Image Classifier using Tensor Flow

S.Manikanta 11812647 B77

Somepalli.manikanta1609@gmail.com

https://github.com/mani1609/INT-248-Proj

Computer Science Engineering- Lovely Professional University

Problem Statement: This project shows how to classify images of flowers. It creates an image classifier using a tf.keras.Sequential model, and loads data using tf.keras.utils.image_dataset_from_directory.

This project follows a basic machine learning workflow:

1. Examine and understand data
2. Build an input pipeline
3. Build the model
4. Train the model
5. Test the model
6. Improve the model and repeat the process

Abstract: In this project image identifications will be done by the help of Advanced CNN (Convolutional Neural Networks with Tensorflow Framework. Here we use Python as a main programming language because Tensorflow is a python library. In this study input data mainly focuses on Plants categories by the help of leaves for identifications. Selecting CNN is the best approach for the training and testing data because it produces promising and continuously improving results on automated plant identifications. Here results are divided in terms of accuracy and time. Using advanced CNN results are above 95% while on others accuracy is below 90% and taking much time than this.Properties like **variance, skewness, curtosis, entropy, class** are used. This project uses a dataset of about 3,700 photos of flowers. The dataset contains five sub-directories, one per class

Keywords: Image Classification, Convolutional Neural Network, Deep Learning, TensorFlow, Relu, epoch.,

1. INTRODUCTION

Recently, image classification is growing and becoming a trend among technology developers especially with the growth of data in different parts of industry such as e-commerce, automotive, healthcare, and gaming. The technology itself almost beats the ability of human in image classification or recognition. One of the dominant approaches for this technology is deep learning. Deep learning falls under the category of Artificial Intelligence where it can act or think like a human. Normally, the system itself will be set with hundreds or maybe thousands of input data in order to make the 'training' session to be more efficient and fast. It starts by giving some sort of 'training' with all the input data. Therefore, image classification is going to be occupied with deep learning system. Machine Vision has its own context when it comes with Image Classification. The ability of this technology is to recognize people, objects, places, action and writing in images. The combination of artificial intelligence software and machine vision technologies can achieve the outstanding result of image classification. The fundamental task of image classification is to make sure all the images are categorized according to its specific sectors or groups. Classification is easy for humans but it has proved to be major problems for machines. It consists of unidentified patterns compared to detecting an object as it should be classified to the proper categories. The various applications such as vehicle navigation, robot navigation and remote sensing by using image classification technology. It is still undergoing challenging work and limited resources are needed to improve it. Image classification has become a major challenge in machine vision and has a long history with it. The project is based on, deep neural network, based on TensorFlow is used with Python as the programming language for image classification. Thousands of images are used as the input data in this project. The accuracy of each percentage of 'train' session will be studied and compared.

2. LITERATURE REVIEW

- Hasbi Ash Shiddieqy, Farkhad Ihsan Hariadi, Trio Adiono "Implementation of Deep-Learning based Image Classification on Single Board Computer", In this project, a deep-learning algorithm based on convolutional neuralnetwork is implemented using python and tflearn for image classification, in which two different structures of CNN are used, namely with two and five layers and It conclude that the CNN with higher layer performs classification process with much higher accuracy.

- Rui Wang, Wei Li, Runnan Qin and JinZhong Wu "Blur Image Classification based on Deep Learning", In this project, a convolution neural network (CNN) of Simplified-Fast-Alexnet (SFA) based on the learning features is proposed for handling the classification issue of defocus blur, Gaussian blur, haze blur and motion blur four blur type images. The experiment results demonstrate that the performance of classification accuracy of SFA, which is 96.99% for simulated blur dataset and 92.75% for natural blur dataset, is equivalent to Alexnet and superior to other classification methods.

- Sameer Khan and Suet-Peng Yong "A Deep Learning Architecture for Classifying Medical Image of Anatomy Object", In this project, a modified CNN architecture that combines multiple convolution and pooling layers for higher level feature learning is proposed. In this, medical image anatomy classification has been carried out and it shows that the proposed CNN feature representation outperforms the three baseline architectures for classifying medical image anatomies.

