Assignment-2

Harshith Reddy Suram

Introduction:

Efficient in tasks such as picture categorization and object detection, convnets have offered a dramatic shift in computer vision. The Cats & Dogs classification problem is a rather typical example of the convolutional neural network demonstrating its use. To solve this problem, one has to either create a new network or use the existence one or another network, which was previously trained. Both have their strengths and weaknesses, or more precisely their strengths are weaknesses – the size of the training dataset and the danger of overfitting. Learning from the smaller dataset often leads to the issue of overfitting; this is the situation in which the model's performance is excellent when solving the training data set problems but inadequate when faced with new data sets that it has not encountered before. From the previous sections it can be concluded that data augmentation, regularization and dropout are some of the fundamental ways which prevent overfitting and allow improving the model's ability to generalize.

In this assignment, we will uncover how training sample size affects the decision of using a convnet trained from above versus using a pre-trained network. Our expectation is to find out the conditions on the training sample size which yield the highest possible prediction performance to the Cats & Dogs categorization problem by reducing the training sample size systematically and applying optimization techniques. This work will therefore help to cast light on the impact of the different approaches for handling variable local data set sizes, as well as specific strategies likely to be most effective.

Problem Statement:

The objective of this task is to explore the degree to which the size of the training sample affects the accuracy of a convnet to classify photographs as either of cats or dogs. We'll compare two methods: Training a new convnet from the base and fine-tuning a new convnet from a pre-trained one. Our goal is to define the conditions in which we reach the highest prediction performance on the Cats & Dogs classification problem by gradually varying the training sample size and using the optimization algorithms.

Methodology:

Data Pre-processing:

I have taken the dataset of cats and dogs of 25000 images as my dataset here. I have divided them up into training and test beforehand and imported them into google colab**.**

```
[ ] import keras
    keras.__version__
```

```
'2.15.0'
```

```
[ ] import os, shutil
```

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

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

```
[ ] !unzip /content/drive/MyDrive/dogs-vs-cats.zip
```

```
Archive:  /content/drive/MyDrive/dogs-vs-cats.zip
  inflating: sampleSubmission.csv
  inflating: test1.zip
  inflating: train.zip
```

```
[ ] !unzip /content/train.zip
    !unzip /content/test1.zip
```

```
  inflating: test1/9720.jpg
  inflating: test1/9721.jpg
  inflating: test1/9722.jpg
  inflating: test1/9723.jpg
  inflating: test1/9724.jpg
  inflating: test1/9725.jpg
  inflating: test1/9726.jpg
  inflating: test1/9727.jpg
  inflating: test1/9728.jpg
  inflating: test1/9729.jpg
  inflating: test1/973.jpg
  inflating: test1/9730.jpg
  inflating: test1/9731.jpg
  inflating: test1/9732.jpg
  inflating: test1/9733.jpg
  inflating: test1/9734.jpg
  inflating: test1/9735.jpg
  inflating: test1/9736.jpg
  inflating: test1/9737.jpg
  inflating: test1/9738.jpg
  inflating: test1/9739.jpg
  inflating: test1/974.jpg
  inflating: test1/9740.jpg
  inflating: test1/9741.jpg
  inflating: test1/9742.jpg
```

```python
import glob
import pandas as pd
from tensorflow.keras.preprocessing.image import ImageDataGenerator
image = []
label = []
base_dir = '/content/'
train_path_content = os.listdir(f'{base_dir}/train')
print(f'train_size: {len(train_path_content)}')
train_dir = os.path.join(base_dir, 'train')

train_imgs = glob.glob(os.path.join(train_dir, "*"))
for img in train_imgs:
    image_name = os.path.splitext(os.path.basename(img))[0]
    if 'cat' in image_name:
    # move it to directory /content/train/cats
        image.append(img)
        label.append('cat')
    elif 'dog' in image_name:
    # move it to directory /content/train/dogs
        image.append(img)
        label.append('dog')
df = pd.DataFrame({'path':image,'label':label})
```

```
train_size: 25000
```

```
# The path to the directory where the original
# dataset was uncompressed
original_dataset_dir = '/content/train'

# The directory where we will
# store our smaller dataset
base_dir = '/content/dog_and_cat_small/'

os.mkdir(base_dir)

# Directories for our training,
# validation and test splits
train_dir = os.path.join(base_dir, 'train')
os.mkdir(train_dir)
validation_dir = os.path.join(base_dir, 'validation')
os.mkdir(validation_dir)
test_dir = os.path.join(base_dir, 'test')
os.mkdir(test_dir)

# Directory with our training cat pictures
train_cats_dir = os.path.join(train_dir, 'cats')
os.mkdir(train_cats_dir)

# Directory with our training dog pictures
train_dogs_dir = os.path.join(train_dir, 'dogs')
os.mkdir(train_dogs_dir)

# Directory with our validation cat pictures
validation_cats_dir = os.path.join(validation_dir, 'cats')
os.mkdir(validation_cats_dir)

# Directory with our validation dog pictures
validation_dogs_dir = os.path.join(validation_dir, 'dogs')
os.mkdir(validation_dogs_dir)

# Directory with our validation cat pictures
test_cats_dir = os.path.join(test_dir, 'cats')
os.mkdir(test_cats_dir)

# Directory with our validation dog pictures
test_dogs_dir = os.path.join(test_dir, 'dogs')
os.mkdir(test_dogs_dir)

# Copy first 1000 cat images to train_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(train_cats_dir, fname)
    shutil.copyfile(src, dst)

# Copy next 500 cat images to validation_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_cats_dir, fname)
    shutil.copyfile(src, dst)

# Copy next 500 cat images to test_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_cats_dir, fname)
    shutil.copyfile(src, dst)

# Copy first 1000 dog images to train_dogs_dir
fnames = ['dog.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(train_dogs_dir, fname)
    shutil.copyfile(src, dst)

# Copy next 500 dog images to validation_dogs_dir
fnames = ['dog.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_dogs_dir, fname)
    shutil.copyfile(src, dst)

# Copy next 500 dog images to test_dogs_dir
fnames = ['dog.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_dogs_dir, fname)
    shutil.copyfile(src, dst)
```

```
print('total training cat images:', len(os.listdir(train_cats_dir)))
```

```
total training cat images: 1000
```

