Mobile & Pervasive Computing

Group Members
Mayank Singh Rajput (B19CSE054)
Mohit Ahirwar (B19CSE055)

Project - EmoChat

Instructor - Dr. Suchetana Chakraborty

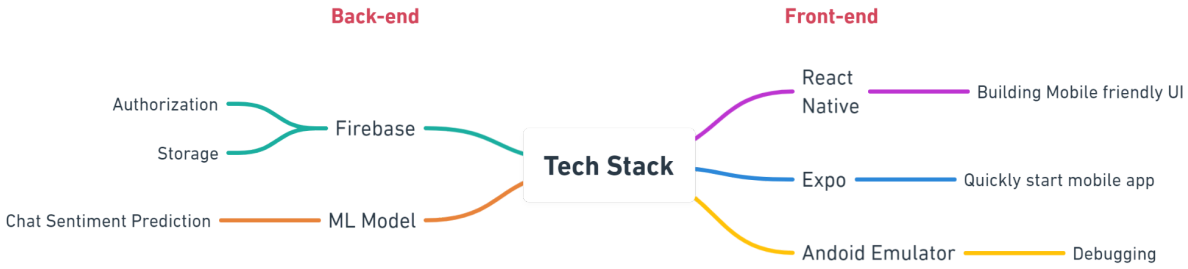
Introduction

EmoChat is a mobile application that has the ability to show live chat sentiments (emotions). Every time when a user sends or receives a message, the sentiment that is reflected by that particular message will be displayed next to that message in real-time.

The sentiment part of the chat message has two parts:

- **Sentiment score** - A value between -10 to 10, representing the polarity of the sentiments. 10 reflects the most positive sentiment whereas -10 is indicated the most negative sentiment.
- **An emoji** - An emoji depending on the sentiment value is also displayed, which essentially represents the possible mood of the sender while we receive their message.

Tech Stack



The tech stacks used here are:

Front-end

React Native

React Native is an open-source framework that enables cross-platform mobile application building using JavaScript and reacts which is an open-source JavaScript library.

React Native uses JSX to create native apps for Android as well as iOS. It is a powerful tool that aids in creating apps with beautiful UIs, which are the same as native apps. It makes mobile app creation very easy and efficient, that is why we chose this front-end framework for my app.

Expo

Expo is a React Native app development framework. It's a collection of React Native-specific tools and services. It will make it simple for us to start creating React Native apps.

It gives us a list of tools to make creating and testing React Native apps easier. Aside from that, Expo offers a more robust and simple development approach that is also more flexible.

Android Emulator

It is a tool that we used to see the app in real-time in order to debug it and add elements to it. Android emulator can be found in Android Studio.

Although we also used my physical mobile phone to run and see the application apart from the emulator.

Back-end

Firebase

Firebase is a Google-backed mobile and web app development platform based on the backend as a service (BaaS) system and consists of several pragmatic services and purposeful APIs to develop high-quality applications.

Firebase provides swift and secure hosting for applications. With trusted Google authentication and stability, It was easy for me to choose firebase for **data storage** (for chats and images) and **authentication**.'

Firebase Database Link: ....

ML Model

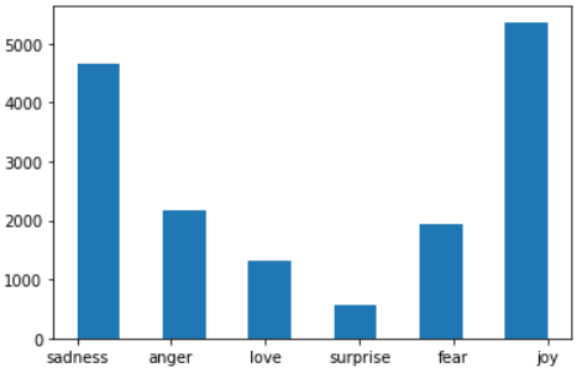
In order to predict live sentiments in chat messages, I prepared an ML model.

