 0.3520
Epoch 3/20
23/23 [==============================] - 42s 2s/step - loss: 0.1859 - accuracy: 0.6630 - val_loss: 1.2566 - val_accuracy: 0.3017
Epoch 4/20
```

```
#Model with more layers
input_shape = (150, 150, 3)
num_classes = 2

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',padding = 'Same',input_shape=input_shape))
model.add(Conv2D(32,kernel_size=(3, 3), activation='relu',padding = 'Same',))
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(64, (3, 3), activation='relu',padding = 'Same'))
model.add(Conv2D(64, (3, 3), activation='relu',padding = 'Same'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 150, 150, 32)      896
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 150, 150, 32)      9248
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 75, 75, 32)        0
_________________________________________________________________
dropout_11 (Dropout)         (None, 75, 75, 32)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 75, 75, 64)        18496
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 75, 75, 64)        36928
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 37, 37, 64)        0
_________________________________________________________________
dropout_12 (Dropout)         (None, 37, 37, 64)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 87616)             0
_________________________________________________________________
dense_9 (Dense)              (None, 128)               11214976
_________________________________________________________________
dropout_13 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_10 (Dense)             (None, 2)                 258
=================================================================
Total params: 11,280,802
Trainable params: 11,280,802
Non-trainable params: 0
_________________________________________________________________
```

```
optimizer = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, decay=0.0, amsgrad=False)
# optimizer = 'adam'

# Compile the model
model.compile(optimizer = optimizer , loss = "sparse_categorical_crossentropy", metrics=["accuracy"])
```

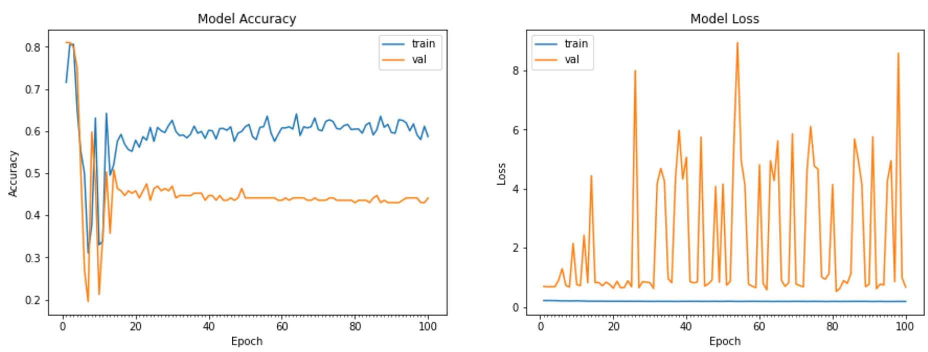
```
epochs = 100
batch_size = 32
#class_weight = {train_generator.class_indices['benign']: 173/900,
                # train_generator.class_indices['malignant']: 727/900}
history = model.fit(train_generator,
                     steps_per_epoch = np.ceil(721/batch_size),
                     epochs=epochs,
                     validation_data=valid_generator,
                     validation_steps=np.ceil(179/batch_size),
                    class_weight=class_weight,
                     callbacks=[learning_rate_reduction]
                     )
```

```
Epoch 1/100
23/23 [==============================] - 37s 2s/step - loss: 0.2165 - accuracy: 0.7157 - val_loss: 0.6911 - val_accuracy: 0.8101
Epoch 2/100
23/23 [==============================] - 36s 2s/step - loss: 0.2149 - accuracy: 0.8058 - val_loss: 0.6801 - val_accuracy: 0.8101
Epoch 3/100
23/23 [==============================] - 36s 2s/step - loss: 0.2142 - accuracy: 0.8058 - val_loss: 0.6837 - val_accuracy: 0.7989
Epoch 4/100
23/23 [==============================] - 36s 2s/step - loss: 0.2134 - accuracy: 0.6505 - val_loss: 0.6818 - val_accuracy: 0.7486
Epoch 5/100
23/23 [==============================] - 36s 2s/step - loss: 0.2087 - accuracy: 0.5520 - val_loss: 0.8979 - val_accuracy: 0.5028

Epoch 00005: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
Epoch 6/100
23/23 [==============================] - 36s 2s/step - loss: 0.2015 - accuracy: 0.4993 - val_loss: 1.2969 - val_accuracy: 0.2682
Epoch 7/100
23/23 [==============================] - 36s 2s/step - loss: 0.2039 - accuracy: 0.3107 - val_loss: 0.7293 - val_accuracy: 0.1955
Epoch 8/100
23/23 [==============================] - 36s 2s/step - loss: 0.2013 - accuracy: 0.3773 - val_loss: 0.6696 - val_accuracy: 0.5978
Epoch 9/100
23/23 [==============================] - 36s 2s/step - loss: 0.2024 - accuracy: 0.6311 - val_loss: 2.1505 - val_accuracy: 0.4804
Epoch 10/100
23/23 [==============================] - 36s 2s/step - loss: 0.2067 - accuracy: 0.3301 - val_loss: 0.7551 - val_accuracy: 0.2123
Epoch 11/100
23/23 [==============================] - 36s 2s/step - loss: 0.2016 - accuracy: 0.3384 - val_loss: 0.7220 - val_accuracy: 0.3520

Epoch 00011: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
Epoch 12/100
23/23 [==============================] - 36s 2s/step - loss: 0.1990 - accuracy: 0.6422 - val_loss: 2.4278 - val_accuracy: 0.5028
Epoch 13/100
23/23 [==============================] - 36s 2s/step - loss: 0.1929 - accuracy: 0.4951 - val_loss: 0.8157 - val_accuracy: 0.3575
Epoch 14/100
23/23 [==============================] - 36s 2s/step - loss: 0.1943 - accuracy: 0.5201 - val_loss: 4.4380 - val_accuracy: 0.5084

Epoch 00014: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
Epoch 15/100
23/23 [==============================] - 36s 2s/step - loss: 0.1932 - accuracy: 0.5756 - val_loss: 0.8250 - val_accuracy: 0.4637
Epoch 16/100
23/23 [==============================] - 36s 2s/step - loss: 0.1941 - accuracy: 0.5922 - val_loss: 0.8243 - val_accuracy: 0.4581
Epoch 17/100
23/23 [==============================] - 36s 2s/step - loss: 0.1948 - accuracy: 0.5687 - val_loss: 0.7159 - val_accuracy: 0.4469

