Inappropriate audio and video detection

week 01 objective:

video classification into inappropriate & appropriate

- We created a video classifier using CNN and RNN. We used Keras and TensorFlow for this.
- We downloaded the dataset from the internet and trained it using TensorFlow
- Then we deployed our model and tested it

code:

Video Classifier Using CNN and RNN

#!dir

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
# os.listdir('dataset')
dataset_path = os.listdir('dataset/train')

label_types = os.listdir('dataset/train')
print (label_types)
```

```
['.ipynb_checkpoints', 'appropriate', 'inappropriate']
```

Preparing Training Data

In [2]:
```
rooms = []

for item in dataset_path:
 # Get all the file names
 all_rooms = os.listdir('dataset/train' + '/' +item)

 # Add them to the list
 for room in all_rooms:
    rooms.append((item, str('dataset/train' + '/' +item) + '/' + room))

# Build a dataframe
train_df = pd.DataFrame(data=rooms, columns=['tag', 'video_name'])
print(train_df.head())
print(train_df.tail())
```

```
          tag                                         video_name
0  appropriate  dataset/train/appropriate/pexels-diva-plavalag...
1  appropriate  dataset/train/appropriate/production_id_369594...
2  appropriate  dataset/train/appropriate/production_id_483221...
3  appropriate  dataset/train/appropriate/production_id_491974...
4  appropriate  dataset/train/appropriate/production_id_512041...
             tag                                         video_name
10  inappropriate    dataset/train/inappropriate/far_distance.gif
11  inappropriate  dataset/train/inappropriate/low_resolution.gif
12  inappropriate        dataset/train/inappropriate/result_1.gif
13  inappropriate        dataset/train/inappropriate/result_2.gif
14  inappropriate       dataset/train/inappropriate/transient.gif
```

In [3]:
```
df = train_df.loc[:,['video_name','tag']]
df
df.to_csv('train.csv')
```

Preparing Test Data

In [4]:
```
dataset_path = os.listdir('dataset/test')
print(dataset_path)

room_types = os.listdir('dataset/test')
print("Types of activities found: ", len(dataset_path))

rooms = []

for item in dataset_path:
 # Get all the file names
 all_rooms = os.listdir('dataset/test' + '/' +item)

 # Add them to the list
 for room in all_rooms:
    rooms.append((item, str('dataset/test' + '/' +item) + '/' + room))

# Build a dataframe
test_df = pd.DataFrame(data=rooms, columns=['tag', 'video_name'])
print(test_df.head())
print(test_df.tail())

df = test_df.loc[:,['video_name','tag']]
df
df.to_csv('test.csv')
```

```
['.ipynb_checkpoints', 'appropriate', 'inappropriate']
Types of activities found:  3
           tag                                    video_name
0  appropriate         dataset/test/appropriate/blocked.gif
1  appropriate         dataset/test/appropriate/crowded.gif
2  appropriate           dataset/test/appropriate/demo1.gif
3  appropriate    dataset/test/appropriate/far_distance.gif
4  appropriate  dataset/test/appropriate/low_resolution.gif
              tag                                              video_name
10  inappropriate  dataset/test/inappropriate/production_id_48322...
11  inappropriate  dataset/test/inappropriate/production_id_49197...
12  inappropriate  dataset/test/inappropriate/production_id_51204...
13  inappropriate       dataset/test/inappropriate/video (1080p).mp4
14  inappropriate       dataset/test/inappropriate/video (2160p).mp4
```

In [3]:
```
!pip install git+https://github.com/tensorflow/docs
```

