```
!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 xf ucf101_top5.tar.gz
```

```
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
```

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

MAX_SEQ_LENGTH = 20
NUM_FEATURES = 2048
```

```
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
448	v_ShavingBeard_g21_c06.avi	ShavingBeard
525	v_TennisSwing_g15_c01.avi	TennisSwing
192	v_PlayingCello_g19_c02.avi	PlayingCello
581	v_TennisSwing_g23_c06.avi	TennisSwing
588	v_TennisSwing_g25_c01.avi	TennisSwing
407	v_ShavingBeard_g15_c06.avi	ShavingBeard
359	v_ShavingBeard_g08_c01.avi	ShavingBeard
207	v_PlayingCello_g21_c03.avi	PlayingCello
222	v_PlayingCello_g23_c06.avi	PlayingCello
585	v_TennisSwing_g24_c04.avi	TennisSwing

```
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)
```

```
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_ordering_tf
87910968/87910968 [==============================] - 0s 0us/step
```

```
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']
```

```
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 = 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 idx, path in enumerate(video_paths):
        frames = load_video(os.path.join(root_dir, path))
        frames = frames[None, ...]
        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"
        )
        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

        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}")
```

```
1/1 [==============================] - 0s 73ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 77ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 64ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
```

```
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")
    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
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: 1.3072 - accuracy: 0.4844
Epoch 1: val_loss improved from inf to 1.63220, saving model to /tmp/video_classifier
13/13 [==============================] - 14s 271ms/step - loss: 1.2843 - accuracy: 0.5012 - val_loss: 1.6322 - val_accuracy: 0.1397
Epoch 2/10
13/13 [==============================] - ETA: 0s - loss: 0.9665 - accuracy: 0.7181
Epoch 2: val_loss improved from 1.63220 to 1.58667, saving model to /tmp/video_classifier
13/13 [==============================] - 0s 38ms/step - loss: 0.9665 - accuracy: 0.7181 - val_loss: 1.5867 - val_accuracy: 0.2011
Epoch 3/10
12/13 [==========================>...] - ETA: 0s - loss: 0.8751 - accuracy: 0.8073
Epoch 3: val_loss did not improve from 1.58667
13/13 [==============================] - 0s 34ms/step - loss: 0.8650 - accuracy: 0.8193 - val_loss: 1.8510 - val_accuracy: 0.2514
Epoch 4/10
12/13 [==========================>...] - ETA: 0s - loss: 0.7857 - accuracy: 0.8333
Epoch 4: val_loss did not improve from 1.58667
13/13 [==============================] - 0s 33ms/step - loss: 0.7725 - accuracy: 0.8386 - val_loss: 1.8268 - val_accuracy: 0.2905
Epoch 5/10
13/13 [==============================] - ETA: 0s - loss: 0.7205 - accuracy: 0.8723
Epoch 5: val_loss did not improve from 1.58667
13/13 [==============================] - 0s 22ms/step - loss: 0.7205 - accuracy: 0.8723 - val_loss: 1.9336 - val_accuracy: 0.3017
Epoch 6/10
13/13 [==============================] - ETA: 0s - loss: 0.6462 - accuracy: 0.9229
Epoch 6: val_loss did not improve from 1.58667
13/13 [==============================] - 0s 21ms/step - loss: 0.6462 - accuracy: 0.9229 - val_loss: 1.9797 - val_accuracy: 0.3240
Epoch 7/10
13/13 [==============================] - ETA: 0s - loss: 0.5791 - accuracy: 0.9253
Epoch 7: val_loss did not improve from 1.58667
13/13 [==============================] - 0s 21ms/step - loss: 0.5791 - accuracy: 0.9253 - val_loss: 2.0976 - val_accuracy: 0.3128
Epoch 8/10
13/13 [==============================] - ETA: 0s - loss: 0.5171 - accuracy: 0.9542
Epoch 8: val_loss did not improve from 1.58667
13/13 [==============================] - 0s 21ms/step - loss: 0.5171 - accuracy: 0.9542 - val_loss: 2.1763 - val_accuracy: 0.3296
Epoch 9/10
13/13 [==============================] - ETA: 0s - loss: 0.4852 - accuracy: 0.9398
Epoch 9: val_loss did not improve from 1.58667
13/13 [==============================] - 0s 21ms/step - loss: 0.4852 - accuracy: 0.9398 - val_loss: 2.2442 - val_accuracy: 0.3128
Epoch 10/10
13/13 [==============================] - ETA: 0s - loss: 0.4472 - accuracy: 0.9566
Epoch 10: val_loss did not improve from 1.58667
13/13 [==============================] - 0s 21ms/step - loss: 0.4472 - accuracy: 0.9566 - val_loss: 2.2161 - val_accuracy: 0.3184
7/7 [==============================] - 0s 7ms/step - loss: 1.0930 - accuracy: 0.7054
Test accuracy: 70.54%
```

```
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
def to_gif(images):
    converted_images = images.astype(np.uint8)
```

```
    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_ShavingBeard_g04_c01.avi
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 2s 2s/step
  ShavingBeard: 26.35%
  PlayingCello: 26.27%
  Punch: 18.13%
  CricketShot: 15.62%
  TennisSwing: 13.62%
```