UVG - RIA - SHAP Visualization Lab with TensorFlow

Autor: Luis Pedro Gonzalez Aldana

Basado en https://jmerida30.github.io/OBJRECLAB/

Github

https://github.com/LPELCRACK896/RIA_LAB2.git

Python

Utilizando `Python 3.12.5` en Windows.

In [ ]:
```
from tensorflow.keras import layers, models
from ipywidgets import interact, IntSlider
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
import shap
import os

print("TensorFlow version:", tf.__version__)
print("GPU is", "available" if tf.config.list_physical_devices('GPU') else "NOT AVAILABLE")
```

```
TensorFlow version: 2.17.0
GPU is NOT AVAILABLE
```

Lo del GPU no sorprende porque solo lo tengo integrado.

In [ ]:
```
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.cifar10.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 15s 0us/step
```

In [ ]:
```
def create_model():
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```

In [ ]:
```
model_path = 'cifar10_cnn.keras'
if os.path.exists(model_path):
    print("Loading pre-trained model...")
    model = tf.keras.models.load_model(model_path)
else:
    print("No pre-trained model found. Training a new model...")
    model = create_model()
    history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))
    model.save(model_path)
    print(f"Model saved to {model_path}")
```

```
No pre-trained model found. Training a new model...
c:\Users\Luis P\Documents\0lps\01 U\0LaU\00000Decimo semestre (y el ultimo)\02 Responsive AI\Labs\Lab2\penv\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 18s 11ms/step - accuracy: 0.3382 - loss: 1.7785 - val_accuracy: 0.5542 - val_loss: 1.2528
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 15s 10ms/step - accuracy: 0.5675 - loss: 1.2161 - val_accuracy: 0.6264 - val_loss: 1.0695
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 15s 9ms/step - accuracy: 0.6345 - loss: 1.0398 - val_accuracy: 0.6494 - val_loss: 1.0114
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 15s 9ms/step - accuracy: 0.6739 - loss: 0.9331 - val_accuracy: 0.6598 - val_loss: 0.9644
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 15s 9ms/step - accuracy: 0.7009 - loss: 0.8515 - val_accuracy: 0.6888 - val_loss: 0.8972
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 15s 10ms/step - accuracy: 0.7238 - loss: 0.7831 - val_accuracy: 0.6921 - val_loss: 0.8850
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 15s 10ms/step - accuracy: 0.7464 - loss: 0.7219 - val_accuracy: 0.6981 - val_loss: 0.8876
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 17s 11ms/step - accuracy: 0.7641 - loss: 0.6752 - val_accuracy: 0.7083 - val_loss: 0.8580
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 16s 10ms/step - accuracy: 0.7769 - loss: 0.6320 - val_accuracy: 0.6938 - val_loss: 0.9006
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 16s 10ms/step - accuracy: 0.7906 - loss: 0.5942 - val_accuracy: 0.7099 - val_loss: 0.8911
Model saved to cifar10_cnn.keras
```

In [ ]:
```
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f"Test accuracy: {test_acc:.2f}")
```

```
313/313 - 1s - 5ms/step - accuracy: 0.7099 - loss: 0.8911
Test accuracy: 0.71
Test accuracy: 0.71
```

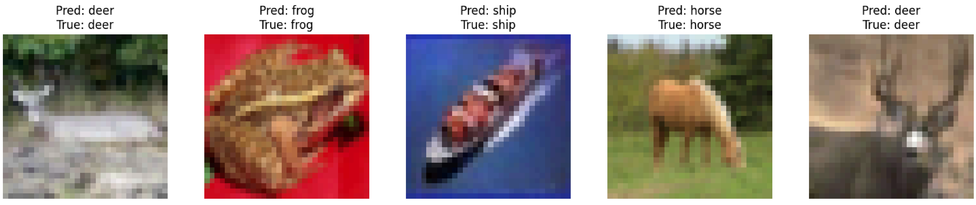
In [ ]:
```
def test_model(model, test_images, test_labels, num_samples=5):
    indices = np.random.choice(test_images.shape[0], num_samples, replace=False)
    sample_images = test_images[indices]
    sample_labels = test_labels[indices]

    predictions = model.predict(sample_images)

    fig, axes = plt.subplots(1, num_samples, figsize=(15, 3))
    for i, ax in enumerate(axes):
        ax.imshow(sample_images[i])
        predicted_class = class_names[np.argmax(predictions[i])]
        true_class = class_names[sample_labels[i][0]]
        ax.set_title(f"Pred: {predicted_class}\nTrue: {true_class}")
        ax.axis('off')
    plt.tight_layout()
    plt.show()
```

In [ ]:
```
# Pues esto lo agregué para ver
test_model(model, test_images, test_labels, num_samples=5)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
```



In [ ]:
```
num_background = 100
background_images = test_images[:num_background]
explainer = shap.GradientExplainer(model, background_images)
```

In [ ]:
```
def shap_visualization(image_index):
    image = test_images[image_index:image_index+1]
    true_label = test_labels[image_index][0]

    # Generate and process SHAP values
    shap_values = explainer.shap_values(image)
    prediction = model.predict(image)
    predicted_class = np.argmax(prediction)
    shap_values_for_class = shap_values[0, ..., predicted_class]
    shap_sum = np.sum(shap_values_for_class, axis=-1)

    # Normalize SHAP values for scatter plot
    shap_normalized = (shap_sum - shap_sum.min()) / (shap_sum.max() - shap_sum.min())

    # Create figure with subplots
    fig, axs = plt.subplots(1, 2, figsize=(12, 6))

    # Original Image
    axs[0].imshow(image[0])
    axs[0].set_title("Original Image\nTrue: " + class_names[true_label])
    axs[0].axis('off')

    # Scatter Plot with Stars on Image
    y, x = np.indices(shap_sum.shape)
    colors = shap_sum.flatten()  # Color by SHAP values
    sizes = 100 * shap_normalized.flatten() + 10  # Size of stars
    axs[1].imshow(image[0], aspect='auto')  # Display the original image as background
    scatter = axs[1].scatter(x.flatten(), y.flatten(), c=colors, s=sizes, cmap='coolwarm', marker='o', alpha=0.6)
    axs[1].set_title("SHAP Scatter on Image\nPredicted: " + class_names[predicted_class])
    axs[1].axis('off')
    fig.colorbar(scatter, ax=axs[1], orientation='vertical', fraction=0.046, pad=0.04)

    plt.tight_layout()
    plt.show()
```

In [ ]:
```
shap_visualization(10)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
```