Image Augmentation & Transfer Learning in Convolutional Neural Network

The goal of this project is to detect Cat and Dog images using Convolutional Neural Network with image augmentation to avoid overfitting.

In [1]:
```
import os
import zipfile
import random
import shutil
import tensorflow as tf
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from shutil import copyfile
from os import getcwd
```

In [2]:
```
# This code block unzips the full Cats-v-Dogs dataset to /tmp
# which will create a tmp/PetImages directory containing subdirectories
# called 'Cat' and 'Dog' (that's how the original researchers structured it)
path_cats_and_dogs = f"{getcwd()}/../tmp2/cats-and-dogs.zip"
shutil.rmtree('/tmp')

local_zip = path_cats_and_dogs
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp')
zip_ref.close()
```

In [3]:
```
print(len(os.listdir('/tmp/PetImages/Cat/')))
print(len(os.listdir('/tmp/PetImages/Dog/')))

# Expected Output:
# 1500
# 1500
```

```
1500
1500
```

In [4]:
```
# Use os.mkdir to create your directories
# You will need a directory for cats-v-dogs, and subdirectories for training
# and testing. These in turn will need subdirectories for 'cats' and 'dogs'
try:
    # create target directory cats-v-dogs in parent directory tmp
    path = os.path.join('/tmp','cats-v-dogs')
    os.mkdir(path)

    #training and testing subdirectories
    path_train = os.path.join('/tmp/cats-v-dogs','training')
    path_test = os.path.join('/tmp/cats-v-dogs','testing')
    os.mkdir(path_train)
    os.mkdir(path_test)
    #cats and dogs subdirectories in training and testing
    subpaths_train_dogs = os.path.join('/tmp/cats-v-dogs/training','dogs')
    subpaths_train_cats = os.path.join('/tmp/cats-v-dogs/training','cats')
    subpaths_test_dogs = os.path.join('/tmp/cats-v-dogs/testing','dogs')
    subpaths_test_cats = os.path.join('/tmp/cats-v-dogs/testing','cats')
    os.mkdir(subpaths_train_dogs)
    os.mkdir(subpaths_train_cats)
    os.mkdir(subpaths_test_dogs)
    os.mkdir(subpaths_test_cats)
except OSError:
    pass
```

In [5]:
```
# Write a python function called split_data which takes
# a SOURCE directory containing the files
# a TRAINING directory that a portion of the files will be copied to
# a TESTING directory that a portion of the files will be copie to
# a SPLIT SIZE to determine the portion
# The files should also be randomized, so that the training set is a random
# X% of the files, and the test set is the remaining files
# SO, for example, if SOURCE is PetImages/Cat, and SPLIT SIZE is .9
# Then 90% of the images in PetImages/Cat will be copied to the TRAINING dir
# and 10% of the images will be copied to the TESTING dir
# Also -- All images should be checked, and if they have a zero file length,
# they will not be copied over
#
# os.listdir(DIRECTORY) gives you a listing of the contents of that directory
# os.path.getsize(PATH) gives you the size of the file
# copyfile(source, destination) copies a file from source to destination
# random.sample(list, len(list)) shuffles a list
def split_data(SOURCE, TRAINING, TESTING, SPLIT_SIZE):
# YOUR CODE STARTS HERE
    random.sample(os.listdir(SOURCE), len(os.listdir(SOURCE)))
    os.path.getsize(SOURCE)
    train_size = len(os.listdir(SOURCE)) * SPLIT_SIZE
    for i in os.listdir(SOURCE):
         if os.path.getsize(os.path.join(SOURCE,i)) !=0:
            if len(os.listdir(TRAINING)) < train_size:
                copyfile(SOURCE + '/' + i, TRAINING + '/'+i)
            else:
                copyfile(SOURCE + '/'+i, TESTING +'/'+i)


CAT_SOURCE_DIR = "/tmp/PetImages/Cat/"
TRAINING_CATS_DIR = "/tmp/cats-v-dogs/training/cats/"
TESTING_CATS_DIR = "/tmp/cats-v-dogs/testing/cats/"
DOG_SOURCE_DIR = "/tmp/PetImages/Dog/"
TRAINING_DOGS_DIR = "/tmp/cats-v-dogs/training/dogs/"
TESTING_DOGS_DIR = "/tmp/cats-v-dogs/testing/dogs/"

split_size = .9
split_data(CAT_SOURCE_DIR, TRAINING_CATS_DIR, TESTING_CATS_DIR, split_size)
split_data(DOG_SOURCE_DIR, TRAINING_DOGS_DIR, TESTING_DOGS_DIR, split_size)
```

