Facial Palsy Image Processing Research Report

Bingnan Huo (Nick)

Backgrounds

Facial nerve paralysis (FNP) is a medical condition that affects the ability of individuals to control certain facial muscles. One of the most common forms of FNP is Bell's Palsy, which affects approximately 4 out of 10,000 people in the United States each year. People with Bell's Palsy experience one or, in rare cases, both sides of their face becoming unresponsive. The exact cause of Bell's Palsy is unknown, and it typically takes up to six months or longer for a patient to recover after being affected.

In the past, all diagnoses of FNP were made by doctors based on their subjective judgement and professional experience. However, previous research [1] has shown that observer bias is common when FNP patients are seen and diagnosed by clinicians, and that a machine learning (ML) based approach found less facial asymmetry in severe FNP patients and more asymmetry in healthy faces than clinicians.

The goal of the research project discussed in this report is to develop software that uses computer vision and machine learning techniques to grade the severity of FNP and assist in the diagnosis and recovery tracking of patients with FNP using the

House-Brackmann scale. The aim is to provide doctors and patients with another tool that can be used for reference and potentially reduce bias in diagnosis.

To achieve this goal, the research project will follow a framework similar to that used in other studies with similar goals. This includes:

1. Using an ML-based landmark localizer to determine the locations of key facial anatomical points (landmarks) on an image of the face
2. Correcting for head tilt or perspective distortion using geometric or ML algorithms
3. Calculating asymmetry in the face using some algorithmic method
4. Using an algorithmic or machine learning-based approach to translate the asymmetry measurements into classifications.

Literature review

These are the most important studies we used in our work

Guarin [2] published Emotrics, the landmark predictor various studies have used. We are also using Emotrics. Emotrics uses dlib to predict for 68 key facial landmark points. It included several trained models, including one trained using the iBUG 300-W dataset and the MEEI facial palsy dataset [3]. We will be using the MEEI model as it is trained on the MEEI dataset [3] that consists of 60 patients with a spectrum of types and severities of FNP. Guarin wrote Emotrics with the intention of creating an app that would help doctors measure facial asymmetries. Thus, Emotrics has a GUI, built with PyQt5. However, we are only interested in using the part of code that would extract the facial landmarks from input images.

Fig. 3. Comparison of automatic facial landmark localizations by models trained using the 300-W **(A, C)** and MEEI **(B, D)** databases among two patients with facial palsy in the validation group.

Bandini [4] explained that having specific patient-population-trained models or fine-tuning pretrained models (using the general population) will lead to lower prediction errors measured by nRMSE%.

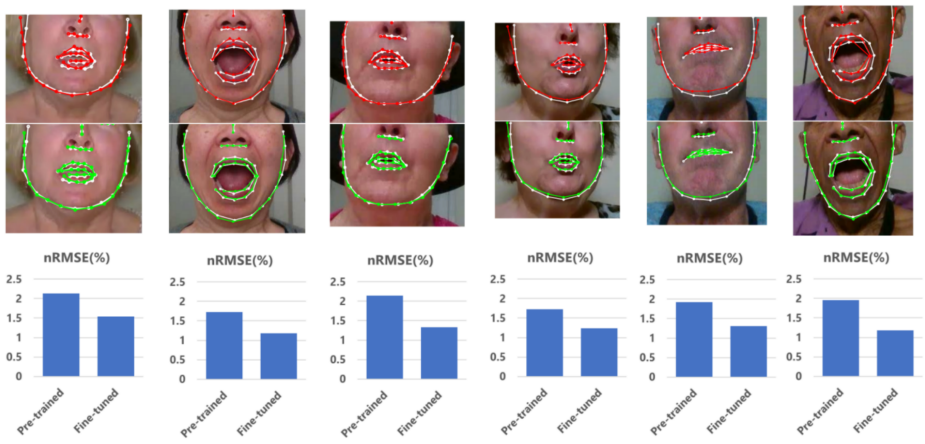


Fig. 3. Comparison between pre-trained FAN (top row) and fine-tuned FAN (middle row). Bar plots (bottom row) show the nRMSE values corresponding to the above sample frames. In these examples, we show how the fine-tuning can improve the landmark localization accuracy of facial contour and mouth regions. White: ground truth landmarks; Red: facial landmarks obtained with pre-trained FAN; Green; landmarks obtained with fine-tuned FAN.

