1. **Data preparation, exploration, visualization**

   In this assignment, not much needed to be done in terms of preparing the data as much of the data was image data sets. In this assignment, the training data was in the form of a zip file so I first had to unzip the data before it could be used for analysis. Once the data was unzipped, I then obtained the 25,000 images that were contained in the training data, classified the images as either dog or cat images based on the category, and created a data frame for the dog and cat images. Using the data frame that was created, I then created a training and validation set within the training data using train test split and proceeded to build the neural network to classify the images as either cats or dogs. The classification was based on the filename. i.e., cats contained the name cat in the filename and dogs contained the name dog in the filename. Also, since the model needed to predict the probability that a given picture is a dog, all dogs were scored as a 1 and all cats were scored as 0.

2. **Review research design and modeling methods**

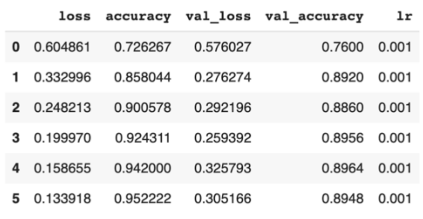
   There are a lot of similarities in creating neural networks using CNNs and creating neural networks in Assignment 6. However, there are also a lot of differences. For example, each layer in the CNN used BatchNormalization to ensure the values of that layer were between 0 and 1 as well as used MaxPooling to select the maximum value from a set of neighboring pixels. The convolutional layer also had parameters such as the size of the filter, the number of filters, the stride value the padding value, etc. The neural network had a fully connected layer with a dropout layer just before the output. The dropout layer arbitrarily drops some percentage of the neurons and this mechanism ensures that the neural network did not overfit to the training data and will generalize to the test data. Once, the CNN trained on the data, I then evaluated the performance of the CNN with regards to the accuracy and loss of both the training and validation sets. Finally, I prepared the test data for submission and submitted my results to Kaggle for evaluation. Below is one such CNN that I created:

Model	Number of layers	Number of filters per layer	Kernel Size	Pool Size	Strides	Dropout
1	5	Filter 1 – 32 Filter 2 – 64 Filter 3 – 128 Filter 4 – 256 Filter 5 – 256	Layer 1 – (3,3) Layer 2 – (3,3) Layer 3 – (3,3) Layer 4 – (3,3) Layer 5 - (3,3)	Layer 1 – (2,2) Layer 2 – (2,2) Layer 3 – (2,2) Layer 4 – (2,2) Layer 5 - (2,2)	Layer 1 – 2 Layer 2 – 2 Layer 3 – 2 Layer 4 – 2 Layer 5 - 2	0.2

3. **Review results, evaluate models**

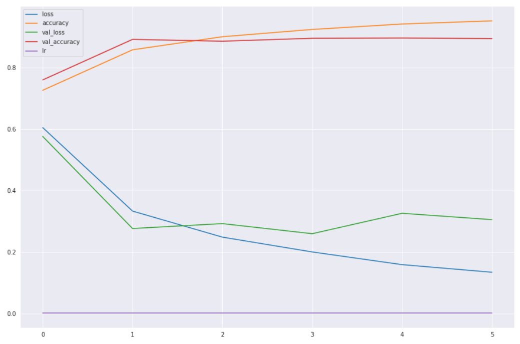
   I felt that the results generated by the CNNs were acceptable. The model that performed the best was the third model as it generated the lowest score when submitted to Kaggle. Deeper neural networks create more abstract features while broader neural networks have the ability to model more information. So, neural networks that are both deep and broad have the ability to model a lot of information which can lead to overfitting. Any model should follow the principles of Occam's Razor i.e. the model should be minimalistic which will also provide the right trade-off between accuracy (precision) and coverage (recall). I do think that one reason why the CNN didn't perform as well as I expected is because of the possibility of overfitting as well as underfitting. For example, as the training loss for model 4 decreased, the validation loss increased significantly starting from epoch 0. In addition, as the training loss for model 3 decreased, the validation loss increased significantly starting from epoch 2, yet that didn't seem to affect the performance of the model as the score ended up being the best of all of the models. Some methods that could be used to account for overfitting include reducing the number of layers in the CNN, adding dropout layers, and applying early stopping so that the CNN stops training as soon as the validation loss increases. Below are some of the results that I got:

   Model 1 Results

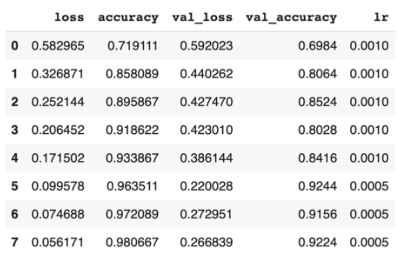


	loss	accuracy	val_loss	val_accuracy	lr
0	0.604861	0.726267	0.576027	0.7600	0.001
1	0.332996	0.858044	0.276274	0.8920	0.001
2	0.248213	0.900578	0.292196	0.8860	0.001
3	0.199970	0.924311	0.259392	0.8956	0.001
4	0.158655	0.942000	0.325793	0.8964	0.001
5	0.133918	0.952222	0.305166	0.8948	0.001

Table 1: Results for Model 1

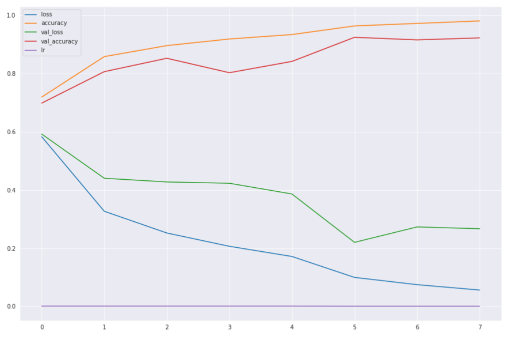


Model 2 Results

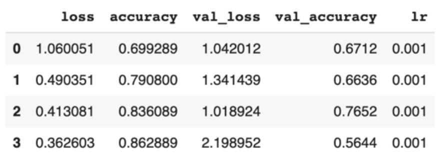


	loss	accuracy	val_loss	val_accuracy	lr
0	0.582965	0.719111	0.592023	0.6984	0.0010
1	0.326871	0.858089	0.440262	0.8064	0.0010
2	0.252144	0.895867	0.427470	0.8524	0.0010
3	0.206452	0.918622	0.423010	0.8028	0.0010
4	0.171502	0.933867	0.386144	0.8416	0.0010
5	0.099578	0.963511	0.220028	0.9244	0.0005
6	0.074688	0.972089	0.272951	0.9156	0.0005
7	0.056171	0.980667	0.266839	0.9224	0.0005

Table 2: Results for Model 2

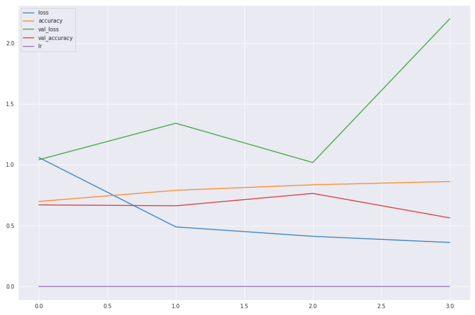


Model 3 Results

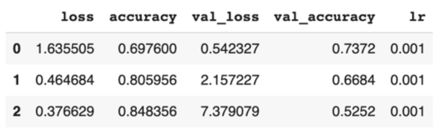


	loss	accuracy	val_loss	val_accuracy	lr
0	1.060051	0.699289	1.042012	0.6712	0.001
1	0.490351	0.790800	1.341439	0.6636	0.001
2	0.413081	0.836089	1.018924	0.7652	0.001
3	0.362603	0.862889	2.198952	0.5644	0.001

Table 3: Results for Model 3

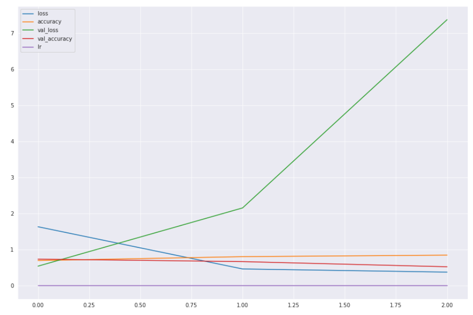


Model 4 Results



	loss	accuracy	val_loss	val_accuracy	lr
0	1.635505	0.697600	0.542327	0.7372	0.001
1	0.464684	0.805956	2.157227	0.6684	0.001
2	0.376629	0.848356	7.379079	0.5252	0.001

Table 4: Results for Model 4



4. **Kaggle Submission relative to Peers**

CNN	Score
1	4.41866
2	5.61748
3	3.50776
4	8.74694

5. **Exposition, problem description and management recommendations**

For image classification problems, neural networks outperform all other algorithms. Specifically, CNNs produce state of the art performance with respect to image recognition while generalizing well (i.e. not overfitting the data). I would recommend using CNNs such as a 1-layer CNN, a 2-layer CNN, and a 3-layer CNN as they don't have many layers and therefore are much less complex to train on, thereby making it less likely to overfit the data. I think that the types of images that work best for neural networks are 2D images such as the MNIST images and the cats and dogs images mainly because compared to other image classification algorithms, CNNs actually use very little preprocessing and therefore can learn the filters that otherwise would have to be hand made in other algorithms. In addition, I think in general images that are definite in nature work best for neural networks. For example, in the Kannada MNIST data, the images weren't that definite due to how certain numbers are written whereas images of items such as clothing items and images of cats and dogs are more definite in nature. This could definitely play a role in how the CNN is able to train on the images that are provided as input

MSDS 422 Assignment 7 (CNN #1)

May 18, 2021

1 Assignment 7: Image Processing With a CNN

This assignment follows the same structure as Assignment 6. We will employ at least a 2x2 completely crossed experimental design. We will again use a simple training-and-test regimen. The factors in the design may include numbers of layers and/or nodes within layers, types of layers (convolutional or pooling), and/or other hyperparameters. You will utilize convolutional neural networks (CNNs) within Python TensorFlow.

This week, you will compete in the Dogs vs. Cats kernels Edition Kaggle.com competition, https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition (Links to an external site.). . (Links to an external site.)Specifically, you will build models using the training set to forecast the test set. The images are in .jpg format, so you will need to research how to handle that. You are required to submit no fewer than four models for evaluation by Kaggle.com, and you must provide your Kaggle.com scores and user ID for validation.

2 Management Problem

Assume that we are providing advice to a website provider who is looking for tools to automatically label images provided by end users. As we look across the factors in the study, making recommendations to management about image classification, we are most concerned about achieving the highest possible accuracy in image classification. That is, we should be willing to sacrifice training time for model accuracy. What type of machine learning model works best? If it is a convolutional neural network, what type of network should we use? Part of this recommendation may concern information about the initial images themselves (input data for the classification task). What types of images work best?

3 1. Preparation of Data and Exploratory Data Analysis

[ ]:
```
# This Python 3 environment comes with many helpful analytics libraries
# installed
# It is defined by the kaggle/python Docker image:
# https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

```
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter)
# will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that
# gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved
# outside of the current session
```

[ ]:
```
import os
import zipfile
import random
import tensorflow as tf
import shutil
import numpy as np
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.regularizers import l2
from shutil import copyfile
from os import getcwd
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive/')
```

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

[ ]:
```
!ls drive/MyDrive/
```

```
'AYUDH Table Presentation (1).ppt'
'AYUDH Table Presentation (1).ppt.gslides'
'AYUDH Table Schedule (Santa Fe - Toronto).gsheet'
'AYUDH Table Summer 2018 (Santa Fe - Toronto) (1).gsheet'
'AYUDH Table Summer 2018 (Santa Fe - Toronto) (1).xlsx'
'AYUDH Table Summer 2018 (Santa Fe - Toronto) (2).xlsx'
'AYUDH Table Summer 2018 (Santa Fe - Toronto).gsheet'
'AYUDH Table Summer 2018 (Santa Fe - Toronto).xlsx'
'Brochure (1).gdoc'
 Brochure.gdoc
'CECS Overview 7.20.2020.gdoc'
 cnn_model2_submission.csv
 cnn_model3_submission.csv
 cnn_model4_submission.csv
 cnn_submission.csv
```

```
'COVID-19 Exploratory Data Analysis in R.gdoc'
'COVID-19 Research Paper Analysis.docx'
'DASS 21 Data v1.gsheet'
'DASS 21 Data v1.xlsx'
'ENGL 001C- Essay 3 Proposal.gdoc'
'ENGL 001C Thesis Brainstorm.gdoc'
'English 001C Essay 2.docx'
'English 001C Essay 2.gdoc'
'English 001C- Final Notes.gdoc'
'English 1B- Essay 2 (final draft).docx'
'English 1B- Essay 2 (peer review).docx'
'English 4 Final Room Assignments F16.docx'
'English 4L Reading quizzes with answers.docx'
'English 4L Reading quizzes with answers.docx.gdoc'
'English 4L Reading quizzes with answers.gdoc'
'Math 144- Chapter 10 problems.gdoc'
'Math 144- Chapter 7 worked out problems.gdoc'
'Math 144 Chapter 8 problems.gdoc'
'Math 144 Worked out problems- Chapter 9.gdoc'
'Math 168 Project 1.gdoc'
'Microsoft Band Sizing Guide_EN.pdf'
'MSDS 422 Assignment 7 (CNN #1).ipynb'
'MSDS 422 Assignment 7 (CNN #2).ipynb'
'MSDS 422 Assignment 7 (CNN #3).ipynb'
'MSDS 422 Assignment 7 (CNN #4).ipynb'
'NY revised.xlsx'
'NY revised.xlsx.gsheet'
'Play Books Notes'
 Report.gdoc
'Safari - Jul 23, 2018 at 01:12.pdf'
 Schedule.gsheet
 test.zip
 train.zip
'Untitled document (1).gdoc'
'Untitled document (2).gdoc'
'Untitled document (3).gdoc'
'Untitled document (4).gdoc'
'Untitled document (5).gdoc'
'Untitled document.gdoc'
'Untitled spreadsheet (1).gsheet'
'Untitled spreadsheet (2).gsheet'
'Untitled spreadsheet.gsheet'
 Week3Lecture_default.mp4
```

[ ]:
```
local_zip = '/content/drive/MyDrive/train.zip'

zip_ref = zipfile.ZipFile(local_zip, 'r')
```

```
zip_ref.extractall('/kaggle/working/')
zip_ref.close()
```

[ ]:
```
base_dir = '/kaggle/working/'
train_dir = os.path.join(base_dir, 'train')
train_img_names = os.listdir(train_dir)
```

[ ]:
```
train_img_names[:10]
```

[ ]:
```
['dog.5577.jpg',
 'dog.8301.jpg',
 'cat.11000.jpg',
 'dog.8878.jpg',
 'dog.11820.jpg',
 'cat.9373.jpg',
 'cat.5203.jpg',
 'dog.831.jpg',
 'dog.2816.jpg',
 'cat.6415.jpg']
```

[ ]:
```
print('total training images :', len(train_img_names ))
```

