HAN_UNIVERSITY
OF APPLIED SCIENCES

Emotion estimation from video footage with LSTM

By: Samer Attrah

Email: s.attrah@student.han.nl

Supervised by:

Dixon Devasia

Submitted to:

Jeroen Veen

Table of Contents

1. Introduction

This chapter includes a general view questions and queries to be answered in the following chapters of the report

1.1. Background

Emotion estimation in general is a field that has been studied for a long time and a number of approaches exist using machine learning.

1.1.1. Breakdown

fig (1) Emotion estimation process

There are many steps for the computer vision process to go through to reach a result of emotion estimation for a human face, and they may be listed as the following:

1. Face detection: at this step the machine learning algorithm needs to find the human face in the image being processed, for example in which corner of the image and draw a border that includes the face and excludes everything else.
2. Landmark detection: after detecting the face in the image, further processing gets done in this (the face) part only, and that includes finding landmarks on the face which is usually a fixed number of points get detected on the face, and by measuring the depth and position of these points in relation to the image and to each other, further information gets learned about the face that has been detected. In some cases, such as the report in hand another type of processing gets carried on to find a smaller number of landmarks that has deeper meaning and contains more information, in our case it is the Blendshapes that will be discussed later in further detail.
3. Expression classification: after finding the landmarks and having an abstract understanding of how the face looks like, a machine learning algorithm processes this information to be able to derive

meaning from it, and for this project it is estimating the emotion of the person.

This project has two distinguished processes could be described as follows:

- Training process: which consists of the following steps,
    - Choosing and collecting a dataset
    - Cleaning and pre-processing the dataset
    - Creating a dataset consists of the blendshapes of the images in the dataset
    - Create a deep neural network model.
    - Search for the best architecture and tune the hyperparameters, to get the best performance possible.
- Prediction process: this process takes place after the training process on the test set and after integrating the model into the full project, and it goes through the following steps:
    - Loading the trained model.
    - Loading the test set (in the case of testing only).
    - Rearranging the blendshapes in a list (for predicting from camera feed only).
    - Reshaping the blendshapes.
    - Predicting the class.

1.1.2. Use case

In this project will be building an emotion estimation algorithm to classify the emotion of the face being detected whether it was happy or sad, and later for it to be integrated to an interface that will be receiving a camera feed and providing other insights. About the detected face such as the eyes gaze direction and the face orientation.

And this project to be integrated later into a robot for social applications.

1.1.3. Tools

Some of the tools that has been used to work on this project are the following:

- Google colab free subscription
- Laptop acer Aspire 7: memory 16GB, processor 12th Gen Intel® Core™ i7-1260P × 16, Graphics NVIDIA Corporation GA107M [GeForce RTX 3050 Mobile].
- Codespace: 2 cores, 8GB RAM, 32 GB storage.

- Programming language: Python
- Libraries and frameworks: mediapipe, numpy, matplotlib, opencv, pathlib, csv, tensorflow, keras tuner, math, time, os.

1.2. Problem definition

In social robotics there are many ways for a robot to sense, understand and interact with the environment and its surroundings, in order to give the expected and correct reaction that is required from it.

In this project a part of a feedback mechanism is developed, to enable a robot to understand the emotion of the person it is looking at, talking to or interacting with.

1.3. Project objective

The task to be addressed is building a machine learning model that accurately classify the landmarks detected from an image whether it is for a happy face or a sad face, and integrating the model to a project that reads a camera stream and process it, with a user interface designed for it.

1.4. Research question

The research question could be phrased as the following:

“Using the available resources what would be the best landmarks emotion classification model be developed?”

From this question could derive for clarification, a few other questions:

- *What would be the best dataset and data pre-processing techniques to be used?*
- *What is the type of landmarks that will be produced and used for the task?*
- *What is the model type that will best fit the task?*

2. Literature survey|

For the task required in this project with the limited resources available, there wasn't a need for a wide literature survey, because the path major decisions to make were almost clear and no further insight was needed.

although it was the first time working with MediaPipe and Landmark vectors so a deeper understanding for the task and the tools are about to be used was required.

2.1. MediaPipe

A suite or a group of libraries [1] built by Google that enable the developers to quickly apply artificial intelligence and machine learning techniques in different applications utilizing GPU to accelerate the processing, to be able to focus on model development [2] and use MediaPipe as an environment.