Gemma Parra-Dominguez's study [5] is fundamental to our work. We are mainly incorporating his proposed framework and adding the severity grading functionality in the final step.

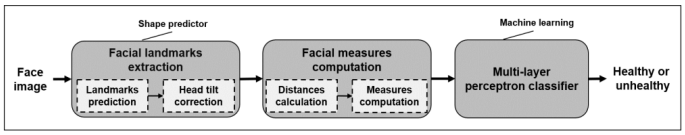


Figure 1. Framework of the proposed facial paralysis classification system.

Gemma proposes to use 51 landmark points instead of 68 as the jaw line is not the main concern for evaluating illness or severity grading of facial palsy. Gemma also defined a set of distances and features, found through a ML program. Gemma's work used a Multilayer Perceptron Network to differentiate between healthy & ill images.

Methods

Our code can be found in the bp_research repo.

Overall Process

Gemma's work [5] demonstrated successful application of ML in determining whether a face is healthy or ill with FNP by using the ML-based software Emotrics [2] to automatically predict key facial landmarks from facial images and then computing facial features to determine health or illness.

We acquired Emotrics from its GitHub Repo. The most important piece of code that was used for prediction was in *GetLandmarks.py*. Then, our process is mostly as follows:

1. First, load the image with cv2.
2. Use Emotrics' landmark localizer to predict for landmarks
3. Tilt correction
4. Distances (with optional scaling) and features computation
5. ML model (SVM / XGBoost / MLP) for severity grading or healthy/ill classification

Now, we will go into a bit more detail about how the Emotrics code works (Get_Landmarks) in predicting landmarks.

Get_Landmarks is a class object that handles the image input and landmark predictions. It requires images stored in numpy arrays (e.g. 2000*1000*3) and a string specifying which pre-trained model (e.g. "iBUG", "MEE", etc.) to use for landmark prediction. It will store the predicted landmark results in Get_Landmarks._shape, which

is a (68*2, int16) numpy array. Methods to process the eye area and predict the location and size of the eyes are included. But we won't heavily utilise this.

The specific process is:

1. Given the loaded image, it first rescale the image to a smaller width (200px) and

   convert it to grayscale. Choosing larger image resolutions will lead to increased processing time. Moreover, after testing on datasets with ground truth information, we found that having a higher width or colored image did not provide any significant benefit.

2. dlib is then used in Emotrics' code to predict for 68 landmark points.

3. In *ProcessResults.py*, we process the predicted landmarks in methods mentioned by Gemma:

   Remap the 68 points to 51 points, as defined by Gemma

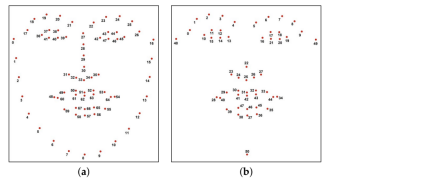
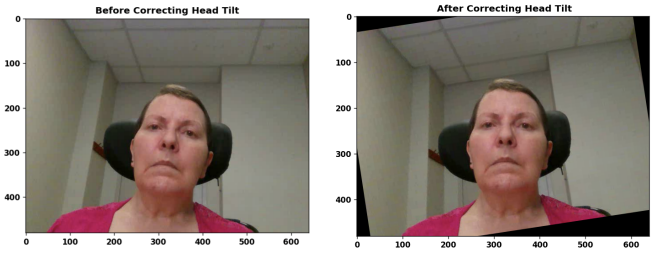


Figure 2. Facial model adapted from the work proposed by Matthews and Baker [3]. (a) Standard 68 key points model; (b) our 51 key points model.



