HUMAN FACIAL EMOTION RECOGNITION

Nguyen Viet Anh
Data Science and Artificial Intelligence
Ha Noi University of Science and Technology
vietanh040902@gmail.com

Tran Bao Chi
Data Science and Artificial Intelligence
Ha Noi University of Science and Technology
chi.tb200083@sis.hust.edu.vn

Nguyen Hoang Dang
Data Science and Artificial Intelligence
Ha Noi University of Science and Technology
dang.nh204873@sis.hust.edu.vn

Nguyen Tran Xuan Manh
Data Science and Artificial Intelligence
Ha Noi University of Science and Technology
xmak9102@gmail.com

Tran Quoc Khanh
Data Science and Artificial Intelligence
Ha Noi University of Science and Technology
khanh2k2@gmail.com

July 6, 2022

ABSTRACT

In the advent of Big Data explosion, methods that can process, extract knowledge, information and produce values are in high demand. Machine learning methods was introduced to process this immense amount of data and applied in many real-world tasks. In this report, we use Machine learning methods in emotion recognition, a task benefiting many domains in real life nowadays.

1 Introduction

Emotion recognition task enable us to understanding the the facial emotion of humans, which brings about advantages to numerous fields such as education, banking, health-care, etc. This task can be viewed as an image classification, which has been extensively researched in Machine learning field. Hence, in this report we also apply ML models with 2 different approaches: (1) Feature engineering accompanying with traditional Machine learning models and (2) Directly apply Convolutional Neural Network (CNN) (Modern Machine learning method). At last, we showcase some results from our conducted experiments.

2 Background

In this part, we will introduce the details of methods applied in feature engineering and modelling

2.1 Feature engineering

In traditional machine learning methods, features is quintessentials, hence, instead of using the raw pixels as features, we use 2 feature extractor methods for facial image: Histogram oriented gradients(HOG) and Facial Landmarks

2.1.1 Histogram of Gradients

a. Image gradient
In mathematics, gradients represent how continuous function changes w.r.t its arguments. Image can be considered as a

discrete function $F(x, y)$, image gradients in 2 dimensions x and y can be approximated by central finite difference (Fig.1), in this report we use the Sobel kernel(Fig.2) which is most prevailing kernel in calculating image gradients.

$$f'(x) \approx \frac{f(x+0.5h) - f(x-0.5h)}{h}$$

Figure 1: Central finite difference

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \mathbf{I}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \mathbf{I}$$

Figure 2: Sobel filter

Gradients of image provides 2 pieces of information of image, the direction and the rate of change in pixel intensity of image, with 2 values of image gradient: **direction** (Fig.3a) and **magnitude** (Fig.3b)

$$G = \sqrt{G_x^2 + G_y^2}$$

(a) Magnitude

$$\theta = \text{acrtan}(\frac{G_y}{G_x})$$

(b) Direction

Figure 3: Gradient information

From gradients calculation, we can display the magnitude and direction of gradient at each pixel:

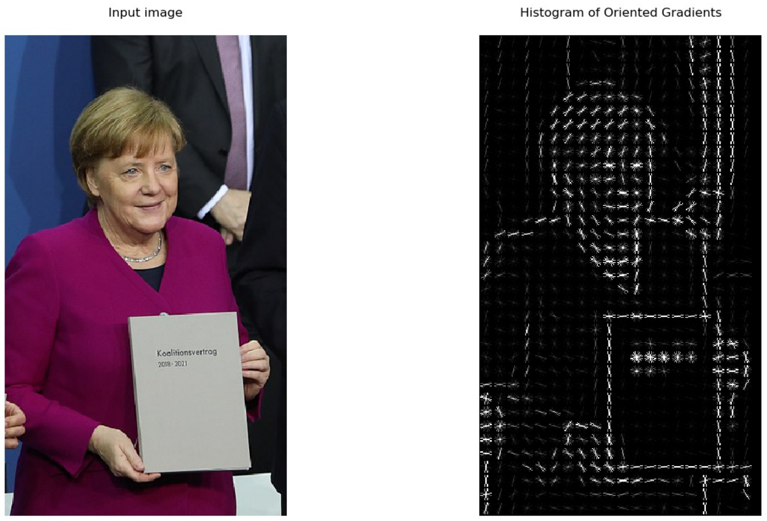
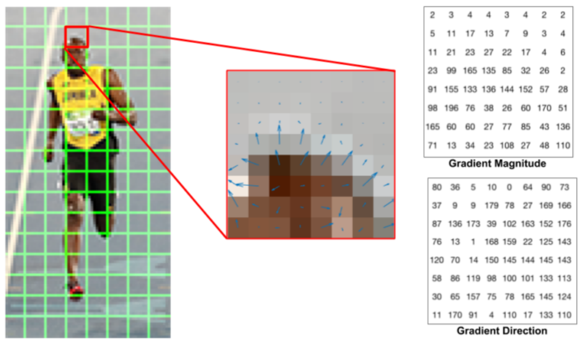