3. DATASET DESCRIPTION:

The dataset used to train the models is taken from the storage zip file of link- "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz".
Data were extracted from the dataset containing the flower images and their parameters. The dataset used about 3,700 photos of flowers. The dataset contains five sub-directories, one per class

3.1 Load data using a Keras utility

Let's load these images off disk using the helpful `tf.keras.utils.image_dataset_from_directory` utility. This will take you from a directory of images on disk to a `tf.data.Dataset` in just a couple lines of code. If you like, you can also write your own data loading code from scratch by visiting the Load and preprocess images project.

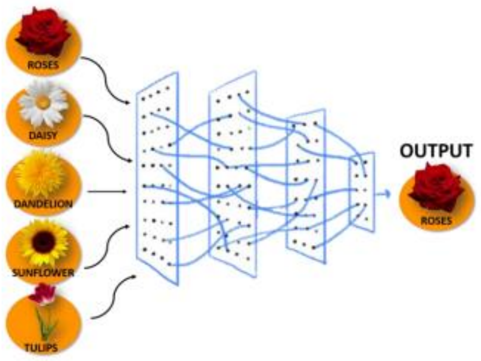
No.	Type of Flowers	No of Images
1.	Daisy	633
2.	Dandelion	898
3.	Roses	641
4.	Sunflower	699
5.	Tulips	799
Total of flower images		3670

3.2 Create a dataset

Define some parameters for the loader: batch_size = 32
img_height = 180
img_width = 180

It's good practice to use a validation split when developing your model. Let's use 80% of the images for training, and 20% for validation. We can find the class names in the `class_names` attribute on these datasets. These correspond to the directory names in alphabetical order.

4. PROPOSED ARCHITECTURE



4.1 Visualize the data

Should train a model using these datasets by passing them to `Model.fit` in a moment. If you like, you can also manually iterate over the dataset and retrieve batches of images. 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. You can call `.numpy()` on the `image_batch` and `labels_batch` tensors to convert them to a `numpy.ndarray`.

4.2 Configure the dataset for performance

Let's make sure to use buffered prefetching so you can yield data from disk without having I/O become blocking. These are two important methods you should use when loading data:

- 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. If your dataset is too large to fit into memory, you can also use this method to create a performant on-disk cache.
- Dataset.prefetch overlaps data preprocessing and model execution while training.

Interested readers can learn more about both methods, as well as how to cache data to disk in the Prefetching section of the Better performance with the tf.data API guide.

4.3 Standardize the data

The RGB channel values are in the `[0, 255]` range. This is not ideal for a neural network; in general you should seek to make your input values small.

Here, you will standardize values to be in the `[0, 1]` range by using `tf.keras.layers.Rescaling`