   Correct for head tilt by using specific landmark points on the left and right side of the head (point #48 and #49). We calculate the angle formed by these two points and rotate the set of predicted landmarks by this angle in the opposite direction.

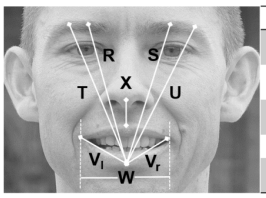
   Something that can be tried for future work: after initial rotation of image and landmarks, run prediction on rotated image again. Since rotations can lead to

worse accuracy in prediction, this initial adjustment should bring better accuracy in further predictions.

Then we calculate the Distances between landmarks and Features, as algebraic summaries of the distances, according to Gemma's definition.

i. We calculate the Euclidean distance between certain landmark points.

ii. (Optional) We could also scale these distances by assuming the iris diameter or distances between eyes to be the population average, and then compute a pixel scale (mm/pixel); or by using the bounding boxes to normalise every photo – translating distances into proportion of bounding box diagonal length. This step is not necessary for the next step in feature computations, but it can be helpful when we want to compare different images with just landmark distances directly. Refer to



Distance	Description
R, S	Distance from the fourth eyebrow point to the middle of the jaw point (left and right)
T, U	Distance from the third eyebrow point to the middle of the jaw point (left and right)
V_l, V_r	Distance from the mouth corner to the outer middle point of the mouth (left and right)
W	Distance from the mouth corners.
X	Distance from the middle nose tip to the outer middle point of the mouth

(c)

Figure 3. Facial distances to compute spatial relations between facial landmarks. (**a**) Distances A to K; (**b**) distances L to Q; (**c**) distances R to Z.

f20	Mouth	Maximum value	$\max(P_u/W, Q_u/W)$
f21	Mouth	Maximum value	$\max(W_l/W, W_r/W)$
f22	Nose	Angle	$\lvert\angle(P23, P27)\rvert$
f23	Combined	Angle	$\lvert\angle(P22, P37)\rvert$
f24	Combined	Maximum value	$\max(J/K, K/J)$
f25	Combined	Maximum value	$\max(T/A, U/A)$
f26	Combined	Maximum value	$\max(R/A, S/A)$
f27	Combined	Ratio	C/A
f28	Combined	Ratio	X/A

In *f19*, *f20* and *f21*, W is the distance depicted in Figure 3c, being the perimeter values W_l and W_r calculated as $W_l = \overline{S}(P28, P29, P30, P31, P37, P38, P39)$ and $W_r = \overline{S}(P31, P32, P33, P34, P35, P36, P37)$.

ProcessResults.scale_results() and *ProcessResults.scale_by_bbox()*

iii. The features we compute are algebraic or geometric operations – such as a ratio, a degree, or a slope – that use the landmark distances and summarise asymmetries in the face. These results should be unaffected by scaling of the distances, since the features are relative to the distances.

iv. These Distances and Features are stored in python dictionaries.

Results can be exported to csv files (named "dists.csv" and "features.csv") using *ProcessResults.save_results()*. Alternatively, distances and features can be

individually exported with *ProcessResults.save_dists()* or *ProcessResults.save_features().*

4. With the computed features, we can then feed them into a machine learning algorithm for severity grading.

   We have attempted using random forest, XGBoost, SVM, Multilayer Perceptron Network. But we were not able to achieve the prediction accuracy in published papers.

Evaluating Landmark Predictor

While testing the Emotrics landmark predictor, we noticed that it often produces inaccurate results, though they are not too far off. Regardless, we decided to carefully evaluate its performance. Later, we also tested it through the Toronto Neuro Face dataset, which contains ground truth landmarks, by calculating the Root Mean Squared Error (RMSE) for our predicted landmarks.