I started with a machine learning model but they were not giving that much accuracy (around 50%), so I used **Deep learning** for sentiment prediction using NLP techniques. The deep learning model was giving an accuracy of over **90%**.

Sentiment Categories

The sentiment analysis is divided into 6 categories - sadness, anger, love, surprise, fear and joy. It means that whenever any word or sentence will be put in this model, it will categorize it in any one of the above mentioned categories.

```
plt.hist(labels,bins =11)
plt.show()
```



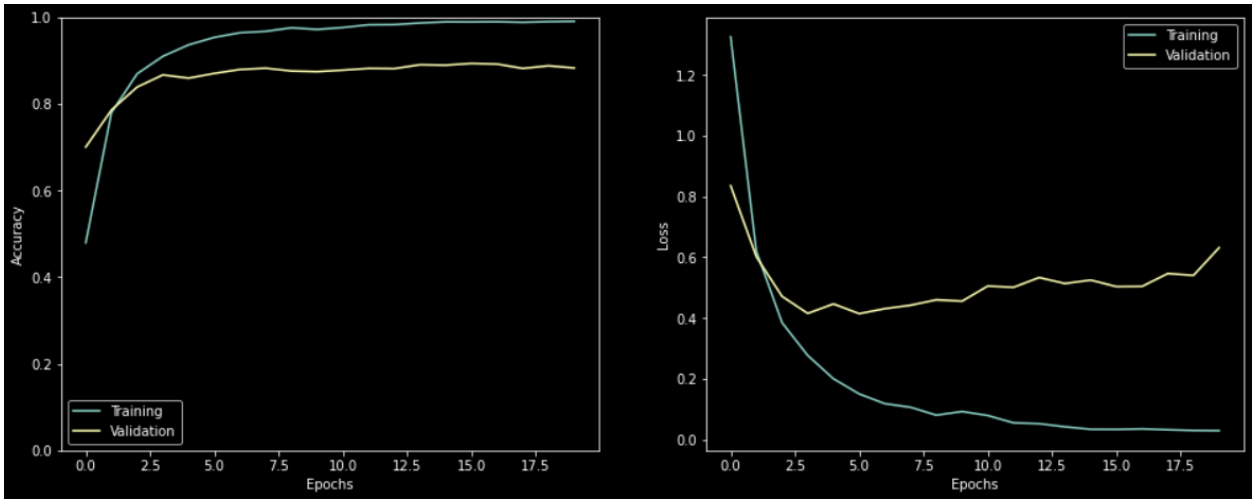

```
class_to_index
```

```
{'anger': 0, 'fear': 3, 'joy': 4, 'love': 2, 'sadness': 5, 'surprise': 1}
```

Accuracy vs Epochs and Loss vs Epochs graph

The graph of epochs vs training and validation accuracy is present on the left and the graph on the right is between epoch vs training and validation loss.

