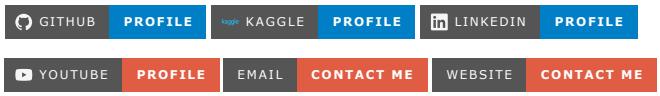
Image Classification with TensorFlow

Author: Irfan Ullah Khan



This project involves building and training an image classification model using TensorFlow and Keras. You can use the CIFAR-10 dataset for this purpose.

```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
```

```
# Load and preprocess the dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
y_train, y_test = to_categorical(y_train), to_categorical(y_test)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
```

```
# Build the model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# Compile and train the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))
```

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 86s 54ms/step - accuracy: 0.3274 - loss: 1.8075 - val_accuracy: 0.5397 - val_loss: 1.2907
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 143s 55ms/step - accuracy: 0.5588 - loss: 1.2413 - val_accuracy: 0.6093 - val_loss: 1.0961
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 138s 52ms/step - accuracy: 0.6245 - loss: 1.0614 - val_accuracy: 0.6512 - val_loss: 0.9898
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 83s 53ms/step - accuracy: 0.6656 - loss: 0.9531 - val_accuracy: 0.6398 - val_loss: 1.0430
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 141s 52ms/step - accuracy: 0.6952 - loss: 0.8671 - val_accuracy: 0.6821 - val_loss: 0.9162
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 52ms/step - accuracy: 0.7131 - loss: 0.8139 - val_accuracy: 0.6903 - val_loss: 0.9069
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 52ms/step - accuracy: 0.7351 - loss: 0.7589 - val_accuracy: 0.6981 - val_loss: 0.8918
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 81s 52ms/step - accuracy: 0.7504 - loss: 0.7155 - val_accuracy: 0.6978 - val_loss: 0.8894
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 83s 53ms/step - accuracy: 0.7644 - loss: 0.6767 - val_accuracy: 0.7019 - val_loss: 0.8792
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 144s 54ms/step - accuracy: 0.7751 - loss: 0.6379 - val_accuracy: 0.6921 - val_loss: 0.9046
```

```
# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```