4.4 Create the model

The Sequential model consists of three convolution blocks (`tf.keras.layers.Conv2D`) with a max pooling layer (`tf.keras.layers.MaxPooling2D`) in each of them. There's a fully-connected layer (`tf.keras.layers.Dense`) with 128 units on top of it that is activated by a ReLU activation function (`'relu'`).

```
num_classes = 5

model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  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.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

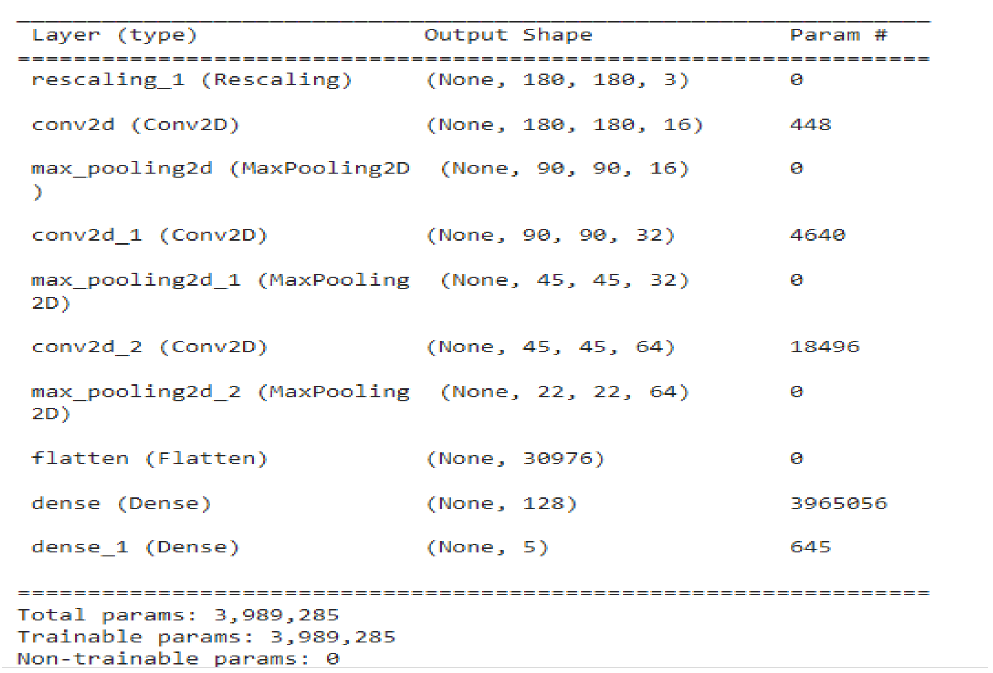
This model has not been tuned for high accuracy—the goal of this project is to show a standard approach.

4.5 Compile the model

For this project, choose the `tf.keras.optimizers.Adam` optimizer and `tf.keras.losses.SparseCategoricalCrossentropy` loss function. To view training and validation accuracy for each training epoch, pass the `metrics` argument to `Model.compile`.

4.6 Model summary

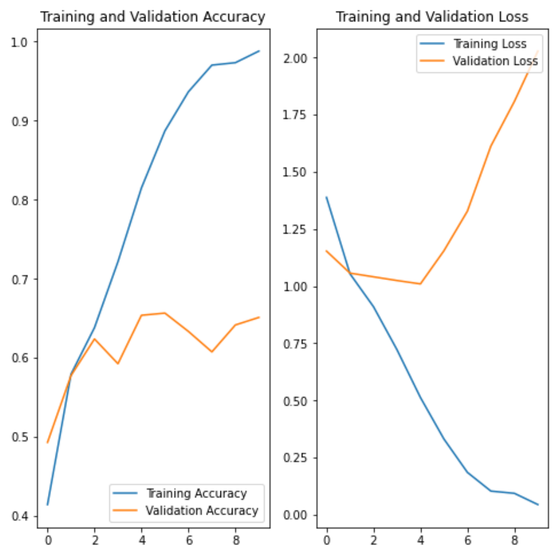
View all the layers of the network using the model's `Model.summary` method.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 180, 180, 16)      448

 max_pooling2d (MaxPooling2D  (None, 90, 90, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 45, 45, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 22, 22, 64)       0
 2D)

 flatten (Flatten)           (None, 30976)             0

 dense (Dense)               (None, 128)               3965056

 dense_1 (Dense)             (None, 5)                 645

=================================================================
Total params: 3,989,285
Trainable params: 3,989,285
Non-trainable params: 0
```

5. RESULTS & SCREENSHOTS

After creating the model and training it the results are as follows.



The plots show that training accuracy and validation accuracy are off by large margins, and the model has achieved only around 60% accuracy on the validation set.

Let's inspect what went wrong and try to increase the overall performance of the model.

Overfitting

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.

When there are a small number of training examples, the model sometimes learns from noises or unwanted details from training examples—to an extent that it negatively impacts the performance of the model on new examples. This phenomenon is known as overfitting. It means that the model will have a difficult time generalizing on a new dataset.

There are multiple ways to fight overfitting in the training process. In this project, you'll use data augmentation and add Dropout to your model.

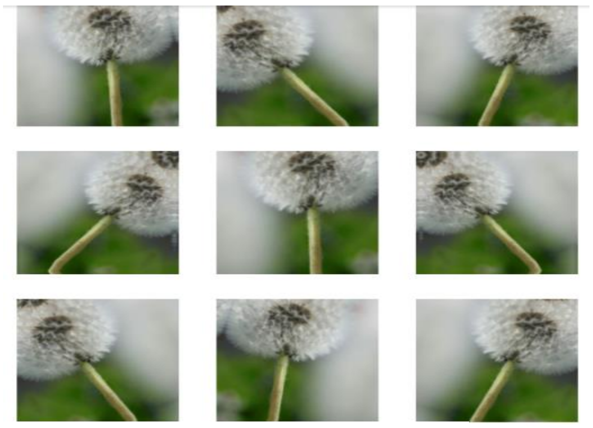
Data augmentation

Overfitting generally occurs when there are a small number of training examples. 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. This helps expose the model to more aspects of the data and generalize better.