In [6]:
```
print(len(os.listdir('/tmp/cats-v-dogs/training/cats/')))
print(len(os.listdir('/tmp/cats-v-dogs/training/dogs/')))
print(len(os.listdir('/tmp/cats-v-dogs/testing/cats/')))
print(len(os.listdir('/tmp/cats-v-dogs/testing/dogs/')))

# Expected output:
# 1350
# 1350
# 150
# 150
```

```
1350
1350
150
150
```

In [7]:
```
#Looking at a few pictures to get a better sense of the images in the datasets
%matplotlib inline

import matplotlib.image as mpimg
import matplotlib.pyplot as plt

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

pic_index = 0 # Index for iterating over images

train_cat_fnames = os.listdir(TRAINING_CATS_DIR)
train_dog_fnames = os.listdir(TRAINING_DOGS_DIR)
```

In [8]:
```
#Displaying a batch of 8cats and 8dogs
# 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_cat_pix = [os.path.join(TRAINING_CATS_DIR, fname)
                for fname in train_cat_fnames[ pic_index-8:pic_index]
               ]

next_dog_pix = [os.path.join(TRAINING_DOGS_DIR, fname)
                for fname in train_dog_fnames[ pic_index-8:pic_index]
               ]

for i, img_path in enumerate(next_cat_pix+next_dog_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()
```


TIDY CATS
I'm not purebred
Bozo
JUST PURE FUN!

In [9]:
```
# DEFINE A KERAS MODEL TO CLASSIFY CATS V DOGS
model = tf.keras.models.Sequential([
# Note the input shape is the desired size of the image 150x150 with 3 bytes c
olor
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(150, 150
, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    # The second convolution
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # The third convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # The fourth convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # The fifth convolution
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(512, activation='relu'),
    # Only 1 output neuron. It will contain a value from 0-1 where 0 for 1 cla
ss ('cats') and 1 for the other ('dogs')
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer=RMSprop(lr=0.001), loss='binary_crossentropy', metrics
=['acc'])
```

In [10]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 148, 148, 16)      448       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 16)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 32)        4640      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 64)        18496     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 64)        36928     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 64)          0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 5, 5, 128)         73856     
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 2, 2, 128)         0         
_________________________________________________________________
flatten (Flatten)            (None, 512)               0         
_________________________________________________________________
dense (Dense)                (None, 512)               262656    
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513       
=================================================================
Total params: 397,537
Trainable params: 397,537
Non-trainable params: 0
_________________________________________________________________
```

In [11]:

```
TRAINING_DIR = "/tmp/cats-v-dogs/training"
train_datagen = ImageDataGenerator( rescale = 1./255,
                                    rotation_range=40,
                                    width_shift_range=0.2,
                                    height_shift_range=0.2,
                                    shear_range=0.2,
                                    zoom_range=0.2,
                                    horizontal_flip=True,
                                    fill_mode='nearest')

# NOTE: YOU MUST USE A BATCH SIZE OF 10 (batch_size=10) FOR THE
# TRAIN GENERATOR.
train_generator = train_datagen.flow_from_directory(TRAINING_DIR,
                                                    batch_size=10,
                                                    class_mode='binary',
                                                    target_size=(150, 150))

VALIDATION_DIR = "/tmp/cats-v-dogs/testing"
validation_datagen = ImageDataGenerator( rescale = 1.0/255. )

# NOTE: YOU MUST USE A BACTH SIZE OF 10 (batch_size=10) FOR THE
# VALIDATION GENERATOR.
validation_generator = validation_datagen.flow_from_directory(VALIDATION_DIR,
                                                              batch_size=10,
                                                              class_mode  = 'binar
y',
                                                              target_size = (150, 1
50))