```
total training images : 25000
```

[ ]:
```
categories= list()
for image in train_img_names:
    category = image.split(".")[0]
    if category == "dog":
        categories.append("1")
    else:
        categories.append("0")

df= pd.DataFrame({"Image":train_img_names, "Category": categories})
```

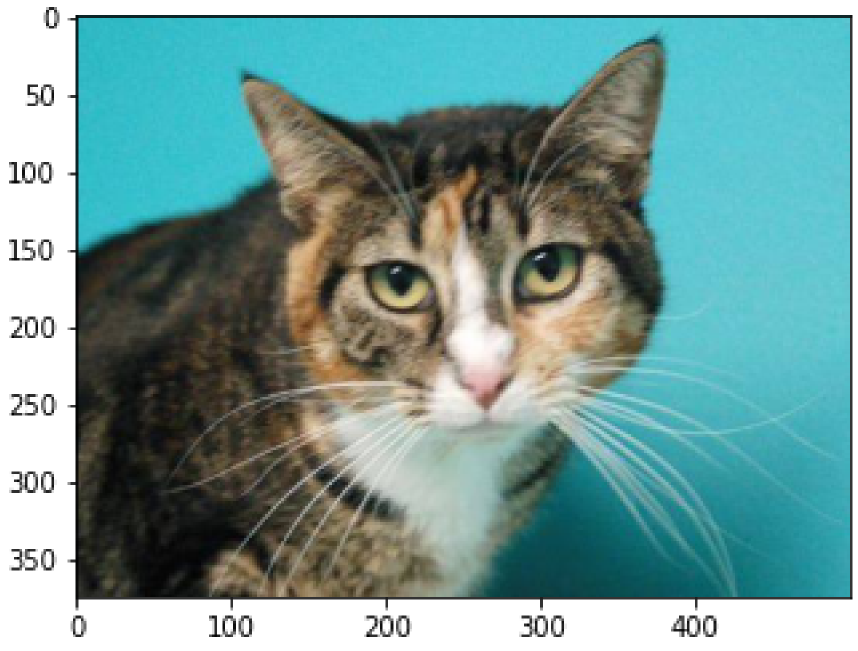
[ ]:
```
df
```

[ ]:
```
                Image Category
0        dog.5577.jpg        1
1        dog.8301.jpg        1
2       cat.11000.jpg        0
3        dog.8878.jpg        1
4       dog.11820.jpg        1
…                 …        …
24995    cat.2927.jpg        0
24996    cat.9101.jpg        0
24997   cat.11962.jpg        0
24998    dog.2348.jpg        1
```

```
24999    dog.542.jpg        1

[25000 rows x 2 columns]
```

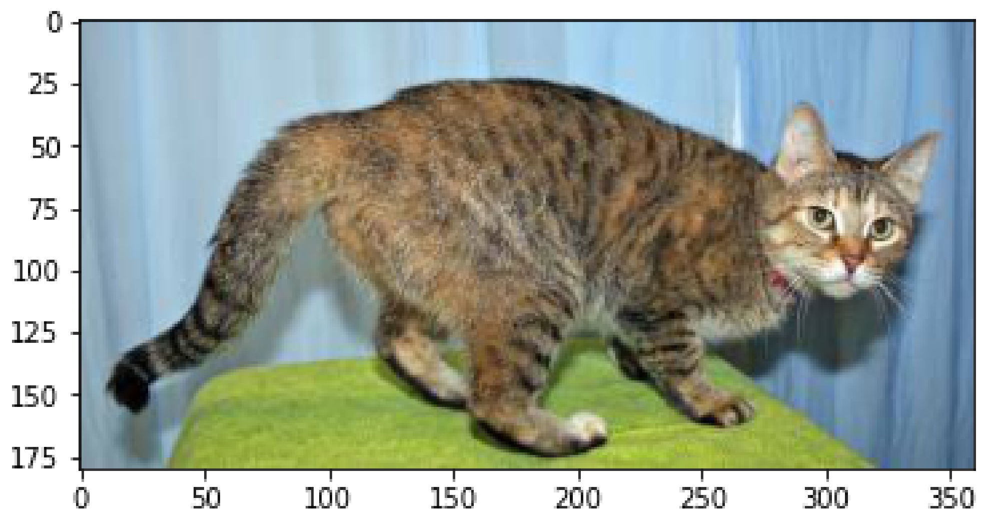
[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import random

sample = random.choice(train_img_names)
plt.imshow(plt.imread(("/kaggle/working/train/"+sample)))
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7fe2dd2c09d0>
```



[ ]:
```
sample = random.choice(train_img_names)
plt.imshow(plt.imread(("/kaggle/working/train/"+sample)))
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7fe2dcdb6a90>
```



```
[ ]: from sklearn.model_selection import train_test_split
     train,validation= train_test_split(df, test_size=0.1)
     train = train.reset_index(drop=True)
     validation = validation.reset_index(drop=True)
```

```
[ ]: train
```

```
[ ]:                Image Category
     0       dog.3303.jpg        1
     1      cat.12101.jpg        0
     2       cat.8414.jpg        0
     3       cat.6280.jpg        0
     4       dog.6531.jpg        1
     …                   …       …
     22495  cat.12390.jpg        0
     22496   cat.3577.jpg        0
     22497   cat.6982.jpg        0
     22498   cat.9416.jpg        0
     22499   cat.4855.jpg        0

     [22500 rows x 2 columns]
```

```
[ ]: validation
```

```
[ ]:               Image Category
     0      dog.8042.jpg        1
     1      cat.7896.jpg        0
     2       dog.666.jpg        1
     3     cat.11354.jpg        0
```

```
4     dog.11241.jpg        1
…             …       …
2495   cat.7325.jpg        0
2496    dog.263.jpg        1
2497   cat.1228.jpg        0
2498  dog.10318.jpg        1
2499   dog.7458.jpg        1

[2500 rows x 2 columns]
```

[ ]:
```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255.
train_datagen = ImageDataGenerator( rescale = 1.0/255. )

# --------------------
# Flow training images in batches of 20 using train_datagen generator
# --------------------
train_generator = train_datagen.flow_from_dataframe(train,
directory="/kaggle/working/train", x_col='Image', y_col='Category',
batch_size=20, class_mode='binary', target_size=(150, 150))
```

```
Found 22500 validated image filenames belonging to 2 classes.
```

[ ]:
```python
validation_datagen  = ImageDataGenerator( rescale = 1.0/255.)
validation_generator =  validation_datagen.flow_from_dataframe(validation,
directory="/kaggle/working/train", x_col='Image', y_col='Category',
batch_size=20, class_mode  = 'binary', target_size = (150, 150))
```

```
Found 2500 validated image filenames belonging to 2 classes.
```

4 2. Building a Small Model from Scratch

[ ]:
```python
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPool2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3,3),activation="relu",
input_shape=(150,150,3)))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=2, strides=2))
model.add(Conv2D(filters=64, kernel_size=3,activation="relu"))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=2, strides=2))

model.add(Conv2D(filters=128, kernel_size=3, activation="relu"))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=2, strides=2))

model.add(Conv2D(filters=256, kernel_size=3, activation="relu"))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=2, strides=2))

model.add(Conv2D(filters=256, kernel_size=3, activation="relu"))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=2, strides=2))

model.add(Flatten())
model.add(Dense(units=1024, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(units=1, activation="sigmoid"))
```

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

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
batch_normalization (BatchNo (None, 148, 148, 32)      128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
batch_normalization_1 (Batch (None, 72, 72, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
batch_normalization_2 (Batch (None, 34, 34, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 256)       295168
_________________________________________________________________
```

```
batch_normalization_3 (Batch (None, 15, 15, 256)       1024
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 256)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 5, 5, 256)         590080
_________________________________________________________________
batch_normalization_4 (Batch (None, 5, 5, 256)         1024
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              1049600
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 1025
=================================================================
Total params: 2,032,065
Trainable params: 2,030,593
Non-trainable params: 1,472
_________________________________________________________________
```

```
[ ]: import tensorflow as tf
     model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=0.001),
                   loss='binary_crossentropy', metrics=['accuracy'])
```

```
[ ]: callback=EarlyStopping(monitor="val_loss", patience=2)
     callback_lr = tensorflow.keras.callbacks.ReduceLROnPlateau(
         monitor='val_accuracy', patience=2, factor=0.5, min_lr=0.00001)
```

```
[ ]: history=model.fit(train_generator, validation_data=validation_generator,
                       epochs=8, callbacks=[callback,callback_lr])
```

```
Epoch 1/8
1125/1125 [==============================] - 93s 67ms/step - loss: 0.8927 -
accuracy: 0.6544 - val_loss: 0.5760 - val_accuracy: 0.7600
Epoch 2/8
1125/1125 [==============================] - 75s 66ms/step - loss: 0.3533 -
accuracy: 0.8474 - val_loss: 0.2763 - val_accuracy: 0.8920
Epoch 3/8
1125/1125 [==============================] - 75s 66ms/step - loss: 0.2522 -
accuracy: 0.8978 - val_loss: 0.2922 - val_accuracy: 0.8860
Epoch 4/8
1125/1125 [==============================] - 73s 65ms/step - loss: 0.1896 -
accuracy: 0.9279 - val_loss: 0.2594 - val_accuracy: 0.8956
Epoch 5/8
1125/1125 [==============================] - 73s 65ms/step - loss: 0.1500 -
```

```
accuracy: 0.9447 - val_loss: 0.3258 - val_accuracy: 0.8964
Epoch 6/8
1125/1125 [==============================] - 73s 65ms/step - loss: 0.1294 -
accuracy: 0.9535 - val_loss: 0.3052 - val_accuracy: 0.8948
```

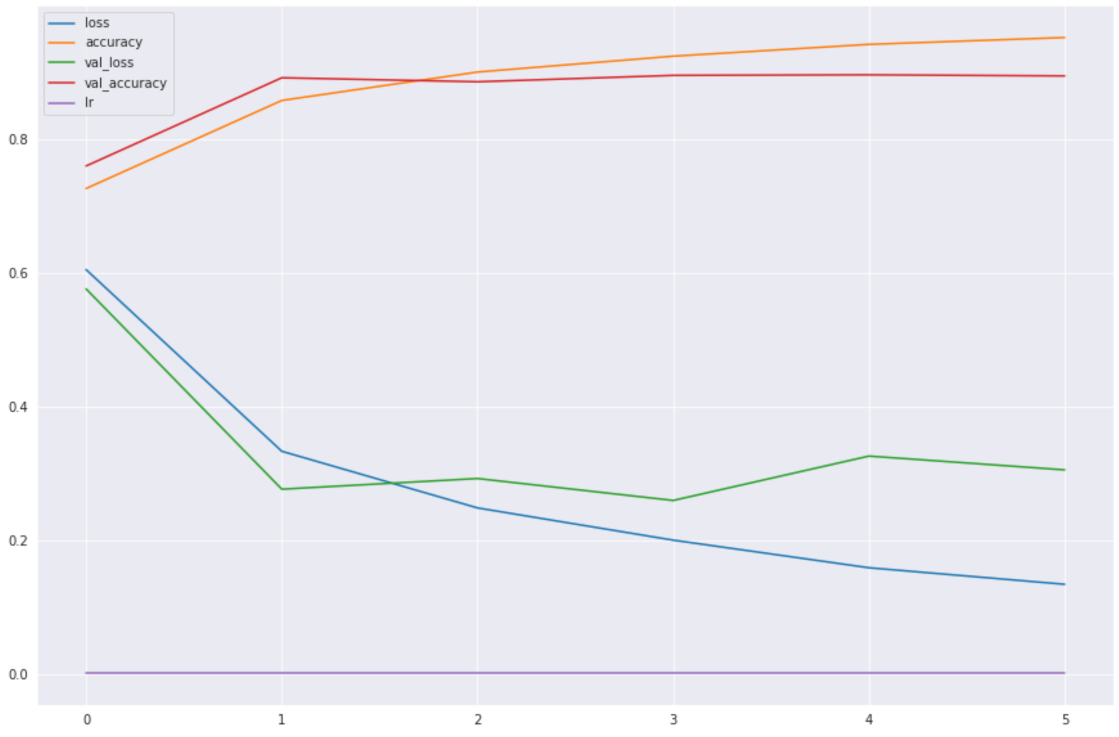
5 3. Performance Evaluation of the Training

[ ]:
```
pd.DataFrame(model.history.history)
```

[ ]:
```
       loss  accuracy  val_loss  val_accuracy     lr
0  0.604861  0.726267  0.576027        0.7600  0.001
1  0.332996  0.858044  0.276274        0.8920  0.001
2  0.248213  0.900578  0.292196        0.8860  0.001
3  0.199970  0.924311  0.259392        0.8956  0.001
4  0.158655  0.942000  0.325793        0.8964  0.001
5  0.133918  0.952222  0.305166        0.8948  0.001
```

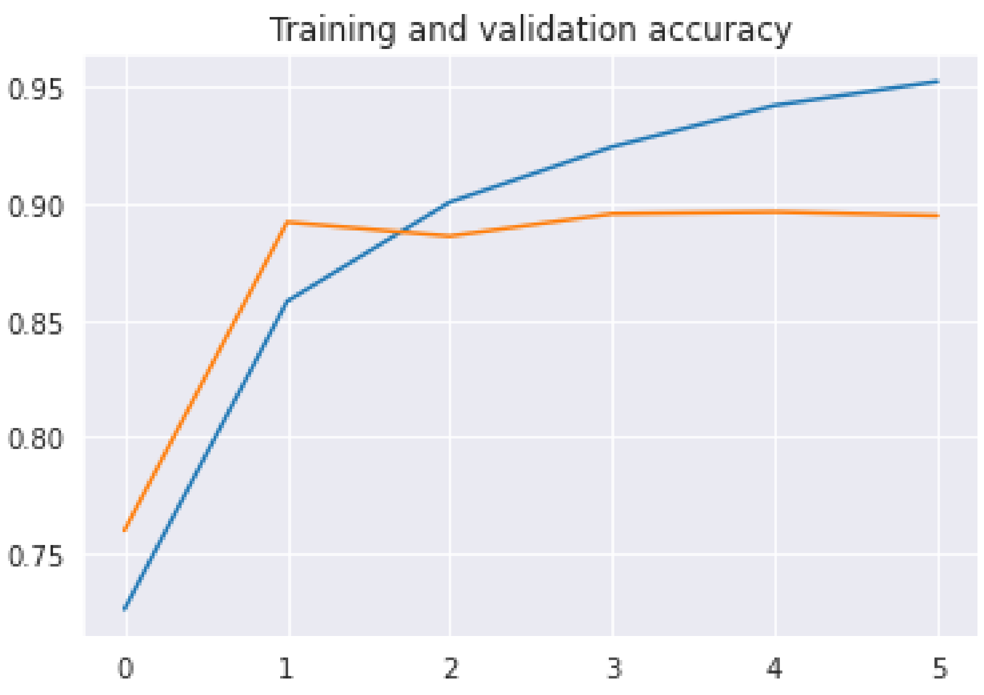
[ ]:
```
sns.set_style("darkgrid")
pd.DataFrame(model.history.history).plot(figsize=(15,10))
```

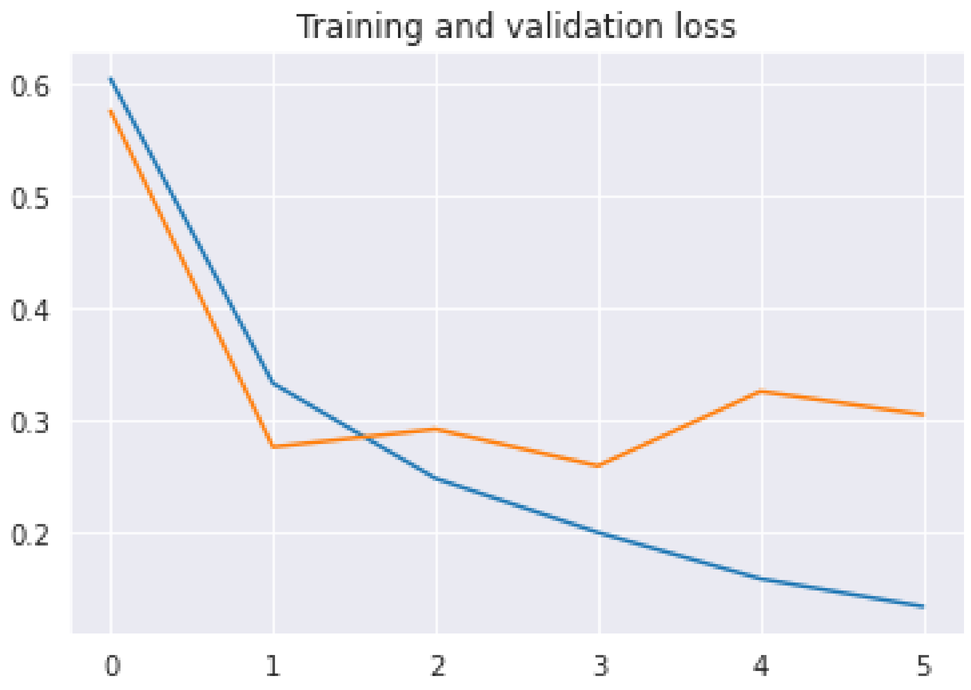
[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe2702d9bd0>
```



