Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 03/22/2021

HOMEWORK #3: Image Classification

CSCI 543: Assignment 3

Problem #2: Classification of Images of Flowers

Report:

Classification of Images of Flowers

Abstract

I implement and run a tutorial (with some modification) on image classification via machine learning(ML) deep neural network. More specifically, I apply convolutional neural network on a flower images dataset and then train this ConvNet model to classify flower images into five flower groups. These flower groups(classes) are (i) Daisy, (ii) Dandelion, (iii) Roses, (iv) Sunflowers, and (v) Tulips. This image dataset stored on disk which are in jpg format. I build a convolutional network with the following layers (i) rescaling (input) layer, (ii) 2D convolutional layer, (iii) 2D max pooling layer, (iv) flatten layer, and (v) dense layer.   I perform this implementation of image classification using python programming language. Besides, I am using Keras, TensorFlow etc. for image preprocessing, image reshaping, image rescaling and building this Convolution Neural Network model for flower image classification. This ML ConvNet model can analyze images from the dataset and able to detect flower types from those images. Thus it performs image classification to separate flower type from images and then predict type of flowers. With 10 epochs this model shows 98% accuracy during training the model and only 66% accuracy during the validation process. This make my ML model "overfitting" with this flower images dataset. To overcome overfitting, I reconstruct the Convolutional Neural Network but this time consider "Data Augmentation" to generate additional training data and use "Dropout" layer to randomly dropout 20% data from the modified flower images dataset. Now, with 15 epochs the modified ML model shows 80% accuracy during training the model and 74% accuracy during the validation process which is much closer than before. Thus, I reduce overfitting this model for data prediction. This model shows impressive confidence for image data prediction with multiple classes.

TensorFlow tutorial link: https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/classification.ipynb

My code in google drive (Colab):

https://colab.research.google.com/drive/1fEST91Ta3zJGrBGbkQXekpCnEeikTdhG?usp=sharing

Tools: Python programming language, Keras, TensorFlow, Google Colab, Google Drive, Google Cloud Storage.

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 03/22/2021

Dataset Details

I am using a dataset of about 3700 images of flowers. This dataset contains 5 sub-directories named (i) Daisy, (ii) Dandelion, (iii) Roses, (iv) Sunflowers, and (v) Tulips. These are the five image classes used in this machine learning ConvNet model. The zip file of this image dataset is stored on google cloud storage system. URL of this unstructured image dataset is - `https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz`. Machine learning model usually required three types of dataset to perform any experimental analysis, (i) 'train dataset' to train the model, (ii) 'validation dataset' for evaluation the quality of the model, (iii) 'test dataset' to test the model after the model has gone through the validation process. I don't go through testing process, that's why; 'test dataset' is not required in this work. Instead I will test this model with new example image. So, I divided this image dataset into train and validation data by 8:2 splitting. I am using 80% of images for training the model and 20% of images for the validation purpose. The distribution of the dataset is shown in the following.

```
Found 3670 files belonging to 5 classes.
Using 2936 files for training.
Using 734 files for validation.
```

Data Preprocessing

Data preprocessing and normalization makes data suitable to fit into the ML model. I am using image_dataset_from_directory API that will read images from the input dataset and convert them to float32 tensors, and feed them (with their labels) to this convolutional network. To normalize image data, I use min-max normalization method. This method rescaled all images by 1./255 which limit the range within 0 to 1. Besides, to be consistent all image is resized to 180 x180 (height x width). I am inserting 32 images at a time by setting batch size at 32. In short each image batch is a tensor of shape (32, 180, 180, 3) == (batch, height, width, RGB color channel). To configure the dataset for performance, I am using two methods when loading the dataset. These are in the following.

i. **Dataset.cache()** keeps the images in memory after they're loaded off disk during the first epoch. This will ensure the dataset does not become a bottleneck while training the model.
ii. **Dataset.prefetch()** overlaps data preprocessing and model execution while training.