Question-1:

Consider the Cats & Dogs example. Start initially with a training sample of 1000, a validation sample of 500, and a test sample of 500 (like in the text). Use any technique to reduce overfitting and improve performance in developing a network that you train from scratch. What performance did you achieve?

```
[ ]  print('total training cat images:', len(os.listdir(train_cats_dir)))
```

```
total training cat images: 1000
```

```
print('total training dog images:', len(os.listdir(train_dogs_dir)))
print('total validation cat images:', len(os.listdir(validation_cats_dir)))
print('total validation dog images:', len(os.listdir(validation_dogs_dir)))
print('total test cat images:', len(os.listdir(test_cats_dir)))
print('total test dog images:', len(os.listdir(test_dogs_dir)))
```

```
total training dog images: 1000
total validation cat images: 500
total validation dog images: 500
total test cat images: 500
total test dog images: 500
```

We will begin with 1000 samples for training, 500 samples for validation, and 500 samples for testing.

```
[ ] from keras import layers
    from keras import models

    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu',
                            input_shape=(150, 150, 3)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(128, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(128, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(512, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 max_pooling2d (MaxPooling2  (None, 74, 74, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 36, 36, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 17, 17, 128)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 15, 15, 128)       147584

 max_pooling2d_3 (MaxPoolin  (None, 7, 7, 128)         0
 g2D)

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 512)               3211776

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 3453121 (13.17 MB)
Trainable params: 3453121 (13.17 MB)
Non-trainable params: 0 (0.00 Byte)
```

This code first calls for the structuring of a convolutional neural network (CNN) that is involved in the differentiation of cats and dogs in photos. The developed CNN model employs Keras and comprises a significant number of convolutional layers, followed by max-pooling ones; the network is flattened and comprises two dense thick layers.

```
[ ] from keras import optimizers

    model.compile(loss='binary_crossentropy',
                  optimizer=optimizers.RMSprop(lr=1e-4),
                  metrics=['acc'])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate`
```

```
from keras.preprocessing.image import ImageDataGenerator

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

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 150x150
        target_size=(150, 150),
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

```
[ ] for data_batch, labels_batch in train_generator:
        print('data batch shape:', data_batch.shape)
        print('labels batch shape:', labels_batch.shape)
        break
```