[ ]:
```
acc      = history.history['accuracy']
val_acc  = history.history['val_accuracy']
loss     = history.history['loss']
val_loss = history.history['val_loss']

epochs   = range(len(acc)) # Get number of epochs

#-----------------------------------------------------------
# Plot training and validation accuracy per epoch
#-----------------------------------------------------------
plt.plot  ( epochs,     acc )
plt.plot  ( epochs, val_acc )
plt.title ('Training and validation accuracy')
plt.figure()

#-----------------------------------------------------------
# Plot training and validation loss per epoch
#-----------------------------------------------------------
plt.plot(epochs, loss)
plt.plot(epochs, val_loss)
plt.title('Training and validation loss')
```

[ ]:
```
Text(0.5, 1.0, 'Training and validation loss')
```





6 4. Preparing Test Data and Submission

[ ]:
```
test_zip = '/content/drive/MyDrive/test.zip'

zip_ref = zipfile.ZipFile(test_zip, 'r')

zip_ref.extractall('/kaggle/working/')
zip_ref.close()
```

[ ]:
```
test_dir = '/kaggle/working/'
test_dir = os.path.join(test_dir, 'test')
test_img_names = [name.replace('test/', '') for name in zip_ref.namelist() if
len(name.replace('test/', '')) > 0]
```

[ ]:
```
test_img_names[:10]
```

[ ]:
```
['1.jpg',
 '10.jpg',
 '100.jpg',
 '1000.jpg',
 '10000.jpg',
 '10001.jpg',
 '10002.jpg',
 '10003.jpg',
 '10004.jpg',
 '10005.jpg']
```

[ ]:
```
len(test_img_names)
```

[ ]:
```
12500
```

[ ]:
```
test_df = pd.DataFrame({'Image': test_img_names})
test_df.head()
```

[ ]:
```
       Image
0      1.jpg
1     10.jpg
2    100.jpg
3   1000.jpg
4  10000.jpg
```

[ ]:
```
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_dataframe(test_df,
directory="/kaggle/working/test/", x_col="Image", y_col=None,
class_mode  = None, target_size=(150,150), shuffle = False, batch_size=20)
```

```
Found 12500 validated image filenames.
```

[ ]:
```
predictions = model.predict(test_generator,steps = np.ceil(12500/20))
%time predictions
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 7.87 µs
```

[ ]:
```
array([[9.8033786e-01],
       [4.6335398e-08],
       [5.9615129e-01],
       …,
       [9.9395639e-01],
       [2.4424034e-11],
       [9.9738710e-02]], dtype=float32)
```

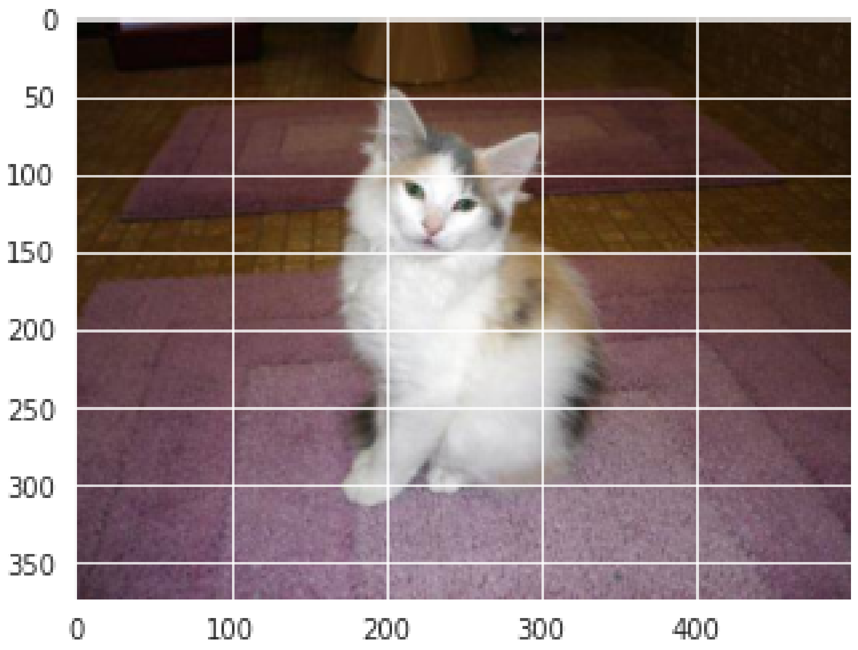
[ ]:
```
test_df["category"]=pd.DataFrame(predictions, columns=["category"])
test_df
```

[ ]:
```
           Image      category
0          1.jpg  9.803379e-01
1         10.jpg  4.633540e-08
2        100.jpg  5.961513e-01
3       1000.jpg  1.000000e+00
4      10000.jpg  5.500238e-01
…            …             …
12495   9995.jpg  4.905735e-01
```

```
12496    9996.jpg  9.999987e-01
12497    9997.jpg  9.939564e-01
12498    9998.jpg  2.442403e-11
12499    9999.jpg  9.973871e-02

[12500 rows x 2 columns]
```

[ ]:
```
plt.imshow(plt.imread(("/kaggle/working/test/10.jpg")))
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7fe2e4eb7190>
```



[ ]:
```
test_df=test_df.rename(columns={"category": "label"})
test_df
```

[ ]:
```
              Image         label
0             1.jpg  9.803379e-01
1            10.jpg  4.633540e-08
2           100.jpg  5.961513e-01
3          1000.jpg  1.000000e+00
4         10000.jpg  5.500238e-01
…               …             …
12495    9995.jpg  4.905735e-01
12496    9996.jpg  9.999987e-01
12497    9997.jpg  9.939564e-01
12498    9998.jpg  2.442403e-11
```

```
12499   9999.jpg  9.973871e-02

[12500 rows x 2 columns]
```

[ ]:
```
test_df.drop("Image", axis=1, inplace=True)
test_df["id"] = np.arange(len(predictions)) + 1
cols = test_df.columns.tolist()
cols = cols[-1:] + cols[:-1]
submission_df = test_df[cols]
```

[ ]:
```
submission_df
```

[ ]:
```
          id        label
0          1  9.803379e-01
1          2  4.633540e-08
2          3  5.961513e-01
3          4  1.000000e+00
4          5  5.500238e-01
…        …            …
12495  12496  4.905735e-01
12496  12497  9.999987e-01
12497  12498  9.939564e-01
12498  12499  2.442403e-11
12499  12500  9.973871e-02

[12500 rows x 2 columns]
```

[ ]:
```
submission_df.to_csv("/content/drive/MyDrive/cnn_submission.csv", index = False)
```

[ ]:
```
```

MSDS 422 Assignment 7 (CNN #2)

May 18, 2021

1 Assignment 7: Image Processing With a CNN

This assignment follows the same structure as Assignment 6. We will employ at least a 2x2 completely crossed experimental design. We will again use a simple training-and-test regimen. The factors in the design may include numbers of layers and/or nodes within layers, types of layers (convolutional or pooling), and/or other hyperparameters. You will utilize convolutional neural networks (CNNs) within Python TensorFlow.

This week, you will compete in the Dogs vs. Cats kernels Edition Kaggle.com competition, https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition (Links to an external site.). . (Links to an external site.)Specifically, you will build models using the training set to forecast the test set. The images are in .jpg format, so you will need to research how to handle that. You are required to submit no fewer than four models for evaluation by Kaggle.com, and you must provide your Kaggle.com scores and user ID for validation.

2 Management Problem

Assume that we are providing advice to a website provider who is looking for tools to automatically label images provided by end users. As we look across the factors in the study, making recommendations to management about image classification, we are most concerned about achieving the highest possible accuracy in image classification. That is, we should be willing to sacrifice training time for model accuracy. What type of machine learning model works best? If it is a convolutional neural network, what type of network should we use? Part of this recommendation may concern information about the initial images themselves (input data for the classification task). What types of images work best?

3 1. Preparation of Data and Exploratory Data Analysis

[ ]:
```
# This Python 3 environment comes with many helpful analytics libraries
# installed
# It is defined by the kaggle/python Docker image:
# https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

```
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter)
# will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that
# gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved
# outside of the current session
```

[ ]:
```
import os
import zipfile
import random
import tensorflow as tf
import shutil
import numpy as np
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.regularizers import l2
from shutil import copyfile
from os import getcwd
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive/')
```

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

[ ]:
```
!ls drive/MyDrive/
```

```
'AYUDH Table Presentation (1).ppt'
'AYUDH Table Presentation (1).ppt.gslides'
'AYUDH Table Schedule (Santa Fe - Toronto).gsheet'
'AYUDH Table Summer 2018 (Santa Fe - Toronto) (1).gsheet'
'AYUDH Table Summer 2018 (Santa Fe - Toronto) (1).xlsx'
'AYUDH Table Summer 2018 (Santa Fe - Toronto) (2).xlsx'
'AYUDH Table Summer 2018 (Santa Fe - Toronto).gsheet'
'AYUDH Table Summer 2018 (Santa Fe - Toronto).xlsx'
'Brochure (1).gdoc'
 Brochure.gdoc
'CECS Overview 7.20.2020.gdoc'
 cnn_submission.csv
'COVID-19 Exploratory Data Analysis in R.gdoc'
'COVID-19 Research Paper Analysis.docx'
'DASS 21 Data v1.gsheet'
```

```
'DASS 21 Data v1.xlsx'
'ENGL 001C- Essay 3 Proposal.gdoc'
'ENGL 001C Thesis Brainstorm.gdoc'
'English 001C Essay 2.docx'
'English 001C Essay 2.gdoc'
'English 001C- Final Notes.gdoc'
'English 1B- Essay 2 (final draft).docx'
'English 1B- Essay 2 (peer review).docx'
'English 4 Final Room Assignments F16.docx'
'English 4L Reading quizzes with answers.docx'
'English 4L Reading quizzes with answers.docx.gdoc'
'English 4L Reading quizzes with answers.gdoc'
'Math 144- Chapter 10 problems.gdoc'
'Math 144- Chapter 7 worked out problems.gdoc'
'Math 144 Chapter 8 problems.gdoc'
'Math 144 Worked out problems- Chapter 9.gdoc'
'Math 168 Project 1.gdoc'
'Microsoft Band Sizing Guide_EN.pdf'
'MSDS_422_Assignment_7_(CNN_1).ipynb'
'MSDS_422_Assignment_7_(CNN_2).ipynb'
'MSDS_422_Assignment_7_(CNN_3).ipynb'
'MSDS_422_Assignment_7_(CNN_4).ipynb'
'NY revised.xlsx'
'NY revised.xlsx.gsheet'
'Play Books Notes'
 Report.gdoc
'Safari - Jul 23, 2018 at 01:12.pdf'
 Schedule.gsheet
 test.zip
 train.zip
'Untitled document (1).gdoc'
'Untitled document (2).gdoc'
'Untitled document (3).gdoc'
'Untitled document (4).gdoc'
'Untitled document (5).gdoc'
'Untitled document.gdoc'
'Untitled spreadsheet (1).gsheet'
'Untitled spreadsheet (2).gsheet'
'Untitled spreadsheet.gsheet'
 Week3Lecture_default.mp4
```

[ ]:
```
local_zip = '/content/drive/MyDrive/train.zip'

zip_ref = zipfile.ZipFile(local_zip, 'r')

zip_ref.extractall('/kaggle/working/')
zip_ref.close()
```

```
[ ]: base_dir = '/kaggle/working/'
     train_dir = os.path.join(base_dir, 'train')
     train_img_names = os.listdir(train_dir)
```

```
[ ]: train_img_names[:10]
```

```
[ ]: ['dog.5577.jpg',
      'dog.8301.jpg',
      'cat.11000.jpg',
      'dog.8878.jpg',
      'dog.11820.jpg',
      'cat.9373.jpg',
      'cat.5203.jpg',
      'dog.831.jpg',
      'dog.2816.jpg',
      'cat.6415.jpg']
```

```
[ ]: print('total training images :', len(train_img_names ))
```

```
total training images : 25000
```

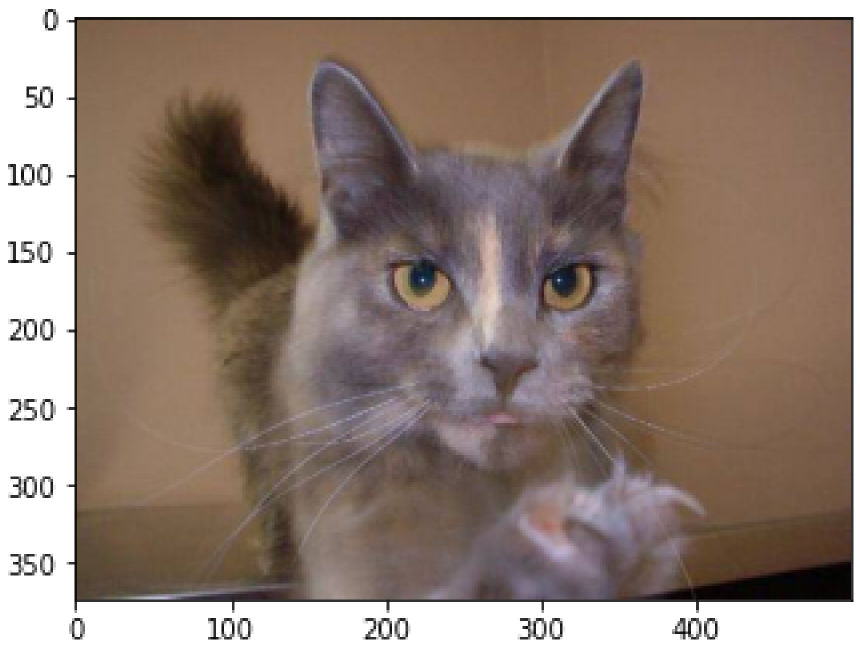
```
[ ]: categories= list()
     for image in train_img_names:
         category = image.split(".")[0]
         if category == "dog":
             categories.append("1")
         else:
             categories.append("0")

     df= pd.DataFrame({"Image":train_img_names, "Category": categories})
```

```
[ ]: df.head()
```

```
[ ]:            Image Category
     0   dog.5577.jpg        1
     1   dog.8301.jpg        1
     2  cat.11000.jpg        0
     3   dog.8878.jpg        1
     4  dog.11820.jpg        1
```

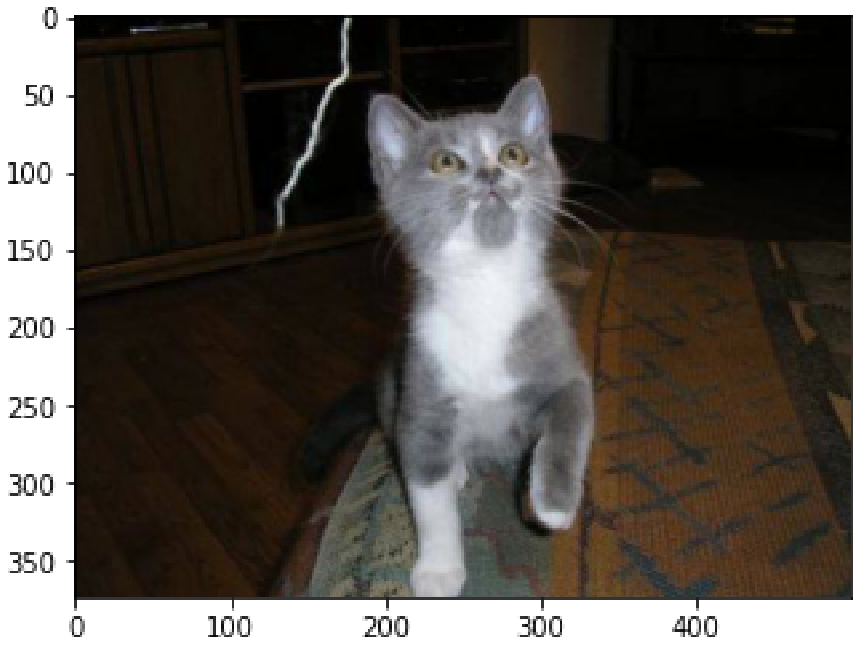
```
[ ]: import matplotlib.pyplot as plt
     import seaborn as sns
     import random

     sample = random.choice(train_img_names)
     plt.imshow(plt.imread(("/kaggle/working/train/"+sample)))
```

