NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY

GREATER NOIDA-201306

(An Autonomous Institute)

Department of Computer Science and Engineering

(Academic Session: 2024-25)

LAB FILE

ON

COMPUTER VISION LAB

(ACSE0701)

(7th Semester)

Submitted To:

Mrs. Shivani Sharma
Associate Professor, CSE

Submitted By:

Abhishek Kumar
Roll No. 2201330109002

Affiliated to Dr. A.P.J Abdul Kalam Technical University, Lucknow, Uttar Pradesh

INDEX

S. No	List of Experiment	Date	Signature	Grade
1.	Building a simple convolutional neural network for spam classification.			
2.	Building a simple convolutional neural network for image classification.			
3.	Implementing different types of pooling layers and comparing their effects on network performance.			
4.	Training a CNN model on a large-scale image classification dataset using cloud-based GPU acceleration.			
5.	Building a simple convolutional neural network for Cats-v-dogs classification			
6.	Fine-tuning a pre-trained CNN for a specific image recognition task.			
7.	Building a simple convolutional neural network for transfer learning using finetuning.			
8.	Building a simple convolutional neural network for transfer learning using feature extraction.			
9.	Building a CNN model for object detection using a pre-trained architecture like YOLO.			
10.	Exploring different activation functions and comparing their effects on network performance.			
11.	Write a program to Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.			
12	Implement a program for Basic image operations.			

1. Program: Building a simple convolutional neural network for spam classification

```
# 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 in

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import matplotlib.pyplot as plt

# Input data files are available in the "../input/" directory.

# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

# we need to fit model with sequence of tokens with specific length

from keras.preprocessing.sequence import pad_sequences

from keras.preprocessing.text import Tokenizer

from keras.models import Sequential

# normal LSTM/GRU and the Version with Cuda

from keras.layers import Dense, Embedding, GRU, LSTM, Dropout, Bidirectional

from keras.callbacks import TensorBoard, EarlyStopping, ModelCheckpoint

from keras.optimizers import Adam, rmsprop

# keras wrapper for k-fold cross-validation

from keras.wrappers.scikit_learn import KerasClassifier

# normsl cross validation

from sklearn.model_selection import cross_val_score, train_test_split

# cross validation for hyperparameter tuning

from sklearn.model_selection import GridSearchCV

import os

print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.
```

```
['spam.csv']
x_raw = []
y_raw = []

with open("../input/spam.csv", encoding = "ISO-8859-1") as f:
    for line in f:
        y_raw.append(line.split()[0])
        x_raw.append(' '.join(i for i in line.split()[1:]))
y = [1 if i=='ham' else 0 for i in y_raw]

print(max(len(s) for s in x_raw))
print(min(len(s)for s in x_raw))
sorted_X = sorted(len(s) for s in x_raw)
print(sorted_X[len(sorted_X) // 2])


909
0
60
```

tokenizer = Tokenizer()

tokenizer.fit_on_texts(x_raw)

sequences = tokenizer.texts_to_sequences(x_raw)

vocab_size = len(tokenizer.word_index)+1

print(vocab_size)

```
8734
# divide sum of length of all sequences by number of all sequences to find
averge length of each sequence
sum([len(x) for x in sequences]) // len(sequences)
14
pad = 'post'
max_len = 25
embedding_size = 100
batch_size = 20
sequences = pad_sequences(sequences, maxlen=max_len, padding=pad,
truncating=pad)
sequences.shape

X_train, X_test, y_train, y_test = train_test_split(sequences, y, test_size
= 0.2, random_state= 0)
```

LSTM MODEL

```
In [7]:
linkcode
model = Sequential()
```

```
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_size,
input_length=max_len))
model.add(Dropout(0.8))
model.add(LSTM(140, return_sequences=False))
model.add(Dropout(0.8))
model.add(Dense(1, activation='sigmoid', name='Classification'))
model.summary()
```

```
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-
packages/keras/backend/tensorflow_backend.py:3445: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be
removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 25, 100)           873400
_________________________________________________________________
dropout_1 (Dropout)          (None, 25, 100)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 140)               134960
_________________________________________________________________
dropout_2 (Dropout)          (None, 140)               0
_________________________________________________________________
Classification (Dense)       (None, 1)                 141
=================================================================
Total params: 1,008,501
Trainable params: 1,008,501
Non-trainable params: 0
```

```
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
#save_best = ModelCheckpoint('SpamDetection.hdf', save_best_only=True,
monitor='val_acc', mode='max')
# callback_early_stopping = EarlyStopping(monitor='val_loss', patience=5,
verbose=1)
```

In [9]:

linkcode

```
# Uses Automatic Verification Datasets (fastest option)
# model.fit(X_train, y_train, epochs=n_epochs, batch_size=batch_size,
validation_split=0.1, callbacks=[callback_early_stopping])
n_epochs = 10
results = model.fit(X_train, y_train, epochs=n_epochs, batch_size=batch_size,
validation_split=0.2, verbose=1)
```

```
WARNING:tensorflow:From /opt/conda/lib/python3.6/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a
future version.
Instructions for updating:
Use tf.cast instead.

Train on 3568 samples, validate on 892 samples
```

```
Epoch 1/10
3568/3568 [==============================] - 10s 3ms/step - loss: 0.0347 -
acc: 0.9947 - val_loss: 3.0947e-07 - val_acc: 1.0000
Epoch 2/10
3568/3568 [==============================] - 9s 3ms/step - loss: 1.2361e-05
- acc: 1.0000 - val_loss: 1.7644e-07 - val_acc: 1.0000
Epoch 3/10
3568/3568 [==============================] - 9s 2ms/step - loss: 1.0764e-05
- acc: 1.0000 - val_loss: 1.3704e-07 - val_acc: 1.0000
Epoch 4/10
3568/3568 [==============================] - 9s 2ms/step - loss: 8.9787e-06
- acc: 1.0000 - val_loss: 1.2414e-07 - val_acc: 1.0000
Epoch 5/10
3568/3568 [==============================] - 9s 2ms/step - loss: 8.1109e-06
- acc: 1.0000 - val_loss: 1.1483e-07 - val_acc: 1.0000
Epoch 6/10
3568/3568 [==============================] - 9s 3ms/step - loss: 4.6303e-06
- acc: 1.0000 - val_loss: 1.0445e-07 - val_acc: 1.0000
Epoch 7/10
3568/3568 [==============================] - 9s 2ms/step - loss: 5.2418e-06
- acc: 1.0000 - val_loss: 1.0436e-07 - val_acc: 1.0000
Epoch 8/10
3568/3568 [==============================] - 9s 2ms/step - loss: 7.5257e-06
- acc: 1.0000 - val_loss: 1.0181e-07 - val_acc: 1.0000
Epoch 9/10
3568/3568 [==============================] - 9s 3ms/step - loss: 4.6587e-06
- acc: 1.0000 - val_loss: 1.0385e-07 - val_acc: 1.0000
Epoch 10/10
3568/3568 [==============================] - 9s 2ms/step - loss: 3.6048e-06
- acc: 1.0000 - val_loss: 1.0091e-07 - val_acc: 1.0000
```

In [10]:

linkcode

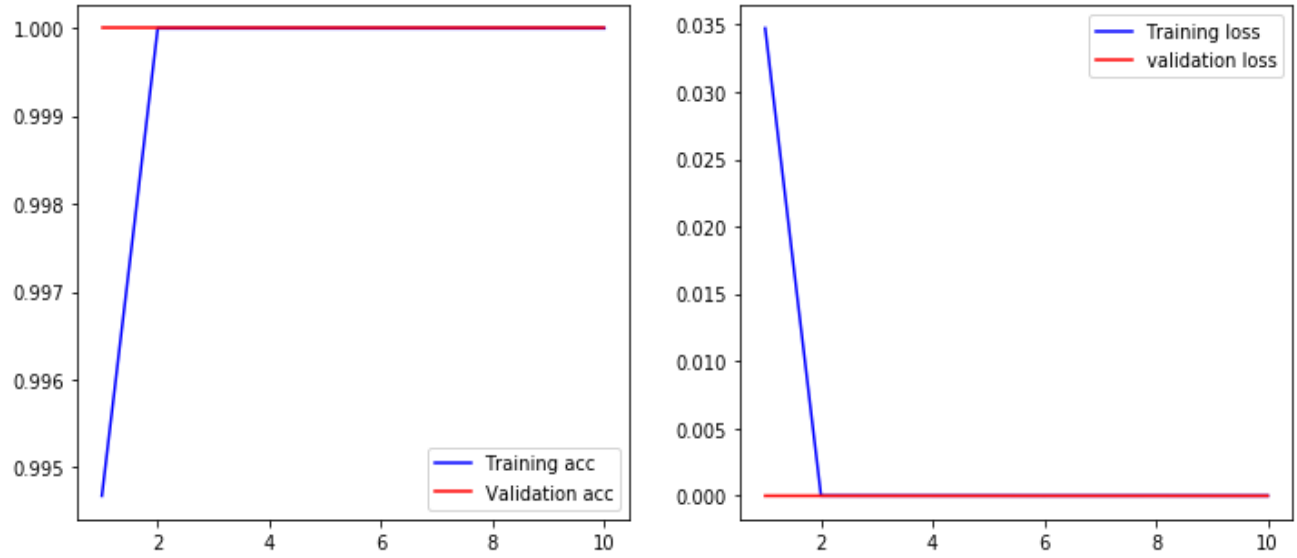
```
# model.load_weights(filepath='SpamDetection.hdf')
eval_ = model.evaluate(X_test, y_test)
print(eval_[0], eval_[1]) # loss / accuracy
```

```
1115/1115 [==============================] - 0s 367us/step
1.0138004321915118e-07 1.0
```

In [11]:

```
def plot_model(result):
    acc = result.history['acc']
    val_acc = result.history['val_acc']
    loss = result.history['loss']
    val_loss = result.history['val_loss']
    x = range(1, len(acc)+1)
    plt.figure(figsize=(12, 5))
    plt.subplot(1,2,1)
    plt.plot(x, acc, 'b', label='Training acc')
    plt.plot(x, val_acc, 'r', label= 'Validation acc')
    plt.legend()
```

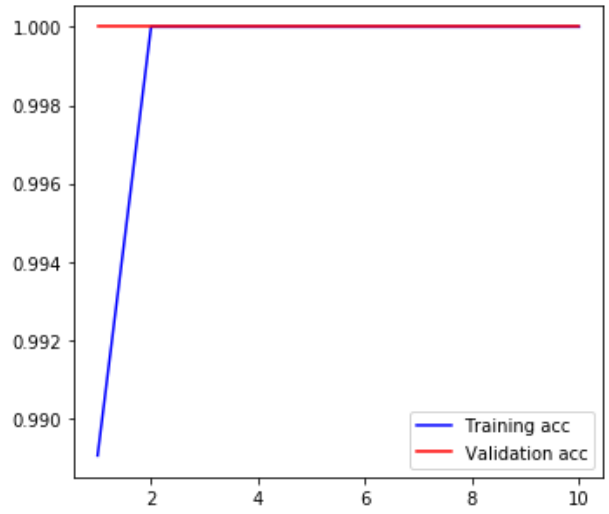
```
    plt.subplot(1,2,2)
 plt.plot(x, loss, 'b', label='Training loss')
    plt.plot(x, val_loss, 'r', label='validation loss')
    plt.legend()
plot_model(results)
```

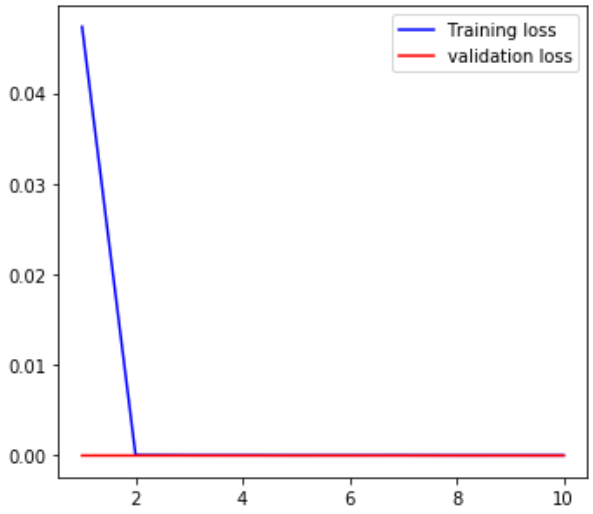



```
model1 = Sequential()
model1.add(Embedding(input_dim=vocab_size, output_dim=embedding_size,
input_length=max_len))
model1.add(Dropout(0.8))
model1.add(GRU(140, return_sequences=False))
model1.add(Dropout(0.86))
model1.add(Dense(1, activation='sigmoid', name='Classification'))
model1.summary()

model1.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
results1 = model1.fit(X_train, y_train, epochs=n_epochs,
batch_size=batch_size, validation_split=0.2)

eval_ = model1.evaluate(X_test, y_test)
print(eval_[0], eval_[1]) # loss / accuracy

plot_model(results1)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 25, 100)           873400
_________________________________________________________________
dropout_3 (Dropout)          (None, 25, 100)           0
_________________________________________________________________
gru_1 (GRU)                  (None, 140)               101220
_________________________________________________________________
dropout_4 (Dropout)          (None, 140)               0
_________________________________________________________________
Classification (Dense)       (None, 1)                 141
=================================================================
```

```
Total params: 974,761
Trainable params: 974,761
Non-trainable params: 0
_________________________________________________________________
Train on 3568 samples, validate on 892 samples
Epoch 1/10
3568/3568 [==============================] - 9s 3ms/step - loss: 0.0473 -
acc: 0.9891 - val_loss: 2.8143e-07 - val_acc: 1.0000
Epoch 2/10
3568/3568 [==============================] - 8s 2ms/step - loss: 5.6320e-05
- acc: 1.0000 - val_loss: 1.7271e-07 - val_acc: 1.0000
Epoch 3/10
3568/3568 [==============================] - 8s 2ms/step - loss: 3.3764e-05
- acc: 1.0000 - val_loss: 1.4434e-07 - val_acc: 1.0000
Epoch 4/10
3568/3568 [==============================] - 8s 2ms/step - loss: 3.0541e-05
- acc: 1.0000 - val_loss: 1.1887e-07 - val_acc: 1.0000
Epoch 5/10
3568/3568 [==============================] - 8s 2ms/step - loss: 2.4430e-05
- acc: 1.0000 - val_loss: 1.0496e-07 - val_acc: 1.0000
Epoch 6/10
3568/3568 [==============================] - 8s 2ms/step - loss: 2.6485e-05
- acc: 1.0000 - val_loss: 1.0443e-07 - val_acc: 1.0000
Epoch 7/10
3568/3568 [==============================] - 8s 2ms/step - loss: 2.8745e-05
- acc: 1.0000 - val_loss: 1.0069e-07 - val_acc: 1.0000
Epoch 8/10
3568/3568 [==============================] - 8s 2ms/step - loss: 1.5913e-05
- acc: 1.0000 - val_loss: 1.0045e-07 - val_acc: 1.0000
Epoch 9/10
3568/3568 [==============================] - 8s 2ms/step - loss: 1.7154e-05
- acc: 1.0000 - val_loss: 1.0037e-07 - val_acc: 1.0000
Epoch 10/10
3568/3568 [==============================] - 8s 2ms/step - loss: 1.5264e-05
- acc: 1.0000 - val_loss: 1.0000e-07 - val_acc: 1.0000
1115/1115 [==============================] - 0s 335us/step
1.0000002248489182e-07 1.0
```





2. Program: Building a simple convolutional neural network for image classification

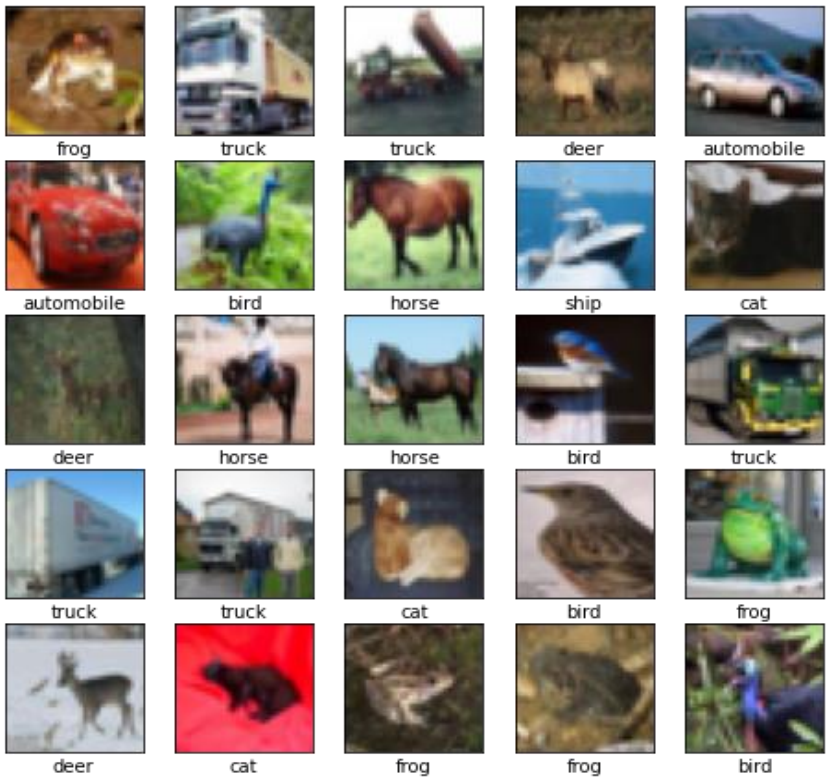
Import TensorFlow

import tensorflow as tf

from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

```
(train_images, train_labels), (test_images, test_labels) =
datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(8,8))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    #which is why we need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```




```
model = models.Sequential()
```

```
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32,
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(64, (3, 3), activation='relu'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
=================================================================
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
```

```
model.add(layers.Flatten())

model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 64)                65600
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
```

Compile and train the model

```
# Adam is the best among the adaptive optimizers in most of the cases
model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# An epoch means training the neural network with all the
# training data for one cycle. Here I use 10 epochs
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

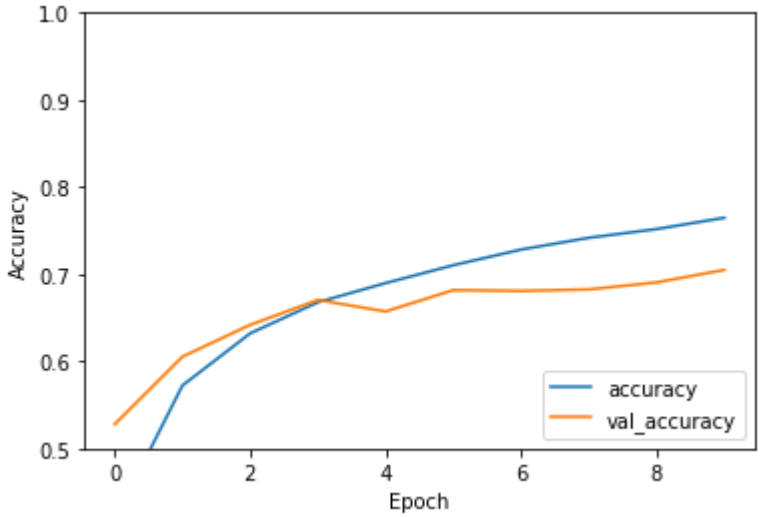
```
Epoch 1/10
1563/1563 [==============================] - 38s 24ms/step - loss: 1.8180 -
accuracy: 0.3202 - val_loss: 1.3074 - val_accuracy: 0.5278
Epoch 2/10
1563/1563 [==============================] - 36s 23ms/step - loss: 1.2492 -
accuracy: 0.5561 - val_loss: 1.0946 - val_accuracy: 0.6053
Epoch 3/10
1563/1563 [==============================] - 36s 23ms/step - loss: 1.0735 -
accuracy: 0.6236 - val_loss: 1.0204 - val_accuracy: 0.6418
Epoch 4/10
1563/1563 [==============================] - 36s 23ms/step - loss: 0.9666 -
accuracy: 0.6624 - val_loss: 0.9410 - val_accuracy: 0.6704
Epoch 5/10
1563/1563 [==============================] - 36s 23ms/step - loss: 0.8956 -
accuracy: 0.6847 - val_loss: 0.9987 - val_accuracy: 0.6572
Epoch 6/10
1563/1563 [==============================] - 36s 23ms/step - loss: 0.8198 -
accuracy: 0.7138 - val_loss: 0.9009 - val_accuracy: 0.6817
Epoch 7/10
1563/1563 [==============================] - 36s 23ms/step - loss: 0.7671 -
accuracy: 0.7316 - val_loss: 0.9355 - val_accuracy: 0.6808
Epoch 8/10
1563/1563 [==============================] - 36s 23ms/step - loss: 0.7241 -
accuracy: 0.7465 - val_loss: 0.9233 - val_accuracy: 0.6825
Epoch 9/10
1563/1563 [==============================] - 36s 23ms/step - loss: 0.6941 -
accuracy: 0.7554 - val_loss: 0.9287 - val_accuracy: 0.6905
Epoch 10/10
1563/1563 [==============================] - 36s 23ms/step - loss: 0.6630 -
accuracy: 0.7690 - val_loss: 0.8843 - val_accuracy: 0.7049
```

Evaluate the model

```
plt.plot(history.history['accuracy'],label='accuracy')
plt.plot(history.history['val_accuracy'],label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
```

```
test_loss, test_acc = model.evaluate(test_images,
                                     test_labels,
                                     verbose=2)
```

```
313/313 - 2s - loss: 0.8843 - accuracy: 0.7049
```




```
print('Test Accuracy is',test_acc)
```

```
Test Accuracy is 0.7049000263214111
```

3. Program: Implementing different types of pooling layers and comparing their effects on network performance

Setup feedback system

from learntools.core import binder

binder.bind(globals())

from learntools.computer_vision.ex3 import *

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

from matplotlib import gridspec

import learntools.computer_vision.visiontools as visiontools

plt.rc('figure', autolayout=True)

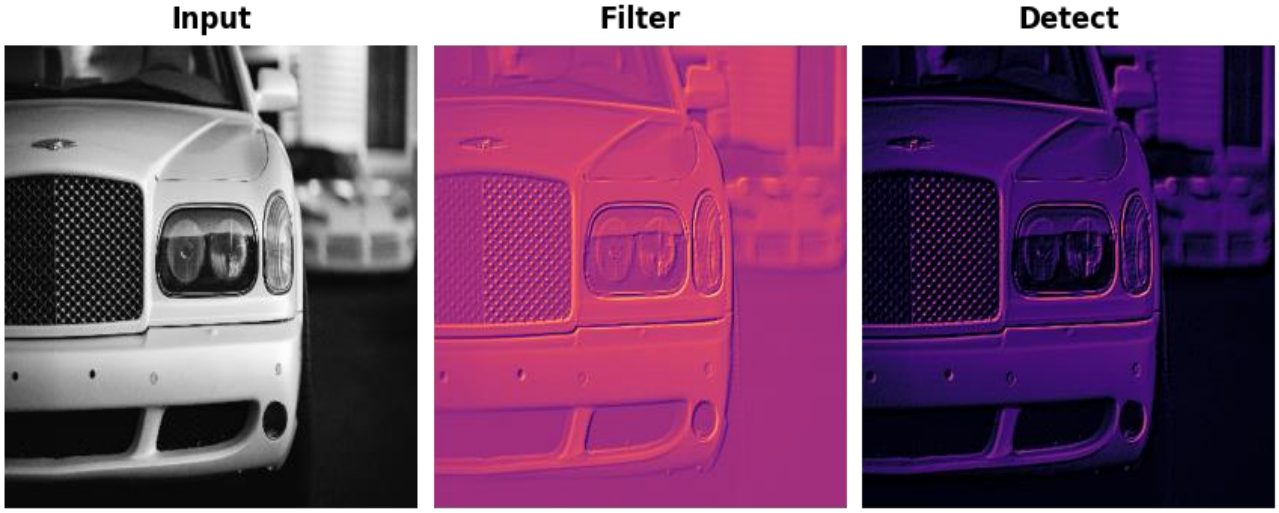
plt.rc('axes', labelweight='bold', labelsize='large',

titleweight='bold', titlesize=18, titlepad=10)

plt.rc('image', cmap='magma')

```
# Read image
image_path = '../input/computer-vision-resources/car_illus.jpg'
image = tf.io.read_file(image_path)
image = tf.io.decode_jpeg(image, channels=1)
image = tf.image.resize(image, size=[400, 400])

# Embossing kernel
kernel = tf.constant([
    [-2, -1, 0],
    [-1, 1, 1],
    [0, 1, 2],
])

# Reformat for batch compatibility.
image = tf.image.convert_image_dtype(image, dtype=tf.float32)
image = tf.expand_dims(image, axis=0)
kernel = tf.reshape(kernel, [*kernel.shape, 1, 1])
kernel = tf.cast(kernel, dtype=tf.float32)

image_filter = tf.nn.conv2d(
    input=image,
    filters=kernel,
    strides=1,
    padding='VALID', )

image_detect = tf.nn.relu(image_filter)
```

```
# Show what we have so far
plt.figure(figsize=(12, 6))
plt.subplot(131)
plt.imshow(tf.squeeze(image), cmap='gray')
plt.axis('off')
plt.title('Input')
plt.subplot(132)
plt.imshow(tf.squeeze(image_filter))
plt.axis('off')
plt.title('Filter')
plt.subplot(133)
plt.imshow(tf.squeeze(image_detect))
plt.axis('off')
plt.title('Detect')
plt.show();
```




```
# YOUR CODE HERE
image_condense = tf.nn.pool(
    input=image_detect,
    window_shape=(2, 2),
    pooling_type='MAX',

    strides=(2, 2),
    padding='SAME',
)

# Check your answer
q_1.check()
# Lines below will give you a hint or solution code
#q_1.hint()
#q_1.solution()
plt.figure(figsize=(8, 6))
plt.subplot(121)
plt.imshow(tf.squeeze(image_detect))
plt.axis('off')
plt.title("Detect (ReLU)")
plt.subplot(122)
plt.imshow(tf.squeeze(image_condense))
```

```
plt.axis('off')
plt.title("Condense (MaxPool)")
plt.show();
```

REPEATS = 4

SIZE = [64, 64]

Create a randomly shifted circle

image = visiontools.circle(SIZE, r_shrink=4, val=1)

image = tf.expand_dims(image, axis=-1)

image = visiontools.random_transform(image, jitter=3, fill_method='replicate')

image = tf.squeeze(image)

plt.figure(figsize=(16, 4))

plt.subplot(1, REPEATS+1, 1)

plt.imshow(image, vmin=0, vmax=1)

plt.title("Original\nShape: {}x{}".format(image.shape[0], image.shape[1]))

plt.axis('off')

Now condense with maximum pooling several times

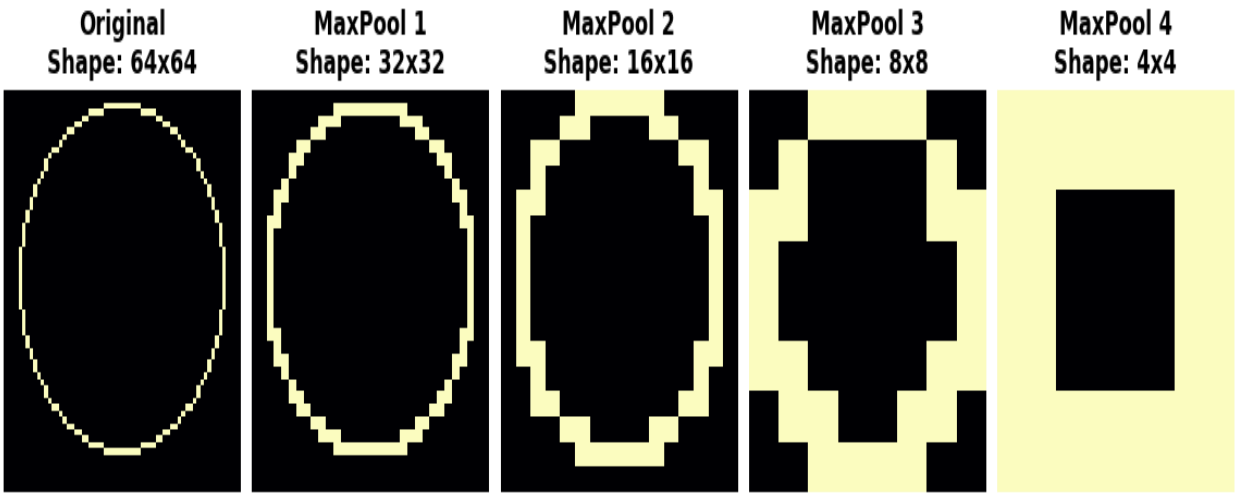
for i in range(REPEATS):

ax = plt.subplot(1, REPEATS+1, i+2)

image = tf.reshape(image, [1, *image.shape, 1])

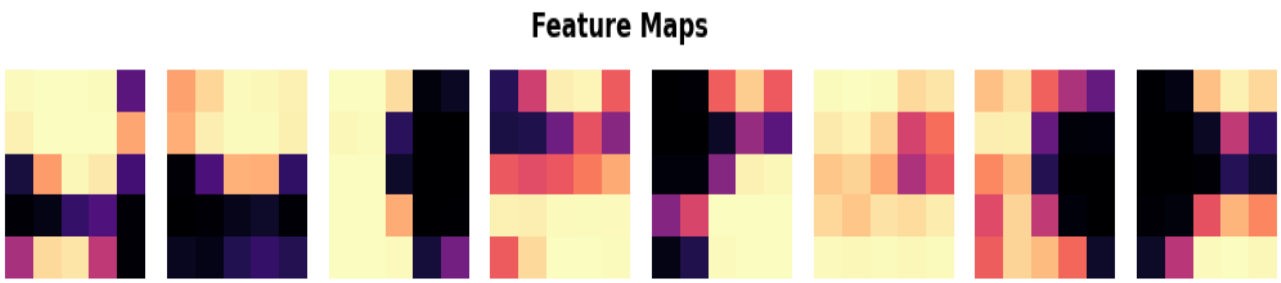
image = tf.nn.pool(image, window_shape=(2,2), strides=(2, 2), padding='SAME', pooling_type='MAX')

```
    image = tf.squeeze(image)
    plt.imshow(image, vmin=0, vmax=1)
 plt.title("MaxPool {}\nShape: {}x{}".format(i+1, image.shape[0], image.shape[1]))
    plt.axis('off')
```




```
# View the solution (Run this code cell to receive credit!)
q_2.solution()
feature_maps = [visiontools.random_map([5, 5], scale=0.1, decay_power=4) for _ in range(8)]
gs = gridspec.GridSpec(1, 8, wspace=0.01, hspace=0.01)
plt.figure(figsize=(18, 2))
for i, feature_map in enumerate(feature_maps):
    plt.subplot(gs[i])
    plt.imshow(feature_map, vmin=0, vmax=1)
    plt.axis('off')
plt.suptitle('Feature Maps', size=18, weight='bold', y=1.1)
plt.show()
# reformat for TensorFlow
feature_maps_tf = [tf.reshape(feature_map, [1, *feature_map.shape, 1])
                   for feature_map in feature_maps]
global_avg_pool = tf.keras.layers.GlobalAvgPool2D()
pooled_maps = [global_avg_pool(feature_map) for feature_map in feature_maps_tf]
```

```
img = np.array(pooled_maps)[:,:,0].T

plt.imshow(img, vmin=0, vmax=1)

plt.axis('off')

plt.title('Pooled Feature Maps')

plt.show();
```




```
import tensorflow.keras as keras
import tensorflow.keras.layers as layers
from tensorflow.keras.preprocessing import image_dataset_from_directory

# Load VGG16
pretrained_base = tf.keras.models.load_model(
    '../input/cv-course-models/cv-course-models/vgg16-pretrained-base',
)

model = keras.Sequential([
    pretrained_base,
    # Attach a global average pooling layer after the base
    layers.GlobalAvgPool2D(),
])

# Load dataset
ds = image_dataset_from_directory(
    '../input/car-or-truck/train',
    labels='inferred',
    label_mode='binary',
    image_size=[128, 128],

interpolation='nearest',
    batch_size=1,
    shuffle=True,
)

ds_iter = iter(ds)
car = next(ds_iter)

car_tf = (tf.image.resize(car[0], size=[192, 192]), car[1])
car_features = model(car_tf)
car_features = tf.reshape(car_features, shape=(16, 32))
label = int(tf.squeeze(car[1]).numpy())
```

```
plt.figure(figsize=(8, 4))
plt.subplot(121)
plt.imshow(tf.squeeze(car[0]))


plt.axis('off')
plt.title(["Car", "Truck"][label])
plt.subplot(122)
plt.imshow(car_features)
plt.title('Pooled Feature Maps')
plt.axis('off')
plt.show();
```


```
# View the solution (Run this code cell to receive credit!)
q_3.check()
```

4. Program: Training a CNN model on a large-scale image classification dataset using cloud-based GPU acceleration.

Step 1: Set Up Your Cloud Environment

You'll need to choose a cloud platform (e.g., AWS, Google Cloud, Azure) and provision a GPU-accelerated instance. Here, we'll assume you're using Google Colab for simplicity, but the process is similar on other platforms:

Go to https://colab.research.google.com/ and create a new notebook.

