### CSCI E-82 Homework 5 on CNNs

### Due by 11/13/18 at 11:59pm EST to the Canvas dropbox

## This is an individual homework so there should be no collaboration for this homework.

### Under each problem, we have a place for you to write the answer, or write runnable code that will produce the answer. Show your work.

This is a busy time of year with homework and an exam coming up. We are looking for a successful working result that builds upon the section code and enables you to gain some proficiency with this important and growing field of deep learning.

Depending on your computer, some of the runs may still take a few minutes per epoch. As a result, Problem 4 may take the better part of a day to run, so plan accordingly.

## Your Name: SHARJIL KHAN

In [60]:
```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.python.client import device_lib

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
from time import time
import shutil
import sys
from IPython.display import display, Image

from sklearn.metrics import auc
from sklearn.metrics import roc_curve


from keras import layers
from keras import models
from keras import optimizers
from keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.callbacks import TensorBoard
from tensorflow.python.eager import context

from keras.preprocessing import image
from keras.utils import layer_utils
from keras.utils.data_utils import get_file
from keras.applications.imagenet_utils import preprocess_input
from keras.callbacks import TensorBoard

from keras.layers import Input, Dense, Activation, ZeroPadding2D, BatchNormali
zation, Flatten, Conv2D
from keras.layers import AveragePooling2D, MaxPooling2D, Dropout
from keras.models import Model, Sequential
from keras.optimizers import Adam,RMSprop,SGD


from keras import backend as K
if K.backend()=='tensorflow':
    K.set_image_data_format('channels_last')

# Config the matlotlib backend as plotting inline in IPython
%matplotlib inline

print("Tensorflow is installed and is version: ",  tf.__version__)
print("Keras is installed and is version: ", tf.keras.__version__)

with tf.device('/gpu:0'):
    a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
    b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
    c = tf.matmul(a, b)

with tf.Session() as sess:
    print (sess.run(c))
```

```
Tensorflow is installed and is version:  1.11.0
Keras is installed and is version:  2.1.6-tf
[[22. 28.]
 [49. 64.]]
```

In [61]:
```
class TrainValTensorBoard(TensorBoard):
    def __init__(self, log_dir='./logs', **kwargs):
        # Make the original `TensorBoard` log to a subdirectory 'training'
        training_log_dir = os.path.join(log_dir, 'training')
        super(TrainValTensorBoard, self).__init__(training_log_dir, **kwargs)

        # Log the validation metrics to a separate subdirectory
        self.val_log_dir = os.path.join(log_dir, 'validation')

    def set_model(self, model):
        # Setup writer for validation metrics
        self.val_writer = tf.summary.FileWriter(self.val_log_dir)
        super(TrainValTensorBoard, self).set_model(model)

    def on_epoch_end(self, epoch, logs=None):
        # Pop the validation logs and handle them separately with
        # `self.val_writer`. Also rename the keys so that they can
        # be plotted on the same figure with the training metrics
        logs = logs or {}
        val_logs = {k.replace('val_', ''): v for k, v in logs.items() if k.sta
rtswith('val_')}
        for name, value in val_logs.items():
            summary = tf.Summary()
            summary_value = summary.value.add()
            summary_value.simple_value = value.item()
            summary_value.tag = name
            self.val_writer.add_summary(summary, epoch)
        self.val_writer.flush()

        # Pass the remaining logs to `TensorBoard.on_epoch_end`
        logs = {k: v for k, v in logs.items() if not k.startswith('val_')}
        super(TrainValTensorBoard, self).on_epoch_end(epoch, logs)

    def on_train_end(self, logs=None):
        super(TrainValTensorBoard, self).on_train_end(logs)
        self.val_writer.close()


def plot_accuracies_loss(history):
    acc = history.history['acc']
    val_acc = history.history['val_acc']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs = range(len(acc))

    plt.plot(epochs, acc, 'g-', label='Training acc', color = 'brown')
    plt.plot(epochs, val_acc, 'g-', label='Validation acc', color = 'orange')
    plt.xlabel("Num of Epochs")
    plt.ylabel("Accuracy")
    plt.title('Training and validation accuracy')
    plt.legend()

    plt.figure()

    plt.plot(epochs, loss, 'g-', label='Training loss', color = 'brown')
```

```
    plt.plot(epochs, val_loss, 'g-', label='Validation loss', color = 'orange'
)
    plt.xlabel("Num of Epochs")
    plt.ylabel("Loss")
    plt.title('Training and validation loss')
    plt.legend()

    plt.show()
```

In [62]:
```
# special matplotlib command for global plot configuration
from matplotlib import rcParams
import matplotlib.cm as cm
import matplotlib as mpl
from matplotlib.colors import ListedColormap
from mpl_toolkits.mplot3d import Axes3D

dark2_colors = [(0.10588235294117647, 0.6196078431372549, 0.4666666666666667),
                (0.9058823529411765, 0.1607843137254902, 0.5411764705882353),
                (0.8509803921568627, 0.37254901960784315, 0.00784313725490196
),
                (0.4588235294117647, 0.4392156862745098, 0.7019607843137254),

                (0.4, 0.6509803921568628, 0.11764705882352941),
                (0.9019607843137255, 0.6705882352941176, 0.00784313725490196),
                (0.6509803921568628, 0.4627450980392157, 0.11372549019607843)]

cmap_set1 = ListedColormap(['#e41a1c', '#377eb8', '#4daf4a'])
dark2_cmap=ListedColormap(dark2_colors)

def set_mpl_params():
    rcParams['figure.figsize'] = (12, 6)
    rcParams['figure.dpi'] = 100
    rcParams['axes.prop_cycle'].by_key()['color'][1]
    rcParams['lines.linewidth'] = 2
    rcParams['axes.facecolor'] = 'white'
    rcParams['font.size'] = 14
    rcParams['patch.edgecolor'] = 'white'
    rcParams['patch.facecolor'] = dark2_colors[0]
    rcParams['font.family'] = 'StixGeneral'

set_mpl_params()
```

In [63]:
```
print(sys.version)
print(device_lib.list_local_devices())
```

```
3.6.5 |Anaconda custom (64-bit)| (default, Apr 29 2018, 16:14:56)
[GCC 7.2.0]
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 11104394809493091072
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 11280557671
locality {
  bus_id: 1
  links {
  }
}
incarnation: 11165909729231536491
physical_device_desc: "device: 0, name: Tesla K80, pci bus id: 0000:00:1e.0,
compute capability: 3.7"
]
```

In [64]:
```
## Given a base directory it creates two directories and uses the truth.txt
## file to split the files into these two directories
def organize_folders (base_dir):
    truth_file = os.path.join(base_dir,'truth.txt')
    portrait_dir = os.path.join(base_dir, 'portrait')
    landscape_dir = os.path.join(base_dir, 'landscape')

    # CREATE THE TWO FOLDERS IF THEY DONT EXIST ALREADY
    if not os.path.exists(portrait_dir):
             os.makedirs(portrait_dir)
    if not os.path.exists(landscape_dir):
             os.makedirs(landscape_dir)

    with open(truth_file) as f:
        for line in f:
            data = line.split()
            image_file = os.path.join(base_dir, data[0])
            image_type = data[1]
            if os.path.exists(image_file):
                if os.path.getsize(image_file) > 0:
                        if image_type == 'portrait':
                              shutil.move(image_file, portrait_dir)
                        if image_type == 'landscape':
                              shutil.move(image_file, landscape_dir)
                print ("Moving:%s to %s"%(image_file,image_type))

# SPECIFY THE THREE BASE DIRECTORIES
train_dir = './images64/train'
validation_dir = './images64/validation'
test_dir = './images64/test'

# ORGANISE EACH DIRECTORY INTO PORTRAIT AND LANDSCAPE DIRs
organize_folders(train_dir)
organize_folders(validation_dir)
organize_folders(test_dir)

# All images will be rescaled by 1./255
datagen = ImageDataGenerator(rescale=1./255)

# generator for the training data
train_generator = datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 64x64
        target_size=(64, 64),
        batch_size=32,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

# generator for the validation data
validation_generator = datagen.flow_from_directory(
        validation_dir,
        target_size=(64, 64),
        batch_size=32,
        class_mode='binary')
```

```
# generator for the test data
test_generator = datagen.flow_from_directory(
        test_dir,
        target_size=(64, 64),
        # NO shuffle for the test set to be able to track the file names
        shuffle = False,
        batch_size=32,  # this just makes sure it gets through memory
        class_mode='binary')
```

```
Found 16315 images belonging to 2 classes.
Found 8158 images belonging to 2 classes.
Found 7379 images belonging to 2 classes.
```

# Dataset

WikiArt is an amazing resource containing centuries of artwork. Since such datasets are wonderful for deep learning, Kaggle has hosted a challenge to characterize the 'fingerprints' of various artists. The Kaggle dataset contains metadata and also a set of images that have been resized so that the shorter dimension is 256 pixels. To make this homework reasonably fast even for those without GPUs, we have further reduced the images to 64 x 64. CNNs and neural networks in general prefer to have consistent sizes. To achieve this, we cut the center 256 pixels from the longer dimension and then shrunk the images by a factor of 4. This isn't a perfect solution since it did cut off a few heads as you will see.

The selected images are for portraits and landscapes. No, we're not talking about the orientation but rather the content of the images. Thanks to help from Rashmi and Dave, we have a small enough data set that should give reasonable results in a timely manner even on just a CPU.

The data were originally divided into a training and a test set. We have further divided the training set into a train and validation set. In this homework you will be using the training set and validation set to train and assess your deep learning models. At the final step, you will see how well your final training worked on the test set. In each of these directories, there is a truth.txt file that has the image name and whether it is a portrait or landscape scene.

# Problem 1 (5 points)

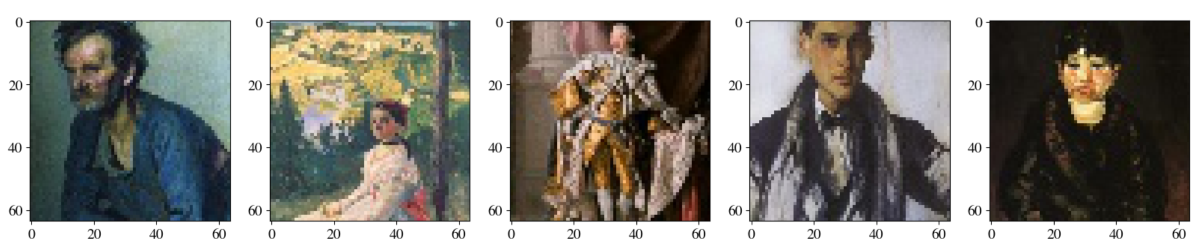
Read in and display the first 5 portraits and the first 5 landscapes. Note, if you are using the OpenCV tools, then the color may be distorted. The cvtColor() method using cv2.COLOR_BGR2RGB may be useful. However, it is likely easier to use the generator and plot_strip example from section.