```
[ ]: <matplotlib.image.AxesImage at 0x7fe4bc26af90>
```



```
[ ]: sample = random.choice(train_img_names)
     plt.imshow(plt.imread(("/kaggle/working/train/"+sample)))
```

```
[ ]: <matplotlib.image.AxesImage at 0x7fe4bbd68450>
```



[ ]:
```
from sklearn.model_selection import train_test_split
train,validation= train_test_split(df, test_size=0.1)
train = train.reset_index(drop=True)
validation = validation.reset_index(drop=True)
```

[ ]:
```
train
```

[ ]:
```
               Image  Category
0       cat.9174.jpg         0
1       dog.4679.jpg         1
2       dog.6191.jpg         1
3       dog.6444.jpg         1
4       dog.3340.jpg         1
…                …       …
22495    dog.857.jpg         1
22496   dog.1442.jpg         1
22497    dog.930.jpg         1
22498  dog.10262.jpg         1
22499   cat.3277.jpg         0

[22500 rows x 2 columns]
```

[ ]:
```
validation
```

[ ]:
```
              Image  Category
0      cat.6091.jpg         0
1      dog.8826.jpg         1
2      dog.7299.jpg         1
3     cat.12162.jpg         0
4      cat.3450.jpg         0
…               …       …
2495  dog.11239.jpg         1
2496   dog.4800.jpg         1
2497   dog.6057.jpg         1
2498  cat.11457.jpg         0
2499   dog.2382.jpg         1

[2500 rows x 2 columns]
```

[ ]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255.
train_datagen = ImageDataGenerator( rescale = 1.0/255. )

# --------------------
# Flow training images in batches of 20 using train_datagen generator
# --------------------
train_generator = train_datagen.flow_from_dataframe(train,
directory="/kaggle/working/train", x_col='Image', y_col='Category',
batch_size=20, class_mode='binary', target_size=(150, 150))
```

```
Found 22500 validated image filenames belonging to 2 classes.
```

[ ]:
```
validation_datagen  = ImageDataGenerator( rescale = 1.0/255.)
validation_generator =  validation_datagen.flow_from_dataframe(validation,
directory="/kaggle/working/train", x_col='Image', y_col='Category',
batch_size=20, class_mode  = 'binary', target_size = (150, 150))
```

```
Found 2500 validated image filenames belonging to 2 classes.
```

4 2. Building a Small Model from Scratch

[ ]:
```
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPool2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential()

model.add(Conv2D(filters=64, kernel_size=(3,3),activation="relu",
input_shape=(150,150,3)))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=3, strides=2))

model.add(Conv2D(filters=128, kernel_size=3,activation="relu"))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=3, strides=2))

model.add(Conv2D(filters=256, kernel_size=3, activation="relu"))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=3, strides=2))

model.add(Conv2D(filters=512, kernel_size=3, activation="relu"))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=3, strides=2))

model.add(Conv2D(filters=512, kernel_size=3, activation="relu"))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=3, strides=2))
model.add(Flatten())
model.add(Dense(units=512, activation="relu"))
model.add(Dropout(0.1))
model.add(Dense(units=1, activation="sigmoid"))
```

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

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 64)      1792
_________________________________________________________________
batch_normalization (BatchNo (None, 148, 148, 64)      256
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 73, 73, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 71, 71, 128)       73856
_________________________________________________________________
batch_normalization_1 (Batch (None, 71, 71, 128)       512
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 35, 35, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 33, 33, 256)       295168
_________________________________________________________________
batch_normalization_2 (Batch (None, 33, 33, 256)       1024
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 256)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 512)       1180160
_________________________________________________________________
batch_normalization_3 (Batch (None, 14, 14, 512)       2048
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 512)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 4, 4, 512)         2359808
_________________________________________________________________
batch_normalization_4 (Batch (None, 4, 4, 512)         2048
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 1, 1, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 512)               262656
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
```

```
=================================================================
Total params: 4,179,841
Trainable params: 4,176,897
Non-trainable params: 2,944
_________________________________________________________________
```

[ ]:
```
import tensorflow as tf
model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=0.001),
              loss='binary_crossentropy', metrics=['accuracy'])
```

[ ]:
```
callback=EarlyStopping(monitor="val_loss", patience=2)
callback_lr = tensorflow.keras.callbacks.ReduceLROnPlateau(
    monitor='val_accuracy', patience=2, factor=0.5, min_lr=0.00001)
```

[ ]:
```
history=model.fit(train_generator, validation_data=validation_generator,
                  epochs=8, callbacks=[callback,callback_lr])
```

```
Epoch 1/8
1125/1125 [==============================] - 89s 63ms/step - loss: 0.8071 -
accuracy: 0.6342 - val_loss: 0.5920 - val_accuracy: 0.6984
Epoch 2/8
1125/1125 [==============================] - 71s 63ms/step - loss: 0.3492 -
accuracy: 0.8469 - val_loss: 0.4403 - val_accuracy: 0.8064
Epoch 3/8
1125/1125 [==============================] - 71s 63ms/step - loss: 0.2587 -
accuracy: 0.8943 - val_loss: 0.4275 - val_accuracy: 0.8524
Epoch 4/8
1125/1125 [==============================] - 71s 63ms/step - loss: 0.2048 -
accuracy: 0.9195 - val_loss: 0.4230 - val_accuracy: 0.8028
Epoch 5/8
1125/1125 [==============================] - 70s 63ms/step - loss: 0.1703 -
accuracy: 0.9344 - val_loss: 0.3861 - val_accuracy: 0.8416
Epoch 6/8
1125/1125 [==============================] - 71s 63ms/step - loss: 0.1014 -
accuracy: 0.9609 - val_loss: 0.2200 - val_accuracy: 0.9244
Epoch 7/8
1125/1125 [==============================] - 71s 63ms/step - loss: 0.0714 -
accuracy: 0.9728 - val_loss: 0.2730 - val_accuracy: 0.9156
Epoch 8/8
1125/1125 [==============================] - 71s 63ms/step - loss: 0.0551 -
accuracy: 0.9817 - val_loss: 0.2668 - val_accuracy: 0.9224
```

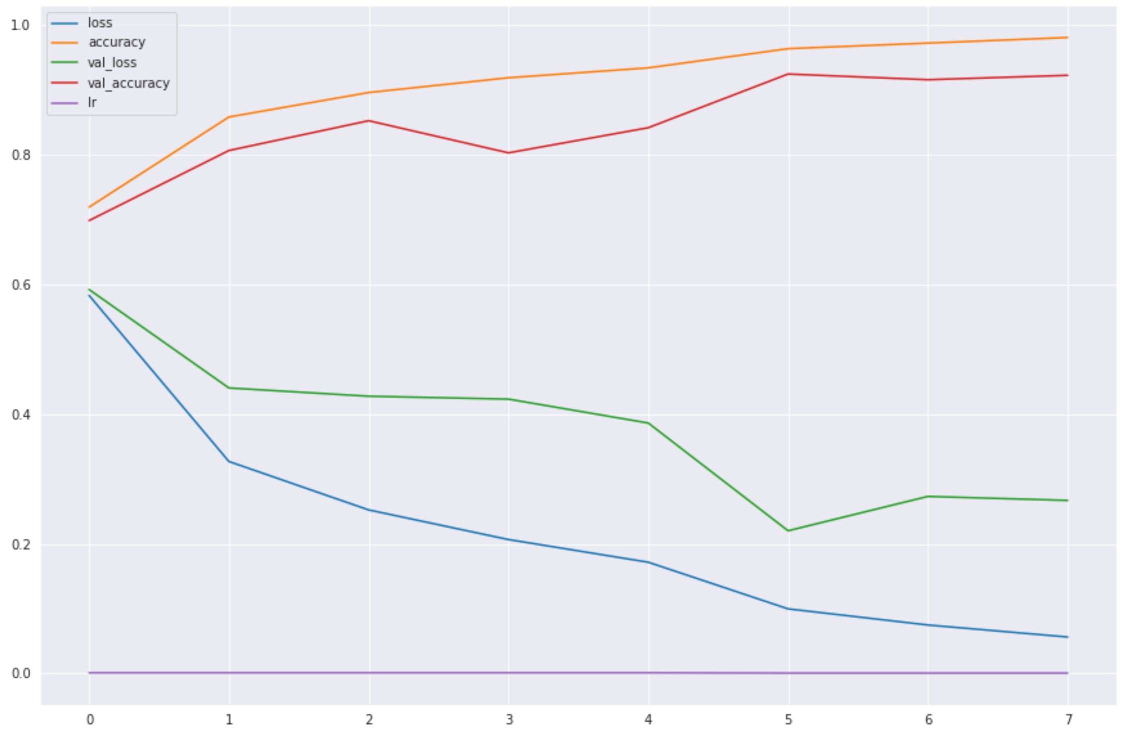
5 3. Performance Evaluation of the Training

[ ]:
```
pd.DataFrame(model.history.history)
```

```
[ ]:       loss  accuracy  val_loss  val_accuracy      lr
     0  0.582965  0.719111  0.592023        0.6984  0.0010
     1  0.326871  0.858089  0.440262        0.8064  0.0010
     2  0.252144  0.895867  0.427470        0.8524  0.0010
     3  0.206452  0.918622  0.423010        0.8028  0.0010
     4  0.171502  0.933867  0.386144        0.8416  0.0010
     5  0.099578  0.963511  0.220028        0.9244  0.0005
     6  0.074688  0.972089  0.272951        0.9156  0.0005
     7  0.056171  0.980667  0.266839        0.9224  0.0005
```

```
[ ]: sns.set_style("darkgrid")
     pd.DataFrame(model.history.history).plot(figsize=(15,10))
```

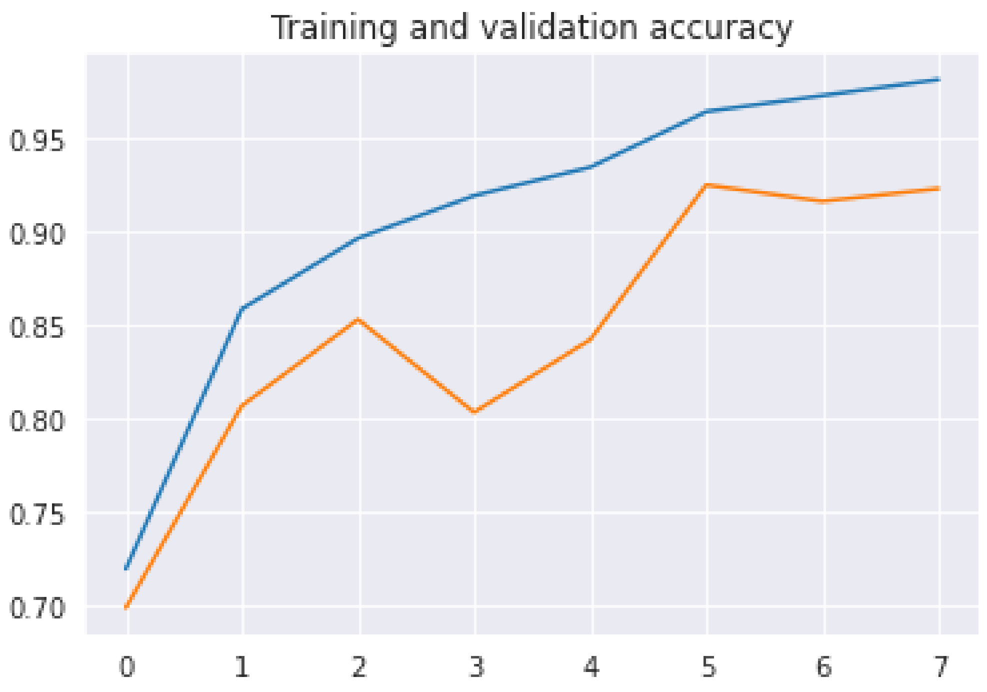
```
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe44a6beb50>
```

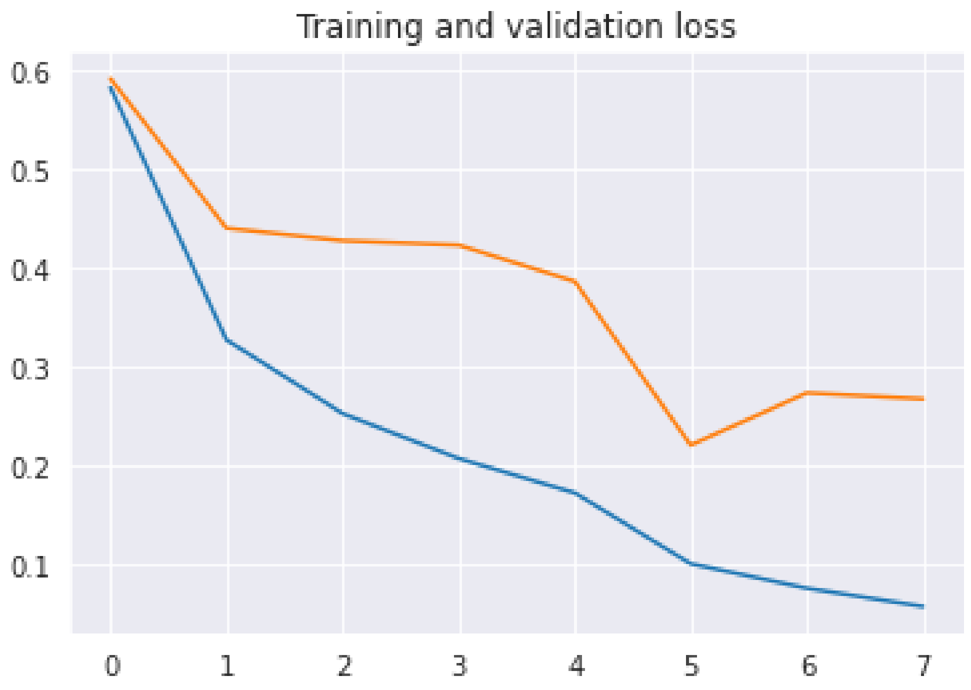


