video

May 10, 2021

1 CNN-LSTM Video Sentiment Analysis

This notebook will walk through the process of training a LSTM on top of a CNN.

This is an early version of the LSTM code, which was updated in server_code/classify_video.py, as it could run quicker there.

[ ]:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dropout, Flatten, Dense, Input,␣
 ↪AveragePooling2D, TimeDistributed, LSTM
from tensorflow.keras.models import Model, load_model, Sequential
from tensorflow.keras.optimizers import SGD, Adam
import tensorflow as tf
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import random
import matplotlib.pyplot as plt
import numpy as np
import argparse
import pickle
import cv2
import os
import utils.data
import utils.model
import gc
import pickle
import glob
import re
```

[ ]:

```
import wandb
from wandb.keras import WandbCallback
wandb.login()
```

Set the parameters for our video training procedure. The CNN model is later placed before a LSTM.

[ ]:

```
with open('fer_label_bin', 'rb') as file:
    lb = pickle.load(file)
dataset = "fer+"
cnn_model = "../models/best-models/resnet50-vgg.h5"
```

Load the data, and setup various functions which help us to use the data.

[ ]:

```
vidlist = glob.glob("../data/dataset/{}/*-30-720.mp4".format(dataset))

emotion_dict = {
        1: "neutral",
        2: "calm",
        3: "happy",
        4: "sad",
        5: "angry",
        6: "fearful",
        7: "disgust",
        8: "surprised"
    }

def get_frame_array(filename):
    frames = []

    vs = cv2.VideoCapture(filename)

    # Loop over video frames
    while True:
        (grabbed, frame) = vs.read()
        if not grabbed:
            break

        # convert to greyscale
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        frame = cv2.cvtColor(frame, cv2.COLOR_GRAY2RGB)

        frame = cv2.resize(frame, (197, 197)).astype("float32")
        frame -= 128.8006
        frame /= 64.6497
        frames.append(frame)

    return np.array(frames)

x = []
y = []
num_frames = 0
for i, video in enumerate(vidlist):
    emotion = emotion_dict[int(re.search('(?<=\/0)\d', video).group())]
    if emotion in lb.classes_:
        x.append(video)
        y.append(emotion)
        num_frames += len(get_frame_array(video))
print(num_frames)

y = lb.transform(y)

# split into test/train/validate
train_x , test_x, train_y, test_y = train_test_split(x, y, test_size=0.2,␣
 ↪random_state=42)
train_x , val_x, train_y, val_y = train_test_split(train_x, train_y,␣
 ↪test_size=0.25, random_state=42)

def train_gen(batch_size, num_frames):
    while True:
        vid_num = 0
        vid_index = 0
        frames = get_frame_array(train_x[vid_num])
        y = train_y[vid_num]

        frames_out = []
        classes_out = []

        for batch_num in range(batch_size):
            if vid_index + num_frames > len(frames):
                vid_num += 1
                vid_index = 0
                frames = get_frame_array(train_x[vid_num])
                y = train_y[vid_num]


            frames_out.append(frames[vid_index: vid_index + num_frames])
            classes_out.append(y)
            vid_index += 1

        yield(np.array(frames_out), np.array(classes_out))


def val_gen(batch_size, num_frames):
    while True:
        vid_num = 0
        vid_index = 0
        frames = get_frame_array(val_x[vid_num])
        y = val_y[vid_num]

        frames_out = []
        classes_out = []

        for batch_num in range(batch_size):
            if vid_index + num_frames > len(frames):
                vid_num += 1
                vid_index = 0
                frames = get_frame_array(val_x[vid_num])
                y = val_y[vid_num]


            frames_out.append(frames[vid_index: vid_index + num_frames])
            classes_out.append(y)
            vid_index += 1

        yield(np.array(frames_out), np.array(classes_out))

def test_gen(batch_size):
    assert batch_size == 1
    while True:
        for i, video_fn in enumerate(test_x):
            yield (get_frame_array(video_fn), test_y[i])
```

Load the CNN model which we have previously saved, and place a LSTM after it.

[ ]:

```
def get_model(num_frames):
    if cnn_model == "../models/best-models/resnet50-vgg.h5":
        base_model = load_model("../models/best-models/resnet50-vgg.h5")

        for layer in base_model.layers:
            layer.trainable = False

        # prune the top two layers from the model
        base_model._layers.pop()
        base_model._layers.pop()

        base_model = Model(inputs=base_model.input, outputs=base_model.
 ↪layers[-1].output)

        base_model = TimeDistributed(base_model, input_shape = [num_frames,␣
 ↪197, 197, 3]) # enable the CNN to be called for each frame

        model = Sequential([
            base_model,
            LSTM(128),
            Dense(64, activation='relu'),
            Dropout(0.5),
            Dense(32, activation='relu'),
            Dropout(0.5),
            Dense(len(lb.classes_))
        ])


    else:
        print("Model {} not supported yet, please implement that".
↪format(cnn_model))

    return model
```

Set out our training procedure, using stochastic gradient descent.

[ ]:

```
def train():
    # default hyperparameters
    config_defaults = {
        'batch_size' : 1,
        'epochs': 30,
        'num_frames' : 32
    }

    wandb.init(project='sentiment', config=config_defaults)
    config = wandb.config

    config.architecture_name = "{} followed by LSTM".format(cnn_model)
    config.dataset = dataset

    # Compile the model
    opt = SGD()
    model = get_model(config.num_frames)
    model.compile(loss="categorical_crossentropy", optimizer=opt,
        metrics=["accuracy"])

    # Now we can start training!
    history = model.fit(
        train_gen(config.batch_size, config.num_frames),
        steps_per_epoch = len(train_x) // config.batch_size,
        validation_data = val_gen(config.batch_size, config.num_frames),
        validation_steps = len(val_x) // config.batch_size,
        epochs = config.epochs,
        callbacks = [WandbCallback()]
    )

    return model
```

Next, we setup a sweep of hyperparameters.

[ ]:

```
sweep_config = {
    "method": "bayes",
    "metric": {
        "name": "val_loss",
        "goal": "minimize"
    },
    "parameters":{
        "epochs": {
            "distribution": "int_uniform",
            "min": 20,
            "max": 40
        },
        "batch_size": {
            "distribution": "int_uniform",
            "min": 30,
            "max": 64
        },
        "learning_rate": {
            "distribution": "uniform",
            "min": 0.00001,
            "max": 0.001
        },
        "momentum": {
            "distribution": "uniform",
            "min": 0.9,
            "max": 0.99
        },
        "decay": {
            "distribution": "uniform",
            "min": 1e-6,
            "max": 1e-2
        }
    },
    "early_terminate" :
    {
        "type": "hyperband",
        "min_iter": 3
    }
}
wandb.sweep(sweep_config, project='sentiment')
wandb.agent(sweep_id, project='sentiment', function=train)
```