In [65]:
```
# USE THE TRAIN DATA GENERATOR CREATED ABOVE TO GENERATE FIRST FIVE PORTRAITS
 AND LANDSCAPES
for data_batch, labels_batch in train_generator:
    print('data batch shape:', data_batch.shape)
    print('labels batch shape:', labels_batch.shape)
    break

first_five_portraits = []
first_five_landscapes = []
for i, j in enumerate (range(len(labels_batch))):
    if labels_batch[j] == 1:
        first_five_portraits.append(data_batch[j,:,:])
    if labels_batch[j] == 0:
        first_five_landscapes.append(data_batch[j,:,:])
    if i == 10:
        break
```

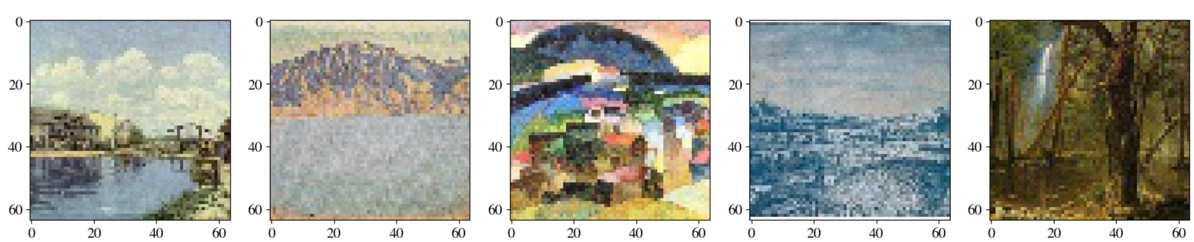
```
data batch shape: (32, 64, 64, 3)
labels batch shape: (32,)
```

In [66]:
```
# FUNCTION TO PLOT OUT THE FIRST FIVE IMAGES FROM A LIST
def plot_first_five (lst):
    plt.rcParams['figure.figsize'] = (20.0, 20.0)
    f, ax = plt.subplots(nrows=1, ncols=5)
    for i in range(5):
        image = lst[i]
        ax[i].imshow(image)
    plt.show()
    return

print("FIRST FIVE PORTRAITS GENERATED BY THE TRAINING GENERATOR:")
plot_first_five(first_five_portraits)
print("FIRST FIVE LANDSCAPES GENERATED BY THE TRAINING GENERATOR:")
plot_first_five(first_five_landscapes)
```

```
FIRST FIVE PORTRAITS GENERATED BY THE TRAINING GENERATOR:
```



```
FIRST FIVE LANDSCAPES GENERATED BY THE TRAINING GENERATOR:
```



# Problem 2 (25 points)

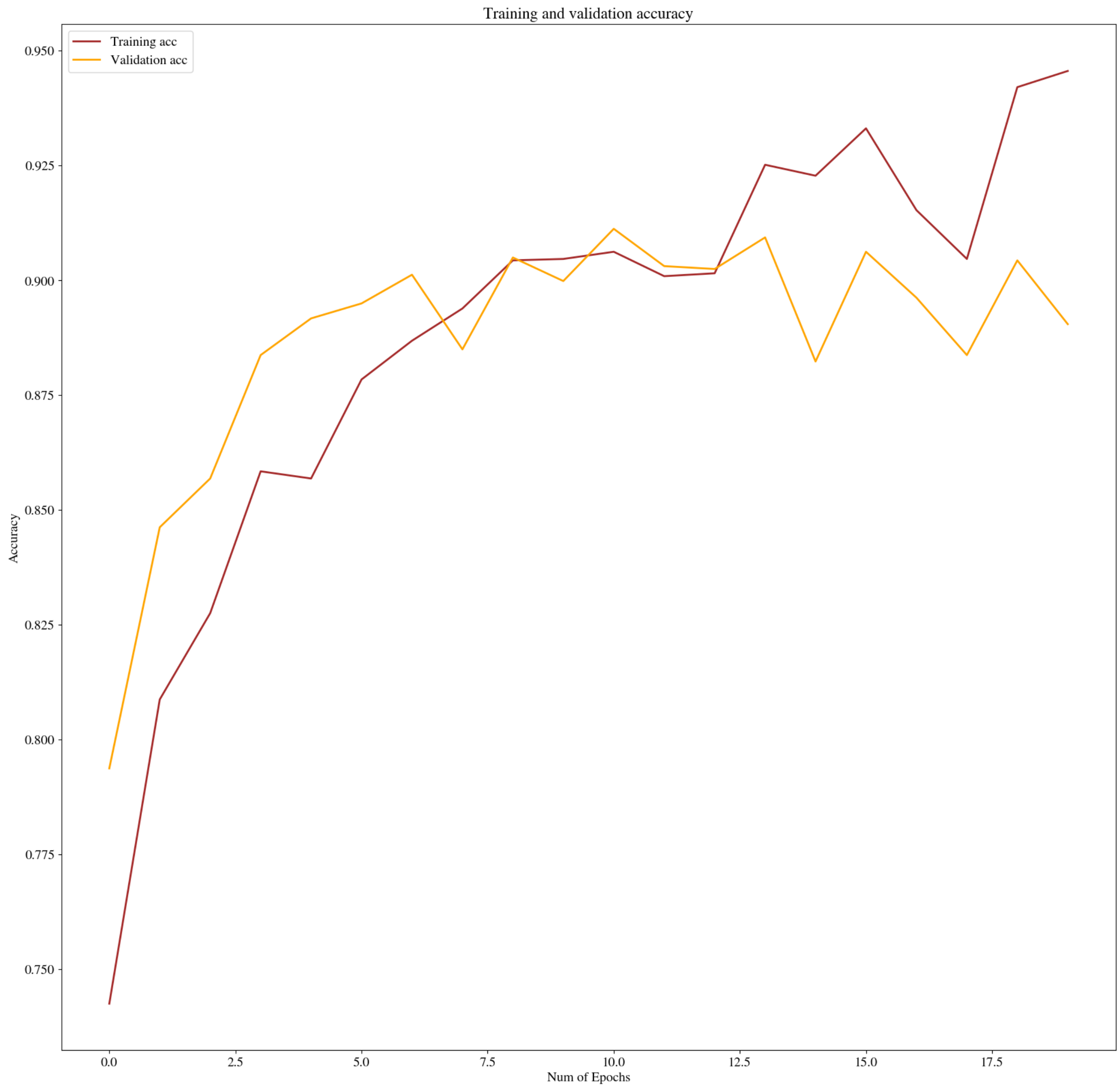
Construct a baseline CNN classifier using Keras for the training set and assess the validation set performance at each epoch. The goal is to correctly classify portraits from landscapes. Plot the resulting performance on the training and validation set as a function of epoch using the criteria over which you are optimizing. You should run at least 20 epochs for this problem.

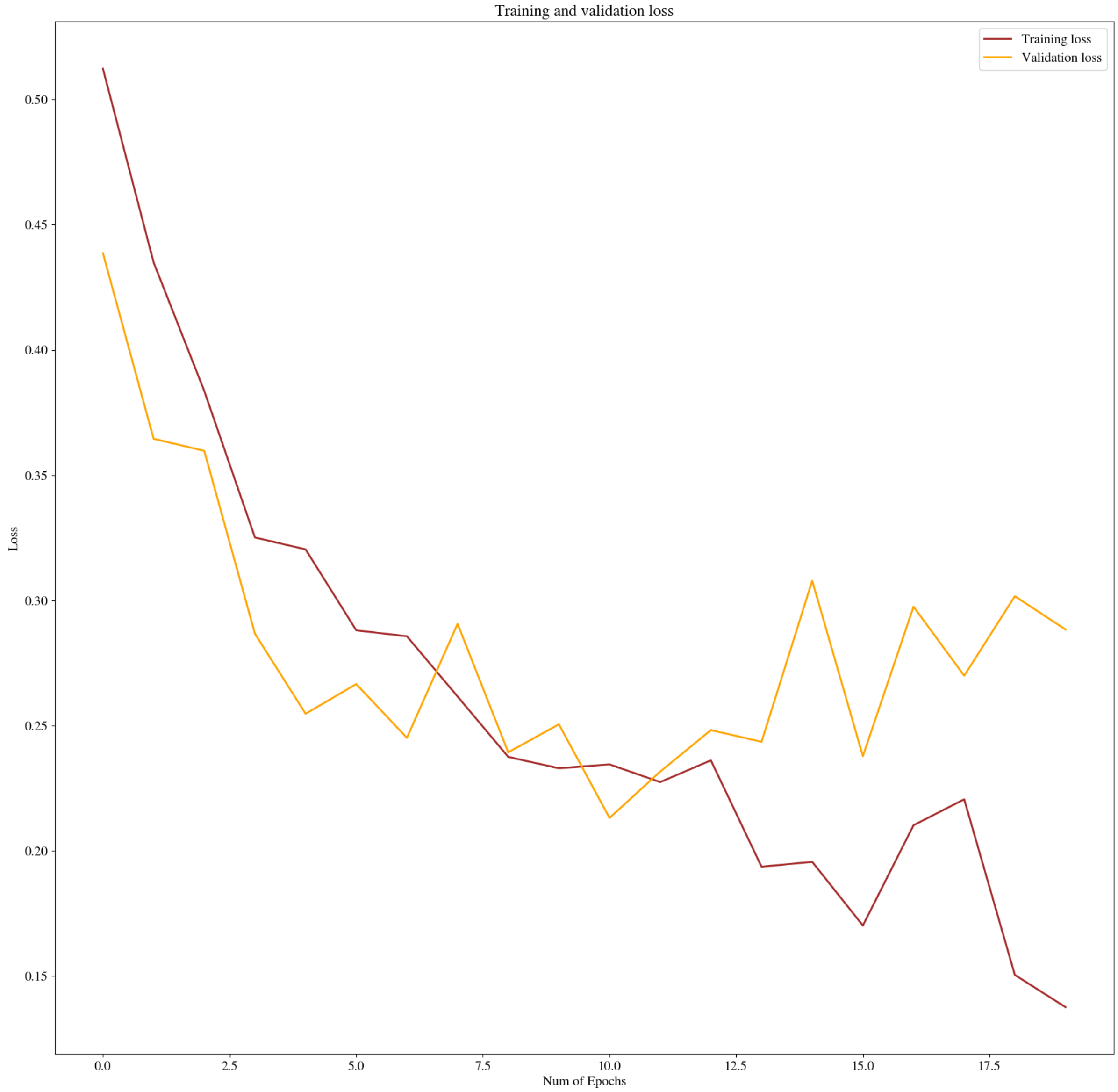
In [59]:
```
K.clear_session()

#tensorboard = TensorBoard(log_dir="logs/{}".format(time()))
tensorboard = TrainValTensorBoard("logs/{}".format(time()), write_graph=True)

def simple_model():
    model = Sequential(name='SimpleModel')
    model.add(Conv2D(32, (2, 2), strides = (1, 1), padding='same',activation=
'relu',
                            input_shape=data_batch.shape[1:], name = 'conv1'))
    model.add(MaxPooling2D((2, 2), strides=2, padding='valid', name='max_pool
1'))
    model.add(Conv2D(64, (2, 2), strides = (1, 1), padding='same', activation=
'relu', name = 'conv2'))
    model.add(MaxPooling2D((2, 2), strides=2, padding='valid', name='max_pool
2'))
    model.add(Flatten())
    model.add(Dense(1024, kernel_initializer='glorot_uniform', activation='rel
u', name='fc1'))
    model.add(Dense(1, kernel_initializer='glorot_uniform', activation='sigmoi
d', name='fc2'))

    sgd = SGD(lr = 0.05, decay=1e-6, momentum=0.9, nesterov=True)

    model.compile(loss='binary_crossentropy',
                  optimizer=sgd,
                  metrics=['accuracy'])
    return model

cnn1_simple = simple_model()
cnn1_simple.summary()

# Fit model
history = cnn1_simple.fit_generator(
    train_generator,
    steps_per_epoch=100,
    epochs=20,
    validation_data=validation_generator,
    validation_steps=50,
    verbose=1,
    callbacks=[tensorboard])
```

```
Layer (type)                 Output Shape              Param #   
=================================================================
conv1 (Conv2D)               (None, 64, 64, 32)        416       
_________________________________________________________________
max_pool1 (MaxPooling2D)     (None, 32, 32, 32)        0         
_________________________________________________________________
conv2 (Conv2D)               (None, 32, 32, 64)        8256      
_________________________________________________________________
max_pool2 (MaxPooling2D)     (None, 16, 16, 64)        0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 16384)             0         
_________________________________________________________________
fc1 (Dense)                  (None, 1024)              16778240  
_________________________________________________________________
fc2 (Dense)                  (None, 1)                 1025      
=================================================================
Total params: 16,787,937
Trainable params: 16,787,937
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
100/100 [==============================] - 3s 30ms/step - loss: 0.5123 - acc: 0.7425 - val_loss: 0.4387 - val_acc: 0.7937
Epoch 2/20
100/100 [==============================] - 3s 27ms/step - loss: 0.4350 - acc: 0.8088 - val_loss: 0.3645 - val_acc: 0.8462
Epoch 3/20
100/100 [==============================] - 3s 27ms/step - loss: 0.3836 - acc: 0.8275 - val_loss: 0.3597 - val_acc: 0.8569
Epoch 4/20
100/100 [==============================] - 3s 27ms/step - loss: 0.3251 - acc: 0.8584 - val_loss: 0.2867 - val_acc: 0.8838
Epoch 5/20
100/100 [==============================] - 3s 27ms/step - loss: 0.3204 - acc: 0.8569 - val_loss: 0.2547 - val_acc: 0.8917
Epoch 6/20
100/100 [==============================] - 3s 27ms/step - loss: 0.2881 - acc: 0.8784 - val_loss: 0.2666 - val_acc: 0.8950
Epoch 7/20
100/100 [==============================] - 3s 27ms/step - loss: 0.2857 - acc: 0.8869 - val_loss: 0.2451 - val_acc: 0.9012
Epoch 8/20
100/100 [==============================] - 3s 27ms/step - loss: 0.2617 - acc: 0.8938 - val_loss: 0.2907 - val_acc: 0.8850
Epoch 9/20
100/100 [==============================] - 3s 27ms/step - loss: 0.2375 - acc: 0.9044 - val_loss: 0.2394 - val_acc: 0.9050
Epoch 10/20
100/100 [==============================] - 3s 27ms/step - loss: 0.2330 - acc: 0.9047 - val_loss: 0.2505 - val_acc: 0.8999
Epoch 11/20
100/100 [==============================] - 3s 27ms/step - loss: 0.2345 - acc: 0.9062 - val_loss: 0.2132 - val_acc: 0.9113
Epoch 12/20
100/100 [==============================] - 3s 27ms/step - loss: 0.2275 - acc: 0.9009 - val_loss: 0.2317 - val_acc: 0.9031
```

```
Epoch 13/20
100/100 [==============================] - 3s 27ms/step - loss: 0.2361 - acc:
0.9016 - val_loss: 0.2482 - val_acc: 0.9025
Epoch 14/20
100/100 [==============================] - 3s 27ms/step - loss: 0.1936 - acc:
0.9253 - val_loss: 0.2436 - val_acc: 0.9094
Epoch 15/20
100/100 [==============================] - 3s 27ms/step - loss: 0.1956 - acc:
0.9228 - val_loss: 0.3079 - val_acc: 0.8824
Epoch 16/20
100/100 [==============================] - 3s 27ms/step - loss: 0.1702 - acc:
0.9331 - val_loss: 0.2378 - val_acc: 0.9062
Epoch 17/20
100/100 [==============================] - 3s 27ms/step - loss: 0.2102 - acc:
0.9153 - val_loss: 0.2975 - val_acc: 0.8962
Epoch 18/20
100/100 [==============================] - 3s 27ms/step - loss: 0.2206 - acc:
0.9047 - val_loss: 0.2699 - val_acc: 0.8838
Epoch 19/20
100/100 [==============================] - 3s 27ms/step - loss: 0.1504 - acc:
0.9421 - val_loss: 0.3017 - val_acc: 0.9044
Epoch 20/20
100/100 [==============================] - 3s 27ms/step - loss: 0.1376 - acc:
0.9456 - val_loss: 0.2884 - val_acc: 0.8905
```