Epoch 00017: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
Epoch 18/100
23/23 [==============================] - 36s 2s/step - loss: 0.1918 - accuracy: 0.5562 - val_loss: 0.8388 - val_accuracy: 0.4581
Epoch 19/100
23/23 [==============================] - 36s 2s/step - loss: 0.1884 - accuracy: 0.5520 - val_loss: 0.7635 - val_accuracy: 0.4525
Epoch 20/100
23/23 [==============================] - 36s 2s/step - loss: 0.1911 - accuracy: 0.5784 - val_loss: 0.6323 - val_accuracy: 0.4581
Epoch 21/100
23/23 [==============================] - 36s 2s/step - loss: 0.1914 - accuracy: 0.5617 - val_loss: 0.8691 - val_accuracy: 0.4413
Epoch 22/100
23/23 [==============================] - 36s 2s/step - loss: 0.1930 - accuracy: 0.5867 - val_loss: 0.6481 - val_accuracy: 0.4581
Epoch 23/100
23/23 [==============================] - 36s 2s/step - loss: 0.1891 - accuracy: 0.5784 - val_loss: 0.6582 - val_accuracy: 0.4749

Epoch 00023: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
Epoch 24/100
23/23 [==============================] - 36s 2s/step - loss: 0.1925 - accuracy: 0.6089 - val_loss: 0.8870 - val_accuracy: 0.4358
Epoch 25/100
23/23 [==============================] - 36s 2s/step - loss: 0.1885 - accuracy: 0.5756 - val_loss: 0.6812 - val_accuracy: 0.4637
Epoch 26/100
23/23 [==============================] - 36s 2s/step - loss: 0.1890 - accuracy: 0.6089 - val_loss: 7.9930 - val_accuracy: 0.4693

Epoch 00026: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
Epoch 27/100
23/23 [==============================] - 36s 2s/step - loss: 0.1908 - accuracy: 0.6006 - val_loss: 0.6501 - val_accuracy: 0.4581
Epoch 28/100
23/23 [==============================] - 36s 2s/step - loss: 0.1896 - accuracy: 0.5964 - val_loss: 0.8520 - val_accuracy: 0.4637
Epoch 29/100
23/23 [==============================] - 36s 2s/step - loss: 0.1864 - accuracy: 0.6130 - val_loss: 0.8424 - val_accuracy: 0.4581

Epoch 00029: ReduceLROnPlateau reducing learning rate to 1e-05.
Epoch 30/100
23/23 [==============================] - 36s 2s/step - loss: 0.1878 - accuracy: 0.6255 - val_loss: 0.8221 - val_accuracy: 0.4693
Epoch 31/100
23/23 [==============================] - 36s 2s/step - loss: 0.1883 - accuracy: 0.5992 - val_loss: 0.6228 - val_accuracy: 0.4413
Epoch 32/100
23/23 [==============================] - 36s 2s/step - loss: 0.1904 - accuracy: 0.5895 - val_loss: 4.1849 - val_accuracy: 0.4469
Epoch 33/100
23/23 [==============================] - 36s 2s/step - loss: 0.1877 - accuracy: 0.5908 - val_loss: 4.6850 - val_accuracy: 0.4469
Epoch 34/100
23/23 [==============================] - 36s 2s/step - loss: 0.1890 - accuracy: 0.5839 - val_loss: 4.2436 - val_accuracy: 0.4469
Epoch 35/100
23/23 [==============================] - 36s 2s/step - loss: 0.1896 - accuracy: 0.5922 - val_loss: 0.9555 - val_accuracy: 0.4469
Epoch 36/100
23/23 [==============================] - 36s 2s/step - loss: 0.1867 - accuracy: 0.6117 - val_loss: 0.8192 - val_accuracy: 0.4525
Epoch 37/100
23/23 [==============================] - 36s 2s/step - loss: 0.1860 - accuracy: 0.5950 - val_loss: 4.3264 - val_accuracy: 0.4525
Epoch 38/100
23/23 [==============================] - 36s 2s/step - loss: 0.1865 - accuracy: 0.5992 - val_loss: 5.9755 - val_accuracy: 0.4525
```