Many usage examples are possible, some of them are:

1. Object detection: where it enables the developer to use real-time object detection from a live camera feed, which includes two functionalities that is tracking and detection, running in parallel. [2]
2. Hand Landmarks Detection: which enables the user to detect hands in images videos or camera streams and draw landmarks on them to detect the gesture and the expression being communicated through the hand.
3. Face landmark detection: where it includes two tasks that are face landmark detection and portrait segmentation, both running synchronously, and further description and details to be discussed later. The following image shows a detected face with MediaPipe

fig (2) Image with a detected face landmarks

4. Pose landmark detection: which enables the user to detect human body landmarks in an image or video and use it to identify key body locations, analyze posture and categorize movements.

And many other use case for MediaPipe all can be found in [1], such as image segmentation, and image embedding.

2.2. Landmark vectors

To understand the face expressions using computer vision, one way is to draw points on the face such as on the tip of the nose, eyebrows, mouth corners, eye corners and possibly many others, and from the positions and movement of these points can understand the facial muscle landmarks and can describe the emotion of the person. [3]

This process is complicated because of the different variability of human facial presence such as pose, position and orientation, also the frontal face objects such as glasses, hair style and beard play a big role in the accuracy of face detection, emotion estimation and expression understanding.

In many approaches of landmark detection for facial recognition the process starts by detecting the tip of the nose tip then segmenting it by cropping the sphere centered at this tip. And then the other points get found on the segmented sphere.

For this project the landmark vector detection process is done by MediaPipe so fine details will not be shown in the programming part.

2.3. Blendshapes

One type of landmark vectors that get generated by MediaPipe is called blendshapes, which is a semantic parameterization method originally was introduced in the computer graphics industry, a single one of them represent an individual facial expression and the full set of them generates a facial pose as a linear combination of a number of facial expressions [5], describe the different parts of the face by assigning weights for every expression bounded by (0,1) [4]. Some of the blendshapes are the following:

- Brow down
- Brow inner
- Eye blink
- Jaw open

- Mouth frown
- Nose sneer

And many others with a total of 52 and according to the Apple ARKit model [12], that get generated for each image or every frame in a video stream.

A histogram for the blendshapes in the image in fig (2) is the following:

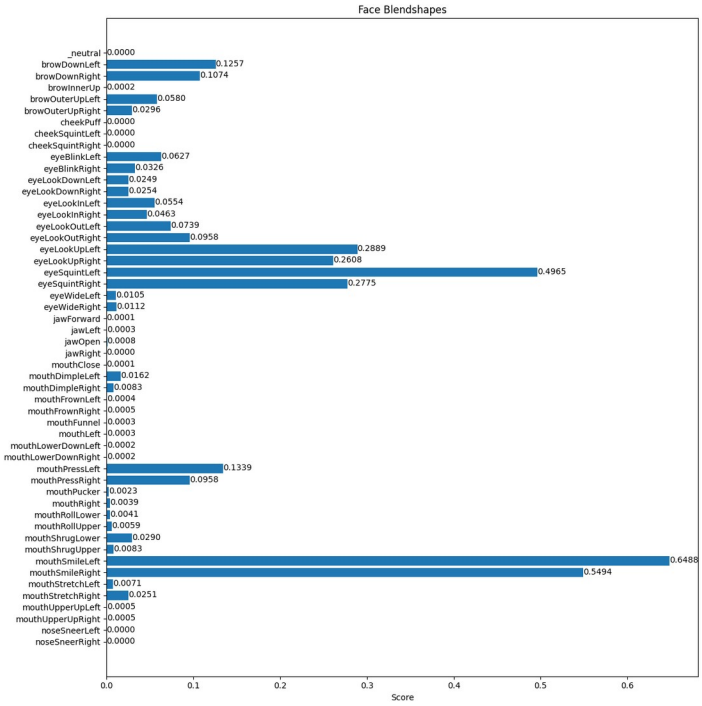

fig (3) blendshapes histogram

This type of vectors is often used in computer graphics where it helps to describe the movie character or video game character, and more or less blendshapes could be designed according to the specific need up to for example 675 blendshape for a face model that takes such as a year to be developed and well describe a complete range of expressions [5]

2.4. Facial emotion recognition

Reviewing literature for facial emotion recognition gives the result that the application being worked on is rarely touched on before, since the common approaches for facial emotion recognition are by classifying images of a dataset, which is different from from detecting facial features in a video stream and then finding the emotion that is being expressed using the feature vector classification.[3][6]

3. Datasets

While surveying the literature for information and understanding for the problem in hand, also there was a range of datasets and image databases that could be found and has been used for this application.