```
data batch shape: (20, 150, 150, 3)
labels batch shape: (20,)
```

```
[ ] history = model.fit_generator(
          train_generator,
          steps_per_epoch=100,
          epochs=30,
          validation_data=validation_generator,
          validation_steps=50)
```

This model is with binary cross entropy as a loss function and RMSprop optimizer. This first class, ImageDataGenerator, rescales the image data and loads the data from directories for training and validation. The training and validation data set consists of photos of 2000 and 1000 in quantity, respectively, each of resize is 150*150 pixels. The model begins with a total of 30 iterations, and in each iteration, the model passes exactly through 100 steps to determine its efficiency on the validation set, with accuracy as the evaluative criteria.

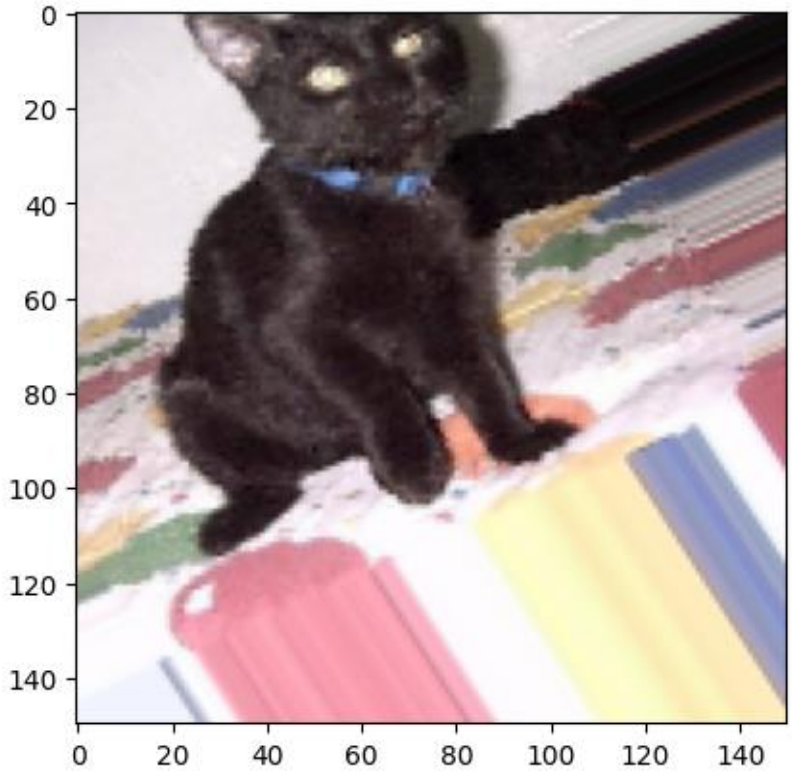
Data Augmentation:

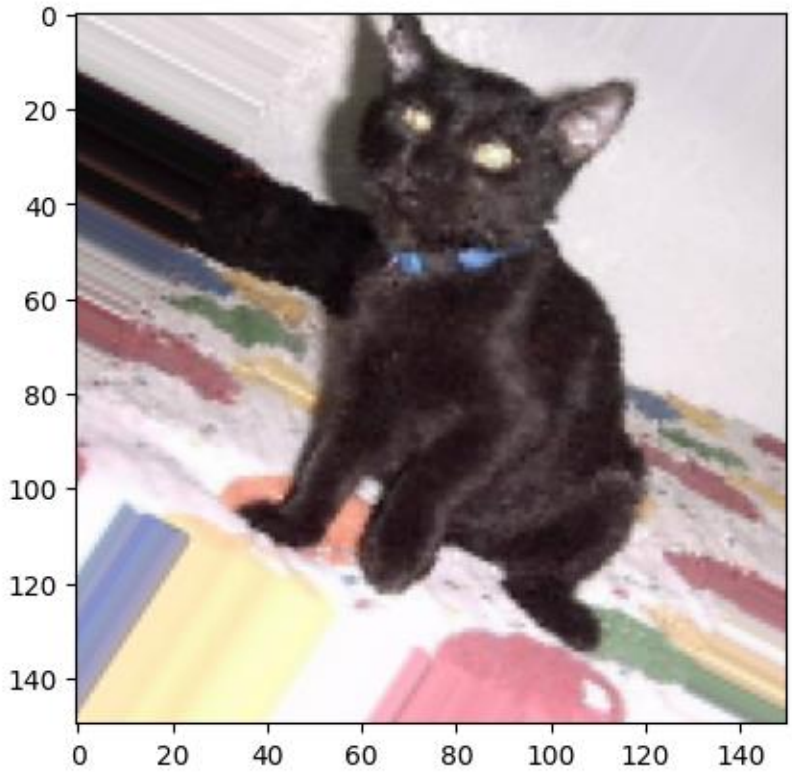
To prevent overfitting and increase performance, the method uses data augmentation and dropout in the creation of a convolutional neural network (CNN) trained from scratch to categorize photos of cats and dogs. Random rotations, width and height shifts, shearing, zooming, and horizontal flipping are all data augmentation techniques that artificially enlarge the training dataset by creating different versions of