```
Epoch 39/100
23/23 [==============================] - 36s 2s/step - loss: 0.1884 - accuracy: 0.5825 - val_loss: 4.3222 - val_accuracy: 0.4358
Epoch 40/100
23/23 [==============================] - 36s 2s/step - loss: 0.1867 - accuracy: 0.6019 - val_loss: 5.0632 - val_accuracy: 0.4469
Epoch 41/100
23/23 [==============================] - 36s 2s/step - loss: 0.1878 - accuracy: 0.6006 - val_loss: 0.8525 - val_accuracy: 0.4469
Epoch 42/100
23/23 [==============================] - 36s 2s/step - loss: 0.1877 - accuracy: 0.5811 - val_loss: 0.8144 - val_accuracy: 0.4358
Epoch 43/100
23/23 [==============================] - 36s 2s/step - loss: 0.1896 - accuracy: 0.6061 - val_loss: 0.8446 - val_accuracy: 0.4469
Epoch 44/100
23/23 [==============================] - 36s 2s/step - loss: 0.1880 - accuracy: 0.6061 - val_loss: 5.7499 - val_accuracy: 0.4358
Epoch 45/100
23/23 [==============================] - 36s 2s/step - loss: 0.1872 - accuracy: 0.6019 - val_loss: 0.7036 - val_accuracy: 0.4358
Epoch 46/100
23/23 [==============================] - 36s 2s/step - loss: 0.1878 - accuracy: 0.6103 - val_loss: 0.7800 - val_accuracy: 0.4413
Epoch 47/100
23/23 [==============================] - 36s 2s/step - loss: 0.1883 - accuracy: 0.5756 - val_loss: 0.9138 - val_accuracy: 0.4358
Epoch 48/100
23/23 [==============================] - 36s 2s/step - loss: 0.1896 - accuracy: 0.5950 - val_loss: 4.0798 - val_accuracy: 0.4413
Epoch 49/100
23/23 [==============================] - 36s 2s/step - loss: 0.1863 - accuracy: 0.5992 - val_loss: 0.8347 - val_accuracy: 0.4637
Epoch 50/100
23/23 [==============================] - 36s 2s/step - loss: 0.1868 - accuracy: 0.6103 - val_loss: 4.1565 - val_accuracy: 0.4413
Epoch 51/100
23/23 [==============================] - 36s 2s/step - loss: 0.1905 - accuracy: 0.6158 - val_loss: 0.7366 - val_accuracy: 0.4413
Epoch 52/100
23/23 [==============================] - 36s 2s/step - loss: 0.1923 - accuracy: 0.5867 - val_loss: 0.8634 - val_accuracy: 0.4413
Epoch 53/100
23/23 [==============================] - 36s 2s/step - loss: 0.1858 - accuracy: 0.5798 - val_loss: 5.6125 - val_accuracy: 0.4413
Epoch 54/100
23/23 [==============================] - 36s 2s/step - loss: 0.1855 - accuracy: 0.6089 - val_loss: 8.9434 - val_accuracy: 0.4413
Epoch 55/100
23/23 [==============================] - 36s 2s/step - loss: 0.1864 - accuracy: 0.6103 - val_loss: 4.9474 - val_accuracy: 0.4413
Epoch 56/100
23/23 [==============================] - 36s 2s/step - loss: 0.1862 - accuracy: 0.6352 - val_loss: 4.1338 - val_accuracy: 0.4413
Epoch 57/100
23/23 [==============================] - 36s 2s/step - loss: 0.1880 - accuracy: 0.5964 - val_loss: 0.7691 - val_accuracy: 0.4413
Epoch 58/100
23/23 [==============================] - 36s 2s/step - loss: 0.1896 - accuracy: 0.5756 - val_loss: 0.7025 - val_accuracy: 0.4413
Epoch 59/100
23/23 [==============================] - 36s 2s/step - loss: 0.1875 - accuracy: 0.5922 - val_loss: 0.6531 - val_accuracy: 0.4358
Epoch 60/100
23/23 [==============================] - 36s 2s/step - loss: 0.1853 - accuracy: 0.6075 - val_loss: 4.8100 - val_accuracy: 0.4358
Epoch 61/100
23/23 [==============================] - 36s 2s/step - loss: 0.1876 - accuracy: 0.6075 - val_loss: 0.7941 - val_accuracy: 0.4413
Epoch 62/100
23/23 [==============================] - 36s 2s/step - loss: 0.1896 - accuracy: 0.6103 - val_loss: 0.5786 - val_accuracy: 0.4358
Epoch 63/100
23/23 [==============================] - 36s 2s/step - loss: 0.1838 - accuracy: 0.6047 - val_loss: 4.9465 - val_accuracy: 0.4413
Epoch 64/100
23/23 [==============================] - 36s 2s/step - loss: 0.1835 - accuracy: 0.6408 - val_loss: 4.2718 - val_accuracy: 0.4413
Epoch 65/100
23/23 [==============================] - 36s 2s/step - loss: 0.1877 - accuracy: 0.5895 - val_loss: 5.6181 - val_accuracy: 0.4413
Epoch 66/100
23/23 [==============================] - 36s 2s/step - loss: 0.1836 - accuracy: 0.6103 - val_loss: 0.8978 - val_accuracy: 0.4413
Epoch 67/100
23/23 [==============================] - 36s 2s/step - loss: 0.1870 - accuracy: 0.6075 - val_loss: 0.6944 - val_accuracy: 0.4358
Epoch 68/100
23/23 [==============================] - 36s 2s/step - loss: 0.1838 - accuracy: 0.6103 - val_loss: 0.8243 - val_accuracy: 0.4358
Epoch 69/100
23/23 [==============================] - 36s 2s/step - loss: 0.1876 - accuracy: 0.6311 - val_loss: 5.8564 - val_accuracy: 0.4413
Epoch 70/100
23/23 [==============================] - 36s 2s/step - loss: 0.1874 - accuracy: 0.6033 - val_loss: 0.7720 - val_accuracy: 0.4358
Epoch 71/100
23/23 [==============================] - 35s 2s/step - loss: 0.1859 - accuracy: 0.6006 - val_loss: 0.7201 - val_accuracy: 0.4358
Epoch 72/100
23/23 [==============================] - 36s 2s/step - loss: 0.1868 - accuracy: 0.6227 - val_loss: 0.6801 - val_accuracy: 0.4358
Epoch 73/100
23/23 [==============================] - 36s 2s/step - loss: 0.1848 - accuracy: 0.6269 - val_loss: 4.6084 - val_accuracy: 0.4413
Epoch 74/100
23/23 [==============================] - 36s 2s/step - loss: 0.1850 - accuracy: 0.6227 - val_loss: 6.1050 - val_accuracy: 0.4413
Epoch 75/100
23/23 [==============================] - 36s 2s/step - loss: 0.1873 - accuracy: 0.6061 - val_loss: 4.7631 - val_accuracy: 0.4358
Epoch 76/100
23/23 [==============================] - 36s 2s/step - loss: 0.1863 - accuracy: 0.6047 - val_loss: 4.6633 - val_accuracy: 0.4358
Epoch 77/100
23/23 [==============================] - 36s 2s/step - loss: 0.1879 - accuracy: 0.6130 - val_loss: 1.0077 - val_accuracy: 0.4358
Epoch 78/100
23/23 [==============================] - 36s 2s/step - loss: 0.1825 - accuracy: 0.6158 - val_loss: 0.9381 - val_accuracy: 0.4358
Epoch 79/100
23/23 [==============================] - 36s 2s/step - loss: 0.1840 - accuracy: 0.6033 - val_loss: 1.1232 - val_accuracy: 0.4358
Epoch 80/100
23/23 [==============================] - 36s 2s/step - loss: 0.1872 - accuracy: 0.6047 - val_loss: 4.1459 - val_accuracy: 0.4302
Epoch 81/100
23/23 [==============================] - 36s 2s/step - loss: 0.1866 - accuracy: 0.6047 - val_loss: 0.5222 - val_accuracy: 0.4358
Epoch 82/100
23/23 [==============================] - 36s 2s/step - loss: 0.1837 - accuracy: 0.5950 - val_loss: 0.6284 - val_accuracy: 0.4358
Epoch 83/100
23/23 [==============================] - 36s 2s/step - loss: 0.1857 - accuracy: 0.6144 - val_loss: 0.8929 - val_accuracy: 0.4358
Epoch 84/100
```