Figure 4: Original Image vs Gradient Image

b. Feature descriptor
From the image gradient, HOG feature descriptor is derived from process below:

- Grouping pixels into block: We divide image into groups of pixels with nxn group size and calculate the gradients



2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

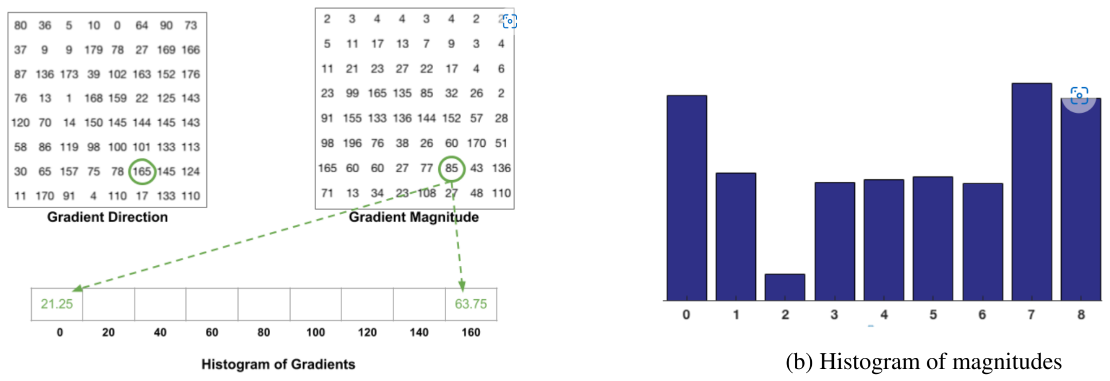
Gradient Magnitude

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

Figure 5: Gradients for 8x8 group

- Calculate HOG for each block: We map the magnitude values to the bins of direction with specified number of bins b (Fig.4a), the value in middle of 2 bins values will be scaled to distribute value to each one (Fig.4b)



80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

2	3	4	4	3	4	2	[illegible]
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

0	20	40	60	80	100	120	140	160
21.25								63.75

Histogram of Gradients

(a) Scaled magnitude to bins

(b) Histogram of magnitudes

Figure 6: Histogram of gradients

- Normalizing: Because of the values in histograms heavily depend on the pixel intensity, we have to normalize it, here we gather the histograms of groups above into histogram matrix h with size mxm to normalize $\frac{h}{\|h\|_2}$

- Gain the feature descriptor: Given the image size axb, by conduct the process above, we get the feature the descriptor with given size: $m * m * b * (\frac{a}{n} - 1)(\frac{b}{n} - 1)$

2.1.2 Facial landmarks

In addition to raw pixels and the HOG feature descriptor, we can extract the facial landmarks, informative features extracted by lots of methods up to now ranging from traditional methods (SVM) to modern approaches (CNN models). Here we extract 68 landmarks from facial images (Fig.7)

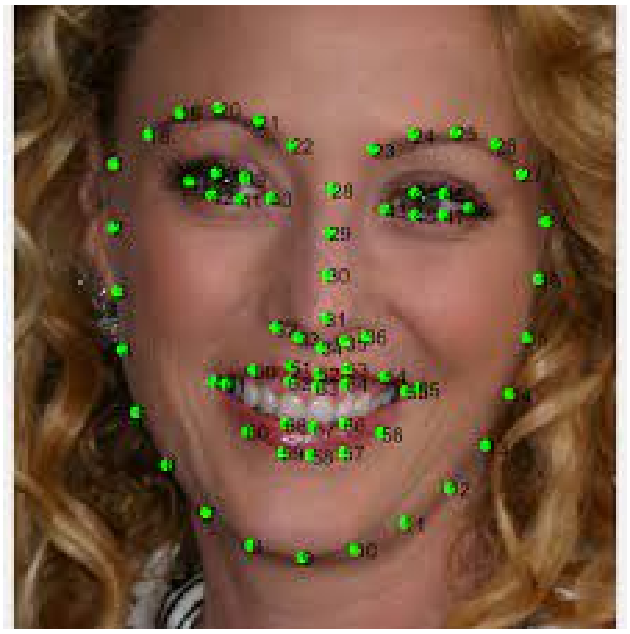


Figure 7: 68 Facial Landmarks

2.2 Traditional Machine Learning

This subsection describes properties and procedure of some traditional ML methods we apply in this report.

