```
!pip install -q git+https://github.com/tensorflow/docs
```

```
    Preparing metadata (setup.py) ... done
    Building wheel for tensorflow-docs (setup.py) ... done
```

```
!wget -q https://github.com/sayakpaul/Action-Recognition-in-TensorFlow/releases/download/v1.0.0/ucf101_top5.tar.gz
!tar xf ucf101_top5.tar.gz
```

Setup

```
from tensorflow_docs.vis import embed
from tensorflow import keras
from imutils import paths

import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
import numpy as np
import imageio
import cv2
import os
```

Define hyperparameters

```
IMG_SIZE = 224
BATCH_SIZE = 64
EPOCHS = 10

MAX_SEQ_LENGTH = 20
NUM_FEATURES = 2048
```

Data preparation

```
train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")

print(f"Total videos for training: {len(train_df)}")
print(f"Total videos for testing: {len(test_df)}")

train_df.sample(10)
```

```
Total videos for training: 594
Total videos for testing: 224
```

	video_name	tag
177	v_PlayingCello_g17_c01.avi	PlayingCello
365	v_ShavingBeard_g08_c07.avi	ShavingBeard
71	v_CricketShot_g18_c03.avi	CricketShot
6	v_CricketShot_g08_c07.avi	CricketShot
561	v_TennisSwing_g20_c04.avi	TennisSwing
144	v_PlayingCello_g11_c07.avi	PlayingCello
323	v_Punch_g20_c04.avi	Punch
583	v_TennisSwing_g24_c02.avi	TennisSwing
556	v_TennisSwing_g19_c05.avi	TennisSwing
224	v_PlayingCello_g24_c01.avi	PlayingCello

```
# The following two methods are taken from this tutorial:
# https://www.tensorflow.org/hub/tutorials/action_recognition_with_tf_hub
```

```
def crop_center_square(frame):
    y, x = frame.shape[0:2]
    min_dim = min(y, x)
    start_x = (x // 2) - (min_dim // 2)
    start_y = (y // 2) - (min_dim // 2)
    return frame[start_y : start_y + min_dim, start_x : start_x + min_dim]
```

```
def load_video(path, max_frames=0, resize=(IMG_SIZE, IMG_SIZE)):
    cap = cv2.VideoCapture(path)
    frames = []
    try:
        while True:
            ret, frame = cap.read()
            if not ret:
                break
            frame = crop_center_square(frame)
            frame = cv2.resize(frame, resize)
            frame = frame[:, :, [2, 1, 0]]
            frames.append(frame)

            if len(frames) == max_frames:
                break
    finally:
        cap.release()
    return np.array(frames)
```

We can use a pre-trained network to extract meaningful features from the extracted frames. The `Keras Applications` module provides a number of state-of-the-art models pre-trained on the ImageNet-1k dataset. We will be using the InceptionV3 model for this purpose.

```
def build_feature_extractor():
    feature_extractor = keras.applications.InceptionV3(
        weights="imagenet",
        include_top=False,
        pooling="avg",
        input_shape=(IMG_SIZE, IMG_SIZE, 3),
    )
    preprocess_input = keras.applications.inception_v3.preprocess_input

    inputs = keras.Input((IMG_SIZE, IMG_SIZE, 3))
    preprocessed = preprocess_input(inputs)

    outputs = feature_extractor(preprocessed)
    return keras.Model(inputs, outputs, name="feature_extractor")


feature_extractor = build_feature_extractor()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_orderin
87910968/87910968 [==============================] - 0s 0us/step
```

The labels of the videos are strings. Neural networks do not understand string values, so they must be converted to some numerical form before they are fed to the model. Here we will use the `StringLookup` layer encode the class labels as integers.

```
label_processor = keras.layers.StringLookup(
    num_oov_indices=0, vocabulary=np.unique(train_df["tag"])
)
print(label_processor.get_vocabulary())
```

```
['CricketShot', 'PlayingCello', 'Punch', 'ShavingBeard', 'TennisSwing']
```