```
Epoch 84/100
23/23 [==============================] - 36s 2s/step - loss: 0.1856 - accuracy: 0.6200 - val_loss: 0.7941 - val_accuracy: 0.4302
Epoch 85/100
23/23 [==============================] - 36s 2s/step - loss: 0.1881 - accuracy: 0.5908 - val_loss: 1.1188 - val_accuracy: 0.4413
Epoch 86/100
23/23 [==============================] - 36s 2s/step - loss: 0.1871 - accuracy: 0.6033 - val_loss: 5.6807 - val_accuracy: 0.4469
Epoch 87/100
23/23 [==============================] - 36s 2s/step - loss: 0.1868 - accuracy: 0.6352 - val_loss: 4.9934 - val_accuracy: 0.4302
Epoch 88/100
23/23 [==============================] - 36s 2s/step - loss: 0.1890 - accuracy: 0.6089 - val_loss: 4.1509 - val_accuracy: 0.4358
Epoch 89/100
23/23 [==============================] - 36s 2s/step - loss: 0.1878 - accuracy: 0.6158 - val_loss: 0.6835 - val_accuracy: 0.4302
Epoch 90/100
23/23 [==============================] - 36s 2s/step - loss: 0.1833 - accuracy: 0.5964 - val_loss: 0.7694 - val_accuracy: 0.4302
Epoch 91/100
23/23 [==============================] - 36s 2s/step - loss: 0.1836 - accuracy: 0.5950 - val_loss: 5.7565 - val_accuracy: 0.4302
Epoch 92/100
23/23 [==============================] - 36s 2s/step - loss: 0.1862 - accuracy: 0.6269 - val_loss: 0.6150 - val_accuracy: 0.4302
Epoch 93/100
23/23 [==============================] - 36s 2s/step - loss: 0.1869 - accuracy: 0.6255 - val_loss: 0.7711 - val_accuracy: 0.4358
Epoch 94/100
23/23 [==============================] - 36s 2s/step - loss: 0.1831 - accuracy: 0.6200 - val_loss: 0.7427 - val_accuracy: 0.4413
Epoch 95/100
23/23 [==============================] - 36s 2s/step - loss: 0.1837 - accuracy: 0.6006 - val_loss: 4.2657 - val_accuracy: 0.4413
Epoch 96/100
23/23 [==============================] - 36s 2s/step - loss: 0.1848 - accuracy: 0.6172 - val_loss: 4.9482 - val_accuracy: 0.4413
Epoch 97/100
23/23 [==============================] - 36s 2s/step - loss: 0.1851 - accuracy: 0.5908 - val_loss: 0.8554 - val_accuracy: 0.4413
Epoch 98/100
23/23 [==============================] - 36s 2s/step - loss: 0.1846 - accuracy: 0.5798 - val_loss: 8.5864 - val_accuracy: 0.4302
Epoch 99/100
23/23 [==============================] - 36s 2s/step - loss: 0.1861 - accuracy: 0.6117 - val_loss: 0.9841 - val_accuracy: 0.4302
Epoch 100/100
23/23 [==============================] - 36s 2s/step - loss: 0.1848 - accuracy: 0.5867 - val_loss: 0.6701 - val_accuracy: 0.4413
```