2.2.1 K nearest neighbor classifier (KNN classifier)

a.Properties
- Supervised
- Lazy: store training data and label the test data by the K nearest neighbors by a specified metrics
- Direct computation
- Simple

b.Model
- Store data X_{train}, y_{train}
- Major voting to label y_{test} for X_{test}

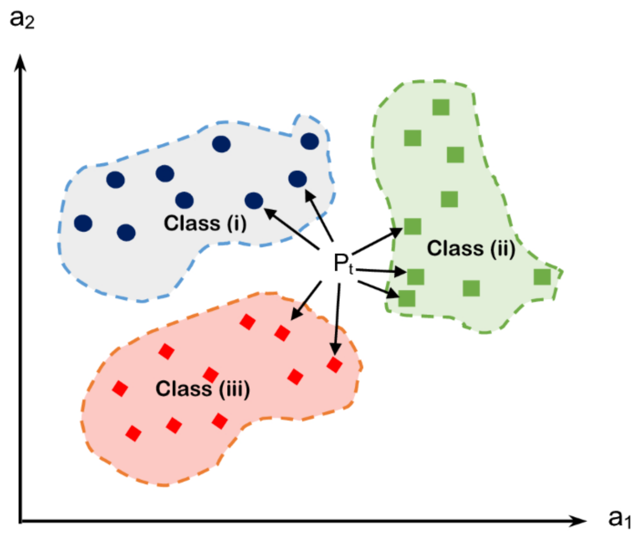


Figure 8: KNN classifier

2.2.2 Gaussian Naive Bayes classifier

a.Properties
- Eager: Assume mutual independence between dimensions of features, construct the hypothesis based on Bayes rule - Direct computation
- Naive assumption

b.Learning
Given training set: data X and label y, for each class c - Bayes rule : $p(y = c|X) \propto p(X|y = c)p(y = c)$, $p(X|y)$ follows Gaussian distribution. - By using MLE or MAP, infer the parameters $mean$ and $sigma$ for training set in each dimension give the mutual depedence between dimensions
- Finding the weights of each classes $p(y)$ which followed categorical distribution, by using the number of training samples in each class.
- Finding the label $y_{test} = argmax(p(X_{test}||y_{test})p(y_{test})$

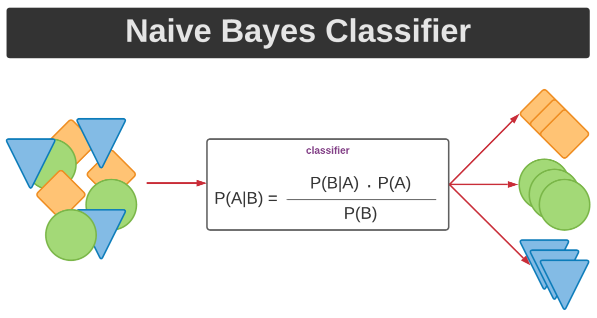


Figure 9: Gaussian Naive Bayes Classifier

2.2.3 Multi-class Support Vector Machine

We apply the Kernel and Soft-margin SVM simultaneously

a.Properties
- Eager: Finding hyperplanes (or lines in 2D data) to separate 2 nearly linearly separable classes(which maybe mapped to another space by kernel functions), meanwhile sacrifice some points. - Local search - Theoretically well explained

b.Learning
Multi-class SVM is many one vs all SVM tasks :

- For training set x_n, y_n, Langarian multipliers, (λ_n, constant C, ϕ mapping of specified Kernel function:

$$\lambda = \arg\max_{\lambda} \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m \Phi(\mathbf{x}_n)^T \Phi(\mathbf{x}_m) \quad s.t: \sum_{n=1}^{N} \lambda_n y_n = 0 \quad and \quad 0 \leq \lambda_n \leq C, \forall n = 1, 2, \ldots, N \tag{1}$$

- For new data points:

$$\mathbf{w}^T \mathbf{x} + b = \sum_{m \in \mathcal{S}} \lambda_m y_m \mathbf{x}_m^T \mathbf{x} + \frac{1}{N_{\mathcal{M}}} \sum_{n \in \mathcal{M}} \left(y_n - \sum_{m \in \mathcal{S}} \lambda_m y_m \phi(\mathbf{x}_m^T) \phi_m^T(\mathbf{x}_n) \right) \tag{2}$$

Knowing how to separate one vs all tasks enable us to classify new data points

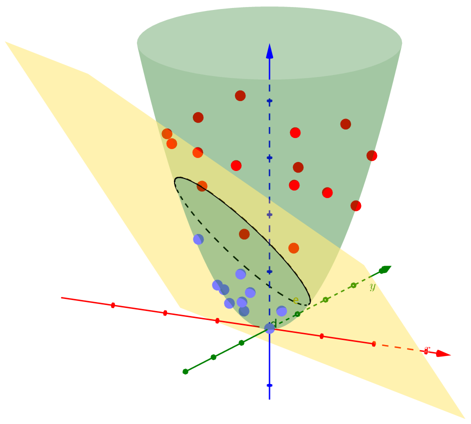


Figure 10: SVM classifier

2.3 Decision trees ensembeling methods

2.3.1 Bagging

Random Forest is example of using bagging ensemble methods to reduce the variance, which help avoid overfitting