Although the final goal of the model being built is to estimate emotions of faces being detected on camera stream, the dataset to be used is an image dataset.

Since as mentioned in the literature survey, MediaPipe will process the camera feed and give the output of each frame separately, just like the outputs given for the images, it will be possible to train the model on images and use it to predict from video processed by MediaPipe.

From [6] can find a list of datasets for facial emotion recognition, some of them require a set of certain skills to have access to or others are expensive to buy with available resources, but others are available online such as FER2013 [7] which is a dataset for challenge on Kaggle and could be found in other accounts on the websites too.

This dataset is saved in the form of an excel sheet with the training instances as rows and for each instance there are three columns emotion, pixels, usage, where the emotion column is for the label of the image, the pixels is for the values of the pixels of the image in greyscale mode, and the usage is for the split of the datasets for training, public test, and private test, which got used later for training, validation and test, respectively.

Since the model is being built to distinguish the happy and sad faces, the data processing started by counting the images of each category for the full count training split, validation split, and test split:

Happy = 8989, 7215, 895, 879

Sad = 6077, 4830, 653, 594

Angry = 4953, 3995, 467, 491

Afraid = 5121, 4097, 496, 528

Surprise = 4002, 3171, 415, 416

Disgust = 547, 436, 56, 55

Neutral = 6198, 4965, 607, 626

And for the first model to be built only the happy and sad images been used and that resulted in having 4000 images for training from each emotion, to keep the model from being biased, by training on unequal counts.

After a few experiments and visualizations besides considering the quality, size and counts of the dataset images the search for a better dataset started and the best ones to look for then was the MultiPIE [8] dataset which was inaccessible and expensive to buy.

Another was Radboud University faces database [9], also AffectNet [10] and both requires research work by the student to be granted access to after they got requested by a professor at the university.

So, the work will be done using the FER2013 dataset.

4. Data processing and cleaning

This chapter will discuss the different techniques used to process the dataset before using it to train and test the model.

4.1. Creating data splits

The first step is to split the dataset into three files that are training, validation, and test, for the last version of the model that got trained on three classes dataset some further processing in the training dataset had to take place in order to have three classes that includes three emotions that are happy sad and unknown emotion.

The first step was to create a Python array from all the training instances, then from each class except happy and sad, include 1500 images except for the disgust class since the available images are around 400 only, and that is to balance the class counts and avoid bias in the training process, and as before each of the happy and sad classes are equal to 4000 training instance which is the maximum available count.

Then the next step is to relabel the classes by assigning 0 to the happiness images, 1 to the unknown images and sadness images takes the number 2 as a label.

Then create a csv file with the training set array and two others for the arrays of the validation and test which includes all the instances available for them in the original dataset.

4.2. Readability by MediaPipe

Since the model is being built to understand the blendshapes produced by MediaPipe after processing a video stream, then including images that are not understandable by MediaPipe intuitively will not be useful in the training process besides that will produce errors.

To check for that, created a Pipeline that will organize the list of pixels of the image in the csv dataset and create an image from it with the shape (48, 48), and convert it from greyscale to RGB, then format it into SRGB to be understandable by MediaPipe, then run it through MediaPipe detector and check if there was an input or it will return an empty list indicating that it was not able to detect the face and calculate blendshapes for it.

The following images shows examples for unreadable images:

fig (4) images unreadable by MediaPipe

The numbers of images from each class in the training set that was not readable by MediaPipe were:

Angry = 597

Disgust = 72

Afraid = 616

Happy = 377

Sad = 835

Surprise = 239

Neutral = 261

4.3. Choosing the relevant Blendshapes

For every image processed by MediaPipe a list of 52 blendshapes get produced, and since the dataset being used is for emotion recognition and includes 7 emotions then possibly not all the Blendshapes are relevant.