```
plot_model_history(history)
```




```
model.save("MoreLayers_100.h5")
```

```
import matplotlib.pyplot as plt
import numpy as np
import sklearn.metrics as metrics
prob = model.predict_generator(valid_generator)
print(prob)
```

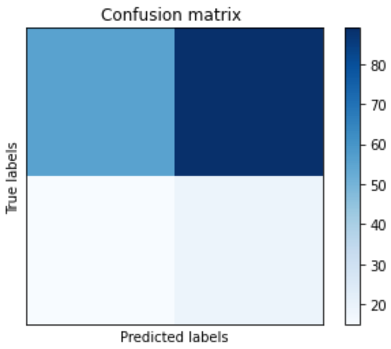
```
[[4.21011716e-01 5.78988254e-01]
 [9.99998093e-01 1.85082217e-06]
 [5.83931327e-01 4.16068673e-01]
 [3.26537490e-01 6.73462510e-01]
 [8.98771942e-01 1.01228058e-01]
 [4.28992748e-01 5.71007252e-01]
 [4.17422384e-01 5.82577646e-01]
 [6.50521219e-01 3.49478751e-01]
 [4.84479308e-01 5.15520692e-01]
 [4.06455487e-01 5.93544483e-01]
 [4.41870391e-01 5.58129609e-01]
 [3.50477785e-01 6.49522245e-01]
 [3.53937119e-01 6.46062851e-01]
 [6.25625074e-01 3.74374926e-01]
 [3.19060981e-01 6.80939078e-01]
 [3.19653362e-01 6.80346608e-01]
 [3.73040140e-01 6.26959860e-01]
 [6.17223561e-01 3.82776409e-01]
 [6.74780726e-01 3.25219303e-01]
 [3.70542616e-01 6.29457414e-01]
 [3.94078165e-01 6.05921805e-01]
 [4.43790019e-01 5.56209981e-01]
 [4.58247900e-01 5.41752040e-01]
 [5.87268651e-01 4.12731349e-01]
 [3.95948946e-01 6.04051054e-01]
 [5.29673517e-01 4.70326513e-01]
 [6.36120856e-01 3.63879114e-01]
 [1.00000000e+00 1.47738363e-10]
 [4.16361928e-01 5.83638072e-01]
 [4.15555418e-01 5.84444582e-01]
 [2.50177175e-01 7.49822795e-01]
 [5.37397027e-01 4.62602973e-01]
 [5.95813394e-01 4.04186636e-01]
 [4.75153863e-01 5.24846137e-01]
 [3.45361978e-01 6.54638052e-01]
 [2.50568628e-01 7.49431372e-01]
 [4.43418294e-01 5.56581676e-01]
 [4.35145557e-01 5.64854443e-01]
 [1.71880826e-01 8.28119159e-01]
 [4.14409727e-01 5.85590243e-01]
 [4.60161090e-01 5.39838910e-01]
 [2.71254569e-01 7.28745461e-01]
 [4.39626664e-01 5.60373366e-01]
 [4.19809192e-01 5.80190837e-01]
 [5.30658305e-01 4.69341666e-01]
 [3.07999551e-01 6.92000449e-01]
 [6.62646234e-01 3.37353826e-01]
 [4.40540493e-01 5.59459448e-01]
 [6.11814320e-01 3.88185680e-01]
 [3.66358995e-01 6.33641005e-01]
 [1.23044245e-01 8.76955748e-01]
 [4.20395523e-01 5.79604447e-01]
 [1.00000000e+00 9.97521464e-20]
 [5.84099412e-01 4.15900618e-01]
 [9.98567224e-01 1.43272348e-03]
 [2.67342657e-01 7.32657373e-01]
 [4.21427876e-01 5.78572154e-01]
 [1.00000000e+00 2.74232151e-14]
 [5.62455952e-01 4.37544048e-01]
 [6.78786397e-01 3.21213603e-01]
 [9.84840319e-02 9.01515961e-01]
 [3.83293301e-01 6.16706669e-01]
 [7.15953469e-01 2.84046531e-01]
 [2.99539864e-01 7.00460196e-01]
 [3.85381371e-01 6.14618659e-01]
 [6.62732482e-01 3.37267548e-01]
 [2.57218897e-01 7.42781103e-01]
 [4.31926847e-01 5.68073153e-01]
 [1.00000000e+00 1.44274518e-28]
 [3.10507566e-01 6.89492464e-01]
 [6.60040438e-01 3.39959592e-01]
 [3.79001856e-01 6.20998144e-01]
 [6.65835619e-01 3.34164381e-01]
 [5.18302858e-01 4.81697172e-01]
 [6.63386047e-01 3.36613923e-01]
 [4.33181524e-01 5.66818476e-01]
 [4.32756990e-01 5.67243040e-01]
 [1.46339357e-01 8.53660643e-01]
 [5.49579740e-01 4.50420231e-01]
 [2.03051537e-01 7.96948493e-01]
 [1.00000000e+00 0.00000000e+00]
 [3.01030755e-01 6.98969245e-01]
 [1.00000000e+00 3.94594315e-23]
 [4.40927625e-01 5.59072375e-01]
 [1.89882711e-01 8.10117245e-01]
 [9.32766199e-02 9.06723440e-01]
 [1.98529735e-01 8.01470280e-01]
 [2.75476307e-01 7.24523723e-01]
 [2.05529928e-01 7.94470072e-01]
 [3.05535018e-01 6.94464982e-01]
```

```
 [5.16314864e-01 4.83685136e-01]
 [1.00000000e+00 3.47983315e-34]
 [1.12480678e-01 8.87519300e-01]
 [5.90553403e-01 4.09446627e-01]
 [3.46420966e-02 9.65357900e-01]
 [3.41949284e-01 6.58050716e-01]
 [4.55453098e-01 5.44546902e-01]
 [4.43387091e-01 5.56612849e-01]
 [5.72853744e-01 4.27146256e-01]
 [8.57076526e-01 1.42923504e-01]
 [4.38573599e-01 5.61426401e-01]
 [7.84299374e-01 2.15700626e-01]
 [1.87972844e-01 8.12027216e-01]
 [4.35521752e-01 5.64478219e-01]
 [6.81531847e-01 3.18468153e-01]
 [6.07176125e-01 3.92823845e-01]
 [2.24462643e-01 7.75537372e-01]
 [7.38275766e-01 2.61724234e-01]
 [4.35733616e-01 5.64266384e-01]
 [6.81027234e-01 3.18972766e-01]
 [5.74870050e-01 4.25129950e-01]
 [4.33956712e-01 5.66043317e-01]
 [3.65959734e-01 6.34040296e-01]
 [3.46439630e-01 6.53560340e-01]
 [4.35244769e-01 5.64755201e-01]
 [4.40909922e-01 5.59090078e-01]
 [5.49002528e-01 4.50997442e-01]
 [1.71760857e-01 8.28239143e-01]
 [2.72833467e-01 7.27166474e-01]
 [4.52515960e-01 5.47484040e-01]
 [6.41858399e-01 3.58141631e-01]
 [1.00000000e+00 2.57926680e-09]
 [3.91339928e-01 6.08660102e-01]
 [6.21900380e-01 3.78099561e-01]
 [4.34051901e-01 5.65948129e-01]
 [2.56535441e-01 7.43464530e-01]
 [4.40163612e-01 5.59836447e-01]
 [6.37425423e-01 3.62574518e-01]
 [1.13729559e-01 8.86270404e-01]
 [3.97999376e-01 6.02000654e-01]
 [6.46976173e-01 3.53023767e-01]
 [1.75767079e-01 8.24232996e-01]
 [5.94944596e-01 4.05055404e-01]
 [1.00000000e+00 1.16443605e-30]
 [3.08209032e-01 6.91790938e-01]
 [4.61661398e-01 5.38338661e-01]
 [6.68024957e-01 3.31975043e-01]
 [6.03867531e-01 3.96132469e-01]
 [5.56907475e-01 4.43092495e-01]
 [6.24364689e-02 9.37563539e-01]
 [1.00000000e+00 2.10321313e-21]
 [6.72776043e-01 3.27223957e-01]
 [4.43109721e-01 5.56890249e-01]
 [4.31849867e-01 5.68150103e-01]
 [4.08198506e-01 5.91801465e-01]
 [5.04122794e-01 4.95877206e-01]
 [5.30877173e-01 4.69122857e-01]
 [4.59461123e-01 5.40538907e-01]
 [3.39484125e-01 6.60515845e-01]
 [3.13559175e-01 6.86440825e-01]
 [3.61704528e-01 6.38295472e-01]
 [5.22267282e-01 4.77732718e-01]
 [4.46620166e-01 5.53379893e-01]
 [4.15529072e-01 5.84470987e-01]
 [5.15741110e-01 4.84258890e-01]
 [5.04840374e-01 4.95159686e-01]
 [5.86883366e-01 4.13116634e-01]
 [2.29610443e-01 7.70389497e-01]
 [5.37198305e-01 4.62801695e-01]
 [6.75594032e-01 3.24405968e-01]
 [6.32633805e-01 3.67366195e-01]
 [3.95070821e-01 6.04929149e-01]
 [3.96931529e-01 6.03068471e-01]
 [3.64052951e-01 6.35947049e-01]
 [1.00000000e+00 7.47773310e-09]
 [4.56570685e-01 5.43429315e-01]
 [4.36170012e-01 5.63829958e-01]
 [1.00000000e+00 3.83393367e-22]
 [3.35342139e-01 6.64657831e-01]
 [1.00000000e+00 4.86515468e-25]
 [3.86484742e-01 6.13515258e-01]
 [5.27148128e-01 4.72851902e-01]
 [3.74404609e-01 6.25595391e-01]
 [3.74115676e-01 6.25884295e-01]
 [6.74646795e-01 3.25353205e-01]
 [4.35660154e-01 5.64339817e-01]
```

