IMAGE CLASSIFICATION

Final Report of Flower Image Classification Using CNN Model

Mansi Rajapkar
CODERSDAILY

Introduction:

This Project include building Convolutional Neural Network (CNN), which identifies different types of flowers such as Daisy, Dandelions, Tulips, Roses, and Sunflowers . It demonstrate the data preprocessing, build CNN Model, add data augmentation & dropout, Compile, train and evaluate model, plot accuracy/loss curves and at the end save model in .keras format.

Objective :

The objective of this project is to build a Convolutional Neural Network (CNN) model that accurately identifies different types of flowers just by analyzing their images. Flowers such as Tulips, Dandelions, Roses, Daisy, and Sunflowers. This project demonstrate how to:

- Load and preprocess real world image data set.
- Splitting data set into two parts : training and validation.
- Autotune the data that tells it figure out the optimal performance setting when loading data.
- Also used 'Relu'-(Rectified Linear Unit) activation function while creating a model.
- Compile and Train the data .
- Normalization code of block that scales the pixel values of image from (0-255) to (0-1).
- Train model with 10 times of epochs for training dataset and train model with 15 times of epochs for data augmentation and dropout.
- Save the Model.

Dataset Overview :

The dataset we use in our project is TensorFlow Flower Photos Dataset, which is publicly available and commonly used for image classification task

SOURCE:

TensorFlow Example Images - Flower Photos

DESCRIPTION :

- Daisies – White Petaled Flowers
- Dandelions – Yellow Puffball Flowers
- Roses – Red and Pink Roses
- Sunflowers – Large Yellow Flowers
- Tulips – Colorful Bell Shaped

Tools Libraries & Used :

- Numpy Library – for numerical calculation in image pixels.
- Pillow Library – to read and show image.
- Matplotlib Library – for displaying and visualizing graphs.
- TensorFlow Library – for deep learning.
- Pathlib Library – to work with the path of dataset downloaded.
- Keras Module – of tensorflow library .
- Sequential Module – for building model .
- Layers – to create a model with some major functionality.

Platform Used – Visual Studio Code

Data Preprocessing :

1. **Loading the dataset** :
   The dataset was loaded directly from the folder structure using:
   tf.keras.utils.image_dataset_from_directory() .

- Images were automatically labelled based on the folder names
- The dataset were split into two parts:
  80% Training
  20% Validation

2. **Resizing Images**:
   All Images were resized to fixed size of 180 X 180 pixel to ensure consistency.
   Image_size=180X180

3. **Batching:**
   Images were grouped into batches of 32 , which allows efficient model usage during training
   Batch_size=32

4. **Shuffling :**
   The training dataset were shuffled before giving dataset to model to randomize the order of images and improve generalization .
   Train_ds=train_ds.shuffle(1000)

5. **Catch and Prefetching**:
   To improve performance and reduce disk I/O during training:
   train_ds =
   train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
   val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

6. **Rescaling and Normalizing:**
   Rescaling the images because neural network train data better when value are in smaller range like (0-1).
   normalization_layer = layers.Rescaling(1./255)
   It normalizing the pixels of images from (0-255) to (0,1) by which neural network train faster and results would be more accurate and stable
   normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))

7. **Data Augmentaion:**
   Means creating a variation of your training images on the fly to make the model generalize better.
   data_augmentation = keras.Sequential([….])
   This helps reduce overfitting and improves the model's ability to generalize.

Model Architecture :

Our model have some specific layers which are like a building blocks of model . In this we gives the data as an input, it needs to go through from layers by which we can train our model . The layers in a model extract features, reduce size, scales the values . All the task done by each layer are briefly described below :

1 layers.Rescaling(…) -   Normalizes pixel value from (0-255) to (0-1).

2 layers.Conv2D (…) – Detects features, edges, colors, textures .

3 layers.MaxPooling 2D(…) – That shrinks the size of image while keeping the most important information.