In [68]:
```
plot_accuracies_loss(history)
```


Training and validation accuracy
Training acc
Validation acc
Accuracy
Num of Epochs
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5




# Problem 3 (5 points)

From the pattern of training and validation curves, describe what is good/bad and what you plan to do next to improve the result.

From the above plots, we can see that we are achieving fairly good validation accuracy of about 89 % with this simple model.
However there are a few things to notice:

1. We are using a kernel size of 2X2 with just two convolution layers of 32 and 64 filters in this simple model. We can certainly make this model more complex and hope to achieve even higher scores.
2. There is definately overfitting which starts to happen beyond about 11 epochs:
   - We see training scores rising while validation scores beginning to decline. We can add some pooling and regularization to help with the overfitting.
3. Accuracy and Loss curves hasn't quite flattened out completely at 20 epochs. So after solving the overfitting issue, training for larger epochs might also tune the parameters a little more to achieve higher scores.

STRATEGY SUMMARY:

1. So my strategy will be to add some more layers and see the results of increasing or decreasing the kernel size and number of filters/maps in each layer.
2. Then, I will fix any overfitting using pooling, batch normalisation & simplifying the model if necessary.
3. Then, I will try to change the activation functions, optimizers and also run for longer epochs to see if scores can be improved.

# Problem 4 (45 points)

This step is where we want you to do most of your personal learning. Your goal is to improve the network using a combination of architecture choices, parameter tuning, and experimenting with different optimizers/dropout/regularization/etc. Treat each of these as separate optimization/exploration steps for now. We would like to see 3 separate steps that cover different areas. The format of the 3 steps should be as follows:

- State the hypothesis/strategy for how you will improve/explore a particular aspect.
- Describe what types of tests you are running and why (i.e. what range of parameters are you choosing and why)
- Include the code and results
- State your interpretation of the results

We're not looking for research in deep learning, but we want you to gain some hands-on experience working with Keras and figuring out what works. A good example may be comparing strategies to overcome overfitting, or comparing a few different CNN architectures in terms of performance and speed, or comparing data augmentation types and results.

### Strategy 1: Increase Model Complexity to try to achieve higher scores

```
- Add 4 more layers with increasingly larger number of filters as we narrow down th
e map size with a few pooling layers
  in between.
- Also increased the number of epochs we are training the parameters for to see if
 accuracies would eventually increase.
```

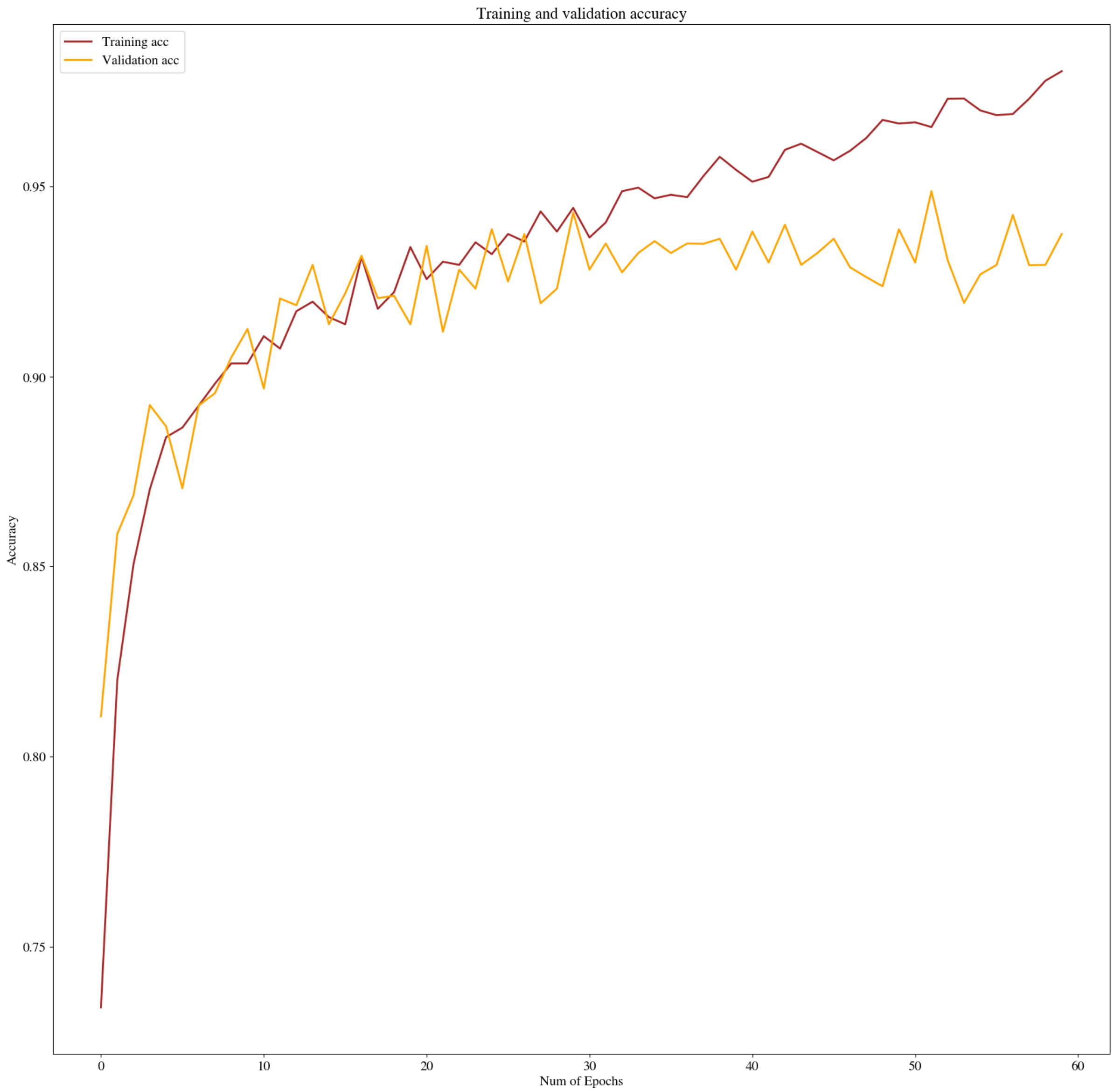
In [71]:
```
#tensorboard = TensorBoard(log_dir="logs/{}".format(time()))
K.clear_session()

def more_complex_model():

    model = Sequential(name='FiveLayerModel')
    model.add(Conv2D(32, (3, 3), padding='same', activation='relu',
                            input_shape=data_batch.shape[1:], name = 'conv1'))
    model.add(MaxPooling2D((2, 2), name='max_pool1'))
    model.add(Conv2D(64, (3, 3), padding='same', activation='relu', name = 'co
nv2'))
    model.add(MaxPooling2D((2, 2), name='max_pool2'))
    model.add(Conv2D(128, (3, 3), padding='same', activation='relu', name = 'c
onv3'))
    model.add(MaxPooling2D((2, 2), name='max_pool3'))
    model.add(Conv2D(128, (3, 3), padding='same', activation='relu', name = 'c
onv4'))
    model.add(Conv2D(256, (3, 3), padding='same', activation='relu', name = 'c
onv5'))
    model.add(Conv2D(512,(3, 3), padding='same', activation='relu', name = 'co
nv6'))
    model.add(MaxPooling2D((2, 2), name='max_pool6'))
    model.add(Flatten())
    model.add(Dense(512, kernel_initializer='glorot_uniform', activation='rel
u', name='fc1'))
    model.add(Dense(1, kernel_initializer='glorot_uniform', activation='sigmoi
d', name='fc2'))

    model.compile(loss='binary_crossentropy',
                  optimizer=optimizers.RMSprop(lr=1e-4),
                  metrics=['acc'])
    #model.compile(optimizer="adam", loss="binary_crossentropy", metrics = ["a
ccuracy"])
    return model

cnn_more_complex = more_complex_model()
cnn_more_complex.summary()

# Fit model
history_complex = cnn_more_complex.fit_generator(
    train_generator,
    steps_per_epoch=100,
    epochs=60,
    validation_data=validation_generator,
    validation_steps=50,
    verbose=1,
    callbacks=[TrainValTensorBoard("logs/{}".format(time()), write_graph=True
)])

plot_accuracies_loss(history_complex)
```