```
y_pred= [list(x).index(True) if True in x else 0 for x in prob > 0.5]
```

```
cm = metrics.confusion_matrix(valid_generator.classes, y_pred)
```

```
# or
#cm = np.array([[1401,    0],[1112, 0]])

plt.imshow(cm, cmap=plt.cm.Blues)
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.xticks([], [])
plt.yticks([], [])
plt.title('Confusion matrix ')
plt.colorbar()
plt.show()
```




```
cm = metrics.confusion_matrix(valid_generator.classes, y_pred)
print(cm)
```

```
[[56 89]
 [15 19]]
```

```
print(metrics.classification_report(valid_generator.classes, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.39      0.52       145
           1       0.18      0.56      0.27        34

    accuracy                           0.42       179
   macro avg       0.48      0.47      0.39       179
weighted avg       0.67      0.42      0.47       179
```

```
input_shape = (150, 150, 3)
num_classes = 2

model1 = Sequential()
model1.add(Conv2D(32, kernel_size=(3, 3),activation='relu',padding = 'Same',input_shape=input_shape))
model1.add(Conv2D(32,kernel_size=(3, 3), activation='relu',padding = 'Same',))
model1.add(MaxPool2D(pool_size = (2, 2)))
model1.add(Dropout(0.25))

model1.add(Conv2D(64, (3, 3), activation='relu',padding = 'Same'))
model1.add(Conv2D(64, (3, 3), activation='relu',padding = 'Same'))
model1.add(MaxPool2D(pool_size=(2, 2)))
model1.add(Dropout(0.40))

model1.add(Flatten())
model1.add(Dense(128, activation='relu'))
model1.add(Dropout(0.5))
model1.add(Dense(num_classes, activation='softmax'))
model1.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 150, 150, 32)      896
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 150, 150, 32)      9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 75, 75, 32)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 75, 75, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 75, 75, 64)        18496
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 75, 75, 64)        36928
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 37, 37, 64)        0
```

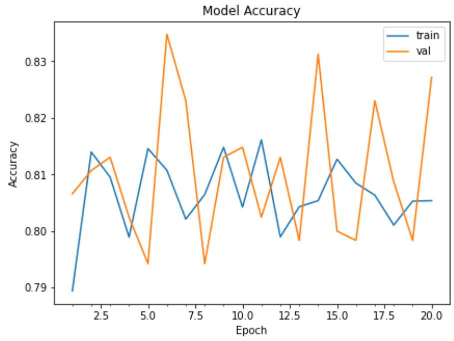
```
model1.compile(optimizer = optimizer , loss = "sparse_categorical_crossentropy", metrics=["accuracy"])
```

```
flatten_2 (Flatten)          (None, 87616)             0
```

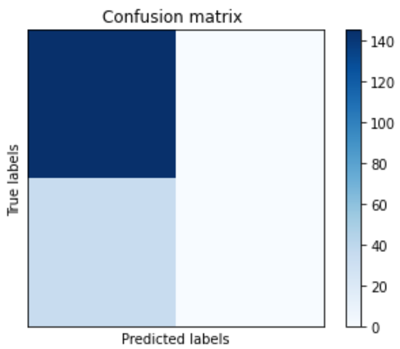
```
epochs = 50
batch_size = 32
#class_weight = {train_generator.class_indices['benign']: 173/900,
#                train_generator.class_indices['malignant']: 727/900}
history = model1.fit(train_generator,
                     steps_per_epoch = np.ceil(944/batch_size),
                     epochs=20,
                     validation_data=valid_generator,
                     validation_steps=np.ceil(235/batch_size),
                    #  class_weight=class_weight,
                     callbacks=[learning_rate_reduction]
                     )
```

```
Epoch 1/20
30/30 [==============================] - 51s 2s/step - loss: 0.5364 - accuracy: 0.7894 - val_loss: 0.6001 - val_accuracy: 0.8066
Epoch 2/20
30/30 [==============================] - 50s 2s/step - loss: 0.4765 - accuracy: 0.8140 - val_loss: 0.5696 - val_accuracy: 0.8107
Epoch 3/20
30/30 [==============================] - 49s 2s/step - loss: 0.4918 - accuracy: 0.8095 - val_loss: 0.6579 - val_accuracy: 0.8130
Epoch 4/20
30/30 [==============================] - 46s 2s/step - loss: 0.4973 - accuracy: 0.7989 - val_loss: 0.6427 - val_accuracy: 0.8025
Epoch 5/20
30/30 [==============================] - 51s 2s/step - loss: 0.4697 - accuracy: 0.8146 - val_loss: 0.5828 - val_accuracy: 0.7942

Epoch 00005: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
Epoch 6/20
30/30 [==============================] - 47s 2s/step - loss: 0.4662 - accuracy: 0.8108 - val_loss: 0.5963 - val_accuracy: 0.8348
Epoch 7/20
30/30 [==============================] - 44s 1s/step - loss: 0.4647 - accuracy: 0.8021 - val_loss: 0.5785 - val_accuracy: 0.8230
Epoch 8/20
30/30 [==============================] - 49s 2s/step - loss: 0.4671 - accuracy: 0.8065 - val_loss: 0.7919 - val_accuracy: 0.7942