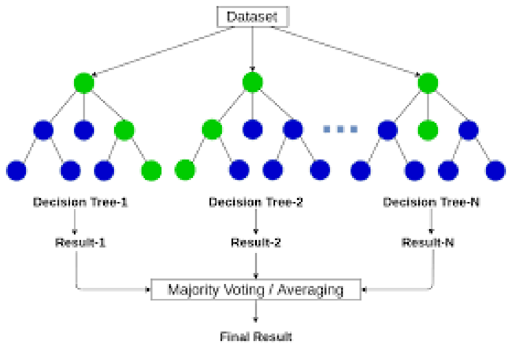


Figure 11: Random forest

2.3.2 Boosting

Boosting ensemble methods increase the bias of ensembling model by sequentially training weak learners, and we aim to a strong model with small loss at last, which has the minimum loss:

a. Adaptive Boosting
The final output is weighted sum of weak learners, with the weight of input in calculating loss for current learner depending on the previous learner (correctly learner will weighted more lightly)

$W_n = \sum_{\mathbf{n-1}} \mathbf{c_i W_i} \quad \textit{with} \quad \textit{confidence} \quad \textit{score} \quad \mathbf{c_i} = \mathbf{f}(\mathbf{Loss}(\mathbf{W_i})), \quad \textit{weak} \quad \textit{learners} \quad \mathbf{W_i}, \quad \textit{and} \quad \textit{strong} \quad \textit{learner} \quad \mathbf{W_n}$

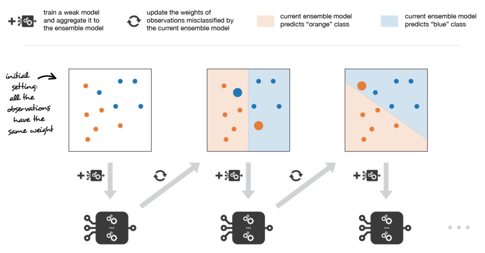

train a weak model and aggregate it to the ensemble model
update the weights of observations misclassified by the current ensemble model
current ensemble model predicts "orange" class
current ensemble model predicts "blue" class
initial setting: all the observations have the same weight

Figure 12: Adaptive Boosting

b. Gradient Boosting
Gradient Boosting instead directly minimizing the specified loss function by calculating the gradient a.k.a pseudo-residual, and fitting a model to residuals with original features, and concatenate the tree models with selected step size for the gradient (residuals).
$W_n = \mathbf{W_{n-1}} - \alpha \mathbf{h_{n-1}} \quad \textit{where} \quad \mathbf{h_{n-1}} = \mathbf{min_h L(r, h)}, \quad \textit{learning} \quad \textit{rate} \quad \alpha \quad \textit{and} \quad \textit{residuals} \quad \mathbf{r} = \frac{\partial \mathbf{L(y, W_{n-1})}}{\partial \mathbf{w}}$

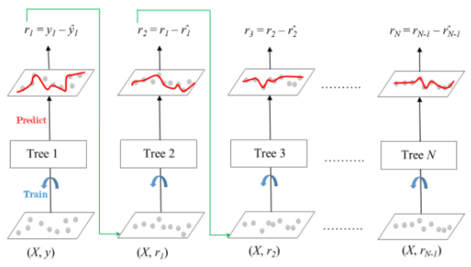


Figure 13: Gradient Boosting

2.3.3 Multi-layer Perceptron

MLP, a fully connected network of nodes between layers by weight W and activation a, learns by back propagation to reduce the loss function. At layer t, With the nodes x_t, weight at layer before W_{t-1}, activation function a_{t-1}, bias b_{t-1}, the operation between layers can be represented:

$$\mathbf{X}_t = \mathbf{a}_{t-1}(\mathbf{W}_{t-1}\mathbf{x}_{t-1} + \mathbf{b}_{t-1}) \tag{3}$$

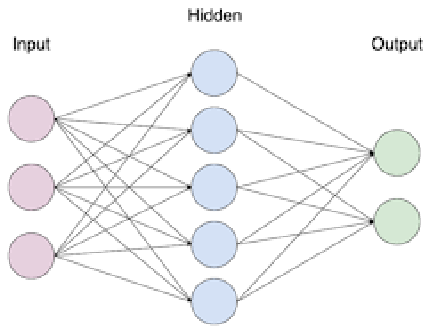


Figure 14: Neural Network

2.4 Modern methods

As shallow neural network mentioned above, now we introduce the overview of modern Computer vision approaches we apply in this work.