the original images. The CNN design is made up of multiple convolutional and max-pooling layers, followed by a flattening layer, a dropout layer (with a 0.5 dropout rate), and dense layers, and finally a sigmoid

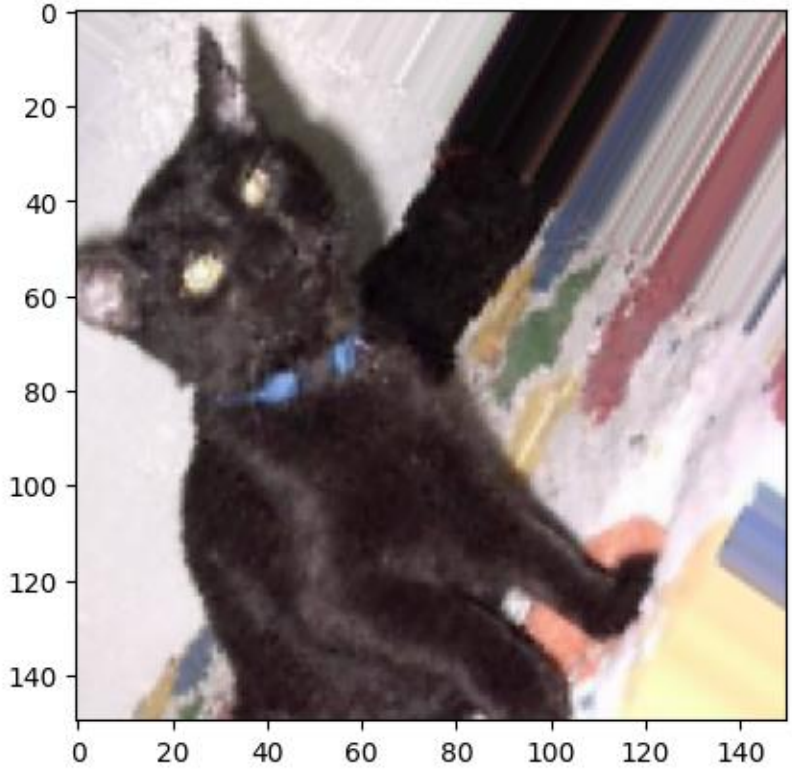
activation function for binary classification. The model is built using binary crossentropy loss and an RMSprop optimizer.

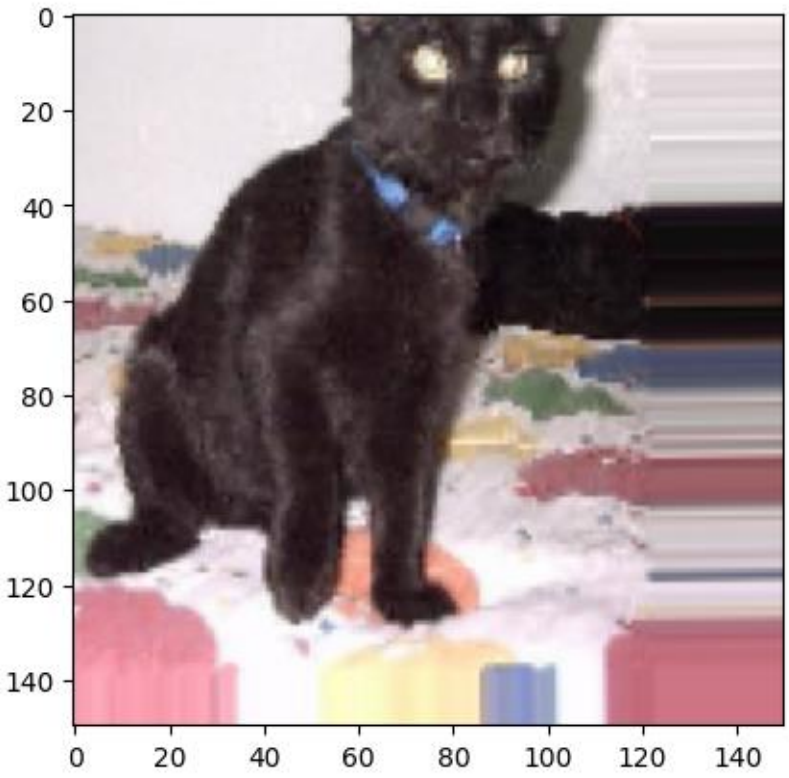
```
[ ] datagen = ImageDataGenerator(
        rotation_range=90,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

```
# This is module with image preprocessing utilities
from keras.preprocessing import image

fnames = [os.path.join(train_cats_dir, fname) for fname in os.listdir(train_cats_

# We pick one image to "augment"
img_path = fnames[3]

# Read the image and resize it
img = image.load_img(img_path, target_size=(150, 150))

# Convert it to a Numpy array with shape (150, 150, 3)
x = image.img_to_array(img)

# Reshape it to (1, 150, 150, 3)
x = x.reshape((1,) + x.shape)

# The .flow() command below generates batches of randomly transformed images.
# It will loop indefinitely, so we need to `break` the loop at some point!
i = 0
for batch in datagen.flow(x, batch_size=1):
    plt.figure(i)
    imgplot = plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 4 == 0:
        break

plt.show()
```