Epoch 00008: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
Epoch 9/20
30/30 [==============================] - 48s 2s/step - loss: 0.4550 - accuracy: 0.8148 - val_loss: 0.5003 - val_accuracy: 0.8130
Epoch 10/20
30/30 [==============================] - 43s 1s/step - loss: 0.4621 - accuracy: 0.8042 - val_loss: 0.5754 - val_accuracy: 0.8148
Epoch 11/20
30/30 [==============================] - 48s 2s/step - loss: 0.4355 - accuracy: 0.8161 - val_loss: 0.5431 - val_accuracy: 0.8025
Epoch 12/20
30/30 [==============================] - 50s 2s/step - loss: 0.4630 - accuracy: 0.7990 - val_loss: 0.5541 - val_accuracy: 0.8130

Epoch 00012: ReduceLROnPlateau reducing learning rate to 1e-05.
Epoch 13/20
30/30 [==============================] - 43s 1s/step - loss: 0.4618 - accuracy: 0.8043 - val_loss: 0.5187 - val_accuracy: 0.7984
Epoch 14/20
30/30 [==============================] - 49s 2s/step - loss: 0.4527 - accuracy: 0.8054 - val_loss: 0.5636 - val_accuracy: 0.8313
Epoch 15/20
30/30 [==============================] - 49s 2s/step - loss: 0.4462 - accuracy: 0.8127 - val_loss: 0.8787 - val_accuracy: 0.8000
Epoch 16/20
30/30 [==============================] - 42s 1s/step - loss: 0.4494 - accuracy: 0.8085 - val_loss: 0.7316 - val_accuracy: 0.7984
Epoch 17/20
30/30 [==============================] - 51s 2s/step - loss: 0.4482 - accuracy: 0.8063 - val_loss: 0.7447 - val_accuracy: 0.8230
Epoch 18/20
30/30 [==============================] - 48s 2s/step - loss: 0.4662 - accuracy: 0.8011 - val_loss: 0.5948 - val_accuracy: 0.8087
Epoch 19/20
30/30 [==============================] - 45s 2s/step - loss: 0.4599 - accuracy: 0.8053 - val_loss: 0.6896 - val_accuracy: 0.7984
Epoch 20/20
30/30 [==============================] - 47s 2s/step - loss: 0.4507 - accuracy: 0.8054 - val_loss: 0.5516 - val_accuracy: 0.8272
```

```
plot_model_history(history)
```




```
prob = model1.predict_generator(valid_generator)
y_pred= [list(x).index(True) for x in prob > 0.1]


cm = metrics.confusion_matrix(valid_generator.classes, y_pred)
# or
#cm = np.array([[1401,    0],[1112, 0]])

plt.imshow(cm, cmap=plt.cm.Blues)
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.xticks([], [])
plt.yticks([], [])
plt.title('Confusion matrix ')
plt.colorbar()
plt.show()

print(cm)
```




```
[[145   0]
 [ 34   0]]
```

```
import keras
from keras.applications import VGG19
from keras.applications.vgg19 import preprocess_input
from keras.layers import Dense, Dropout
from keras.models import Model
from keras import models
from keras import layers
from keras import optimizers

# Create the base model of VGG19
vgg19 = VGG19(weights='imagenet', include_top=False, input_shape = (150, 150, 3), classes = 10)
```

```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg19_weights_tf_dim_ordering_tf_ke
80142336/80134624 [==============================] - 1s 0us/step
```

```
# Preprocessing the input
# X_train = preprocess_input(train_generator)
# X_val = preprocess_input(valid_generator)
# X_test = preprocess_input(X_test)

# Extracting features
train_features = vgg19.predict_generator(train_generator, verbose=1)
# test_features = vgg19.predict(np.array(X_test), batch_size=256, verbose=1)
val_features = vgg19.predict_generator(valid_generator,  verbose=1)

# # Flatten extracted features
```

```
# train_features = np.reshape(train_features, (721, 4*4*512))
# # test_features = np.reshape(test_features, (10000, 4*4*512))
# val_features = np.reshape(val_features, (235, 4*4*512))
```

```
23/23 [==============================] - 37s 2s/step
6/6 [==============================] - 7s 1s/step
```

```
# Flatten extracted features
train_features = np.reshape(train_features, (721, 4*4*512))
# test_features = np.reshape(test_features, (10000, 4*4*512))
val_features = np.reshape(val_features, (179, 4*4*512))
```

```
# Add Dense and Dropout layers on top of VGG19 pre-trained
modelVG = models.Sequential()
modelVG.add(layers.Dense(512, activation='relu', input_dim=4 * 4 * 512))
modelVG.add(layers.Dropout(0.5))
modelVG.add(layers.Dense(2, activation="softmax"))
```

```
# Compile the model
modelVG.compile(loss=keras.losses.sparse_categorical_crossentropy,
              optimizer=keras.optimizers.Adam(),
              metrics=['accuracy'])
```

```
modelVG.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 512)               4194816
_________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 2)                 1026
=================================================================
Total params: 4,195,842
Trainable params: 4,195,842
Non-trainable params: 0
_________________________________________________________________
```

```
epochs = 50
batch_size = 32
#class_weight = {train_generator.class_indices['benign']: 173/900,
                # train_generator.class_indices['malignant']: 727/900}
history = modelVG.fit(train_features, train_generator.classes,
                     steps_per_epoch = int(np.ceil(721/batch_size)),
                     epochs=50,
                     validation_data=(val_features, valid_generator.classes),
                     validation_steps=int(np.ceil(179/batch_size))
                    #  class_weight=class_weight,
                    #  callbacks=[learning_rate_reduction]
                     )
```