The ML Approach

I am performing image classification using Convolutional Neural Network that is a well-known Machine Learning (ML) approach for image data analysis. I build a Convolutional Deep Neural Network with multiple layers to train and evaluate this ML model with the image dataset. This is a ten-layer Deep Neural Network ML model. These are - One Rescaling layer, three convolution layers, three max pooling layers, one flatten layer, two dense layers. These ten layer sequentially build the classifier to classify flower images into five classes. Model layers are given below.

i. **Rescaling(Input) Layer:** This layer takes image data as input for further analysis. All image data must be in same size before fed into the model. In this layer each image is rescaled by 1./255 and reshaped by 150x150x3 (height, width, RGB color channel).
ii. **Convolution Layer(2D):** This is a convolutional layer which support 2D images. I am using 3 stack of Convolution and Max pooling layer in this ML model. The first convolution layer extracts 16 filters, 2nd one extract 32 filters and last one extract 64 filters. Each of these layers consider 3x3 matrix window.
iii. **Max Pooling Layer (2D):** Max Polling layer supports 2D images. This extract maximum vector from previous layer matrix and reduce the size using 2x2 matrix window. I am using 3 max pooling layers with 3 convolution layers.
iv. **Flatten Layer:** This layer map features from the previous layer to a 1-dimensinal tensor so that this can be connected with a fully connected layer.
v. **Dense Layer:** Output from the previous layer piped through 128 hidden neurons.
vi. **Dense(Output) Layer:** Final five neurons for five flower classes.

The sequence of these layers are illustrated in the following figure 1. The figure 1 also shows output shape of each layer and total number of trainable parameters generated from this machine learning model.