Set the runtime type to "GPU" by going to Runtime -> Change runtime type.

Step 2: Load and Preprocess the Data

```
# Load and preprocess data (adjust for your dataset)

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1.0/255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,

horizontal_flip=True,
    fill_mode='nearest'

)
train_generator = train_datagen.flow_from_directory(
    'path_to_train_data',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)
```

Step 3: Build the CNN Model

Define your CNN model using a deep learning framework like TensorFlow or PyTorch. Below is an example using TensorFlow and the Keras API:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
```

```
Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])
```

Step 4: Compile and Train the Model

Compile the model with a suitable optimizer and loss function and then train it on your data:

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(
    train_generator,
    epochs=10,  # Adjust as needed
    steps_per_epoch=len(train_generator),
    verbose=1
)
```

Step 5: Evaluate and Fine-Tune the Model

Evaluate the model on a validation set and fine-tune it as needed:

```
# Evaluate the model
validation_datagen = ImageDataGenerator(rescale=1.0/255)
validation_generator = validation_datagen.flow_from_directory(
    'path_to_validation_data',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)
evaluation = model.evaluate(validation_generator)
print("Validation Loss:", evaluation[0])
print("Validation Accuracy:", evaluation[1])
```

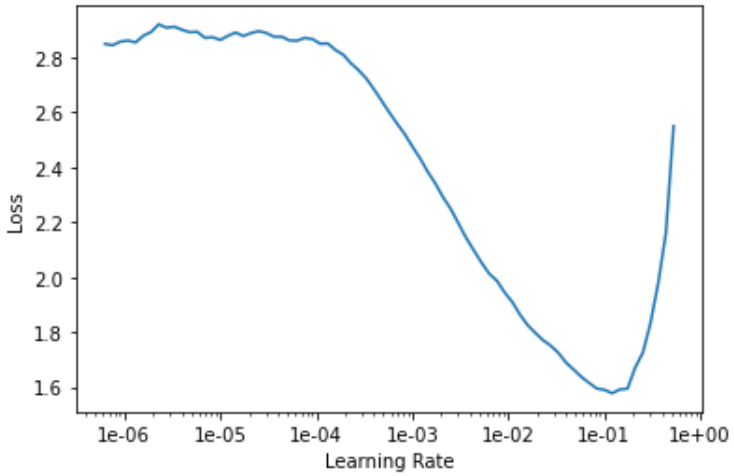
Step 6: Save the Model

Save the trained model for later use:

```
model.save('my_model.h5')
```

Step 7: Deploy and Use the Model

OUTPUT



epoch	train_loss	valid_loss	accuracy	time
0	0.405578	0.290146	0.901283	01:33
1	0.383602	0.299907	0.902708	01:29
2	0.336319	0.306456	0.911974	01:29
3	0.343718	0.326254	0.892374	01:30
4	0.364961	0.304092	0.906985	01:29
5	0.515605	0.546500	0.875624	01:30
6	0.622033	0.581009	0.892730	01:30
7	0.681928	0.578457	0.893443	01:30
8	1.094560	1.054756	0.897006	01:29
9	0.889135	1.874039	0.889522	01:30

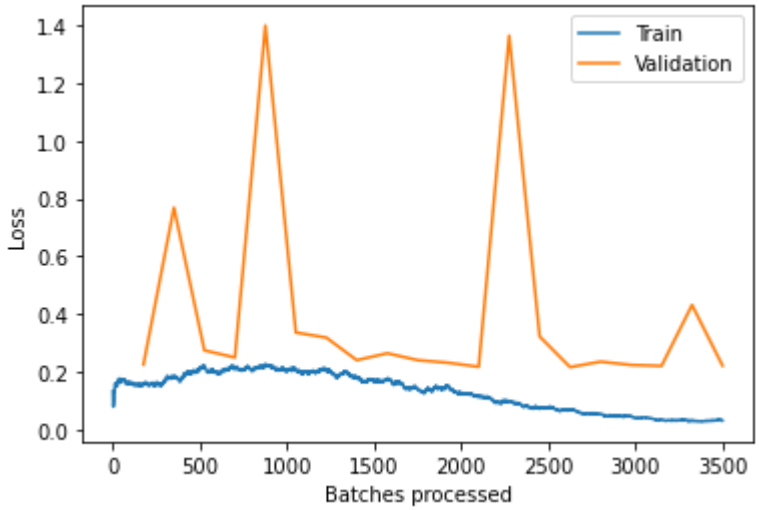
10	0.922833	0.993172	0.873129	01:30
11	1.075472	0.505854	0.904847	01:30
12	0.880544	1.152409	0.903421	01:29
13	0.948685	0.579885	0.908767	01:29
14	0.899647	4.366896	0.711689	01:30
15	1.108699	1.961280	0.892374	01:28
16	0.901288	0.625867	0.907698	01:29
17	0.689079	3.617636	0.872416	01:30
18	0.530735	0.456810	0.907341	01:28
19	0.805070	0.463680	0.915538	01:30
20	0.774058	0.893849	0.915895	01:31
21	0.581867	0.590447	0.906272	01:31
22	0.486774	0.639138	0.922666	01:30
23	0.482500	0.359934	0.919815	01:30
24	0.449640	21.844734	0.829294	01:30
25	0.326962	0.478932	0.924448	01:29
26	0.301796	0.600344	0.915182	01:29
27	0.270193	0.612086	0.913756	01:29
28	0.298829	0.291706	0.926230	01:29
29	0.233580	0.258509	0.920884	01:29
30	0.228145	0.269283	0.926230	01:29
31	0.212270	0.298909	0.927299	01:29
32	0.179242	0.326935	0.927655	01:29
33	0.185605	0.218010	0.931219	01:28
34	0.179383	0.349236	0.928724	01:28
35	0.166706	0.221297	0.934783	01:29
36	0.163385	0.212441	0.933001	01:29
37	0.141837	0.209331	0.934783	01:29
38	0.155481	0.211586	0.934426	01:29
39	0.151176	0.230446	0.933713	01:29

Hyper Parameter

epoch	train_loss	valid_loss	accuracy	time
0	0.159835	0.224613	0.935495	01:29
1	0.180235	0.768976	0.919815	01:30
2	0.219724	0.273913	0.918746	01:31

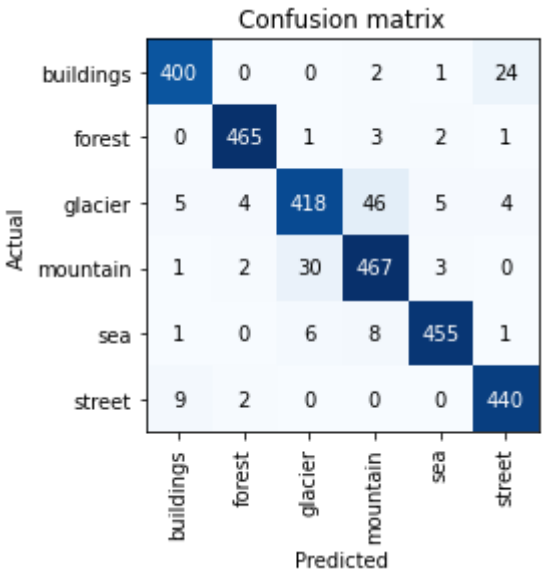
epoch	train_loss	valid_loss	accuracy	time
3	0.214327	0.249109	0.926230	01:30
4	0.225790	1.400049	0.875267	01:31
5	0.208292	0.335781	0.908054	01:31
6	0.204567	0.318150	0.900570	01:31
7	0.182223	0.239845	0.919458	01:30
8	0.174720	0.263758	0.914469	01:31
9	0.139013	0.239959	0.928368	01:31
10	0.151706	0.230929	0.931219	01:30
11	0.113556	0.216972	0.935495	01:30
12	0.097784	1.365048	0.932644	01:29
13	0.072197	0.321990	0.935139	01:34
14	0.067746	0.215753	0.941197	01:32
15	0.053457	0.234636	0.935852	01:28
16	0.046622	0.223260	0.941554	01:29

epoch	train_loss	valid_loss	accuracy	time
17	0.034932	0.220026	0.941910	01:30
18	0.030557	0.431193	0.940841	01:30
19	0.030724	0.221086	0.942623	01:29



Training and Loss Data

Confusion Matrix



5. Program: Building a simple convolutional neural network for Cats-v-dogs classification

```
import numpy as np
import pandas as pd
from keras.preprocessing.image import ImageDataGenerator, load_img
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import random
import os
print(os.listdir("../input"))
['train', 'test1', 'sampleSubmission.csv']
Define Constants
FAST_RUN = False
IMAGE_WIDTH=128
IMAGE_HEIGHT=128
IMAGE_SIZE=(IMAGE_WIDTH, IMAGE_HEIGHT)
IMAGE_CHANNELS=3

filenames = os.listdir("../input/train/train")
categories = []
for filename in filenames:
    category = filename.split('.')[0]
    if category == 'dog':
        categories.append(1)
    else:
        categories.append(0)

df = pd.DataFrame({
    'filename': filenames,
    'category': categories
})
df.head()
```

filename	category	
0	cat.8572.jpg	0
1	dog.11754.jpg	1
2	cat.3314.jpg	0
3	dog.11723.jpg	1
4	dog.4602.jpg	1

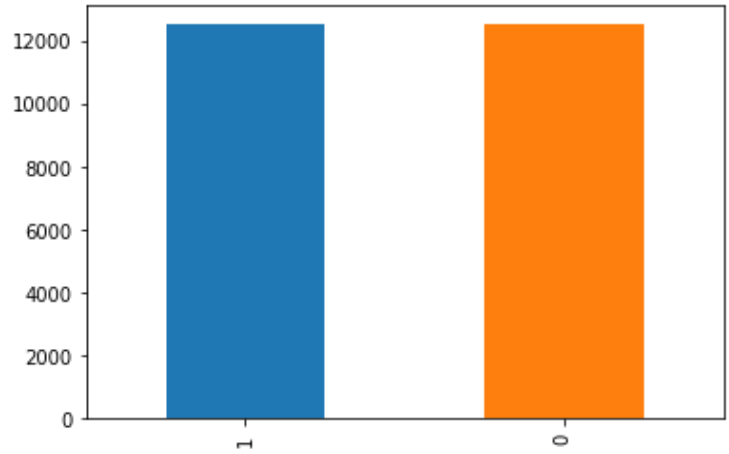
```
df.tail()
```

filename	category

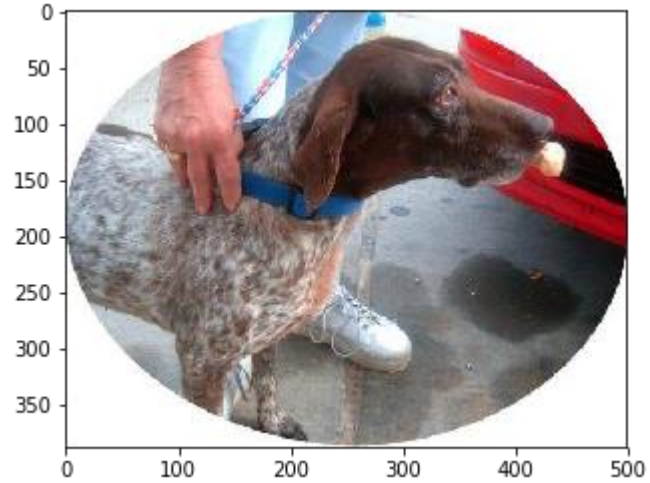
filename	category	
24995	cat.9040.jpg	0
24996	dog.5406.jpg	1
24997	cat.6371.jpg	0
24998	dog.2213.jpg	1
24999	cat.10115.jpg	0

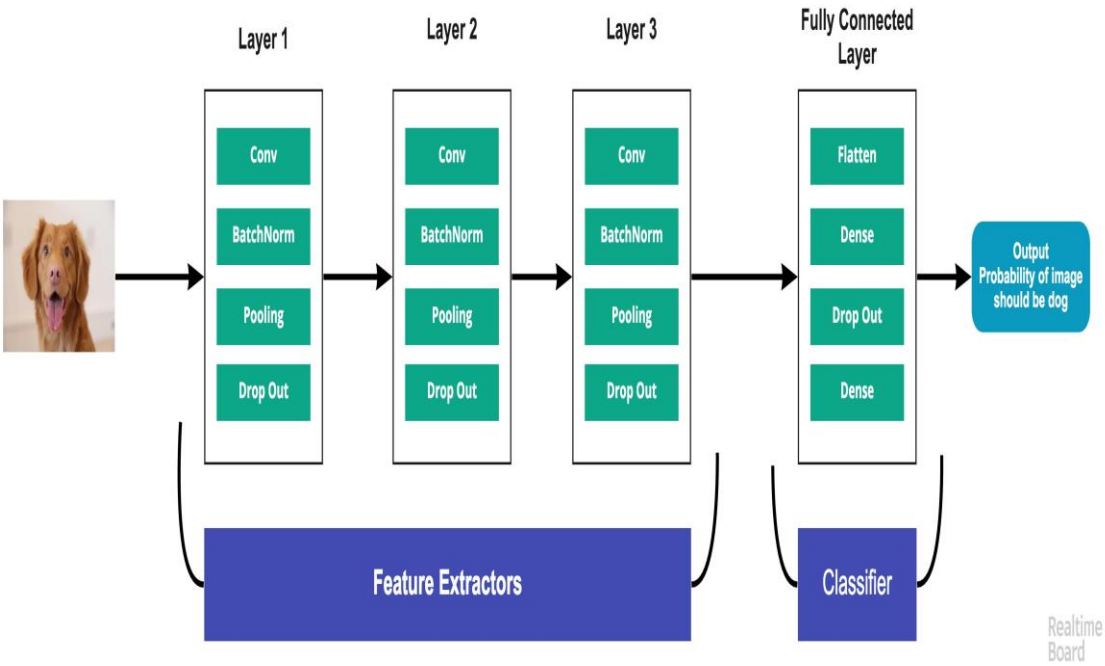
See Total In count

df['category'].value_counts().plot.bar()



sample = random.choice(filenames)
image = load_img("../input/train/train/"+sample)
plt.imshow(image)






```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation,
BatchNormalization
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(IMAGE_WIDTH,
IMAGE_HEIGHT, IMAGE_CHANNELS)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax')) # 2 because we have cat and dog classes
model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
model.summary()
```

```
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 126, 126, 32)      896
_________________________________________________________________
batch_normalization_1 (Batch (None, 126, 126, 32)      128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 63, 63, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 61, 61, 64)        18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 61, 61, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 128)       73856
_________________________________________________________________
batch_normalization_3 (Batch (None, 28, 28, 128)       512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 14, 14, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               12845568
_________________________________________________________________
batch_normalization_4 (Batch (None, 512)               2048
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 1026
=================================================================
Total params: 12,942,786
Trainable params: 12,941,314
Non-trainable params: 1,472
_________________________________________________________________
```

Callbacks

```
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
earlystop = EarlyStopping(patience=10)
```

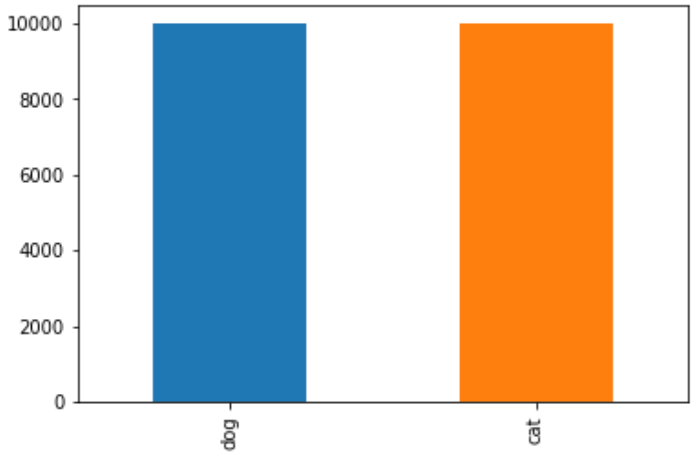
Learning Rate Reduction

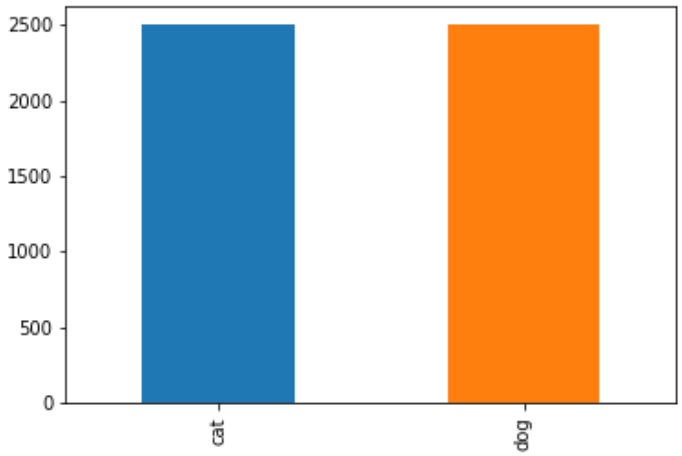
```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
```

```
                                            patience=2,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
callbacks = [earlystop, learning_rate_reduction]
```

Prepare data

```
df["category"] = df["category"].replace({0: 'cat', 1: 'dog'})
train_df, validate_df = train_test_split(df, test_size=0.20, random_state=42)
train_df = train_df.reset_index(drop=True)
validate_df = validate_df.reset_index(drop=True)
linkcode
train_df['category'].value_counts().plot.bar()
```





total_train = train_df.shape[0]
total_validate = validate_df.shape[0]
batch_size=15

Traning Generator

```
train_datagen = ImageDataGenerator(


 rotation_range=15,
    rescale=1./255,
    shear_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True,
    width_shift_range=0.1,
    height_shift_range=0.1
)
train_generator = train_datagen.flow_from_dataframe(
    train_df,
    "../input/train/train/",
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    class_mode='categorical',
    batch_size=batch_size
)
validation_datagen = ImageDataGenerator(rescale=1./255)
validation_generator = validation_datagen.flow_from_dataframe(
    validate_df,
    "../input/train/train/",
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    class_mode='categorical',
    batch_size=batch_size
)

example_df = train_df.sample(n=1).reset_index(drop=True)
example_generator = train_datagen.flow_from_dataframe(
    example_df,
    "../input/train/train/",
    x_col='filename',
    y_col='category',
    target_size=IMAGE_SIZE,
    class_mode='categorical'
)
plt.figure(figsize=(12, 12))
for i in range(0, 15):
    plt.subplot(5, 3, i+1)
    for X_batch, Y_batch in example_generator:
        image = X_batch[0]
```

```
        plt.imshow(image)
        break
plt.tight_layout()
plt.show()
```

Fit Model

```
epochs=3 if FAST_RUN else 50
history = model.fit_generator(
    train_generator,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=total_validate//batch_size,
    steps_per_epoch=total_train//batch_size,
    callbacks=callbacks
)
```

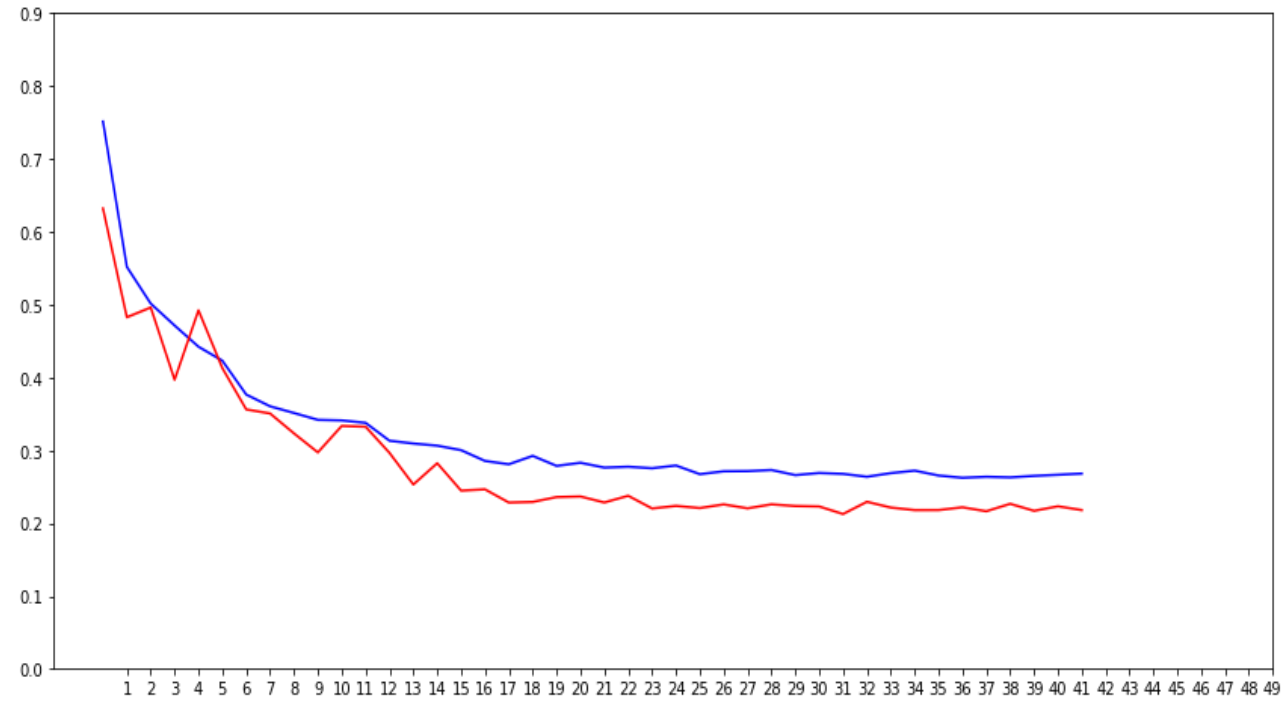
```
Epoch 1/50
1333/1333 [==============================] - 247s 186ms/step - loss: 0.7514
- acc: 0.6291 - val_loss: 0.6320 - val_acc: 0.6603
Epoch 2/50
1333/1333 [==============================] - 199s 149ms/step - loss: 0.5521
- acc: 0.7241 - val_loss: 0.4828 - val_acc: 0.7749
Epoch 3/50
1333/1333 [==============================] - 197s 148ms/step - loss: 0.5015
- acc: 0.7627 - val_loss: 0.4962 - val_acc: 0.7535
Epoch 4/50
1333/1333 [==============================] - 197s 148ms/step - loss: 0.4718
- acc: 0.7775 - val_loss: 0.3970 - val_acc: 0.8371
Epoch 5/50
1333/1333 [==============================] - 196s 147ms/step - loss: 0.4423
- acc: 0.8007 - val_loss: 0.4924 - val_acc: 0.7840
Epoch 6/50
1333/1333 [==============================] - 196s 147ms/step - loss: 0.4233
- acc: 0.8096 - val_loss: 0.4130 - val_acc: 0.8233
Epoch 00006: ReduceLROnPlateau reducing learning rate to
0.00050000000237487257.
Epoch 7/50
1333/1333 [==============================] - 195s 146ms/step - loss: 0.3770
- acc: 0.8334 - val_loss: 0.3563 - val_acc: 0.8548
Epoch 8/50
1333/1333 [==============================] - 195s 147ms/step - loss: 0.3606
- acc: 0.8439 - val_loss: 0.3507 - val_acc: 0.8552
Epoch 9/50
1309/1333 [============================>.] - ETA: 3s - loss: 0.3524 - acc:
0.8490
```

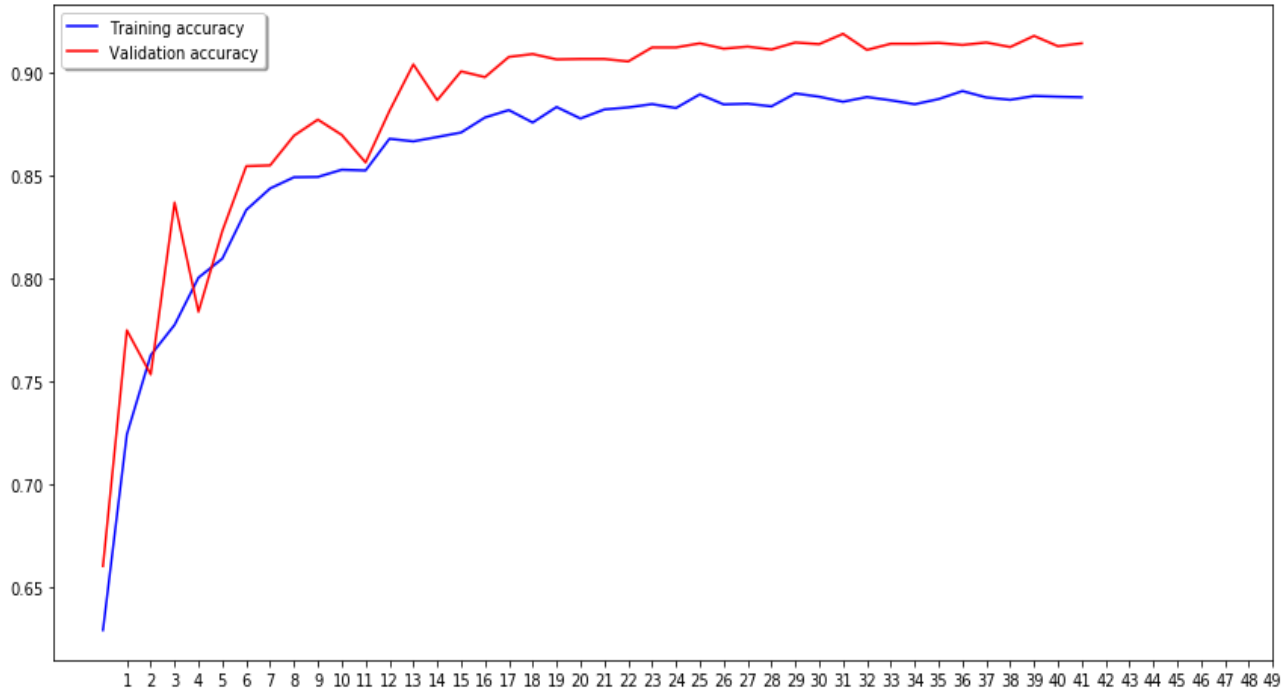
Save Model

```
model.save_weights("model.h5")
```

Virtualize Training

```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))
ax1.plot(history.history['loss'], color='b', label="Training loss")
ax1.plot(history.history['val_loss'], color='r', label="validation loss")
ax1.set_xticks(np.arange(1, epochs, 1))
ax1.set_yticks(np.arange(0, 1, 0.1))

ax2.plot(history.history['acc'], color='b', label="Training accuracy")
ax2.plot(history.history['val_acc'], color='r',label="Validation accuracy")
ax2.set_xticks(np.arange(1, epochs, 1))

legend = plt.legend(loc='best', shadow=True)
plt.tight_layout()
plt.show()
```





Prepare Testing Data

```
test_filenames = os.listdir("../input/test1/test1")
test_df = pd.DataFrame({
    'filename': test_filenames
})

nb_samples = test_df.shape[0]
```

Create Testing Generator

```
test_gen = ImageDataGenerator(rescale=1./255)
test_generator = test_gen.flow_from_dataframe(
```

```
    test_df,
    "../input/test1/test1/",
    x_col='filename',
    y_col=None,
    class_mode=None,
    target_size=IMAGE_SIZE,
    batch_size=batch_size,
    shuffle=False
)
```

```
Found 12500 images.
```