# Expected Output:
# Found 2700 images belonging to 2 classes.
# Found 300 images belonging to 2 classes.
```

```
Found 2700 images belonging to 2 classes.
Found 300 images belonging to 2 classes.
```

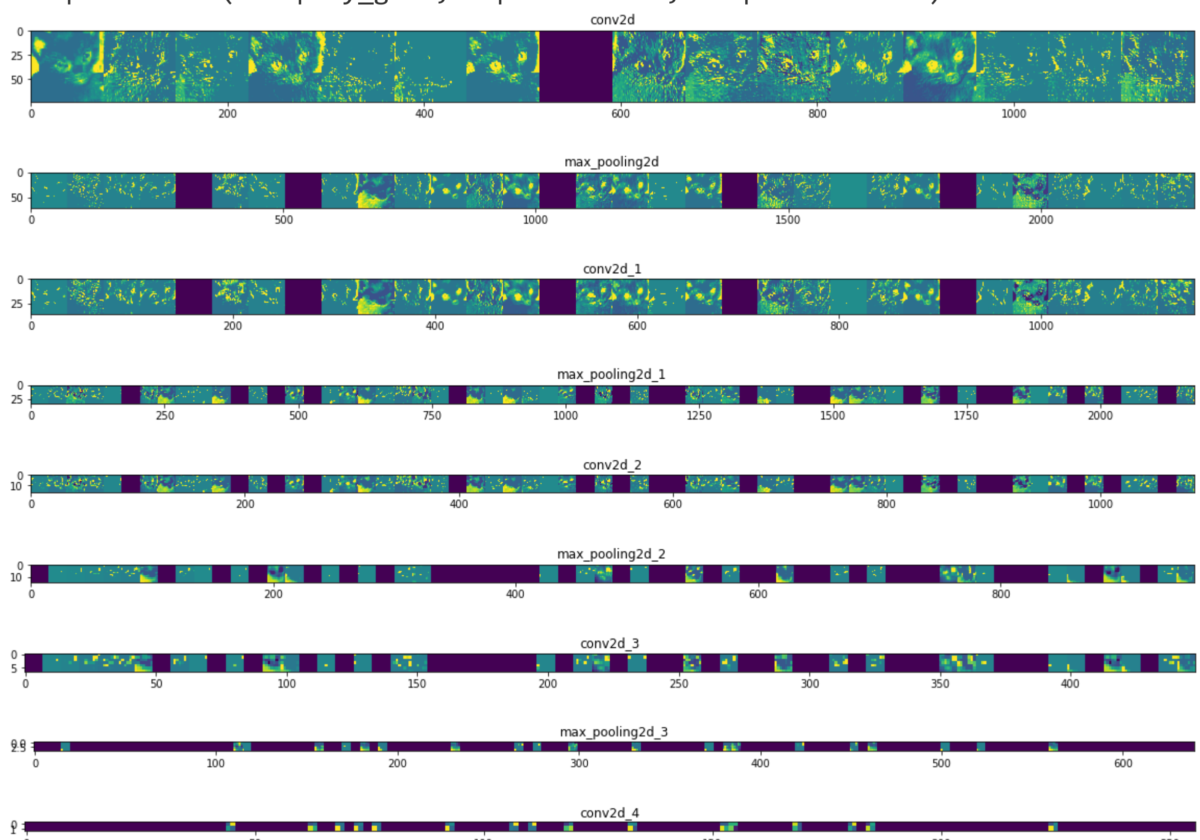
In [12]:
```
history = model.fit_generator(train_generator,
                              epochs=20,
                              verbose=1,
                              validation_data=validation_generator)
```