#Now there are some repeating blocks or layers, this is because each layer learns more complex features-

layers.conv2D(16) – learns simple features such as edges and corners.

4 Layers.conv2D(32) – learns more detailed features such as textures and colour blob.

5 Layers.conv2D(64) – complex shapes such as petals, stems, or whole flower.

6 Layers.flatten() – Takes the output of the last maxpooling layer(which 3D) and converts it into 1D vector like a list of no. because 2D pooled feature map always takes fully connected network .

7 layers.dense() – This is fully connected layer with 128 neurons , it takes flatten features and learn pattern from them

It also Applies 'RelU' activation function to bring non-linearity.

8 layers.dense(num_classes) – This is the final output layer of model

So each Convolutional layer is build on top of the previous one .

Visual Representation of model preparation :

```
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)
])
```

Analogy:

Imagine understanding flower:

1 First see the lines & colours (conv2D 16)

2 Then you detect petals & circles (conv2D 32)

3 Finally your brain sees the whole flower shape (conv2D 64)

Model Compilation & Training :

It prepares model for training by setting:

- How the model learns(optimizer)
- How the error is calculated (loss)
- What to monitor during training (metrics)

Optimizer=adam' : optimizer controls how the model updates weights during training .

'adam' : is short for adaptive moment

It trains the model that if the answer is wrong, do right it or if it is right it will remain same.

Loss () : It tells the model how wrong it is during training .

SparseCategorialCrossentropy() : is used when :

- You have multi-class classification.
- Your labels are integer encoded (0,1,2).

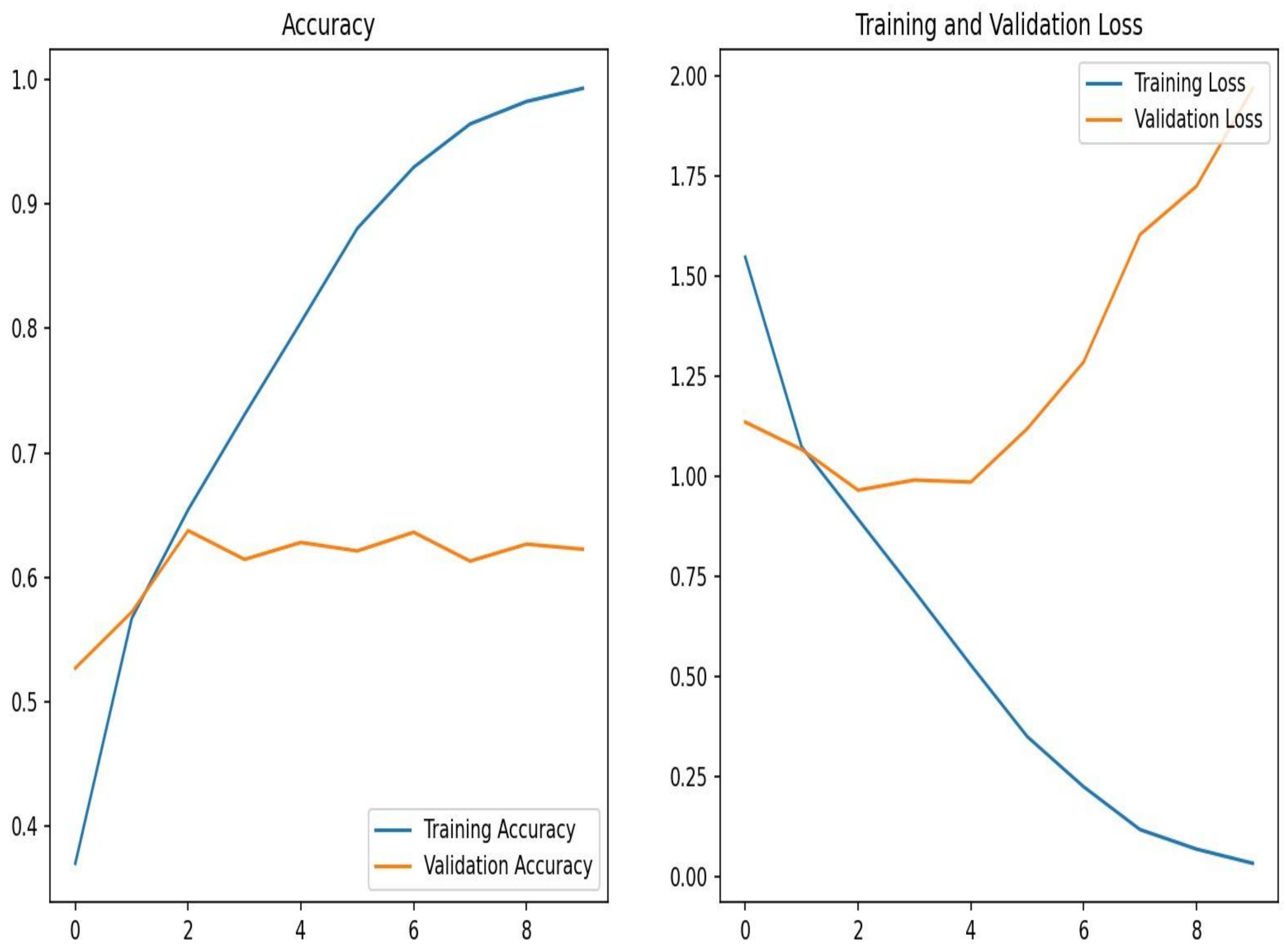
(from_logits) = True : means the model's final layer gives raw output (logits), not softmax probability.