To check for the most relevant blendshapes used a threshold of (0.4) been chosen to pass the Blendshapes values on and count for how many images this threshold get passed by a blendshape, to include this blendshapes if it pass in more than 100 and exclude it if it does not meet the relevancy criteria. Then create a python list of the indices of the relevant blendshapes.

The following figure shows scatter plots for the 52 blend shapes only for the first two thousand instances of the training dataset to give a better idea of how some of the blendshapes are not relevant

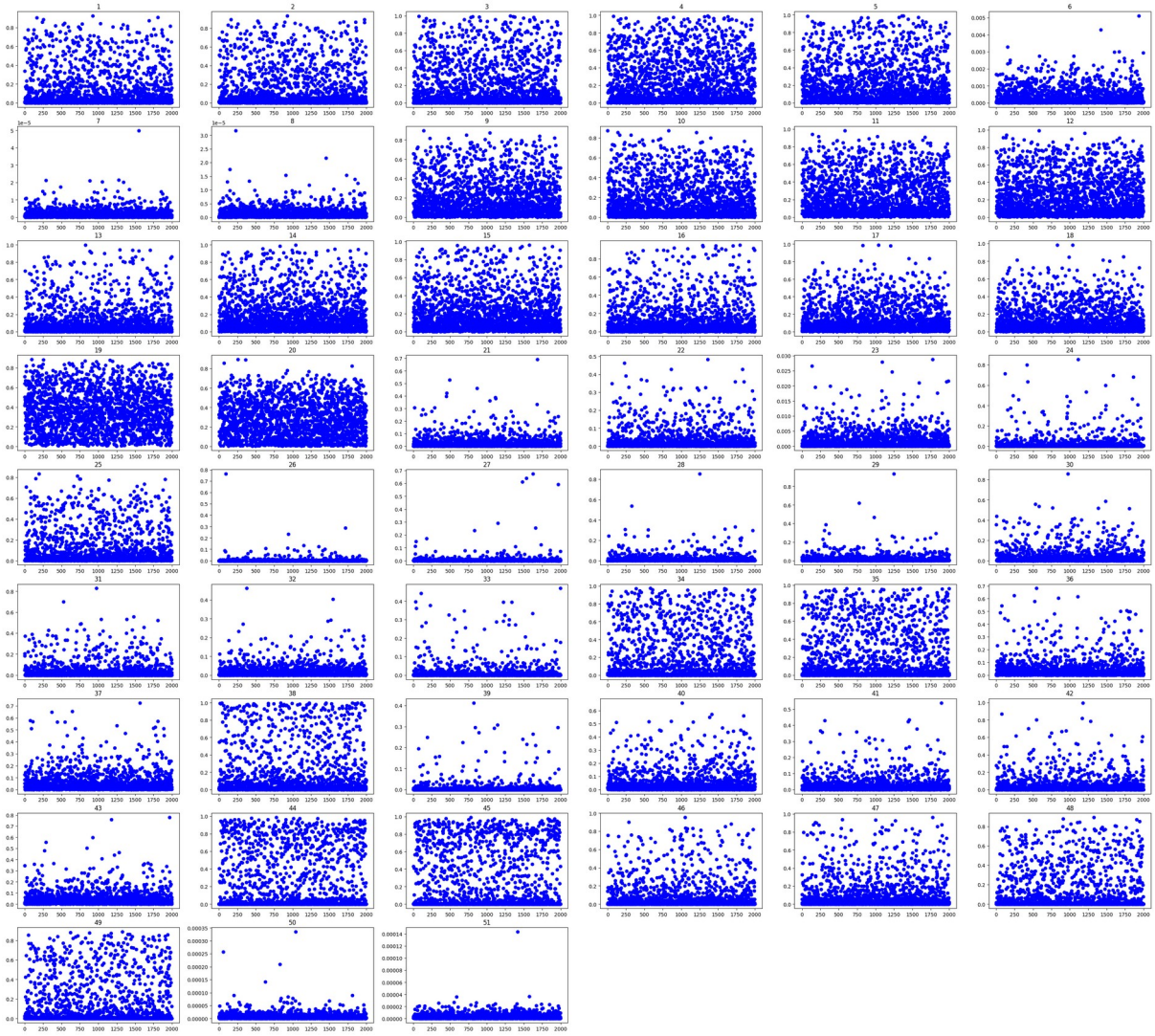

fig (5) scatter plot of the blendshapes in the training set

This figure show for example the blendshapes 6, 7, 8 are all close to the bottom which is zero in value so including them in the training will not add detail to the feature set but will be consuming in time and computation resources, although the final model reached did not train on a dataset has this thresholding process applied to because of the time cap for the project was reached so further processing was not feasible.