```
[ ]: acc      = history.history['accuracy']
     val_acc  = history.history['val_accuracy']
     loss     = history.history['loss']
     val_loss = history.history['val_loss']

     epochs   = range(len(acc)) # Get number of epochs

     #-----------------------------------------------
```

```
# Plot training and validation accuracy per epoch
#------------------------------------------------
plt.plot  ( epochs,     acc )
plt.plot  ( epochs, val_acc )
plt.title ('Training and validation accuracy')
plt.figure()

#------------------------------------------------
# Plot training and validation loss per epoch
#------------------------------------------------
plt.plot(epochs, loss)
plt.plot(epochs, val_loss)
plt.title('Training and validation loss')
```

```
[ ]: Text(0.5, 1.0, 'Training and validation loss')
```





6 4. Preparing Test Data and Submission

[ ]:
```
test_zip = '/content/drive/MyDrive/test.zip'

zip_ref = zipfile.ZipFile(test_zip, 'r')

zip_ref.extractall('/kaggle/working/')
zip_ref.close()
```

[ ]:
```
test_dir = '/kaggle/working/'
test_dir = os.path.join(test_dir, 'test')
test_img_names = [name.replace('test/', '') for name in zip_ref.namelist() if
len(name.replace('test/', '')) > 0]
```

[ ]:
```
test_img_names[:10]
```

[ ]:
```
['1.jpg',
 '10.jpg',
 '100.jpg',
 '1000.jpg',
 '10000.jpg',
 '10001.jpg',
 '10002.jpg',
 '10003.jpg',
 '10004.jpg',
 '10005.jpg']
```

[ ]:
```
len(test_img_names)
```

[ ]:
```
12500
```

[ ]:
```
test_df = pd.DataFrame({'Image': test_img_names})
test_df.head()
```

[ ]:
```
       Image
0      1.jpg
1     10.jpg
2    100.jpg
3   1000.jpg
4  10000.jpg
```

[ ]:
```
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_dataframe(test_df,
directory="/kaggle/working/test/", x_col="Image", y_col=None,
class_mode  = None, target_size=(150,150), shuffle = False, batch_size=20)
```

```
Found 12500 validated image filenames.
```

[ ]:
```
predictions = model.predict(test_generator,steps = np.ceil(12500/20))
%time predictions
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 6.44 µs
```

[ ]:
```
array([[9.9999976e-01],
       [8.3923250e-11],
       [9.9603879e-01],
       …,
       [9.9924845e-01],
       [9.6556281e-13],
       [7.0483126e-02]], dtype=float32)
```

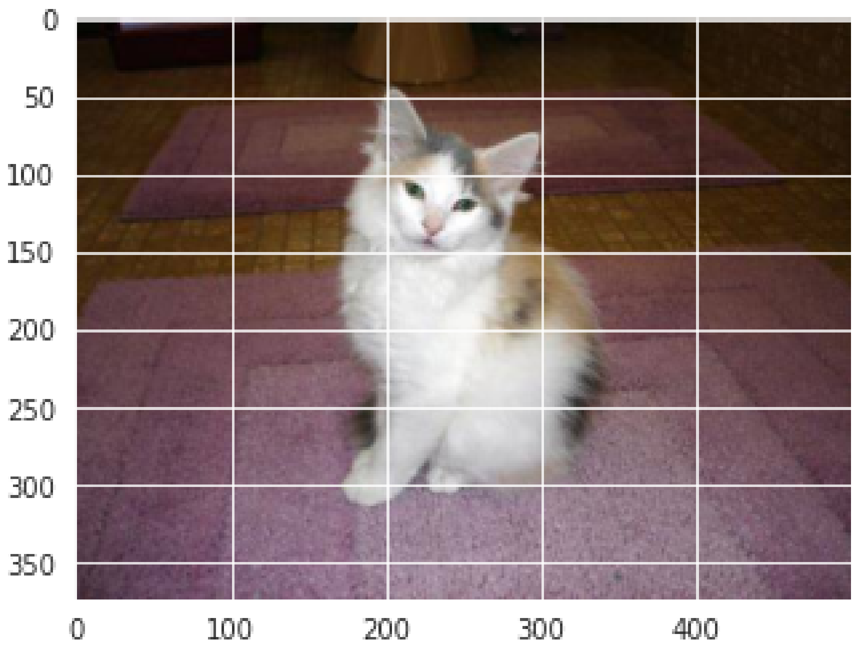
[ ]:
```
test_df["category"]=pd.DataFrame(predictions, columns=["category"])
test_df
```

[ ]:
```
           Image      category
0          1.jpg  9.999998e-01
1         10.jpg  8.392325e-11
2        100.jpg  9.960388e-01
3       1000.jpg  9.999999e-01
4      10000.jpg  9.595186e-01
…            …             …
12495   9995.jpg  1.549663e-02
```

```
12496   9996.jpg  2.737038e-02
12497   9997.jpg  9.992484e-01
12498   9998.jpg  9.655628e-13
12499   9999.jpg  7.048313e-02

[12500 rows x 2 columns]
```

[ ]:
```
plt.imshow(plt.imread(("/kaggle/working/test/10.jpg")))
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7fe4c3dde610>
```



[ ]:
```
test_df=test_df.rename(columns={"category": "label"})
test_df
```

[ ]:
```
            Image         label
0           1.jpg  9.999998e-01
1          10.jpg  8.392325e-11
2         100.jpg  9.960388e-01
3        1000.jpg  9.999999e-01
4       10000.jpg  9.595186e-01
…             …             …
12495    9995.jpg  1.549663e-02
12496    9996.jpg  2.737038e-02
12497    9997.jpg  9.992484e-01
12498    9998.jpg  9.655628e-13
```

```
12499   9999.jpg  7.048313e-02

[12500 rows x 2 columns]
```

[ ]:
```
test_df.drop("Image", axis=1, inplace=True)
test_df["id"] = np.arange(len(predictions)) + 1
cols = test_df.columns.tolist()
cols = cols[-1:] + cols[:-1]
submission_df = test_df[cols]
```

[ ]:
```
submission_df
```

[ ]:
```
          id         label
0          1  9.999998e-01
1          2  8.392325e-11
2          3  9.960388e-01
3          4  9.999999e-01
4          5  9.595186e-01
…        …             …
12495  12496  1.549663e-02
12496  12497  2.737038e-02
12497  12498  9.992484e-01
12498  12499  9.655628e-13
12499  12500  7.048313e-02

[12500 rows x 2 columns]
```

[ ]:
```
submission_df.to_csv("/content/drive/MyDrive/cnn_model2_submission.csv", index␣
 ↪= False)
```

[ ]:
```

```

MSDS 422 Assignment 7 (CNN #3)

May 18, 2021

1 Assignment 7: Image Processing With a CNN

This assignment follows the same structure as Assignment 6. We will employ at least a 2x2 completely crossed experimental design. We will again use a simple training-and-test regimen. The factors in the design may include numbers of layers and/or nodes within layers, types of layers (convolutional or pooling), and/or other hyperparameters. You will utilize convolutional neural networks (CNNs) within Python TensorFlow.

This week, you will compete in the Dogs vs. Cats kernels Edition Kaggle.com competition, https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition (Links to an external site.). . (Links to an external site.)Specifically, you will build models using the training set to forecast the test set. The images are in .jpg format, so you will need to research how to handle that. You are required to submit no fewer than four models for evaluation by Kaggle.com, and you must provide your Kaggle.com scores and user ID for validation.

2 Management Problem

Assume that we are providing advice to a website provider who is looking for tools to automatically label images provided by end users. As we look across the factors in the study, making recommendations to management about image classification, we are most concerned about achieving the highest possible accuracy in image classification. That is, we should be willing to sacrifice training time for model accuracy. What type of machine learning model works best? If it is a convolutional neural network, what type of network should we use? Part of this recommendation may concern information about the initial images themselves (input data for the classification task). What types of images work best?

3 1. Preparation of Data and Exploratory Data Analysis

[ ]:
```
# This Python 3 environment comes with many helpful analytics libraries
# installed
# It is defined by the kaggle/python Docker image:
# https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

```
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter)
# will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that
# gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved
# outside of the current session
```

[ ]:
```
import os
import zipfile
import random
import tensorflow as tf
import shutil
import numpy as np
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.regularizers import l2
from shutil import copyfile
from os import getcwd
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive/')
```

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

[ ]:
```
!ls drive/MyDrive/
```

```
'AYUDH Table Presentation (1).ppt'
'AYUDH Table Presentation (1).ppt.gslides'
'AYUDH Table Schedule (Santa Fe - Toronto).gsheet'
'AYUDH Table Summer 2018 (Santa Fe - Toronto) (1).gsheet'
'AYUDH Table Summer 2018 (Santa Fe - Toronto) (1).xlsx'
'AYUDH Table Summer 2018 (Santa Fe - Toronto) (2).xlsx'
'AYUDH Table Summer 2018 (Santa Fe - Toronto).gsheet'
'AYUDH Table Summer 2018 (Santa Fe - Toronto).xlsx'
'Brochure (1).gdoc'
 Brochure.gdoc
'CECS Overview 7.20.2020.gdoc'
 cnn_model2_submission.csv
 cnn_submission.csv
'COVID-19 Exploratory Data Analysis in R.gdoc'
'COVID-19 Research Paper Analysis.docx'
```

```
'DASS 21 Data v1.gsheet'
'DASS 21 Data v1.xlsx'
'ENGL 001C- Essay 3 Proposal.gdoc'
'ENGL 001C Thesis Brainstorm.gdoc'
'English 001C Essay 2.docx'
'English 001C Essay 2.gdoc'
'English 001C- Final Notes.gdoc'
'English 1B- Essay 2 (final draft).docx'
'English 1B- Essay 2 (peer review).docx'
'English 4 Final Room Assignments F16.docx'
'English 4L Reading quizzes with answers.docx'
'English 4L Reading quizzes with answers.docx.gdoc'
'English 4L Reading quizzes with answers.gdoc'
'Math 144- Chapter 10 problems.gdoc'
'Math 144- Chapter 7 worked out problems.gdoc'
'Math 144 Chapter 8 problems.gdoc'
'Math 144 Worked out problems- Chapter 9.gdoc'
'Math 168 Project 1.gdoc'
'Microsoft Band Sizing Guide_EN.pdf'
'MSDS 422 Assignment 7 (CNN #1).ipynb'
'MSDS 422 Assignment 7 (CNN #2).ipynb'
'MSDS 422 Assignment 7 (CNN #3).ipynb'
'MSDS 422 Assignment 7 (CNN #4).ipynb'
'NY revised.xlsx'
'NY revised.xlsx.gsheet'
'Play Books Notes'
 Report.gdoc
'Safari - Jul 23, 2018 at 01:12.pdf'
 Schedule.gsheet
 test.zip
 train.zip
'Untitled document (1).gdoc'
'Untitled document (2).gdoc'
'Untitled document (3).gdoc'
'Untitled document (4).gdoc'
'Untitled document (5).gdoc'
'Untitled document.gdoc'
'Untitled spreadsheet (1).gsheet'
'Untitled spreadsheet (2).gsheet'
'Untitled spreadsheet.gsheet'
 Week3Lecture_default.mp4
```

[ ]:
```
local_zip = '/content/drive/MyDrive/train.zip'

zip_ref = zipfile.ZipFile(local_zip, 'r')

zip_ref.extractall('/kaggle/working/')
zip_ref.close()
```

[ ]:
```
base_dir = '/kaggle/working/'
train_dir = os.path.join(base_dir, 'train')
train_img_names = os.listdir(train_dir)
```

[ ]:
```
train_img_names[:10]
```

[ ]:
```
['dog.5577.jpg',
 'dog.8301.jpg',
 'cat.11000.jpg',
 'dog.8878.jpg',
 'dog.11820.jpg',
 'cat.9373.jpg',
 'cat.5203.jpg',
 'dog.831.jpg',
 'dog.2816.jpg',
 'cat.6415.jpg']
```

[ ]:
```
print('total training images :', len(train_img_names ))
```

```
total training images : 25000
```

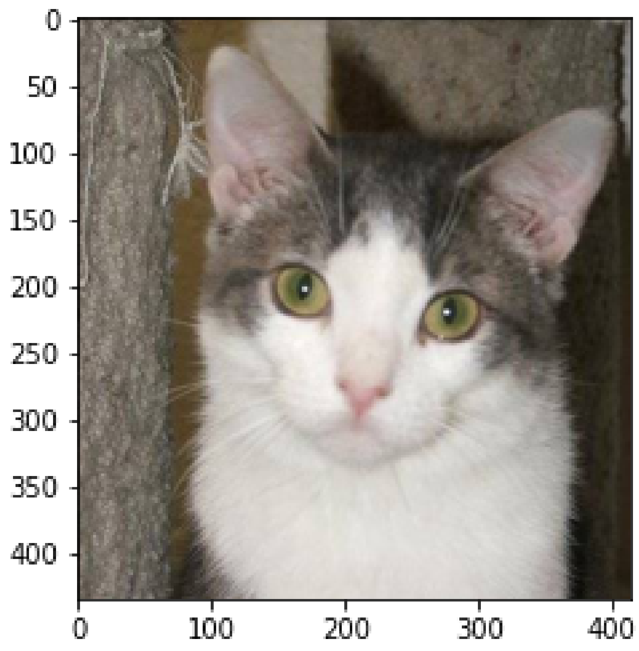
[ ]:
```
categories= list()
for image in train_img_names:
    category = image.split(".")[0]
    if category == "dog":
        categories.append("1")
    else:
        categories.append("0")

df= pd.DataFrame({"Image":train_img_names, "Category": categories})
```

[ ]:
```
df.head()
```

[ ]:
```
           Image Category
0   dog.5577.jpg        1
1   dog.8301.jpg        1
2  cat.11000.jpg        0
3   dog.8878.jpg        1
4  dog.11820.jpg        1
```

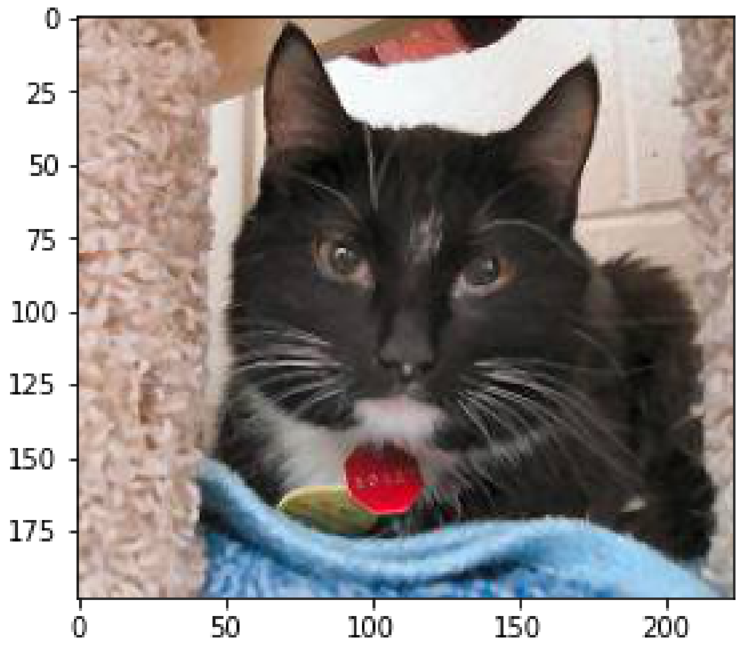
[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import random

sample = random.choice(train_img_names)
plt.imshow(plt.imread(("/kaggle/working/train/"+sample)))
```

```
[ ]: <matplotlib.image.AxesImage at 0x7f32795904d0>
```



```
[ ]: sample = random.choice(train_img_names)
     plt.imshow(plt.imread(("/kaggle/working/train/"+sample)))
```