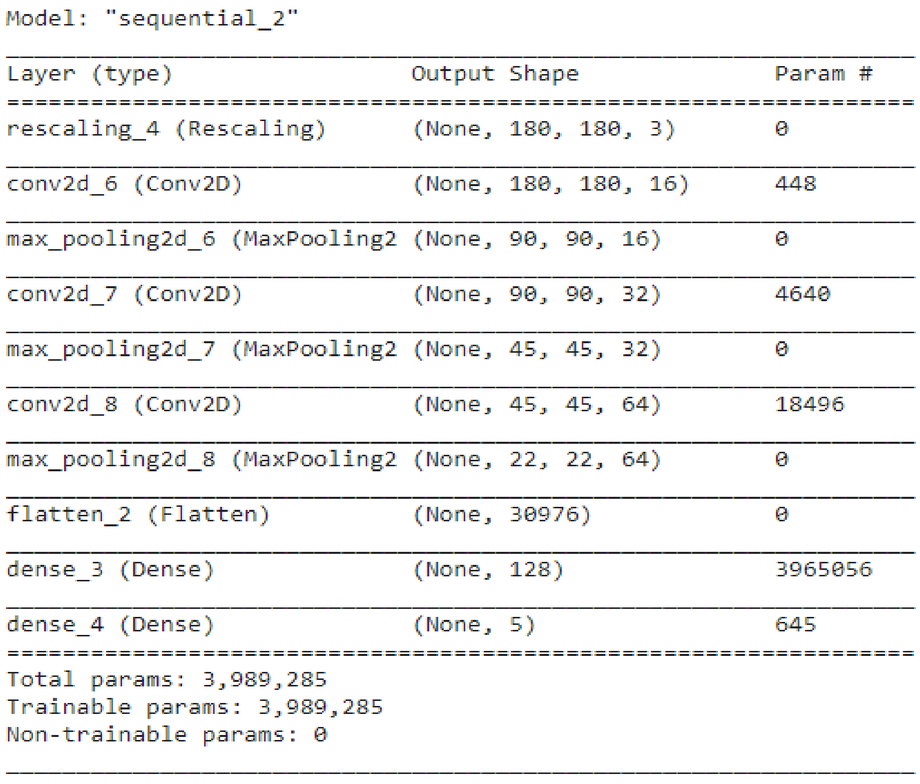

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
rescaling_4 (Rescaling)      (None, 180, 180, 3)       0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 180, 180, 16)      448       
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 90, 90, 16)        0         
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 90, 90, 32)        4640      
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 45, 45, 32)        0         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 45, 45, 64)        18496     
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 22, 22, 64)        0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 30976)             0         
_________________________________________________________________
dense_3 (Dense)              (None, 128)               3965056   
_________________________________________________________________
dense_4 (Dense)              (None, 5)                 645       
=================================================================
Total params: 3,989,285
Trainable params: 3,989,285
Non-trainable params: 0
_________________________________________________________________
```

Figure 1: Model Summary.

Relu activation function is used within these layers. Relu overcome vanishing gradient prolem, learning faster and better. Relu is default for Convolutional Neural Network. **Adam** optimizer is used as optimization algorithm in this model.

