Setup

```
import itertools
import os
import matplotlib.pylab as plt
import numpy as np
import tensorflow as tf
import tensorflow_hub as hub

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

```
TF version: 2.15.0
Hub version: 0.16.1
GPU is available
```

Set up dataset

Inputs are suitably resized for the selected module. Dataset augmentation (i.e., random distortions of an image each time it is read) improves training, esp. when fine-tuning.

```
#mounting google drive
from google.colab import drive
import shutil
import glob
import re
import random

drive.mount("/content/drive", force_remount=True)
src_dir = '/content/drive/MyDrive/AI4ALL/fire_dataset'
```

```
Mounted at /content/drive
```

```
from sklearn.model_selection import train_test_split
import os
import shutil
import random
from google.colab import drive


# Define the directory paths
fire_images_dir = '/content/drive/MyDrive/AI4ALL/fire_dataset/fire_images'
non_fire_images_dir = '/content/drive/MyDrive/AI4ALL/fire_dataset/non_fire_images'


# List files in the fire_images directory
fire_images_files = os.listdir(fire_images_dir)


# List files in the non_fire_images directory
non_fire_images_files = os.listdir(non_fire_images_dir)

# Print the number of files in each directory
print(f"Number of files in fire_images directory: {len(fire_images_files)}")
print(f"Number of files in non_fire_images directory: {len(non_fire_images_files)}")
print()

# Remove the ".png" extension from image filenames
fire_images_files = [file.rsplit('.', 1)[0] for file in fire_images_files]
non_fire_images_files = [file.rsplit('.', 1)[0] for file in non_fire_images_files]

# Define labels for fire and non-fire images
fire_labels = [file.rsplit('.', 1)[0] for file in fire_images_files]
non_fire_labels = [file.rsplit('.', 1)[0] for file in non_fire_images_files]

# Split training data for fire and non-fire images separately
train_fire_images, test_fire_images, train_fire_labels, test_fire_labels = train_test_split(fire_images_files, fire_labels, test_size=0.2, random_sta
train_non_fire_images, test_non_fire_images, train_non_fire_labels, test_non_fire_labels = train_test_split(non_fire_images_files, non_fire_labels, t

# Split testing/validation data for fire and non-fire images separately
test_fire_images, val_fire_images, test_fire_labels, val_fire_labels = train_test_split(test_fire_images, test_fire_labels, test_size=0.5, random_sta
test_non_fire_images, val_non_fire_images, test_non_fire_labels, val_non_fire_labels = train_test_split(test_non_fire_images, test_non_fire_labels, t
# Combine the datasets
train_images = train_fire_images + train_non_fire_images
test_images = test_fire_images + test_non_fire_images
val_images = val_fire_images + val_non_fire_images
train_labels = train_fire_labels + train_non_fire_labels
test_labels = test_fire_labels + test_non_fire_labels
val_labels = val_fire_labels + val_non_fire_labels

# Print the sizes of training and testing sets
print(f"Number of training samples for fire data: {len(train_fire_images)}")
print(f"Number of testing samples for fire data: {len(test_fire_images)}")
print(f"Number of validation samples for fire data: {len(val_fire_labels)}")
print(f"Number of training samples for non-fire data: {len(train_non_fire_images)}")
print(f"Number of testing samples for non-fire data: {len(test_non_fire_images)}")
print(f"Number of validation samples for non-fire data: {len(val_non_fire_labels)}")
print()

# Print the sizes of training and testing sets
print(f"Number of training samples: {len(train_images)}")
print(f"Number of testing samples: {len(test_images)}")
print(f"Number of validation samples: {len(val_images)}")
print()

# Print the first few samples in each set
print("Training samples:")
print(train_images[:5])
print(train_labels[:5])
print(train_images[-5:])
print(train_labels[-5:])
print("Testing samples:")
print(test_images[:5])
print(test_labels[:5])
print(test_images[-5:])
print(test_labels[-5:])
print("Validation samples:")
print(val_images[:5])
print(val_images[:5])
print(val_images[-5:])
print(val_images[-5:])
print()

class_names = ['fire', 'non_fire']