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

You will implement data augmentation using the following Keras preprocessing layers: `tf.keras.layers.RandomFlip`, `tf.keras.layers.RandomRotation`, and `tf.keras.layers.RandomZoom`. These can be included inside your model like other layers, and run on the GPU.

```
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

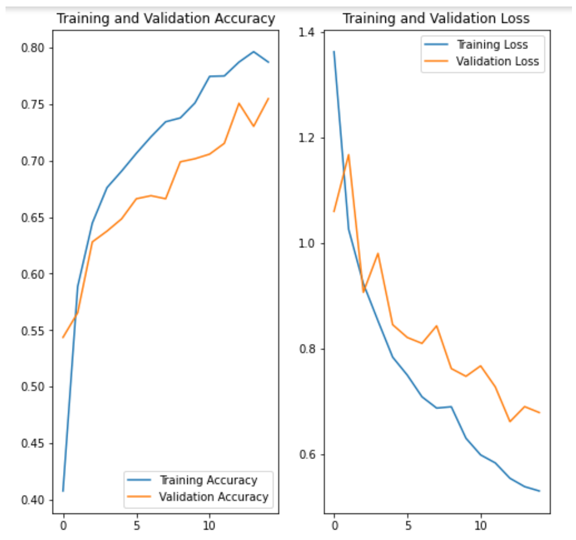
Another technique to reduce overfitting is to introduce dropout regularization to the network.

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. Dropout takes a fractional number as its input value, in the form such as 0.1, 0.2, 0.4, etc. This means dropping out 10%, 20% or 40% of the output units randomly from the applied layer.

Let's create a new neural network with `tf.keras.layers.Dropout` before training it using the augmented images:

```
model = Sequential([
  data_augmentation,
  layers.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),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

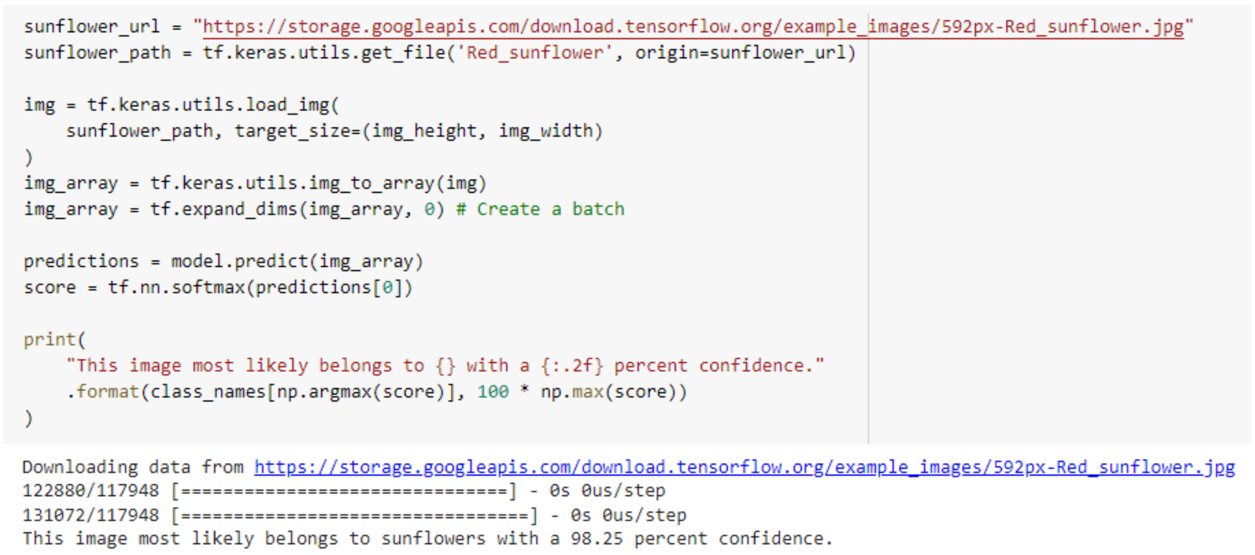
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_1 (Sequential)   (None, 180, 180, 3)       0