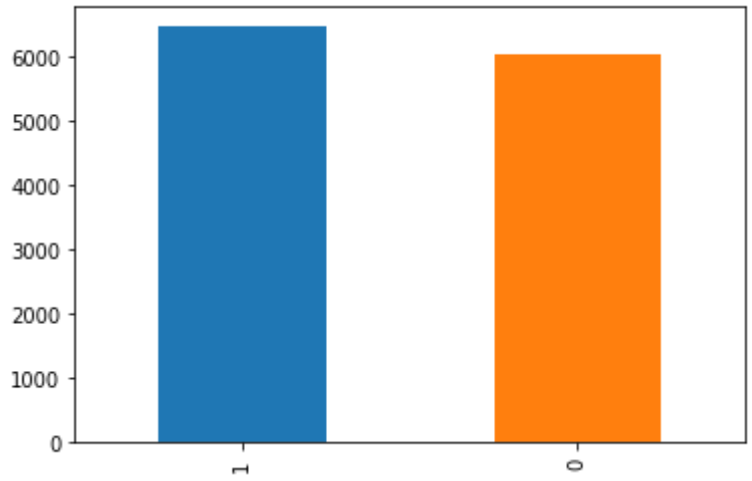
Predict

```
predict = model.predict_generator(test_generator,
steps=np.ceil(nb_samples/batch_size))
```

```
test_df['category'] = np.argmax(predict, axis=-1)
label_map = dict((v,k) for k,v in train_generator.class_indices.items())
test_df['category'] = test_df['category'].replace(label_map)
test_df['category'] = test_df['category'].replace({ 'dog': 1, 'cat': 0 })
```

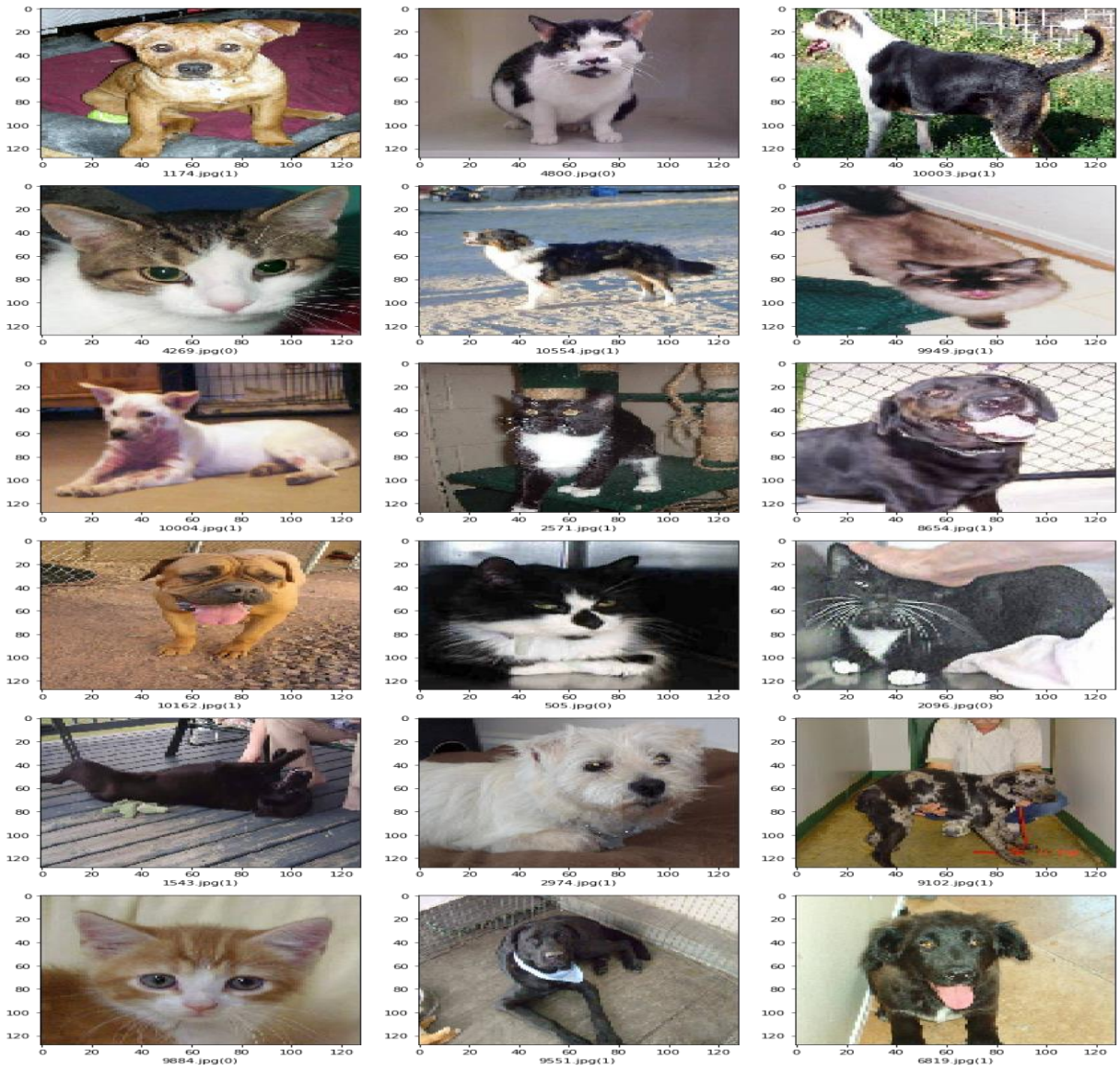
Virtaulize Result

```
test_df['category'].value_counts().plot.bar()
```




```
sample_test = test_df.head(18)
sample_test.head()
plt.figure(figsize=(12, 24))
for index, row in sample_test.iterrows():
    filename = row['filename']
    category = row['category']
    img = load_img("../input/test1/test1/"+filename, target_size=IMAGE_SIZE)
    plt.subplot(6, 3, index+1)
    plt.imshow(img)
```

```
    plt.xlabel(filename + '(' + "{}".format(category) + ')' )
plt.tight_layout()
plt.show()
```




```
submission_df = test_df.copy()
submission_df['id'] = submission_df['filename'].str.split('.').str[0]
submission_df['label'] = submission_df['category']
submission_df.drop(['filename', 'category'], axis=1, inplace=True)
submission_df.to_csv('submission.csv', index=False)
```

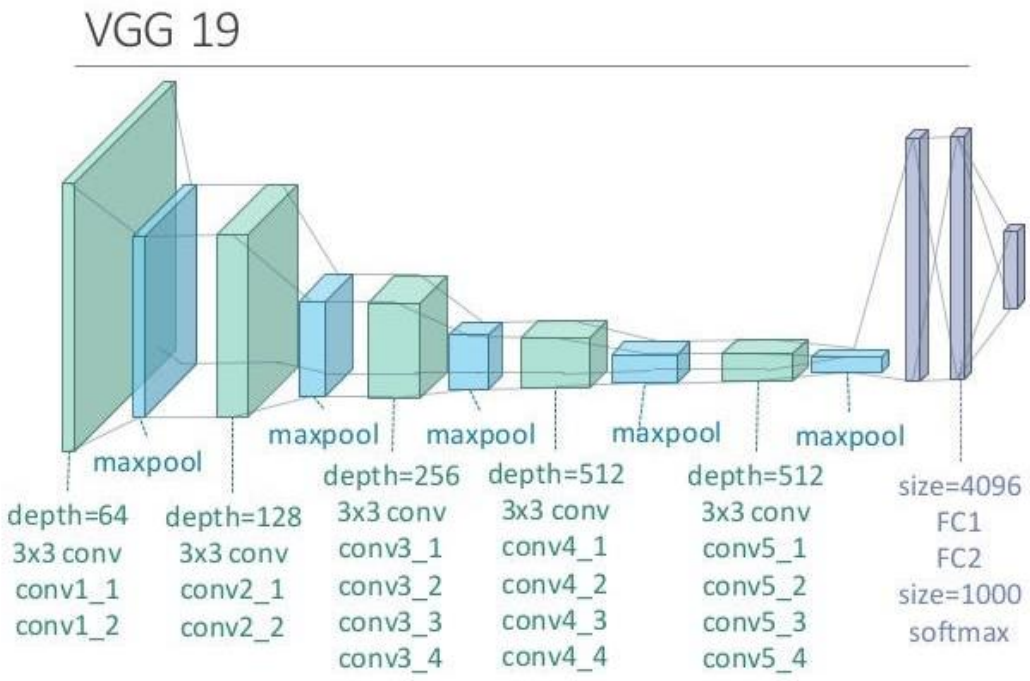
6. Program: Fine-tuning a pre-trained CNN for a specific image recognition task

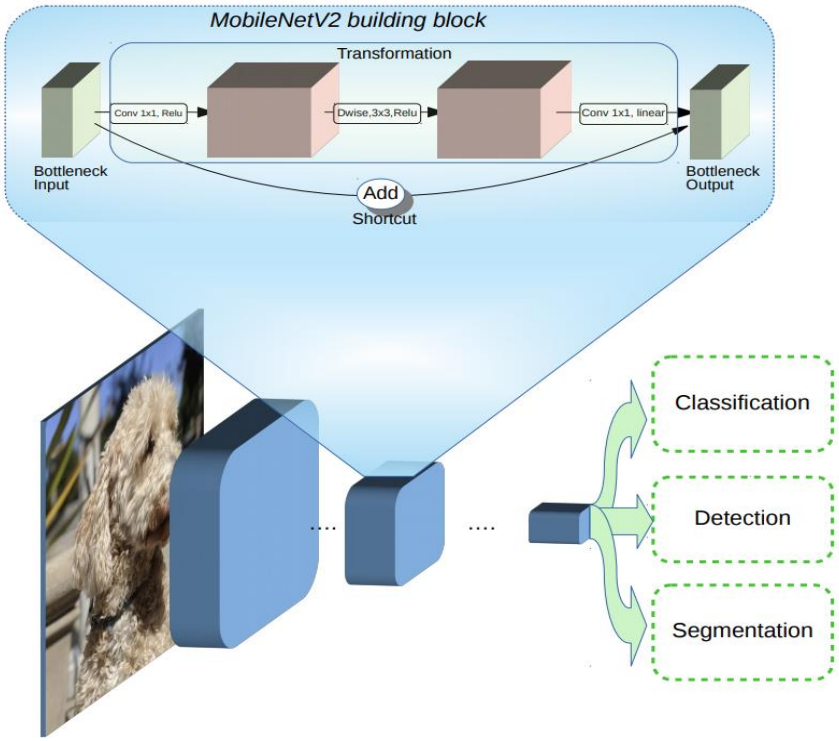
Fine-tuning a pre-trained Convolutional Neural Network (CNN) for a specific image recognition task is a common practice in deep learning. This process allows you to leverage the knowledge learned by a pre-trained model on a large dataset and adapt it to your specific task, which often requires a smaller dataset.

```
# import stuff
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns
from PIL import Image
from IPython.display import Image, display
import os
from sklearn.metrics import confusion_matrix
from sklearn.datasets import load_files
from sklearn.model_selection import train_test_split
import tensorflow_hub as hub
from keras.utils import to_categorical
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.optimizers import Adam, RMSprop, SGD
from tensorflow.keras.applications import ResNet50, VGG16, VGG19, MobileNetV2
from tensorflow.keras.applications.resnet50 import preprocess_input as prepro_res50
from tensorflow.keras.applications.vgg19 import preprocess_input as prepro_vgg19
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Flatten, GlobalAveragePooling2D,
BatchNormalization, Dropout, MaxPool2D, MaxPooling2D

# load the backend
from keras import backend as K

# prevent Tensorflow memory leakage
K.clear_session()
print(os.listdir('../input/'))
['flowers-recognition', 'digit-recognizer']

# Let's implement it
input_shape = (224, 224, 3)

my_VGG16 = Sequential([Conv2D(64, (3, 3), input_shape=input_shape,
padding='same', activation='relu'),
                       Conv2D(64, (3, 3), activation='relu', padding='same'),
```

```
MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),
                        Conv2D(128, (3, 3), activation='relu', padding='same'),
                        Conv2D(128, (3, 3), activation='relu', padding='same'),
                        MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),
                        Conv2D(256, (3, 3), activation='relu', padding='same'),
                        Conv2D(256, (3, 3), activation='relu', padding='same'),
                        Conv2D(256, (3, 3), activation='relu', padding='same'),
                        MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),
                        Conv2D(512, (3, 3), activation='relu', padding='same'),
                        Conv2D(512, (3, 3), activation='relu', padding='same'),
                        Conv2D(512, (3, 3), activation='relu', padding='same'),
                        MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),
                        Conv2D(512, (3, 3), activation='relu', padding='same'),
                        Conv2D(512, (3, 3), activation='relu', padding='same'),
                        Conv2D(512, (3, 3), activation='relu', padding='same'),
                        MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),
                        Flatten(),                          # Convert 3D matrices
into 1D vector
                        Dense(4096, activation='relu'),     # Add Fully-connected
layers
                        Dense(4096, activation='relu'),
                        Dense(1000, activation='softmax')   # Final Fully-
connected layer for predictions
                       ])

my_VGG16.summary()
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 64)      1792
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 224, 224, 64)      36928
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 64)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 128)     73856
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 112, 112, 128)     147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 56, 56, 256)       295168
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 56, 56, 256)       590080
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 56, 56, 256)       590080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 256)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 512)       1180160
```

```
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 28, 28, 512)       2359808
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 28, 28, 512)       2359808
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 512)       0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 4096)              102764544
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```





Load & explore data

path_data = '../input/flowers-recognition/flowers/flowers/'
print(os.listdir(path_data))

```
['tulip', 'daisy', 'rose', 'dandelion', 'sunflower']
```

from os.path import join

img_folders = [join(path_data, folder) for folder in os.listdir(path_data)]
list(img_folders)

```
['../input/flowers-recognition/flowers/flowers/tulip',
 '../input/flowers-recognition/flowers/flowers/daisy',
 '../input/flowers-recognition/flowers/flowers/rose',
 '../input/flowers-recognition/flowers/flowers/dandelion',
 '../input/flowers-recognition/flowers/flowers/sunflower']
```

```
data_dir = '../input/flowers-recognition/flowers/flowers/'

data = load_files(data_dir, random_state=28, shuffle=True)
X = np.array(data['filenames'])    # files location of each flower
y = np.array(data['target'])       # target label of each flower
labels = np.array(data['target_names'])

# remove eventual .pyc or .py files
```

```
pyc_file = (np.where(file==X) for file in X if file.endswith(('.pyc','.py')))
for pos in pyc_file:
    X = np.delete(X, pos)
    y = np.delete(y, pos)

print(f'Data files - {X}')
print(f'Target labels - {y}')   # numbers are corresponding to class label,
                                # we have to change them to a vector of 5 elements
print(f'Name labels - {labels}')
print(f'Number of training files : {X.shape[0]}')
```

```
Data files - ['../input/flowers-
recognition/flowers/flowers/sunflower/1022552002_2b93faf9e7_n.jpg'
 '../input/flowers-
recognition/flowers/flowers/daisy/17101762155_2577a28395.jpg'
 '../input/flowers-
recognition/flowers/flowers/dandelion/11545123_50a340b473_m.jpg'
 ...
 '../input/flowers-
recognition/flowers/flowers/dandelion/5446666484_365f3be83a_n.jpg'
 '../input/flowers-
recognition/flowers/flowers/tulip/5718512892_a175a94c45_n.jpg'
 '../input/flowers-
recognition/flowers/flowers/dandelion/3451646670_3eff7094b7_n.jpg']
Target labels - [3 0 1 ... 1 4 1]
Name labels - ['daisy' 'dandelion' 'rose' 'sunflower' 'tulip']
Number of training files : 4323
```

# Flower species number
df = pd.DataFrame({'species': y})
print(df.shape)
df.head()

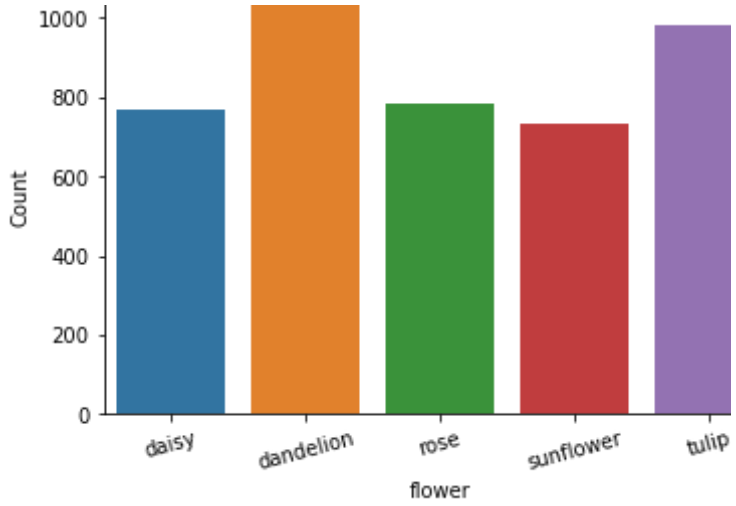
```
(4323, 1)
```

	species
0	3
1	0
2	1
3	4
4	4

associate names to species number
df['flower'] = df['species'].astype('category')
df['flower'].cat.categories = labels
df.head()

species	flower	
0	3	sunflower
1	0	daisy
2	1	dandelion
3	4	tulip
4	4	tulip

```
fig, ax = plt.subplots()
ax = sns.countplot(x="flower", data=df)
ax.set(ylabel='Count', title='Flower species distribution')
ax.tick_params(axis='x', rotation=15)
```



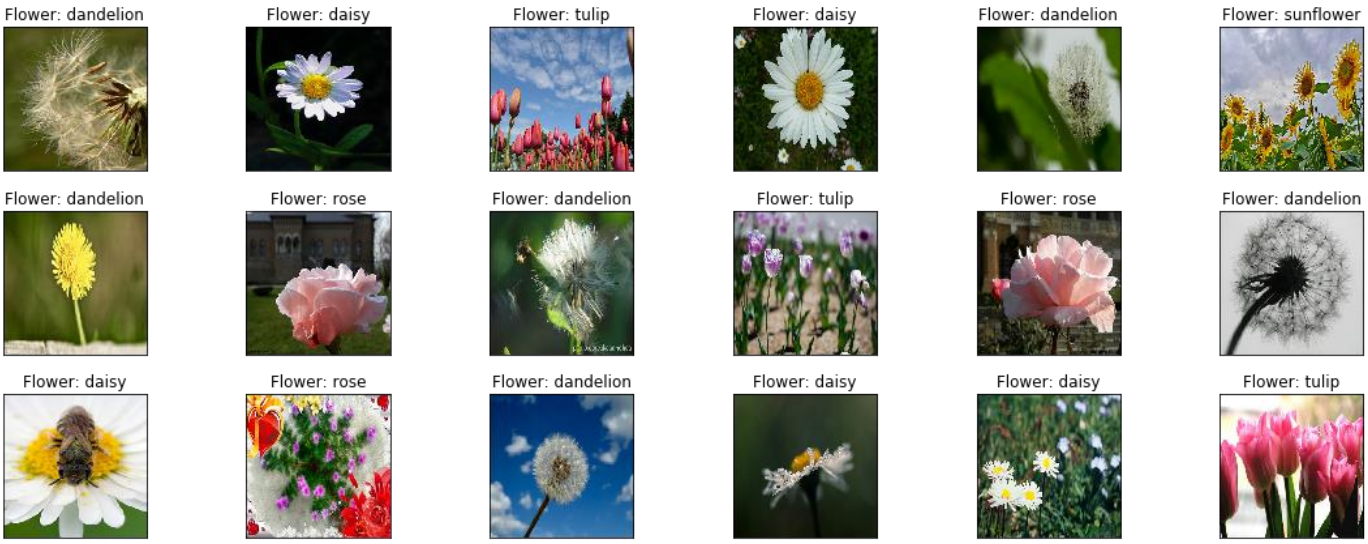

```
image_size = 224     # standard value for Transfer learning usecase (MobileNet, ResNet50, VGG16, VGG19)

def read_and_prep_images(img_paths, img_height=image_size, img_width=image_size):
    imgs = [load_img(img_path, target_size=(img_height, img_width)) for img_path in img_paths]
# load image
    img_array = np.array([img_to_array(img) for img in imgs])   # image to array
    return(img_array)

X = np.array(read_and_prep_images(X))
print(X.shape)  # (4323, 224, 224, 3) = (num_images, height_size, width_size, depth=RGB)
```

```
(4323, 224, 224, 3)
# Let's have a look at 6 randomly picked flowers.
```

In [13]:

linkcode

```
N = 18  # flowers to display
fig, axes = plt.subplots(3, 6, figsize=(16,6))
for ax, j in zip(axes.flat, np.random.randint(0, len(X), N)):
    ax.imshow(X[j].astype(np.uint8))
    ax.set(title=f'Flower: {labels[y[j]]}', xticks=[], yticks=[])
fig.tight_layout()
```



Label encoding

```
num_classes = len(np.unique(y))
print(f'Number of classes: {num_classes} --> {labels}')
```

```
Number of classes: 5 --> ['daisy' 'dandelion' 'rose' 'sunflower' 'tulip']
```

```
y = to_categorical(y, num_classes)
print(y.shape)
```

```
(4323, 5)
```

Split training and validation set

```
#train, validation and test from the train dataset
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, shuffle=True,
                                    test_size=0.25, random_state=28)

Xval, Xtest, yval, ytest = train_test_split(Xtest, ytest, test_size=0.5,
                                    shuffle=True, random_state=28)
print(f'Train dataset: {Xtrain.shape[0]}')


print(f'Validation dataset: {Xval.shape[0]}')
print(f'Test dataset: {Xtest.shape[0]}')
```

```
Train dataset: 3242
Validation dataset: 540
```

```
Test dataset: 541
```

Define the model

```
# Load the VGG19 model without the final layers (include_top=False)
img_shape = (image_size, image_size, 3)

print('Loading MobileNetV2 ...')
base_model = MobileNetV2(input_shape=img_shape,
                         include_top=False,
                         weights='imagenet')
print('MobileNetV2 loaded')

base_model.trainable = False

#base_model.summary()
```

```
Loading MobileNetV2 ...
Downloading data from
https://github.com/JonathanCMitchell/mobilenet_v2_keras/releases/download/v1
.1/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5
9412608/9406464 [==============================] - 0s 0us/step
MobileNetV2 loaded
```

```
base_model.output_shape
```

```
(None, 7, 7, 1280)
```

```
model = Sequential([base_model,
                    GlobalAveragePooling2D(),
                    Dense(num_classes, activation='softmax')
                   ])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_224 (Model) (None, 7, 7, 1280)        2257984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 6405
=================================================================



Total params: 2,264,389
Trainable params: 6,405
Non-trainable params: 2,257,984
```

```
----------------------------------------------------------------------
# callbacks
weight_path = '{}_best_weights.hdf5'.format('flower')
checkpointer = ModelCheckpoint(weight_path,
                               monitor='val_accuracy',
                               verbose=1,
                               save_best_only=True,
                               mode='auto',
                               save_weights_only=True)

# set a learning rate annealer
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.7,
                                            min_lr=0.00001)

# early stop if not improvement of accuracy after 5 epochs
early = EarlyStopping(patience=6,
                      verbose=1)

callbacks = [checkpointer, learning_rate_reduction] #, early]

# Optimizer
opt = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)

# Compilation
model.compile(optimizer=opt,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Data augmentation

```
image_size = 224
batch_size = 32
path = '../input/flowers-recognition/flowers/flowers/'

#train_gen = train_aug.flow(Xtrain, ytrain, batch_size=batch_size)
# The validation data must not have data augmentation
#valid_gen = valid_no_aug.flow(Xval, yval, batch_size=batch_size)

train_datagen = ImageDataGenerator(
        rescale=1./255,           # rescale pixel values [0,255] to [0,1]
        horizontal_flip=True,     # random horizontal flip
        width_shift_range=0.2,    # random shift images horizontally (fraction of
total width)
        height_shift_range=0.2,   # random shift images vertically (fraction of
total height)
        zoom_range=0.2)           # random zoom image
```

```
#rotation_range=20,       # random rotation
        #shear_range=0.2)         # shear transfo
        #validation_split=0.2)    # splitting train / test datasets

test_datagen = ImageDataGenerator(
        rescale=1./255)
        #validation_split=0.2)

train_gen = train_datagen.flow(
        Xtrain, ytrain,
        batch_size=batch_size,
        shuffle=False)              # already applied

valid_gen = test_datagen.flow(
        Xval, yval,
        batch_size=batch_size,
        shuffle=False)
```

Feature extraction

batch_size = 32
epochs_0 = 40
steps_per_epoch = len(train_gen.x) // train_gen.batch_size
validation_steps = len(valid_gen.x) // valid_gen.batch_size

history = model.fit(
  train_gen,
  steps_per_epoch=len(Xtrain) // batch_size, # or batch_size=32
  epochs=epochs_0 ,
  validation_data=valid_gen,
  validation_steps=len(Xval) // batch_size,
  callbacks=callbacks)

```
Train for 101 steps, validate for 16 steps
Epoch 1/40
100/101 [============================>.] - ETA: 0s - loss: 0.8650 -
accuracy: 0.6869
Epoch 00001: val_accuracy improved from -inf to 0.75195, saving model to
flower_best_weights.hdf5
101/101 [==============================] - 43s 426ms/step - loss: 0.8616 -
accuracy: 0.6882 - val_loss: 0.6295 - val_accuracy: 0.7520
Epoch 2/40
100/101 [============================>.] - ETA: 0s - loss: 0.5395 -
accuracy: 0.8037
Epoch 00002: val_accuracy did not improve from 0.75195
101/101 [==============================] - 36s 358ms/step - loss: 0.5385 -
accuracy: 0.8044 - val_loss: 0.7646 - val_accuracy: 0.7109
Epoch 3/40
```

```
100/101 [============================>.] - ETA: 0s - loss: 0.4721 -
accuracy: 0.8357
Epoch 00003: val_accuracy improved from 0.75195 to 0.79492, saving model to
flower_best_weights.hdf5
101/101 [==============================] - 37s 363ms/step - loss: 0.4719 -
accuracy: 0.8361 - val_loss: 0.5152 - val_accuracy: 0.7949
Epoch 4/40
100/101 [============================>.] - ETA: 0s - loss: 0.4363 -
accuracy: 0.8531
Epoch 00004: val_accuracy did not improve from 0.79492
101/101 [==============================] - 36s 356ms/step - loss: 0.4339 -
accuracy: 0.8539 - val_loss: 0.6437 - val_accuracy: 0.7461
Epoch 5/40
100/101 [============================>.] - ETA: 0s - loss: 0.4123 -
accuracy: 0.8540
Epoch 00005: val_accuracy did not improve from 0.79492
101/101 [==============================] - 37s 364ms/step - loss: 0.4111 -
accuracy: 0.8542 - val_loss: 0.5558 - val_accuracy: 0.7832
Epoch 6/40
100/101 [============================>.] - ETA: 0s - loss: 0.3909 -
accuracy: 0.8656
Epoch 00006: val_accuracy did not improve from 0.79492
Epoch 00006: ReduceLROnPlateau reducing learning rate to
0.0007000000332482159.
101/101 [==============================] - 36s 359ms/step - loss: 0.3896 -
accuracy: 0.8660 - val_loss: 0.5382 - val_accuracy: 0.7773
Epoch 7/40
100/101 [============================>.] - ETA: 0s - loss: 0.3655 -
accuracy: 0.8754
Epoch 00007: val_accuracy improved from 0.79492 to 0.79883, saving model to
flower_best_weights.hdf5
101/101 [==============================] - 37s 363ms/step - loss: 0.3649 -
accuracy: 0.8757 - val_loss: 0.5021 - val_accuracy: 0.7988
Epoch 8/40
100/101 [============================>.] - ETA: 0s - loss: 0.3448 -
accuracy: 0.8770
Epoch 00008: val_accuracy did not improve from 0.79883
101/101 [==============================] - 37s 363ms/step - loss: 0.3438 -
accuracy: 0.8773 - val_loss: 0.5642 - val_accuracy: 0.7754
Epoch 9/40
100/101 [============================>.] - ETA: 0s - loss: 0.3396 -
accuracy: 0.8814
Epoch 00009: val_accuracy did not improve from 0.79883
101/101 [==============================] - 36s 352ms/step - loss: 0.3393 -
accuracy: 0.8813 - val_loss: 0.5414 - val_accuracy: 0.7852
Epoch 10/40
100/101 [============================>.] - ETA: 0s - loss: 0.3365 -
accuracy: 0.8870
Epoch 00010: val_accuracy did not improve from 0.79883
```

```
Epoch 00010: ReduceLROnPlateau reducing learning rate to
0.0004900000232737511.
101/101 [==============================] - 37s 371ms/step - loss: 0.3367 -
accuracy: 0.8875 - val_loss: 0.5881 - val_accuracy: 0.7734
Epoch 11/40
100/101 [============================>.] - ETA: 0s - loss: 0.3310 -
accuracy: 0.8861
Epoch 00011: val_accuracy did not improve from 0.79883
101/101 [==============================] - 35s 349ms/step - loss: 0.3302 -
accuracy: 0.8863 - val_loss: 0.5418 - val_accuracy: 0.7910
Epoch 12/40
100/101 [============================>.] - ETA: 0s - loss: 0.3198 -
accuracy: 0.8902
Epoch 00012: val_accuracy did not improve from 0.79883
101/101 [==============================] - 37s 361ms/step - loss: 0.3222 -
accuracy: 0.8891 - val_loss: 0.5605 - val_accuracy: 0.7930
Epoch 13/40
100/101 [============================>.] - ETA: 0s - loss: 0.3161 -
accuracy: 0.8949
Epoch 00013: val_accuracy did not improve from 0.79883
Epoch 00013: ReduceLROnPlateau reducing learning rate to
0.00034300000406801696.
101/101 [==============================] - 36s 354ms/step - loss: 0.3139 -
accuracy: 0.8960 - val_loss: 0.5485 - val_accuracy: 0.7910
Epoch 14/40
100/101 [============================>.] - ETA: 0s - loss: 0.3065 -
accuracy: 0.8962
Epoch 00014: val_accuracy did not improve from 0.79883
101/101 [==============================] - 36s 355ms/step - loss: 0.3059 -
accuracy: 0.8963 - val_loss: 0.5536 - val_accuracy: 0.7910
Epoch 15/40
100/101 [============================>.] - ETA: 0s - loss: 0.3029 -
accuracy: 0.8908
Epoch 00015: val_accuracy did not improve from 0.79883
101/101 [==============================] - 37s 365ms/step - loss: 0.3050 -
accuracy: 0.8903 - val_loss: 0.5360 - val_accuracy: 0.7949
Epoch 16/40
100/101 [============================>.] - ETA: 0s - loss: 0.3069 -
accuracy: 0.8927
Epoch 00016: val_accuracy improved from 0.79883 to 0.80078, saving model to
flower_best_weights.hdf5
101/101 [==============================] - 35s 351ms/step - loss: 0.3063 -
accuracy: 0.8925 - val_loss: 0.5310 - val_accuracy: 0.8008
Epoch 17/40
100/101 [============================>.] - ETA: 0s - loss: 0.2957 -
accuracy: 0.9047
Epoch 00017: val_accuracy improved from 0.80078 to 0.80664, saving model to
flower_best_weights.hdf5
101/101 [==============================] - 37s 369ms/step - loss: 0.3027 -
accuracy: 0.9025 - val_loss: 0.5082 - val_accuracy: 0.8066
Epoch 18/40
```

```
100/101 [============================>.] - ETA: 0s - loss: 0.2911 -
accuracy: 0.8971
Epoch 00018: val_accuracy did not improve from 0.80664
101/101 [==============================] - 37s 365ms/step - loss: 0.2914 -
accuracy: 0.8975 - val_loss: 0.5112 - val_accuracy: 0.8047
Epoch 19/40
100/101 [============================>.] - ETA: 0s - loss: 0.2933 -
accuracy: 0.9021
Epoch 00019: val_accuracy did not improve from 0.80664
101/101 [==============================] - 36s 353ms/step - loss: 0.2927 -
accuracy: 0.9019 - val_loss: 0.4960 - val_accuracy: 0.8047
Epoch 20/40
100/101 [============================>.] - ETA: 0s - loss: 0.2932 -
accuracy: 0.8984
Epoch 00020: val_accuracy did not improve from 0.80664

Epoch 00020: ReduceLROnPlateau reducing learning rate to
0.00024009999469853935.
101/101 [==============================] - 36s 358ms/step - loss: 0.2917 -
accuracy: 0.8988 - val_loss: 0.5484 - val_accuracy: 0.8008
Epoch 21/40
100/101 [============================>.] - ETA: 0s - loss: 0.2876 -
accuracy: 0.9021
Epoch 00021: val_accuracy improved from 0.80664 to 0.81641, saving model to
flower_best_weights.hdf5
101/101 [==============================] - 37s 369ms/step - loss: 0.2856 -
accuracy: 0.9031 - val_loss: 0.4939 - val_accuracy: 0.8164
Epoch 22/40
100/101 [============================>.] - ETA: 0s - loss: 0.2807 -
accuracy: 0.9053
Epoch 00022: val_accuracy did not improve from 0.81641
101/101 [==============================] - 37s 363ms/step - loss: 0.2813 -
accuracy: 0.9044 - val_loss: 0.5274 - val_accuracy: 0.8125
Epoch 23/40
100/101 [============================>.] - ETA: 0s - loss: 0.2834 -
accuracy: 0.9047
Epoch 00023: val_accuracy did not improve from 0.81641
101/101 [==============================] - 36s 355ms/step - loss: 0.2846 -
accuracy: 0.9040 - val_loss: 0.5534 - val_accuracy: 0.8008
Epoch 24/40
100/101 [============================>.] - ETA: 0s - loss: 0.2840 -
accuracy: 0.9025
Epoch 00024: val_accuracy did not improve from 0.81641
Epoch 00024: ReduceLROnPlateau reducing learning rate to
0.00016806999628897755.
101/101 [==============================] - 35s 350ms/step - loss: 0.2832 -
accuracy: 0.9028 - val_loss: 0.5279 - val_accuracy: 0.8008
Epoch 25/40
100/101 [============================>.] - ETA: 0s - loss: 0.2799 -
accuracy: 0.9012
Epoch 00025: val_accuracy did not improve from 0.81641
```