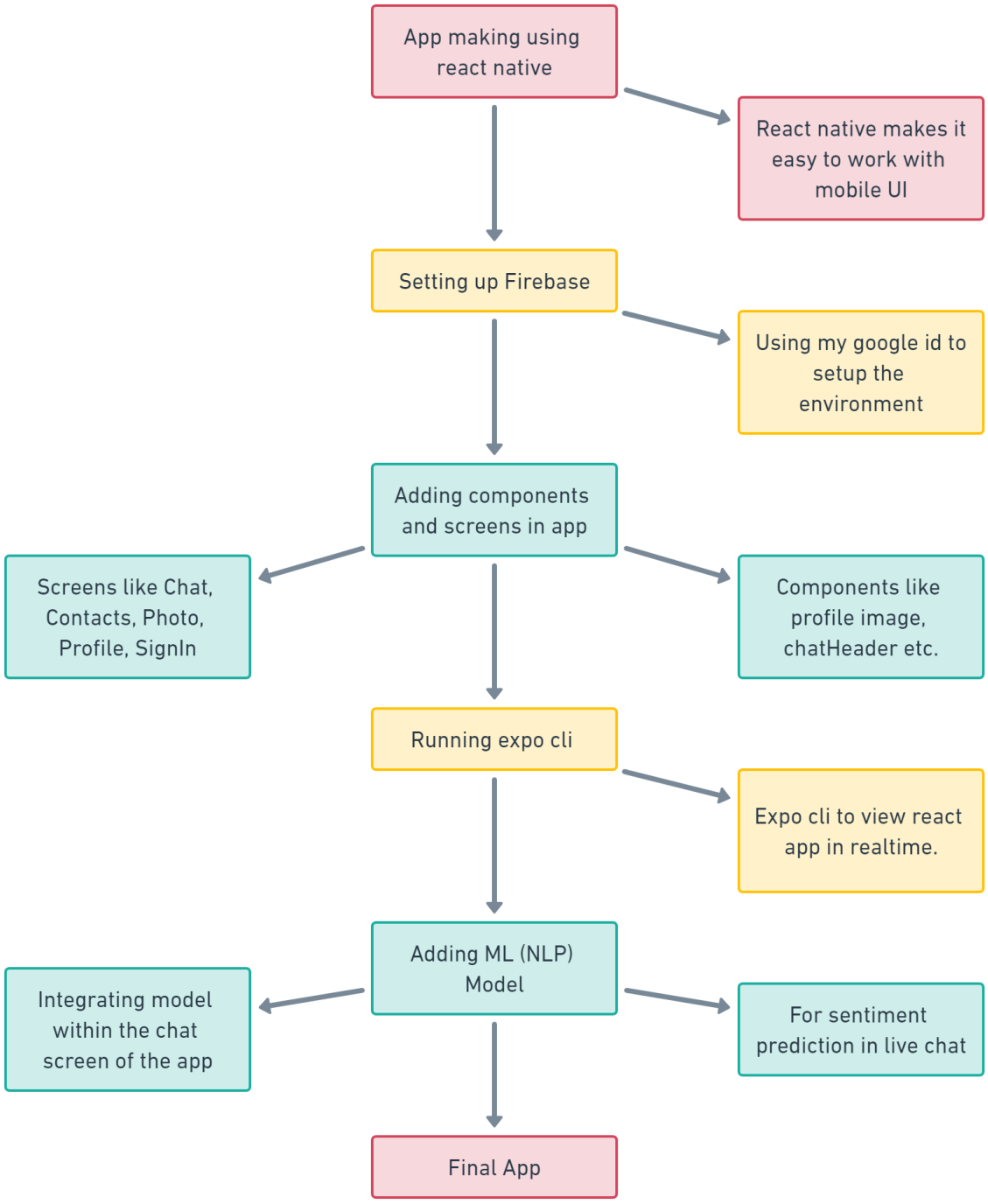
The number of epochs used was 20.



Final Accuracy - around 97%

Google Colab Link: ....

WorkFlow


App making using react native
React native makes it easy to work with mobile UI
Setting up Firebase
Using my google id to setup the environment
Adding components and screens in app
Screens like Chat, Contacts, Photo, Profile, SignIn
Components like profile image, chatHeader etc.
Running expo cli
Expo cli to view react app in realtime.
Adding ML (NLP) Model
Integrating model within the chat screen of the app
For sentiment prediction in live chat
Final App

App creation

The app was created using React native. For this, we have to set up node modules and other files using `npm install -g create-react-native-app.`

Initiated work on the app till the login page before setting up firebase for authentication.

Setting up Firebase

The next step was to set up authentication for the app in firebase, in order to make the app login and signup process hassle-free.

Adding components and Screens

Then We moved on to adding various components and screens to the app.

Components include:

- Avatar.js
- ChatHeader.js
- ContactsFloatingIcon
- ListItem

Screens include:

- Chat.js
    - Individual chat screen for a separate room. Room (in the database) signifies a one-to-one conversation between two users.
- Chats.js
    - This screen is basically the collection of rooms as it shows all the users who you chatted with recently.
- Contacts.js
    - This screen shows all the lists of contacts that have been registered in the app's database and are present in our contact list.
- Photo.js
    - This screen comes up every time we upload or send a photo within the app.
- Profile.js

    - This screen is the next screen after Signup/Login. It requires a name and an optional profile picture when the user Signs in for the first time.
- SignIn.js
    - This is the first screen with which a new user will interact.
    - Of course, an already existing user won't meet this screen as the app will automatically recognize him/her as an existing user.