# Define function to copy images to directories
def copy_images_to_directory(image_list, label_list, destination_dir, source_dir):
     # Create fire and non-fire subdirectories
    fire_dir = os.path.join(destination_dir, 'fire')
    non_fire_dir = os.path.join(destination_dir, 'non_fire')
    os.makedirs(fire_dir, exist_ok=True)
    os.makedirs(non_fire_dir, exist_ok=True)

    for image_name, label in zip(image_list, label_list):
        # Determine the destination directory based on the label
        if label == 'fire':
            label_dir = fire_dir
        else:
            label_dir = non_fire_dir
        # Check if the file already exists in the destination directory
        destination_file_path = os.path.join(label_dir, image_name + '.png')
        if not os.path.exists(destination_file_path):
            # Copy image to corresponding directory
            source_file_path = os.path.join(source_dir, image_name + '.png')
            shutil.copy(source_file_path, destination_file_path)
            print(str(label_dir) + ": Images added")
        else:
            print(str(label_dir) + ": Up to date")
            break

# Define directories for training, testing, and validation sets
train_dir = '/content/drive/MyDrive/AI4ALL/fire_dataset/train'
test_dir = '/content/drive/MyDrive/AI4ALL/fire_dataset/test'
val_dir = '/content/drive/MyDrive/AI4ALL/fire_dataset/val'

# Define source directories for fire and non-fire images
fire_source_dir = '/content/drive/MyDrive/AI4ALL/fire_dataset/fire_images'
non_fire_source_dir = '/content/drive/MyDrive/AI4ALL/fire_dataset/non_fire_images'

# # Copy fire images to training directory
# copy_images_to_directory(train_fire_images, train_fire_labels, train_dir, fire_source_dir)

# # Copy non-fire images to training directory
# copy_images_to_directory(train_non_fire_images, train_non_fire_labels, train_dir, non_fire_source_dir)
# copy_images_to_directory(train_non_fire_images, train_non_fire_labels, train_dir, non_fire_source_dir)

# # Copy fire images to testing directory
# copy_images_to_directory(test_fire_images, test_fire_labels, test_dir, fire_source_dir)

# # Copy non-fire images to testing directory
# copy_images_to_directory(test_non_fire_images, test_non_fire_labels, test_dir, non_fire_source_dir)

# # Copy fire images to validation directory
# copy_images_to_directory(val_fire_images, val_fire_labels, val_dir, fire_source_dir)

# # Copy non-fire images to validation directory
# copy_images_to_directory(val_non_fire_images, val_non_fire_labels, val_dir, non_fire_source_dir)
```

```
Number of files in fire_images directory: 755
Number of files in non_fire_images directory: 244

Number of training samples for fire data: 604
Number of testing samples for fire data: 75
Number of validation samples for fire data: 76
Number of training samples for non-fire data: 195
Number of testing samples for non-fire data: 24
Number of validation samples for non-fire data: 25

Number of training samples: 799
Number of testing samples: 99
Number of validation samples: 101

Training samples:
['fire.686', 'fire.209', 'fire.643', 'fire.739', 'fire.699']
['fire', 'fire', 'fire', 'fire', 'fire']
['non_fire.229', 'non_fire.61', 'non_fire.232', 'non_fire.169', 'non_fire.223']
['non_fire', 'non_fire', 'non_fire', 'non_fire', 'non_fire']
Testing samples:
['fire.23', 'fire.722', 'fire.59', 'fire.386', 'fire.400']
['fire', 'fire', 'fire', 'fire', 'fire']
['non_fire.209', 'non_fire.218', 'non_fire.41', 'non_fire.193', 'non_fire.17']
['non_fire', 'non_fire', 'non_fire', 'non_fire', 'non_fire']
Validation samples:
['fire.508', 'fire.513', 'fire.204', 'fire.563', 'fire.405']
['fire.508', 'fire.513', 'fire.204', 'fire.563', 'fire.405']
['non_fire.222', 'non_fire.118', 'non_fire.140', 'non_fire.32', 'non_fire.242']
['non_fire.222', 'non_fire.118', 'non_fire.140', 'non_fire.32', 'non_fire.242']
```

ResNet Model

Data Preprocessing

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

IMAGE_SIZE = (224, 224)
BATCH_SIZE = 32

# Create data generators with data augmentation for training
# Normalize pixel values to [0,1]
train_datagen = ImageDataGenerator(rescale=1./255)
validation_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches from directory
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='binary'  # Since it's binary classification, use 'binary' mode
)

# Flow validation images in batches from directory
validation_generator = validation_datagen.flow_from_directory(
    val_dir,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='binary'
)

# Flow test images in batches from directory
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='binary'
)
```

```
Found 803 images belonging to 2 classes.
Found 101 images belonging to 2 classes.
Found 99 images belonging to 2 classes.
```