0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140



Question-2:
Increase your training sample size. You may pick any amount. Keep the validation and test samples the same as above. Optimize your network (again training from scratch). What performance did you achieve?

First, I have increased the training samples to 3000 training samples while keeping the validation and test samples as 500 samples. Results are in the results section. Code is given below:

```
# Copy first 3000 cat images to train_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in range(3000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(train_cats_dir, fname)
    shutil.copyfile(src, dst)

# Copy next 500 cat images to validation_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_cats_dir, fname)
    shutil.copyfile(src, dst)

# Copy next 500 cat images to test_cats_dir
fnames = ['cat.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_cats_dir, fname)
    shutil.copyfile(src, dst)

# Copy first 3000 dog images to train_dogs_dir
fnames = ['dog.{}.jpg'.format(i) for i in range(3000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(train_dogs_dir, fname)
    shutil.copyfile(src, dst)

# Copy next 500 dog images to validation_dogs_dir
fnames = ['dog.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_dogs_dir, fname)
    shutil.copyfile(src, dst)

# Copy next 500 dog images to test_dogs_dir
fnames = ['dog.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_dogs_dir, fname)
    shutil.copyfile(src, dst)
```

```
print('total training dog images:', len(os.listdir(train_dogs_dir)))
print('total validation cat images:', len(os.listdir(validation_cats_dir)))
print('total validation dog images:', len(os.listdir(validation_dogs_dir)))
print('total test cat images:', len(os.listdir(test_cats_dir)))
print('total test dog images:', len(os.listdir(test_dogs_dir)))
```

```
total training dog images: 3000
total validation cat images: 500
total validation dog images: 500
total test cat images: 500
total test dog images: 500
```

```
datagen = ImageDataGenerator(
      rotation_range=90,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.RMSprop.
```

```
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,)

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 150x150
        target_size=(150, 150),
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')

history = model.fit(
      train_generator,
      steps_per_epoch=100,
      epochs=100,
      validation_data=validation_generator,
```

Question-3:

Now change your training sample so that you achieve better performance than those from Steps 1 and 2. This sample size may be larger, or smaller than those in the previous steps. The objective is to find the ideal training sample size to get best prediction results

Question-4:

Repeat Steps 1-3, but now using a pretrained network. The sample sizes you use in Steps 2 and 3 for the pretrained network may be the same or different from those using the network where you trained from scratch. Again, use all optimization techniques to get best performance.

A pretrained VGG-16 model was employed to classify dog and cat photos. With over a million images and a thousand classes pre-trained on the ImageNet dataset, the popular deep convolutional neural network model VGG-16 was created. Rich feature representations from a wide range of datasets can be benefited from by utilizing the pretrained weights and architecture of VGG-16.

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 150, 150, 3)]     0

 block1_conv1 (Conv2D)       (None, 150, 150, 64)      1792

 block1_conv2 (Conv2D)       (None, 150, 150, 64)      36928

 block1_pool (MaxPooling2D)  (None, 75, 75, 64)        0

 block2_conv1 (Conv2D)       (None, 75, 75, 128)       73856

 block2_conv2 (Conv2D)       (None, 75, 75, 128)       147584

 block2_pool (MaxPooling2D)  (None, 37, 37, 128)       0

 block3_conv1 (Conv2D)       (None, 37, 37, 256)       295168

 block3_conv2 (Conv2D)       (None, 37, 37, 256)       590080

 block3_conv3 (Conv2D)       (None, 37, 37, 256)       590080

 block3_pool (MaxPooling2D)  (None, 18, 18, 256)       0

 block4_conv1 (Conv2D)       (None, 18, 18, 512)       1180160

 block4_conv2 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_conv3 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 9, 9, 512)         0

 block5_conv1 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv2 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv3 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 14714688 (56.13 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator

base_dir = '/content/dog_and_cat_small'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

datagen = ImageDataGenerator(rescale=1./255)
batch_size = 20

def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count, 4, 4, 512))
    labels = np.zeros(shape=(sample_count))
    generator = datagen.flow_from_directory(
        directory,
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='binary')
    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = conv_base.predict(inputs_batch)
        features[i * batch_size : (i + 1) * batch_size] = features_batch
        labels[i * batch_size : (i + 1) * batch_size] = labels_batch
        i += 1
        if i * batch_size >= sample_count:
            # Note that since generators yield data indefinitely in a loop,
            # we must `break` after every image has been seen once.
            break
    return features, labels