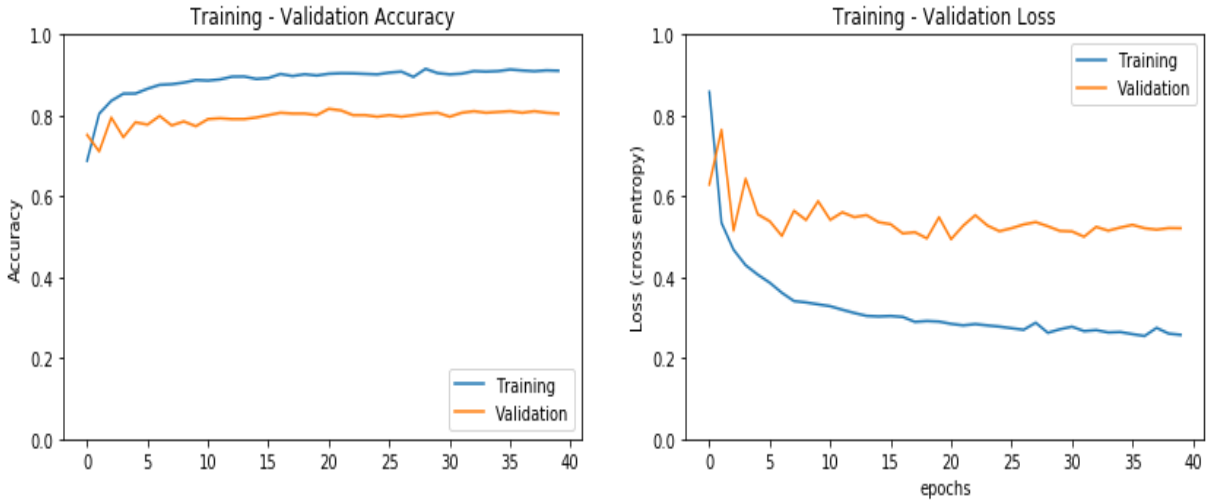
```
101/101 [==============================] - 36s 360ms/step - loss: 0.2790 -
accuracy: 0.9016 - val_loss: 0.5138 - val_accuracy: 0.7969
Epoch 26/40
100/101 [============================>.] - ETA: 0s - loss: 0.2755 -
accuracy: 0.9062
Epoch 00026: val_accuracy did not improve from 0.81641
101/101 [==============================] - 35s 349ms/step - loss: 0.2751 -
accuracy: 0.9059 - val_loss: 0.5211 - val_accuracy: 0.8008
Epoch 27/40
100/101 [============================>.] - ETA: 0s - loss: 0.2724 -
accuracy: 0.9078
Epoch 00027: val_accuracy did not improve from 0.81641
Epoch 00027: ReduceLROnPlateau reducing learning rate to
0.00011764899536501615.
101/101 [==============================] - 36s 353ms/step - loss: 0.2713 -
accuracy: 0.9084 - val_loss: 0.5304 - val_accuracy: 0.7969
Epoch 28/40
100/101 [============================>.] - ETA: 0s - loss: 0.2872 -
accuracy: 0.8962
Epoch 00028: val_accuracy did not improve from 0.81641
101/101 [==============================] - 37s 369ms/step - loss: 0.2897 -
accuracy: 0.8953 - val_loss: 0.5362 - val_accuracy: 0.8008
Epoch 29/40
100/101 [============================>.] - ETA: 0s - loss: 0.2608 -
accuracy: 0.9157
Epoch 00029: val_accuracy did not improve from 0.81641
101/101 [==============================] - 36s 355ms/step - loss: 0.2627 -
accuracy: 0.9153 - val_loss: 0.5260 - val_accuracy: 0.8047
Epoch 30/40
100/101 [============================>.] - ETA: 0s - loss: 0.2729 -
accuracy: 0.9050
Epoch 00030: val_accuracy did not improve from 0.81641
Epoch 00030: ReduceLROnPlateau reducing learning rate to 8.235429777414538e-
05.
101/101 [==============================] - 37s 363ms/step - loss: 0.2741 -
accuracy: 0.9047 - val_loss: 0.5145 - val_accuracy: 0.8066
Epoch 31/40
100/101 [============================>.] - ETA: 0s - loss: 0.2807 -
accuracy: 0.9018
Epoch 00031: val_accuracy did not improve from 0.81641
101/101 [==============================] - 36s 354ms/step - loss: 0.2806 -
accuracy: 0.9016 - val_loss: 0.5132 - val_accuracy: 0.7969
Epoch 32/40
100/101 [============================>.] - ETA: 0s - loss: 0.2705 -
accuracy: 0.9028
Epoch 00032: val_accuracy did not improve from 0.81641
101/101 [==============================] - 36s 361ms/step - loss: 0.2690 -
accuracy: 0.9034 - val_loss: 0.5001 - val_accuracy: 0.8066
Epoch 33/40
```

```
100/101 [============================>.] - ETA: 0s - loss: 0.2730 -
accuracy: 0.9087
Epoch 00033: val_accuracy did not improve from 0.81641
Epoch 00033: ReduceLROnPlateau reducing learning rate to 5.76480058953166e-
05.
101/101 [==============================] - 36s 361ms/step - loss: 0.2713 -
accuracy: 0.9097 - val_loss: 0.5245 - val_accuracy: 0.8105
Epoch 34/40
100/101 [============================>.] - ETA: 0s - loss: 0.2639 -
accuracy: 0.9084
Epoch 00034: val_accuracy did not improve from 0.81641
101/101 [==============================] - 36s 355ms/step - loss: 0.2640 -
accuracy: 0.9084 - val_loss: 0.5151 - val_accuracy: 0.8066
Epoch 35/40
100/101 [============================>.] - ETA: 0s - loss: 0.2661 -
accuracy: 0.9094
Epoch 00035: val_accuracy did not improve from 0.81641
101/101 [==============================] - 37s 371ms/step - loss: 0.2669 -
accuracy: 0.9097 - val_loss: 0.5228 - val_accuracy: 0.8086
Epoch 36/40
100/101 [============================>.] - ETA: 0s - loss: 0.2593 -
accuracy: 0.9135
Epoch 00036: val_accuracy did not improve from 0.81641
Epoch 00036: ReduceLROnPlateau reducing learning rate to
4.0353603617404586e-05.
101/101 [==============================] - 37s 365ms/step - loss: 0.2594 -
accuracy: 0.9137 - val_loss: 0.5295 - val_accuracy: 0.8105
Epoch 37/40
100/101 [============================>.] - ETA: 0s - loss: 0.2534 -
accuracy: 0.9116
Epoch 00037: val_accuracy did not improve from 0.81641
101/101 [==============================] - 36s 360ms/step - loss: 0.2550 -
accuracy: 0.9112 - val_loss: 0.5212 - val_accuracy: 0.8066
Epoch 38/40
100/101 [============================>.] - ETA: 0s - loss: 0.2766 -
accuracy: 0.9097
Epoch 00038: val_accuracy did not improve from 0.81641
101/101 [==============================] - 37s 366ms/step - loss: 0.2772 -
accuracy: 0.9093 - val_loss: 0.5177 - val_accuracy: 0.8105
Epoch 39/40
100/101 [============================>.] - ETA: 0s - loss: 0.2614 -
accuracy: 0.9116
Epoch 00039: val_accuracy did not improve from 0.81641
Epoch 00039: ReduceLROnPlateau reducing learning rate to
2.8247522277524694e-05.
101/101 [==============================] - 36s 360ms/step - loss: 0.2611 -
accuracy: 0.9115 - val_loss: 0.5215 - val_accuracy: 0.8066
Epoch 40/40
```

```
100/101 [============================>.] - ETA: 0s - loss: 0.2604 -
accuracy: 0.9103
Epoch 00040: val_accuracy did not improve from 0.81641
101/101 [==============================] - 37s 367ms/step - loss: 0.2607 -
accuracy: 0.9106 - val_loss: 0.5211 - val_accuracy: 0.8047
```

```
def plot_history(history, loss_max=5):
    """
    Check loss and accuracy evolution.
    """
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    fig, (ax1, ax2) = plt.subplots(1,2,figsize=(14, 4))
    ax1.plot(acc, label='Training')
    ax1.plot(val_acc, label='Validation')
    ax1.legend(loc='lower right')
    ax1.set(ylabel='Accuracy', title='Training - Validation Accuracy',
            ylim=([min(plt.ylim()),1]))
    ax2.plot(loss, label='Training')
    ax2.plot(val_loss, label='Validation')
    ax2.legend(loc='upper right')
    ax2.set(ylabel='Loss (cross entropy)', xlabel='epochs',
           title='Training - Validation Loss', ylim=([0, loss_max]))
    plt.show()
```

```
plot_history(history, loss_max=1)
```




```
# Generator for test dataset
datagen = ImageDataGenerator(
        rescale=1./255)
```

```
eval_datagen = datagen.flow(
        Xtest, ytest,
        batch_size=batch_size,
        shuffle=False)      # since shuffle was already during splitting into train, valid, test
# Evaluation on the test dataset
loss, acc = model.evaluate_generator(eval_datagen, verbose=0)
print(f'Test loss: {loss:.2f}')
print(f'Test accuracy: {acc*100:.2f}%')
```

```
Test loss: 0.62
Test accuracy: 78.00%
```

Fine tuning

```
base_model.trainable = True
# Let's take a look to see how many layers are in the base model
print(f'Number of layers in the base model: {len(base_model.layers)}')
```

```
Number of layers in the base model: 155
```

```
# Fine-tune from this layer onwards
fine_tuning = 100

# Freeze all the layers before fine_tuned_ind
for layer in base_model.layers[:fine_tuning]:
    layer.trainable =  False
# Load best weights
# model.load_weights(weight_path)

# Finer learning rate now
opt = RMSprop(lr=0.0001, rho=0.9, epsilon=1e-08, decay=0.0)

# Compilation
model.compile(optimizer=opt,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_224 (Model) (None, 7, 7, 1280)        2257984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 6405
=================================================================
Total params: 2,264,389
Trainable params: 1,868,997
Non-trainable params: 395,392
_________________________________________________________________
```

```
fine_tuned_epochs = 40

total_epochs = epochs_0 + fine_tuned_epochs
history_fined = model.fit_generator(
    train_gen,
    steps_per_epoch=len(Xtrain) // batch_size,   # or batch_size=32
    epochs=total_epochs,
    initial_epoch=history.epoch[-1],
    validation_data=valid_gen,
    validation_steps=len(Xval) // batch_size,
    callbacks=callbacks)
Train for 101 steps, validate for 16 steps
Epoch 40/80
100/101 [============================>.] - ETA: 0s - loss: 0.3129 -
accuracy: 0.8839
Epoch 00040: val_accuracy improved from 0.81641 to 0.86328, saving model to
flower_best_weights.hdf5
101/101 [==============================] - 44s 431ms/step - loss: 0.3124 -
accuracy: 0.8835 - val_loss: 0.4331 - val_accuracy: 0.8633
Epoch 41/80
100/101 [============================>.] - ETA: 0s - loss: 0.1710 -
accuracy: 0.9383
Epoch 00041: val_accuracy did not improve from 0.86328
101/101 [==============================] - 36s 359ms/step - loss: 0.1700 -
accuracy: 0.9386 - val_loss: 0.6918 - val_accuracy: 0.8242
Epoch 42/80
100/101 [============================>.] - ETA: 0s - loss: 0.0891 -
accuracy: 0.9676
Epoch 00042: val_accuracy did not improve from 0.86328
101/101 [==============================] - 37s 367ms/step - loss: 0.0896 -
accuracy: 0.9673 - val_loss: 0.8149 - val_accuracy: 0.8027
Epoch 43/80
100/101 [============================>.] - ETA: 0s - loss: 0.0633 -
accuracy: 0.9816
Epoch 00043: val_accuracy did not improve from 0.86328
Epoch 00043: ReduceLROnPlateau reducing learning rate to 6.999999823165126e-
05.
101/101 [==============================] - 37s 362ms/step - loss: 0.0629 -
accuracy: 0.9816 - val_loss: 1.0869 - val_accuracy: 0.7832
Epoch 44/80
100/101 [============================>.] - ETA: 0s - loss: 0.0356 -
accuracy: 0.9909
Epoch 00044: val_accuracy did not improve from 0.86328
101/101 [==============================] - 36s 353ms/step - loss: 0.0355 -
accuracy: 0.9910 - val_loss: 0.8453 - val_accuracy: 0.8086
Epoch 45/80
100/101 [============================>.] - ETA: 0s - loss: 0.0295 -
accuracy: 0.9921
Epoch 00045: val_accuracy did not improve from 0.86328
```

```
101/101 [==============================] - 37s 363ms/step - loss: 0.0300 -
accuracy: 0.9919 - val_loss: 0.6663 - val_accuracy: 0.8359
Epoch 46/80
100/101 [============================>.] - ETA: 0s - loss: 0.0186 -
accuracy: 0.9953
Epoch 00046: val_accuracy did not improve from 0.86328
Epoch 00046: ReduceLROnPlateau reducing learning rate to 4.899999621557071e-
05.
101/101 [==============================] - 35s 348ms/step - loss: 0.0187 -
accuracy: 0.9953 - val_loss: 0.7875 - val_accuracy: 0.8223
Epoch 47/80
100/101 [============================>.] - ETA: 0s - loss: 0.0139 -
accuracy: 0.9959
Epoch 00047: val_accuracy improved from 0.86328 to 0.87500, saving model to
flower_best_weights.hdf5
101/101 [==============================] - 37s 367ms/step - loss: 0.0143 -
accuracy: 0.9956 - val_loss: 0.5653 - val_accuracy: 0.8750
Epoch 48/80
100/101 [============================>.] - ETA: 0s - loss: 0.0097 -
accuracy: 0.9975
Epoch 00048: val_accuracy did not improve from 0.87500
101/101 [==============================] - 37s 362ms/step - loss: 0.0096 -
accuracy: 0.9975 - val_loss: 0.5745 - val_accuracy: 0.8730
Epoch 49/80
100/101 [============================>.] - ETA: 0s - loss: 0.0086 -
accuracy: 0.9978
Epoch 00049: val_accuracy did not improve from 0.87500
101/101 [==============================] - 35s 350ms/step - loss: 0.0085 -
accuracy: 0.9978 - val_loss: 0.7023 - val_accuracy: 0.8633
Epoch 50/80
100/101 [============================>.] - ETA: 0s - loss: 0.0058 -
accuracy: 0.9987
Epoch 00050: val_accuracy did not improve from 0.87500
Epoch 00050: ReduceLROnPlateau reducing learning rate to
3.4299996332265434e-05.
101/101 [==============================] - 36s 359ms/step - loss: 0.0058 -
accuracy: 0.9988 - val_loss: 0.7334 - val_accuracy: 0.8672
Epoch 51/80
100/101 [============================>.] - ETA: 0s - loss: 0.0062 -
accuracy: 0.9978
Epoch 00051: val_accuracy did not improve from 0.87500
101/101 [==============================] - 36s 360ms/step - loss: 0.0062 -
accuracy: 0.9978 - val_loss: 0.7350 - val_accuracy: 0.8691
Epoch 52/80
100/101 [============================>.] - ETA: 0s - loss: 0.0042 -
accuracy: 0.9991
Epoch 00052: val_accuracy improved from 0.87500 to 0.88281, saving model to
flower_best_weights.hdf5
101/101 [==============================] - 37s 368ms/step - loss: 0.0042 -
accuracy: 0.9991 - val_loss: 0.5849 - val_accuracy: 0.8828
Epoch 53/80
```

```
100/101 [============================>.] - ETA: 0s - loss: 0.0041 -
accuracy: 0.9987
Epoch 00053: val_accuracy did not improve from 0.88281
101/101 [==============================] - 38s 374ms/step - loss: 0.0040 -
accuracy: 0.9988 - val_loss: 0.6574 - val_accuracy: 0.8770
Epoch 54/80
100/101 [============================>.] - ETA: 0s - loss: 0.0025 -
accuracy: 0.9997
Epoch 00054: val_accuracy improved from 0.88281 to 0.88477, saving model to
flower_best_weights.hdf5
101/101 [==============================] - 37s 361ms/step - loss: 0.0025 -
accuracy: 0.9997 - val_loss: 0.5772 - val_accuracy: 0.8848
Epoch 55/80
100/101 [============================>.] - ETA: 0s - loss: 0.0046 -
accuracy: 0.9987
Epoch 00055: val_accuracy did not improve from 0.88477
101/101 [==============================] - 36s 361ms/step - loss: 0.0045 -
accuracy: 0.9988 - val_loss: 0.6961 - val_accuracy: 0.8750
Epoch 56/80
100/101 [============================>.] - ETA: 0s - loss: 0.0027 -
accuracy: 0.9994
Epoch 00056: val_accuracy did not improve from 0.88477
101/101 [==============================] - 35s 351ms/step - loss: 0.0027 -
accuracy: 0.9994 - val_loss: 0.6988 - val_accuracy: 0.8789
Epoch 57/80
100/101 [============================>.] - ETA: 0s - loss: 0.0029 -
accuracy: 0.9997
Epoch 00057: val_accuracy did not improve from 0.88477
Epoch 00057: ReduceLROnPlateau reducing learning rate to 2.400999692326877e-
05.
101/101 [==============================] - 37s 362ms/step - loss: 0.0029 -
accuracy: 0.9997 - val_loss: 0.7334 - val_accuracy: 0.8711
Epoch 58/80
100/101 [============================>.] - ETA: 0s - loss: 0.0021 -
accuracy: 0.9997
Epoch 00058: val_accuracy improved from 0.88477 to 0.89062, saving model to
flower_best_weights.hdf5
101/101 [==============================] - 36s 361ms/step - loss: 0.0020 -
accuracy: 0.9997 - val_loss: 0.6130 - val_accuracy: 0.8906
Epoch 59/80
100/101 [============================>.] - ETA: 0s - loss: 0.0018 -
accuracy: 0.9997
Epoch 00059: val_accuracy improved from 0.89062 to 0.89258, saving model to
flower_best_weights.hdf5
101/101 [==============================] - 36s 354ms/step - loss: 0.0018 -
accuracy: 0.9997 - val_loss: 0.6082 - val_accuracy: 0.8926
Epoch 60/80
100/101 [============================>.] - ETA: 0s - loss: 0.0017 -
accuracy: 0.9994
Epoch 00060: val_accuracy did not improve from 0.89258
101/101 [==============================] - 37s 361ms/step - loss: 0.0018 -
accuracy: 0.9994 - val_loss: 0.6781 - val_accuracy: 0.8730
```

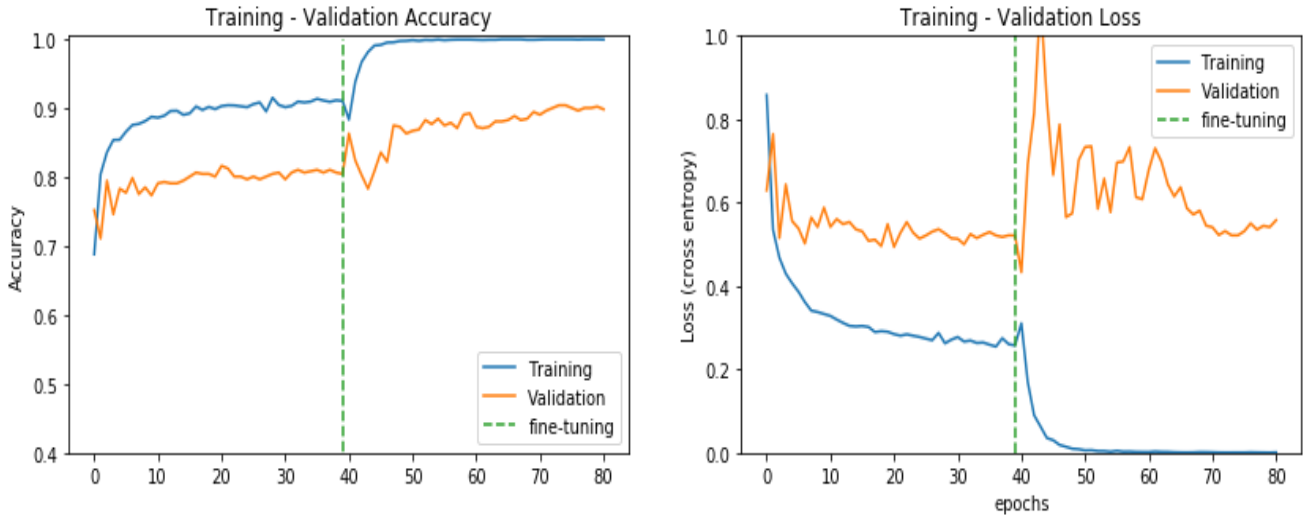
```
Epoch 61/80
100/101 [============================>.] - ETA: 0s - loss: 0.0029 -
accuracy: 0.9991
Epoch 00061: val_accuracy did not improve from 0.89258
101/101 [==============================] - 36s 358ms/step - loss: 0.0029 -
accuracy: 0.9991 - val_loss: 0.7306 - val_accuracy: 0.8711
Epoch 62/80
100/101 [============================>.] - ETA: 0s - loss: 0.0021 -
accuracy: 0.9994
Epoch 00062: val_accuracy did not improve from 0.89258
Epoch 00062: ReduceLROnPlateau reducing learning rate to
1.6806997336971108e-05.
101/101 [==============================] - 36s 355ms/step - loss: 0.0021 -
accuracy: 0.9994 - val_loss: 0.6980 - val_accuracy: 0.8730
Epoch 63/80
100/101 [============================>.] - ETA: 0s - loss: 0.0020 -
accuracy: 0.9994
Epoch 00063: val_accuracy did not improve from 0.89258
101/101 [==============================] - 37s 367ms/step - loss: 0.0020 -
accuracy: 0.9994 - val_loss: 0.6438 - val_accuracy: 0.8809
Epoch 64/80
100/101 [============================>.] - ETA: 0s - loss: 0.0012 -
accuracy: 1.0000
Epoch 00064: val_accuracy did not improve from 0.89258
101/101 [==============================] - 35s 348ms/step - loss: 0.0012 -
accuracy: 1.0000 - val_loss: 0.6148 - val_accuracy: 0.8809
Epoch 65/80
100/101 [============================>.] - ETA: 0s - loss: 7.6357e-04 -
accuracy: 1.0000
Epoch 00065: val_accuracy did not improve from 0.89258
Epoch 00065: ReduceLROnPlateau reducing learning rate to
1.1764897499233484e-05.
101/101 [==============================] - 36s 355ms/step - loss: 7.5680e-04
- accuracy: 1.0000 - val_loss: 0.6370 - val_accuracy: 0.8828
Epoch 66/80
100/101 [============================>.] - ETA: 0s - loss: 6.3722e-04 -
accuracy: 1.0000
Epoch 00066: val_accuracy did not improve from 0.89258
101/101 [==============================] - 36s 357ms/step - loss: 6.3511e-04
- accuracy: 1.0000 - val_loss: 0.5862 - val_accuracy: 0.8887
Epoch 67/80
100/101 [============================>.] - ETA: 0s - loss: 8.4205e-04 -
accuracy: 1.0000
Epoch 00067: val_accuracy did not improve from 0.89258
101/101 [==============================] - 36s 358ms/step - loss: 8.4151e-04
- accuracy: 1.0000 - val_loss: 0.5716 - val_accuracy: 0.8828
Epoch 68/80
100/101 [============================>.] - ETA: 0s - loss: 0.0016 -
accuracy: 0.9994
Epoch 00068: val_accuracy did not improve from 0.89258
Epoch 00068: ReduceLROnPlateau reducing learning rate to 1e-05.
```

```
101/101 [==============================] - 36s 358ms/step - loss: 0.0016 -
accuracy: 0.9994 - val_loss: 0.5808 - val_accuracy: 0.8848
Epoch 69/80
100/101 [============================>.] - ETA: 0s - loss: 0.0015 -
accuracy: 0.9994
Epoch 00069: val_accuracy improved from 0.89258 to 0.89453, saving model to
flower_best_weights.hdf5
101/101 [==============================] - 35s 349ms/step - loss: 0.0015 -
accuracy: 0.9994 - val_loss: 0.5449 - val_accuracy: 0.8945
Epoch 70/80
100/101 [============================>.] - ETA: 0s - loss: 0.0011 -
accuracy: 0.9997
Epoch 00070: val_accuracy did not improve from 0.89453
101/101 [==============================] - 37s 364ms/step - loss: 0.0011 -
accuracy: 0.9997 - val_loss: 0.5409 - val_accuracy: 0.8906
Epoch 71/80
100/101 [============================>.] - ETA: 0s - loss: 6.2025e-04 -
accuracy: 1.0000
Epoch 00071: val_accuracy improved from 0.89453 to 0.89648, saving model to
flower_best_weights.hdf5
101/101 [==============================] - 37s 370ms/step - loss: 6.2044e-04
- accuracy: 1.0000 - val_loss: 0.5222 - val_accuracy: 0.8965
Epoch 72/80
100/101 [============================>.] - ETA: 0s - loss: 5.0926e-04 -
accuracy: 1.0000
Epoch 00072: val_accuracy improved from 0.89648 to 0.90039, saving model to
flower_best_weights.hdf5
101/101 [==============================] - 36s 354ms/step - loss: 5.0715e-04
- accuracy: 1.0000 - val_loss: 0.5316 - val_accuracy: 0.9004
Epoch 73/80
100/101 [============================>.] - ETA: 0s - loss: 5.3943e-04 -
accuracy: 1.0000
Epoch 00073: val_accuracy improved from 0.90039 to 0.90430, saving model to
flower_best_weights.hdf5
101/101 [==============================] - 37s 365ms/step - loss: 5.3627e-04
- accuracy: 1.0000 - val_loss: 0.5216 - val_accuracy: 0.9043
Epoch 74/80
100/101 [============================>.] - ETA: 0s - loss: 5.8440e-04 -
accuracy: 1.0000
Epoch 00074: val_accuracy did not improve from 0.90430
101/101 [==============================] - 35s 348ms/step - loss: 5.7943e-04
- accuracy: 1.0000 - val_loss: 0.5215 - val_accuracy: 0.9043
Epoch 75/80
100/101 [============================>.] - ETA: 0s - loss: 5.4420e-04 -
accuracy: 1.0000
Epoch 00075: val_accuracy did not improve from 0.90430
101/101 [==============================] - 37s 367ms/step - loss: 5.4117e-04
- accuracy: 1.0000 - val_loss: 0.5313 - val_accuracy: 0.9004
Epoch 76/80
100/101 [============================>.] - ETA: 0s - loss: 0.0011 -
accuracy: 0.9997
Epoch 00076: val_accuracy did not improve from 0.90430
```

```
101/101 [==============================] - 37s 364ms/step - loss: 0.0011 -
accuracy: 0.9997 - val_loss: 0.5502 - val_accuracy: 0.8965
Epoch 77/80
100/101 [============================>.] - ETA: 0s - loss: 7.7876e-04 -
accuracy: 1.0000
Epoch 00077: val_accuracy did not improve from 0.90430
101/101 [==============================] - 37s 367ms/step - loss: 7.7230e-04
- accuracy: 1.0000 - val_loss: 0.5353 - val_accuracy: 0.9004
Epoch 78/80
100/101 [============================>.] - ETA: 0s - loss: 5.9856e-04 -
accuracy: 1.0000
Epoch 00078: val_accuracy did not improve from 0.90430
101/101 [==============================] - 36s 356ms/step - loss: 5.9405e-04
- accuracy: 1.0000 - val_loss: 0.5443 - val_accuracy: 0.9004
Epoch 79/80
100/101 [============================>.] - ETA: 0s - loss: 3.5312e-04 -
accuracy: 1.0000
Epoch 00079: val_accuracy did not improve from 0.90430
101/101 [==============================] - 35s 351ms/step - loss: 3.5570e-04
- accuracy: 1.0000 - val_loss: 0.5413 - val_accuracy: 0.9023
Epoch 80/80
100/101 [============================>.] - ETA: 0s - loss: 8.2249e-04 -
accuracy: 0.9997
Epoch 00080: val_accuracy did not improve from 0.90430
101/101 [==============================] - 37s 366ms/step - loss: 8.1458e-04
- accuracy: 0.9997 - val_loss: 0.5577 - val_accuracy: 0.8984
```

```
def plot_history_fined(history, history_fined, initial_epochs=epochs_0,
loss_max=1):
    """
    Check loss and accuracy evolution after fine tuning
    """
        acc = history.history['accuracy'][:epochs_0]
    acc += history_fined.history['accuracy']
    val_acc = history.history['val_accuracy'][:epochs_0]
    val_acc += history_fined.history['val_accuracy']
        loss = history.history['loss'][:epochs_0]
    loss += history_fined.history['loss']
    val_loss = history.history['val_loss'][:epochs_0]
    val_loss += history_fined.history['val_loss']
         fig, (ax1, ax2) = plt.subplots(1,2,figsize=(14, 4))
    ax1.plot(acc, label='Training')
    ax1.plot(val_acc, label='Validation')
    ax1.plot([initial_epochs-1,initial_epochs-1],
              plt.ylim(), label='fine-tuning', ls='--')
    ax1.legend(loc='lower right')
```