```
[ ]: <matplotlib.image.AxesImage at 0x7f32788892d0>
```



```
[ ]: from sklearn.model_selection import train_test_split
     train,validation= train_test_split(df, test_size=0.1)
     train = train.reset_index(drop=True)
     validation = validation.reset_index(drop=True)
```

```
[ ]: train
```

```
[ ]:                Image  Category
     0        dog.655.jpg         1
     1       cat.2070.jpg         0
     2       dog.4610.jpg         1
     3      cat.10252.jpg         0
     4       dog.6897.jpg         1
     …                  …       …
     22495   dog.6908.jpg         1
     22496   dog.5101.jpg         1
     22497   dog.7393.jpg         1
     22498   cat.4236.jpg         0
     22499   cat.7506.jpg         0

     [22500 rows x 2 columns]
```

```
[ ]: validation
```

```
[ ]:               Image  Category
     0      cat.4299.jpg         0
     1     dog.11267.jpg         1
     2      cat.5335.jpg         0
     3      cat.4969.jpg         0
     4      dog.4580.jpg         1
     …                 …       …
     2495   cat.4826.jpg         0
     2496  dog.10144.jpg         1
     2497   dog.3297.jpg         1
     2498   dog.3809.jpg         1
     2499  dog.10822.jpg         1

     [2500 rows x 2 columns]
```

```
[ ]: from tensorflow.keras.preprocessing.image import ImageDataGenerator

     # All images will be rescaled by 1./255.
     train_datagen = ImageDataGenerator( rescale = 1.0/255. )

     # --------------------
     # Flow training images in batches of 20 using train_datagen generator
     # --------------------
     train_generator = train_datagen.flow_from_dataframe(train,
     directory="/kaggle/working/train", x_col='Image', y_col='Category',
     batch_size=20, class_mode='binary', target_size=(150, 150))
```

```
Found 22500 validated image filenames belonging to 2 classes.
```

[ ]:
```
validation_datagen  = ImageDataGenerator( rescale = 1.0/255.)
validation_generator =  validation_datagen.flow_from_dataframe(validation,
directory="/kaggle/working/train", x_col='Image', y_col='Category',
batch_size=20, class_mode  = 'binary', target_size = (150, 150))
```

```
Found 2500 validated image filenames belonging to 2 classes.
```

4 2. Building a Small Model from Scratch

[ ]:
```
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPool2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3,3),activation="relu",
input_shape=(150,150,3)))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=2, strides=2))

model.add(Conv2D(filters=64, kernel_size=3,activation="relu"))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=2, strides=2))

model.add(Conv2D(filters=128, kernel_size=3, activation="relu"))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=2, strides=2))

model.add(Flatten())
model.add(Dense(units=512, activation="relu"))
model.add(Dropout(0.2))
model.add(Dense(units=1, activation="sigmoid"))
```

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

```
Model: "sequential"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
batch_normalization (BatchNo (None, 148, 148, 32)      128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
batch_normalization_1 (Batch (None, 72, 72, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
batch_normalization_2 (Batch (None, 34, 34, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 36992)             0
_________________________________________________________________
dense (Dense)                (None, 512)               18940416
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 19,035,073
Trainable params: 19,034,625
Non-trainable params: 448
_________________________________________________________________
```

[ ]:
```
import tensorflow as tf
model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=0.001),
loss='binary_crossentropy', metrics=['accuracy'])
```

[ ]:
```
callback=EarlyStopping(monitor="val_loss", patience=2)
callback_lr = tensorflow.keras.callbacks.ReduceLROnPlateau(
monitor='val_accuracy', patience=2, factor=0.5, min_lr=0.00001)
```

[ ]:
```
history=model.fit(train_generator, validation_data=validation_generator,
epochs=4, callbacks=[callback,callback_lr])
```

```
Epoch 1/4
1125/1125 [==============================] - 91s 66ms/step - loss: 2.8848 -
accuracy: 0.6420 - val_loss: 1.0420 - val_accuracy: 0.6712
```

```
Epoch 2/4
1125/1125 [==============================] - 73s 65ms/step - loss: 0.5027 -
accuracy: 0.7858 - val_loss: 1.3414 - val_accuracy: 0.6636
Epoch 3/4
1125/1125 [==============================] - 73s 65ms/step - loss: 0.4182 -
accuracy: 0.8310 - val_loss: 1.0189 - val_accuracy: 0.7652
Epoch 4/4
1125/1125 [==============================] - 72s 64ms/step - loss: 0.3563 -
accuracy: 0.8646 - val_loss: 2.1990 - val_accuracy: 0.5644
```

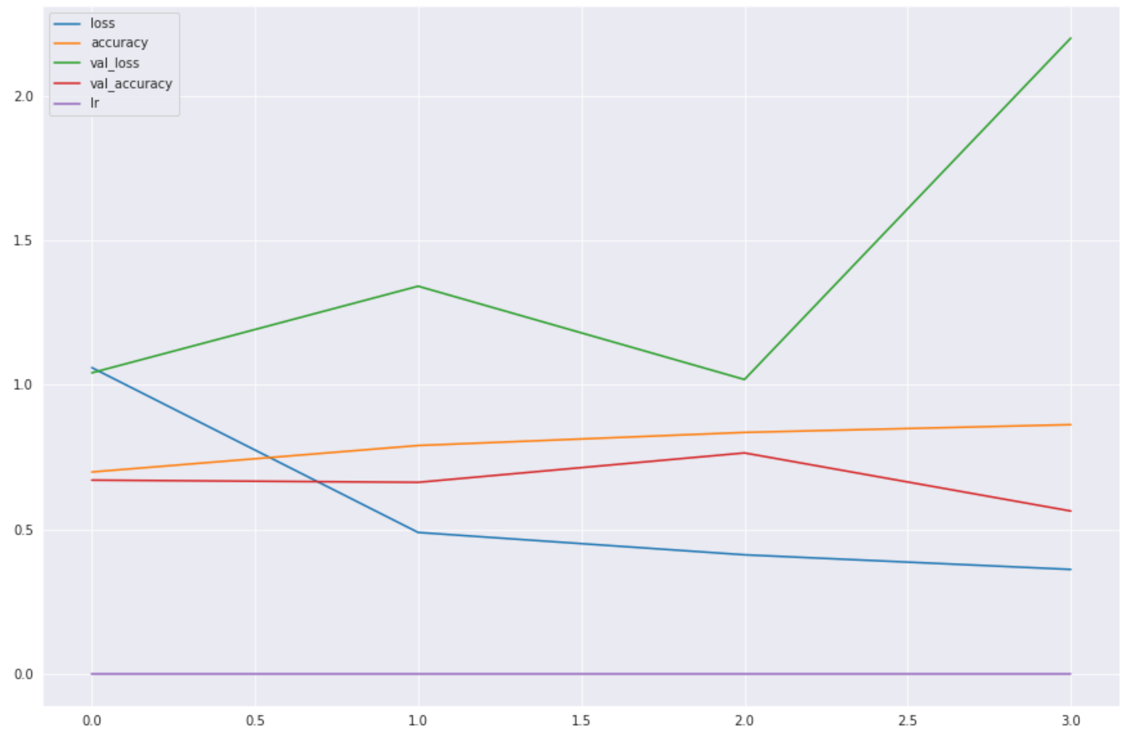
5 3. Performance Evaluation of the Training

[ ]:
```
pd.DataFrame(model.history.history)
```

[ ]:
```
       loss  accuracy  val_loss  val_accuracy     lr
0  1.060051  0.699289  1.042012        0.6712  0.001
1  0.490351  0.790800  1.341439        0.6636  0.001
2  0.413081  0.836089  1.018924        0.7652  0.001
3  0.362603  0.862889  2.198952        0.5644  0.001
```

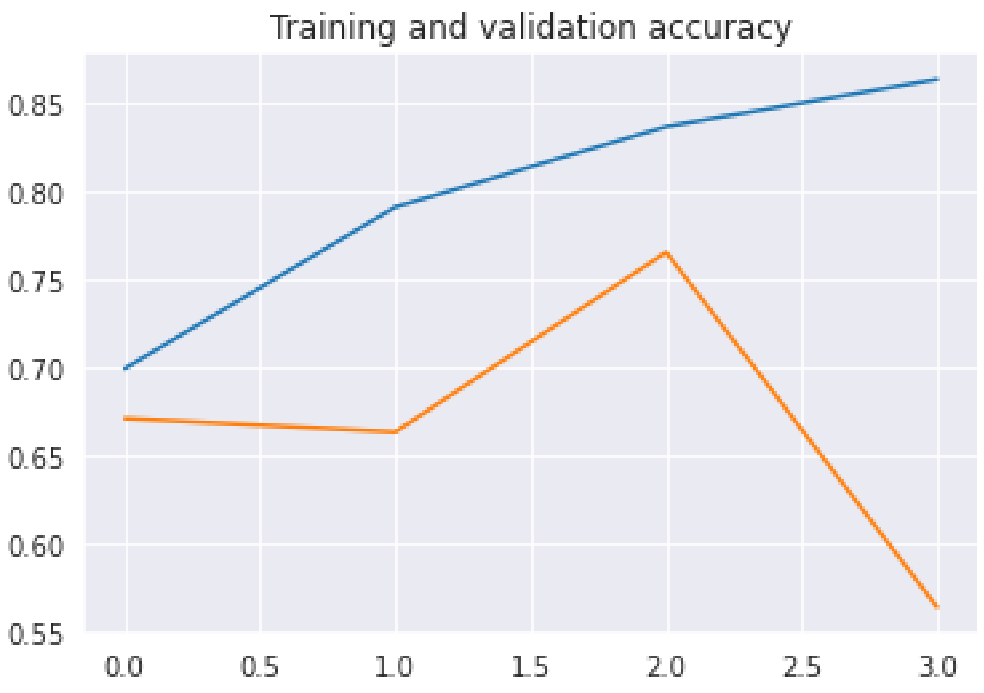
[ ]:
```
sns.set_style("darkgrid")
pd.DataFrame(model.history.history).plot(figsize=(15,10))
```

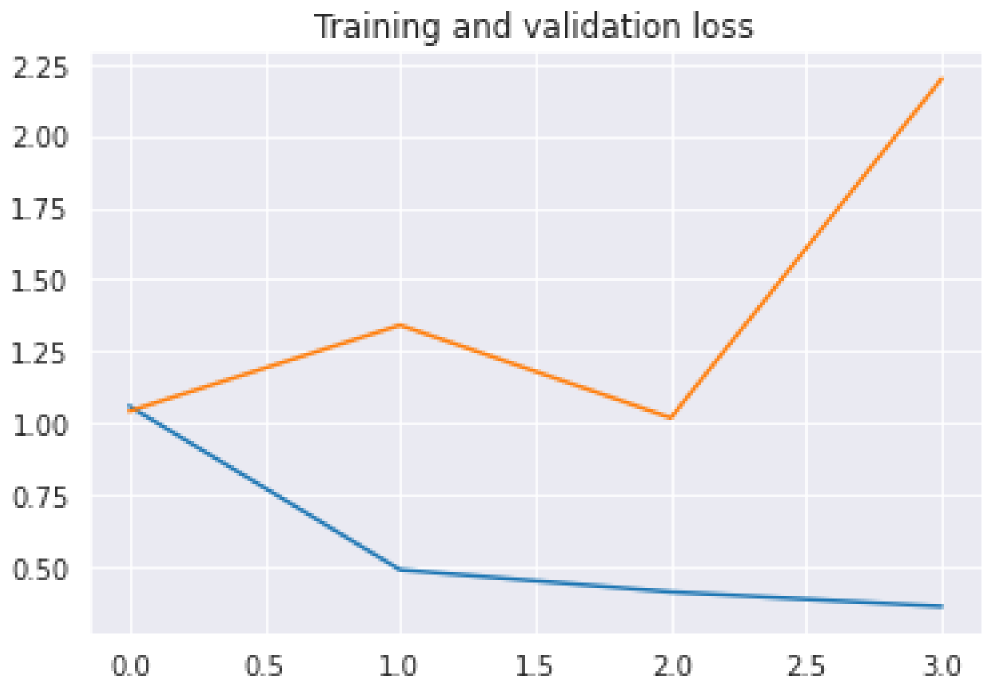
[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f32260ffdd0>
```



```
[ ]: acc      = history.history['accuracy']
     val_acc  = history.history['val_accuracy']
     loss     = history.history['loss']
     val_loss = history.history['val_loss']

     epochs   = range(len(acc)) # Get number of epochs

     #------------------------------------------------
     # Plot training and validation accuracy per epoch
     #------------------------------------------------
     plt.plot  ( epochs,     acc )
     plt.plot  ( epochs, val_acc )
     plt.title ('Training and validation accuracy')
     plt.figure()

     #------------------------------------------------
     # Plot training and validation loss per epoch
     #------------------------------------------------
     plt.plot(epochs, loss)
     plt.plot(epochs, val_loss)
     plt.title('Training and validation loss')
```

```
[ ]: Text(0.5, 1.0, 'Training and validation loss')
```





6 4. Preparing Test Data and Submission

```
[ ]: test_zip = '/content/drive/MyDrive/test.zip'

     zip_ref = zipfile.ZipFile(test_zip, 'r')

     zip_ref.extractall('/kaggle/working/')
     zip_ref.close()
```

```
[ ]: test_dir = '/kaggle/working/'
     test_dir = os.path.join(test_dir, 'test')
     test_img_names = [name.replace('test/', '') for name in zip_ref.namelist() if
     len(name.replace('test/', '')) > 0]
```

```
[ ]: test_img_names[:10]
```

```
[ ]: ['1.jpg',
      '10.jpg',
      '100.jpg',
      '1000.jpg',
      '10000.jpg',
      '10001.jpg',
      '10002.jpg',
      '10003.jpg',
      '10004.jpg',
      '10005.jpg']
```

[ ]:
```
len(test_img_names)
```

[ ]:
```
12500
```

[ ]:
```
test_df = pd.DataFrame({'Image': test_img_names})
test_df.head()
```

[ ]:
```
       Image
0      1.jpg
1     10.jpg
2    100.jpg
3   1000.jpg
4  10000.jpg
```

[ ]:
```
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_dataframe(test_df,
directory="/kaggle/working/test/", x_col="Image", y_col=None,
class_mode  = None, target_size=(150,150), shuffle = False, batch_size=20)
```

```
Found 12500 validated image filenames.
```

[ ]:
```
predictions = model.predict(test_generator,steps = np.ceil(12500/20))
%time predictions
```

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 7.63 µs
```

[ ]:
```
array([[1.1963607e-03],
       [3.9878934e-05],
       [4.8457336e-02],
       …,
       [2.7554328e-04],
       [1.3181327e-05],
       [3.9139090e-04]], dtype=float32)
```

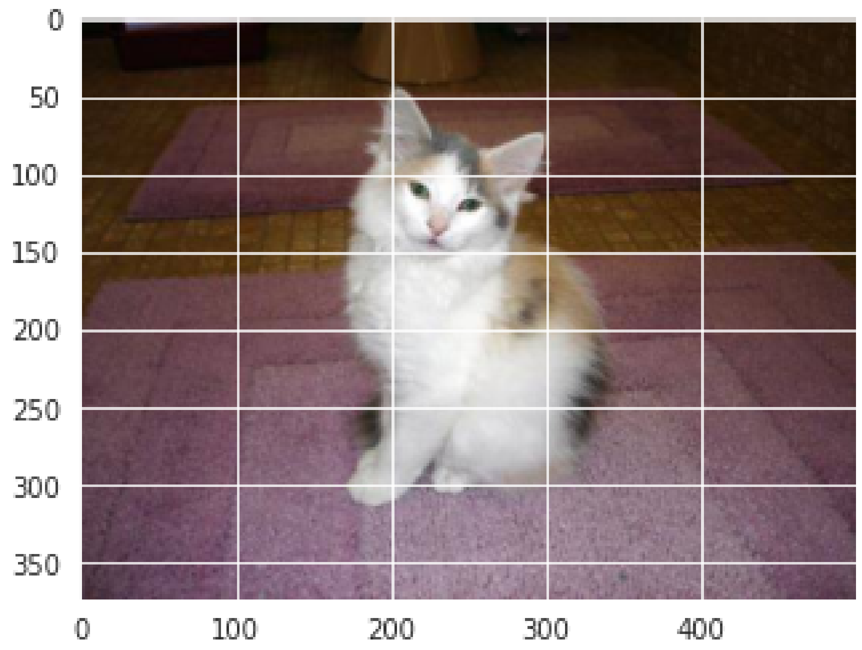
[ ]:
```
test_df["category"]=pd.DataFrame(predictions, columns=["category"])
test_df
```

[ ]:
```
           Image  category
0          1.jpg  0.001196
1         10.jpg  0.000040
2        100.jpg  0.048457
3       1000.jpg  0.046779
4      10000.jpg  0.183960
…            …         …
12495   9995.jpg  0.000003
```