4.4. Indexing the test set

As an error analysis technique created an index for the test set images to be able to track the image after it get transferred into its Blendshapes form in the later steps.

And be able to diagnose the images that will be causing the misclassification in the evaluation and prediction steps for the trained model, and find if there was any pattern between them.

4.5. Augmenting the training set

To improve the training set and increase the number of training instances, used some augmentation techniques, augmented the training set randomly and then added the augmented images with the non-augmented ones. And the expected result from that is less variance.

Augmentations used are,

- Random horizontal flip: since it is likely to happen in reality, and the images will stay real and relevant for the camera application after being horizontally flipped.
- Random rotation by 0.2 Pi CW and CCW: which could be an improvement for the model when predicting, in case the person in front of the camera is leaning on one side instead of sitting or standing straight.

Since the next step is to run the images through MediaPipe and get there blendshapes then the image format needs to be understandable by MediaPipe, and to achieve that a few more transformation steps need to be done, and as follows:

- The training instance needs to be rescaled to (1/255), Since augmentation layer does not process the images without rescaling them.
- Apply augmentations, flipping and rotations.
- Rescale the image by (1*255), to be understandable by MediaPipe.
- Cast the image pixels type to integers.
- Convert it to a numpy array.

And after this it would be augmented and understandable by MediaPipe. And they get processed to become the same form as the instances in the dataset spreadsheet file which consists of three columns a label, image pixels, and usage.

Examples for augmented images are as follows:

fig (6) augmented images

After augmenting the training set images and adding them to the set the full count of the set sums to more than 20,000 images.

4.6. Blendshapes dataset

To train the model to be able to predict the person emotions after the face of the person get detected by MediaPipe, this prediction needs to be done on the blendshapes of the face that gets produced by MediaPipe and not the original image, the model needs to be trained on a blendshapes dataset, and for that all the images needs to be processed using MediaPipe and their blendshapes found.
To do that, the images of the three datasets training, validation and testing must get reshaped and formatted in a way understandable by MediaPipe and processed by it and an array gets created for each image with the relevant blendshapes only.

Then a new spreadsheet file gets written from these blendshapes arrays.

The values of the blendshapes are in the range (0, 1) so considered normalizing the data but that will not give an improvement, given the small range the features are already in.

4.7. Classes dataset splits

To have better insight into the dataset and the features of each class and which one is the most relevant to the class, created another split for the dataset by clustering the blendshapes that belongs to one class into a separate file, for them to be used later in visualizations and error analysis.

4.8. labels One-hot encoding

For the data and the task, the model is being built for a certain loss function need to be used and later will mention this in further details, but for now to be able to use this loss function the labels of the dataset must get encoded as a one-hot encoding.

One-hot encoding means creating an array with rows equal to the number of training instances and columns equal to the number of classes, and for the first class insert one in the first column and zeros in the two other columns, and for the second class insert 1 in the second column and zeros in the other two columns, and so on for the third class.

Class	On-hot encoding
1	100
2	010
3	001

5. Model building

As a first step to build a model that is fit for the application, need to choose the type of the network that will be used, and following the two choices that has been made for the models:

5.1. LSTM Model

The first choice to be worked on is a fully LSTM model, and that choice was made for the following:

- The model being built is to be integrated to a video stream and classify the faces in it, which means it is a time series data application, so using one type of recurrent neural network is intuitive. Because it considers the past states besides the current input.
- Another reason for this choice is that LSTM is a type of gated recurrent unit (GRU), which means one of its advantages is that it has an inside gate that will be keeping out the non-relevant features from changing the loss in the model and considering only the critical ones.
- LSTM stands for Long-Short Term Memory, that reflects one advantage for this type of network on the other types of recurrent neural networks, which is the ability for it to keep track for long term dependencies well, and not only short term dependencies, as the case with RNN and GRU networks.

And experiments shows that indeed this is the best type among the ones got tested and worked with, since it does get to the best result according to the metrics of accuracy and F1-score and losses, besides the precision and recall.