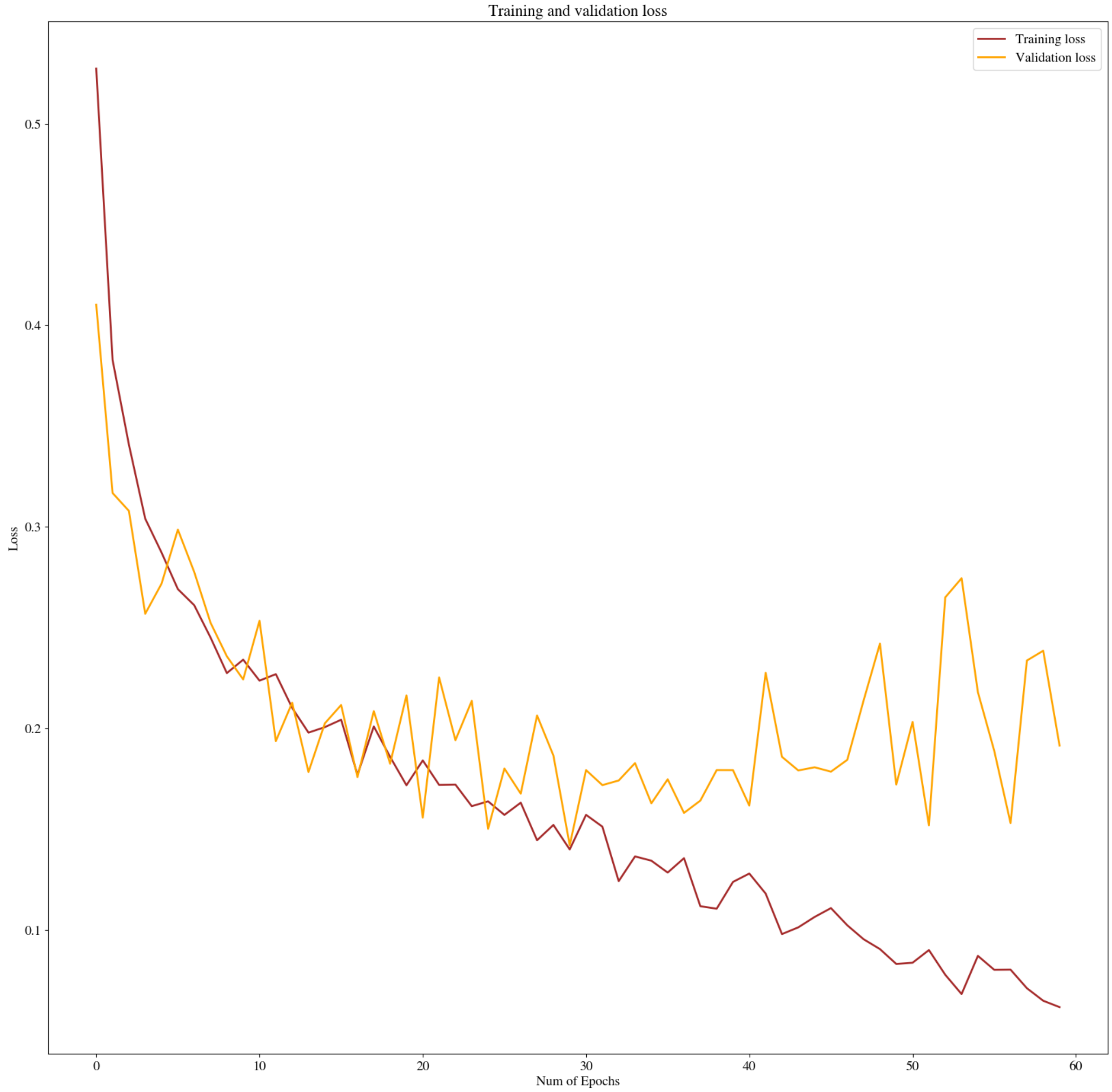
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1 (Conv2D)               (None, 64, 64, 32)        896       
_________________________________________________________________
max_pool1 (MaxPooling2D)     (None, 32, 32, 32)        0         
_________________________________________________________________
conv2 (Conv2D)               (None, 32, 32, 64)        18496     
_________________________________________________________________
max_pool2 (MaxPooling2D)     (None, 16, 16, 64)        0         
_________________________________________________________________
conv3 (Conv2D)               (None, 16, 16, 128)       73856     
_________________________________________________________________
max_pool3 (MaxPooling2D)     (None, 8, 8, 128)         0         
_________________________________________________________________
conv4 (Conv2D)               (None, 8, 8, 128)         147584    
_________________________________________________________________
conv5 (Conv2D)               (None, 8, 8, 256)         295168    
_________________________________________________________________
conv6 (Conv2D)               (None, 8, 8, 512)         1180160   
_________________________________________________________________
max_pool6 (MaxPooling2D)     (None, 4, 4, 512)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0         
_________________________________________________________________
fc1 (Dense)                  (None, 512)               4194816   
_________________________________________________________________
fc2 (Dense)                  (None, 1)                 513       
=================================================================
Total params: 5,911,489
Trainable params: 5,911,489
Non-trainable params: 0
_________________________________________________________________
Epoch 1/60
100/100 [==============================] - 4s 40ms/step - loss: 0.5273 - acc: 0.7341 - val_loss: 0.4102 - val_acc: 0.8106
Epoch 2/60
100/100 [==============================] - 4s 36ms/step - loss: 0.3827 - acc: 0.8201 - val_loss: 0.3168 - val_acc: 0.8586
Epoch 3/60
100/100 [==============================] - 4s 36ms/step - loss: 0.3408 - acc: 0.8506 - val_loss: 0.3079 - val_acc: 0.8688
Epoch 4/60
100/100 [==============================] - 4s 36ms/step - loss: 0.3041 - acc: 0.8703 - val_loss: 0.2569 - val_acc: 0.8925
Epoch 5/60
100/100 [==============================] - 4s 35ms/step - loss: 0.2872 - acc: 0.8841 - val_loss: 0.2718 - val_acc: 0.8869
Epoch 6/60
100/100 [==============================] - 4s 35ms/step - loss: 0.2691 - acc: 0.8866 - val_loss: 0.2986 - val_acc: 0.8706
Epoch 7/60
100/100 [==============================] - 4s 36ms/step - loss: 0.2612 - acc: 0.8922 - val_loss: 0.2776 - val_acc: 0.8924
Epoch 8/60
100/100 [==============================] - 4s 36ms/step - loss: 0.2451 - acc: 0.8981 - val_loss: 0.2524 - val_acc: 0.8956
```

```
Epoch 9/60
100/100 [==============================] - 4s 36ms/step - loss: 0.2274 - acc:
0.9034 - val_loss: 0.2358 - val_acc: 0.9050
Epoch 10/60
100/100 [==============================] - 4s 35ms/step - loss: 0.2341 - acc:
0.9034 - val_loss: 0.2243 - val_acc: 0.9125
Epoch 11/60
100/100 [==============================] - 4s 36ms/step - loss: 0.2237 - acc:
0.9106 - val_loss: 0.2534 - val_acc: 0.8969
Epoch 12/60
100/100 [==============================] - 4s 36ms/step - loss: 0.2270 - acc:
0.9074 - val_loss: 0.1937 - val_acc: 0.9205
Epoch 13/60
100/100 [==============================] - 4s 36ms/step - loss: 0.2102 - acc:
0.9172 - val_loss: 0.2128 - val_acc: 0.9187
Epoch 14/60
100/100 [==============================] - 4s 35ms/step - loss: 0.1979 - acc:
0.9197 - val_loss: 0.1784 - val_acc: 0.9294
Epoch 15/60
100/100 [==============================] - 4s 36ms/step - loss: 0.2006 - acc:
0.9156 - val_loss: 0.2025 - val_acc: 0.9137
Epoch 16/60
100/100 [==============================] - 4s 36ms/step - loss: 0.2043 - acc:
0.9137 - val_loss: 0.2116 - val_acc: 0.9219
Epoch 17/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1769 - acc:
0.9315 - val_loss: 0.1758 - val_acc: 0.9318
Epoch 18/60
100/100 [==============================] - 4s 36ms/step - loss: 0.2010 - acc:
0.9178 - val_loss: 0.2086 - val_acc: 0.9206
Epoch 19/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1859 - acc:
0.9222 - val_loss: 0.1825 - val_acc: 0.9213
Epoch 20/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1718 - acc:
0.9341 - val_loss: 0.2164 - val_acc: 0.9137
Epoch 21/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1841 - acc:
0.9256 - val_loss: 0.1557 - val_acc: 0.9344
Epoch 22/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1720 - acc:
0.9303 - val_loss: 0.2253 - val_acc: 0.9118
Epoch 23/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1721 - acc:
0.9294 - val_loss: 0.1941 - val_acc: 0.9281
Epoch 24/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1614 - acc:
0.9353 - val_loss: 0.2137 - val_acc: 0.9231
Epoch 25/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1638 - acc:
0.9322 - val_loss: 0.1502 - val_acc: 0.9387
Epoch 26/60
100/100 [==============================] - 4s 35ms/step - loss: 0.1571 - acc:
0.9375 - val_loss: 0.1801 - val_acc: 0.9250
Epoch 27/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1632 - acc:
0.9356 - val_loss: 0.1676 - val_acc: 0.9375
```

```
Epoch 28/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1445 - acc:
0.9434 - val_loss: 0.2064 - val_acc: 0.9193
Epoch 29/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1521 - acc:
0.9381 - val_loss: 0.1866 - val_acc: 0.9231
Epoch 30/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1399 - acc:
0.9444 - val_loss: 0.1420 - val_acc: 0.9431
Epoch 31/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1571 - acc:
0.9366 - val_loss: 0.1793 - val_acc: 0.9281
Epoch 32/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1511 - acc:
0.9406 - val_loss: 0.1719 - val_acc: 0.9350
Epoch 33/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1242 - acc:
0.9488 - val_loss: 0.1742 - val_acc: 0.9274
Epoch 34/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1365 - acc:
0.9497 - val_loss: 0.1828 - val_acc: 0.9325
Epoch 35/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1344 - acc:
0.9469 - val_loss: 0.1628 - val_acc: 0.9356
Epoch 36/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1285 - acc:
0.9478 - val_loss: 0.1748 - val_acc: 0.9325
Epoch 37/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1356 - acc:
0.9472 - val_loss: 0.1581 - val_acc: 0.9350
Epoch 38/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1117 - acc:
0.9528 - val_loss: 0.1642 - val_acc: 0.9349
Epoch 39/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1105 - acc:
0.9578 - val_loss: 0.1793 - val_acc: 0.9363
Epoch 40/60
100/100 [==============================] - 4s 35ms/step - loss: 0.1238 - acc:
0.9544 - val_loss: 0.1793 - val_acc: 0.9281
Epoch 41/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1280 - acc:
0.9513 - val_loss: 0.1617 - val_acc: 0.9381
Epoch 42/60
100/100 [==============================] - 4s 36ms/step - loss: 0.1181 - acc:
0.9525 - val_loss: 0.2275 - val_acc: 0.9300
Epoch 43/60
100/100 [==============================] - 4s 35ms/step - loss: 0.0980 - acc:
0.9596 - val_loss: 0.1859 - val_acc: 0.9399
Epoch 44/60
100/100 [==============================] - 4s 35ms/step - loss: 0.1013 - acc:
0.9613 - val_loss: 0.1791 - val_acc: 0.9294
Epoch 45/60
100/100 [==============================] - 3s 35ms/step - loss: 0.1065 - acc:
0.9591 - val_loss: 0.1807 - val_acc: 0.9325
Epoch 46/60
100/100 [==============================] - 4s 35ms/step - loss: 0.1108 - acc:
0.9569 - val_loss: 0.1785 - val_acc: 0.9363
```