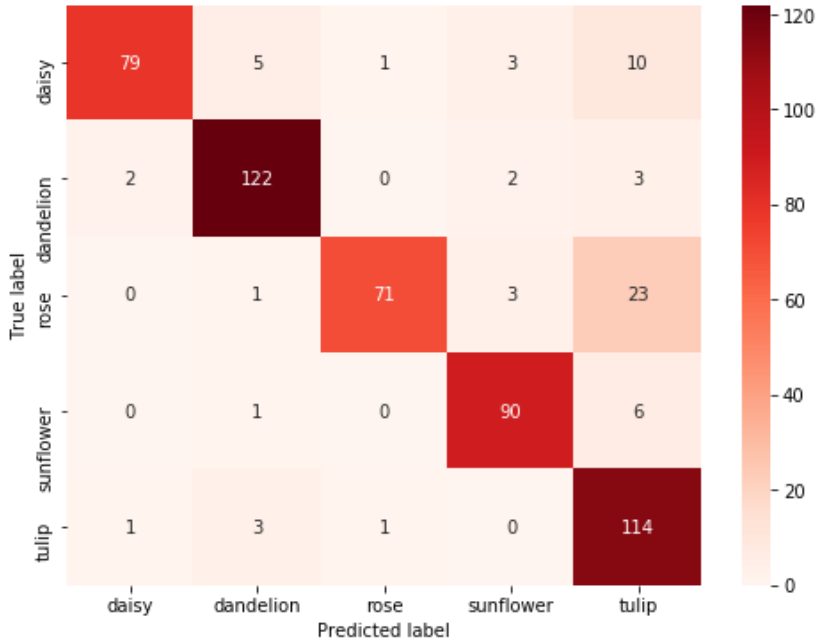
```
    ax1.set(ylabel='Accuracy', title='Training - Validation Accuracy',
            ylim=([0.4,1.005]))
    ax2.plot(loss, label='Training')
    ax2.plot(val_loss, label='Validation')
    ax2.plot([initial_epochs-1,initial_epochs-1],
             [0,1] , label='fine-tuning', ls='--')
    ax2.legend(loc='upper right')

ax2.set(ylabel='Loss (cross entropy)', xlabel='epochs',
           title='Training - Validation Loss', ylim=([0, loss_max]))
    plt.show()
plot_history_fined(history, history_fined)
```



Confusion matrix

```
import seaborn as sns
from sklearn import metrics
pred = model.predict(eval_datagen, verbose=1)
# get most likely class
y_pred = pred.argmax(axis=1)
y_true = ytest.argmax(axis=1)
print(metrics.classification_report(y_true, y_pred))
# confusion matrix
mat = metrics.confusion_matrix(y_true, y_pred)
df_mat = pd.DataFrame(mat, index=labels, columns=labels)
plt.figure(figsize=(8,6))
sns.heatmap(df_mat, annot=True, fmt='d', cmap=plt.cm.Reds)
#plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
17/17 [==============================] - 1s 76ms/step
              precision    recall  f1-score   support
```

```
              0       0.96      0.81      0.88        98
              1       0.92      0.95      0.93       129
              2       0.97      0.72      0.83        98
              3       0.92      0.93      0.92        97
              4       0.73      0.96      0.83       119

     accuracy                           0.88       541
    macro avg       0.90      0.87      0.88       541
 weighted avg       0.90      0.88      0.88       541
```

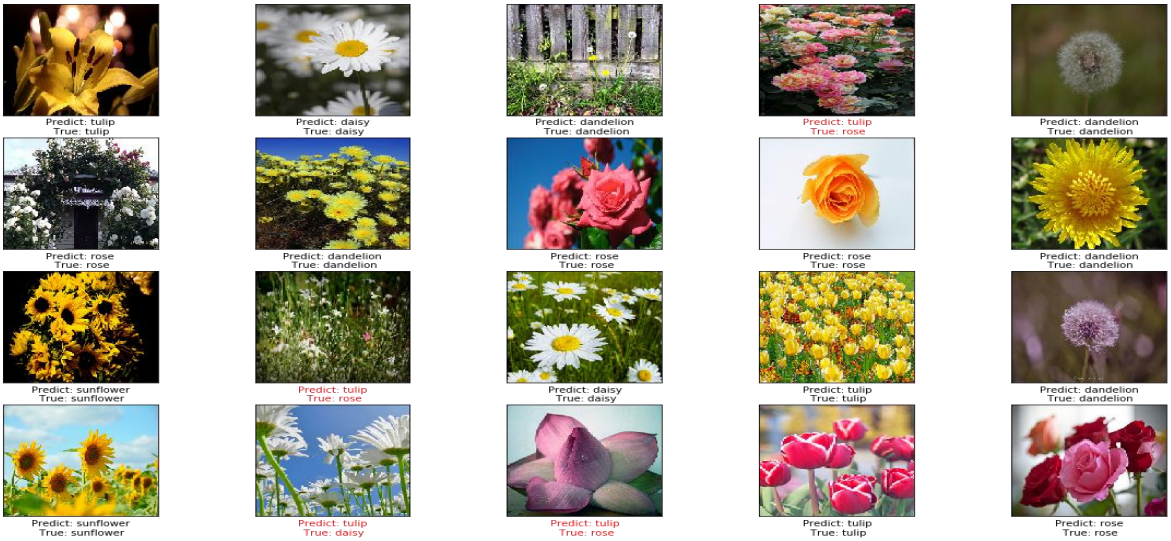


Prediction vizualisations

```
N = 20  # flowers to display
fig, axes = plt.subplots(4, 5, figsize=(20,12))
for i, ax in enumerate(axes.flat):
    ax.imshow(Xtest[i].astype(np.uint8))
    ax.set(xticks=[], yticks=[])
    true = y_true[i]
    prediction = y_pred[i]
    ax.set_xlabel(f'Predict: {labels[prediction]}\n True: {labels[true]}',
                  color='black' if true == prediction else 'red')

#fig.tight_layout()
fig.suptitle('Predicted flowers; Incorrect Labels in Red', size=14)
```

```
Text(0.5, 0.98, 'Predicted flowers; Incorrect Labels in Red')
```

Predicted flowers; Incorrect Labels in Red



7. Program: Building a simple convolutional neural network for transfer learning using finetuning.

1. Feature Extraction: Use the representations learned by a previous network to extract meaningful features from new samples. You simply add a new classifier, which will be trained from scratch, on top of the pretrained model so that you can repurpose the feature maps learned previously for the dataset.

   You do not need to (re)train the entire model. The base convolutional network already contains features that are generically useful for classifying pictures. However, the final, classification part of the pretrained model is specific to the original classification task, and subsequently specific to the set of classes on which the model was trained.

2. Fine-Tuning: Unfreeze a few of the top layers of a frozen model base and jointly train both the newly-added classifier layers and the last layers of the base model. This allows us to "fine-tune" the higher-order feature representations in the base model in order to make them more relevant for the specific task.

1. Examine and understand the data
2. Build an input pipeline, in this case using Keras ImageDataGenerator
3. Compose the model
   - Load in the pretrained base model (and pretrained weights)
   - Stack the classification layers on top
4. Train the model
5. Evaluate model

```
import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf
```

Data preprocessing

```
_URL = 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip'
path_to_zip = tf.keras.utils.get_file('cats_and_dogs.zip', origin=_URL, extract=True)
PATH = os.path.join(os.path.dirname(path_to_zip), 'cats_and_dogs_filtered')

train_dir = os.path.join(PATH, 'train')
validation_dir = os.path.join(PATH, 'validation')

BATCH_SIZE = 32
IMG_SIZE = (160, 160)

train_dataset = tf.keras.utils.image_dataset_from_directory(train_dir,
                                                            shuffle=True,
                                                            batch_size=BATCH_SIZE,
                                                            image_size=IMG_SIZE)
```

```
Downloading data from https://storage.googleapis.com/mledu-
datasets/cats_and_dogs_filtered.zip
68608000/68606236 [==============================] - 2s 0us/step
68616192/68606236 [==============================] - 2s 0us/step
Found 2000 files belonging to 2 classes.
```

```
validation_dataset = tf.keras.utils.image_dataset_from_directory(validation_dir,
                                                 shuffle=True,
                                                 batch_size=BATCH_SIZE,
                                                 image_size=IMG_SIZE)
```

```
Found 1000 files belonging to 2 classes.
```

```
class_names = train_dataset.class_names
plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):

ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```


```
val_batches = tf.data.experimental.cardinality(validation_dataset)
test_dataset = validation_dataset.take(val_batches // 5)
validation_dataset = validation_dataset.skip(val_batches // 5)
print('Number of validation batches: %d' % tf.data.experimental.cardinality(validation_dataset))
print('Number of test batches: %d' % tf.data.experimental.cardinality(test_dataset))
```

```
Number of validation batches: 26
Number of test batches: 6
```

```
AUTOTUNE = tf.data.AUTOTUNE
```

```
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

Use data augmentation

```
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal'),
  tf.keras.layers.RandomRotation(0.2),])
for image, _ in train_dataset.take(1):
  plt.figure(figsize=(10, 10))
  first_image = image[0]
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    augmented_image = data_augmentation(tf.expand_dims(first_image, 0))
    plt.imshow(augmented_image[0] / 255)
    plt.axis('off')
```

```
preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
rescale = tf.keras.layers.Rescaling(1./127.5, offset=-1)
```

Create the base model from the pre-trained convnets

```
# Create the base model from the pre-trained model MobileNet V2
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                               include_top=False, weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.
0_160_no_top.h5
9412608/9406464 [==============================] - 0s 0us/step
9420800/9406464 [==============================] - 0s 0us/step
```

```
image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(32, 5, 5, 1280)
```

Feature extraction

```
base_model.trainable = False
```

Let's take a look at the base model architecture

base_model.summary()

```
Model: "mobilenetv2_1.00_160"
_________________________________________________________________________________
_____________________
Layer (type)                 Output Shape              Param #     Connected
to
=================================================================================
=====================
input_1 (InputLayer)         [(None, 160, 160, 3) 0
_________________________________________________________________________________
_____________________
Conv1 (Conv2D)               (None, 80, 80, 32)   864
input_1[0][0]
_________________________________________________________________________________
_____________________
bn_Conv1 (BatchNormalization)   (None, 80, 80, 32)   128         Conv1[0][0]
_________________________________________________________________________________
_____________________
Conv1_relu (ReLU)            (None, 80, 80, 32)   0
bn_Conv1[0][0]
_________________________________________________________________________________
_____________________
expanded_conv_depthwise (Depthw (None, 80, 80, 32)   288
Conv1_relu[0][0]
_________________________________________________________________________________
_____________________
expanded_conv_depthwise_BN (Bat (None, 80, 80, 32)   128
expanded_conv_depthwise[0][0]
_________________________________________________________________________________
_____________________
expanded_conv_depthwise_relu (R (None, 80, 80, 32)   0
expanded_conv_depthwise_BN[0][0]
_________________________________________________________________________________
_____________________
expanded_conv_project (Conv2D)  (None, 80, 80, 16)   512
expanded_conv_depthwise_relu[0][0
```

```
__________________________________________________________________________________
______________________
expanded_conv_project_BN (Batch (None, 80, 80, 16)   64
expanded_conv_project[0][0]
__________________________________________________________________________________
______________________
block_1_expand (Conv2D)         (None, 80, 80, 96)   1536
expanded_conv_project_BN[0][0]
__________________________________________________________________________________
______________________
block_1_expand_BN (BatchNormali (None, 80, 80, 96)   384
block_1_expand[0][0]
__________________________________________________________________________________
______________________
block_1_expand_relu (ReLU)      (None, 80, 80, 96)   0
block_1_expand_BN[0][0]
__________________________________________________________________________________
______________________
block_1_pad (ZeroPadding2D)     (None, 81, 81, 96)   0
block_1_expand_relu[0][0]
__________________________________________________________________________________
______________________
block_1_depthwise (DepthwiseCon (None, 40, 40, 96)   864
block_1_pad[0][0]
__________________________________________________________________________________
______________________
block_1_depthwise_BN (BatchNorm (None, 40, 40, 96)   384
block_1_depthwise[0][0]
__________________________________________________________________________________
______________________
block_1_depthwise_relu (ReLU)   (None, 40, 40, 96)   0
block_1_depthwise_BN[0][0]
__________________________________________________________________________________
______________________
block_1_project (Conv2D)        (None, 40, 40, 24)   2304
block_1_depthwise_relu[0][0]
__________________________________________________________________________________
______________________
block_1_project_BN (BatchNormal (None, 40, 40, 24)   96
block_1_project[0][0]
__________________________________________________________________________________
______________________
block_2_expand (Conv2D)         (None, 40, 40, 144)  3456
block_1_project_BN[0][0]
__________________________________________________________________________________
______________________
block_2_expand_BN (BatchNormali (None, 40, 40, 144)  576
block_2_expand[0][0]
__________________________________________________________________________________
______________________
block_2_expand_relu (ReLU)      (None, 40, 40, 144)  0
block_2_expand_BN[0][0]
```

```
----------------------------------------------------------------------------------------------------
block_2_depthwise (DepthwiseCon (None, 40, 40, 144)  1296
block_2_expand_relu[0][0]
----------------------------------------------------------------------------------------------------
block_2_depthwise_BN (BatchNorm (None, 40, 40, 144)  576
block_2_depthwise[0][0]
----------------------------------------------------------------------------------------------------
block_2_depthwise_relu (ReLU)   (None, 40, 40, 144)  0
block_2_depthwise_BN[0][0]
----------------------------------------------------------------------------------------------------
block_2_project (Conv2D)        (None, 40, 40, 24)   3456
block_2_depthwise_relu[0][0]
----------------------------------------------------------------------------------------------------
block_2_project_BN (BatchNormal (None, 40, 40, 24)   96
block_2_project[0][0]
----------------------------------------------------------------------------------------------------
block_2_add (Add)               (None, 40, 40, 24)   0
block_1_project_BN[0][0]

block_2_project_BN[0][0]
----------------------------------------------------------------------------------------------------
block_3_expand (Conv2D)         (None, 40, 40, 144)  3456
block_2_add[0][0]
----------------------------------------------------------------------------------------------------
block_3_expand_BN (BatchNormali (None, 40, 40, 144)  576
block_3_expand[0][0]
----------------------------------------------------------------------------------------------------
block_3_expand_relu (ReLU)      (None, 40, 40, 144)  0
block_3_expand_BN[0][0]
----------------------------------------------------------------------------------------------------
block_3_pad (ZeroPadding2D)     (None, 41, 41, 144)  0
block_3_expand_relu[0][0]
----------------------------------------------------------------------------------------------------
block_3_depthwise (DepthwiseCon (None, 20, 20, 144)  1296
block_3_pad[0][0]
----------------------------------------------------------------------------------------------------
block_3_depthwise_BN (BatchNorm (None, 20, 20, 144)  576
block_3_depthwise[0][0]
----------------------------------------------------------------------------------------------------
----------
```

```
block_3_depthwise_relu (ReLU)   (None, 20, 20, 144)  0
block_3_depthwise_BN[0][0]
__________________________________________________________________________________
_____________________
block_3_project (Conv2D)        (None, 20, 20, 32)   4608
block_3_depthwise_relu[0][0]
__________________________________________________________________________________
_____________________
block_3_project_BN (BatchNormal (None, 20, 20, 32)   128
block_3_project[0][0]
__________________________________________________________________________________
_____________________
block_4_expand (Conv2D)         (None, 20, 20, 192)  6144
block_3_project_BN[0][0]
__________________________________________________________________________________
_____________________
block_4_expand_BN (BatchNormali (None, 20, 20, 192)  768
block_4_expand[0][0]
__________________________________________________________________________________
_____________________
block_4_expand_relu (ReLU)      (None, 20, 20, 192)  0
block_4_expand_BN[0][0]
__________________________________________________________________________________
_____________________
block_4_depthwise (DepthwiseCon (None, 20, 20, 192)  1728
block_4_expand_relu[0][0]
__________________________________________________________________________________
_____________________
block_4_depthwise_BN (BatchNorm (None, 20, 20, 192)  768
block_4_depthwise[0][0]
__________________________________________________________________________________
_____________________
block_4_depthwise_relu (ReLU)   (None, 20, 20, 192)  0
block_4_depthwise_BN[0][0]
__________________________________________________________________________________
_____________________
block_4_project (Conv2D)        (None, 20, 20, 32)   6144
block_4_depthwise_relu[0][0]
__________________________________________________________________________________
_____________________
block_4_project_BN (BatchNormal (None, 20, 20, 32)   128
block_4_project[0][0]
__________________________________________________________________________________
_____________________
block_4_add (Add)               (None, 20, 20, 32)   0
block_3_project_BN[0][0]

block_4_project_BN[0][0]
__________________________________________________________________________________
_____________________
```

```
block_5_expand (Conv2D)         (None, 20, 20, 192)  6144
block_4_add[0][0]
__________________________________________________________________________________
______________________
block_5_expand_BN (BatchNormali (None, 20, 20, 192)  768
block_5_expand[0][0]
__________________________________________________________________________________
______________________
block_5_expand_relu (ReLU)      (None, 20, 20, 192)  0
block_5_expand_BN[0][0]
__________________________________________________________________________________
______________________
block_5_depthwise (DepthwiseCon (None, 20, 20, 192)  1728
block_5_expand_relu[0][0]
__________________________________________________________________________________
______________________
block_5_depthwise_BN (BatchNorm (None, 20, 20, 192)  768
block_5_depthwise[0][0]
__________________________________________________________________________________
______________________
block_5_depthwise_relu (ReLU)   (None, 20, 20, 192)  0
block_5_depthwise_BN[0][0]
__________________________________________________________________________________
______________________
block_5_project (Conv2D)        (None, 20, 20, 32)   6144
block_5_depthwise_relu[0][0]
__________________________________________________________________________________
______________________
block_5_project_BN (BatchNormal (None, 20, 20, 32)   128
block_5_project[0][0]
__________________________________________________________________________________
______________________
block_5_add (Add)               (None, 20, 20, 32)   0
block_4_add[0][0]
block_5_project_BN[0][0]
__________________________________________________________________________________
______________________
block_6_expand (Conv2D)         (None, 20, 20, 192)  6144
block_5_add[0][0]
__________________________________________________________________________________
______________________
block_6_expand_BN (BatchNormali (None, 20, 20, 192)  768
block_6_expand[0][0]
__________________________________________________________________________________
______________________
block_6_expand_relu (ReLU)      (None, 20, 20, 192)  0
block_6_expand_BN[0][0]
__________________________________________________________________________________
______________________
block_6_pad (ZeroPadding2D)     (None, 21, 21, 192)  0
block_6_expand_relu[0][0]
```

```
----------------------------------------------------------------------------------------------------
block_6_depthwise (DepthwiseCon (None, 10, 10, 192)  1728
block_6_pad[0][0]
----------------------------------------------------------------------------------------------------
block_6_depthwise_BN (BatchNorm (None, 10, 10, 192)  768
block_6_depthwise[0][0]
----------------------------------------------------------------------------------------------------
block_6_depthwise_relu (ReLU)   (None, 10, 10, 192)  0
block_6_depthwise_BN[0][0]
----------------------------------------------------------------------------------------------------
block_6_project (Conv2D)        (None, 10, 10, 64)   12288
block_6_depthwise_relu[0][0]
----------------------------------------------------------------------------------------------------
block_6_project_BN (BatchNormal (None, 10, 10, 64)   256
block_6_project[0][0]
----------------------------------------------------------------------------------------------------
block_7_expand (Conv2D)         (None, 10, 10, 384)  24576
block_6_project_BN[0][0]
----------------------------------------------------------------------------------------------------
block_7_expand_BN (BatchNormali (None, 10, 10, 384)  1536
block_7_expand[0][0]
----------------------------------------------------------------------------------------------------
block_7_expand_relu (ReLU)      (None, 10, 10, 384)  0
block_7_expand_BN[0][0]
----------------------------------------------------------------------------------------------------
block_7_depthwise (DepthwiseCon (None, 10, 10, 384)  3456
block_7_expand_relu[0][0]
----------------------------------------------------------------------------------------------------
block_7_depthwise_BN (BatchNorm (None, 10, 10, 384)  1536
block_7_depthwise[0][0]
----------------------------------------------------------------------------------------------------
block_7_depthwise_relu (ReLU)   (None, 10, 10, 384)  0
block_7_depthwise_BN[0][0]
----------------------------------------------------------------------------------------------------
block_7_project (Conv2D)        (None, 10, 10, 64)   24576
block_7_depthwise_relu[0][0]
----------------------------------------------------------------------------------------------------
block_7_project_BN (BatchNormal (None, 10, 10, 64)   256
block_7_project[0][0]
```

```
__________________________________________________________________________________
______________________

block_7_add (Add)               (None, 10, 10, 64)   0
block_6_project_BN[0][0]
block_7_project_BN[0][0]
__________________________________________________________________________________
______________________
block_8_expand (Conv2D)         (None, 10, 10, 384)  24576
block_7_add[0][0]
__________________________________________________________________________________
______________________
block_8_expand_BN (BatchNormali (None, 10, 10, 384)  1536
block_8_expand[0][0]
__________________________________________________________________________________
______________________
block_8_expand_relu (ReLU)      (None, 10, 10, 384)  0
block_8_expand_BN[0][0]
__________________________________________________________________________________
______________________
block_8_depthwise (DepthwiseCon (None, 10, 10, 384)  3456
block_8_expand_relu[0][0]
__________________________________________________________________________________
______________________
block_8_depthwise_BN (BatchNorm (None, 10, 10, 384)  1536
block_8_depthwise[0][0]
__________________________________________________________________________________
______________________
block_8_depthwise_relu (ReLU)   (None, 10, 10, 384)  0
block_8_depthwise_BN[0][0]
__________________________________________________________________________________
______________________
block_8_project (Conv2D)        (None, 10, 10, 64)   24576
block_8_depthwise_relu[0][0]
__________________________________________________________________________________
______________________
block_8_project_BN (BatchNormal (None, 10, 10, 64)   256
block_8_project[0][0]
__________________________________________________________________________________
______________________
block_8_add (Add)               (None, 10, 10, 64)   0
block_7_add[0][0]

block_8_project_BN[0][0]
__________________________________________________________________________________
______________________
block_9_expand (Conv2D)         (None, 10, 10, 384)  24576
block_8_add[0][0]
__________________________________________________________________________________
______________________
block_9_expand_BN (BatchNormali (None, 10, 10, 384)  1536
block_9_expand[0][0]
```

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

block_9_expand_relu (ReLU)      (None, 10, 10, 384)  0
block_9_expand_BN[0][0]
----------------------------------------------------------------------------------------------------
block_9_depthwise (DepthwiseCon (None, 10, 10, 384)  3456
block_9_expand_relu[0][0]
----------------------------------------------------------------------------------------------------
block_9_depthwise_BN (BatchNorm (None, 10, 10, 384)  1536
block_9_depthwise[0][0]
----------------------------------------------------------------------------------------------------
block_9_depthwise_relu (ReLU)   (None, 10, 10, 384)  0
block_9_depthwise_BN[0][0]
----------------------------------------------------------------------------------------------------
block_9_project (Conv2D)        (None, 10, 10, 64)   24576
block_9_depthwise_relu[0][0]
----------------------------------------------------------------------------------------------------
block_9_project_BN (BatchNormal (None, 10, 10, 64)   256
block_9_project[0][0]
----------------------------------------------------------------------------------------------------
block_9_add (Add)               (None, 10, 10, 64)   0
block_8_add[0][0]