2.4.1 Convolutional Neural Network (CNN)

While shallow neural network above use fully connected layers, CNN instead use convolution layers which has following edges over the fully-connected layers:
- **Locality**: Focus on local regions
- **Translation invariance**: Network treats patch similarly regardless of the position in the image
with hidden layer H, input X, weight W, thee layer is formulated as below:

$$[\mathbf{H}]_{i,j} = u + \sum_a \sum_b [\mathbf{W}]_{a,b}[\mathbf{X}]_{i+a,j+b}. \tag{4}$$

In this report, we only apply the traditional CNN models, which consists of convolution layers for feature extraction following by fully connected layers for classification (Fig.15)

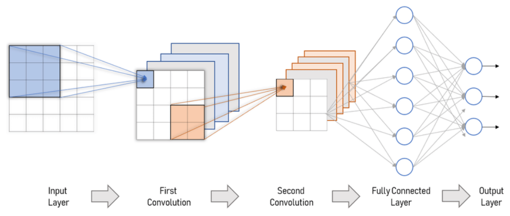
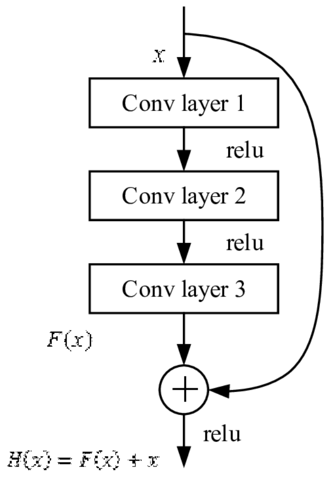


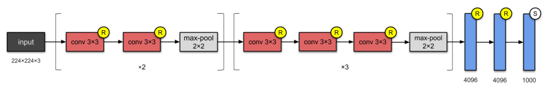
Figure 15: CNN architecture

2.4.2 CNN models

We apply 2 prevailing CNN models: Resnet and VGG, which use skip-connection mechanism and very deep CNN layers (details will be skipped for the sake of conciseness).



(a) Resnet skip-connection



(b) VGG

Figure 16: CNN models

3 Experiments

In this section, we will describe our pipeline for Human facial emotion recognition.

3.1 Dataset

We gather data from 2 datasets: FER2013 contains 28,709 images for training and 7178 for public and private testing, external source CK+ consists of 981 images.

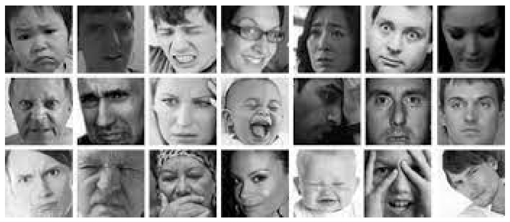


Figure 17: Dataset samples

The dataset comprising 48x48 grey pixel images in 7 classes: (Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral) have the imbalanced number of samples for each class in both training and testing set, which shown below:

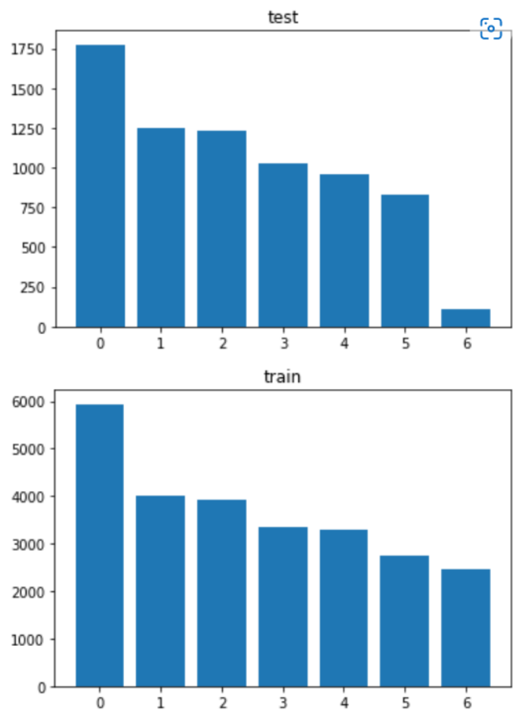


Figure 18: Training and testing data

3.2 Preprocessing

- We convert the orginal images and labels to proper format, scale with MinMaxScaler for applying methods proposed above and store in feather format (.fth) which enable fast read/write process. (Fig.18)
- We also simple check the mean value of image to detect the black images in our dataset and get rid of them.