Despite the dlib prediction method being deterministic (implemented via cascade of regression trees), we observed that small changes to the input image could lead to somewhat different predictions. This is against the claim that it would “provide accurate facial landmark localization results under multitude of pose, illumination, and expression conditions.” Thus, we tried to apply test-time augmentations, which could help to deal with the randomness in prediction results. We implemented the test-time augmentation in LandmarkTester.py with:

i. Random rotations of the image

- *LandmarkTester.test_random_rotation()*

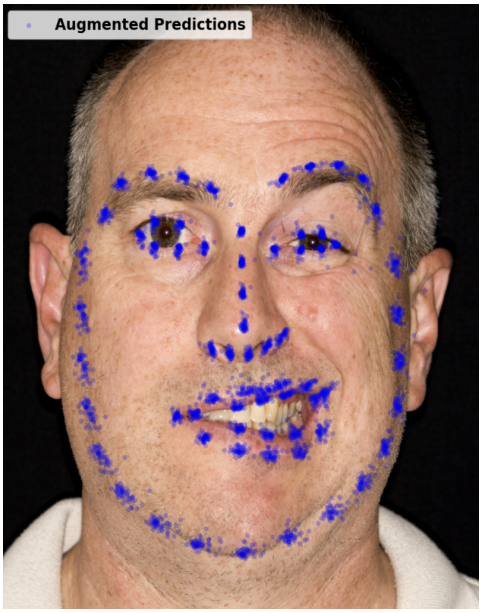
- We noticed that large rotations produce less accurate predictions. This is also why we need to correct for head tilt.

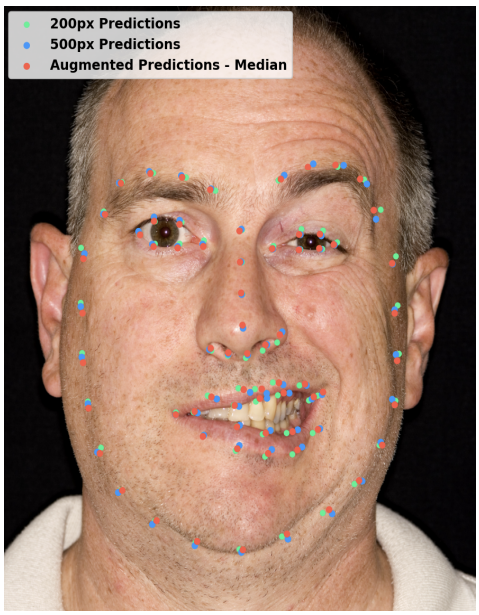
ii. Add noise to the image or shifting the overall intensity (brightness) of an image

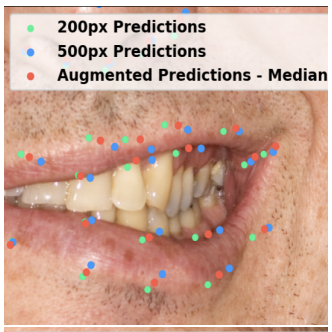
- *LandmarkTester.test_intensities()* & *LandmarkTester.test_noises()*
- We also observed that larger noise results in less accurate prediction.

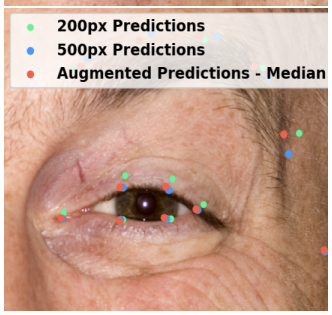
iii. We repeat these processes on 30 or 40 images to augment them. The landmarks predicted from these augmented images will then be used to calculate the median landmark.

- We hypothesised that the median will be more accurate than the predicted landmarks. And we later tested this with RMSEs.