Data Plotting

```
import matplotlib.pyplot as plt

# Function to plot 5 images from a generator
def plot_images(generator, num_images=5):
    # Get a batch of images and labels
    images, labels = next(generator)

    # Plot the images
    plt.figure(figsize=(10, 10))
    for i in range(num_images):
        plt.subplot(1, num_images, i+1)
        plt.imshow(images[i])
        plt.title('Label: {}'.format(labels[i]))
        plt.axis('off')
    plt.show()

# Plot images from the training generator
plot_images(train_generator)

# Plot images from the test generator
plot_images(test_generator)

# Plot images from the validation generator
plot_images(validation_generator)
```

Training the Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.optimizers import Adam

# Step 1: Pretrained Model Selection
pretrained_base = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Step 2: Model Definition and Transfer Learning
model = Sequential([
    pretrained_base,
    Flatten(),
    Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),  # Add dropout regularization
    Dense(1, activation='sigmoid')  # Output layer for binary classification
])

# Freeze the pretrained layers
pretrained_base.trainable = False

# Compile the model
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])

# Step 3: Model Training

history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=5,
    validation_data=validation_generator,
    validation_steps=len(validation_generator),
)

# Step 4: Model Evaluation
test_loss, test_accuracy = model.evaluate(test_generator, steps=len(test_generator))
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")

# Step 5: Model Deployment
model.save('wildfire_classification_resnet50_model.h5')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 4s 0us/step
Epoch 1/5
26/26 [==============================] - 431s 16s/step - loss: 2.4388 - accuracy: 0.6575 - val_loss: 0.5513 - val_accuracy: 0.7525
Epoch 2/5
26/26 [==============================] - 19s 739ms/step - loss: 0.5788 - accuracy: 0.7435 - val_loss: 0.5495 - val_accuracy: 0.7525
```

```
    Epoch 3/5
    26/26 [==============================] - 20s 790ms/step - loss: 0.5616 - accuracy: 0.7522 - val_loss: 0.5541 - val_accuracy: 0.7525
    Epoch 4/5
    26/26 [==============================] - 19s 728ms/step - loss: 0.5355 - accuracy: 0.7422 - val_loss: 0.5445 - val_accuracy: 0.7525
    Epoch 5/5
    26/26 [==============================] - 20s 778ms/step - loss: 0.5334 - accuracy: 0.7522 - val_loss: 0.4734 - val_accuracy: 0.7525
    4/4 [==============================] - 42s 14s/step - loss: 0.5031 - accuracy: 0.7576
    Test Loss: 0.5031171441078186
    Test Accuracy: 0.7575757503509521
    /usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.sav
      saving_api.save_model(
```

Load the Model