The complete procedure of this ML approach for Image Classification is shown below.

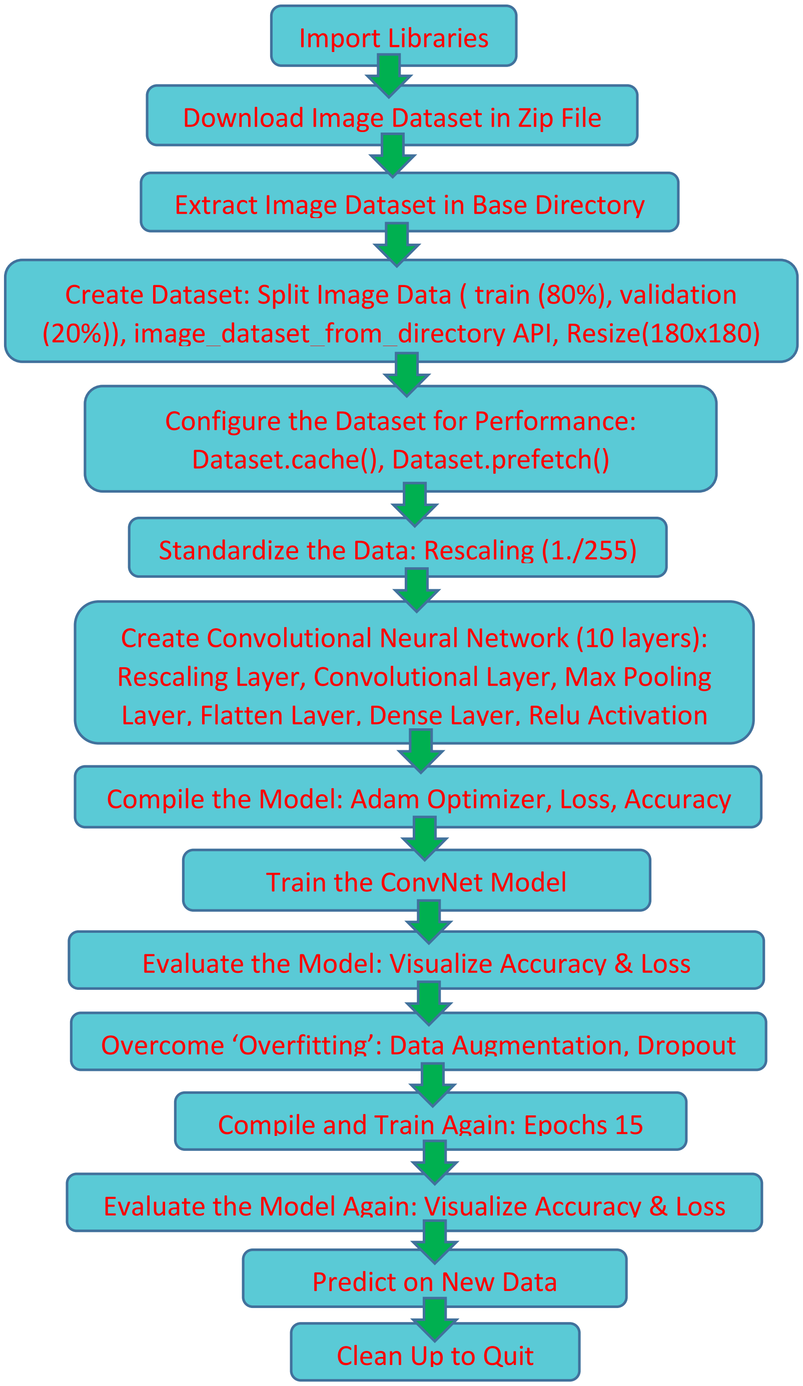

Import Libraries
Download Image Dataset in Zip File
Extract Image Dataset in Base Directory
Create Dataset: Split Image Data (train (80%), validation (20%)), image_dataset_from_directory API, Resize(180x180)
Configure the Dataset for Performance: Dataset.cache(), Dataset.prefetch()
Standardize the Data: Rescaling (1./255)
Create Convolutional Neural Network (10 layers): Rescaling Layer, Convolutional Layer, Max Pooling Layer, Flatten Layer, Dense Layer, Relu Activation
Compile the Model: Adam Optimizer, Loss, Accuracy
Train the ConvNet Model
Evaluate the Model: Visualize Accuracy & Loss
Overcome 'Overfitting': Data Augmentation, Dropout
Compile and Train Again: Epochs 15
Evaluate the Model Again: Visualize Accuracy & Loss
Predict on New Data
Clean Up to Quit

Figure 2: Complete procedure of this ConvNet ML approach for Image Classification

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 03/22/2021

ML Model Performance

To evaluate the performance of this ConvNet model, I perform loss analysis for the training and validation dataset using ‘`SparseCategoricalCrossentropy`’ loss function and also determine the accuracy for these two datasets. Figure 3 presents the accuracy and loss during training and validation phase. Maximum accuracy from this model is given below.