 rescaling_2 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_3 (Conv2D)           (None, 180, 180, 16)      448

 max_pooling2d_3 (MaxPooling  (None, 90, 90, 16)       0
 2D)

 conv2d_4 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_4 (MaxPooling  (None, 45, 45, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 22, 22, 64)       0
 2D)

 dropout (Dropout)           (None, 22, 22, 64)        0

 flatten_1 (Flatten)         (None, 30976)             0

 dense_2 (Dense)             (None, 128)               3965056

 dense_3 (Dense)             (None, 5)                 645

=================================================================
Total params: 3,989,285
Trainable params: 3,989,285
Non-trainable params: 0
_________________________________________________________________
```



Predict on new data

Finally, let's use our model to classify an image that wasn't included in the training or validation sets.

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 = tf.keras.utils.load_img(
    sunflower_path, target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

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

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/example_images/592px-Red_sunflower.jpg
122880/117948 [===============================] - 0s 0us/step
131072/117948 [=================================] - 0s 0us/step
This image most likely belongs to sunflowers with a 98.25 percent confidence.
```

From the result by classifying the image we got that it is most likely to be sunflowers with a 98.25 percent confidence

->Roses- 98.585%, Daises-99.626%, Dandelion-99.825% etc.

6. CONCLUSION

In conclusion, this project is about image classification by using deep learning via framework TensorFlow. It has three (3) objectives that have achieved throughout this project. The objectives are linked directly with conclusions because it can determine whether all objectives are successfully achieved or not. It can be concluded that all results that have been obtained, showed quite impressive outcomes. The deep neural network (DNN) becomes the main agenda for this project, especially in image classification technology. DNN technique was studied in more details starting from assembling, training model and to classify images into categories. The roles of epochs in DNN was able to control accuracy and also prevent any problems such as overfitting. Implementation of deep learning by using framework TensorFlow also gave good results as it is able to simulate, train and classified with up to 90% percent of accuracy towards five (5) different types of flowers that have become a trained model. Lastly, Python have been used as the programming language throughout this project since it comes together with framework TensorFlow which leads to designing of the system involved Python from start until ends.

The result of this project depends on the objectives that need to be achieved. Other than that, certain parameters also played its roles to determine the accuracy of the image classification by using the deep neural network (DNN). The first result of this project was tested by conducting classification for each of the types of flowers. It can be seen all of the five (5) types of flowers showed up to 90% accuracy in terms of implementation of the system of image classification by using DNN. This happened due to the abundantly set of data that was being used in order to train the model and of course DNN worked excellent when there were lots of data. Then, the system was tested with an image of cow.jpg where actually this images were not included during the training model. The images also were not one of the categories of flowers instead it fell under animals. There were some errors after doing the classification. The errors state that 'Not Found Error' which meant that the images cannot be recognized by the systems as it was not trained so that model trained can recognize it as an animal named as cow.

6.1 FUTURE SCOPE

While imaging was associated with security functions and surveillance missions, the term has grown to represent something larger in recent years. Thanks to advancements in science and technology, image processing is now an integral part of AI systems.

Various new types of processing systems, which have come up recently help with chemical, thermal and molecular imaging. Furthermore, the use of such systems has led to tremendous growth in the field of space exploration. Most new satellites make use of different sensors to obtain useful information from outer space. Satellite imaging and military applications are regarded as future trends in the field of image processing. Furthermore, advances in broadband devices and mobile technology will help in the improvement of image processing systems in hand-held devices. To put things in perspective, the future of image processing looks bright and solid

7. REFERENCES

[1] Rui Wang, Wei Li, Runnan Qin and JinZhong Wu "Blur Image Classification based on Deep Learning", IEEE, ISBN 978-1-5386-1621-5 pp. 1-6, 2017

[2] Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., & Wierstra, D. (2015). DRAW: A Recurrent Neural Network For Image Generation. https://doi.org/10.1038/nature14236

[3] Chauhan, K., & Ram, S. (2018). International Journal of Advance Engineering and Project Image Classification with Deep Learning and Comparison between Different Convolutional Neural Network Structures using Tensorflow and Keras, 533–538.