Also when this model get integrated to the camera stream to receive the video feed from the camera and predict from the blendshapes found in it, it gives the highest stability in results and does not fluctuate and keep jumping between classes far from each other. And that is understandable because it takes into consideration the previous states.

5.2. Fully-connected (Dense) Neural network

To get faster prediction time than the one of LSTM network decided to build another network that is basic and simple and chose the fully connected because it is a better fit for a structured data processing application, than convolutional networks.

After a few trials also considered using a transformer network for the task but due to the limited time and resources available did not work on that.

5.3. Architecture search and hyperparameter optimization

After choosing to work with LSTM, turn into building the model architecture and optimizing the hyperparameters, since the model is being built with keras the way to do that efficiently is by using keras tuner, which is a library that help picking the optimal values, by using search algorithms such as grid search, or random search which is being used in this model development, to narrow the scope of the search need to provide ranges for the search to be done within.

Another consideration is the parameters to be changed and optimized, and which ones to be left on the default value.

The following is a list of parameters tried to change and get a better result by doing that:

- For the LSTM layers: layer_units, kernel_regularization, activation function, weight decay, recurrent activation function, kernel initializer, dropout, return_sequences, go_backwards.
- Optimizer:
    - AdamW: learning rate, beta_1, beta_2, amsgrad, weight decay, clip norm, global clip norm, use ema, ema momentum.
    - SGD: learning rate, momentum, weight decay, global clip norm, use ema, ema momentum.
- Loss function: mean squared error, categorical crossentropy.
- Others: epochs, batch size, class weight.

Also, callbacks for the model got used such as checkpoint callback, early stop callback.

Many of these parameters used for a short time and only a few experiments, and the others that got focused on to experiment with for a few weeks, the search algorithm was set for a random search since it will be faster to cover a wider area of the search range than using other searching methods, and the following are the parameters with their final search ranges:

- Learning rate - (1e-6, 1e-3)
- Layer units - (10, 32, step = 4)
- Kernel regularization - (1e-10, 1e-5)

- Activation function - (selu, tanh, relu, leaky_relu)
- Weight decay - (1e-10, 0.0009)

Searching within these values was decided after a few extreme experiments to find what will the effect of using extreme values, and making sure that there is no chance for the model performance to be improving outside of this range.

One thing worth mentioning is that the selu function is included here while it is not one of the most common activation functions because at one stage of the model development when the relu and tanh functions were considered the model showed loss explosion with relu, might be as a result of the fact that relu does not have a negative part and it is equal to zero and high underfitting behavior from tanh possibly because it has a limited slope area and the slope goes to zero for big numbers. So the selu was selected as a potential solution and it gives improved results.

The best model that was trained for 5000 epochs that took around 1.5 to 2 days with batch size of 128 had an architecture as follows:

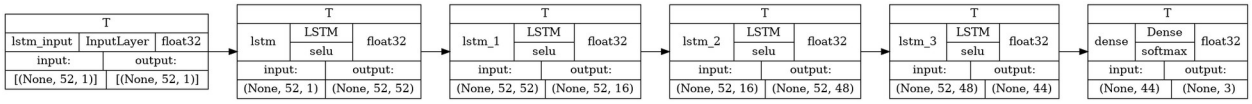

fig (7) LSTM model structure

And the other parameters are:

- Kernel regularizer l2= 0.00000195
- Optemizer: AdamW, learning_rate = 1.0955e-06, global_clipnorm=1, amsgrad = **True**

Experiments to optimize the model did not stop even after reaching the result to be mentioned in the next chapter, since it shows that there is a big possibility for improvement.

The metrics were used to find the best parameters through the optimization and training process were the following:

- Loss: which is calculated using the categorical cross entropy function which is a multiclass classification function, and get calculated by summing the dot products of the logarithm of the predicted softmax output with the corresponding ground truth label.
- Categorical crossentropy: is the loss of the model predictions when the labels are encoded in one-hot vector, and it is calculated with the following equation:

$$CE = \sum_{i=0}^{classes} y_{ilabel} \cdot \log\left(y_{ipred}\right)$$

- Categorical accuracy: used instead of ‘accuracy’ since the classification is on a multiclass data so using the one hot vector encoding for the labels and the categorical cross entropy loss make the categorical accuracy more relevant. Since in the case of working with three classes there will always be two zeros and a one in the ground truth labels, so when calculating the accuracy there will always be one zeros is actually classified, while categorical accuracy counts only when the one is being correctly classified, and neglecting the zeros.
- F1-score: it is a certain ratio between the precision and recall values get used to simplify the calculation of an improved model over using the precision and recall as separate metrics, and it is represented in the following equation:

$$F1score = \frac{Precision \cdot Recall}{Precision + Recall}$$

For the fully connected network was not there an optimization process since after a few experiments the metrics showed high performance from the network, but when it gets integrated to the camera stream, it does not give acceptable results so the work stopped on it.