```
Epoch 47/60
100/100 [==============================] - 3s 35ms/step - loss: 0.1023 - acc:
0.9594 - val_loss: 0.1844 - val_acc: 0.9287
Epoch 48/60
100/100 [==============================] - 4s 35ms/step - loss: 0.0954 - acc:
0.9628 - val_loss: 0.2139 - val_acc: 0.9262
Epoch 49/60
100/100 [==============================] - 4s 35ms/step - loss: 0.0905 - acc:
0.9675 - val_loss: 0.2421 - val_acc: 0.9237
Epoch 50/60
100/100 [==============================] - 3s 35ms/step - loss: 0.0831 - acc:
0.9666 - val_loss: 0.1721 - val_acc: 0.9387
Epoch 51/60
100/100 [==============================] - 4s 35ms/step - loss: 0.0838 - acc:
0.9669 - val_loss: 0.2032 - val_acc: 0.9300
Epoch 52/60
100/100 [==============================] - 4s 35ms/step - loss: 0.0900 - acc:
0.9656 - val_loss: 0.1519 - val_acc: 0.9487
Epoch 53/60
100/100 [==============================] - 4s 35ms/step - loss: 0.0778 - acc:
0.9730 - val_loss: 0.2650 - val_acc: 0.9305
Epoch 54/60
100/100 [==============================] - 4s 35ms/step - loss: 0.0682 - acc:
0.9731 - val_loss: 0.2745 - val_acc: 0.9194
Epoch 55/60
100/100 [==============================] - 4s 35ms/step - loss: 0.0872 - acc:
0.9700 - val_loss: 0.2180 - val_acc: 0.9269
Epoch 56/60
100/100 [==============================] - 4s 35ms/step - loss: 0.0802 - acc:
0.9688 - val_loss: 0.1889 - val_acc: 0.9294
Epoch 57/60
100/100 [==============================] - 4s 35ms/step - loss: 0.0803 - acc:
0.9691 - val_loss: 0.1530 - val_acc: 0.9425
Epoch 58/60
100/100 [==============================] - 3s 35ms/step - loss: 0.0711 - acc:
0.9731 - val_loss: 0.2337 - val_acc: 0.9293
Epoch 59/60
100/100 [==============================] - 4s 35ms/step - loss: 0.0649 - acc:
0.9778 - val_loss: 0.2385 - val_acc: 0.9294
Epoch 60/60
100/100 [==============================] - 4s 35ms/step - loss: 0.0617 - acc:
0.9803 - val_loss: 0.1915 - val_acc: 0.9375
```


Training and validation accuracy
Training acc
Validation acc
Accuracy
0.75
0.80
0.85
0.90
0.95
0
10
20
30
40
50
60
Num of Epochs




**INTERPRETATION OF RESULTS FOR STRATEGY 1:**

We see that increasing the number of layers and number of epochs have increased the accuracy score by about 5%.
But as we run for more epochs we also see a clear divergence between training set accuracy and validation set accuracy. In the next step we will try to reduce the overfit.

**Strategy 2: Try to reduce overfit by adding pooling, dropout and simplifying the model a little bit.**

- Increase Pooling Layers and decrease the number of filters in each layer to minimize overfit.
- Also add a few dropout layers

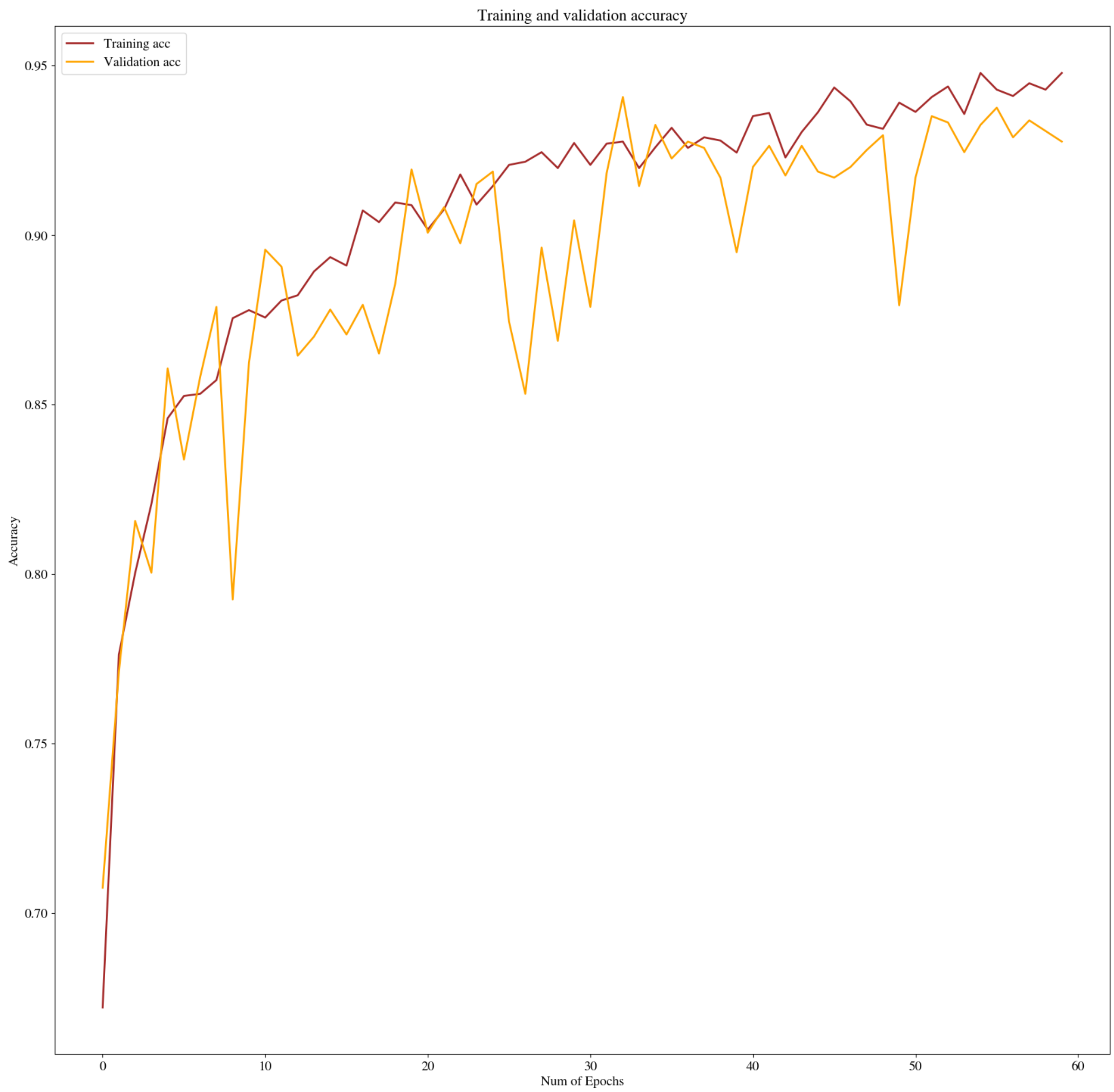
In [74]:
```
K.clear_session()

def reduce_overfit_model():

    model = Sequential(name='FiveLayerModel')
    model.add(Conv2D(32, (3, 3), padding='same', activation='relu',
                        input_shape=data_batch.shape[1:], name = 'conv1'))
    model.add(MaxPooling2D((2, 2), name='max_pool1'))
    model.add(Dropout(rate=0.2))
    model.add(Conv2D(64, (3, 3), padding='same', activation='relu', name = 'co
nv2'))
    model.add(MaxPooling2D((2, 2), name='max_pool2'))
    model.add(Dropout(rate=0.2))
    model.add(Conv2D(128, (3, 3), padding='same', activation='relu', name = 'c
onv3'))
    model.add(MaxPooling2D((2, 2), name='max_pool3'))
    model.add(Dropout(rate=0.2))
    model.add(Conv2D(128, (3, 3), padding='same', activation='relu', name = 'c
onv4'))
    model.add(MaxPooling2D((2, 2), name='max_pool4'))
    model.add(Conv2D(256, (3, 3), padding='same', activation='relu', name = 'c
onv5'))
    model.add(Conv2D(512,(3, 3), padding='same', activation='relu', name = 'co
nv6'))
    model.add(MaxPooling2D((2, 2), name='max_pool5'))
    model.add(Flatten())
    model.add(Dense(512, kernel_initializer='glorot_uniform', activation='rel
u', name='fc1'))
    model.add(Dense(1, kernel_initializer='glorot_uniform', activation='sigmoi
d', name='fc2'))

    model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
    #model.compile(optimizer="adam", loss="binary_crossentropy", metrics = ["a
ccuracy"])
    return model

cnn_reduce_overfit = reduce_overfit_model()
cnn_reduce_overfit .summary()

# Fit model
history_reduce_overfit = cnn_reduce_overfit .fit_generator(
    train_generator,
    steps_per_epoch=100,
    epochs=60,
    validation_data=validation_generator,
    validation_steps=50,
    verbose=1,
    callbacks=[TrainValTensorBoard("logs/{}".format(time()), write_graph=True
)])