```
Epoch 1/20
270/270 [==============================] - 69s 255ms/step - loss: 0.6965 - ac
c: 0.5270 - val_loss: 0.6833 - val_acc: 0.5433
Epoch 2/20
270/270 [==============================] - 63s 234ms/step - loss: 0.6836 - ac
c: 0.5656 - val_loss: 0.6463 - val_acc: 0.6300
Epoch 3/20
270/270 [==============================] - 62s 231ms/step - loss: 0.6655 - ac
c: 0.6137 - val_loss: 0.6446 - val_acc: 0.6333
Epoch 4/20
270/270 [==============================] - 62s 230ms/step - loss: 0.6583 - ac
c: 0.5996 - val_loss: 0.6238 - val_acc: 0.6033
Epoch 5/20
270/270 [==============================] - 61s 224ms/step - loss: 0.6540 - ac
c: 0.6196 - val_loss: 0.6147 - val_acc: 0.6367
Epoch 6/20
270/270 [==============================] - 62s 229ms/step - loss: 0.6426 - ac
c: 0.6348 - val_loss: 0.6271 - val_acc: 0.6933
Epoch 7/20
270/270 [==============================] - 62s 230ms/step - loss: 0.6356 - ac
c: 0.6422 - val_loss: 0.6114 - val_acc: 0.6733
Epoch 8/20
270/270 [==============================] - 62s 231ms/step - loss: 0.6375 - ac
c: 0.6474 - val_loss: 0.5979 - val_acc: 0.7100
Epoch 9/20
270/270 [==============================] - 62s 230ms/step - loss: 0.6341 - ac
c: 0.6507 - val_loss: 0.5640 - val_acc: 0.7067
Epoch 10/20
270/270 [==============================] - 63s 234ms/step - loss: 0.6323 - ac
c: 0.6496 - val_loss: 0.5766 - val_acc: 0.7133
Epoch 11/20
270/270 [==============================] - 62s 229ms/step - loss: 0.6155 - ac
c: 0.6726 - val_loss: 0.5798 - val_acc: 0.6900
Epoch 12/20
270/270 [==============================] - 64s 235ms/step - loss: 0.6013 - ac
c: 0.6804 - val_loss: 0.5568 - val_acc: 0.7333
Epoch 13/20
270/270 [==============================] - 62s 230ms/step - loss: 0.5994 - ac
c: 0.6819 - val_loss: 0.5833 - val_acc: 0.7100
Epoch 14/20
270/270 [==============================] - 64s 237ms/step - loss: 0.5968 - ac
c: 0.6933 - val_loss: 0.5485 - val_acc: 0.7200
Epoch 15/20
270/270 [==============================] - 63s 234ms/step - loss: 0.5959 - ac
c: 0.6896 - val_loss: 0.5644 - val_acc: 0.7433
Epoch 16/20
270/270 [==============================] - 65s 240ms/step - loss: 0.6005 - ac
c: 0.7004 - val_loss: 0.5832 - val_acc: 0.7267
Epoch 17/20
270/270 [==============================] - 62s 229ms/step - loss: 0.5784 - ac
c: 0.7037 - val_loss: 0.6477 - val_acc: 0.6300
Epoch 18/20
270/270 [==============================] - 63s 233ms/step - loss: 0.5856 - ac
c: 0.7011 - val_loss: 0.6618 - val_acc: 0.6400
Epoch 19/20
270/270 [==============================] - 63s 234ms/step - loss: 0.5963 - ac
c: 0.6941 - val_loss: 0.5560 - val_acc: 0.7067
```

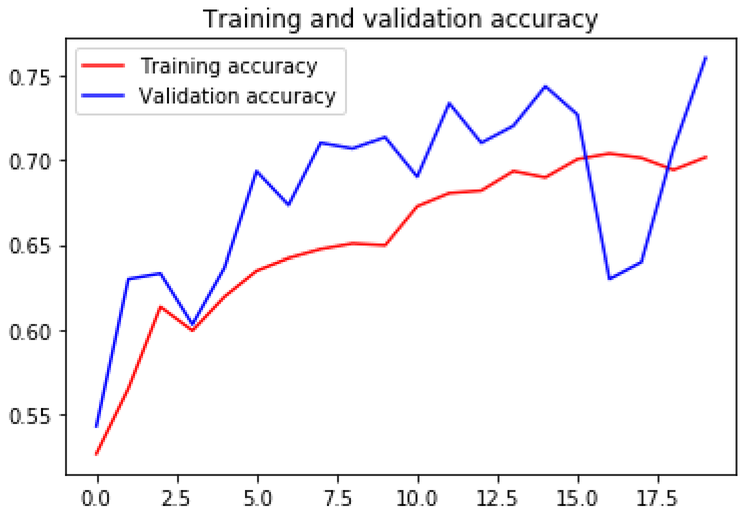
```
Epoch 20/20
270/270 [==============================] - 63s 234ms/step - loss: 0.5991 - ac
c: 0.7015 - val_loss: 0.5175 - val_acc: 0.7600
```

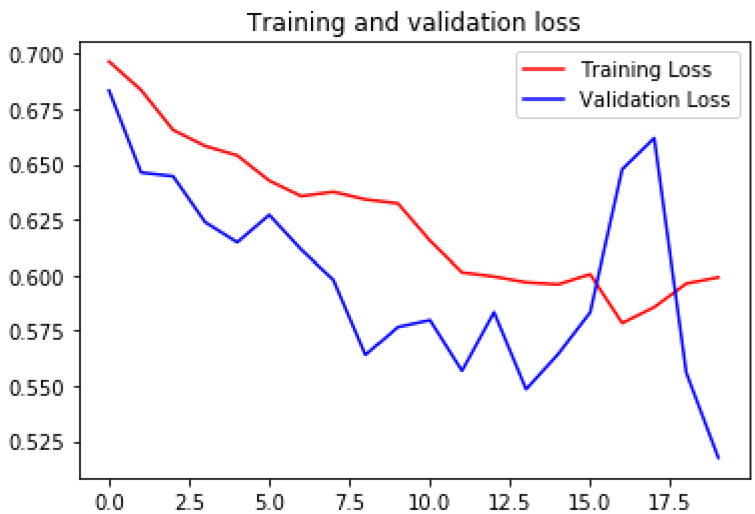
In [13]:
```
#Visualizing intermediate representations during training
import numpy as np
from   tensorflow.keras.preprocessing.image import img_to_array, load_img
# Let's define a new Model that will take an image as input, and will output
# intermediate representations for all layers in the previous model after
# the first.
successive_outputs = [layer.output for layer in model.layers[1:]]

