Capstone Project

Live Class Monitoring System Face emotion Recognition

Subham Choudhary

Contents

- ❑ Introduction
- ❑ Problem Statement
- ❑ Data Processing
- ❑ Convolutional Neural Network
- ❑ Accessing webcam for model testing
- ❑ App development using streamlit
- ❑ Deploying app on cloud-based platform
- ❑ Challenges
- ❑ Conclusion

Introduction

Facial emotion recognition is the process of detecting human emotions from facial expressions. The human brain recognizes emotions automatically, and software has now been developed that can recognize emotions as well. This technology is becoming more accurate all the time, and will eventually be able to read emotions as well as our brains do.

Problem Statement

Digital classrooms are conducted via video telephony software program (ex: Zoom) where it’s not possible for medium scale class (25-50) to see all students and access the mood. Because of this drawback, students are not focusing on content due to lack of surveillance.

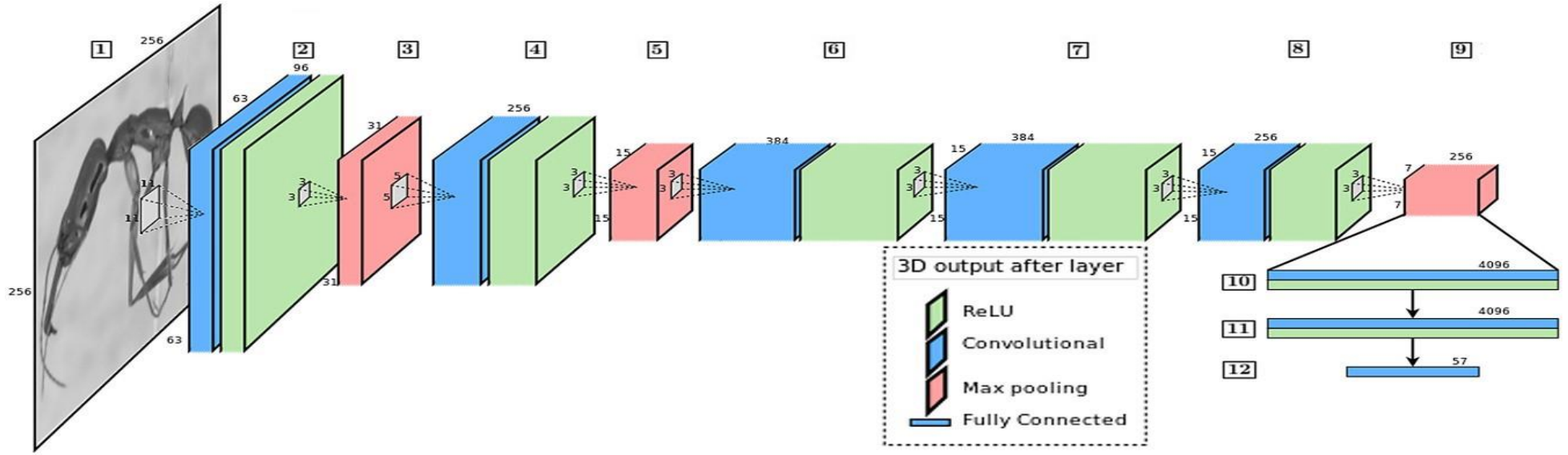
We will solve the above-mentioned challenge by applying deep learning algorithms to live video data. The solution to this problem is by recognizing facial emotions

Data Processing

- The images for the project is obtained from **https://www.kaggle.com/datasets/msambare/fer2013**
- The dataset consists of 35887 grayscale, 48x48 sized face images with seven emotions - angry, disgusted, fearful, happy, neutral, sad and surprised.
- The whole dataset is divided into train set and test set with 28779 images on train set and 7188 images on test set.
- Some image processing techniques were used on train set for image augmentation.