```
Train on 721 samples, validate on 179 samples
Epoch 1/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0470 - accuracy: 0.9926 - val_loss: 0.2261 - val_accuracy: 4.6592
Epoch 2/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0413 - accuracy: 0.9952 - val_loss: 0.2373 - val_accuracy: 4.6592
Epoch 3/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0402 - accuracy: 0.9949 - val_loss: 0.2415 - val_accuracy: 4.5922
Epoch 4/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0353 - accuracy: 0.9963 - val_loss: 0.2473 - val_accuracy: 4.6592
Epoch 5/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0336 - accuracy: 0.9966 - val_loss: 0.2521 - val_accuracy: 4.5587
Epoch 6/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0304 - accuracy: 0.9979 - val_loss: 0.2551 - val_accuracy: 4.5251
Epoch 7/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0287 - accuracy: 0.9983 - val_loss: 0.2609 - val_accuracy: 4.5251
Epoch 8/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0258 - accuracy: 0.9979 - val_loss: 0.2655 - val_accuracy: 4.5251
Epoch 9/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0246 - accuracy: 0.9984 - val_loss: 0.2844 - val_accuracy: 4.5922
Epoch 10/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0227 - accuracy: 0.9986 - val_loss: 0.2861 - val_accuracy: 4.5251
Epoch 11/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0196 - accuracy: 0.9996 - val_loss: 0.2946 - val_accuracy: 4.5922
Epoch 12/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0195 - accuracy: 0.9992 - val_loss: 0.2968 - val_accuracy: 4.5922
Epoch 13/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0186 - accuracy: 0.9993 - val_loss: 0.3100 - val_accuracy: 4.6257
Epoch 14/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0180 - accuracy: 0.9992 - val_loss: 0.3073 - val_accuracy: 4.5251
Epoch 15/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0164 - accuracy: 0.9994 - val_loss: 0.3085 - val_accuracy: 4.5251
Epoch 16/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0149 - accuracy: 0.9995 - val_loss: 0.3160 - val_accuracy: 4.4916
Epoch 17/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0141 - accuracy: 0.9998 - val_loss: 0.3241 - val_accuracy: 4.4916
Epoch 18/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0134 - accuracy: 0.9997 - val_loss: 0.3243 - val_accuracy: 4.4916
Epoch 19/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0126 - accuracy: 0.9996 - val_loss: 0.3318 - val_accuracy: 4.5251
Epoch 20/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0124 - accuracy: 0.9995 - val_loss: 0.3440 - val_accuracy: 4.5922
Epoch 21/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0126 - accuracy: 0.9999 - val_loss: 0.3390 - val_accuracy: 4.6257
Epoch 22/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0116 - accuracy: 0.9993 - val_loss: 0.3444 - val_accuracy: 4.5922
Epoch 23/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0102 - accuracy: 0.9998 - val_loss: 0.3481 - val_accuracy: 4.5922
Epoch 24/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0103 - accuracy: 0.9999 - val_loss: 0.3448 - val_accuracy: 4.4916
Epoch 25/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0098 - accuracy: 0.9998 - val_loss: 0.3518 - val_accuracy: 4.5587
Epoch 26/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0094 - accuracy: 0.9999 - val_loss: 0.3607 - val_accuracy: 4.5251
Epoch 27/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0091 - accuracy: 0.9998 - val_loss: 0.3645 - val_accuracy: 4.5251
Epoch 28/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0089 - accuracy: 0.9998 - val_loss: 0.3735 - val_accuracy: 4.5587
Epoch 29/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0087 - accuracy: 0.9998 - val_loss: 0.3803 - val_accuracy: 4.5922
Epoch 30/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0083 - accuracy: 0.9996 - val_loss: 0.3795 - val_accuracy: 4.5922
Epoch 31/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0078 - accuracy: 0.9999 - val_loss: 0.3779 - val_accuracy: 4.4916
Epoch 32/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0074 - accuracy: 0.9998 - val_loss: 0.3854 - val_accuracy: 4.5587
Epoch 33/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0076 - accuracy: 0.9998 - val_loss: 0.4015 - val_accuracy: 4.6592
Epoch 34/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0078 - accuracy: 0.9996 - val_loss: 0.4014 - val_accuracy: 4.5587
Epoch 35/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0067 - accuracy: 0.9999 - val_loss: 0.3939 - val_accuracy: 4.5587
Epoch 36/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0067 - accuracy: 0.9999 - val_loss: 0.3914 - val_accuracy: 4.4916
Epoch 37/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0061 - accuracy: 0.9998 - val_loss: 0.4046 - val_accuracy: 4.5251
Epoch 38/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0052 - accuracy: 0.9999 - val_loss: 0.4141 - val_accuracy: 4.6592
Epoch 39/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0057 - accuracy: 0.9999 - val_loss: 0.4152 - val_accuracy: 4.5922
Epoch 40/50
23/23 [==============================] - 0s 9ms/step - loss: 0.0055 - accuracy: 0.9999 - val_loss: 0.4135 - val_accuracy: 4.6257
```

```
val_features.shape
```

```
(179, 8192)
Epoch 43/50
```

```
train_features.shape
```

```
(721, 8192)
```

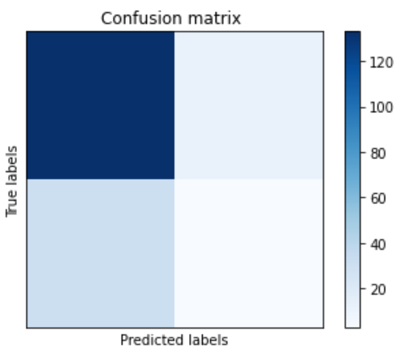
```
train_generator.classes
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int32)
```

```
import matplotlib.pyplot as plt
import numpy as np
import sklearn.metrics as metrics
prob = modelVG.predict(val_features)
```

```
y_pred= [list(x).index(True) for x in prob > 0.5]
```

```
cm = metrics.confusion_matrix(valid_generator.classes, y_pred)
# or
#cm = np.array([[1401,    0],[1112, 0]])

plt.imshow(cm, cmap=plt.cm.Blues)
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.xticks([], [])
plt.yticks([], [])
plt.title('Confusion matrix ')
plt.colorbar()
plt.show()
```




```
cm = metrics.confusion_matrix(valid_generator.classes, y_pred)
print(cm)
```

```
[[133  12]
 [ 31   3]]
```

```
plot_model_history(history)
```