#visualization_model = Model(img_input, successive_outputs)
visualization_model = tf.keras.models.Model(inputs = model.input, outputs = su
ccessive_outputs)

# Let's prepare a random input image of a cat or dog from the training set.
cat_img_files = [os.path.join(TRAINING_CATS_DIR, f) for f in train_cat_fnames]
dog_img_files = [os.path.join(TRAINING_DOGS_DIR, f) for f in train_dog_fnames]

img_path = random.choice(cat_img_files + dog_img_files)
img = load_img(img_path, target_size=(150, 150))  # this is a PIL image

x   = img_to_array(img)                           # Numpy array with shape (15
0, 150, 3)
x   = x.reshape((1,) + x.shape)                   # Numpy array with shape (1,
150, 150, 3)

# Rescale by 1/255
x /= 255.0

# Let's run our image through our network, thus obtaining all
# intermediate representations for this image.
successive_feature_maps = visualization_model.predict(x)

# These are the names of the layers, so can have them as part of our plot
layer_names = [layer.name for layer in model.layers]

# -----------------------------------------------------------------------
# Now let's display our representations
# -----------------------------------------------------------------------
for layer_name, feature_map in zip(layer_names, successive_feature_maps):

  if len(feature_map.shape) == 4:

    #-------------------------------------------
    # Just do this for the conv / maxpool layers, not the fully-connected laye
rs
    #-------------------------------------------
    n_features = feature_map.shape[-1]  # number of features in the feature ma
p
    size       = feature_map.shape[ 1]  # feature map shape (1, size, size, n_
features)

    # We will tile our images in this matrix
    display_grid = np.zeros((size, size * n_features))

    #-------------------------------------------------
    # Postprocess the feature to be visually palatable
    #-------------------------------------------------
```

```
    for i in range(n_features):
      x  = feature_map[0, :, :, i]
      x -= x.mean()
      x /= x.std ()
      x *=  64
      x += 128
      x  = np.clip(x, 0, 255).astype('uint8')
      display_grid[:, i * size : (i + 1) * size] = x # Tile each filter into a horizontal grid