Finally, we can put all the pieces together to create our data processing utility.

```
def prepare_all_videos(df, root_dir):
    num_samples = len(df)
    video_paths = df["video_name"].values.tolist()
    labels = df["tag"].values
    labels = label_processor(labels[..., None]).numpy()

    # `frame_masks` and `frame_features` are what we will feed to our sequence model.
    # `frame_masks` will contain a bunch of booleans denoting if a timestep is
    # masked with padding or not.
    frame_masks = np.zeros(shape=(num_samples, MAX_SEQ_LENGTH), dtype="bool")
    frame_features = np.zeros(
        shape=(num_samples, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32"
    )

    # For each video.
    for idx, path in enumerate(video_paths):
```

```
        # Gather all its frames and add a batch dimension.
        frames = load_video(os.path.join(root_dir, path))
        frames = frames[None, ...]

        # Initialize placeholders to store the masks and features of the current video.
        temp_frame_mask = np.zeros(shape=(1, MAX_SEQ_LENGTH,), dtype="bool")
        temp_frame_features = np.zeros(
            shape=(1, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32"
        )

        # Extract features from the frames of the current video.
        for i, batch in enumerate(frames):
            video_length = batch.shape[0]
            length = min(MAX_SEQ_LENGTH, video_length)
            for j in range(length):
                temp_frame_features[i, j, :] = feature_extractor.predict(
                    batch[None, j, :]
                )
            temp_frame_mask[i, :length] = 1  # 1 = not masked, 0 = masked

        frame_features[idx,] = temp_frame_features.squeeze()
        frame_masks[idx,] = temp_frame_mask.squeeze()

    return (frame_features, frame_masks), labels


train_data, train_labels = prepare_all_videos(train_df, "train")
test_data, test_labels = prepare_all_videos(test_df, "test")

print(f"Frame features in train set: {train_data[0].shape}")
print(f"Frame masks in train set: {train_data[1].shape}")
```

Streaming output truncated to the last 5000 lines.

```
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 34ms/step
```

```
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 34ms/step
```

The above code block will take ~20 minutes to execute depending on the machine it's being executed.

The sequence model

Now, we can feed this data to a sequence model consisting of recurrent layers like `GRU`.

```
# Utility for our sequence model.
def get_sequence_model():
    class_vocab = label_processor.get_vocabulary()

    frame_features_input = keras.Input((MAX_SEQ_LENGTH, NUM_FEATURES))
    mask_input = keras.Input((MAX_SEQ_LENGTH,), dtype="bool")

    # Refer to the following tutorial to understand the significance of using `mask`:
    # https://keras.io/api/layers/recurrent_layers/gru/
    x = keras.layers.GRU(16, return_sequences=True)(
        frame_features_input, mask=mask_input
    )
    x = keras.layers.GRU(8)(x)
    x = keras.layers.Dropout(0.4)(x)
    x = keras.layers.Dense(8, activation="relu")(x)
    output = keras.layers.Dense(len(class_vocab), activation="softmax")(x)

    rnn_model = keras.Model([frame_features_input, mask_input], output)

    rnn_model.compile(
        loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"]
    )
    return rnn_model


# Utility for running experiments.
def run_experiment():
    filepath = "/tmp/video_classifier"
    checkpoint = keras.callbacks.ModelCheckpoint(
        filepath, save_weights_only=True, save_best_only=True, verbose=1
    )

    seq_model = get_sequence_model()
    history = seq_model.fit(
        [train_data[0], train_data[1]],
        train_labels,
        validation_split=0.3,
        epochs=EPOCHS,
        callbacks=[checkpoint],
    )

    seq_model.load_weights(filepath)
    _, accuracy = seq_model.evaluate([test_data[0], test_data[1]], test_labels)
    print(f"Test accuracy: {round(accuracy * 100, 2)}%")

    return history, seq_model


_, sequence_model = run_experiment()
```

```
Epoch 1/10
12/13 [==========================>...] - ETA: 0s - loss: 1.4486 - accuracy: 0.3464
Epoch 1: val_loss improved from inf to 1.76701, saving model to /tmp/video_classifier
13/13 [==============================] - 14s 310ms/step - loss: 1.4283 - accuracy: 0.3566 - val_loss: 1.7670 - val_accuracy: 0.0000
Epoch 2/10
13/13 [==============================] - ETA: 0s - loss: 1.0994 - accuracy: 0.5325
Epoch 2: val_loss did not improve from 1.76701
13/13 [==============================] - 0s 19ms/step - loss: 1.0994 - accuracy: 0.5325 - val_loss: 1.8752 - val_accuracy: 0.0000e+
Epoch 3/10
10/13 [======================>.......] - ETA: 0s - loss: 1.0081 - accuracy: 0.5531
Epoch 3: val_loss did not improve from 1.76701
13/13 [==============================] - 0s 19ms/step - loss: 0.9831 - accuracy: 0.5711 - val_loss: 1.9777 - val_accuracy: 0.0000e+
Epoch 4/10
13/13 [==============================] - ETA: 0s - loss: 0.8861 - accuracy: 0.5855
Epoch 4: val_loss did not improve from 1.76701
13/13 [==============================] - 0s 21ms/step - loss: 0.8861 - accuracy: 0.5855 - val_loss: 2.1222 - val_accuracy: 0.0000e+
Epoch 5/10
13/13 [==============================] - ETA: 0s - loss: 0.8114 - accuracy: 0.5880
Epoch 5: val_loss did not improve from 1.76701
13/13 [==============================] - 0s 21ms/step - loss: 0.8114 - accuracy: 0.5880 - val_loss: 2.2483 - val_accuracy: 0.0000e+
Epoch 6/10
13/13 [==============================] - ETA: 0s - loss: 0.7724 - accuracy: 0.6024
```

```
Epoch 6: val_loss did not improve from 1.76701
13/13 [==============================] - 0s 22ms/step - loss: 0.7724 - accuracy: 0.6024 - val_loss: 2.3961 - val_accuracy: 0.0000e+
Epoch 7/10
12/13 [==========================>...] - ETA: 0s - loss: 0.7187 - accuracy: 0.6120
Epoch 7: val_loss did not improve from 1.76701
13/13 [==============================] - 0s 21ms/step - loss: 0.7226 - accuracy: 0.6096 - val_loss: 2.5370 - val_accuracy: 0.0000e+
Epoch 8/10
10/13 [======================>.......] - ETA: 0s - loss: 0.6848 - accuracy: 0.6812
Epoch 8: val_loss did not improve from 1.76701
13/13 [==============================] - 0s 19ms/step - loss: 0.6706 - accuracy: 0.6819 - val_loss: 2.5707 - val_accuracy: 0.0615
Epoch 9/10
11/13 [========================>.....] - ETA: 0s - loss: 0.6324 - accuracy: 0.6790
Epoch 9: val_loss did not improve from 1.76701
13/13 [==============================] - 0s 19ms/step - loss: 0.6331 - accuracy: 0.6867 - val_loss: 2.7218 - val_accuracy: 0.3017
Epoch 10/10
10/13 [======================>.......] - ETA: 0s - loss: 0.5912 - accuracy: 0.7844
Epoch 10: val_loss did not improve from 1.76701
13/13 [==============================] - 0s 20ms/step - loss: 0.5788 - accuracy: 0.8024 - val_loss: 2.7857 - val_accuracy: 0.3296
7/7 [==============================] - 0s 8ms/step - loss: 1.3359 - accuracy: 0.4107
Test accuracy: 41.07%
```