Convolutional Neural Network

A convolutional neural network is used to detect and classify objects in an image. In CNN-based approaches, the input image is convolved through a filter collection in the convolution layers to produce a feature map. Each feature map is then combined to fully connected networks, and the face expression is recognized as belonging to a particular class-based the output of the softmax algorithm.



```python
# create model structure
emotion_model = Sequential()

emotion_model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(48, 48, 1)))
emotion_model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Dropout(0.25))

emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Dropout(0.25))

emotion_model.add(Flatten())
emotion_model.add(Dense(1024, activation='relu'))
emotion_model.add(Dropout(0.5))
emotion_model.add(Dense(7, activation='softmax'))

emotion_model.compile(loss='categorical_crossentropy', optimizer=adam_v2.Adam(learning_rate=0.0001,
decay=1e-6), metrics=['accuracy'])

# Train the neural network/model
emotion_model_info = emotion_model.fit(
        train_generator,
        steps_per_epoch=28709 // 64,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=7178 // 64)
```

This step is the most important part of the entire process as we design the CNN through which we will pass our features to train the model and eventually test it using the test features. We have used a combination of several different functions to construct CNN which we will discuss one by one.

- **Sequential() -** A sequential model is just a linear stack of layers which is putting layers on top of each other as we progress from the input layer to the output layer.

- **model.add(Conv2D()) -** This is a 2D Convolutional layer which performs the convolution operation. This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. Here, we are using a 3x3 kernel size and Rectified Linear Unit (ReLU) as our activation function.

- **model.add(MaxPooling2D()) -** This function performs the pooling operation on the data, which takes the maximum value in each window which decreases the feature map size while keeping the significant information.. We are taking a pooling window of 2x2 with 2x2 strides in this model.

- **model.add(Dropout()) -** Dropout is a technique where randomly selected neurons are ignored during the training. They are “dropped out” randomly and i.e., 25% of the nodes dropped at each layer. This reduces overfitting.

- **model.add(Flatten()) -** This just flattens the input from ND to 1D and does not affect the batch size.

- **model.add(Dense()) -** According to Keras Documentation, Dense implements the operation: output = activation(dot(input, kernel)) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer. In simple words, it is the final nail in the coffin which uses the features learned using the layers and maps it to the label. During testing, this layer is responsible for creating the final label for the image being processed.

- Next, the model is compiled with categorical_crossentropy as the loss function and using Adam optimizer. We are using accuracy as the metrics for validation. Next, we are fitting the model with the fixed batch size (64 here), epochs (50 here) and validation data which we obtained from the local drive. Finally, the model is saved in json for some custom tests and the learned weights are stored in h5 file.

Accessing webcam for model testing

- Opencv library is used for real time face emotion detection
- An emotion dictionary is created as {0: "Angry", 1: "Disgusted", 2: "Fearful", 3: "Happy", 4: "Neutral", 5: "Sad", 6: "Surprised"} because while mapping the output, the emotions can be labelled accordingly.
- For detecting the face in the frame harcascade_frontalface_default.xml file is downloaded. Haar Cascade frontal face XML is an Object Detection Algorithm used to identify faces in an image or a real time video.
- Faces detected using Cascade classifier and resized to 48x48. Then the image is sent to model to detect the emotion of the images.

App development using streamlit

- Two important library for this step: - streamlit and streamlit-webrtc.
- Streamlit is one of the most powerful way to build data apps, including the ability to display and style data, draw charts and maps, add interactive widgets, customize app layouts, cache computation, and define themes.
- WebRTC extends Streamlit’s powerful capabilities to transmit video, audio, and arbitrary data streams between frontend and backend processes, like browser JavaScript and server-side Python.
- Next steps are very similar to the testing of model. With the help of Video Transformer, we can stream the video using streamlit application.

Deploying app on cloud-based platform

Amazon web services(AWS)

- EC2 instance created and a public key generated.
- Private key generated using Puttygen.
- Winscp was used to connect local drive with the server.
- Putty was used to install all dependencies and run the app.