    #-----------------
    # Display the grid
    #-----------------

    scale = 20. / n_features
    plt.figure( figsize=(scale * n_features, scale) )
    plt.title ( layer_name )
    plt.grid  ( False )
    plt.imshow( display_grid, aspect='auto', cmap='viridis' )
```



In [14]:
```
# PLOT LOSS AND ACCURACY
%matplotlib inline
import matplotlib.pyplot as plt
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, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()

plt.plot(epochs, loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()

# Desired output. Charts with training and validation metrics. No crash :)
```





Applying Transfer Learning & Regularization

I could not achieve low bias and low variance even with image augmentation, deeper network, and longer training (epoch=100) for this project. Hence, I decided to apply transfer learning from an already built Inception Network.

NOTE: Epoch value of 20 was intentionally used in this realization. Higher training times have been used. Comparing the two models, it shows that within 20 epochs, the Inception model achieves lower bias and variance error compared to the first.

In [15]:
```
from tensorflow.keras import layers
from tensorflow.keras import Model
from os import getcwd
path_inception = f"{getcwd()}/../tmp2/inception_v3_weights_tf_dim_ordering_tf_
kernels_notop.h5"

# Import the inception model
from tensorflow.keras.applications.inception_v3 import InceptionV3

# Create an instance of the inception model from the local pre-trained weights
local_weights_file = path_inception

pre_trained_model = InceptionV3(input_shape = (150,150,3),
                                include_top = False,
                                weights = None)

pre_trained_model.load_weights(local_weights_file)

# Make all the layers in the pre-trained model non-trainable
for layer in pre_trained_model.layers:
    layer.trainable = False

# Print the model summary
#pre_trained_model.summary()
```

In [16]:
```
last_layer = pre_trained_model.get_layer('mixed7')
print('last layer output shape: ', last_layer.output_shape)
last_output = last_layer.output
```

```
last layer output shape:  (None, 7, 7, 768)
```

In [17]:
```
# Defining a Callback class that stops training once accuracy reaches 97.0%
class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('acc')>0.97):
            print("\nReached 97.0% accuracy so cancelling training!")
            self.model.stop_training = True
```

In [18]:
```
# Flatten the output layer to 1 dimension
x = layers.Flatten()(last_output)
# Add a fully connected layer with 1,024 hidden units and ReLU activation
x = layers.Dense(1024, activation='relu')(x)
# Add a dropout rate of 0.2
x = layers.Dropout(0.2)(x)
# Add a final sigmoid layer for classification
x = layers.Dense  (1, activation='sigmoid')(x)

model = Model( pre_trained_model.input, x)

model.compile(optimizer = RMSprop(lr=0.0001),
              loss = 'binary_crossentropy',
              metrics = ['accuracy'])