```
Training Data Accuracy: 0.9833106398582458 (98%)
Validation Data Accuracy 0.664850115776062 (66%)
```

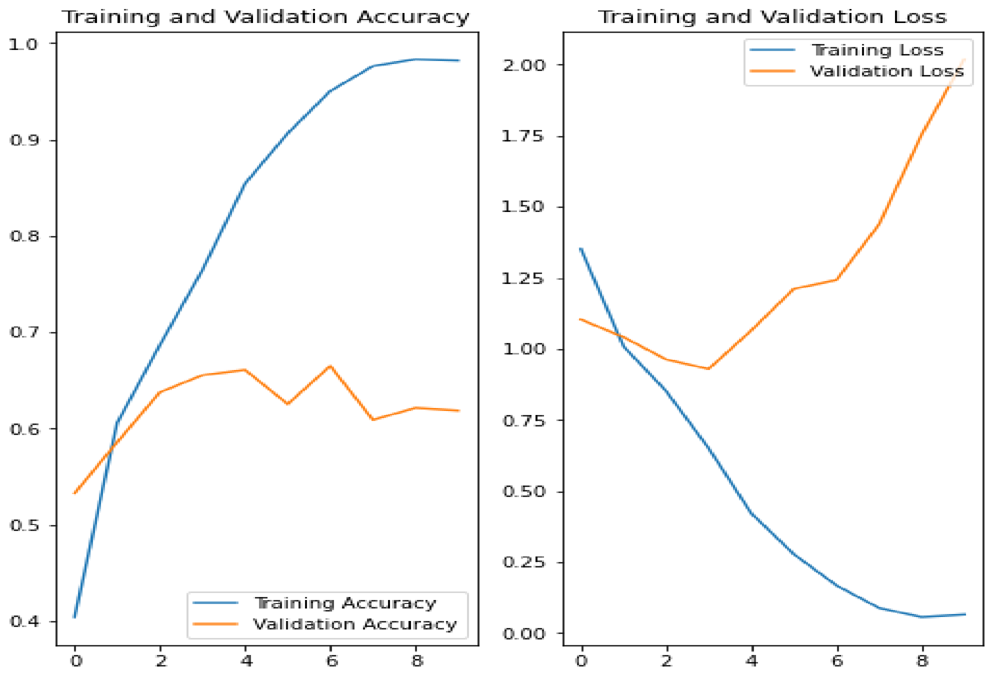


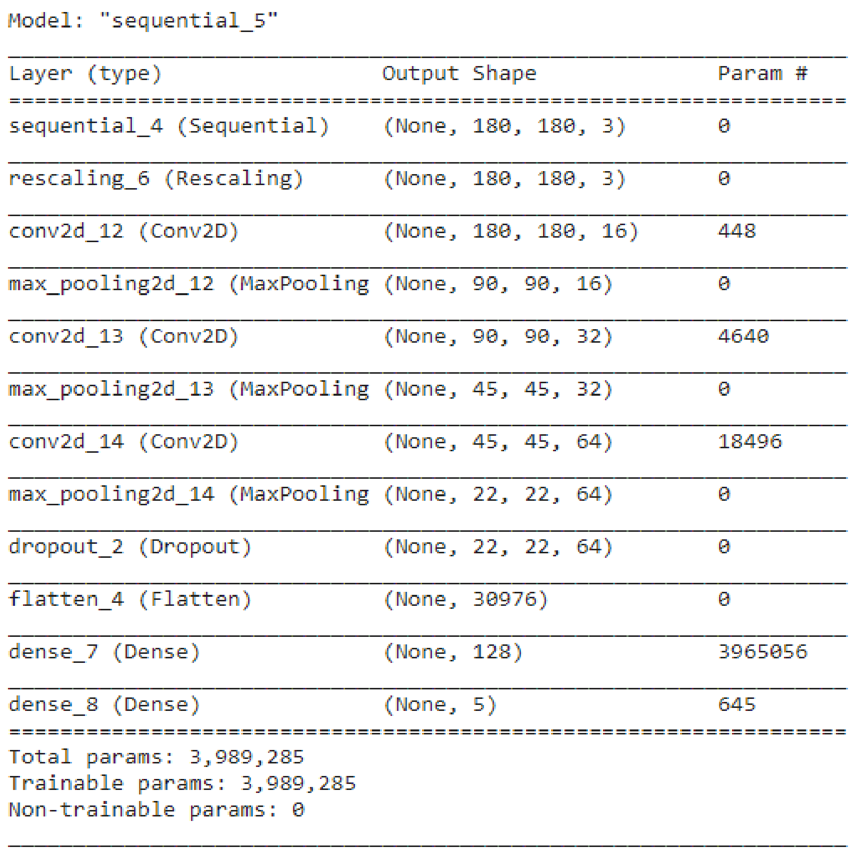
Figure 3: Training and Validation Accuracy(left), Loss(right)

From figure 3 it is clear that this ML model is “overfitting”. Figure 3 shows training accuracy of this model is very close to 98% while during validation accuracy is near to 66%. This is because this model considers some features which are irrelevant for image prediction. When there are a small number of training examples, the model sometimes learns from noises or unwanted details from training examples. To overcome this “overfitting” one easy solution is train this model with more data. I am using following two methods to overcome “overfitting” of this model.

i. Reduce overfitting by **data augmentation**. This approach generates additional training data from the existing examples by augmenting them using random transformations that yield believable-looking images. For example, rotate image, zoom in or zoom out etc.
ii. Reduce overfitting by **“dropout”** is a form of regularization. Applying dropout to a layer means randomly drops out (by setting the activation to zero) a number of output units from the layer during the training process.

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 03/22/2021

I reconstruct the Convolutional Neural Network but this time consider “Data Augmentation” to generate additional training data and use “Dropout” layer to randomly dropout 20% data from the modified flower image dataset. The modified ConvNet model is illustrated in the following figure 4.