Running expo cli

Expo cli is used hand-in-hand with react native in order to see the app live for debugging.

For this, we have to first run the react native app using `npm start` which in turn runs the expo cli interface having a QR code and all the links to run the app on various screens and Operating systems.

Adding ML (NLP) Model

Until now, We have pretty much wrapped up the app UI part. Now We need to add the NLP layer over live chats to show relevant **sentiment score** and **mood emoji** with the msgs themself.

The first step is to train the model. For this - these steps must be followed:

- We need a dataset, fortunately, an emotion dataset is available in the nlp library.
    - ```
      dataset =  nlp.load_dataset('emotion')
      ```
- For feeding the model with the dataset, We processed it using Tokenization, word to sequence conversion etc.

  ```
  tokenizer = Tokenizer(num_words=10000, oov_token ='<UNK>')
  tokenizer.fit_on_texts(tweets)
  ```

    - ```
      tokenizer.texts_to_sequences([tweets[0]])
      ```
- After tokenization, padding should be done which ensures the sentence length of the same size for feeding the model.

```
maxlen = 50
from tensorflow.keras.preprocessing.sequence import pad_sequences
def get_sequences(tokenizer,tweets):
  sequence = tokenizer.texts_to_sequences(tweets)
  padded = pad_sequences(sequence, truncating = 'post', padding = 'post', maxlen = maxlen)
  return padded
padded_train_seq = get_sequences(tokenizer,tweets)
```

- The main step comes to actually running the model training. For this We found optimum accuracy to be coming at 20 epochs.

```
h = model.fit(
    padded_train_seq, train_labels,
    validation_data = (val_seq,val_labels),
    epochs = 20,
)
```

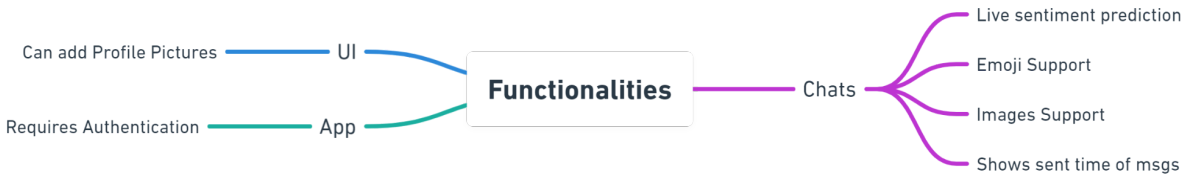
```
Epoch 1/20
500/500 [==============================] - 33s 49ms/step - loss: 1.3253 - accuracy: 0.4790 - val_loss: 0.8356 - val_accuracy: 0.7000
Epoch 2/20
500/500 [==============================] - 22s 44ms/step - loss: 0.6200 - accuracy: 0.7796 - val_loss: 0.6015 - val_accuracy: 0.7855
Epoch 3/20
500/500 [==============================] - 22s 45ms/step - loss: 0.3848 - accuracy: 0.8691 - val_loss: 0.4712 - val_accuracy: 0.8385
Epoch 4/20
500/500 [==============================] - 25s 49ms/step - loss: 0.2766 - accuracy: 0.9095 - val_loss: 0.4150 - val_accuracy: 0.8665
Epoch 5/20
500/500 [==============================] - 23s 46ms/step - loss: 0.2000 - accuracy: 0.9359 - val_loss: 0.4464 - val_accuracy: 0.8590
Epoch 6/20
500/500 [==============================] - 23s 46ms/step - loss: 0.1502 - accuracy: 0.9531 - val_loss: 0.4143 - val_accuracy: 0.8700
Epoch 7/20
500/500 [==============================] - 23s 45ms/step - loss: 0.1181 - accuracy: 0.9638 - val_loss: 0.4307 - val_accuracy: 0.8790
Epoch 8/20
500/500 [==============================] - 23s 46ms/step - loss: 0.1062 - accuracy: 0.9671 - val_loss: 0.4420 - val_accuracy: 0.8820
Epoch 9/20
500/500 [==============================] - 22s 45ms/step - loss: 0.0802 - accuracy: 0.9753 - val_loss: 0.4598 - val_accuracy: 0.8755
Epoch 10/20
500/500 [==============================] - 27s 54ms/step - loss: 0.0923 - accuracy: 0.9715 - val_loss: 0.4553 - val_accuracy: 0.8740
Epoch 11/20
500/500 [==============================] - 22s 45ms/step - loss: 0.0792 - accuracy: 0.9759 - val_loss: 0.5051 - val_accuracy: 0.8775
```