```
Collecting git+https://github.com/tensorflow/docs
  Cloning https://github.com/tensorflow/docs (https://github.com/tensorflow/
docs) to c:\users\acer\appdata\local\temp\pip-req-build-ndw3inet
  Resolved https://github.com/tensorflow/docs (https://github.com/tensorflo
w/docs) to commit 2b700605aaf42a346624aaff5c84879999d4c407
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Requirement already satisfied: astor in c:\users\acer\anaconda3\envs\tf\lib
\site-packages (from tensorflow-docs==2023.9.4.19335) (0.8.1)
Requirement already satisfied: absl-py in c:\users\acer\anaconda3\envs\tf\li
b\site-packages (from tensorflow-docs==2023.9.4.19335) (1.4.0)
Requirement already satisfied: jinja2 in c:\users\acer\anaconda3\envs\tf\lib
\site-packages (from tensorflow-docs==2023.9.4.19335) (3.1.2)
Requirement already satisfied: nbformat in c:\users\acer\anaconda3\envs\tf\l
ib\site-packages (from tensorflow-docs==2023.9.4.19335) (5.9.2)
Requirement already satisfied: protobuf>=3.12 in c:\users\acer\anaconda3\env
s\tf\lib\site-packages (from tensorflow-docs==2023.9.4.19335) (3.20.3)
Requirement already satisfied: pyyaml in c:\users\acer\anaconda3\envs\tf\lib
\site-packages (from tensorflow-docs==2023.9.4.19335) (6.0.1)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\acer\anaconda3\en
vs\tf\lib\site-packages (from jinja2->tensorflow-docs==2023.9.4.19335) (2.1.
1)
Requirement already satisfied: fastjsonschema in c:\users\acer\anaconda3\env
s\tf\lib\site-packages (from nbformat->tensorflow-docs==2023.9.4.19335) (2.1
6.2)
Requirement already satisfied: jsonschema>=2.6 in c:\users\acer\anaconda3\en
vs\tf\lib\site-packages (from nbformat->tensorflow-docs==2023.9.4.19335) (4.
17.3)
Requirement already satisfied: jupyter-core in c:\users\acer\anaconda3\envs
\tf\lib\site-packages (from nbformat->tensorflow-docs==2023.9.4.19335) (5.3.
0)
Requirement already satisfied: traitlets>=5.1 in c:\users\acer\anaconda3\env
s\tf\lib\site-packages (from nbformat->tensorflow-docs==2023.9.4.19335) (5.
7.1)
Requirement already satisfied: attrs>=17.4.0 in c:\users\acer\anaconda3\envs
\tf\lib\site-packages (from jsonschema>=2.6->nbformat->tensorflow-docs==202
3.9.4.19335) (22.1.0)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0
in c:\users\acer\anaconda3\envs\tf\lib\site-packages (from jsonschema>=2.6->
nbformat->tensorflow-docs==2023.9.4.19335) (0.18.0)
Requirement already satisfied: platformdirs>=2.5 in c:\users\acer\anaconda3
\envs\tf\lib\site-packages (from jupyter-core->nbformat->tensorflow-docs==20
23.9.4.19335) (3.10.0)
Requirement already satisfied: pywin32>=300 in c:\users\acer\anaconda3\envs
\tf\lib\site-packages (from jupyter-core->nbformat->tensorflow-docs==2023.9.
4.19335) (305.1)

  Running command git clone --filter=blob:none --quiet https://github.com/te
nsorflow/docs (https://github.com/tensorflow/docs) 'C:\Users\ACER\AppData\Lo
cal\Temp\pip-req-build-ndw3inet'
```

In [5]:
```
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
```

In [6]:
```
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
  try:
    tf.config.experimental.set_virtual_device_configuration(
        gpus[0],[tf.config.experimental.VirtualDeviceConfiguration(memory_limi
  except RuntimeError as e:
    print(e)
```

Data preparation

In [7]:
```
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: 15
Total videos for testing: 15
```

Out[7]:

	Unnamed: 0	video_name	tag
11	11	dataset/train/inappropriate/low_resolution.gif	inappropriate
4	4	dataset/train/appropriate/production_id_512041...	appropriate
1	1	dataset/train/appropriate/production_id_369594...	appropriate
7	7	dataset/train/inappropriate/blocked.gif	inappropriate
9	9	dataset/train/inappropriate/demo1.gif	inappropriate
8	8	dataset/train/inappropriate/crowded.gif	inappropriate
10	10	dataset/train/inappropriate/far_distance.gif	inappropriate
13	13	dataset/train/inappropriate/result_2.gif	inappropriate
5	5	dataset/train/appropriate/video (1080p).mp4	appropriate
2	2	dataset/train/appropriate/production_id_483221...	appropriate

Feed the videos to a network:

In [9]:
```
# The following two methods are taken from this tutorial:
# https://www.tensorflow.org/hub/tutorials/action_recognition_with_tf_hub
IMG_SIZE = 224


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)
```

Feature Extraction

In [11]:
```
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()
```

Label Encoding

StringLookup layer encode the class labels as integers.

In [13]:
```
label_processor = keras.layers.StringLookup(num_oov_indices=0, vocabulary=np.u
print(label_processor.get_vocabulary())

labels = train_df["tag"].values
labels = label_processor(labels[..., None]).numpy()
labels
```

```
['appropriate', 'inappropriate']
```

Out[13]:
```
array([[0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1]], dtype=int64)
```

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

In [10]:
```
#print(train_data[0].shape)
#train_data[0]
```

In [20]:
```
#Define hyperparameters

IMG_SIZE = 224
BATCH_SIZE = 64
EPOCHS = 100

MAX_SEQ_LENGTH = 20
NUM_FEATURES = 2048
```

In [24]:
```
def prepare_all_videos(df, root_dir):
    num_samples = len(df)
    video_paths = df["video_name"].values.tolist()