```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_4 (Sequential)    (None, 180, 180, 3)       0
_________________________________________________________________
rescaling_6 (Rescaling)      (None, 180, 180, 3)       0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 180, 180, 16)      448
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 90, 90, 16)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 90, 90, 32)        4640
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 45, 45, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 45, 45, 64)        18496
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 22, 22, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 22, 22, 64)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 30976)             0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               3965056
_________________________________________________________________
dense_8 (Dense)              (None, 5)                 645
=================================================================
Total params: 3,989,285
Trainable params: 3,989,285
Non-trainable params: 0
_________________________________________________________________
```

Figure 4: Reconstructed Model Summary.

After applying these two methods, now with 15 epochs the modified ConvNet model shows maximum 80% accuracy during training the model and maximum 74% accuracy during the validation process which is much closer than before. This reduces the overfitting issue and increase the acceptance of this model to do prediction on flower image classification. Following figure 5 shows the plot of accuracy and loss with 15 epochs.

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 03/22/2021

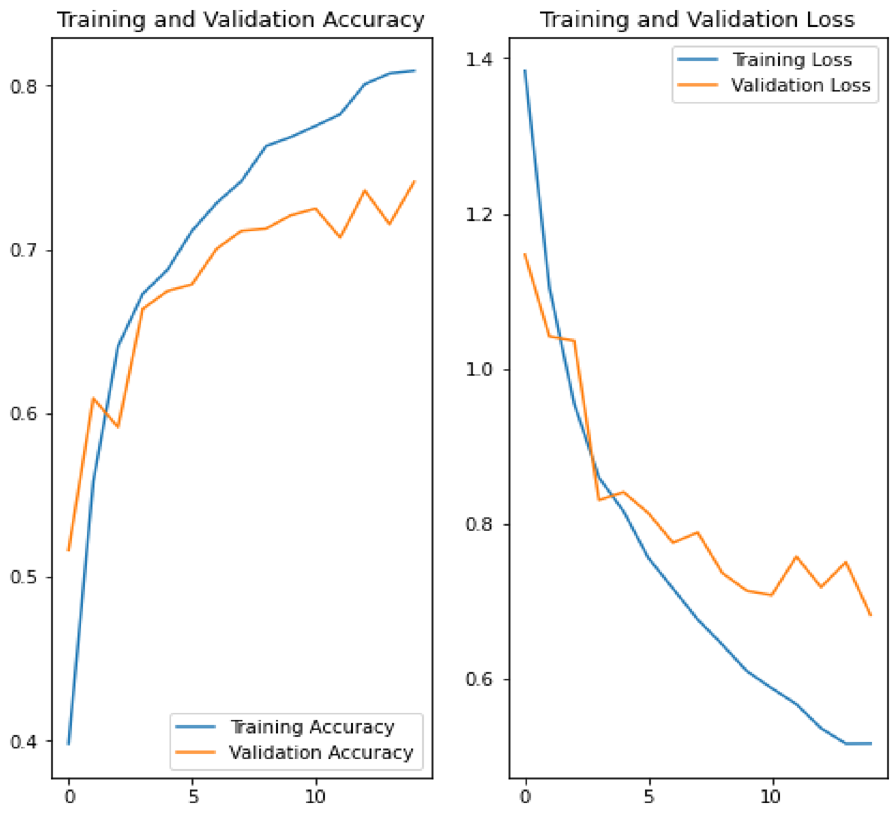


Figure 5: Training and Validation Accuracy(left), Loss(right) after reconstruction of this model

Output (Inference on New Data)

To get prediction on new example image I simply call model.predict(). First, I preprocess the new image data, reshape the image into 180 x 180. In this case, I am using only 1 image to test the prediction (https://storage.googleapis.com/download.tensorflow.org/example_images/592px-Red_sunflower.jpg). So, I set up batch size to 1 by using `tf.expand_dims()` tensorflow framework library function. Data Augmentation and Dropout layers are inactive at inference time. To show the confidence score for this prediction I am using "softmax" function. Final result from this ConvNet deep neural network shows a good prediction result. This can accurately classify the flower image. Following is the result for the above example image.