After training, we saved the model using inbuilt python methods and then integrated it with my native app UI.

Final App

After completing every subpart in the workflow, the final App is ready for action!

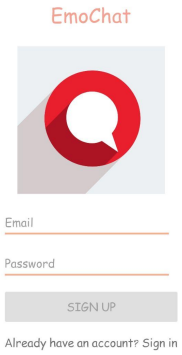
Functionalities



App

The application requires mandatory user registration in order to access it. If a user is visiting the app for the first time, he/she needs to register himself/herself on the app and can use that id and password for logging in the next time.

This functionality is implemented using firebase, which verifies the entered user id and password with its database. If it is matched with an entry in the database, then it allows the user to access the application.



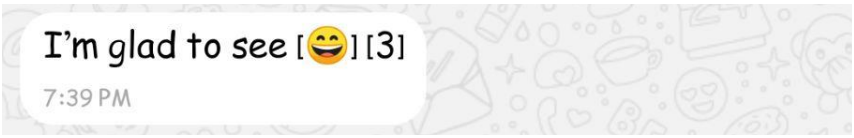
UI

In the Application UI, users can also add their profile pictures in order to make it more intuitive and not that simple. It also makes it easy to identify a chat without having to see the name.

Chats

Live Sentiment Prediction

It is the most unique feature of this application. We have used chatting apps like WhatsApp, Facebook, Instagram, and whatnot. But my application can show in real-time the sentiment which is reflected by each message. It makes it very easy **not-to-misinterpret** the sense of the message and the user can then reply accordingly.



Emoji Support

This app also supports the use of emojis. Some apps show emojis as garbage texts but this app shows them as relevant emojis.

Images Support

Apart from text, users can also send and receive images to each other in the chats. Note that images don't play any role in sentiment prediction.

To send an image, the user just needs to tap the camera button which is present before the send button in the chat UI.

Time Stamp

The messages which are sent and received in the chat UI also carry their time of sending along with them, which makes the chat more informative.

Challenges

Integrating ML Model with React native

Integrating the ML NLP Model was a challenge as it was not easy to create and manipulate the API of the ML model.

We had to first save the trained model and then Its API can be used in the parsing of chat messages.

To apply the model, We took the last message from the database and passed it to the ML model, which gave a value (a number) as an output. We then used that number to show relevant emojis along with the sentiment value using if-else statements.

Adding Time Stamp

To add this, We have to grab the time of sending the message from the database using **getTime()** function and then manipulate the received value to show just below the sent and received message.

Future Work

Weekly/Monthly Mood Indicator

We can use the collected chat data for a week or month to make an overall sentiment analysis of the user. This might help in detecting **psychological stress** he/she might be going through.

- For this, we need to add an export data button within the app to export those chats and then this data can be analyzed by our ml model.
- We can also show the results within the app in order to make this feature more usable.

More Functionalities within the app

More functionalities like

- ability to delete chats,
- read receipts,
- voice message support

References

[React Native · Learn once, write anywhere]

[Expo CLI - Expo Documentation - Expo Documentation]

[How to connect ML model which is made in python to react native app - Stack Overflow]

[Deploy Machine Learning Model using Flask - GeeksforGeeks]