While predicting for the landmarks of one image can take from 0.5 seconds to 4 seconds depending on the the dimension of the given image, this can add up to be quite

a long time in test-time augmentation as we repeatedly predict the landmarks. Thus, we are looking for ways to speed up this process. As dlib predictions only run on the CPU single-thread, we then tried to speed things up by implementing multiprocessing. It provided roughly 5x+ speed improvements.

We used the multiprocess library, which is different from the multiprocessing library. Basically, we will let all of our CPU threads perform these single-threaded augmentation (adding rotations & noise and landmark predictions) at the same time. We rely on the *"with … as pool"* and *pool.map()*.

```
#with mp.get_context("spawn").Pool(processes=19, maxtasksperchild=15) as pool:
with mp.get_context("spawn").Pool() as pool:
    self._test_landmarks: Any = pool.map(func, self._test_params)
```

In *test_emotrics_parameters.ipynb*, we ran a test comparing if some parameters will lead to better accuracy (i.e. lower RMSE). The four different sets of parameters are:

1. Original: 200px width, grayscale image
2. Original colored: 200px width, colored image
3. Hi-res: 300px width, grayscale image
4. Hi-res colored: 300px width, colored image

In addition, we are also doing test-time augmentations to compute the median landmarks associated with each set of parameters. For each image, we will also normalise their RMSE value by the bounding box size. Without normalising, we would not be able to compare the RMSE of different images.

Dataset I/O Tools

Toronto Neuro Face dataset

DatasetTester.py is used to load and use the TNF dataset. This dataset is stored first by patient type, then by frames (images) and ground truth landmarks & bounding boxes. Inside the landmarks folder, we can find files corresponding to some patient doing some task. In these files, we can find that several frames of the patient doing that task have been labelled with the ground truth landmarks. We then use the file names of these landmarks files to separate the different patients, tasks, and frames.

These data are stored in a nested dictionary containing information on all the patients. Each patient in the patient's dictionary is also a dictionary. And each patient will have tasks. In each task dictionary, there are different frames. For each of the frame dictionaries, the ground truth landmarks, bounding boxes, and images are stored there. And only with this dataset, we were able to evaluate Emotrics' performance with respect to the different parameters. (See *test_emotrics_parameters.ipynb*.)

	frame	f0	f1	f2	f3	f4	f5	f6	f7	f8	...
0	14	4.175257	4.655471	9.091028	1.066030	0.073001	0.081433	0.160013	1.279485	1.055445	...
1	34	0.591429	2.440080	0.691288	1.012876	0.010323	0.042613	0.012066	0.692982	1.095588	...
2	60	4.452492	3.774128	2.191985	1.045201	0.077867	0.065966	0.038276	0.481127	1.056630	...
3	105	0.451318	3.213133	1.636566	1.024191	0.007877	0.056139	0.028571	1.074877	1.086093	...
4	124	0.207511	2.018164	3.362474	1.016940	0.003622	0.035238	0.058754	0.114674	1.116456	...
5	138	0.588625	1.468147	1.280177	1.009646	0.010274	0.025630	0.022347	0.425059	1.041859	...
6	174	2.456519	4.194554	3.155546	1.039348	0.042901	0.073340	0.055130	1.565416	1.116056	...
7	190	1.887970	2.908428	1.475049	1.031041	0.032963	0.050805	0.025750	1.441686	1.040726	...
8	227	2.359467	2.997962	3.098227	1.036182	0.041204	0.052372	0.054127	1.418686	1.128171	...
9	266	2.476753	3.658986	2.796396	1.045956	0.043254	0.063948	0.048845	1.776917	1.059595	...
10	287	1.696872	3.313147	5.650455	1.041103	0.029625	0.057890	0.098940	0.487139	1.020475	...