Note: To keep the runtime of this example relatively short, we just used a few training examples. This number of training examples is low with respect to the sequence model being used that has 99,909 trainable parameters. You are encouraged to sample more data from the UCF101 dataset using the notebook mentioned above and train the same model.

Inference

```
def prepare_single_video(frames):
    frames = frames[None, ...]
    frame_mask = np.zeros(shape=(1, MAX_SEQ_LENGTH,), dtype="bool")
    frame_features = np.zeros(shape=(1, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32")

    for i, batch in enumerate(frames):
        video_length = batch.shape[0]
        length = min(MAX_SEQ_LENGTH, video_length)
        for j in range(length):
            frame_features[i, j, :] = feature_extractor.predict(batch[None, j, :])
        frame_mask[i, :length] = 1  # 1 = not masked, 0 = masked

    return frame_features, frame_mask


def sequence_prediction(path):
    class_vocab = label_processor.get_vocabulary()

    frames = load_video(os.path.join("test", path))
    frame_features, frame_mask = prepare_single_video(frames)
    probabilities = sequence_model.predict([frame_features, frame_mask])[0]

    for i in np.argsort(probabilities)[::-1]:
        print(f"  {class_vocab[i]}: {probabilities[i] * 100:5.2f}%")
    return frames


# This utility is for visualization.
# Referenced from:
# https://www.tensorflow.org/hub/tutorials/action_recognition_with_tf_hub
def to_gif(images):
    converted_images = images.astype(np.uint8)
    imageio.mimsave("animation.gif", converted_images, duration=100)
    return embed.embed_file("animation.gif")


test_video = np.random.choice(test_df["video_name"].values.tolist())
print(f"Test video path: {test_video}")
test_frames = sequence_prediction(test_video)
to_gif(test_frames[:MAX_SEQ_LENGTH])
```

```
Test video path: v_CricketShot_g05_c03.avi
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 4s 4s/step
  CricketShot: 42.21%
  PlayingCello: 22.59%
  ShavingBeard: 12.87%
  TennisSwing: 11.93%
  Punch: 10.41%
```