Metrics=['accuracy'] : This tells the model to monitor accuracy during the training and validation

- You are telling the model to go through the entire training data 10 times or 15 times
- 1 epoch = 1 complete pass through the training dataset.
- More epochs = more learning .

Evaluation & Result :

We have checked Training & validation accuracy level and Training and Validation Loss level and plotted in a graph as below :



Data Augmentation :

Purpose: Data Augmentation means creating variation of your images on the fly to make the model generalize better .

Working on data augmentation make your model robust.

Techniques Applied:

- Layers.RandomFlip() : this function flips the left to right .
  "horizontal" parameter in t he function helps model to understand that orientation can vary.
  "input_shape(180,180,3)" this tells the layer the size of input images.

```
• layers.RandomFlip("horizontal",
•                             input_shape=(img_height,
•                                          img_width,
•                                          3))
```

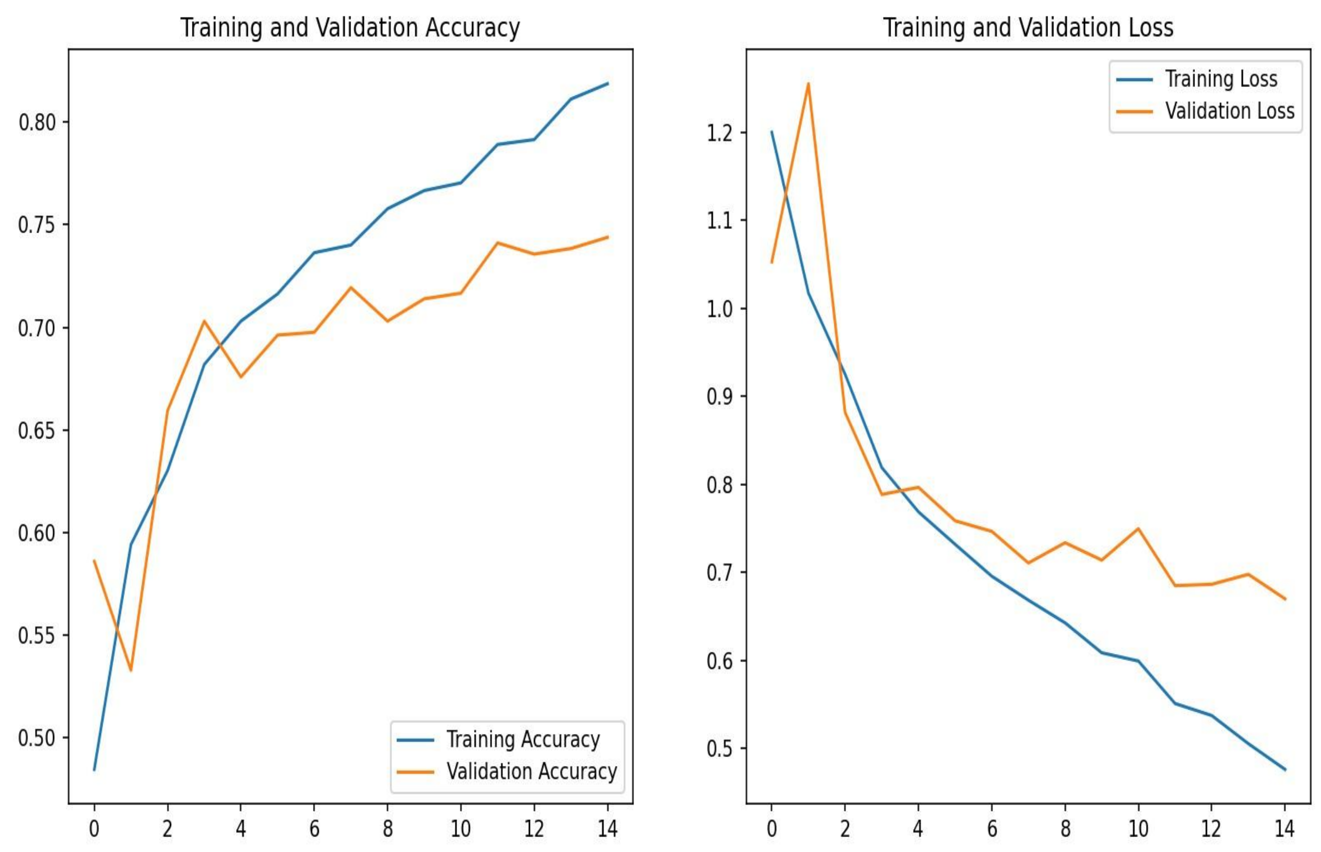
- Layers.RandomRotation() : it rotates the image slightly (10% = ~ 36')
  Helps model to understand tilt images

```
layers.RandomRotation(0.1),
```

- Layers.RamdomZoom() : randomly zooms in or out (by 10%), useful when objects appear at different distances.

```
• layers.RandomZoom(0.1),
```

Graphical Representation of Training & Validation Accuracy and Training & Validation Loss after Data Augmentation.



Model Saving:

Saves your trained model to a file named flower_model.keras

We can reload this model later without retraining .

What we have done:

- At first we compile the model by we get optimizer and loss function,
- then we train the mode 15x times (epochs) ,
- created accuracy and loss graph for training and validation dataset.
- And the end we finally save the model.

```
model.save('flower_model.keras') #save the model
```

Conclusion :

In this project the Convolutional Neural Network (CNN) was developed using tensorflow and keras to classify images of flowers into five categories : sunflower, daisy, dandelions, roses, and tulips.

The dataset was preprocessed using normalization, batching , caching and data augmented techniques to improve model performance and generalization . The model was trained and evaluated using training and validation dataset , and performance was visualized using accuracy & loss graph.

The final trained model demonstrated good accuracy and learning behavior, showing the effectiveness of CNNs in solving real-world image classification problems.

Future Work :

Although the current model performance well, several enhancement can be made:

- **Deploy the model** using a web app interface (e.g., Streamlit or Flask).
- **Add a testing dataset** to evaluate the model's performance on completely unseen data.
- **Increase dataset size** or add more classes for advanced classification tasks.
- **Experiment with hyperparameters** like learning rate, optimizer, batch size, and dropout rate to improve results.

References :

- Kears Documentation : https://keras.io/
- Tensorflow Documentation : https://www.tensorflow.org/