MEEI dataset

In *dataset_loading_MEEI.ipynb*, we loaded the MEEI dataset. We mainly extracted necessary information, such as patient category and #id, and patient severity, from the

excel spreadsheet provided with the dataset. Using those information, we then used pathlib to create the folder paths for each individual patient. In each folder, 8 images and 1 video of the patient doing each of 8 assigned tasks (see info sheet in the dataset folder). Unfortunately, the dataset we acquired from MEEI does not contain the ground truth landmarks for each patient (mentioned in Guarin's paper). Further attempts can be made to contact Guarin and acquire that dataset.

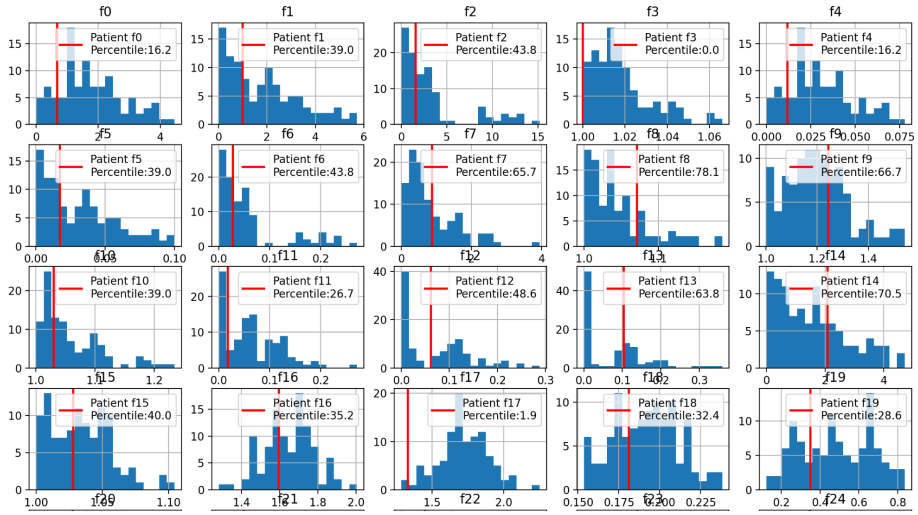
After all the folders were being located, we loaded each image, its predicted landmarks, calculated features, and severity grade on the House-Brackmann scale to their respective np arrays. With these data, we then were able to look at the distribution of features and separability of data in the MEEI dataset. This allowed us to better understand what the data is like, and will help in the later step when building the machine learning model for severity grading.

	Category	Sub-category	#	Side	Gender	Age	Path	HB_scale
0	Flaccid	Complete	1	Left	Male	54	MEEI_Standard_Set/Flaccid/CompleteFlaccid/Comp...	6
1	Flaccid	Complete	2	Right	Female	16	MEEI_Standard_Set/Flaccid/CompleteFlaccid/Comp...	6
2	Flaccid	Complete	3	Right	Male	38	MEEI_Standard_Set/Flaccid/CompleteFlaccid/Comp...	6
3	Flaccid	Complete	4	Left	Female	31	MEEI_Standard_Set/Flaccid/CompleteFlaccid/Comp...	6
4	Flaccid	Complete	5	Left	Female	52	MEEI_Standard_Set/Flaccid/CompleteFlaccid/Comp...	6
5	Flaccid	Mild	1	Left	Male	72	MEEI_Standard_Set/Flaccid/MildFlaccid/MildFlac...	3
6	Flaccid	Mild	2	Right	Female	55	MEEI_Standard_Set/Flaccid/MildFlaccid/MildFlac...	3
7	Flaccid	Mild	3	Left	Male	75	MEEI_Standard_Set/Flaccid/MildFlaccid/MildFlac...	3
8	Flaccid	Moderate	1	Left	Male	51	MEEI_Standard_Set/Flaccid/ModerateFlaccid/Mode...	4
9	Flaccid	Moderate	3	Left	Female	62	MEEI_Standard_Set/Flaccid/ModerateFlaccid/Mode...	4
10	Flaccid	Moderate	4	Right	Male	21	MEEI_Standard_Set/Flaccid/ModerateFlaccid/Mode...	4
11	Flaccid	Moderate	5	Right	Female	64	MEEI_Standard_Set/Flaccid/ModerateFlaccid/Mode...	4
12	Flaccid	NearNormal	1	Left	Female	39	MEEI_Standard_Set/Flaccid/NearNormalFlaccid/Ne...	2
13	Flaccid	NearNormal	2	Right	Female	53	MEEI_Standard_Set/Flaccid/NearNormalFlaccid/Ne...	2