#model.summary()
```

In [19]:
```
callbacks = myCallback()
history = model.fit_generator(train_generator,
                              epochs=20,
                              verbose=1,
                              validation_data=validation_generator)
```

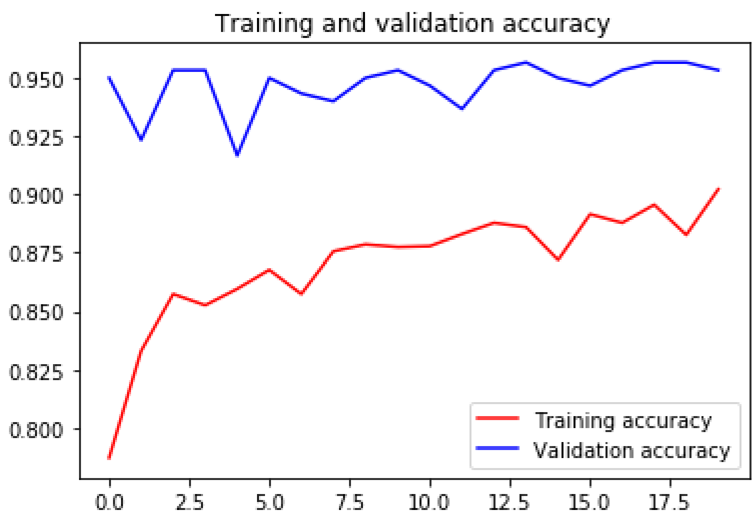
```
Epoch 1/20
270/270 [==============================] - 141s 521ms/step - loss: 0.4743 - a
ccuracy: 0.7870 - val_loss: 0.2512 - val_accuracy: 0.9500
Epoch 2/20
270/270 [==============================] - 135s 499ms/step - loss: 0.3997 - a
ccuracy: 0.8330 - val_loss: 0.5016 - val_accuracy: 0.9233
Epoch 3/20
270/270 [==============================] - 136s 504ms/step - loss: 0.3518 - a
ccuracy: 0.8574 - val_loss: 0.3940 - val_accuracy: 0.9533
Epoch 4/20
270/270 [==============================] - 136s 502ms/step - loss: 0.3675 - a
ccuracy: 0.8526 - val_loss: 0.3888 - val_accuracy: 0.9533
Epoch 5/20
270/270 [==============================] - 137s 506ms/step - loss: 0.3347 - a
ccuracy: 0.8596 - val_loss: 0.6543 - val_accuracy: 0.9167
Epoch 6/20
270/270 [==============================] - 135s 500ms/step - loss: 0.3297 - a
ccuracy: 0.8678 - val_loss: 0.4375 - val_accuracy: 0.9500
Epoch 7/20
270/270 [==============================] - 137s 509ms/step - loss: 0.3396 - a
ccuracy: 0.8574 - val_loss: 0.5035 - val_accuracy: 0.9433
Epoch 8/20
270/270 [==============================] - 137s 508ms/step - loss: 0.3293 - a
ccuracy: 0.8756 - val_loss: 0.4743 - val_accuracy: 0.9400
Epoch 9/20
270/270 [==============================] - 136s 503ms/step - loss: 0.3088 - a
ccuracy: 0.8785 - val_loss: 0.5008 - val_accuracy: 0.9500
Epoch 10/20
270/270 [==============================] - 137s 509ms/step - loss: 0.3307 - a
ccuracy: 0.8774 - val_loss: 0.5184 - val_accuracy: 0.9533
Epoch 11/20
270/270 [==============================] - 138s 510ms/step - loss: 0.3080 - a
ccuracy: 0.8778 - val_loss: 0.4856 - val_accuracy: 0.9467
Epoch 12/20
270/270 [==============================] - 137s 506ms/step - loss: 0.3103 - a
ccuracy: 0.8830 - val_loss: 0.6337 - val_accuracy: 0.9367
Epoch 13/20
270/270 [==============================] - 134s 498ms/step - loss: 0.2833 - a
ccuracy: 0.8878 - val_loss: 0.4993 - val_accuracy: 0.9533
Epoch 14/20
270/270 [==============================] - 139s 514ms/step - loss: 0.3108 - a
ccuracy: 0.8859 - val_loss: 0.3284 - val_accuracy: 0.9567
Epoch 15/20
270/270 [==============================] - 137s 509ms/step - loss: 0.3090 - a
ccuracy: 0.8719 - val_loss: 0.4936 - val_accuracy: 0.9500
Epoch 16/20
270/270 [==============================] - 139s 513ms/step - loss: 0.2975 - a
ccuracy: 0.8915 - val_loss: 0.5877 - val_accuracy: 0.9467
Epoch 17/20
270/270 [==============================] - 144s 534ms/step - loss: 0.2908 - a
ccuracy: 0.8878 - val_loss: 0.5277 - val_accuracy: 0.9533
Epoch 18/20
270/270 [==============================] - 136s 504ms/step - loss: 0.2948 - a
ccuracy: 0.8956 - val_loss: 0.4450 - val_accuracy: 0.9567
Epoch 19/20
270/270 [==============================] - 141s 523ms/step - loss: 0.3098 - a
ccuracy: 0.8826 - val_loss: 0.4800 - val_accuracy: 0.9567
```

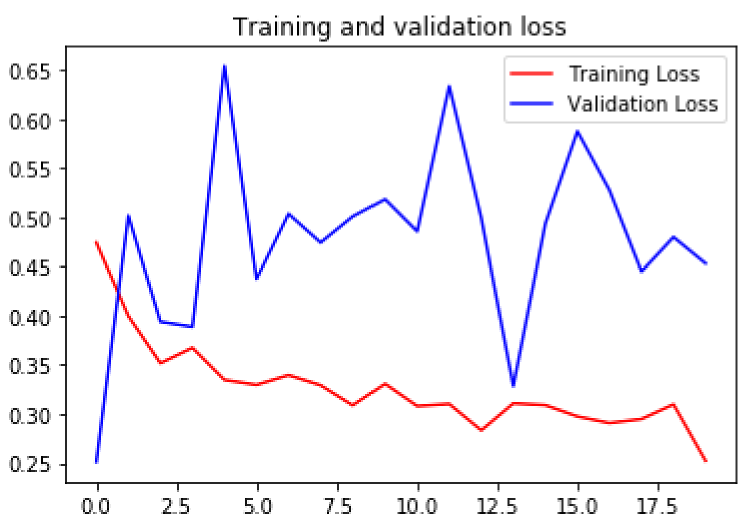
```
Epoch 20/20
270/270 [==============================] - 136s 504ms/step - loss: 0.2523 - a
ccuracy: 0.9022 - val_loss: 0.4533 - val_accuracy: 0.9533
```

In [26]:
```
#Visualizing intermediate representations during training
import numpy as np
from   tensorflow.keras.preprocessing.image import img_to_array, load_img
# Let's define a new Model that will take an image as input, and will output
# intermediate representations for all layers in the previous model after
# the first.
successive_outputs = [layer.output for layer in model.layers[1:]]