plot_accuracies_loss(history_reduce_overfit)
```

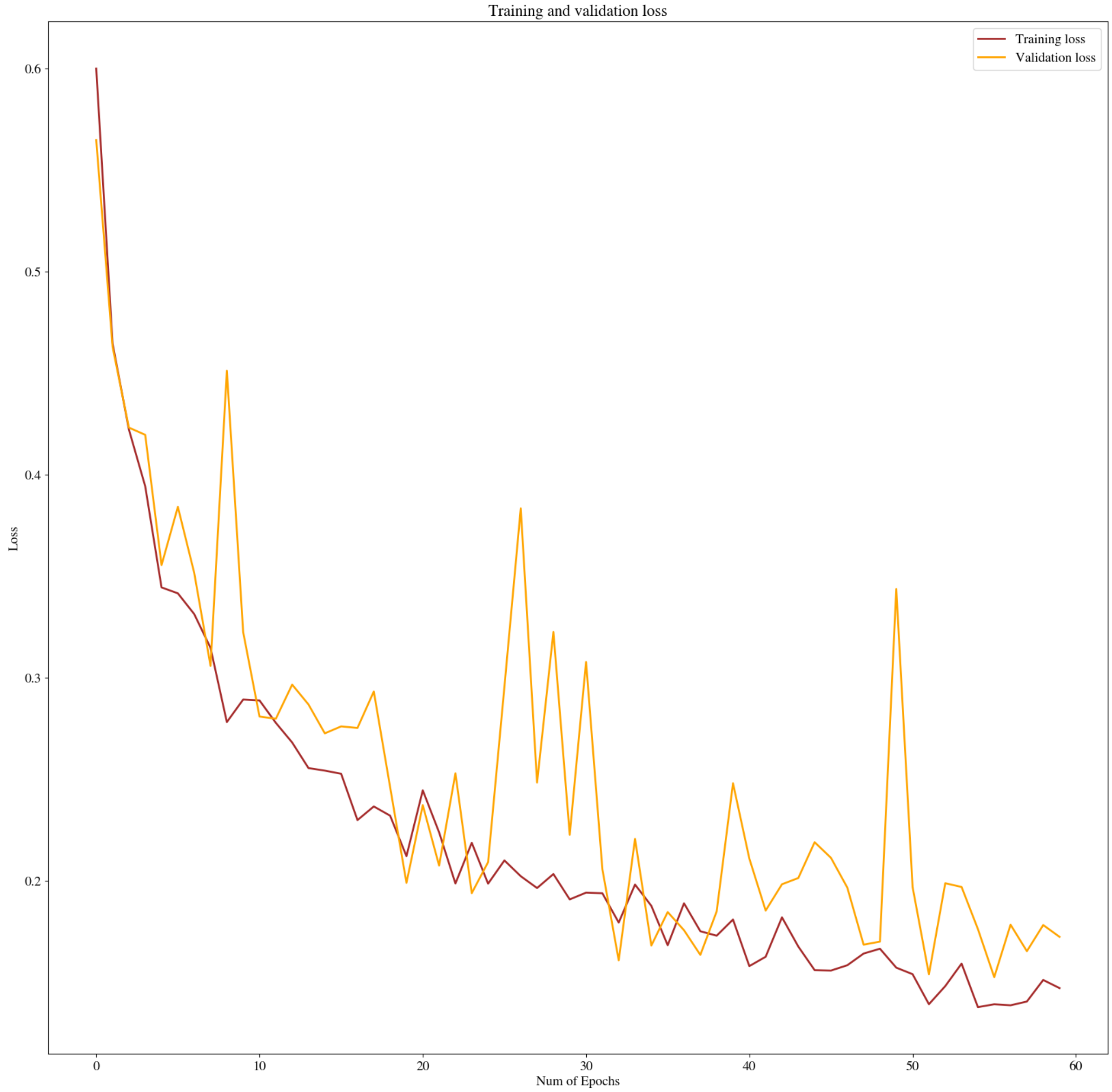
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1 (Conv2D)               (None, 64, 64, 32)        896       
_________________________________________________________________
max_pool1 (MaxPooling2D)     (None, 32, 32, 32)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 32)        0         
_________________________________________________________________
conv2 (Conv2D)               (None, 32, 32, 64)        18496     
_________________________________________________________________
max_pool2 (MaxPooling2D)     (None, 16, 16, 64)        0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 64)        0         
_________________________________________________________________
conv3 (Conv2D)               (None, 16, 16, 128)       73856     
_________________________________________________________________
max_pool3 (MaxPooling2D)     (None, 8, 8, 128)         0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 8, 8, 128)         0         
_________________________________________________________________
conv4 (Conv2D)               (None, 8, 8, 128)         147584    
_________________________________________________________________
max_pool4 (MaxPooling2D)     (None, 4, 4, 128)         0         
_________________________________________________________________
conv5 (Conv2D)               (None, 4, 4, 256)         295168    
_________________________________________________________________
conv6 (Conv2D)               (None, 4, 4, 512)         1180160   
_________________________________________________________________
max_pool5 (MaxPooling2D)     (None, 2, 2, 512)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0         
_________________________________________________________________
fc1 (Dense)                  (None, 512)               1049088   
_________________________________________________________________
fc2 (Dense)                  (None, 1)                 513       
=================================================================
Total params: 2,765,761
Trainable params: 2,765,761
Non-trainable params: 0
_________________________________________________________________
Epoch 1/60
100/100 [==============================] - 4s 40ms/step - loss: 0.5999 - acc: 0.6722 - val_loss: 0.5647 - val_acc: 0.7075
Epoch 2/60
100/100 [==============================] - 3s 33ms/step - loss: 0.4646 - acc: 0.7762 - val_loss: 0.4629 - val_acc: 0.7712
Epoch 3/60
100/100 [==============================] - 3s 33ms/step - loss: 0.4222 - acc: 0.8003 - val_loss: 0.4231 - val_acc: 0.8156
Epoch 4/60
100/100 [==============================] - 3s 35ms/step - loss: 0.3942 - acc: 0.8206 - val_loss: 0.4196 - val_acc: 0.8004
Epoch 5/60
100/100 [==============================] - 3s 34ms/step - loss: 0.3444 - acc: 0.8459 - val_loss: 0.3554 - val_acc: 0.8606
Epoch 6/60
```

```
100/100 [==============================] - 3s 33ms/step - loss: 0.3415 - acc:
0.8525 - val_loss: 0.3841 - val_acc: 0.8337
Epoch 7/60
100/100 [==============================] - 3s 33ms/step - loss: 0.3313 - acc:
0.8531 - val_loss: 0.3516 - val_acc: 0.8581
Epoch 8/60
100/100 [==============================] - 3s 34ms/step - loss: 0.3146 - acc:
0.8572 - val_loss: 0.3058 - val_acc: 0.8788
Epoch 9/60
100/100 [==============================] - 3s 33ms/step - loss: 0.2781 - acc:
0.8753 - val_loss: 0.4512 - val_acc: 0.7925
Epoch 10/60
100/100 [==============================] - 3s 33ms/step - loss: 0.2892 - acc:
0.8778 - val_loss: 0.3224 - val_acc: 0.8623
Epoch 11/60
100/100 [==============================] - 3s 34ms/step - loss: 0.2888 - acc:
0.8756 - val_loss: 0.2808 - val_acc: 0.8956
Epoch 12/60
100/100 [==============================] - 3s 33ms/step - loss: 0.2777 - acc:
0.8806 - val_loss: 0.2797 - val_acc: 0.8906
Epoch 13/60
100/100 [==============================] - 3s 33ms/step - loss: 0.2680 - acc:
0.8822 - val_loss: 0.2965 - val_acc: 0.8644
Epoch 14/60
100/100 [==============================] - 3s 32ms/step - loss: 0.2554 - acc:
0.8891 - val_loss: 0.2868 - val_acc: 0.8700
Epoch 15/60
100/100 [==============================] - 3s 33ms/step - loss: 0.2542 - acc:
0.8934 - val_loss: 0.2725 - val_acc: 0.8780
Epoch 16/60
100/100 [==============================] - 3s 33ms/step - loss: 0.2526 - acc:
0.8909 - val_loss: 0.2760 - val_acc: 0.8706
Epoch 17/60
100/100 [==============================] - 3s 33ms/step - loss: 0.2298 - acc:
0.9072 - val_loss: 0.2752 - val_acc: 0.8794
Epoch 18/60
100/100 [==============================] - 3s 33ms/step - loss: 0.2365 - acc:
0.9038 - val_loss: 0.2932 - val_acc: 0.8650
Epoch 19/60
100/100 [==============================] - 3s 33ms/step - loss: 0.2317 - acc:
0.9097 - val_loss: 0.2461 - val_acc: 0.8856
Epoch 20/60
100/100 [==============================] - 3s 33ms/step - loss: 0.2121 - acc:
0.9087 - val_loss: 0.1989 - val_acc: 0.9193
Epoch 21/60
100/100 [==============================] - 3s 33ms/step - loss: 0.2445 - acc:
0.9016 - val_loss: 0.2372 - val_acc: 0.9006
Epoch 22/60
100/100 [==============================] - 3s 33ms/step - loss: 0.2238 - acc:
0.9075 - val_loss: 0.2074 - val_acc: 0.9081
Epoch 23/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1986 - acc:
0.9178 - val_loss: 0.2529 - val_acc: 0.8975
Epoch 24/60
100/100 [==============================] - 3s 33ms/step - loss: 0.2188 - acc:
0.9088 - val_loss: 0.1938 - val_acc: 0.9150
Epoch 25/60
```

```
100/100 [==============================] - 3s 33ms/step - loss: 0.1985 - acc:
0.9144 - val_loss: 0.2091 - val_acc: 0.9186
Epoch 26/60
100/100 [==============================] - 3s 33ms/step - loss: 0.2100 - acc:
0.9206 - val_loss: 0.2955 - val_acc: 0.8744
Epoch 27/60
100/100 [==============================] - 3s 33ms/step - loss: 0.2022 - acc:
0.9216 - val_loss: 0.3834 - val_acc: 0.8531
Epoch 28/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1963 - acc:
0.9244 - val_loss: 0.2483 - val_acc: 0.8962
Epoch 29/60
100/100 [==============================] - 3s 33ms/step - loss: 0.2032 - acc:
0.9197 - val_loss: 0.3225 - val_acc: 0.8688
Epoch 30/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1907 - acc:
0.9271 - val_loss: 0.2226 - val_acc: 0.9043
Epoch 31/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1941 - acc:
0.9206 - val_loss: 0.3077 - val_acc: 0.8788
Epoch 32/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1938 - acc:
0.9269 - val_loss: 0.2056 - val_acc: 0.9181
Epoch 33/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1794 - acc:
0.9275 - val_loss: 0.1608 - val_acc: 0.9406
Epoch 34/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1981 - acc:
0.9197 - val_loss: 0.2206 - val_acc: 0.9144
Epoch 35/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1880 - acc:
0.9258 - val_loss: 0.1680 - val_acc: 0.9324
Epoch 36/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1682 - acc:
0.9316 - val_loss: 0.1845 - val_acc: 0.9225
Epoch 37/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1888 - acc:
0.9256 - val_loss: 0.1756 - val_acc: 0.9275
Epoch 38/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1751 - acc:
0.9287 - val_loss: 0.1634 - val_acc: 0.9256
Epoch 39/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1729 - acc:
0.9278 - val_loss: 0.1848 - val_acc: 0.9169
Epoch 40/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1810 - acc:
0.9242 - val_loss: 0.2480 - val_acc: 0.8949
Epoch 41/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1579 - acc:
0.9350 - val_loss: 0.2109 - val_acc: 0.9200
Epoch 42/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1625 - acc:
0.9359 - val_loss: 0.1853 - val_acc: 0.9263
Epoch 43/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1819 - acc:
0.9228 - val_loss: 0.1982 - val_acc: 0.9175
Epoch 44/60
```

```
100/100 [==============================] - 3s 33ms/step - loss: 0.1675 - acc:
0.9303 - val_loss: 0.2013 - val_acc: 0.9263
Epoch 45/60
100/100 [==============================] - 3s 32ms/step - loss: 0.1557 - acc:
0.9362 - val_loss: 0.2189 - val_acc: 0.9186
Epoch 46/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1557 - acc:
0.9434 - val_loss: 0.2112 - val_acc: 0.9169
Epoch 47/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1583 - acc:
0.9394 - val_loss: 0.1966 - val_acc: 0.9200
Epoch 48/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1641 - acc:
0.9325 - val_loss: 0.1685 - val_acc: 0.9250
Epoch 49/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1665 - acc:
0.9313 - val_loss: 0.1700 - val_acc: 0.9294
Epoch 50/60
100/100 [==============================] - 3s 34ms/step - loss: 0.1572 - acc:
0.9389 - val_loss: 0.3437 - val_acc: 0.8792
Epoch 51/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1539 - acc:
0.9363 - val_loss: 0.1968 - val_acc: 0.9169
Epoch 52/60
100/100 [==============================] - 3s 34ms/step - loss: 0.1391 - acc:
0.9406 - val_loss: 0.1538 - val_acc: 0.9350
Epoch 53/60
100/100 [==============================] - 3s 34ms/step - loss: 0.1480 - acc:
0.9437 - val_loss: 0.1987 - val_acc: 0.9331
Epoch 54/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1591 - acc:
0.9356 - val_loss: 0.1969 - val_acc: 0.9244
Epoch 55/60
100/100 [==============================] - 3s 34ms/step - loss: 0.1376 - acc:
0.9478 - val_loss: 0.1762 - val_acc: 0.9324
Epoch 56/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1391 - acc:
0.9428 - val_loss: 0.1525 - val_acc: 0.9375
Epoch 57/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1386 - acc:
0.9409 - val_loss: 0.1783 - val_acc: 0.9287
Epoch 58/60
100/100 [==============================] - 3s 33ms/step - loss: 0.1405 - acc:
0.9447 - val_loss: 0.1652 - val_acc: 0.9337
Epoch 59/60
100/100 [==============================] - 3s 34ms/step - loss: 0.1511 - acc:
0.9428 - val_loss: 0.1781 - val_acc: 0.9306
Epoch 60/60
100/100 [==============================] - 3s 34ms/step - loss: 0.1471 - acc:
0.9477 - val_loss: 0.1723 - val_acc: 0.9275
```


Training and validation accuracy
Training acc
Validation acc
Accuracy
Num of Epochs
0.95
0.90
0.85
0.80
0.75
0.70
0
10
20
30
40
50
60




**INTERPRETATION OF RESULTS FOR STRATEGY 2:**

The above results show that the introduction of dropouts and pooling layers have decreased the overfit considerably. This is indicated by the fact that the training accuracy and the validation accuracy is staying fairly close together upto about 55 epochs.

I also conclude that the validation accuracy begins to decline after about 55 epochs due to overfitting so I will train our models upto 55 epochs going forward.

In the next step, I will try to use some different optimizers and mask sizes and filter numbers to further enhance accuracy score.

**Strategy 3: Try some different parameters:**

- Try changing the activation function from relu to sigmoid which will capture both negetive and positive values.
- Also try different optimization algorithms like adam optimizer or stochastic gradiant descent.

In [91]:
```
K.clear_session()

kernel_size = 5
#OPTIMIZER = optimizers.RMSprop(lr=1e-4)
OPTIMIZER =  Adam(lr=0.001)
#OPTIMIZER = optimizers.SGD(lr = 0.05, decay=1e-6, momentum=0.9, nesterov=Tru
e)

def reduce_overfit_model_tune(OPTIMIZER, kernel_size):

    model = Sequential(name='FiveLayerModel')
    model.add(Conv2D(32, (kernel_size, kernel_size), padding='same', activatio
n='relu',
                            input_shape=data_batch.shape[1:], name = 'conv1'))
    model.add(MaxPooling2D((2, 2), name='max_pool1'))
    model.add(Dropout(rate=0.2))
    model.add(Conv2D(64, (kernel_size, kernel_size), padding='same', activatio
n='relu', name = 'conv2'))
    model.add(MaxPooling2D((2, 2), name='max_pool2'))
    model.add(Dropout(rate=0.2))
    model.add(Conv2D(128, (kernel_size, kernel_size), padding='same', activati
on='relu', name = 'conv3'))
    model.add(MaxPooling2D((2, 2), name='max_pool3'))
    model.add(Dropout(rate=0.2))
    model.add(Conv2D(128, (kernel_size,kernel_size), padding='same', activatio
n='relu', name = 'conv4'))
    model.add(MaxPooling2D((2, 2), name='max_pool4'))
    model.add(Conv2D(256, (kernel_size, kernel_size), padding='same', activati
on='relu', name = 'conv5'))
    model.add(Conv2D(512,(kernel_size, kernel_size), padding='same', activatio
n='relu', name = 'conv6'))
    model.add(MaxPooling2D((2, 2), name='max_pool5'))
    model.add(Flatten())
    model.add(Dense(512, kernel_initializer='glorot_uniform', activation='rel
u', name='fc1'))
    model.add(Dense(256, kernel_initializer='glorot_uniform', activation='rel
u', name='fc2'))
    model.add(Dense(1, kernel_initializer='glorot_uniform', activation='sigmoi
d', name='fc3'))

    model.compile(loss='binary_crossentropy',
                  optimizer=OPTIMIZER,
                  metrics=['acc'])
    #model.compile(optimizer="adam", loss="binary_crossentropy", metrics = ["a
ccuracy"])
    return model

cnn_reduce_overfit_tune = reduce_overfit_model_tune(OPTIMIZER, kernel_size)
cnn_reduce_overfit_tune.summary()

# Fit model
history_reduce_overfit_tune = cnn_reduce_overfit_tune.fit_generator(
    train_generator,
    steps_per_epoch=100,
    epochs=55,
    validation_data=validation_generator,
```