    ##take all classlabels from train_df column named 'tag' and store in label
    labels = df["tag"].values

    #convert classlabels to label encoding
    labels = label_processor(labels[..., None]).numpy()

    # `frame_masks` and `frame_features` are what we will feed to our sequence
    # `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

    # 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 curre
        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}")



print(f"train_labels in train set: {train_labels.shape}")

print(f"test_labels in train set: {test_labels.shape}")

# MAX_SEQ_LENGTH = 20, NUM_FEATURES = 2048. We have defined this above under h
```

```
Frame features in train set: (15, 20, 2048)
Frame masks in train set: (15, 20)
train_labels in train set: (15, 1)
test_labels in train set: (15, 1)
```

In [ ]:

The sequence model

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

In [25]:
```
# 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
    # https://keras.io/api/layers/recurrent_layers/gru/
    x = keras.layers.GRU(16, return_sequences=True)(frame_features_input, mask
    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=["ac
    )
    return rnn_model

EPOCHS = 30
# 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 23/30
1/1 [==============================] - ETA: 0s - loss: 0.6846 - accuracy:
0.7000
Epoch 23: val_loss did not improve from 0.69415
1/1 [==============================] - 0s 136ms/step - loss: 0.6846 - acc
uracy: 0.7000 - val_loss: 0.7163 - val_accuracy: 0.0000e+00
Epoch 24/30
1/1 [==============================] - ETA: 0s - loss: 0.6842 - accuracy:
0.7000
Epoch 24: val_loss did not improve from 0.69415
1/1 [==============================] - 0s 137ms/step - loss: 0.6842 - acc
uracy: 0.7000 - val_loss: 0.7173 - val_accuracy: 0.0000e+00
Epoch 25/30
1/1 [==============================] - ETA: 0s - loss: 0.6839 - accuracy:
0.7000
Epoch 25: val_loss did not improve from 0.69415
1/1 [==============================] - 0s 111ms/step - loss: 0.6839 - acc
uracy: 0.7000 - val_loss: 0.7183 - val_accuracy: 0.0000e+00
Epoch 26/30
1/1 [==============================] - ETA: 0s - loss: 0.6835 - accuracy:
```

Inference

In [32]:
```
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='

    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] * 100:5.6f}%")
    return frames

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

test_frames = sequence_prediction(test_video)
```

```
Test video path: dataset/test/appropriate/result_1.gif
1/1 [==============================] - 0s 60ms/step
  appropriate: 50.049996%
  inappropriate: 49.949998%
```

In [ ]:

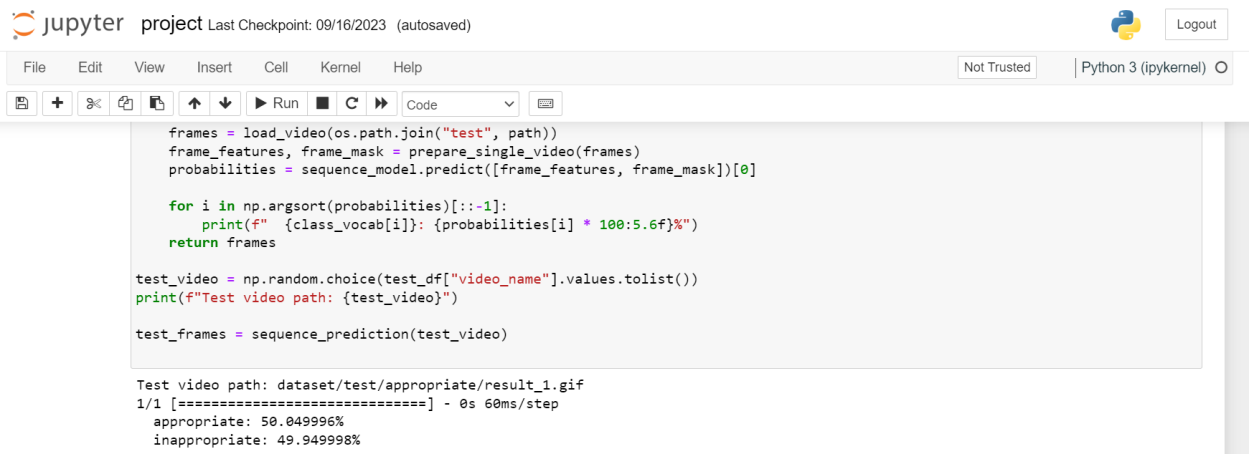
In [ ]:

In [ ]:

In [ ]:

In [ ]:

output:



random data was selected and the output was printed as probability of being appropriate or inappropriate

week 02 objective:

to create subdivisions in inappropriate dataset such as violence, drug abuse, nudity & add more datasets & improve the formula for more accuracy