```
This image most likely belongs to sunflowers with a 99.31 percent confidence.
```

Codes with Comments:

I am giving explanation of each line segment from my implementation of this ConvNet neural network to perform image classification.

Colab link for the code:
https://colab.research.google.com/drive/1fEST91Ta3zJGrBGbkQXekpCnEeikTdhG?usp=sharing

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 03/22/2021

```
"""Image_Classification_CNN_2.ipynb


Automatically generated by Colaboratory.


Original file is located at
    https://colab.research.google.com/drive/1fEST91Ta3zJGrBGbkQXekpCnEeikTdhG
"""


# Import libraries


import matplotlib.pyplot as plt
import numpy as np
import os
import PIL  # Python Imaging Library (PIL)
import tensorflow as tf


from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential


# Download and explore the dataset


import pathlib
dataset_url =
"https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"
data_dir = tf.keras.utils.get_file('flower_photos', origin=dataset_url, untar=True)
data_dir = pathlib.Path(data_dir)


# Count total image
```

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 03/22/2021

```
# All subfolders, then all jpg files

image_count = len(list(data_dir.glob('*/*.jpg')))

print(image_count)


# look into only "roses" subfolder

import glob

roses = list(data_dir.glob('roses/*'))

#roses = data_dir.glob('roses/*')

#PIL.Image.open(str(roses[0]))

PIL.Image.open(roses[0])

#for i in os.listdir(roses):

#  path = roses + "/" + i

#  im=PIL.Image.open(path)

#  plt.imshow(im)

#  plt.show()


tulips = list(data_dir.glob('tulips/*'))

PIL.Image.open(str(tulips[0]))


PIL.Image.open(str(tulips[1]))


# Create a dataset

# Set image size and number of images in each batch


batch_size = 32

img_height = 180

img_width = 180
```

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 03/22/2021

```
# Data preprocessing

# Generates a tf.data.Dataset from image files in a directory by using
tf.keras.preprocessing.image_dataset_from_directory.

# Split the image dataset into 8:2. Use 80% of the images for training, and 20% for validation



train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)



val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)


# check the class names in the class_names attribute on these datasets


class_names = train_ds.class_names
print(class_names)
val_class_names = val_ds.class_names
```

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 03/22/2021

```
print(val_class_names)


# Visualize the data

# Show 9 images from train dataset with image labels.


plt.figure(figsize=(10, 10))

for images, labels in train_ds.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")


# The image_batch is a tensor of the shape (32, 180, 180, 3). This is a batch of 32 images of shape 180x180x3 (the last dimension refers to color channels RGB).

# The label_batch is a tensor of the shape (32,), these are corresponding labels to the 32 images.


#print("Image Sahpe:", train_ds.shape)

for image_batch, labels_batch in train_ds:

  print("Image Shape with Batch:",image_batch.shape)

  print("Label batch Shape:",labels_batch.shape)

  break


# Configure the dataset for performance

# Use buffered prefetching to yield data from disk without having I/O become blocking

# Dataset.cache() keeps the images in memory after they're loaded off disk during the first epoch. This will ensure the dataset does not become a bottleneck while training your model.

# Dataset.prefetch() overlaps data preprocessing and model execution while training.
```

```
AUTOTUNE = tf.data.AUTOTUNE


train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)

val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)


# Standardize the data

# Preprocess images by normalizing the pixel values to be in the [0, 1] range (originally all values are in the [0, 255] range).

# All images will be rescaled by 1./255 (min-max normalization)


normalization_layer = layers.experimental.preprocessing.Rescaling(1./255)


normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))

image_batch, labels_batch = next(iter(normalized_ds))

first_image = image_batch[0]

# Notice the pixels values are now in `[0,1]`.

print(np.min(first_image), np.max(first_image))


# Create the model

# The model consists of three convolution blocks with a max pool layer in each of them. There's a fully connected layer with 128 units on top of it that is activated by a relu activation function.


num_classes = 5


model = Sequential([

  # All images will be rescaled by 1./255 (min-max normalization)

  # Image shape is (180x180x3)

  layers.experimental.preprocessing.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  # First convolution extracts 16 filters that are 3x3
  # Convolution is followed by max-pooling layer with a 2x2 window
  # Relu overcome vanishing gradient prolem, learning faster and better. Relu is default for
Convolutional Neural Network.
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),

  # Second convolution extracts 32 filters that are 3x3
  # Convolution is followed by max-pooling layer with a 2x2 window
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),

  # Third convolution extracts 64 filters that are 3x3
  # Convolution is followed by max-pooling layer with a 2x2 window
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),

  # Flatten feature map to a 1-dim tensor so we can add fully connected layers
  layers.Flatten(),

  # Create a fully connected layer with ReLU activation and 128 hidden units
  layers.Dense(128, activation='relu'),

  # Create output layer with five classifier
  layers.Dense(num_classes)
])

# Compile the model
```

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 03/22/2021

```
# Using the optimizers.Adam optimizer and losses.SparseCategoricalCrossentropy loss function. To view training and validation accuracy for each training epoch, pass the metrics argument.