6. Results

After training the model with the parameters mentioned for the number of epochs required, it gives result below satisfactory to meet the project goal which is integrating the model to a camera stream and later to a robot. but still a lot can be learned from the experiments

And for the model that was considered as the final model when evaluated on the test set, the metrics values were:

Loss = 0.6238

Categorical crossentropy = 0.6235

Categorical accuracy = 0.7199

F1 score = 0.6298

And the confusion matrix for this model is:

	predicted			
Ground truth	class	0	1	2
	0	251	35	5
	1	110	850	150
	2	21	140	84

Trying different architectures or parameters and training for longer time does not give better performance but the confusion matrix keeps oscillating between an improvement for the happiness class classification one time and for sadness class classification another and the one above is biased to the happiness class.

And when the model gets integrated to the camera stream it indeed gives a stable output for happiness and sadness passing by the unknown class when the face look changes, and for all other facial poses the unknown class is the output. Without any fluctuations, and of course with accuracy of around 72%.

Unlike the other types of models such as the fully connected which when it get integrated to the camera stream, does not show sufficient stability, and it jumps from happiness to sadness without passing through the unknown class more specifically the neutral expression, and also with other face poses it is much less stable than LSTM.

7. Discussion

In this chapter will discuss a few of the decisions that have been made while working on the project or alternative paths that could have delivered different results for the same task.

For every face gets detected by MediaPipe three outputs results from the process, that are the blendshapes of the face, the landmarks of the face, and the facial transformation matrix, and for this model used the blendshapes since they are a compact representation for the landmarks, but the landmarks are still a choice to be used in case of the availability of more computation resources, also in that case finding more blendshapes for the face based on the landmarks would be a choice to look for but that is what is provided by MediaPipe for now, also that will result a requirement for a longer detection and prediction time when the model get deployed. and the facial transformation matrix get used as the name indicates for transformations and not as a record or a representation for the face expressions.

Another idea is that taking 1000 image from the unknown class five element classes instead of 1500, because that will reduce the bias in the dataset, that might be valid but still the features of the unknown class are so various and complex in comparison to the happy class or the sad class which has 4000 instances each, so there will still be unevenness.

According to the benchmark of the dataset [11] the best classification accuracy that could be gotten on it is 72% and that is what this model is showing, and taking into consideration that the images are going through MediaPipe and only the available blendshapes are being classified.

Getting classification results with high bias shows that there are many issues in the process so starting from the dataset can notice that the image being misclassified in the test set includes some type of block for the face and as the images shows