```
import numpy as np
from tensorflow.keras.preprocessing import image

# Load the saved model
model = tf.keras.models.load_model('wildfire_classification_resnet50_model.h5')
```

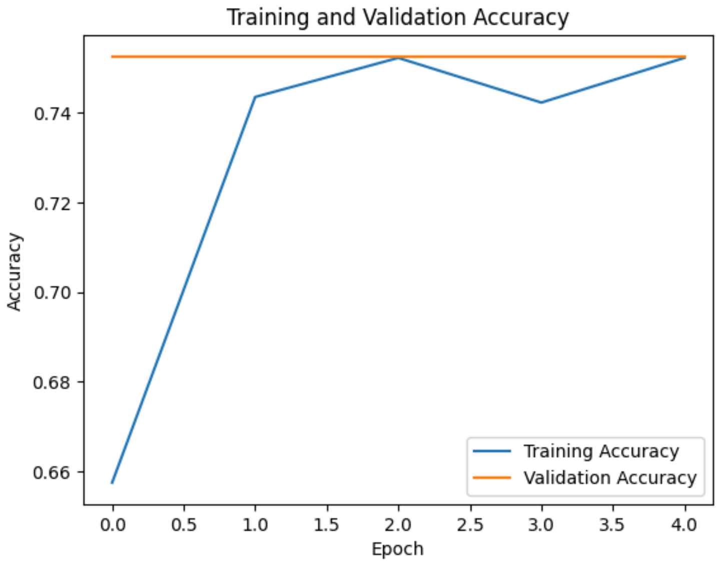
Using the Model

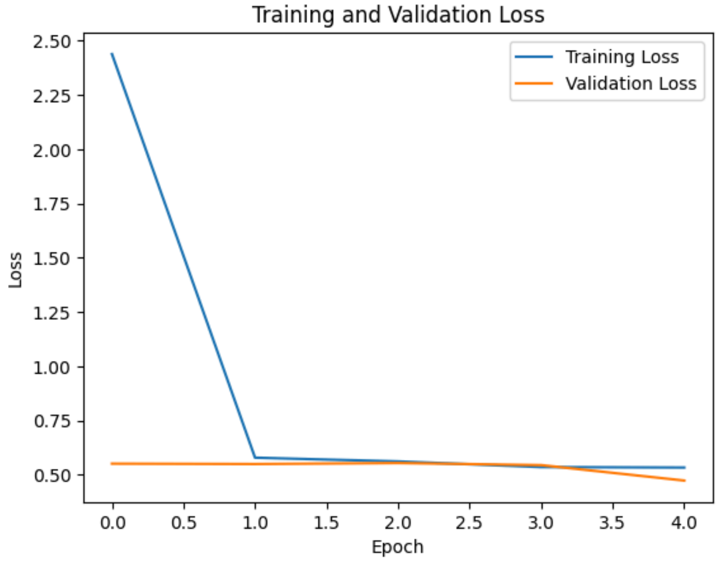
```
# Define a function to preprocess the input image
def preprocess_image(image_path):
    img = image.load_img(image_path, target_size=(224, 224))  # Resize image to match model input size
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    img_array = img_array / 255.0  # Normalize pixel values
    return img_array

# Path to the image you want to classify
image_path = '/content/drive/MyDrive/AI4ALL/fire_dataset/val/fire/fire.116.png'

# Preprocess the input image
processed_image = preprocess_image(image_path)

# Make predictions
predictions = model.predict(processed_image)
print(predictions)

# Interpret the predictions
if predictions[0] > 0.5:
    print("The image contains a wildfire.")
else:
    print("The image does not contain a wildfire.")
```

```
    1/1 [==============================] - 2s 2s/step
    [[0.38049394]]
    The image does not contain a wildfire.
```

Testing Model

```
# Plot training history
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```





```
test_loss, test_accuracy = model.evaluate(test_generator, steps=len(test_generator))
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

```
     4/4 [==============================] - 2s 512ms/step - loss: 0.5031 - accuracy: 0.7576
     Test Loss: 0.5031171441078186
     Test Accuracy: 0.7575757503509521
```

```
from sklearn.metrics import confusion_matrix

# Predict labels for test data
test_predictions = model.predict(test_generator)

# Convert probabilities to binary predictions
binary_predictions = np.where(test_predictions > 0.5, 1, 0)

# Get true labels
true_labels = test_generator.classes

# Generate confusion matrix
conf_matrix = confusion_matrix(true_labels, binary_predictions)
tn, fp, fn, tp = conf_matrix.ravel()


print("Confusion Matrix:")
print(conf_matrix)
print(tn)
print(fp)
print(fn)
print(tp)
```

```
4/4 [==============================] - 4s 1s/step
Confusion Matrix:
[[75  0]
 [24  0]]
75
0
24
0
```

```
from sklearn.metrics import classification_report

# Generate classification report
class_report = classification_report(true_labels, binary_predictions, target_names=class_names)

print("Classification Report:")
print(class_report)
```

```
sification Report:
          precision    recall  f1-score   support

    fire       0.76      1.00      0.86        75
non_fire       0.00      0.00      0.00        24

accuracy                           0.76        99
acro avg       0.38      0.50      0.43        99
hted avg       0.57      0.76      0.65        99