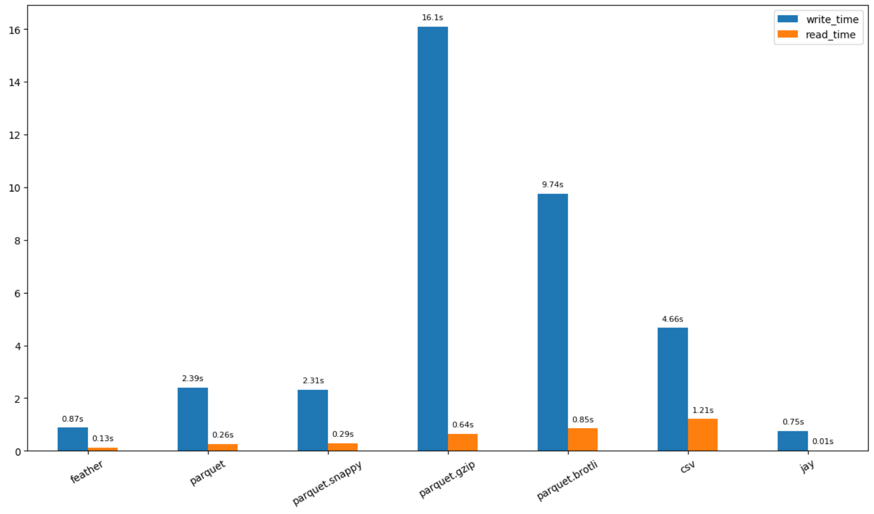


Figure 19: File format write/read speed comparison

- We also stratify-split the original training dataset into training and validation (dev) set, test set is the combination of public and private test of the challenges.
- We can crop image to smaller size (44) to eliminate the background in Modern approach (CNN models)

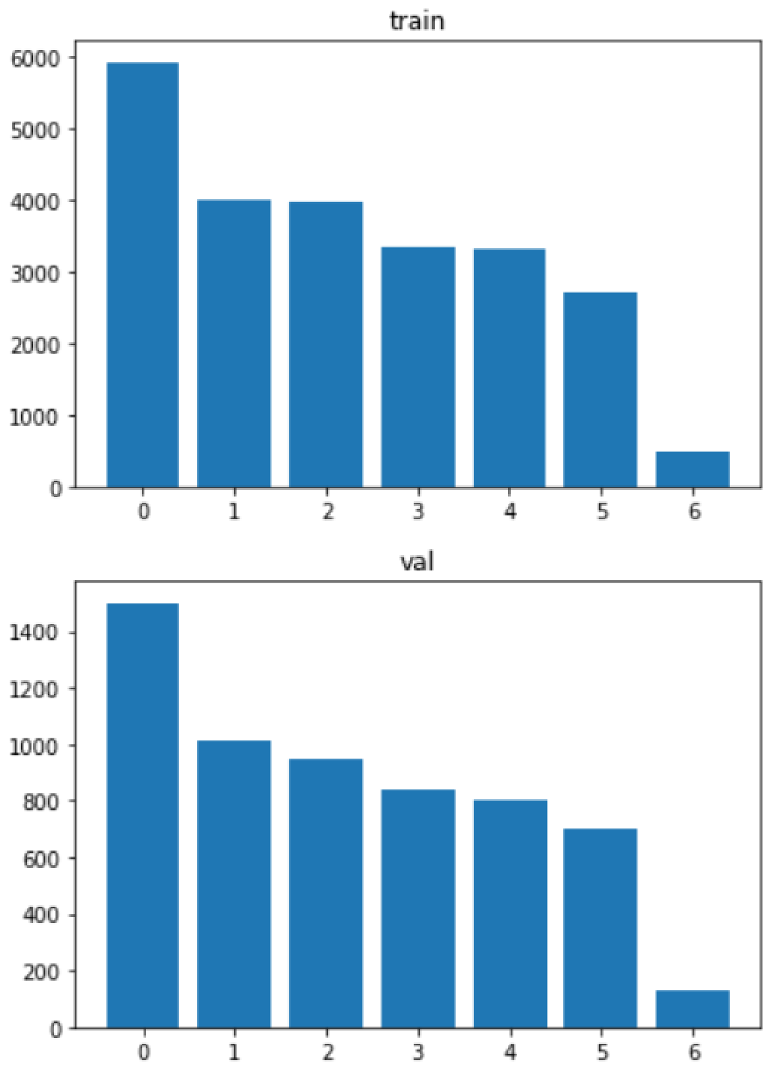


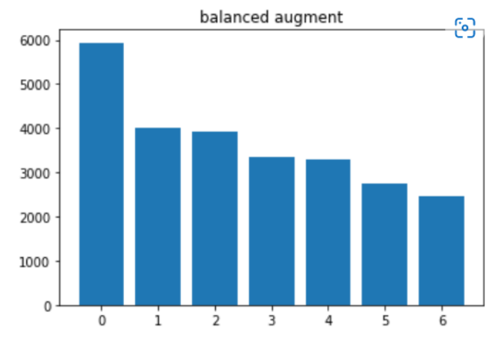
Figure 20: Train and validation split

3.3 Regularization

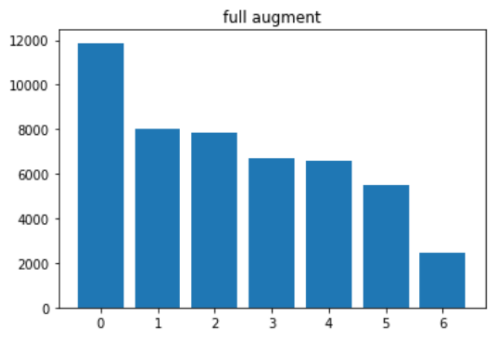
We use several methods to regularize the learning process