train_features, train_labels = extract_features(train_dir, 2000)
validation_features, validation_labels = extract_features(validation_dir, 1000)
test_features, test_labels = extract_features(test_dir, 1000)
```

Results:

Metric table:

S No	Method	Training Size	Training Accuracy	Validation Accuracy	Test Accuracy
1	Training for 1000 samples	1000	99.25	74.10	76.30
2	With Augmentation	1000	96.35	83.10	85.26
3.	Optimizing the model	3000	92.30	86.40	89.2
4.	With Augmentation	3000	96.75	90.2	92.32

5.	Optimizing the model	7000	82.30	81.60	85
6.	Pre-trained model (VGG)	1000	99.45	92.40	93.46
7.	Data Augmentation	1000	99.44	93.40	96.77
8.	Pre-trained model (VGG)	3000	98.96	92.50	94.23
9.	Data Augmentation	3000	99.65	94.50	96.78
10.	Pre-trained model (VGG)	6100	99.18	95.60	98.40

Graphs:

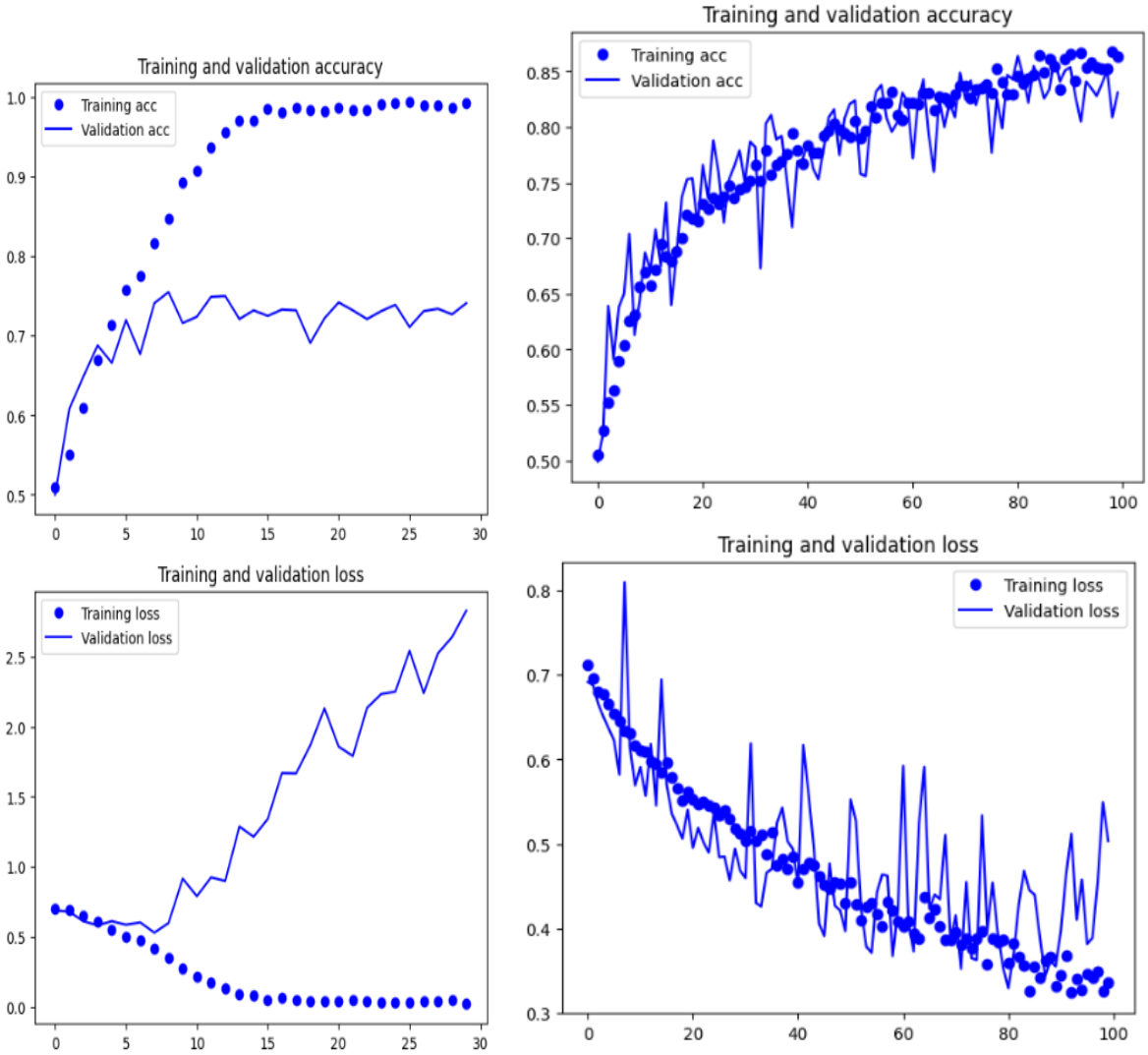