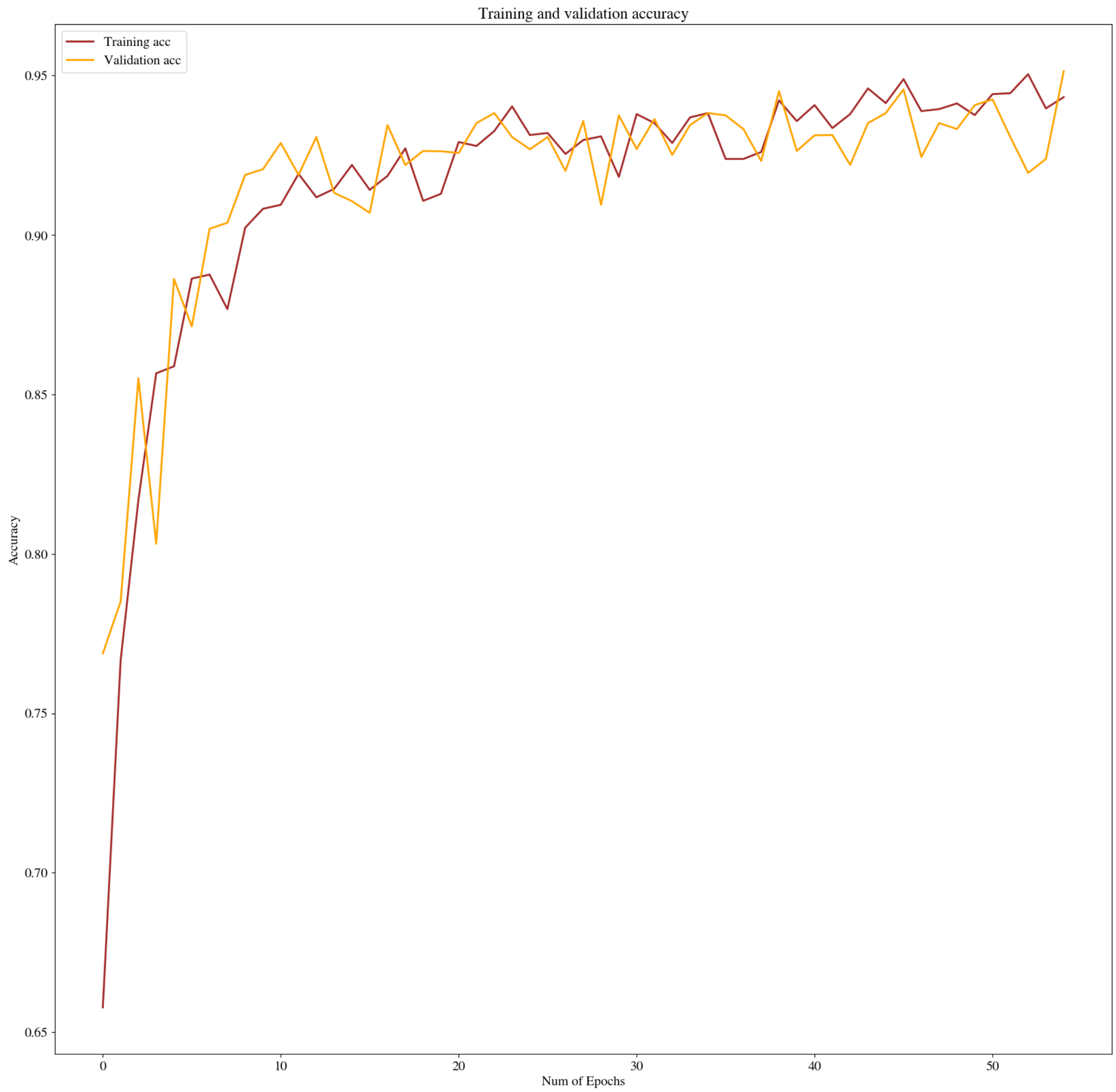
```
    validation_steps=50,
    verbose=1,
    callbacks=[TrainValTensorBoard("logs/{}".format(time()), write_graph=True
)])

plot_accuracies_loss(history_reduce_overfit_tune)
```

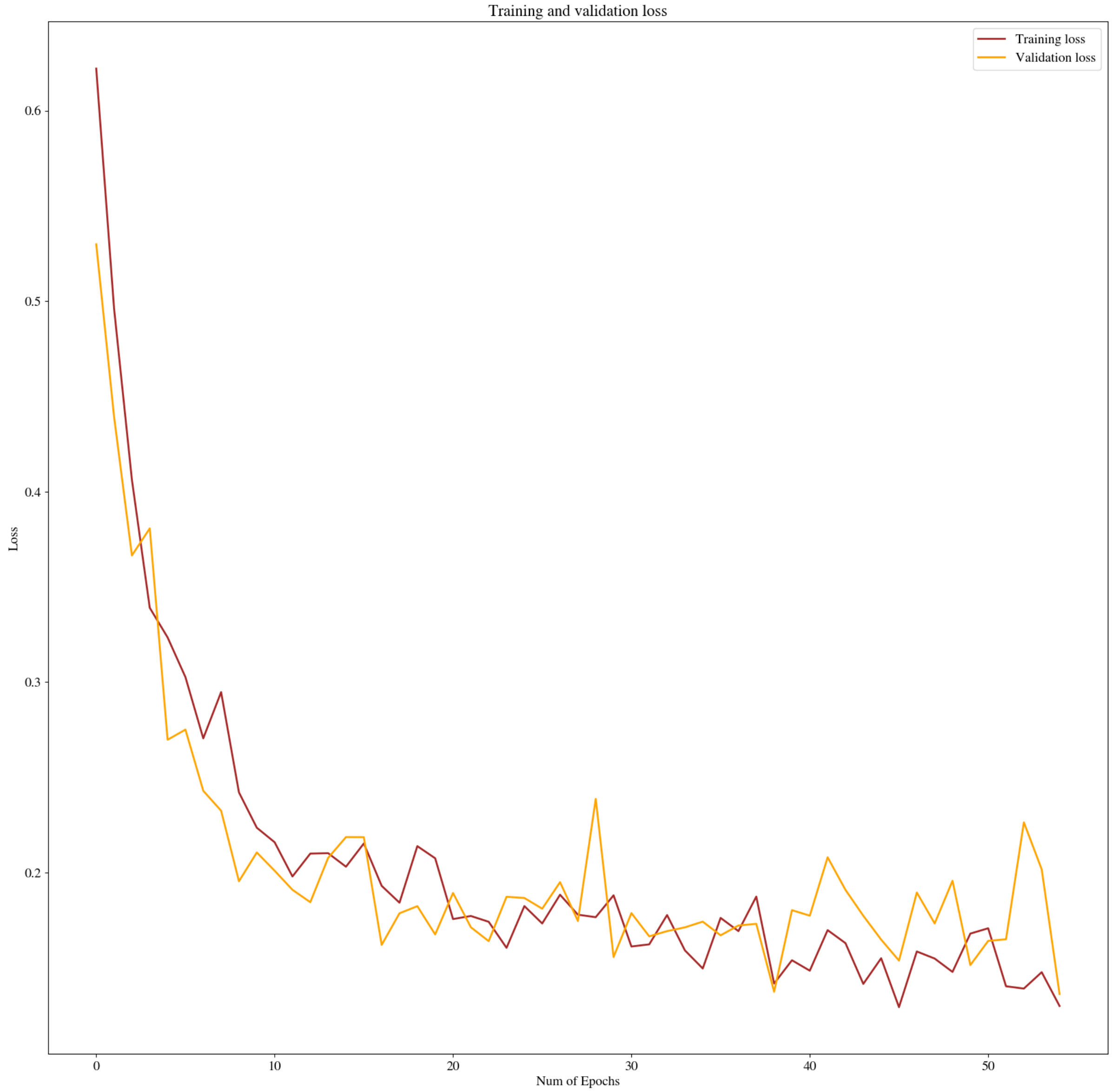
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1 (Conv2D)               (None, 64, 64, 32)        2432      
_________________________________________________________________
max_pool1 (MaxPooling2D)     (None, 32, 32, 32)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 32, 32)        0         
_________________________________________________________________
conv2 (Conv2D)               (None, 32, 32, 64)        51264     
_________________________________________________________________
max_pool2 (MaxPooling2D)     (None, 16, 16, 64)        0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 64)        0         
_________________________________________________________________
conv3 (Conv2D)               (None, 16, 16, 128)       204928    
_________________________________________________________________
max_pool3 (MaxPooling2D)     (None, 8, 8, 128)         0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 8, 8, 128)         0         
_________________________________________________________________
conv4 (Conv2D)               (None, 8, 8, 128)         409728    
_________________________________________________________________
max_pool4 (MaxPooling2D)     (None, 4, 4, 128)         0         
_________________________________________________________________
conv5 (Conv2D)               (None, 4, 4, 256)         819456    
_________________________________________________________________
conv6 (Conv2D)               (None, 4, 4, 512)         3277312   
_________________________________________________________________
max_pool5 (MaxPooling2D)     (None, 2, 2, 512)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0         
_________________________________________________________________
fc1 (Dense)                  (None, 512)               1049088   
_________________________________________________________________
fc2 (Dense)                  (None, 256)               131328    
_________________________________________________________________
fc3 (Dense)                  (None, 1)                 257       
=================================================================
Total params: 5,945,793
Trainable params: 5,945,793
Non-trainable params: 0
_________________________________________________________________
Epoch 1/55
100/100 [==============================] - 7s 67ms/step - loss: 0.6221 - acc:
0.6578 - val_loss: 0.5298 - val_acc: 0.7688
Epoch 2/55
100/100 [==============================] - 6s 60ms/step - loss: 0.4961 - acc:
0.7666 - val_loss: 0.4392 - val_acc: 0.7850
Epoch 3/55
100/100 [==============================] - 6s 60ms/step - loss: 0.4063 - acc:
0.8168 - val_loss: 0.3664 - val_acc: 0.8550
Epoch 4/55
100/100 [==============================] - 6s 60ms/step - loss: 0.3390 - acc:
0.8566 - val_loss: 0.3807 - val_acc: 0.8031
Epoch 5/55
100/100 [==============================] - 6s 60ms/step - loss: 0.3234 - acc:
```

```
0.8588 - val_loss: 0.2697 - val_acc: 0.8861
Epoch 6/55
100/100 [==============================] - 6s 60ms/step - loss: 0.3026 - acc:
0.8863 - val_loss: 0.2750 - val_acc: 0.8712
Epoch 7/55
100/100 [==============================] - 6s 60ms/step - loss: 0.2704 - acc:
0.8875 - val_loss: 0.2429 - val_acc: 0.9019
Epoch 8/55
100/100 [==============================] - 6s 60ms/step - loss: 0.2945 - acc:
0.8768 - val_loss: 0.2324 - val_acc: 0.9038
Epoch 9/55
100/100 [==============================] - 6s 59ms/step - loss: 0.2421 - acc:
0.9022 - val_loss: 0.1954 - val_acc: 0.9187
Epoch 10/55
100/100 [==============================] - 6s 60ms/step - loss: 0.2235 - acc:
0.9081 - val_loss: 0.2105 - val_acc: 0.9205
Epoch 11/55
100/100 [==============================] - 6s 60ms/step - loss: 0.2159 - acc:
0.9094 - val_loss: 0.2009 - val_acc: 0.9287
Epoch 12/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1979 - acc:
0.9191 - val_loss: 0.1909 - val_acc: 0.9187
Epoch 13/55
100/100 [==============================] - 6s 60ms/step - loss: 0.2098 - acc:
0.9118 - val_loss: 0.1844 - val_acc: 0.9306
Epoch 14/55
100/100 [==============================] - 6s 59ms/step - loss: 0.2101 - acc:
0.9144 - val_loss: 0.2077 - val_acc: 0.9131
Epoch 15/55
100/100 [==============================] - 6s 59ms/step - loss: 0.2030 - acc:
0.9219 - val_loss: 0.2186 - val_acc: 0.9105
Epoch 16/55
100/100 [==============================] - 6s 59ms/step - loss: 0.2152 - acc:
0.9141 - val_loss: 0.2185 - val_acc: 0.9069
Epoch 17/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1930 - acc:
0.9184 - val_loss: 0.1620 - val_acc: 0.9344
Epoch 18/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1841 - acc:
0.9270 - val_loss: 0.1786 - val_acc: 0.9219
Epoch 19/55
100/100 [==============================] - 6s 59ms/step - loss: 0.2138 - acc:
0.9106 - val_loss: 0.1823 - val_acc: 0.9263
Epoch 20/55
100/100 [==============================] - 6s 60ms/step - loss: 0.2074 - acc:
0.9128 - val_loss: 0.1675 - val_acc: 0.9262
Epoch 21/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1755 - acc:
0.9291 - val_loss: 0.1892 - val_acc: 0.9256
Epoch 22/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1771 - acc:
0.9278 - val_loss: 0.1712 - val_acc: 0.9350
Epoch 23/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1741 - acc:
0.9325 - val_loss: 0.1639 - val_acc: 0.9381
Epoch 24/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1604 - acc:
```

```
0.9403 - val_loss: 0.1872 - val_acc: 0.9306
Epoch 25/55
100/100 [==============================] - 6s 59ms/step - loss: 0.1824 - acc:
0.9313 - val_loss: 0.1866 - val_acc: 0.9268
Epoch 26/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1732 - acc:
0.9319 - val_loss: 0.1810 - val_acc: 0.9306
Epoch 27/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1883 - acc:
0.9253 - val_loss: 0.1949 - val_acc: 0.9200
Epoch 28/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1778 - acc:
0.9297 - val_loss: 0.1745 - val_acc: 0.9356
Epoch 29/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1763 - acc:
0.9309 - val_loss: 0.2386 - val_acc: 0.9094
Epoch 30/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1880 - acc:
0.9181 - val_loss: 0.1556 - val_acc: 0.9374
Epoch 31/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1611 - acc:
0.9378 - val_loss: 0.1787 - val_acc: 0.9269
Epoch 32/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1622 - acc:
0.9350 - val_loss: 0.1665 - val_acc: 0.9363
Epoch 33/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1776 - acc:
0.9287 - val_loss: 0.1692 - val_acc: 0.9250
Epoch 34/55
100/100 [==============================] - 6s 59ms/step - loss: 0.1589 - acc:
0.9368 - val_loss: 0.1712 - val_acc: 0.9344
Epoch 35/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1496 - acc:
0.9381 - val_loss: 0.1742 - val_acc: 0.9381
Epoch 36/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1761 - acc:
0.9237 - val_loss: 0.1670 - val_acc: 0.9374
Epoch 37/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1692 - acc:
0.9237 - val_loss: 0.1722 - val_acc: 0.9331
Epoch 38/55
100/100 [==============================] - 6s 59ms/step - loss: 0.1873 - acc:
0.9259 - val_loss: 0.1730 - val_acc: 0.9231
Epoch 39/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1416 - acc:
0.9421 - val_loss: 0.1374 - val_acc: 0.9450
Epoch 40/55
100/100 [==============================] - 6s 59ms/step - loss: 0.1539 - acc:
0.9356 - val_loss: 0.1802 - val_acc: 0.9263
Epoch 41/55
100/100 [==============================] - 6s 59ms/step - loss: 0.1484 - acc:
0.9406 - val_loss: 0.1773 - val_acc: 0.9312
Epoch 42/55
100/100 [==============================] - 6s 59ms/step - loss: 0.1697 - acc:
0.9334 - val_loss: 0.2080 - val_acc: 0.9313
Epoch 43/55
100/100 [==============================] - 6s 59ms/step - loss: 0.1629 - acc:
```