3.3.1 Data augmentation

To deal with the imbalance in dataset and make the learning more robust, we have 2 different approaches for augmentation:
- Augment the inferior class
- Augment all the classes



(a) Balanced Augment



(b) Full augment

Figure 21: Augmentation

and we apply following methods:
- Horizontal Flip
- Random Brightness
- Random Contrast

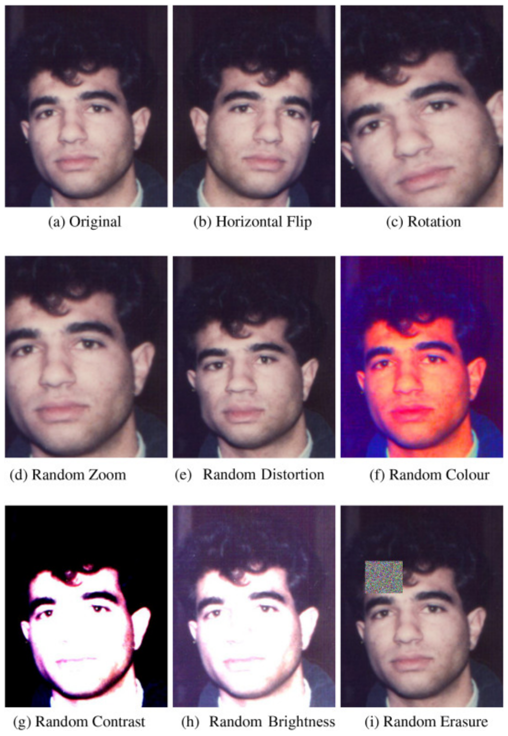


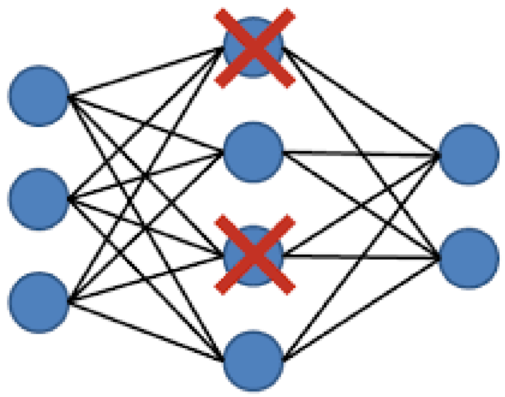
Figure 22: Augmentation methods

3.3.2 Learning regularization

Regularization for training neural networks being prone to overfitting
- Dropout: Randomly turn off effects of some nodes in each iteration
- Weight decay: regularize the weights of models not become to large

$$w \rightarrow w - \eta \frac{\partial C_0}{\partial w} - \frac{\eta \lambda}{n} w$$
$$= \left(1 - \frac{\eta \lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w}.$$

(a) Weight decay



(b) Dropout

Figure 23: Learning regularization

3.4 Training and results

After training and tuning hyperparameters, we compare the results of applied methods with different common metrics for classification tasks.

3.4.1 Task description

- Traditional approach:
- Input: X is concatenation of HOG feature descriptor $\in \mathbb{R}_{900}$ ($8 * 8$ group, 9 bins, $2 * 2$ normalizing block) and 68 2-D landmark points $\in \mathbb{R}_{136}$, hence $X \in \mathbb{R}_{1036}$
- Output: label $y \in \{0, 1, 2, 3, 4, 5, 6\}$
- Hypothesis: Apply traditional ML methods we mentioned above.