#visualization_model = Model(img_input, successive_outputs)
visualization_model = tf.keras.models.Model(inputs = model.input, outputs = su
ccessive_outputs)

# Let's prepare a random input image of a cat or dog from the training set.
cat_img_files = [os.path.join(TRAINING_CATS_DIR, f) for f in train_cat_fnames]
dog_img_files = [os.path.join(TRAINING_DOGS_DIR, f) for f in train_dog_fnames]

img_path = random.choice(cat_img_files + dog_img_files)
img = load_img(img_path, target_size=(150, 150))  # this is a PIL image

x   = img_to_array(img)                           # Numpy array with shape (15
0, 150, 3)
x   = x.reshape((1,) + x.shape)                   # Numpy array with shape (1,
150, 150, 3)

# Rescale by 1/255
x /= 255.0

# Let's run our image through our network, thus obtaining all
# intermediate representations for this image.
successive_feature_maps = visualization_model.predict(x)

# These are the names of the layers, so can have them as part of our plot
layer_names = [layer.name for layer in model.layers]

# -----------------------------------------------------------------------
# Now let's display our representations
# -----------------------------------------------------------------------
for layer_name, feature_map in zip(layer_names, successive_feature_maps):

  if len(feature_map.shape) == 4:

    #-------------------------------------------
    # Just do this for the conv / maxpool layers, not the fully-connected laye
rs
    #-------------------------------------------
    n_features = feature_map.shape[-1]  # number of features in the feature ma
p
    size       = feature_map.shape[ 1]  # feature map shape (1, size, size, n_
features)

    # We will tile our images in this matrix
    display_grid = np.zeros((size, size * n_features))

    #-------------------------------------------------
    # Postprocess the feature to be visually palatable
    #-------------------------------------------------
```

```
    for i in range(n_features):
      x  = feature_map[0, :, :, i]
      x -= x.mean()
      x /= x.std ()
      x *=  64
      x += 128
      x  = np.clip(x, 0, 255).astype('uint8')
      display_grid[:, i * size : (i + 1) * size] = x # Tile each filter into a horizontal grid

    #-----------------
    # Display the grid
    #-----------------

    scale = 20. / n_features
    plt.figure( figsize=(scale * n_features, scale) )
    plt.title ( layer_name )
    plt.grid  ( False )
    plt.imshow( display_grid, aspect='auto', cmap='viridis' )
```

In [25]:
```
# PLOT LOSS AND ACCURACY
%matplotlib inline
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()

plt.plot(epochs, loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





In [ ]: