Image Classification using CNN

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

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

Downloading Data Set

```
import pathlib

dataset_url =
"https://storage.googleapis.com/download.tensorflow.org/example_images/flo
wer_photos.tgz"
data_dir = tf.keras.utils.get_file('flower_photos.tar',
origin=dataset_url, extract=True)
data_dir = pathlib.Path(data_dir).with_suffix('')
```

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

```
roses = list(data_dir.glob('roses/*'))
PIL.Image.open(str(roses[0]))
```

Preparing dataset

```
batch_size = 32
img_height = 180
img_width = 180
```

Training Data set

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

```
  batch_size=batch_size)
```

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

Validation Dataset

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

Displaying some images

```
import matplotlib.pyplot as plt

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")
```

Displaying Batch size

```
for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

```
normalization_layer = layers.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 pixel values are now in `[0,1]`.
```

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

Building CNN Model

```
num_classes = len(class_names)

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)
])
```

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

```
epochs=10
myCNN_Model = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

Model Evaluation

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

loss = myCNN_Model.history['loss']
val_loss = myCNN_Model.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()
```

Overfit? How to Overcome: Data Augmentation and Dropout layer

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

```
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")
```

New Model

```
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, name="outputs")
])
```

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

```
epochs = 15
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

Model2 Evaluation:

```
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()
```

```
epochs = 15
myCNN_model2 = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
acc = myCNN_model2.history['accuracy']
val_acc = myCNN_model2.history['val_accuracy']

loss = myCNN_model2.history['loss']
val_loss = myCNN_model2.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()
```

Testin with a unknown image:

```
sunflower_url =
"https://storage.googleapis.com/download.tensorflow.org/example_images/592
px-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])
```

Class Predicted by our model

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