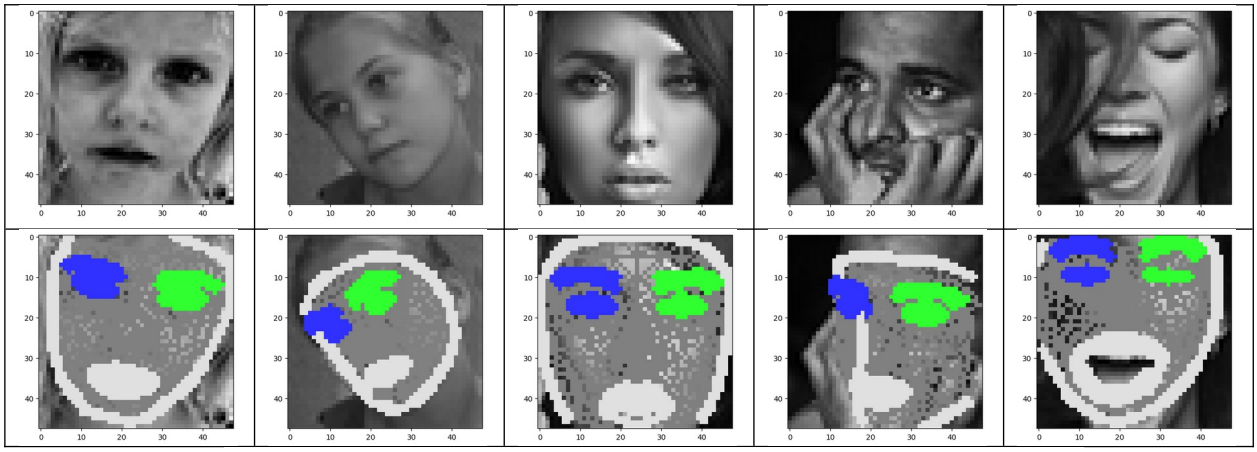

fig (8) misdetected faces

Another big issue that also shows in the visualized images is that the dimensions are very small, so MediaPipe is having trouble detecting the face boundaries and features, and that will negatively affect the results for both landmarks and blendshapes.

To get the best results for a happy, sad classification model for a social robot also discussed the possibility of categorizing the emotions as positive (happy) and negative (sad), by including disgust, angry, fear, and sad in one class and happy, surprise, and neutral in the other class but of course that will give an inaccurate perspective to the robot about the real emotions the person detected is having, but it could be considered and according to the use case and application of the robot.

To improve the speed of prediction for the model when implemented, two methods got considered, the first is changing the model architecture and the other is removing some features (blendshapes) from the dataset, but both had some other down sides that stopped the idea from getting developed further.

8. Conclusion

From the work done can conclude that the LSTM is a suitable choice for the application since it shows good results on video stream detection, even after training on an images dataset, and not video. Augmentation is important and improves the performance of the model on the blendshapes dataset significantly.

To continue improving the model in the way of building a social robot, need an improved dataset with higher quality images, bigger dimension size, and count. And in this case more computation resources will be required for training the model but much better weights results to predict at inference stage.

9. References

[1] https://ai.google.dev/edge/mediapipe/solutions/guide

[2] Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M., ... & Grundmann, M. (2019, June). Mediapipe: A framework for perceiving and processing reality. In *Third workshop on computer vision for AR/VR at IEEE computer vision and pattern recognition (CVPR)* (Vol. 2019).

[3] Farkhod, A., Abdusalomov, A. B., Mukhiddinov, M., & Cho, Y. I. (2022). Development of real-time landmark-based emotion recognition CNN for masked faces. *Sensors*, *22*(22), 8704.

[4] Carlisi, C. O., Reed, K., Helmink, F. G., Lachlan, R., Cosker, D. P., Viding, E., & Mareschal, I. (2021). Using genetic algorithms to uncover individual differences in how humans represent facial emotion. *Royal Society open science*, *8*(10), 202251.

[5] Lewis, J. P., Anjyo, K., Rhee, T., Zhang, M., Pighin, F. H., & Deng, Z. (2014). Practice and theory of blendshape facial models. *Eurographics (State of the Art Reports)*, *1*(8), 2.

[6] Li, S., & Deng, W. (2020). Deep facial expression recognition: A survey. *IEEE transactions on affective computing*, *13*(3), 1195-1215.

[7] https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data?select=train.csv

[8] R. Gross, I. Matthews, J. Cohn, T. Kanade, and S. Baker, “Multi-pie,” Image and Vision Computing, vol. 28, no. 5, pp. 807–813, 2010.

[9] https://rafd.socsci.ru.nl/RaFD2/RaFD?p=register

[10] http://mohammadmahoor.com/affectnet-request-form/

[11] https://paperswithcode.com/sota/facial-expression-recognition-on-fer2013-1

[12] https://arkit-face-blendshapes.com/

Appendix A | Cleaner

When using the checkpoint callback to save the model every epoch and running the training for thousands of epochs, a big number of saved model will accumulate after a few hours so had to create a simple script that will run every 10 minutes to clean the directory where the checkpoint models are being saved and keep only the two that are showing the best metric performance

Appendix B | Image generator

One strategy to get extra images for training besides dataset got considered was using generative AI to generate faces images using replicate: kandinsky 2.3 model, but since this process takes so much time, and some companies around the word such as Microsoft are in opposition to mix generated faces images with real ones, had to stop considering it, and work with available datasets.