block_9_project_BN[0][0]
----------------------------------------------------------------------------------------------------
block_10_expand (Conv2D)        (None, 10, 10, 384)  24576
block_9_add[0][0]
----------------------------------------------------------------------------------------------------
block_10_expand_BN (BatchNormal (None, 10, 10, 384)  1536
block_10_expand[0][0]
----------------------------------------------------------------------------------------------------
block_10_expand_relu (ReLU)     (None, 10, 10, 384)  0
block_10_expand_BN[0][0]
----------------------------------------------------------------------------------------------------
block_10_depthwise (DepthwiseCo (None, 10, 10, 384)  3456
block_10_expand_relu[0][0]
----------------------------------------------------------------------------------------------------
block_10_depthwise_BN (BatchNor (None, 10, 10, 384)  1536
block_10_depthwise[0][0]
```

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

block_10_depthwise_relu (ReLU)  (None, 10, 10, 384)  0
block_10_depthwise_BN[0][0]
----------------------------------------------------------------------------------------------------
block_10_project (Conv2D)       (None, 10, 10, 96)   36864
block_10_depthwise_relu[0][0]
----------------------------------------------------------------------------------------------------
block_10_project_BN (BatchNorma (None, 10, 10, 96)   384
block_10_project[0][0]
----------------------------------------------------------------------------------------------------
block_11_expand (Conv2D)        (None, 10, 10, 576)  55296
block_10_project_BN[0][0]
----------------------------------------------------------------------------------------------------
block_11_expand_BN (BatchNormal (None, 10, 10, 576)  2304
block_11_expand[0][0]
----------------------------------------------------------------------------------------------------
block_11_expand_relu (ReLU)     (None, 10, 10, 576)  0
block_11_expand_BN[0][0]
----------------------------------------------------------------------------------------------------
block_11_depthwise (DepthwiseCo (None, 10, 10, 576)  5184
block_11_expand_relu[0][0]
----------------------------------------------------------------------------------------------------
block_11_depthwise_BN (BatchNor (None, 10, 10, 576)  2304
block_11_depthwise[0][0]
----------------------------------------------------------------------------------------------------
block_11_depthwise_relu (ReLU)  (None, 10, 10, 576)  0
block_11_depthwise_BN[0][0]
----------------------------------------------------------------------------------------------------
block_11_project (Conv2D)       (None, 10, 10, 96)   55296
block_11_depthwise_relu[0][0]
----------------------------------------------------------------------------------------------------
block_11_project_BN (BatchNorma (None, 10, 10, 96)   384
block_11_project[0][0]
----------------------------------------------------------------------------------------------------
block_11_add (Add)              (None, 10, 10, 96)   0
block_10_project_BN[0][0]

block_11_project_BN[0][0]
```

```
__________________________________________________________________________________
______________________
block_12_expand (Conv2D)        (None, 10, 10, 576)  55296
block_11_add[0][0]
__________________________________________________________________________________
______________________
block_12_expand_BN (BatchNormal (None, 10, 10, 576)  2304
block_12_expand[0][0]
__________________________________________________________________________________
______________________
block_12_expand_relu (ReLU)     (None, 10, 10, 576)  0
block_12_expand_BN[0][0]
__________________________________________________________________________________
______________________
block_12_depthwise (DepthwiseCo (None, 10, 10, 576)  5184
block_12_expand_relu[0][0]
__________________________________________________________________________________
______________________
block_12_depthwise_BN (BatchNor (None, 10, 10, 576)  2304
block_12_depthwise[0][0]
__________________________________________________________________________________
______________________
block_12_depthwise_relu (ReLU)  (None, 10, 10, 576)  0
block_12_depthwise_BN[0][0]
__________________________________________________________________________________
______________________
block_12_project (Conv2D)       (None, 10, 10, 96)   55296
block_12_depthwise_relu[0][0]
__________________________________________________________________________________
______________________
block_12_project_BN (BatchNorma (None, 10, 10, 96)   384
block_12_project[0][0]
__________________________________________________________________________________
______________________
block_12_add (Add)              (None, 10, 10, 96)   0
block_11_add[0][0]
block_12_project_BN[0][0]
__________________________________________________________________________________
______________________
block_13_expand (Conv2D)        (None, 10, 10, 576)  55296
block_12_add[0][0]
__________________________________________________________________________________
______________________
block_13_expand_BN (BatchNormal (None, 10, 10, 576)  2304
block_13_expand[0][0]
__________________________________________________________________________________
______________________
block_13_expand_relu (ReLU)     (None, 10, 10, 576)  0
block_13_expand_BN[0][0]
__________________________________________________________________________________
______________________
```

```
block_13_pad (ZeroPadding2D)    (None, 11, 11, 576)  0
block_13_expand_relu[0][0]
__________________________________________________________________________
_____________________
block_13_depthwise (DepthwiseCo (None, 5, 5, 576)    5184
block_13_pad[0][0]
__________________________________________________________________________
_____________________
block_13_depthwise_BN (BatchNor (None, 5, 5, 576)    2304
block_13_depthwise[0][0]
__________________________________________________________________________
_____________________
block_13_depthwise_relu (ReLU)  (None, 5, 5, 576)    0
block_13_depthwise_BN[0][0]
__________________________________________________________________________
_____________________
block_13_project (Conv2D)       (None, 5, 5, 160)    92160
block_13_depthwise_relu[0][0]
__________________________________________________________________________
_____________________
block_13_project_BN (BatchNorma (None, 5, 5, 160)    640
block_13_project[0][0]
__________________________________________________________________________
_____________________
block_14_expand (Conv2D)        (None, 5, 5, 960)    153600
block_13_project_BN[0][0]
__________________________________________________________________________
_____________________
block_14_expand_BN (BatchNormal (None, 5, 5, 960)    3840
block_14_expand[0][0]
__________________________________________________________________________
_____________________
block_14_expand_relu (ReLU)     (None, 5, 5, 960)    0
block_14_expand_BN[0][0]
__________________________________________________________________________
_____________________
block_14_depthwise (DepthwiseCo (None, 5, 5, 960)    8640
block_14_expand_relu[0][0]
__________________________________________________________________________
_____________________
block_14_depthwise_BN (BatchNor (None, 5, 5, 960)    3840
block_14_depthwise[0][0]
__________________________________________________________________________
_____________________
block_14_depthwise_relu (ReLU)  (None, 5, 5, 960)    0
block_14_depthwise_BN[0][0]
__________________________________________________________________________
_____________________
block_14_project (Conv2D)       (None, 5, 5, 160)    153600
block_14_depthwise_relu[0][0]
__________________________________________________________________________
_____________________
```

```
block_14_project_BN (BatchNorma (None, 5, 5, 160)    640
block_14_project[0][0]
__________________________________________________________________________________
______________________
block_14_add (Add)              (None, 5, 5, 160)    0
block_13_project_BN[0][0]

block_14_project_BN[0][0]
__________________________________________________________________________________
______________________
block_15_expand (Conv2D)        (None, 5, 5, 960)    153600
block_14_add[0][0]
__________________________________________________________________________________
______________________
block_15_expand_BN (BatchNormal (None, 5, 5, 960)    3840
block_15_expand[0][0]
__________________________________________________________________________________
______________________
block_15_expand_relu (ReLU)     (None, 5, 5, 960)    0
block_15_expand_BN[0][0]
__________________________________________________________________________________
______________________
block_15_depthwise (DepthwiseCo (None, 5, 5, 960)    8640
block_15_expand_relu[0][0]
__________________________________________________________________________________
______________________
block_15_depthwise_BN (BatchNor (None, 5, 5, 960)    3840
block_15_depthwise[0][0]
__________________________________________________________________________________
______________________
block_15_depthwise_relu (ReLU)  (None, 5, 5, 960)    0
block_15_depthwise_BN[0][0]
__________________________________________________________________________________
______________________
block_15_project (Conv2D)       (None, 5, 5, 160)    153600
block_15_depthwise_relu[0][0]
__________________________________________________________________________________
______________________
block_15_project_BN (BatchNorma (None, 5, 5, 160)    640
block_15_project[0][0]
__________________________________________________________________________________
______________________
block_15_add (Add)              (None, 5, 5, 160)    0
block_14_add[0][0]
block_15_project_BN[0][0]
__________________________________________________________________________________
______________________
block_16_expand (Conv2D)        (None, 5, 5, 960)    153600
block_15_add[0][0]
__________________________________________________________________________________
______________________
```

```
block_16_expand_BN (BatchNormal (None, 5, 5, 960)    3840
block_16_expand[0][0]
__________________________________________________________________________________
__________________
block_16_expand_relu (ReLU)     (None, 5, 5, 960)    0
block_16_expand_BN[0][0]
__________________________________________________________________________________
__________________
block_16_depthwise (DepthwiseCo (None, 5, 5, 960)    8640
block_16_expand_relu[0][0]
__________________________________________________________________________________
__________________
block_16_depthwise_BN (BatchNor (None, 5, 5, 960)    3840
block_16_depthwise[0][0]
__________________________________________________________________________________
__________________
block_16_depthwise_relu (ReLU)  (None, 5, 5, 960)    0
block_16_depthwise_BN[0][0]
__________________________________________________________________________________
__________________
block_16_project (Conv2D)       (None, 5, 5, 320)    307200
block_16_depthwise_relu[0][0]
__________________________________________________________________________________
__________________
block_16_project_BN (BatchNorma (None, 5, 5, 320)    1280
block_16_project[0][0]
__________________________________________________________________________________
__________________
Conv_1 (Conv2D)                 (None, 5, 5, 1280)   409600
block_16_project_BN[0][0]
__________________________________________________________________________________
__________________
Conv_1_bn (BatchNormalization)  (None, 5, 5, 1280)   5120
Conv_1[0][0]
__________________________________________________________________________________
__________________
out_relu (ReLU)                 (None, 5, 5, 1280)   0
Conv_1_bn[0][0]
==================================================================================
==================
Total params: 2,257,984
Trainable params: 0
Non-trainable params: 2,257,984
__________________________________________________________________________________
__________________
```

```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(32, 1280)
```

```
prediction_layer = tf.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(32, 1)
```

```
inputs = tf.keras.Input(shape=(160, 160, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)

outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

Compile the model

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 160, 160, 3)]     0
_________________________________________________________________
sequential (Sequential)      (None, 160, 160, 3)       0
_________________________________________________________________
tf.math.truediv (TFOpLambda) (None, 160, 160, 3)       0
_________________________________________________________________
tf.math.subtract (TFOpLambda (None, 160, 160, 3)       0
_________________________________________________________________
mobilenetv2_1.00_160 (Functi (None, 5, 5, 1280)        2257984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dropout (Dropout)            (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 1281
=================================================================
Total params: 2,259,265
Trainable params: 1,281
Non-trainable params: 2,257,984
_________________________________________________________________
```

```
len(model.trainable_variables)
```

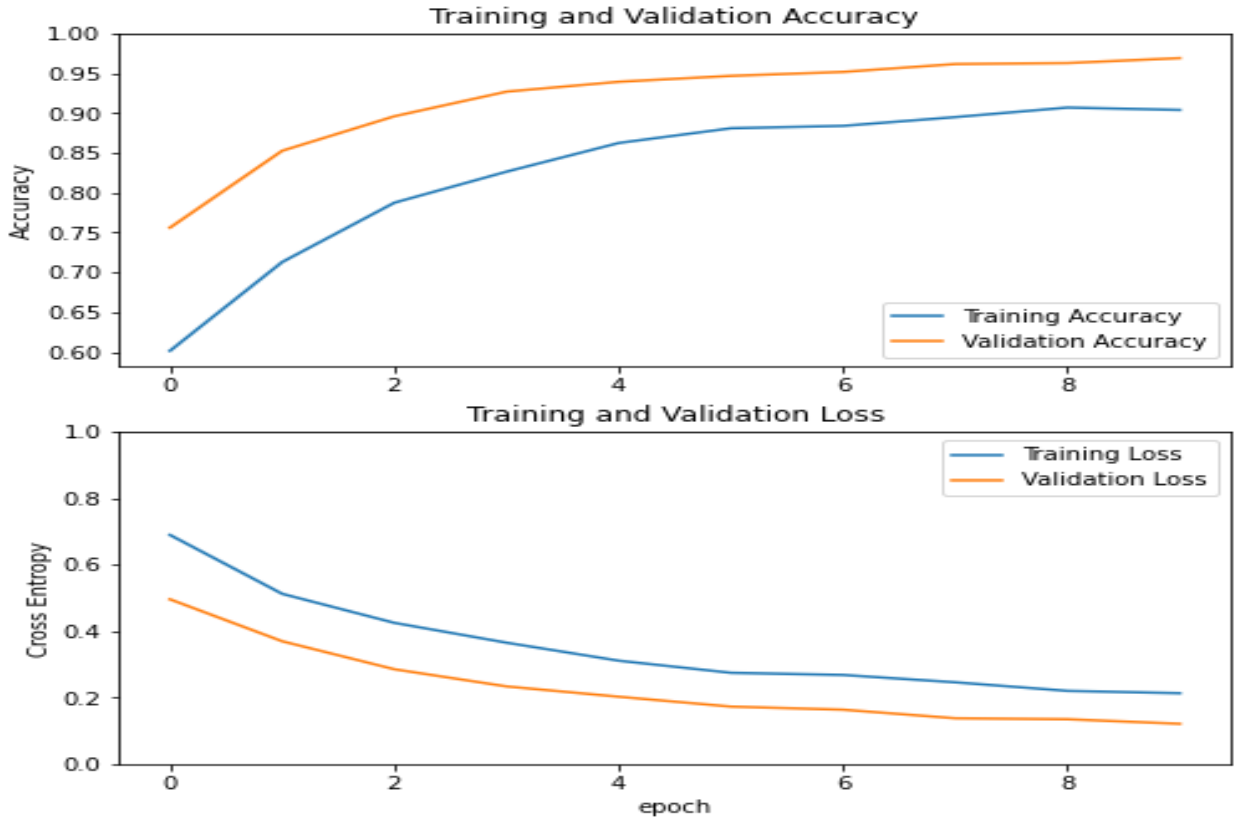
```
2
```

```
initial_epochs = 10

loss0, accuracy0 = model.evaluate(validation_dataset)
```

```
26/26 [==============================] - 4s 37ms/step - loss: 0.7442 -
accuracy: 0.5631
```

```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
initial loss: 0.74
initial accuracy: 0.56
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset)
Epoch 1/10
63/63 [==============================] - 8s 65ms/step - loss: 0.6895 -
accuracy: 0.6015 - val_loss: 0.4956 - val_accuracy: 0.7562
Epoch 2/10
63/63 [==============================] - 5s 69ms/step - loss: 0.5121 -
accuracy: 0.7130 - val_loss: 0.3692 - val_accuracy: 0.8527
Epoch 3/10
63/63 [==============================] - 5s 75ms/step - loss: 0.4247 -
accuracy: 0.7875 - val_loss: 0.2849 - val_accuracy: 0.8960
Epoch 4/10
63/63 [==============================] - 5s 76ms/step - loss: 0.3649 -
accuracy: 0.8265 - val_loss: 0.2332 - val_accuracy: 0.9270
Epoch 5/10
63/63 [==============================] - 4s 64ms/step - loss: 0.3107 -
accuracy: 0.8625 - val_loss: 0.2021 - val_accuracy: 0.9394
Epoch 6/10
63/63 [==============================] - 5s 68ms/step - loss: 0.2742 -
accuracy: 0.8810 - val_loss: 0.1724 - val_accuracy: 0.9468
Epoch 7/10
63/63 [==============================] - 4s 63ms/step - loss: 0.2677 -
accuracy: 0.8840 - val_loss: 0.1631 - val_accuracy: 0.9517
Epoch 8/10
63/63 [==============================] - 4s 64ms/step - loss: 0.2456 -
accuracy: 0.8950 - val_loss: 0.1372 - val_accuracy: 0.9616
Epoch 9/10
63/63 [==============================] - 4s 62ms/step - loss: 0.2198 -
accuracy: 0.9070 - val_loss: 0.1344 - val_accuracy: 0.9629
Epoch 10/10
63/63 [==============================] - 5s 73ms/step - loss: 0.2127 -
accuracy: 0.9040 - val_loss: 0.1207 - val_accuracy: 0.9691

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
```

```
plt.title('Training and Validation Accuracy')
plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Fine tuning

```
base_model.trainable = True
# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable = False
```

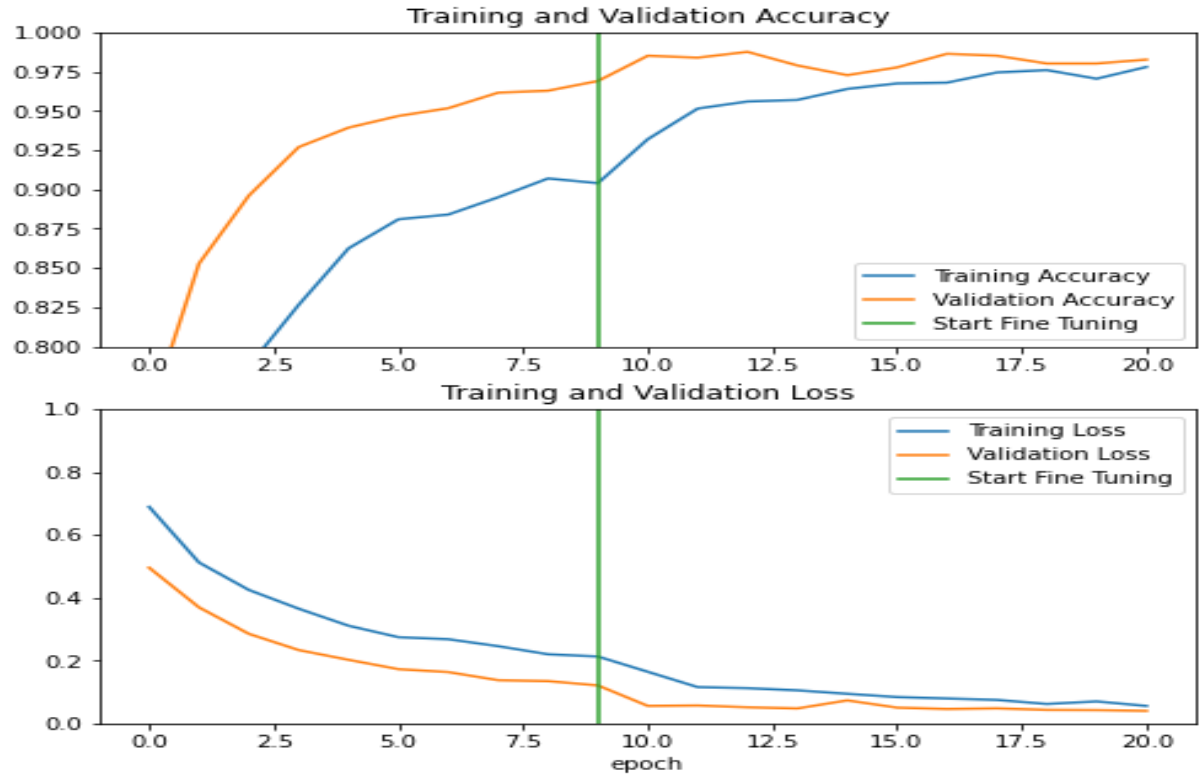
```
Number of layers in the base model:  154
```

Compile the model

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer =
tf.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
```

```
              metrics=['accuracy'])
model.summary()
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 160, 160, 3)]     0
_________________________________________________________________
sequential (Sequential)      (None, 160, 160, 3)       0
_________________________________________________________________
tf.math.truediv (TFOpLambda) (None, 160, 160, 3)       0
_________________________________________________________________
tf.math.subtract (TFOpLambda (None, 160, 160, 3)       0
_________________________________________________________________
mobilenetv2_1.00_160 (Functi (None, 5, 5, 1280)        2257984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dropout (Dropout)            (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 1281
=================================================================
Total params: 2,259,265
Trainable params: 1,862,721
Non-trainable params: 396,544
_________________________________________________________________
len(model.trainable_variables)
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset)
Epoch 10/20
63/63 [==============================] - 12s 97ms/step - loss: 0.1640 -
accuracy: 0.9320 - val_loss: 0.0554 - val_accuracy: 0.9851
Epoch 11/20
63/63 [==============================] - 5s 77ms/step - loss: 0.1153 -
accuracy: 0.9515 - val_loss: 0.0567 - val_accuracy: 0.9839
Epoch 12/20
63/63 [==============================] - 6s 82ms/step - loss: 0.1119 -
accuracy: 0.9560 - val_loss: 0.0510 - val_accuracy: 0.9876
Epoch 13/20
63/63 [==============================] - 5s 70ms/step - loss: 0.1050 -
accuracy: 0.9570 - val_loss: 0.0473 - val_accuracy: 0.9790
Epoch 14/20
63/63 [==============================] - 5s 73ms/step - loss: 0.0940 -
accuracy: 0.9640 - val_loss: 0.0730 - val_accuracy: 0.9728
Epoch 15/20
63/63 [==============================] - 5s 71ms/step - loss: 0.0835 -
accuracy: 0.9675 - val_loss: 0.0499 - val_accuracy: 0.9777
```

```
Epoch 16/20
63/63 [==============================] - 5s 71ms/step - loss: 0.0790 -
accuracy: 0.9680 - val_loss: 0.0455 - val_accuracy: 0.9864
Epoch 17/20
63/63 [==============================] - 5s 78ms/step - loss: 0.0745 -
accuracy: 0.9745 - val_loss: 0.0476 - val_accuracy: 0.9851
Epoch 18/20
63/63 [==============================] - 5s 77ms/step - loss: 0.0616 -
accuracy: 0.9760 - val_loss: 0.0427 - val_accuracy: 0.9802
Epoch 19/20
63/63 [==============================] - 5s 76ms/step - loss: 0.0697 -
accuracy: 0.9705 - val_loss: 0.0420 - val_accuracy: 0.9802
Epoch 20/20
63/63 [==============================] - 5s 72ms/step - loss: 0.0553 -
accuracy: 0.9780 - val_loss: 0.0396 - val_accuracy: 0.9827

acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']
loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0.8, 1])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



Evaluation and prediction

```
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

```
6/6 [==============================] - 1s 46ms/step - loss: 0.0307 -
accuracy: 0.9792
Test accuracy : 0.9791666865348816
```

```
# Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()
# Apply a sigmoid since our model returns logits
predictions = tf.nn.sigmoid(predictions)
predictions = tf.where(predictions < 0.5, 0, 1)
print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)

plt.figure(figsize=(10, 10))
for i in range(9):
  ax = plt.subplot(3, 3, i + 1)
  plt.imshow(image_batch[i].astype("uint8"))
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

```
Predictions:


[0 0 1 0 0 0 1 1 1 0 0 0 1 0 1 0 1 0 1 1 1 1 0 1 1 1 1 1 1 1 0 1]
Labels:
 [0 0 1 0 0 0 1 1 1 0 0 0 1 0 1 0 1 0 1 1 1 1 0 1 1 1 1 1 1 1 0 1]
```

cats

cats

dogs

cats

cats

cats

dogs

dogs

dogs

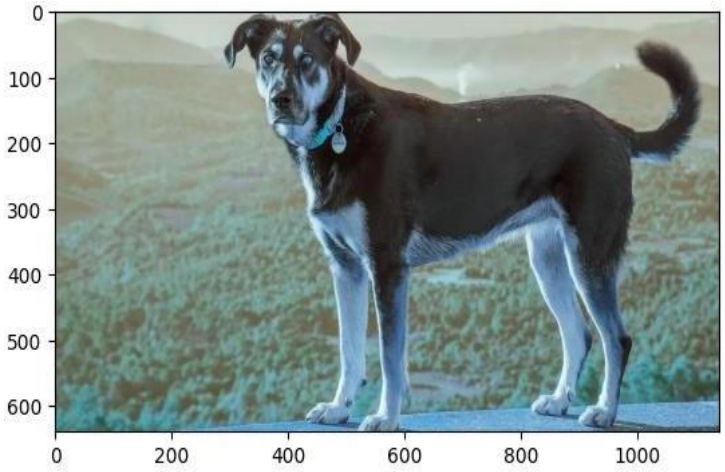
8. Program: Building a simple convolutional neural network for transfer learning using feature extraction.

#Extracting the ZIP File:

```
import zipfile
zip_ref = zipfile.ZipFile('/content/dogs-vs-cats.zip', 'r')
zip_ref.extractall('/content')
zip_ref.close()

#Importing TensorFlow and Keras:
import tensorflow
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense, Flatten
from keras.applications.vgg16 import VGG16

#Building the VGG16-Based Model:
conv_base = VGG16(weights='imagenet', include_top=False, input_shape=(150,
150, 3))
conv_base.summary()
model = Sequential()
model.add(conv_base)
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()

#Freezing the Base Model:
conv_base.trainable = False
model.summary()

#Loading and Processing the Datasets:
train_ds = keras.utils.image_dataset_from_directory(
    directory='/content/train',
    labels='inferred',
    label_mode='int',
    batch_size=32,
    image_size=(150, 150)
)

validation_ds = keras.utils.image_dataset_from_directory(
    directory='/content/test',
    labels='inferred',
    label_mode='int',
    batch_size=32,
```

```
    image_size=(150, 150)
)

import tensorflow as tf
def process(image, label):
    image = tf.cast(image / 255.0, tf.float32)
    return image, label

train_ds = train_ds.map(process)
validation_ds = validation_ds.map(process)
#Compiling and Training the Model:

model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
history = model.fit(train_ds, epochs=10, validation_data=validation_ds)
#Testing with Images:

import cv2
import cv2 as cv
from matplotlib import pyplot as plt

test_img = cv2.imread('/content/dog-
4615198_1280_1701927561029_1701927572424.jpg')
test_img = cv2.resize(test_img, (150, 150))
test_input = test_img.reshape((1, 150, 150, 3))
plt.imshow(test_img)
plt.show()
model.predict(test_input)

test_img = cv2.imread('/content/07CAT-STRIPES-superJumbo.jpg')
test_img = cv2.resize(test_img, (150, 150))
test_input = test_img.reshape((1, 150, 150, 3))
plt.imshow(test_img)
plt.show()
model.predict(test_input)

OUTPUT:
Epoch 1/10 625/625 [==============================] - 62s 87ms/step - loss:
0.2696 - accuracy: 0.8852 - val_loss: 0.2074 - val_accuracy: 0.9136 Epoch
2/10 625/625 [==============================] - 54s 86ms/step - loss: 0.1898
- accuracy: 0.9233 - val_loss: 0.2029 - val_accuracy: 0.9174 Epoch 3/10
625/625 [==============================] - 54s 85ms/step - loss: 0.1638 -
accuracy: 0.9329 - val_loss: 0.2047 - val_accuracy: 0.9176 Epoch 4/10
625/625 [==============================] - 54s 86ms/step - loss: 0.1349 -
accuracy: 0.9448 - val_loss: 0.2179 - val_accuracy: 0.9146 Epoch 5/10
```

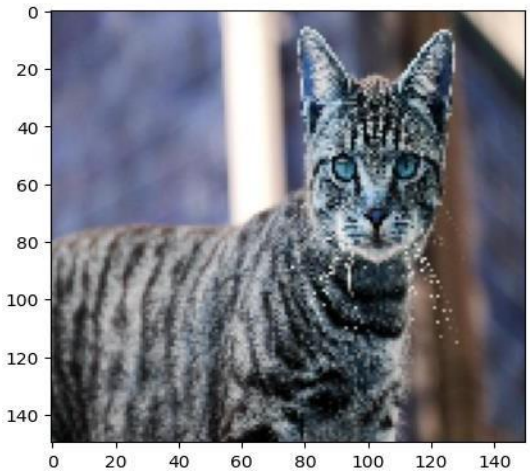
```
625/625 [==============================] - 64s 102ms/step - loss: 0.1148 -
accuracy: 0.9535 - val_loss: 0.2913 - val_accuracy: 0.8980 Epoch 6/10
625/625 [==============================] - 54s 87ms/step - loss: 0.0946 -
accuracy: 0.9618 - val_loss: 0.3276 - val_accuracy: 0.8942 Epoch 7/10
625/625 [==============================] - 63s 101ms/step - loss: 0.0689 -
accuracy: 0.9740 - val_loss: 0.2580 - val_accuracy: 0.9112 Epoch 8/10
625/625 [==============================] - 64s 103ms/step - loss: 0.0475 -
accuracy: 0.9833 - val_loss: 0.2855 - val_accuracy: 0.9134 Epoch 9/10
625/625 [==============================] - 64s 102ms/step - loss: 0.0430 -
accuracy: 0.9840 - val_loss: 0.2983 - val_accuracy: 0.9130 Epoch 10/10
625/625 [==============================] - 54s 86ms/step - loss: 0.0403 -
accuracy: 0.9847 - val_loss: 0.3299 - val_accuracy: 0.9118
```



model.predict(test_input)

1/1 [==============================] - 1s 1s/step

array([[1.]], dtype=float32)



model.predict(test_input)

9. Program: Building a CNN model for object detection using a pre-trained architecture like YOLO.

Dependencies

```
import tensorflow as tf
import numpy as np
from PIL import Image, ImageDraw, ImageFont
from IPython.display import display
from seaborn import color_palette
import cv2
```

Model hyperparameters

```
_BATCH_NORM_DECAY = 0.9
_BATCH_NORM_EPSILON = 1e-05


_LEAKY_RELU = 0.1
_ANCHORS = [(10, 13), (16, 30), (33, 23),
            (30, 61), (62, 45), (59, 119),
            (116, 90), (156, 198), (373, 326)]
_MODEL_SIZE = (416, 416)
```

Model definition

```
def batch_norm(inputs, training, data_format):
    """Performs a batch normalization using a standard set of parameters."""
    return tf.layers.batch_normalization(
        inputs=inputs, axis=1 if data_format == 'channels_first' else 3,
        momentum=_BATCH_NORM_DECAY, epsilon=_BATCH_NORM_EPSILON,
        scale=True, training=training)

def fixed_padding(inputs, kernel_size, data_format):
    """ResNet implementation of fixed padding.

    Pads the input along the spatial dimensions independently of input size.

    Args:
        inputs: Tensor input to be padded.
        kernel_size: The kernel to be used in the conv2d or max_pool2d.
        data_format: The input format.
    Returns:
        A tensor with the same format as the input.
    """
    pad_total = kernel_size - 1
    pad_beg = pad_total // 2
    pad_end = pad_total - pad_beg

    if data_format == 'channels_first':
        padded_inputs = tf.pad(inputs, [[0, 0], [0, 0],
                                        [pad_beg, pad_end],
```

```
                                        [pad_beg, pad_end]])

    else:
            padded_inputs = tf.pad(inputs, [[0, 0], [pad_beg, pad_end],
                                        [pad_beg, pad_end], [0, 0]])
        return padded_inputs

    def conv2d_fixed_padding(inputs, filters, kernel_size, data_format, strides=1):
        """Strided 2-D convolution with explicit padding."""
        if strides > 1:
            inputs = fixed_padding(inputs, kernel_size, data_format)

        return tf.layers.conv2d(
            inputs=inputs, filters=filters, kernel_size=kernel_size,
            strides=strides, padding=('SAME' if strides == 1 else 'VALID'),
            use_bias=False, data_format=data_format)
```

Feature extraction: Darknet-53

```
    def darknet53_residual_block(inputs, filters, training, data_format,
                                 strides=1):
        """Creates a residual block for Darknet."""
        shortcut = inputs

        inputs = conv2d_fixed_padding(
            inputs, filters=filters, kernel_size=1, strides=strides,
            data_format=data_format)
        inputs = batch_norm(inputs, training=training, data_format=data_format)
        inputs = tf.nn.leaky_relu(inputs, alpha=_LEAKY_RELU)

        inputs = conv2d_fixed_padding(
            inputs, filters=2 * filters, kernel_size=3, strides=strides,
            data_format=data_format)
        inputs = batch_norm(inputs, training=training, data_format=data_format)
        inputs = tf.nn.leaky_relu(inputs, alpha=_LEAKY_RELU)

        inputs += shortcut
        return inputs

    def darknet53(inputs, training, data_format):
        """Creates Darknet53 model for feature extraction."""
        inputs = conv2d_fixed_padding(inputs, filters=32, kernel_size=3,
                                      data_format=data_format)
        inputs = batch_norm(inputs, training=training, data_format=data_format)
        inputs = tf.nn.leaky_relu(inputs, alpha=_LEAKY_RELU)
        inputs = conv2d_fixed_padding(inputs, filters=64, kernel_size=3,
                                      strides=2, data_format=data_format)
        inputs = batch_norm(inputs, training=training, data_format=data_format)
        inputs = tf.nn.leaky_relu(inputs, alpha=_LEAKY_RELU)

        inputs = darknet53_residual_block(inputs, filters=32, training=training,
                                          data_format=data_format)
        inputs = conv2d_fixed_padding(inputs, filters=128, kernel_size=3,
                                      strides=2, data_format=data_format)
```

```
    inputs = batch_norm(inputs, training=training, data_format=data_format)
    inputs = tf.nn.leaky_relu(inputs, alpha=_LEAKY_RELU)
    for _ in range(2):
        inputs = darknet53_residual_block(inputs, filters=64,
                                          training=training,
                                          data_format=data_format)

    inputs = conv2d_fixed_padding(inputs, filters=256, kernel_size=3,
                                  strides=2, data_format=data_format)
    inputs = batch_norm(inputs, training=training, data_format=data_format)
    inputs = tf.nn.leaky_relu(inputs, alpha=_LEAKY_RELU)

    for _ in range(8):
        inputs = darknet53_residual_block(inputs, filters=128,
                                  training=training, data_format=data_format)

    route1 = inputs
    inputs = conv2d_fixed_padding(inputs, filters=512, kernel_size=3,
                                  strides=2, data_format=data_format)
    inputs = batch_norm(inputs, training=training, data_format=data_format)
    inputs = tf.nn.leaky_relu(inputs, alpha=_LEAKY_RELU)

    for _ in range(8):
        inputs = darknet53_residual_block(inputs, filters=256,
                                    training=training,data_format=data_format)

    route2 = input
    inputs = conv2d_fixed_padding(inputs, filters=1024, kernel_size=3,
                                  strides=2, data_format=data_format)
    inputs = batch_norm(inputs, training=training, data_format=data_format)
    inputs = tf.nn.leaky_relu(inputs, alpha=_LEAKY_RELU)

    for _ in range(4):
        inputs = darknet53_residual_block(inputs, filters=512,
                                   training=training,data_format=data_format)
    return route1, route2, inputs
```

Convolution layers

```
def yolo_convolution_block(inputs, filters, training, data_format):
    """Creates convolution operations layer used after Darknet."""
    inputs = conv2d_fixed_padding(inputs, filters=filters, kernel_size=1,
                                  data_format=data_format)
    inputs = batch_norm(inputs, training=training, data_format=data_format)
    inputs = tf.nn.leaky_relu(inputs, alpha=_LEAKY_RELU)
    inputs = conv2d_fixed_padding(inputs, filters=2 * filters, kernel_size=3,
                                  data_format=data_format)
    inputs = batch_norm(inputs, training=training, data_format=data_format)
    inputs = tf.nn.leaky_relu(inputs, alpha=_LEAKY_RELU)
    inputs = conv2d_fixed_padding(inputs, filters=filters, kernel_size=1,
                                  data_format=data_format)
    inputs = batch_norm(inputs, training=training, data_format=data_format)
    inputs = tf.nn.leaky_relu(inputs, alpha=_LEAKY_RELU)

    inputs = conv2d_fixed_padding(inputs, filters=2 * filters, kernel_size=3,
                                  data_format=data_format)
```

```
    inputs = batch_norm(inputs, training=training, data_format=data_format)
    inputs = tf.nn.leaky_relu(inputs, alpha=_LEAKY_RELU)

    inputs = conv2d_fixed_padding(inputs, filters=filters, kernel_size=1,
                                  data_format=data_format)
    inputs = batch_norm(inputs, training=training, data_format=data_format)

    inputs = tf.nn.leaky_relu(inputs, alpha=_LEAKY_RELU)

    route = inputs

    inputs = conv2d_fixed_padding(inputs, filters=2 * filters, kernel_size=3,
                                  data_format=data_format)
    inputs = batch_norm(inputs, training=training, data_format=data_format)
    inputs = tf.nn.leaky_relu(inputs, alpha=_LEAKY_RELU)

    return route, inputs
```

Detection layers

```
def yolo_layer(inputs, n_classes, anchors, img_size, data_format):
    """Creates Yolo final detection layer.