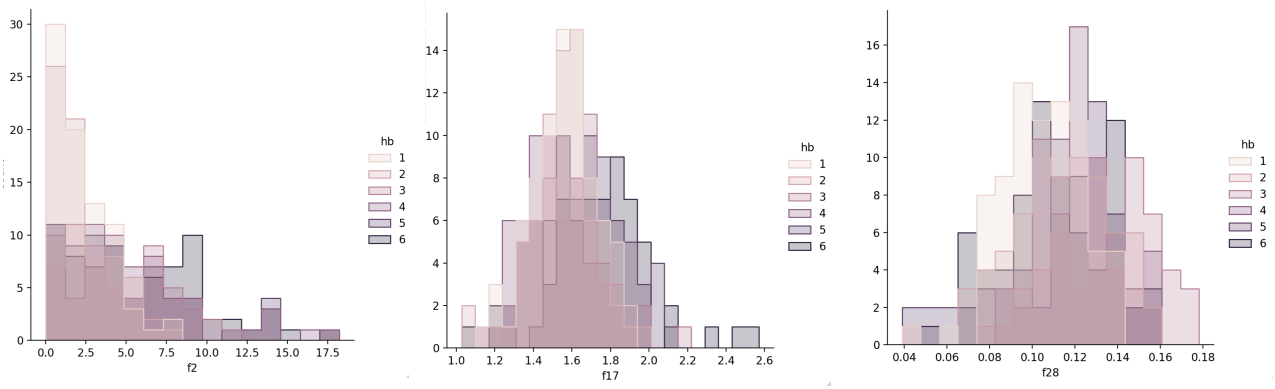
Building ML Models

Before building the machine learning models, we want to at least have a grasp of what our data looks like. Therefore, we created different visualisations to help us understand the distributions of patient features, the separability, and the variability of the data.

a. Histograms on distributions of features from healthy people from the TNF dataset. While this is not on the patient distribution. We are able to highlight certain characteristics (features) that specific patients are abnormal relative to healthy patients. For this patient though, it seems that most of the features are within the “normal” range, except maybe f13, where the patient is at the upper 80 percentile.

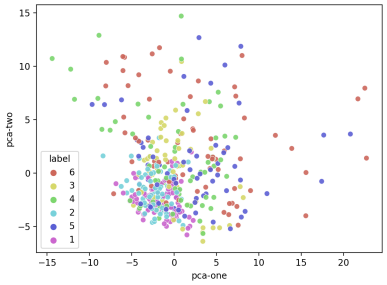


b. We can also see the distributions of patient features, grouped by their severity, to better understand how much the data are overlapping, for different severity groups. These are plots generated from the MEEI dataset.

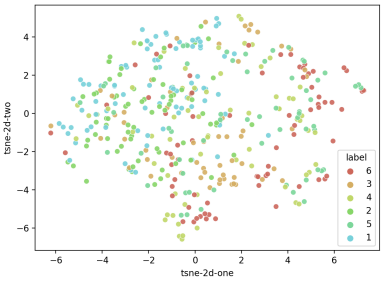


These graphs revealed that the MEEI data, in fact, has a lot of overlap in features between patients of different severities. Thus, it will likely be hard for the machine learning model to correctly grade their severities.

c. The t-scne plot is another method to see the variability and distribution of high dimensional data on lower dimensions such as 2D. However, this plot also indicates that our data is hard to separate between the severity groups.



d. Principal Component Analysis is also doing something similar. Here, we are reassigning the basis, and we are displaying the first two PCA directions. We are also seeing that features from all severities are clustered together.