/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and b
arn_prf(average, modifier, msg_start, len(result))
/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and b
arn_prf(average, modifier, msg_start, len(result))
/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and b
arn_prf(average, modifier, msg_start, len(result))
```

Inception-ResNetV2 Model

Data Preprocessing

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

IMAGE_SIZE = (299, 299)
BATCH_SIZE = 32

# Define image data generators
train_datagen = ImageDataGenerator(rescale=1./255)
validation_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=IMAGE_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='binary')

validation_generator = validation_datagen.flow_from_directory(
        val_dir,
        target_size=IMAGE_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='binary')

test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=IMAGE_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='binary')
```

```
Found 803 images belonging to 2 classes.
Found 101 images belonging to 2 classes.
Found 99 images belonging to 2 classes.
```

Data Plotting

```
import matplotlib.pyplot as plt

# Function to plot 5 images from a generator
def plot_images(generator, num_images=5):
    # Get a batch of images and labels
    images, labels = next(generator)

    # Plot the images
    plt.figure(figsize=(10, 10))
    for i in range(num_images):
        plt.subplot(1, num_images, i+1)
        plt.imshow(images[i])
        plt.title('Label: {}'.format(labels[i]))
        plt.axis('off')
    plt.show()

# Plot images from the training generator
plot_images(train_generator)

# Plot images from the test generator
plot_images(test_generator)

# Plot images from the validation generator
plot_images(validation_generator)
```

Training the Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.applications import InceptionResNetV2
from tensorflow.keras.optimizers import Adam

# Load Inception-ResNetV2 model
pretrained_base = InceptionResNetV2(weights='imagenet', include_top=False, input_shape=(299, 299, 3))

# Transfer Learning Setup
model = Sequential([
    pretrained_base,
    Flatten(),
    tf.keras.layers.Dropout(0.5),
    Dense(1, activation='sigmoid')  # Output layer for binary classification
])

# Freeze the pretrained layers
pretrained_base.trainable = False

# Compile the model
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=5,
    validation_data=validation_generator,
    validation_steps=len(validation_generator)
)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_generator, steps=len(test_generator))
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")

# Save the model
model.save('inception_resnetv2_model.tf')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_resnet_v2/inception_resnet_v2_weights_tf_dim_order:
219055592/219055592 [==============================] - 8s 0us/step
Epoch 1/5
26/26 [==============================] - 58s 2s/step - loss: 1.2827 - accuracy: 0.8356 - val_loss: 0.0078 - val_accuracy: 1.0000
Epoch 2/5
26/26 [==============================] - 24s 902ms/step - loss: 0.1890 - accuracy: 0.9701 - val_loss: 0.0016 - val_accuracy: 1.0000
Epoch 3/5
26/26 [==============================] - 25s 991ms/step - loss: 0.0343 - accuracy: 0.9913 - val_loss: 0.0020 - val_accuracy: 1.0000
```

```
Epoch 4/5
26/26 [==============================] - 22s 853ms/step - loss: 0.0077 - accuracy: 0.9963 - val_loss: 0.0033 - val_accuracy: 1.0000
Epoch 5/5
26/26 [==============================] - 24s 874ms/step - loss: 0.0074 - accuracy: 0.9975 - val_loss: 0.0059 - val_accuracy: 1.0000
4/4 [==============================] - 2s 388ms/step - loss: 0.7140 - accuracy: 0.9596
Test Loss: 0.7139979600906372
Test Accuracy: 0.9595959782600403
```

Load the Model

```
import cv2
import numpy as np
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import load_model

import tensorflow as tf

# Load model
model = load_model('/content/inception_resnetv2_model.tf')

# Now the model is ready for predictions or evaluations
```

Using the Model

```
# test with images
# Define the path to the image you want to test
image_path = '/content/drive/MyDrive/Extra_Test_Images/weird_images/tvfire.png'

# Load and preprocess the image
img = cv2.imread(image_path)
img = cv2.resize(img, (299, 299))
img = img / 255.0  # Rescale pixel values
img = np.expand_dims(img, axis=0)  # Add batch dimension
```