Flower Classification Using CNN

Deep learning, has shown a significant performance and accuracy improvement in the field of computer vision. A breakthrough in building image-based models came with the discovery of convolutional neural network or CNN for short. They introduced the concept of end-to-end learning. There's no longer a need to define the features and do feature engineering. The network does that for you. A CNN takes just the image's raw pixel data as input and learns how to extract these features, and ultimately infer what the image constitutes.

In this article, we will be building a CNN based image classification model and use it to categorize images of flowers!!!

For an in-depth understanding of Convolutional neural network, check out these excellent articles: -

1. *Link to some analyticsindiamag article*

2. *Link to some analyticsindiamag article*

We will follow the basic machine learning workflow for this task.

1. Load and Explore data
2. Build the model
3. Train the model
4. Test the model

We will use Keras, a high-level API, to build and train models in TensorFlow.

Import dependencies

```
import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf
from keras import layers
from keras.models import Sequential
```

Loading Dataset

The dataset which we will be using can be found here:
https://www.kaggle.com/ayaanmustafa/flower

This dataset has images of three kinds of flowers: - Lotus, rose, and sunflower.

Images for testing and validation are available to us in separate folders. Also, the photos are labeled by their filename in which they are present.

We will be using **image_dataset_from_directory()** utility available in Keras for loading images in batches.

as the images are of different size, let's convert them to a fixed size of **200 x 200 x 3** .

```
train_data =tf.keras.preprocessing.image_dataset_from_directory(
  'training/',
  seed=42,
  shuffle=True,
  image_size=(200, 200),
  batch_size=16)
```

```
Found 271 files belonging to 3 classes.
```

```
val_data =tf.keras.preprocessing.image_dataset_from_directory(
  'validation/',
  seed=42,
  shuffle=True,
  image_size=(200, 200),
  batch_size=16)
```

```
Found 59 files belonging to 3 classes.
```

```
class_names = train_data.class_names
print(class_names)
```

```
['lotus', 'rose', 'sunflower']
```

Let's visualize a few images and see exactly what we are dealing with…

```
plt.figure(figsize=(10, 10))
for images, labels in train_data.take(1):
  for i in range(4):
    ax = plt.subplot(2, 2, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

Building Model

Let's build a Simple CNN model for our image classification task. It will consist of two convolutional blocks with a maxpool layer in each of them which will work as a feature extractor, followed by a couple of fully connected layers which will work as our classifier.

Note that the input image shape needs to be specified beforehand.

```
model = Sequential([
  layers.Conv2D(32, 3, padding='same', activation='relu',input_shape=(200,200, 3)),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(3, activation='softmax')
])
```

We will compile our model with **adam** optimizer and choose **SparseCategoricalCrossentropy** as our loss function; this is because the problem is of multiclass classification and our target variable is not one-hot encoded.

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

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 200, 200, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 100, 100, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 100, 100, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 50, 50, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 160000)            0
_________________________________________________________________
dense (Dense)                (None, 128)               20480128
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 387
=================================================================
Total params: 20,499,907
Trainable params: 20,499,907
Non-trainable params: 0
_________________________________________________________________
```

Notice that majority of the parameter in our model belongs to the fully connected layers i.e. our classifier.

Training

```
model.fit(train_data,validation_data=val_data,epochs=3)
```

```
Epoch 1/3
17/17 [==============================] - 18s 1s/step - loss: 1330.5374 - accuracy: 0.5203 - val_loss: 24.8727 - val_accuracy: 0.661
Epoch 2/3
17/17 [==============================] - 18s 1s/step - loss: 3.9905 - accuracy: 0.8266 - val_loss: 2.2927 - val_accuracy: 0.8983
Epoch 3/3
17/17 [==============================] - 18s 1s/step - loss: 0.3956 - accuracy: 0.9373 - val_loss: 0.5464 - val_accuracy: 0.9153
<tensorflow.python.keras.callbacks.History at 0x7f84e07be0b8>
```

With only three epochs, we are able to achieve a training accuracy of 0.93 and a validation accuracy of 0.91.

As the training and validation accuracy are very close, our model is not overfitting.

Testing

Let's test our model on a new image which was not included in the training and validation set and see how it is performing.

```
test_img_path='image.jpg'

img = tf.keras.preprocessing.image.load_img(test_img_path, target_size=(200, 200))

plt.imshow(img)
plt.axis("off")

img_array = tf.keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)
pred = model.predict(img_array)
print(class_names[np.argmax(pred)]," ", 100 * np.max(pred)," % sure")
```

```
lotus   87.43373155593872  % sure
```

Summary: -

In this article, we discovered how to build a convolutional neural network-based model to classify images of flowers.

Specifically, we learned:

1. How to load and process images for modelling.
2. How to develop and train a convolutional neural network for classification tasks.

Things we can do to improve the performance of our CNN model:-

1.Increase size of Dataset

2.Data Augmentation

3. Transfer Learning

4.Hyperparameter optimization