Figure1. training and validation accuracy and loss for training from scartch and data augmented training

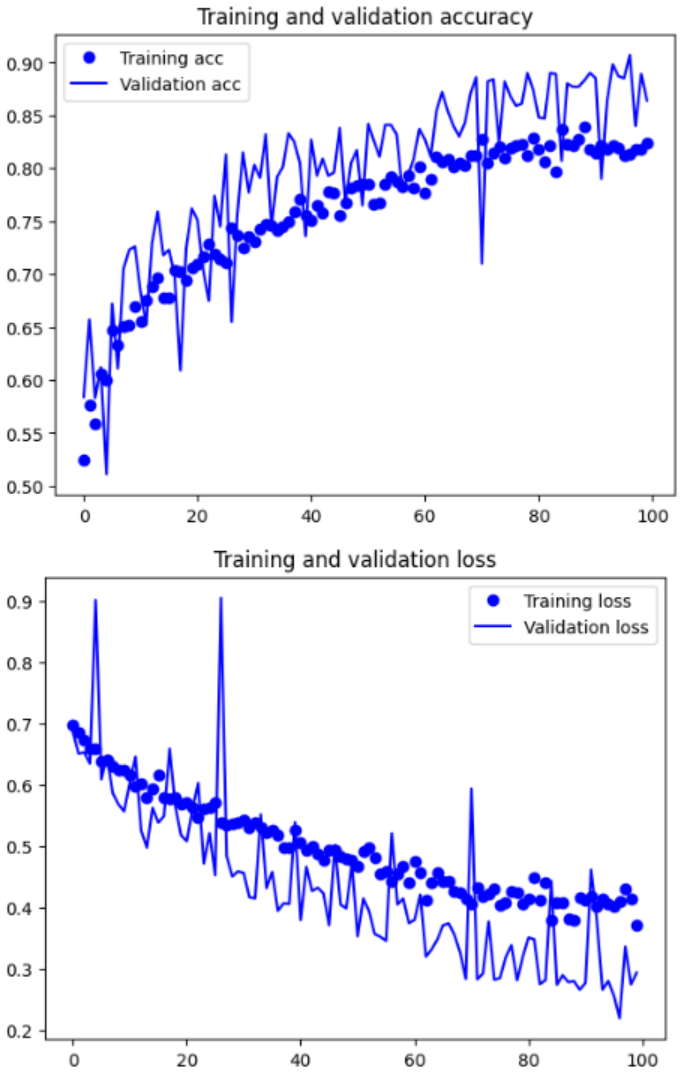


Figure2. Training and validation accuracy and loss for 3000 training samples.

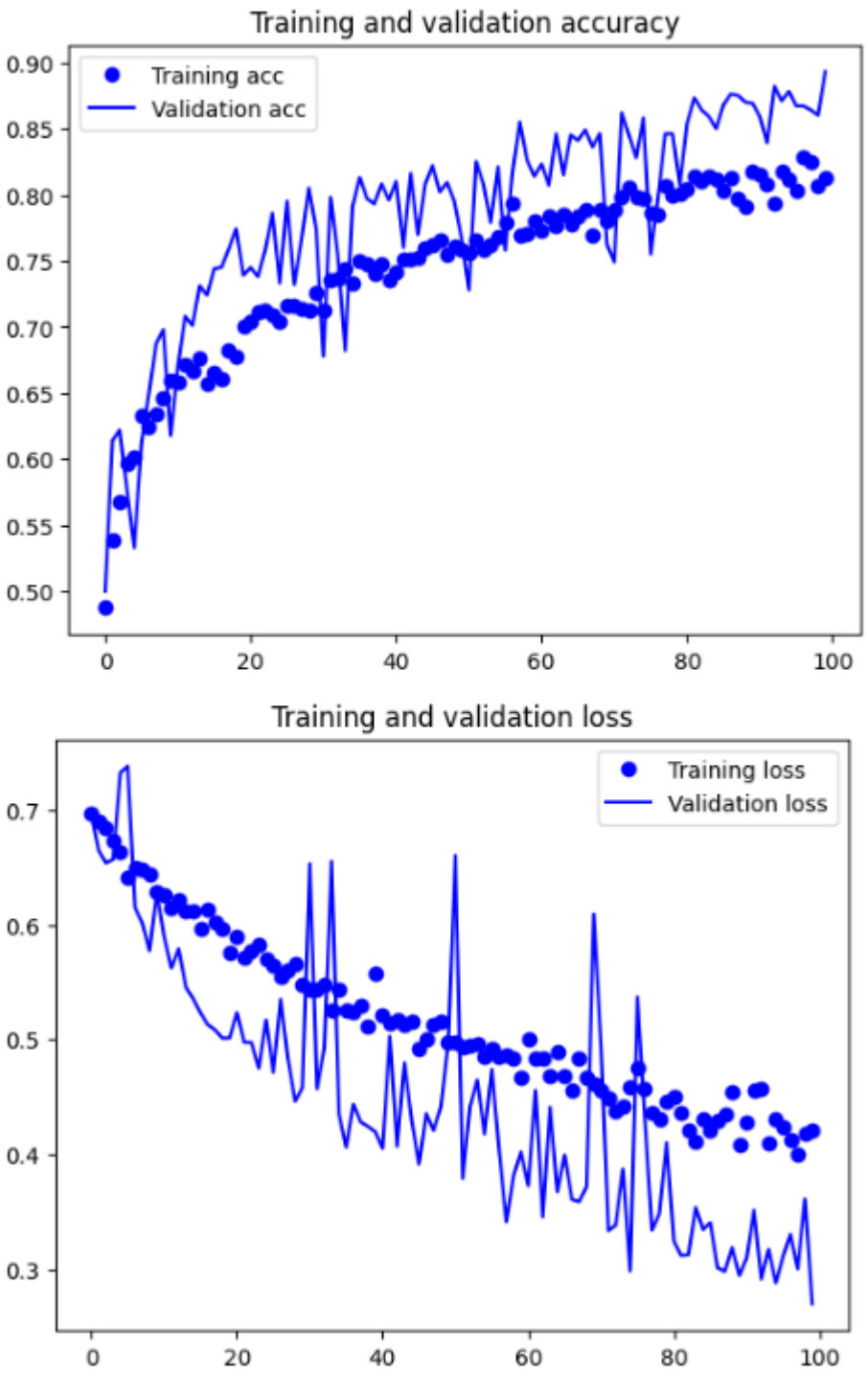


Figure3. training and validation accuracy and loss for training sample size of 7000 samples.

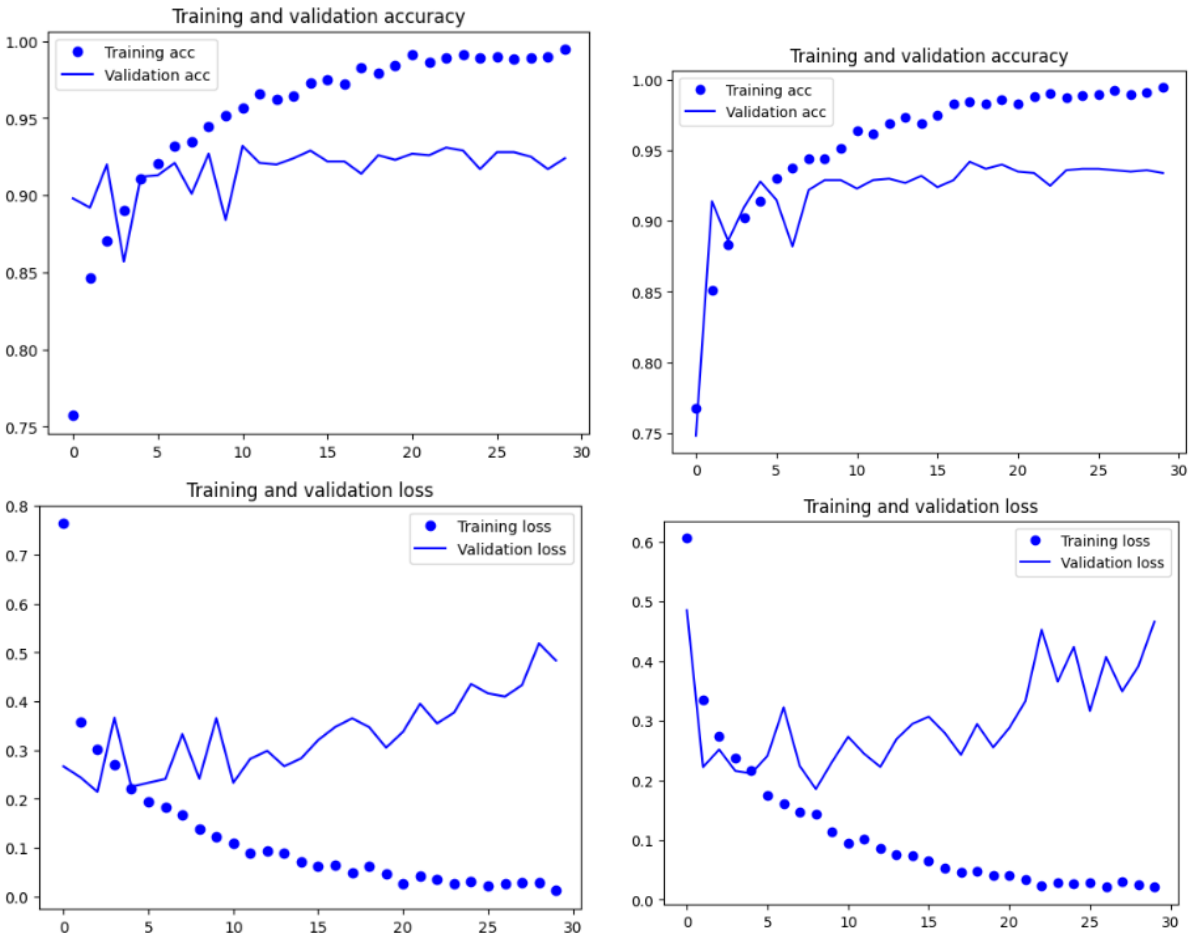


Figure4. Training and validation accuracy and loss for VGG model for different sizes.

Conclusion:

In this report, we proposed analyzing the methodology of training cat and dog images using convolutional neural networks and whether to train the model from raw data or not. In this case, I illustrated how using pretrained networks, especially the VGG-16 architecture can improve results as when training with less numbers of data. The implementation of transfer learning made it possible to utilize feature representations learned from a massive dataset, which in turn helped when classifying new data. By applying such methods like data augmentation and dropout, it was possible to prevent overfitting and enhance the effectiveness of the models. From the results, it was also seen that the models which used the Pretrained VGG-16 model were better than the model trained from 6100 samples in terms of accuracy along with having lesser valid loss. This underlines the necessity to establish proper methodologies and architectures in Deep Learning tasks and particularly while working with limited data. In summary, knowing how to apply transfer learning is important in computer vision and these results lay a strong foundation for further research and improvement in image classification.