```
12496   9996.jpg  0.896210
12497   9997.jpg  0.000276
12498   9998.jpg  0.000013
12499   9999.jpg  0.000391

[12500 rows x 2 columns]
```

[ ]:
```
plt.imshow(plt.imread(("/kaggle/working/test/10.jpg")))
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7f3281125b10>
```



[ ]:
```
test_df=test_df.rename(columns={"category": "label"})
test_df
```

[ ]:
```
            Image     label
0           1.jpg  0.001196
1          10.jpg  0.000040
2         100.jpg  0.048457
3        1000.jpg  0.046779
4       10000.jpg  0.183960
…             …         …
12495    9995.jpg  0.000003
12496    9996.jpg  0.896210
12497    9997.jpg  0.000276
12498    9998.jpg  0.000013
```

```
12499   9999.jpg  0.000391

[12500 rows x 2 columns]
```

[ ]:
```
test_df.drop("Image", axis=1, inplace=True)
test_df["id"] = np.arange(len(predictions)) + 1
cols = test_df.columns.tolist()
cols = cols[-1:] + cols[:-1]
submission_df = test_df[cols]
```

[ ]:
```
submission_df
```

[ ]:
```
          id     label
0          1  0.001196
1          2  0.000040
2          3  0.048457
3          4  0.046779
4          5  0.183960
…        …        …
12495  12496  0.000003
12496  12497  0.896210
12497  12498  0.000276
12498  12499  0.000013
12499  12500  0.000391

[12500 rows x 2 columns]
```

[ ]:
```
submission_df.to_csv("/content/drive/MyDrive/cnn_model3_submission.csv", index␣
 ↪= False)
```

[ ]:
```

```

MSDS 422 Assignment 7 (CNN #4)

May 18, 2021

1 Assignment 7: Image Processing With a CNN

This assignment follows the same structure as Assignment 6. We will employ at least a 2x2 completely crossed experimental design. We will again use a simple training-and-test regimen. The factors in the design may include numbers of layers and/or nodes within layers, types of layers (convolutional or pooling), and/or other hyperparameters. You will utilize convolutional neural networks (CNNs) within Python TensorFlow.

This week, you will compete in the Dogs vs. Cats kernels Edition Kaggle.com competition, https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition (Links to an external site.). . (Links to an external site.)Specifically, you will build models using the training set to forecast the test set. The images are in .jpg format, so you will need to research how to handle that. You are required to submit no fewer than four models for evaluation by Kaggle.com, and you must provide your Kaggle.com scores and user ID for validation.

2 Management Problem

Assume that we are providing advice to a website provider who is looking for tools to automatically label images provided by end users. As we look across the factors in the study, making recommendations to management about image classification, we are most concerned about achieving the highest possible accuracy in image classification. That is, we should be willing to sacrifice training time for model accuracy. What type of machine learning model works best? If it is a convolutional neural network, what type of network should we use? Part of this recommendation may concern information about the initial images themselves (input data for the classification task). What types of images work best?

3 1. Preparation of Data and Exploratory Data Analysis

[ ]:
```
# This Python 3 environment comes with many helpful analytics libraries
# installed
# It is defined by the kaggle/python Docker image:
# https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

```
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter)
# will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that
# gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved
# outside of the current session
```

[ ]:
```
import os
import zipfile
import random
import tensorflow as tf
import shutil
import numpy as np
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.regularizers import l2
from shutil import copyfile
from os import getcwd
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive/')
```

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

[ ]:
```
!ls drive/MyDrive/
```

```
'AYUDH Table Presentation (1).ppt'
'AYUDH Table Presentation (1).ppt.gslides'
'AYUDH Table Schedule (Santa Fe - Toronto).gsheet'
'AYUDH Table Summer 2018 (Santa Fe - Toronto) (1).gsheet'
'AYUDH Table Summer 2018 (Santa Fe - Toronto) (1).xlsx'
'AYUDH Table Summer 2018 (Santa Fe - Toronto) (2).xlsx'
'AYUDH Table Summer 2018 (Santa Fe - Toronto).gsheet'
'AYUDH Table Summer 2018 (Santa Fe - Toronto).xlsx'
'Brochure (1).gdoc'
 Brochure.gdoc
'CECS Overview 7.20.2020.gdoc'
 cnn_model2_submission.csv
 cnn_model3_submission.csv
 cnn_submission.csv
'COVID-19 Exploratory Data Analysis in R.gdoc'
```

```
'COVID-19 Research Paper Analysis.docx'
'DASS 21 Data v1.gsheet'
'DASS 21 Data v1.xlsx'
'ENGL 001C- Essay 3 Proposal.gdoc'
'ENGL 001C Thesis Brainstorm.gdoc'
'English 001C Essay 2.docx'
'English 001C Essay 2.gdoc'
'English 001C- Final Notes.gdoc'
'English 1B- Essay 2 (final draft).docx'
'English 1B- Essay 2 (peer review).docx'
'English 4 Final Room Assignments F16.docx'
'English 4L Reading quizzes with answers.docx'
'English 4L Reading quizzes with answers.docx.gdoc'
'English 4L Reading quizzes with answers.gdoc'
'Math 144- Chapter 10 problems.gdoc'
'Math 144- Chapter 7 worked out problems.gdoc'
'Math 144 Chapter 8 problems.gdoc'
'Math 144 Worked out problems- Chapter 9.gdoc'
'Math 168 Project 1.gdoc'
'Microsoft Band Sizing Guide_EN.pdf'
'MSDS 422 Assignment 7 (CNN #1).ipynb'
'MSDS 422 Assignment 7 (CNN #2).ipynb'
'MSDS 422 Assignment 7 (CNN #3).ipynb'
'MSDS 422 Assignment 7 (CNN #4).ipynb'
'NY revised.xlsx'
'NY revised.xlsx.gsheet'
'Play Books Notes'
 Report.gdoc
'Safari - Jul 23, 2018 at 01:12.pdf'
 Schedule.gsheet
 test.zip
 train.zip
'Untitled document (1).gdoc'
'Untitled document (2).gdoc'
'Untitled document (3).gdoc'
'Untitled document (4).gdoc'
'Untitled document (5).gdoc'
'Untitled document.gdoc'
'Untitled spreadsheet (1).gsheet'
'Untitled spreadsheet (2).gsheet'
'Untitled spreadsheet.gsheet'
 Week3Lecture_default.mp4
```

[ ]:
```
local_zip = '/content/drive/MyDrive/train.zip'

zip_ref = zipfile.ZipFile(local_zip, 'r')
```

```
zip_ref.extractall('/kaggle/working/')
zip_ref.close()
```

[ ]:
```
base_dir = '/kaggle/working/'
train_dir = os.path.join(base_dir, 'train')
train_img_names = os.listdir(train_dir)
```

[ ]:
```
train_img_names[:10]
```

[ ]:
```
['dog.5577.jpg',
 'dog.8301.jpg',
 'cat.11000.jpg',
 'dog.8878.jpg',
 'dog.11820.jpg',
 'cat.9373.jpg',
 'cat.5203.jpg',
 'dog.831.jpg',
 'dog.2816.jpg',
 'cat.6415.jpg']
```

[ ]:
```
print('total training images :', len(train_img_names ))
```

```
total training images : 25000
```

[ ]:
```
categories= list()
for image in train_img_names:
    category = image.split(".")[0]
    if category == "dog":
        categories.append("1")
    else:
        categories.append("0")

df= pd.DataFrame({"Image":train_img_names, "Category": categories})
```

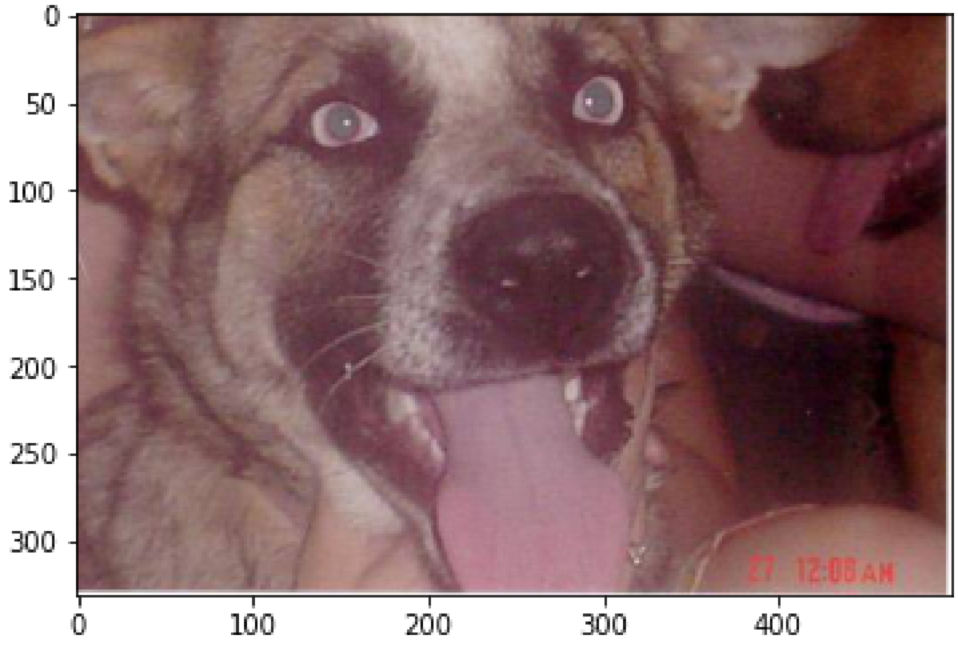
[ ]:
```
df.head()
```

[ ]:
```
           Image Category
0   dog.5577.jpg        1
1   dog.8301.jpg        1
2  cat.11000.jpg        0
3   dog.8878.jpg        1
4  dog.11820.jpg        1
```

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import random

sample = random.choice(train_img_names)
```

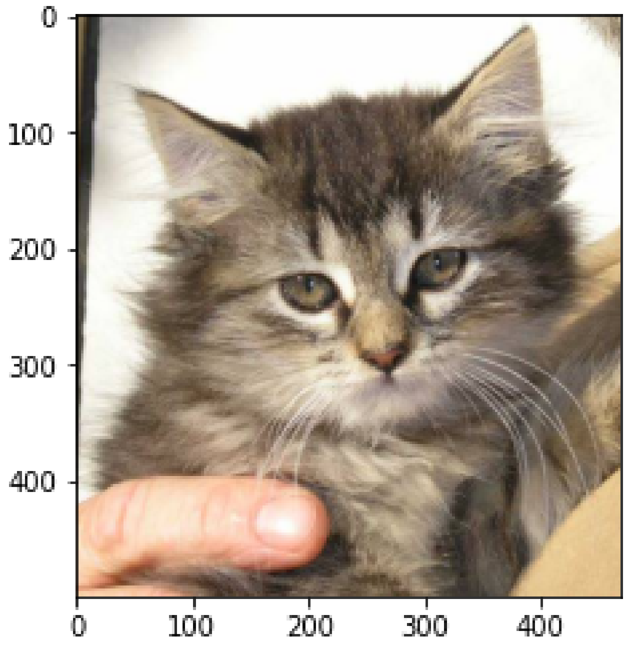
```
plt.imshow(plt.imread(("/kaggle/working/train/"+sample)))
```

```
[ ]: <matplotlib.image.AxesImage at 0x7f00d975c5d0>
```



```
[ ]: sample = random.choice(train_img_names)
     plt.imshow(plt.imread(("/kaggle/working/train/"+sample)))
```

```
[ ]: <matplotlib.image.AxesImage at 0x7f00d8254190>
```



```
[ ]: from sklearn.model_selection import train_test_split
     train,validation= train_test_split(df, test_size=0.1)
     train = train.reset_index(drop=True)
     validation = validation.reset_index(drop=True)
```

```
[ ]: train
```

```
[ ]:                Image  Category
     0        cat.2870.jpg        0
     1        dog.3950.jpg        1
     2         cat.416.jpg        0
     3        dog.7579.jpg        1
     4        dog.8479.jpg        1
     …                 …        …
     22495    cat.2943.jpg        0
     22496    dog.7821.jpg        1
     22497    dog.2481.jpg        1
     22498    dog.2213.jpg        1
     22499   dog.10521.jpg        1

     [22500 rows x 2 columns]
```

```
[ ]: validation
```

[ ]:
```
              Image  Category
0     dog.12198.jpg         1
1       cat.703.jpg         0
2      dog.8214.jpg         1
3      cat.6025.jpg         0
4      cat.2970.jpg         0
…             …        …
2495   dog.8707.jpg         1
2496   cat.6429.jpg         0
2497    cat.820.jpg         0
2498   dog.2261.jpg         1
2499   cat.6412.jpg         0

[2500 rows x 2 columns]
```

[ ]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255.
train_datagen = ImageDataGenerator( rescale = 1.0/255. )

# --------------------
# Flow training images in batches of 20 using train_datagen generator
# --------------------
train_generator = train_datagen.flow_from_dataframe(train,
directory="/kaggle/working/train", x_col='Image', y_col='Category',
batch_size=20, class_mode='binary', target_size=(150, 150))
```

```
Found 22500 validated image filenames belonging to 2 classes.
```

[ ]:
```
validation_datagen  = ImageDataGenerator( rescale = 1.0/255.)
validation_generator =  validation_datagen.flow_from_dataframe(validation,
directory="/kaggle/working/train", x_col='Image', y_col='Category',
batch_size=20, class_mode  = 'binary', target_size = (150, 150))
```

```
Found 2500 validated image filenames belonging to 2 classes.
```

4 2. Building a Small Model from Scratch

[ ]:
```
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPool2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential()
```

```
model.add(Conv2D(filters=64, kernel_size=(3,3),activation="relu",
input_shape=(150,150,3)))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=3, strides=2))

model.add(Conv2D(filters=128, kernel_size=3,activation="relu"))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=3, strides=2))

model.add(Conv2D(filters=256, kernel_size=3, activation="relu"))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=3, strides=2))

model.add(Flatten())
model.add(Dense(units=512, activation="relu"))
model.add(Dropout(0.1))
model.add(Dense(units=1, activation="sigmoid"))
```

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

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 64)      1792
_________________________________________________________________
batch_normalization (BatchNo (None, 148, 148, 64)      256
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 73, 73, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 71, 71, 128)       73856
_________________________________________________________________
batch_normalization_1 (Batch (None, 71, 71, 128)       512
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 35, 35, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 33, 33, 256)       295168
_________________________________________________________________
batch_normalization_2 (Batch (None, 33, 33, 256)       1024
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 256)       0
_________________________________________________________________
flatten (Flatten)            (None, 65536)             0
_________________________________________________________________
dense (Dense)                (None, 512)               33554944
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
```