    Detects boxes with respect to anchors.

    Args:
        inputs: Tensor input.
        n_classes: Number of labels.
        anchors: A list of anchor sizes.
        img_size: The input size of the model.
        data_format: The input format.

    Returns:
        Tensor output.
    """
    n_anchors = len(anchors)

    inputs = tf.layers.conv2d(inputs, filters=n_anchors * (5 + n_classes),
                              kernel_size=1, strides=1, use_bias=True,
                              data_format=data_format)
    shape = inputs.get_shape().as_list()
    grid_shape = shape[2:4] if data_format == 'channels_first' else shape[1:3]
    if data_format == 'channels_first':
        inputs = tf.transpose(inputs, [0, 2, 3, 1])
    inputs = tf.reshape(inputs, [-1, n_anchors * grid_shape[0] * grid_shape[1],
                                 5 + n_classes])

    strides = (img_size[0] // grid_shape[0], img_size[1] // grid_shape[1])

    box_centers, box_shapes, confidence, classes = \
        tf.split(inputs, [2, 2, 1, n_classes], axis=-1)

    x = tf.range(grid_shape[0], dtype=tf.float32)
    y = tf.range(grid_shape[1], dtype=tf.float32)
    x_offset, y_offset = tf.meshgrid(x, y)
    x_offset = tf.reshape(x_offset, (-1, 1))
```

```
    y_offset = tf.reshape(y_offset, (-1, 1))
    x_y_offset = tf.concat([x_offset, y_offset], axis=-1)
    x_y_offset = tf.tile(x_y_offset, [1, n_anchors])
    x_y_offset = tf.reshape(x_y_offset, [1, -1, 2])
    box_centers = tf.nn.sigmoid(box_centers)
    box_centers = (box_centers + x_y_offset) * strides

    anchors = tf.tile(anchors, [grid_shape[0] * grid_shape[1], 1])
    box_shapes = tf.exp(box_shapes) * tf.to_float(anchors)

    confidence = tf.nn.sigmoid(confidence)

    classes = tf.nn.sigmoid(classes)

    inputs = tf.concat([box_centers, box_shapes,
                        confidence, classes], axis=-1)

    return inputs
```

Upsample layer

```
def upsample(inputs, out_shape, data_format):

   """Upsamples to `out_shape` using nearest neighbor interpolation."""

   if data_format == 'channels_first':

      inputs = tf.transpose(inputs, [0, 2, 3, 1])

      new_height = out_shape[3]

      new_width = out_shape[2]

   else:

      new_height = out_shape[2]

      new_width = out_shape[1]

  inputs = tf.image.resize_nearest_neighbor(inputs, (new_height, new_width))

   if data_format == 'channels_first':

      inputs = tf.transpose(inputs, [0, 3, 1, 2])

      return inputs
```

```
def build_boxes(inputs):
    """Computes top left and bottom right points of the boxes."""
    center_x, center_y, width, height, confidence, classes = \
        tf.split(inputs, [1, 1, 1, 1, 1, -1], axis=-1)

    top_left_x = center_x - width / 2
    top_left_y = center_y - height / 2
    bottom_right_x = center_x + width / 2
```

```
    bottom_right_y = center_y + height / 2

    boxes = tf.concat([top_left_x, top_left_y,
                       bottom_right_x, bottom_right_y,
                       confidence, classes], axis=-1)
    return boxes

def non_max_suppression(inputs, n_classes, max_output_size, iou_threshold,
                        confidence_threshold):
    """Performs non-max suppression separately for each class.

    Args:
        inputs: Tensor input.
        n_classes: Number of classes.
        max_output_size: Max number of boxes to be selected for each class.
        iou_threshold: Threshold for the IOU.
        confidence_threshold: Threshold for the confidence score.
    Returns:
        A list containing class-to-boxes dictionaries
            for each sample in the batch.
    """
    batch = tf.unstack(inputs)
    boxes_dicts = []
    for boxes in batch:
        boxes = tf.boolean_mask(boxes, boxes[:, 4] > confidence_threshold)
        classes = tf.argmax(boxes[:, 5:], axis=-1)
        classes = tf.expand_dims(tf.to_float(classes), axis=-1)
        boxes = tf.concat([boxes[:, :5], classes], axis=-1)

        boxes_dict = dict()
        for cls in range(n_classes):
            mask = tf.equal(boxes[:, 5], cls)
            mask_shape = mask.get_shape()
            if mask_shape.ndims != 0:
                class_boxes = tf.boolean_mask(boxes, mask)
                boxes_coords, boxes_conf_scores, _ = tf.split(class_boxes,
                                                              [4, 1, -1],
                                                              axis=-1)
                boxes_conf_scores = tf.reshape(boxes_conf_scores, [-1])
                indices = tf.image.non_max_suppression(boxes_coords,
                                                       boxes_conf_scores,
                                                       max_output_size,
                                                       iou_threshold)
                class_boxes = tf.gather(class_boxes, indices)
                boxes_dict[cls] = class_boxes[:, :5]

        boxes_dicts.append(boxes_dict)

    return boxes_dicts
class Yolo_v3:
    """Yolo v3 model class."""

    def __init__(self, n_classes, model_size, max_output_size, iou_threshold,
                 confidence_threshold, data_format=None):
```

```
        """Creates the model.

        Args:
            n_classes: Number of class labels.
            model_size: The input size of the model.
            max_output_size: Max number of boxes to be selected for each class.
            iou_threshold: Threshold for the IOU.
            confidence_threshold: Threshold for the confidence score.
            data_format: The input format.

        Returns:
            None.
        """
        if not data_format:
            if tf.test.is_built_with_cuda():
                data_format = 'channels_first'
            else:
                data_format = 'channels_last'

        self.n_classes = n_classes
        self.model_size = model_size
        self.max_output_size = max_output_size
        self.iou_threshold = iou_threshold
        self.confidence_threshold = confidence_threshold
        self.data_format = data_format

    def __call__(self, inputs, training):
        """Add operations to detect boxes for a batch of input images.

        Args:
            inputs: A Tensor representing a batch of input images.
            training: A boolean, whether to use in training or inference mode.

        Returns:
            A list containing class-to-boxes dictionaries
                for each sample in the batch.
        """
        with tf.variable_scope('yolo_v3_model'):
            if self.data_format == 'channels_first':
                inputs = tf.transpose(inputs, [0, 3, 1, 2])

            inputs = inputs / 255

            route1, route2, inputs = darknet53(inputs, training=training,
                                               data_format=self.data_format)
            route, inputs = yolo_convolution_block(
                inputs, filters=512, training=training,
                data_format=self.data_format)
            detect1 = yolo_layer(inputs, n_classes=self.n_classes,
anchors=_ANCHORS[6:9],img_size=self.model_size,data_format=self.data_format)
            inputs = conv2d_fixed_padding(route, filters=256, kernel_size=1,
                                          data_format=self.data_format)

inputs=batch_norm(inputs,training=training,data_format=self.data_format)
```

```
            inputs = tf.nn.leaky_relu(inputs, alpha=_LEAKY_RELU)
            upsample_size = route2.get_shape().as_list()
            inputs = upsample(inputs, out_shape=upsample_size,
                              data_format=self.data_format)
            axis = 1 if self.data_format == 'channels_first' else 3
            inputs = tf.concat([inputs, route2], axis=axis)
            route, inputs = yolo_convolution_block(
                inputs, filters=256, training=training,
                data_format=self.data_format)
            detect2 = yolo_layer(inputs, n_classes=self.n_classes,
                                 anchors=_ANCHORS[3:6],
                                 img_size=self.model_size,
                                 data_format=self.data_format)
            inputs = conv2d_fixed_padding(route, filters=128, kernel_size=1,
                                          data_format=self.data_format)
            inputs = batch_norm(inputs, training=training,
                                data_format=self.data_format)
            inputs = tf.nn.leaky_relu(inputs, alpha=_LEAKY_RELU)
            upsample_size = route1.get_shape().as_list()
            inputs = upsample(inputs, out_shape=upsample_size,
                              data_format=self.data_format)
            inputs = tf.concat([inputs, route1], axis=axis)
            route, inputs = yolo_convolution_block(
                inputs, filters=128, training=training,
                data_format=self.data_format)
            detect3 = yolo_layer(inputs, n_classes=self.n_classes,
                                 anchors=_ANCHORS[0:3],
                                 img_size=self.model_size,
                                 data_format=self.data_format)

            inputs = tf.concat([detect1, detect2, detect3], axis=1)
            inputs = build_boxes(inputs)

            boxes_dicts = non_max_suppression(
                inputs, n_classes=self.n_classes,
                max_output_size=self.max_output_size,
                iou_threshold=self.iou_threshold,
                confidence_threshold=self.confidence_threshold)

            return boxes_dicts
def load_images(img_names, model_size):
    """Loads images in a 4D array.

    Args:
        img_names: A list of images names.
        model_size: The input size of the model.
        data_format: A format for the array returned
            ('channels_first' or 'channels_last').

    Returns:
        A 4D NumPy array.
    """
    imgs = []
```

```
    for img_name in img_names:
        img = Image.open(img_name)
        img = img.resize(size=model_size)
        img = np.array(img, dtype=np.float32)
        img = np.expand_dims(img, axis=0)
        imgs.append(img)

    imgs = np.concatenate(imgs)

    return imgs

def load_class_names(file_name):
    """Returns a list of class names read from `file_name`."""
    with open(file_name, 'r') as f:
        class_names = f.read().splitlines()
    return class_names

def draw_boxes(img_names, boxes_dicts, class_names, model_size):
    """Draws detected boxes.

    Args:
        img_names: A list of input images names.
        boxes_dict: A class-to-boxes dictionary.
        class_names: A class names list.
        model_size: The input size of the model.

    Returns:
        None.
    """
    colors = ((np.array(color_palette("hls", 80)) * 255)).astype(np.uint8)
    for num, img_name, boxes_dict in zip(range(len(img_names)), img_names,
                                         boxes_dicts):
        img = Image.open(img_name)
        draw = ImageDraw.Draw(img)
        font = ImageFont.truetype(font='../input/futur.ttf',
                                  size=(img.size[0] + img.size[1]) // 100)
        resize_factor = \
            (img.size[0] / model_size[0], img.size[1] / model_size[1])
        for cls in range(len(class_names)):
            boxes = boxes_dict[cls]
            if np.size(boxes) != 0:
                color = colors[cls]
                for box in boxes:
                    xy, confidence = box[:4], box[4]
                    xy = [xy[i] * resize_factor[i % 2] for i in range(4)]
                    x0, y0 = xy[0], xy[1]
                    thickness = (img.size[0] + img.size[1]) // 200
                    for t in np.linspace(0, 1, thickness):
                        xy[0], xy[1] = xy[0] + t, xy[1] + t
                        xy[2], xy[3] = xy[2] - t, xy[3] - t
                        draw.rectangle(xy, outline=tuple(color))
                    text = '{} {:.1f}%'.format(class_names[cls],
                                               confidence * 100)
                    text_size = draw.textsize(text, font=font)
```

```
                    draw.rectangle(
                        [x0, y0 - text_size[1], x0 + text_size[0], y0],
                        fill=tuple(color))
                    draw.text((x0, y0 - text_size[1]), text, fill='black',
                              font=font)

        display(img)
```

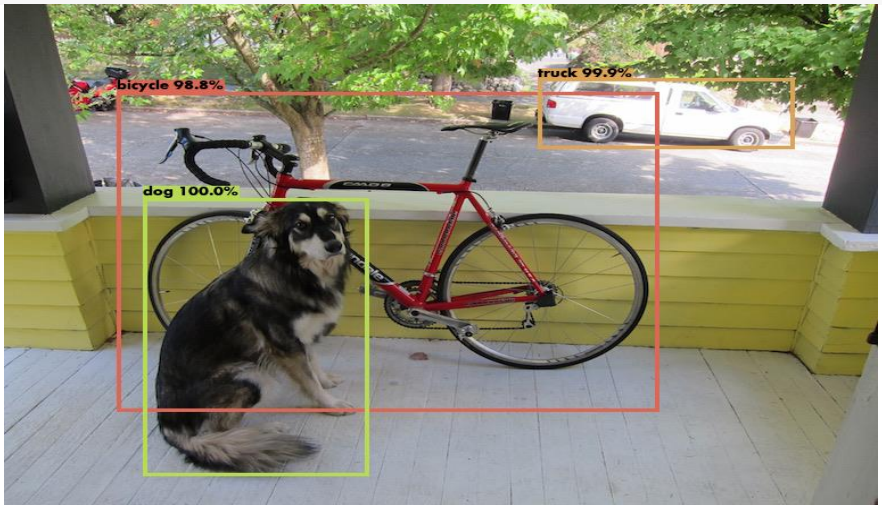
Converting weights to Tensorflow format

Running the model

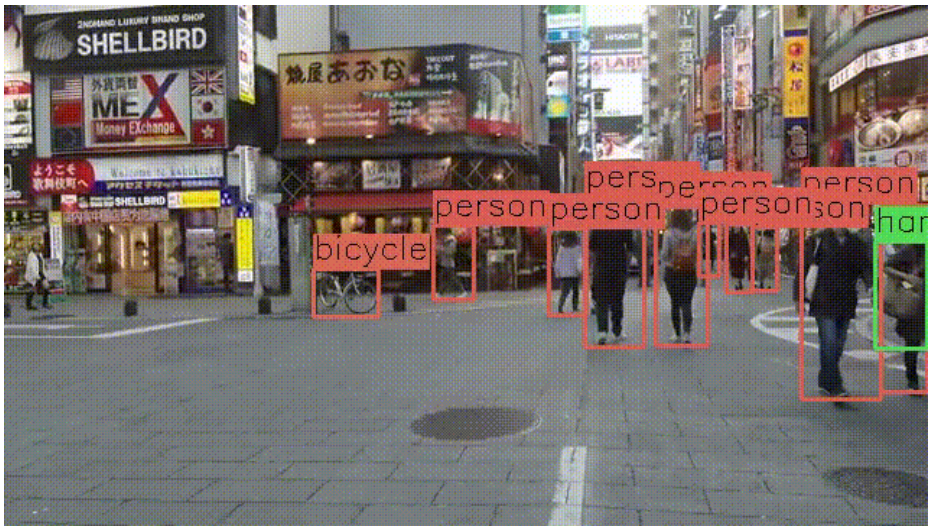
```
img_names = ['../input/dog.jpg', '../input/office.jpg']
for img in img_names: display(Image.open(img))
```

```
batch_size = len(img_names)
batch = load_images(img_names, model_size=_MODEL_SIZE)
class_names = load_class_names('../input/coco.names')
n_classes = len(class_names)
max_output_size = 10
iou_threshold = 0.5
confidence_threshold = 0.5

model = Yolo_v3(n_classes=n_classes, model_size=_MODEL_SIZE,
                max_output_size=max_output_size,
                iou_threshold=iou_threshold,
                confidence_threshold=confidence_threshold)

inputs = tf.placeholder(tf.float32, [batch_size, 416, 416, 3])

detections = model(inputs, training=False)

model_vars = tf.global_variables(scope='yolo_v3_model')
assign_ops = load_weights(model_vars, '../input/yolov3.weights')



with tf.Session() as sess:
    sess.run(assign_ops)
```

```
    detection_result = sess.run(detections, feed_dict={inputs: batch})

draw_boxes(img_names, detection_result, class_names, _MODEL_SIZE)
```



Videoprocessing



10. Program: Exploring different activation functions and comparing their effects on network performance.

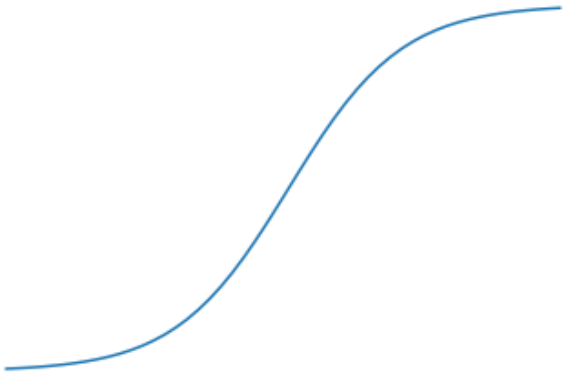
```
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD
import numpy as np

def sample_threeclass(n, ratio=0.8):
    np.random.seed(42)
    y_0 = np.random.randint(2, size=(n, 1))
    switch = (np.random.random(size=(n, 1)) <= ratio)
    y_1 = ~y_0 & switch
    y_2 = ~y_0 & ~switch
    y = np.concatenate([y_0, y_1, y_2], axis=1)
    X = y_0 + (np.random.normal(size=n) / 5)[np.newaxis].T
    return (X, y)
X_train, y_train = sample_threeclass(1000)
X_test, y_test = sample_threeclass(100)

Activation functions
Linear

clf = Sequential()
clf.add(Dense(3, activation='linear', input_shape=(1,), name='hidden'))
clf.add(Dense(3, activation='softmax', name='out'))
clf.compile(loss='categorical_crossentropy', optimizer=SGD(),
metrics=['accuracy'])
clf.fit(X_train, y_train, epochs=10, batch_size=16)

Epoch 1/10
1000/1000 [==============================] - 0s 400us/step - loss: 0.7590 -
acc: 0.7520
Epoch 2/10
1000/1000 [==============================] - 0s 79us/step - loss: 0.6491 -
acc: 0.8150
Epoch 3/10
1000/1000 [==============================] - 0s 75us/step - loss: 0.5746 -
acc: 0.8520
Epoch 4/10
1000/1000 [==============================] - 0s 78us/step - loss: 0.5208 -
acc: 0.8690
Epoch 5/10
1000/1000 [==============================] - 0s 77us/step - loss: 0.4805 -
acc: 0.8800
Epoch 6/10
1000/1000 [==============================] - 0s 73us/step - loss: 0.4500 -
acc: 0.8830
Epoch 7/10
```

```
1000/1000 [==============================] - 0s 77us/step - loss: 0.4266 -
acc: 0.8870
Epoch 8/10
1000/1000 [==============================] - 0s 75us/step - loss: 0.4083 -
acc: 0.8880
Epoch 9/10
1000/1000 [==============================] - 0s 76us/step - loss: 0.3936 -
acc: 0.8920
Epoch 10/10
1000/1000 [==============================] - 0s 77us/step - loss: 0.3817 -
acc: 0.8920

<keras.callbacks.History at 0x7f36a4fa9e10>
```

Logistic (sigmoid)

Non-linear activation functions come in many shapes. The most general class of non-linear activation function is so-called **sigmoid functions**, which are distinguishable by having an S-shaped value curve.

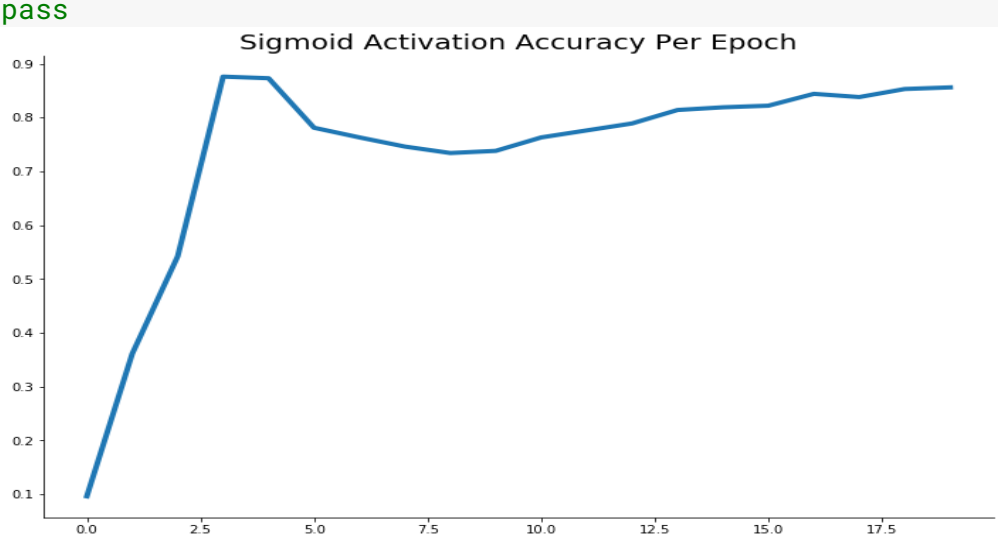
```
def logistic_func(x): return np.e**x/(np.e**x + 1)
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(np.arange(-5, 5, 0.2), [logistic_func(x) for x in np.arange(-5, 5,
0.2)])
plt.axis('off')
```



```
(-5.49, 5.290000000000008, -0.04256437797184291, 1.0410946577429678)
clf = Sequential()
clf.add(Dense(3, activation='sigmoid', input_shape=(1,), name='hidden'))
clf.add(Dense(3, activation='softmax', name='out'))
clf.compile(loss='categorical_crossentropy', optimizer=SGD(),
metrics=['accuracy'])

clf.fit(X_train, y_train, epochs=20, batch_size=16, verbose=0)
```

```
fig = plt.figure(figsize=(12, 8))
plt.plot(range(len(clf.history.history['acc'])), clf.history.history['acc'],
linewidth=4)
import seaborn as sns; sns.despine()
plt.title("Sigmoid Activation Accuracy Per Epoch", fontsize=20)
```

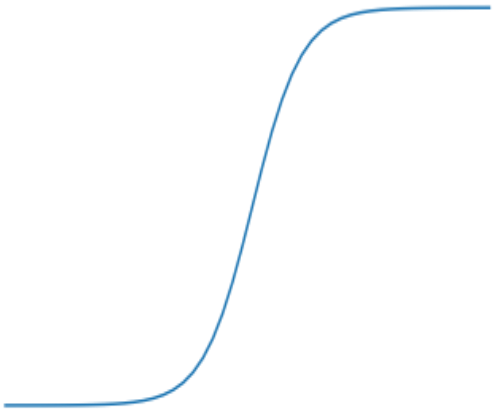
```
pass
```



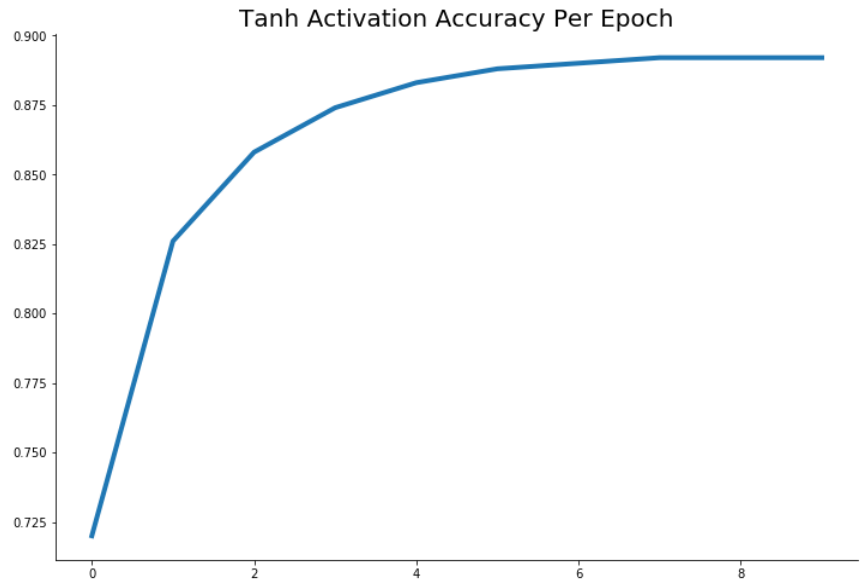
Tanh

An alternative to the logistic function is arctan or "tanh". This curve has different properties:

```
plt.plot(np.arange(-5, 5, 0.2), [np.tanh(x) for x in np.arange(-5, 5, 0.2)])
plt.axis('off')
```



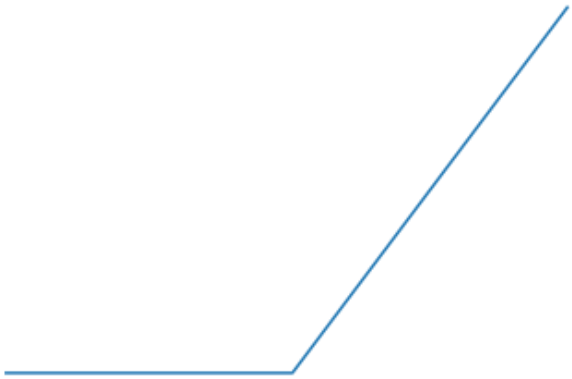
```
(-5.49, 5.290000000000008, -1.099897892060763, 1.0998532394989282)
clf = Sequential()
clf.add(Dense(3, activation='tanh', input_shape=(1,), name='hidden'))
clf.add(Dense(3, activation='softmax', name='out'))
clf.compile(loss='categorical_crossentropy', optimizer=SGD(),
metrics=['accuracy'])

clf.fit(X_train, y_train, epochs=10, batch_size=16, verbose=0)
fig = plt.figure(figsize=(12, 8))
plt.plot(range(len(clf.history.history['acc'])), clf.history.history['acc'],
linewidth=4)
import seaborn as sns; sns.despine()
plt.title("Tanh Activation Accuracy Per Epoch", fontsize=20)
pass
```



Rectified linear (ReLU)

The rectified linear function is a piecewise function which staples together a flat and a linear activation function. It is non-linear, but very much not sigmoid:

```
def relu(x):
    return 0 if x <= 0 else x
plt.plot(np.arange(-5, 5, 0.2), [relu(x) for x in np.arange(-5, 5, 0.2)])
plt.axis('off')
pass
```



```
clf = Sequential()
clf.add(Dense(3, activation='relu', input_shape=(1,), name='hidden'))
clf.add(Dense(3, activation='softmax', name='out'))
clf.compile(loss='categorical_crossentropy', optimizer=SGD(),
metrics=['accuracy'])
clf.fit(X_train, y_train, epochs=10, batch_size=16, verbose=0)
fig = plt.figure(figsize=(12, 8))
plt.plot(range(len(clf.history.history['acc'])), clf.history.history['acc'],
linewidth=4)
import seaborn as sns; sns.despine()
plt.title("ReLU Activation Accuracy Per Epoch", fontsize=20)
pass
```

11. Program: Write a program to Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

```
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 "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list
the files in the input directory
import os
print(os.listdir("../input/"))
# Any results you write to the current directory are saved as output.
# importing from keras

from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import adam
from sklearn.model_selection import train_test_split
['Attachment_1556072961 - Copy.csv']
Using TensorFlow backend.
Data Exploration
# loading the digit dataset
df = np.loadtxt("../input/Attachment_1556072961 - Copy.csv")#, delimiter=",")
# split into input (X) and output (Y) variables
# split into input and output variables
X = df[:,1:256]
Y = df[:,0]
```

Splitting the for training and testing

```
# seed for reproducing same results
seed = 20
np.random.seed(seed)
# split the data into training (80%) and testing (20%)
(X_train, X_test, Y_train, Y_test) = train_test_split(X, Y, test_size=0.20,
random_state=seed)
```

Model Creating

```
# create the model
model = Sequential()
model.add(Dense(100, input_dim=255, init='uniform', activation='relu'))
model.add(Dense(100, init='uniform', activation='relu'))
model.add(Dense(1, init='uniform', activation='sigmoid'))
# compile the model
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# fit the model
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10,
batch_size=5)#, verbose=0)

Train on 5832 samples, validate on 1459 samples
Epoch 1/10
5832/5832 [==============================] - 2s 386us/step - loss: -46.9793
- acc: 0.2262 - val_loss: -47.8293 - val_acc: 0.2680
```