After having a preview of the data. We hope that applying data scaling will lead to better machine learning model performance. Finally on the machine learning model, in *MEEI_mlp_sklearn.ipynb*, we tried to rescale the features using

- StandardScaler: standardise features by subtracting the mean and scaling to unit variance
- MinMaxScaler: scale features so that all of them are between 0 and 1
- PowerTransformer (Box-Cox): apply a power transform featurewise to make data more Gaussian-like
- QuantileTransformer (Gaussian): transform features using quantiles information

However, using scikit-learn to implement a MLP only got us about 80% accuracy. This is not great compared to Gemma's results [5]. Trying to grade the severity yielded even worse results: around 40% accuracy. More work still needs to be done regarding the

machine learning model or the landmark predictor before this can actually be helpful, grading patient severities in a practical sense.

Installation Guide

Mainly for linux

1. Install Nvidia Drivers, Cuda Toolkit and cuDNN
2. Install conda/miniconda and optionally mamba
3. Load the bp conda environment (which includes all necessary Python libraries) with bp.yml file by running: `conda env create -f bp.yml`
4. If loading the conda environment was not successful, try to create your own conda environment with these packages and Python 3.10: **opencv, dlib, multiprocess, numpy, scipy, scikit-learn, matplotlib, pandas**, and potentially others not mentioned here.

Future work

While we attempted to recreate the results in Gemma (classifying patient and non-patient), the highest accuracy

Table 2. Cross-validation results for tests on the MEEI database.

n-Fold	TN	TP	FN	FP	Accuracy [%]
1	22	32	8	2	84.37
2	24	37	3	0	95.31
3	24	33	7	0	89.06
4	24	38	2	0	96.87
5	24	40	0	0	100
6	23	38	2	1	95.31
7	24	38	2	0	96.87
8	22	38	2	2	93.75
9	24	35	5	0	92.19
10	24	38	2	0	96.87
Average	-	-	-	-	**94.06**

Table 3. Resulting confusion matrix of the test on the actual MEEI database.

Healthy = 0	Unhealthy = 1	Correct Label
80	0	healthy
1	399	unhealthy

we have achieved on the MEEI dataset was only around 80% with cross-validation, lower than the 95%+ accuracy claimed in the paper. Future work could try to build better machine learning models to achieve better classification accuracy.

Moreover, more work is still needed to develop a mobile APP that acts as the interface for capturing patient images and getting predicted severity gradings.

References

[1] Miller, Matthew Q., et al. “The Auto-eFACE: Machine Learning-Enhanced Program Yields Automated Facial Palsy Assessment Tool.” Plastic and reconstructive surgery vol. 147, 2 (2021): 467-474. doi:10.1097/PRS.0000000000007572

[2] Guarin, Diego L., et al. “Toward an Automatic System for Computer-Aided Assessment in Facial Palsy.” Facial plastic surgery & aesthetic medicine vol. 22, 1 (2020): 42-49. doi:10.1089/fpsam.2019.29000.gua

[3] Greene, Jacqueline J., et al. “The spectrum of facial palsy: The MEEI facial palsy photo and video standard set.” The Laryngoscope vol. 130, 1 (2020): 32-37. doi:10.1002/lary.27986

[4] Bandini, Andrea, et al. “A New Dataset for Facial Motion Analysis in Individuals With Neurological Disorders.” IEEE journal of biomedical and health informatics vol. 25, 4 (2021): 1111-1119. doi:10.1109/JBHI.2020.3019242

[5] Parra-Dominguez, et al. “Facial Paralysis Detection on Images Using Key Point Analysis.” Appl. Sci. 2021, 11, 2435. https://doi.org/10.3390/app11052435