```
-----------------------------------------------------------------
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 33,928,065
Trainable params: 33,927,169
Non-trainable params: 896

-----------------------------------------------------------------
```

[ ]:
```
import tensorflow as tf
model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=0.001),
loss='binary_crossentropy', metrics=['accuracy'])
```

[ ]:
```
callback=EarlyStopping(monitor="val_loss", patience=2)
callback_lr = tensorflow.keras.callbacks.ReduceLROnPlateau(
monitor='val_accuracy', patience=2, factor=0.5, min_lr=0.00001)
```

[ ]:
```
history=model.fit(train_generator, validation_data=validation_generator,
epochs=4, callbacks=[callback,callback_lr])
```

```
Epoch 1/4
1125/1125 [==============================] - 93s 68ms/step - loss: 5.4949 -
accuracy: 0.6395 - val_loss: 0.5423 - val_accuracy: 0.7372
Epoch 2/4
1125/1125 [==============================] - 76s 67ms/step - loss: 0.4816 -
accuracy: 0.7935 - val_loss: 2.1572 - val_accuracy: 0.6684
Epoch 3/4
1125/1125 [==============================] - 76s 67ms/step - loss: 0.3877 -
accuracy: 0.8437 - val_loss: 7.3791 - val_accuracy: 0.5252
```

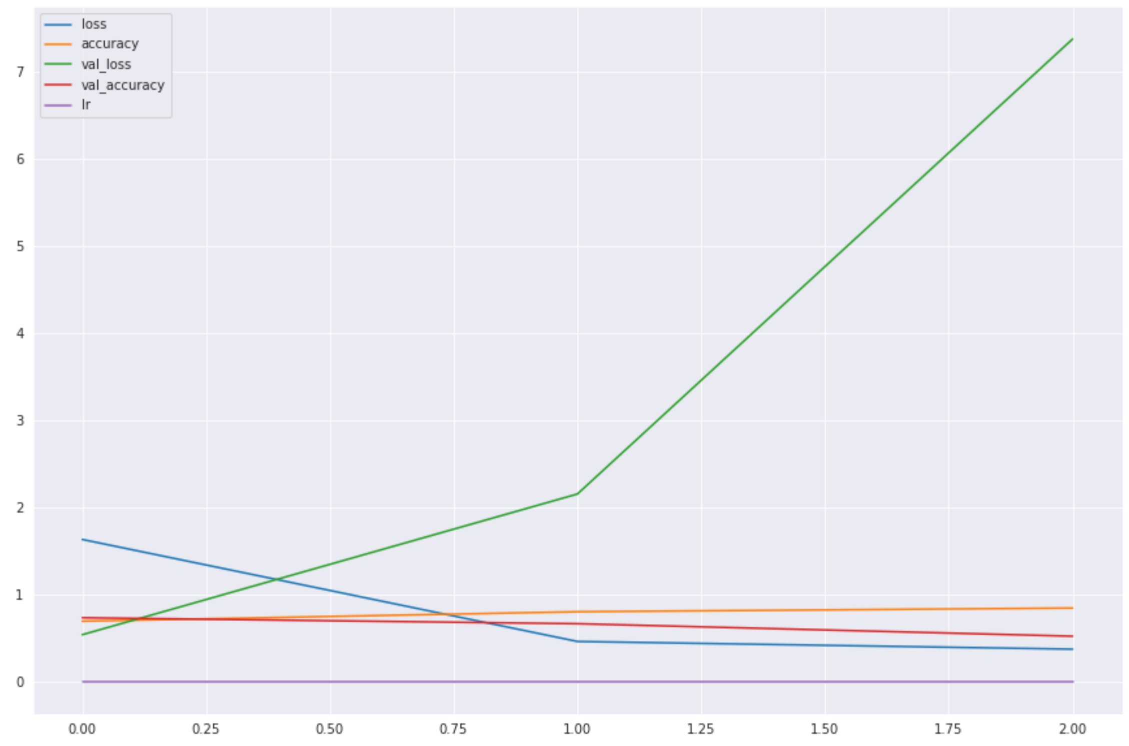
5 3. Performance Evaluation of the Training

[ ]:
```
pd.DataFrame(model.history.history)
```

[ ]:
```
       loss  accuracy  val_loss  val_accuracy     lr
0  1.635505  0.697600  0.542327        0.7372  0.001
1  0.464684  0.805956  2.157227        0.6684  0.001
2  0.376629  0.848356  7.379079        0.5252  0.001
```

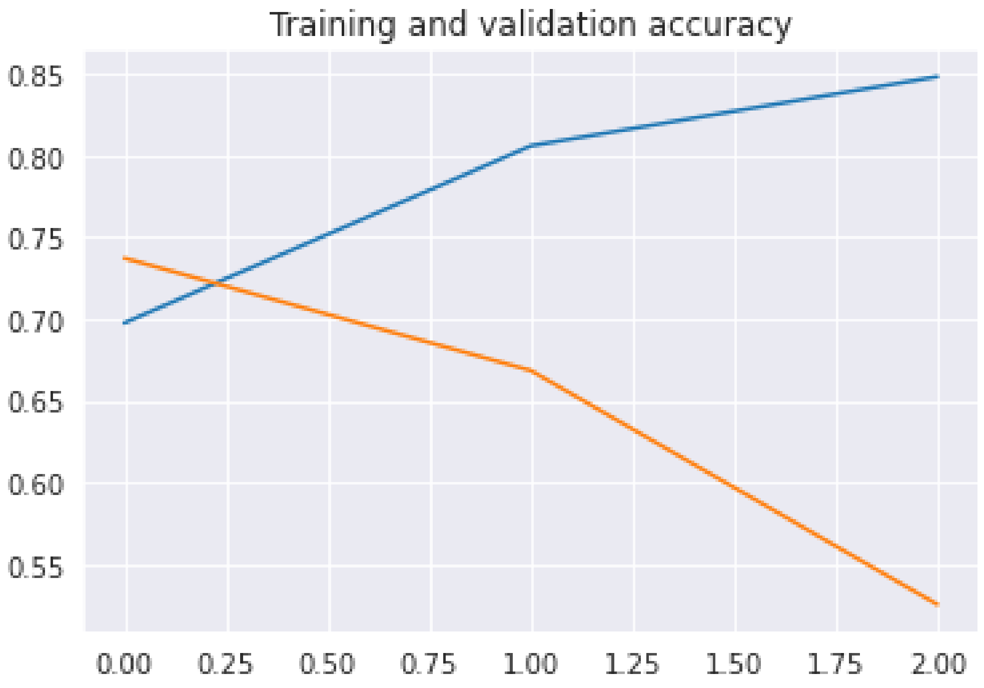
[ ]:
```
sns.set_style("darkgrid")
pd.DataFrame(model.history.history).plot(figsize=(15,10))
```

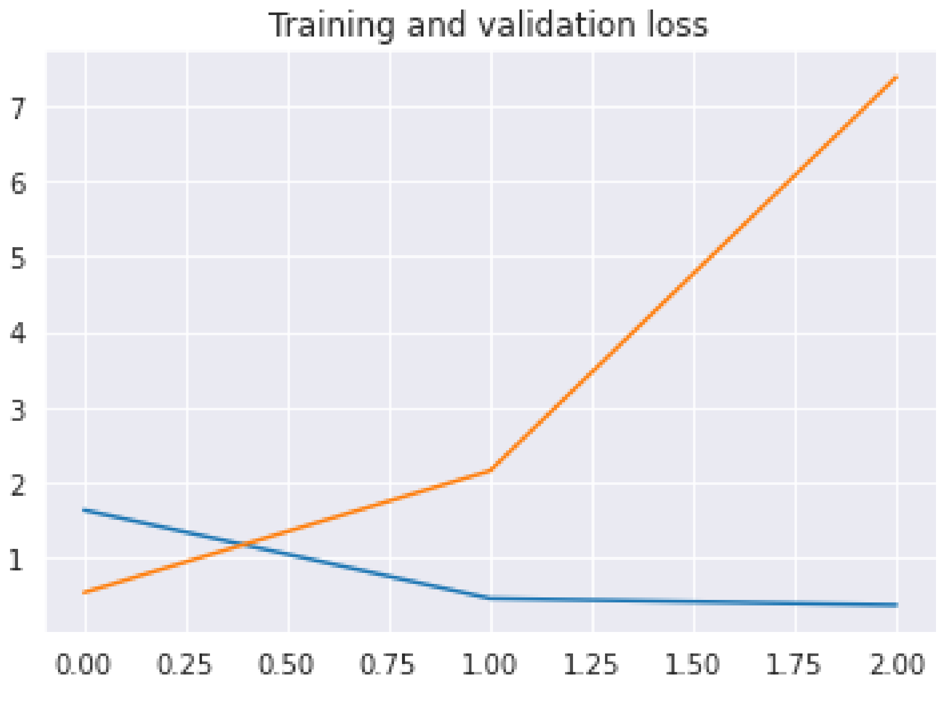
[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f00723084d0>
```



```
[ ]: acc      = history.history['accuracy']
     val_acc  = history.history['val_accuracy']
     loss     = history.history['loss']
     val_loss = history.history['val_loss']

     epochs   = range(len(acc)) # Get number of epochs

     #------------------------------------------------
     # Plot training and validation accuracy per epoch
     #------------------------------------------------
     plt.plot  ( epochs,     acc )
     plt.plot  ( epochs, val_acc )
     plt.title ('Training and validation accuracy')
     plt.figure()

     #------------------------------------------------
     # Plot training and validation loss per epoch
     #------------------------------------------------
     plt.plot(epochs, loss)
     plt.plot(epochs, val_loss)
     plt.title('Training and validation loss')
```

```
[ ]: Text(0.5, 1.0, 'Training and validation loss')
```


Training and validation accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Training and validation loss
7
6
5
4
3
2
1
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

6 4. Preparing Test Data and Submission

[ ]:
```
test_zip = '/content/drive/MyDrive/test.zip'

zip_ref = zipfile.ZipFile(test_zip, 'r')

zip_ref.extractall('/kaggle/working/')
zip_ref.close()
```

[ ]:
```
test_dir = '/kaggle/working/'
test_dir = os.path.join(test_dir, 'test')
test_img_names = [name.replace('test/', '') for name in zip_ref.namelist() if
len(name.replace('test/', '')) > 0]
```

[ ]:
```
test_img_names[:10]
```

[ ]:
```
['1.jpg',
 '10.jpg',
 '100.jpg',
 '1000.jpg',
 '10000.jpg',
 '10001.jpg',
 '10002.jpg',
 '10003.jpg',
 '10004.jpg',
 '10005.jpg']
```

[ ]:
```
len(test_img_names)
```

[ ]:
```
12500
```

[ ]:
```
test_df = pd.DataFrame({'Image': test_img_names})
test_df.head()
```

[ ]:
```
       Image
0      1.jpg
1     10.jpg
2    100.jpg
3   1000.jpg
4  10000.jpg
```

[ ]:
```
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_dataframe(test_df,
directory="/kaggle/working/test/", x_col="Image", y_col=None,
class_mode  = None, target_size=(150,150), shuffle = False, batch_size=20)
```

```
Found 12500 validated image filenames.
```

```
[ ]: predictions = model.predict(test_generator,steps = np.ceil(12500/20))
     %time predictions
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 6.68 µs
```

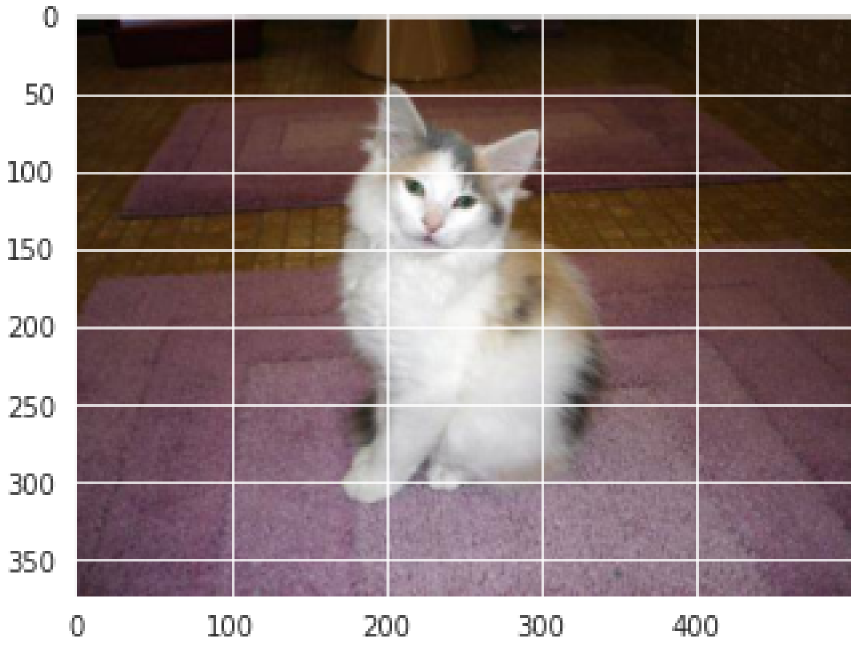
```
[ ]: array([[1.4293794e-08],
            [3.0209268e-12],
            [4.6461904e-05],
            …,
            [1.7628545e-05],
            [6.7324270e-17],
            [2.0469974e-10]], dtype=float32)
```

```
[ ]: test_df["category"]=pd.DataFrame(predictions, columns=["category"])
     test_df
```

```
[ ]:            Image      category
     0          1.jpg  1.429379e-08
     1         10.jpg  3.020927e-12
     2        100.jpg  4.646190e-05
     3       1000.jpg  8.717768e-04
     4      10000.jpg  6.020341e-02
     …            …           …
     12495   9995.jpg  1.091128e-14
     12496   9996.jpg  3.024104e-01
     12497   9997.jpg  1.762854e-05
     12498   9998.jpg  6.732427e-17
     12499   9999.jpg  2.046997e-10

     [12500 rows x 2 columns]
```

```
[ ]: plt.imshow(plt.imread(("/kaggle/working/test/10.jpg")))
```

```
[ ]: <matplotlib.image.AxesImage at 0x7f00e12fe710>
```



```
[ ]: test_df=test_df.rename(columns={"category": "label"})
     test_df
```

```
[ ]:             Image          label
     0           1.jpg   1.429379e-08
     1          10.jpg   3.020927e-12
     2         100.jpg   4.646190e-05
     3        1000.jpg   8.717768e-04
     4       10000.jpg   6.020341e-02
     …           …             …
     12495    9995.jpg   1.091128e-14
     12496    9996.jpg   3.024104e-01
     12497    9997.jpg   1.762854e-05
     12498    9998.jpg   6.732427e-17
     12499    9999.jpg   2.046997e-10

     [12500 rows x 2 columns]
```

```
[ ]: test_df.drop("Image", axis=1, inplace=True)
     test_df["id"] = np.arange(len(predictions)) + 1
     cols = test_df.columns.tolist()
     cols = cols[-1:] + cols[:-1]
     submission_df = test_df[cols]
```

```
[ ]: submission_df
[ ]:           id         label
     0          1  1.429379e-08
     1          2  3.020927e-12
     2          3  4.646190e-05
     3          4  8.717768e-04
     4          5  6.020341e-02
     …        …           …
     12495  12496  1.091128e-14
     12496  12497  3.024104e-01
     12497  12498  1.762854e-05
     12498  12499  6.732427e-17
     12499  12500  2.046997e-10

     [12500 rows x 2 columns]
```

```
[ ]: submission_df.to_csv("/content/drive/MyDrive/cnn_model4_submission.csv", index␣
      ↪= False)
```

```
[ ]:
```