```
Epoch 2/10
5832/5832 [==============================] - 2s 283us/step - loss: -48.6105
- acc: 0.2723 - val_loss: -48.1041 - val_acc: 0.2981
Epoch 3/10
5832/5832 [==============================] - 2s 285us/step - loss: -48.7469
- acc: 0.2812 - val_loss: -47.6877 - val_acc: 0.2536
Epoch 4/10
5832/5832 [==============================] - 2s 285us/step - loss: -48.8135
- acc: 0.2853 - val_loss: -47.9958 - val_acc: 0.3036
Epoch 5/10
5832/5832 [==============================] - 2s 282us/step - loss: -48.8432
- acc: 0.2867 - val_loss: -47.9490 - val_acc: 0.3043
Epoch 6/10
5832/5832 [==============================] - 2s 286us/step - loss: -48.8382
- acc: 0.2865 - val_loss: -48.0402 - val_acc: 0.2872
Epoch 7/10
5832/5832 [==============================] - 2s 286us/step - loss: -48.8358
- acc: 0.2864 - val_loss: -48.1838 - val_acc: 0.3002
Epoch 8/10
5832/5832 [==============================] - 2s 285us/step - loss: -48.8907
- acc: 0.2898 - val_loss: -48.2231 - val_acc: 0.3050
Epoch 9/10
5832/5832 [==============================] - 2s 284us/step - loss: -48.8796
- acc: 0.2891 - val_loss: -48.1436 - val_acc: 0.3016
Epoch 10/10
5832/5832 [==============================] - 2s 285us/step - loss: -48.9255
- acc: 0.2922 - val_loss: -48.1886 - val_acc: 0.2995
```

Out[4]:

Evaluation

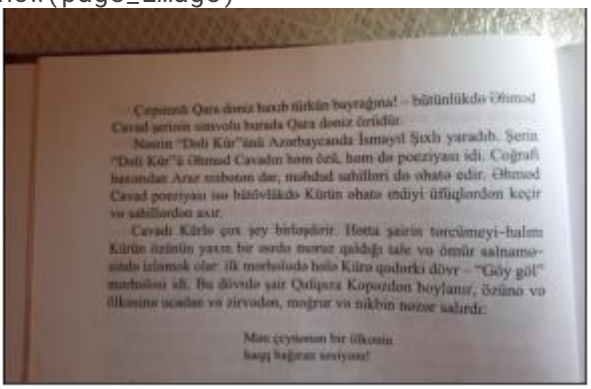
```
# evaluate the model
scores = model.evaluate(X_test, Y_test) #29.27
print("Accuracy: %.2f%%" % (scores[1]*100))
1459/1459 [==============================] - 0s 23us/step
Accuracy: 29.95%
```

12. Program: Implement a program for Basic image operations

```
# Importing modules
import cv2
import numpy as np
import matplotlib.pyplot as plt
def show(img):
    img=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.imshow(img, cmap='gray')
    plt.xticks([]), plt.yticks([])  # to hide tick values on X and Y axis
    plt.show()
#This function will plot two images side by side
def plot_image(image_1, image_2,title_1="Orignal",title_2="New Image"):
    plt.figure(figsize=(10,10))
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(image_1, cv2.COLOR_BGR2RGB))
    plt.title(title_1)
    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(image_2, cv2.COLOR_BGR2RGB))
    plt.title(title_2)
    plt.show()
```

Read and Display Image

```
path=r"/kaggle/input/images-for-opencv"
page_image=cv2.imread(path+r'/book.jpeg')
show(page_image)
```

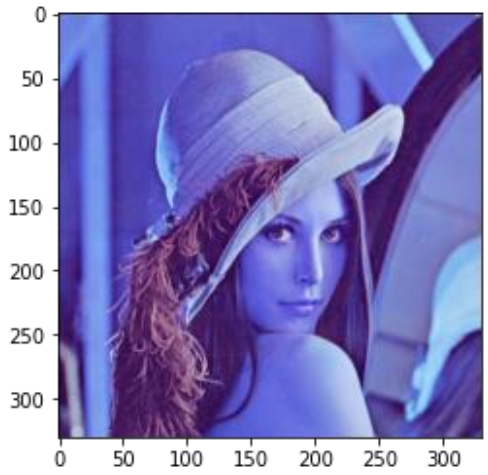


```
print(page_image)
```

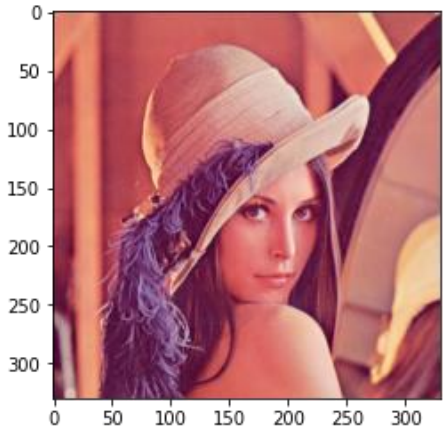
```
[[[ 20  39  82]
  [ 21  40  83]
  [ 21  40  83]
  ...
  [105 127 155]
  [107 129 157]
  [111 133 161]]

 [[ 20  39  82]
  [ 21  40  83]
  [ 21  40  83]
  ...
  [104 126 154]
  [106 128 156]
  [109 131 159]]
```

```
 [[ 21  40  83]
  [ 21  40  83]
  [ 21  40  83]
  ...
  [104 127 153]
  [105 128 154]
  [107 130 156]]
 ...
 [[ 69  80 110]
  [ 69  80 110]
  [ 69  80 110]
  ...
  [169 160 163]
  [169 160 163]
  [168 159 162]]

 [[ 69  80 110]
  [ 69  80 110]
  [ 69  80 110]
  ...
  [170 161 164]
  [169 160 163]
  [169 160 163]]

 [[ 69  80 110]
  [ 69  80 110]
  [ 69  80 110]
  ...
  [170 161 164]
  [169 160 163]
  [169 160 163]]]
# Checkin the Type of Image
print("Type of image is {}".format(type(page_image)))
print(len(page_image))
960
print(len(page_image[0]))
1280
print(len(page_image[0][0]))
3
#Dimension for image
print("Dimension for image is {}".format(page_image.shape))
Dimension for image is (960, 1280, 3)
print(page_image.dtype)
uint8
print(page_image.size)
3686400
page_image.max()
255
page_image.min()
0
length=page_image.shape[0]
```

```
width=page_image.shape[1]
channels=page_image.shape[2]
print("Length of image is {}".format(length))
print("width of image is {}".format(width))
print("No of channels is {}".format(channels))
Length of image is 960
width of image is 1280
No of channels is 3
lenna=cv2.imread(path+r'/Lenna.png')
plt.imshow(lenna)
plt.show()
```

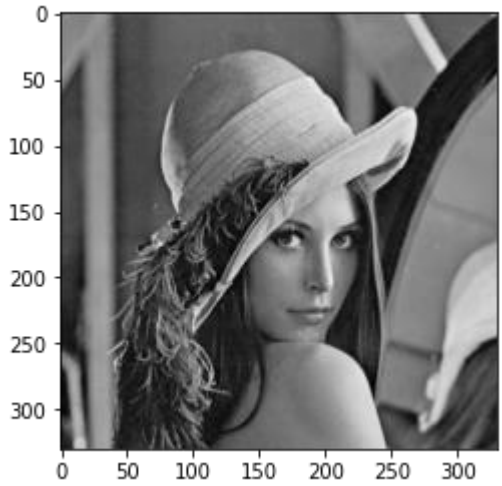



```
new_image=cv2.cvtColor(lenna, cv2.COLOR_BGR2RGB)

plt.imshow(new_image)
plt.show()
```




```
# get grayscale image
def get_grayscale(image):
    return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
image_gray=get_grayscale(new_image)
show(image_gray)
```

```
image_gray.shape
(330, 330)
plt.imshow(image_gray, cmap='gray')
plt.show()
```



Indexing

```
rows = 256
show(lenna[0:rows, :, :])
```

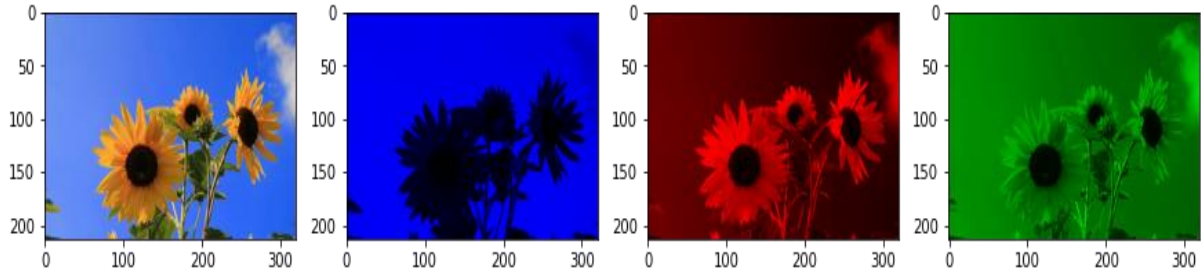
```
columns = 256
show(lenna[:,0:columns,:])
```

Cropping an Image

```
"""
Cropping image where following
Start Row (y1)
Start Column(x1)
End row(y1+h)
End columns(x1+w)
"""
###################################
"""
def crop_np(image):
    y=375
    x=300
    h=425
    w=645

    cropped_img=image[y:y+h, x:x+w]  ## ROI Extraction from Image

    return cropped_img
"""
def cropping(image):
    height, width = image.shape[:2]

    # Let's get the starting pixel coordiantes (top  left of cropping rectangle)
    start_row, start_col = int(height * .31), int(width * 0.17)

    # Let's get the ending pixel coordinates (bottom right)
    end_row, end_col = int(height * .99), int(width * .94)

    # Simply use indexing to crop out the rectangle we desire
    cropped = image[start_row:end_row , start_col:end_col]

    return cropped
crop_image=cropping(page_image)

plot_image(page_image, crop_image, title_2='Crop')
flowers=cv2.imread(path+r'/sunflower.jpg')
show(flowers)
```

RGB Color Space

```
#Converting BGR color space into RGB color space
rgb_image = cv2.cvtColor(flowers, cv2.COLOR_BGR2RGB)
image_r=rgb_image.copy()
image_g=rgb_image.copy()
image_b=rgb_image.copy()
For blue color channel
# making R channel zero
#making G channel zero
image_b[:, :, :2]=0

# making B channel zero
#making G channel zero
image_r[:, :, 1:]=0
# making R channel zero
#making B channel zero
image_g[:, :, [0, 2]]=0
f, axes = plt.subplots(1, 4, figsize = (15,15))
images= [flowers, image_b, image_r, image_g]

i = 0
for ax in axes:
    if i==0:
        ax.imshow(cv2.cvtColor(images[i], cv2.COLOR_BGR2RGB))

    else:
        ax.imshow(images[i])
    i+=1
```



Copying Images

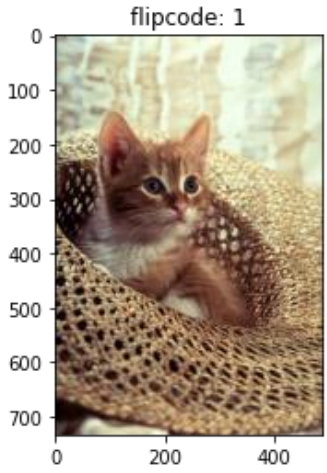
```
A = lenna.copy()
show(A)
```

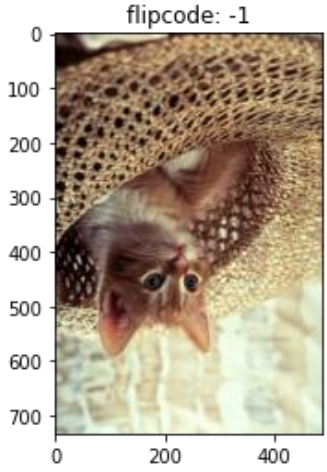
Manipulating Images

```
cat = cv2.imread(path+r"/cat.png")
show(cat)
```

```
width, height,C=cat.shape
print('width, height, C',width, height, C)
array_flip = np.zeros((width, height,C),dtype=np.uint8)
for i,row in enumerate(cat):
        array_flip[width-1-i,:,:]=row
show(array_flip)
```

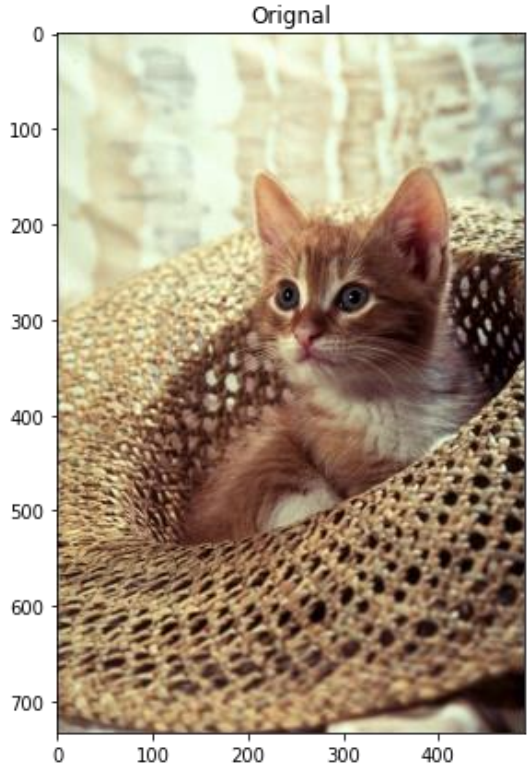
```
for flipcode in [0,1,-1]:
    im_flip =  cv2.flip(cat,flipcode )
    plt.imshow(cv2.cvtColor(im_flip,cv2.COLOR_BGR2RGB))
    plt.title("flipcode: "+str(flipcode))
    plt.show()
```

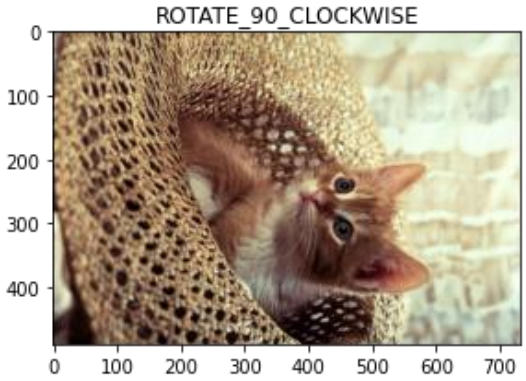


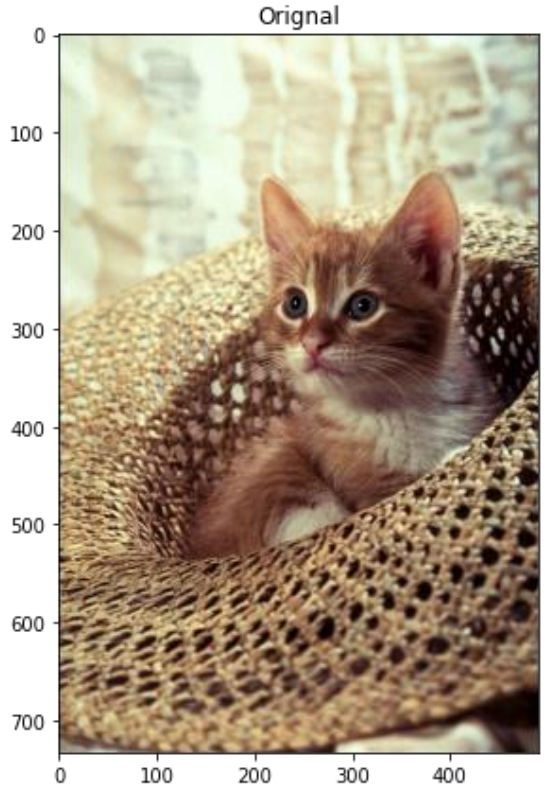


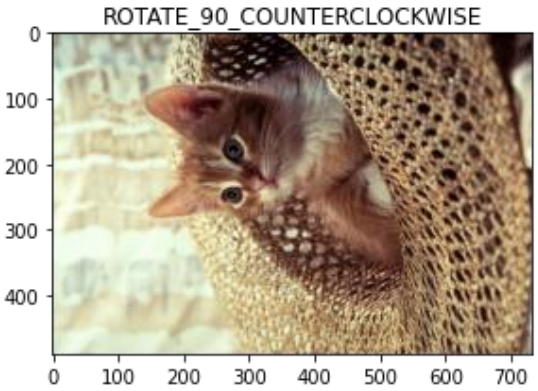

```
im_flip = cv2.rotate(cat, 0)
show(im_flip)
```

```
flip =
{"ROTATE_90_CLOCKWISE":cv2.ROTATE_90_CLOCKWISE,"ROTATE_90_COUNTERCLOCKWISE":cv2.
ROTATE_90_COUNTERCLOCKWISE,"ROTATE_180":cv2.ROTATE_180}
```









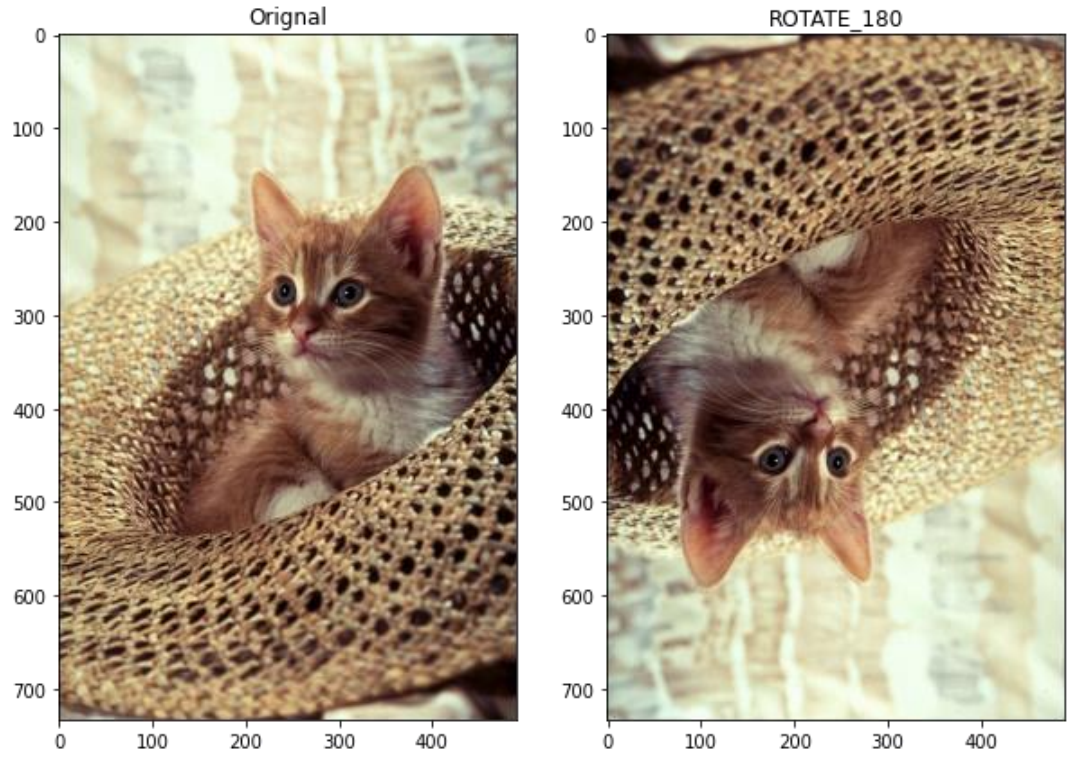
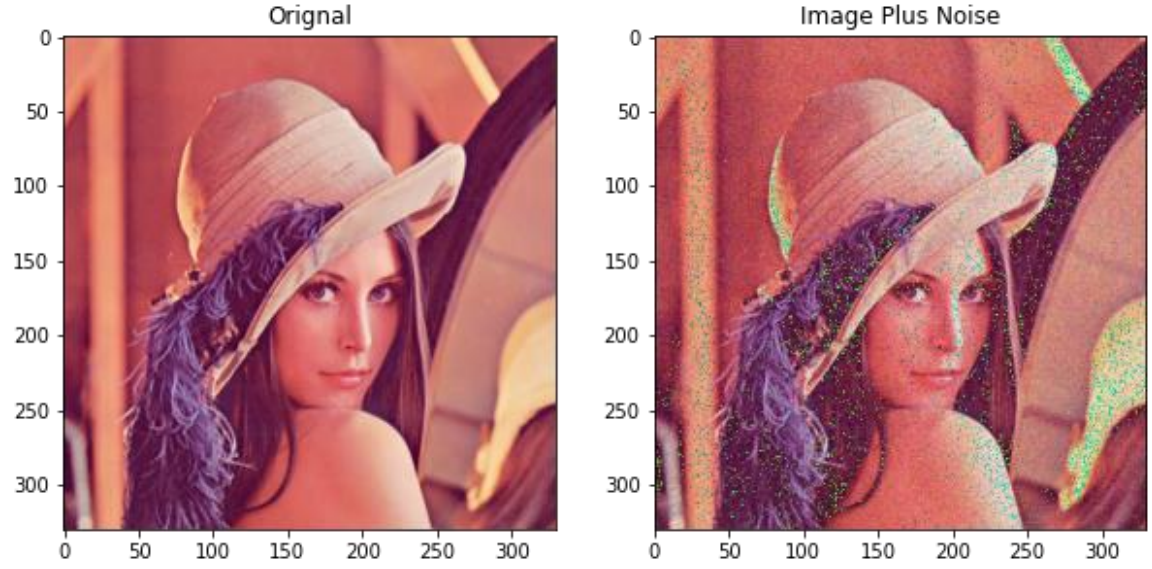


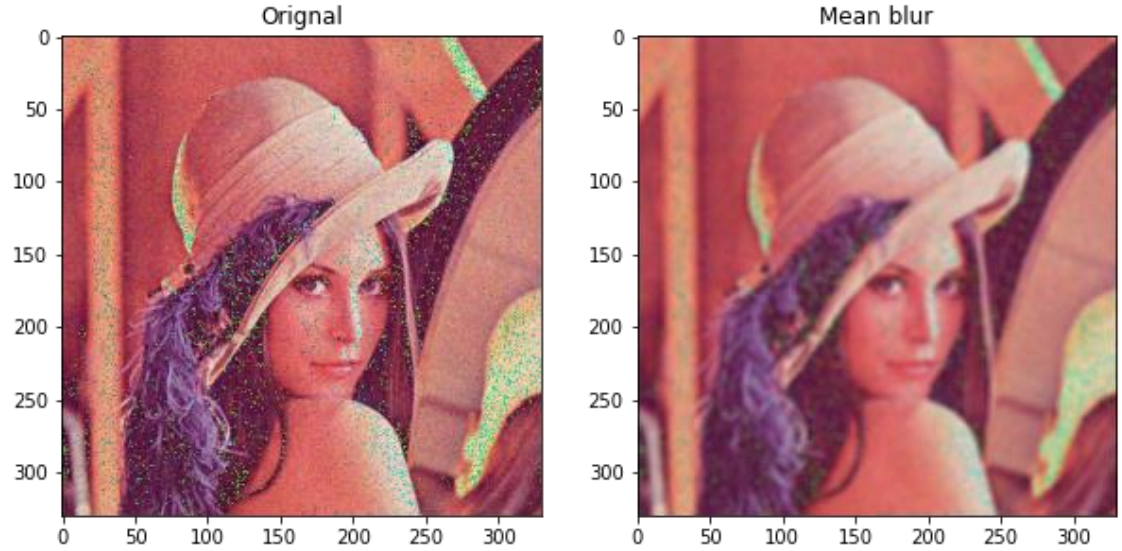
Image Smoothing

```
# Get the number of rows and columns in the image
rows, cols,_= lenna.shape
# Creates values using a normal distribution with a mean of 0 and standard
deviation of 15, the values are converted to unit8 which means the values are
between 0 and 255
noise = np.random.normal(0,15,(rows,cols,3)).astype(np.uint8)
# Add the noise to the image
noisy_image = lenna + noise
# Plots the original image and the image with noise using the function defined at
the top
plot_image(lenna, noisy_image, title_1="Orignal",title_2="Image Plus Noise")
```

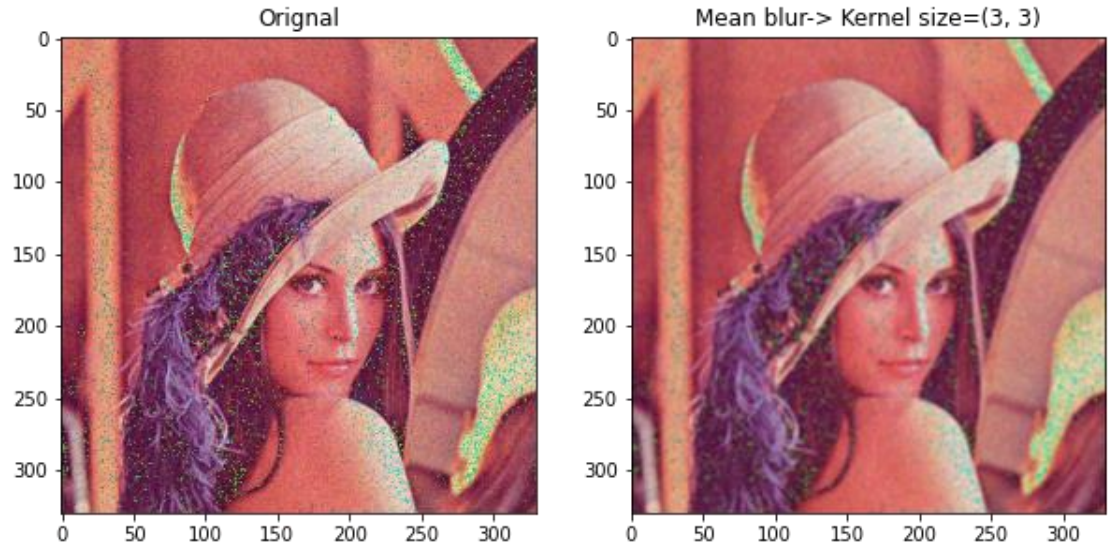


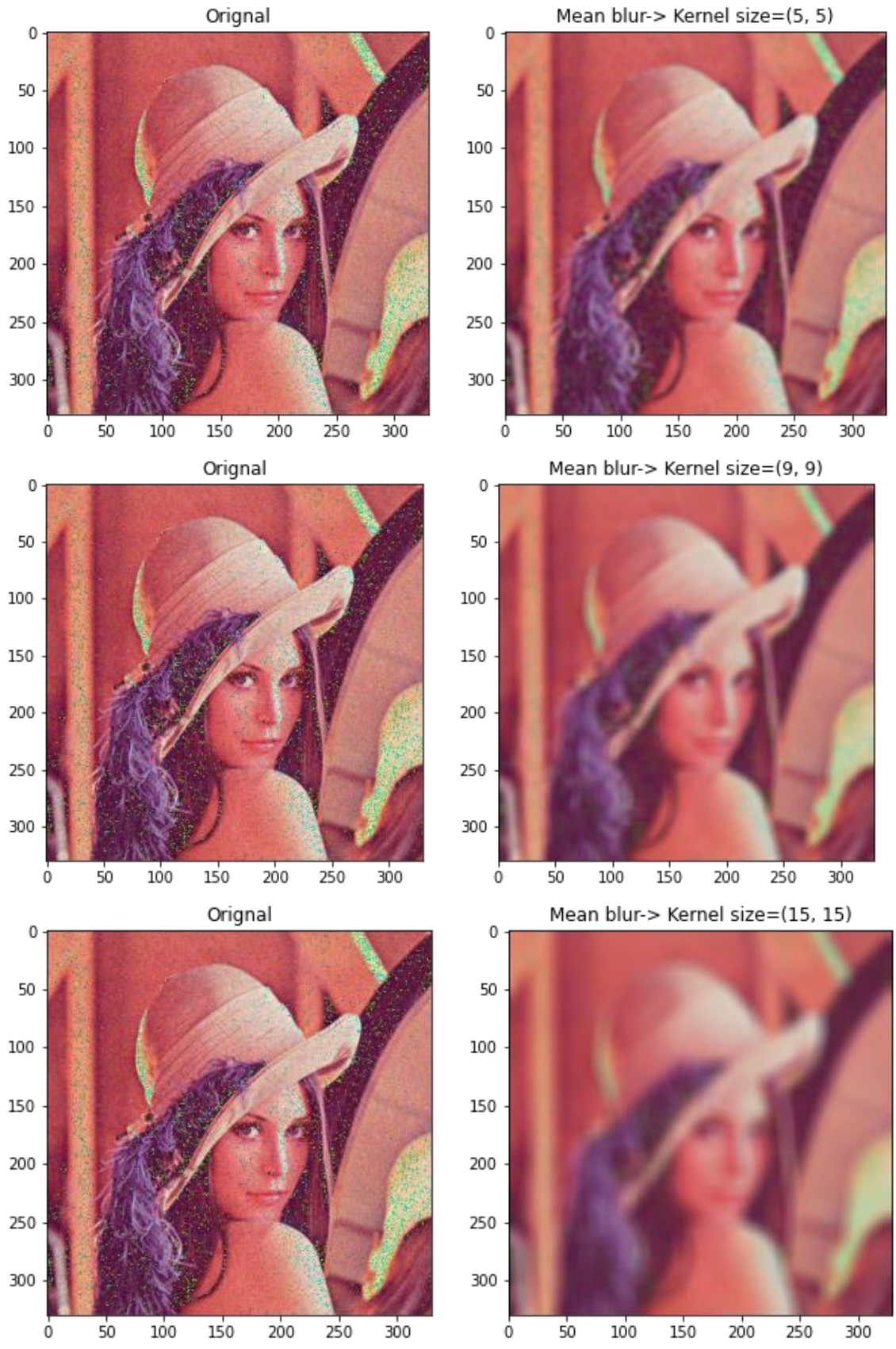

```
def mean_blur(img, kernel=(5, 5)):
```

```
blur = cv2.blur(img, kernel)

    return blur
In [55]:
linkcode
image_mean=mean_blur(noisy_image, (5, 5))

plot_image(noisy_image, image_mean, title_2='Mean blur')
```




```
kernel_sizes = [(3, 3), (5, 5), (9, 9), (15, 15)]

for kX, kY in kernel_sizes:
    image_mean=mean_blur(noisy_image, (kX, kY))

    plot_image(noisy_image, image_mean, title_2=f'Mean blur-> Kernel size=({kX},
{kY})')
```




Orignal
Mean blur-> Kernel size=(5, 5)
Orignal
Mean blur-> Kernel size=(9, 9)
Orignal
Mean blur-> Kernel size=(15, 15)