```
0.9378 - val_loss: 0.1909 - val_acc: 0.9219
Epoch 44/55
100/100 [==============================] - 6s 59ms/step - loss: 0.1416 - acc:
0.9458 - val_loss: 0.1773 - val_acc: 0.9350
Epoch 45/55
100/100 [==============================] - 6s 59ms/step - loss: 0.1550 - acc:
0.9412 - val_loss: 0.1647 - val_acc: 0.9381
Epoch 46/55
100/100 [==============================] - 6s 59ms/step - loss: 0.1293 - acc:
0.9488 - val_loss: 0.1537 - val_acc: 0.9456
Epoch 47/55
100/100 [==============================] - 6s 59ms/step - loss: 0.1585 - acc:
0.9387 - val_loss: 0.1895 - val_acc: 0.9244
Epoch 48/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1548 - acc:
0.9394 - val_loss: 0.1732 - val_acc: 0.9350
Epoch 49/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1476 - acc:
0.9412 - val_loss: 0.1956 - val_acc: 0.9331
Epoch 50/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1679 - acc:
0.9375 - val_loss: 0.1514 - val_acc: 0.9406
Epoch 51/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1707 - acc:
0.9441 - val_loss: 0.1641 - val_acc: 0.9424
Epoch 52/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1402 - acc:
0.9444 - val_loss: 0.1649 - val_acc: 0.9306
Epoch 53/55
100/100 [==============================] - 6s 60ms/step - loss: 0.1390 - acc:
0.9503 - val_loss: 0.2263 - val_acc: 0.9194
Epoch 54/55
100/100 [==============================] - 6s 59ms/step - loss: 0.1475 - acc:
0.9397 - val_loss: 0.2016 - val_acc: 0.9237
Epoch 55/55
100/100 [==============================] - 6s 59ms/step - loss: 0.1299 - acc:
0.9431 - val_loss: 0.1362 - val_acc: 0.9513
```


Training and validation accuracy
Training acc
Validation acc
0.95
0.90
0.85
0.80
0.75
0.70
0.65
Accuracy
0
10
20
30
40
50
Num of Epochs




**INTERPRETATION OF RESULTS FOR STRATEGY 3:**

After changing the optimizer to adam optimizer and changing the kernel sizes for each convolution layer to 5X5 I am able to reach a validation accuracy of about 94%.

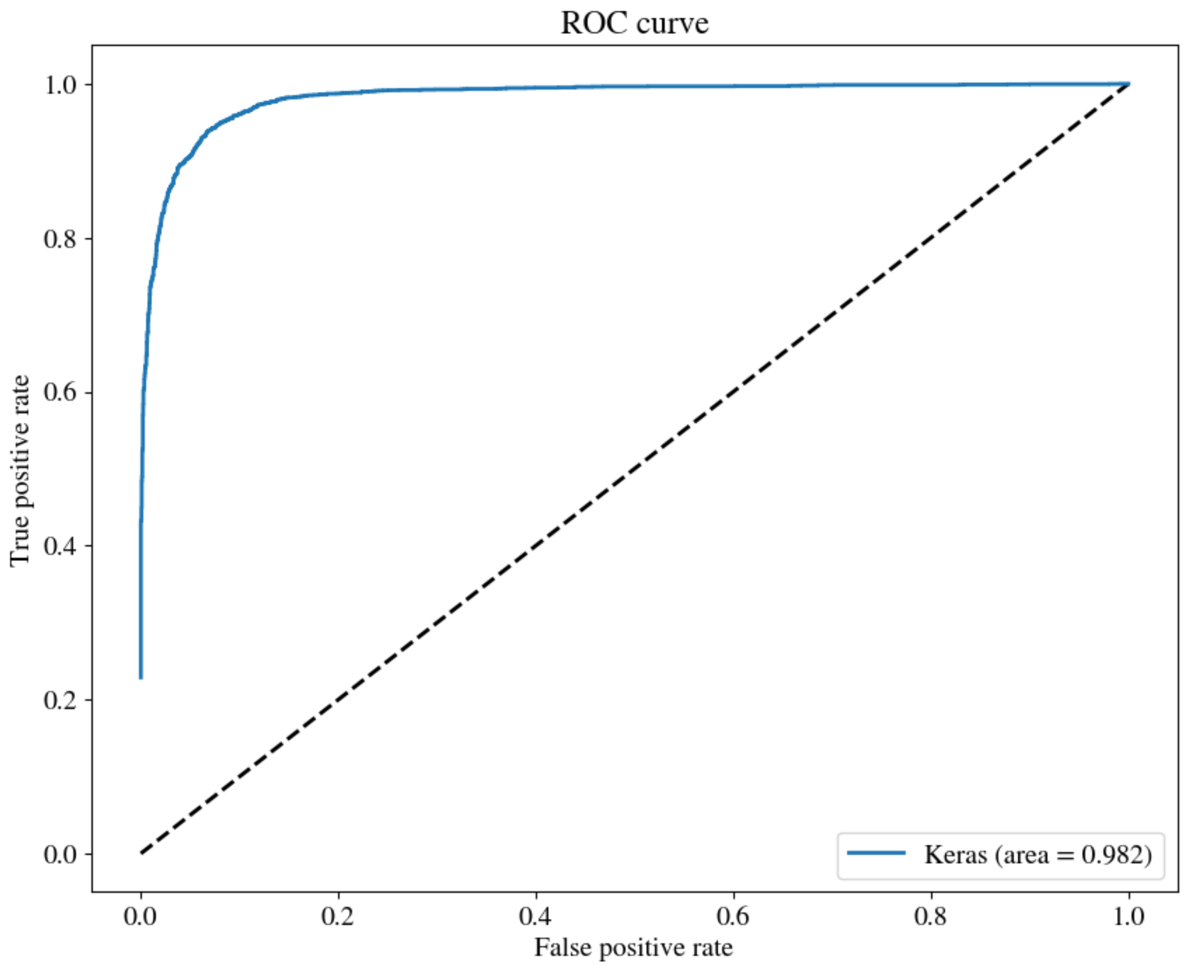
I also added another fully connected layer at the end to be able to capture some more detail before the last layer.

# Problem 5 (10 points)

Assess your best model on the test data. Plot the corresponding ROC curve from the results (since we've provided the truth). This was not directly covered in section, but will require a prediction using images in the same format as the training. We suggest referring to the Keras API else use a Google to search to find how to make predictions.

In [92]:
```
### PICK THE BEST MODEL #####
best_model = cnn_reduce_overfit_tune

# GET ACCURACY SCORE ON THE TEST SET
test_loss, test_acc = best_model.evaluate_generator(test_generator, steps=100)
print('\nTEST accuracy:', test_acc)
print('TEST loss:', test_loss)


predictions = []
labels = []
indexes = []
i = 0
for data_batch, labels_batch in test_generator:
    labels.extend(labels_batch)
    predictions.extend(best_model.predict(data_batch).T[0])
    i = i + 1
    if i==len(test_generator):
        break

for i in range (len(test_generator)):
     indexes.extend(next(test_generator.index_generator))

#print(len(indexes))
#print(len(labels))
#print(len(predictions))

#https://www.dlology.com/blog/simple-guide-on-how-to-generate-roc-plot-for-ker
as-classifier/
fpr, tpr, thresholds = roc_curve(labels, predictions)
auc = auc(fpr, tpr)

plt.figure(figsize=(10,8))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr, label='Keras (area = {:.3f})'.format(auc))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```

```
TEST accuracy: 0.9391277064377886
TEST loss: 0.16808429474942094
```

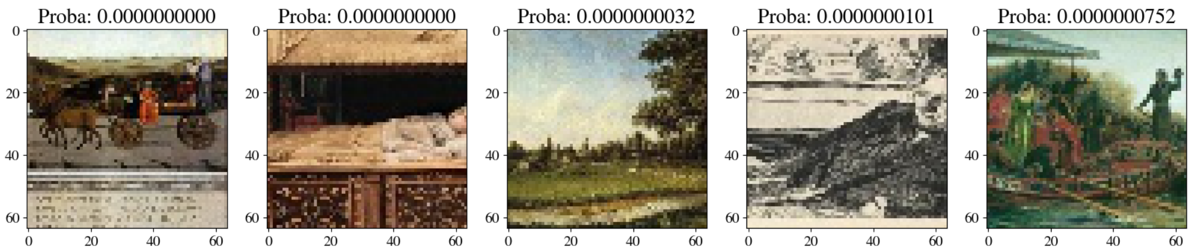



## Problem 6 (5 points)

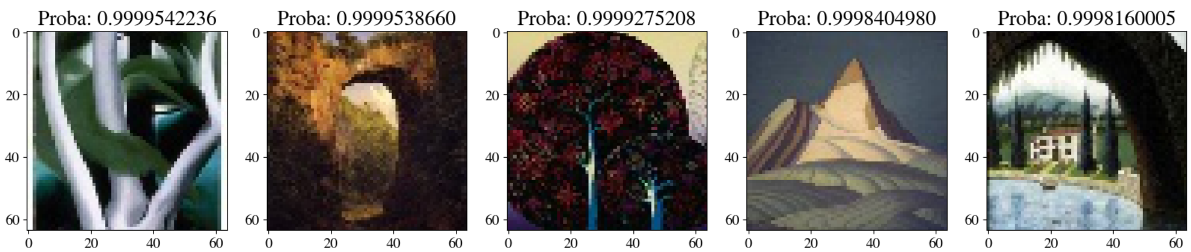
Display the 5 images [worst] misclassified images for each class. Worst is in brackets since certain architectures may only make a binary decision rather than a score. In that case, plot 5 of each.

In [95]:
```
## Misclassified Portraits
missed_portraits = []
missed_landscapes = []

def find_missclassified(tp):
    lst = []
    for i in range(len(indexes)):
        portrait_miss= (labels[i] == 1.0) & (predictions[i] < 0.5)
        landscape_miss = (labels[i] == 0.0) & (predictions[i] >= 0.5)
        if ((tp == "portrait") & portrait_miss) | ((tp == 'landscape') & lands
cape_miss)  :
            filename= 'images64/test/'+ test_generator.filenames[indexes[i]]
            #print(filename)
            #print(predictions[i])
            lst.append((filename,predictions[i]))
    return lst


missed_portraits = sorted(find_missclassified('portrait'), key = lambda tup: t
up[1])
missed_landscapes = sorted(find_missclassified('landscape'), key = lambda tup:
 tup[1], reverse = True)


def plot_top_five (missed_list):
    plt.rcParams['figure.figsize'] = (20.0, 20.0)
    f, ax = plt.subplots(nrows=1, ncols=5)
    for i in range(5):
        filename, prediction = missed_list[i]
        test = image.load_img(filename)
        ax[i].set_title("Proba: %0.10f" % prediction , loc='center', fontsize
= 20)
        ax[i].imshow(test)
    plt.show()
    return

print("\nWORST 5 MISSCLASSIFIED PORTRAITS:")
plot_top_five(missed_portraits)

print("\nWORST 5 MISSCLASSIFIED LANDSCAPES:")
plot_top_five(missed_landscapes)
```

```
WORST 5 MISSCLASSIFIED PORTRAITS:
```




```
WORST 5 MISSCLASSIFIED LANDSCAPES:
```




# Problem 7 (2 points)

How many hours did this homework take you? The answer to this question will not affect your grade.

In [ ]:
```
20
```

# Last step (3 points)

Save this notebook as LastnameFirstnameHW5.ipynb such as PriceDavid.ipynb. Create a pdf of this notebook named similarly. Submit both the python notebook and the pdf version to the Canvas dropbox. We require both versions.