Streamlit-cloud

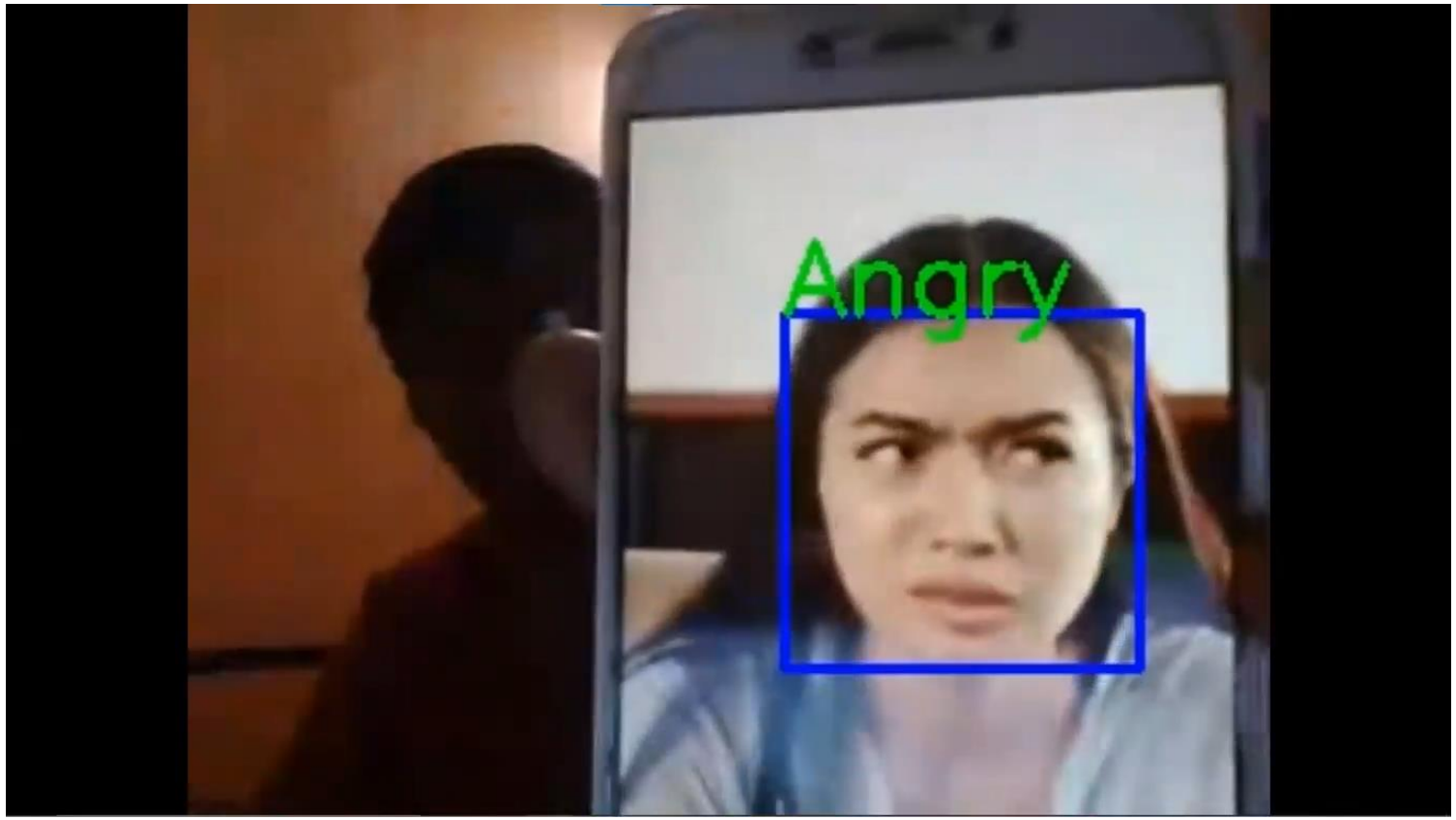
- Streamlit account is connected with the Github account.
- Github repository imports all necessary files.
- Installation of dependencies and app starts working.

Challenges

- Due to lack of gpu, the model training took more than 2 hours.
- Installation of tensorflow library caused some errors during streamlit app development, instead tensorflow-cpu works for us.
- Opencv doesn’t work in our case, we installed opencv-contrib-headless worked for us.
- Video Transformer is deprecated, but while using video processor the video streaming freezes.
- During app deployment on AWS, the installation of streamlit-webrtc caused error. Unable to rectify the error, we left the idea of deploying on AWS.
- The app deployed perfectly on streamlit cloud but the video streaming didn’t start and the app starts loading and stops after some time.

Conclusion

The application worked perfectly on local host.



Thank You