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

# Model summary

model.summary()

# Train the model

epochs=10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs,
  verbose = 2
)

# Visualize training results
# Create plots of loss and accuracy on the training and validation sets

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

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

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 03/22/2021

```
epochs_range = range(epochs)


plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')


plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()


# print maximun accuracy or traning and validation data
# In the plots above, the training accuracy is increasing linearly over time, whereas validation accuracy stalls around 60% in the training process.
# Also, the difference in accuracy between training and validation accuracy is noticeable—a sign of overfitting.


print("Training Data Accuracy:",max(acc))
print("Validation Data Accuracy",max(val_acc))


# Use data augmentation and add Dropout to your model to overcome "overfitting".
# Data augmentation takes the approach of generating additional training data from your existing examples by augmenting them using random transformations that yield believable-looking images.
```

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 03/22/2021

```
# Implement data augmentation using the layers from tf.keras.layers.experimental.preprocessing. These can be included inside ML model like other layers, and run on the GPU.


data_augmentation = keras.Sequential(
  [
    layers.experimental.preprocessing.RandomFlip("horizontal",
                                                 input_shape=(img_height,
                                                              img_width,
                                                              3)),
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.1),
  ]
)


# Let's visualize what a few augmented examples look like by applying data augmentation to the same image several times


plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")


# Dropout
# When you apply Dropout to a layer it randomly drops out (by setting the activation to zero) a number of output units from the layer during the training process.
# Create a new neural network using layers.Dropout, then train it using augmented images.
```

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 03/22/2021

```
num_classes =5


model = Sequential([
  data_augmentation,
  layers.experimental.preprocessing.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2), # 20% dropout
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])

# Compile and train the model


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


# Model summary


model.summary()
```

```
# Train the model


epochs = 15

history = model.fit(

  train_ds,

  validation_data=val_ds,

  epochs=epochs,

  verbose = 1

)


# After applying data augmentation and Dropout, there is less overfitting than before, and
training and validation accuracy are closer aligned


acc = history.history['accuracy']

val_acc = history.history['val_accuracy']


loss = history.history['loss']

val_loss = history.history['val_loss']


epochs_range = range(epochs)


plt.figure(figsize=(8, 8))

plt.subplot(1, 2, 1)

plt.plot(epochs_range, acc, label='Training Accuracy')

plt.plot(epochs_range, val_acc, label='Validation Accuracy')

plt.legend(loc='lower right')

plt.title('Training and Validation Accuracy')


plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')

plt.plot(epochs_range, val_loss, label='Validation Loss')

plt.legend(loc='upper right')

plt.title('Training and Validation Loss')

plt.show()


import statistics as st

print("Training Data Accuracy:",max(acc))

print("Validation Data Accuracy",max(val_acc))

print("Training Data Accuracy:",st.mean(acc))

print("Validation Data Accuracy",st.mean(val_acc))


# Predict on new data

# Note: Data augmentation and Dropout layers are inactive at inference time.


sunflower_url =
"https://storage.googleapis.com/download.tensorflow.org/example_images/592px-
Red_sunflower.jpg"

sunflower_path = tf.keras.utils.get_file('Red_sunflower', origin=sunflower_url)


img = keras.preprocessing.image.load_img(

    sunflower_path, target_size=(img_height, img_width)

)

img_array = keras.preprocessing.image.img_to_array(img)

img_array = tf.expand_dims(img_array, 0) # Create a batch (of zero)


predictions = model.predict(img_array)

score = tf.nn.softmax(predictions[0])
```

Submitted by: Md. Saifur Rahman
Email: mdsaifur.rahman.1@und.edu
Date: 03/22/2021

```
print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```