- Modern approach:
- Input: X is 2-d matrix $\in \mathbb{R}_{48} * \mathbb{R}_{48}$
- Output: label $y \in \{0, 1, 2, 3, 4, 5, 6\}$
- Hypothesis: CNN models
- Objective function: Cross Entropy Loss

$$L(\hat{y}, y) = -\sum_{k}^{K} y^{(k)} \log \hat{y}^{(k)}$$

Figure 24: Cross Entropy loss

3.4.2 Tuning hyperparameters

- Traditional approach: Due to the high-dimensional data, except MLP (mentioned in modern approach), we select value for the model's parameters and data augmentation methods based on Grid search on landmark feature for the sake of running time, and we will manually check the proximity of selected values for the full features. (Fig.20)

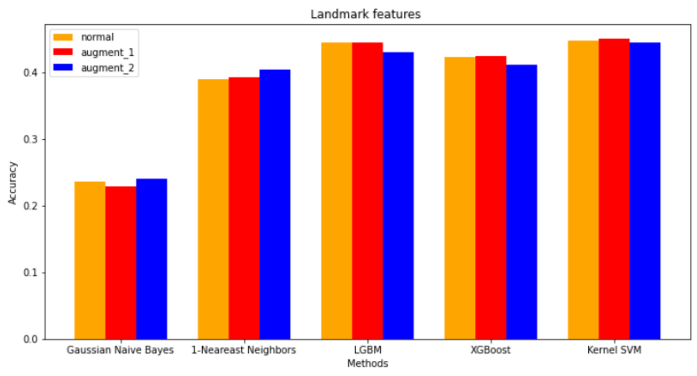


Figure 25: Select data and arguments

As we can observed, in powerful models like SVM, LGBM, XGBoost the balanced augmented data performs best, while in infer-from-data model like Naive Bayes and KNN, the more data the better. So we tune the parameters and select the train data performing best for each models. For full features, we only use the LGBM as Gradient Boosting example to compare with others because of its performance and execution time.
- Modern approach
- We select the balanced augmented training set for comparison with other powerful models
- We select Resnet18 and VGG19 for CNN Models. - The procedure of training our MLP is similar to CNN models. We manually tuning the parameters required for model architecture, optimization algorithm and scheduler, select loss functions,...

3.4.3 Results

By tuning and training process, we come up with final parameters for results comparison.
a. Proof of feature competency
We use the model without tuning to compare the performance of methods on set of feature: raw pixels, HOG descriptor, Landmarks, HOG + Landmarks. (Fig.22)

As displayed, while our extracted features is competent, HOG descriptor plays most significant role, and the combination of them outperform the mere raw pixels

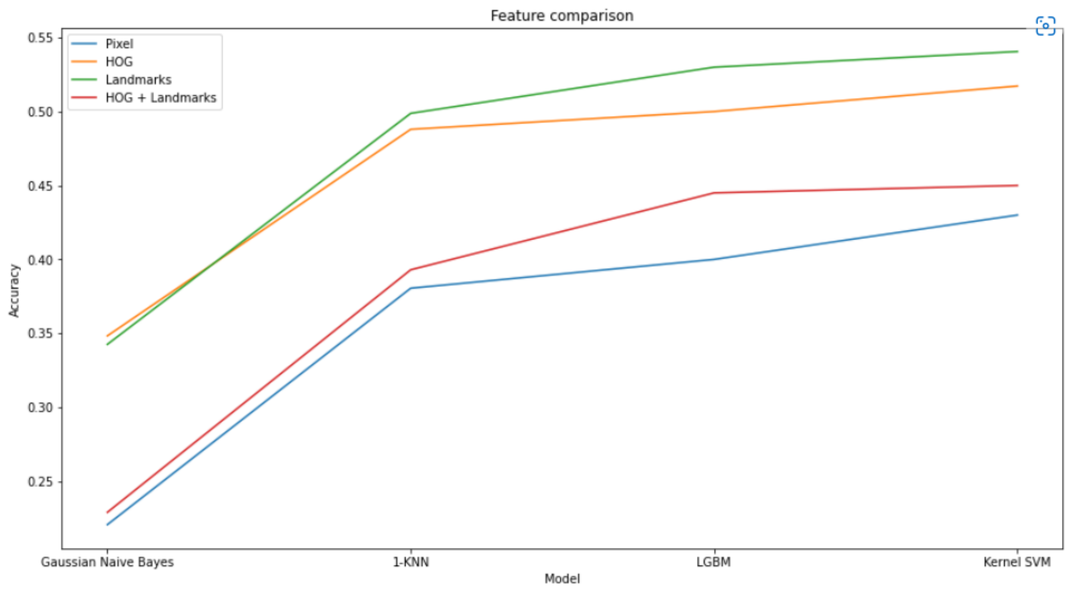


Figure 26: Feature Competency

b. Final tuned models
Below is final models we used with their optimal parameters and hyperpameters obtained by our tuning result

Table 1: Traiditional Methods.

Parameters tuning	
Methods	Tuning result
KNN	neighbors: 1, distance: **Minkowski**
LGBM	estimators: 1500, depth: 9
Kernel SVM	C: 15, kernel: **rbf**

As select amongst hyperparameters space predefined for optimization in neural networks: optimizer $\in$ SGD, Adam, scheduler $\in$ CosineAnnealing, LambdaLr, Constant lr, etc

Table 2: Neural networks.

Hyperparameters tuning	
Methods	Tuning result
MLP	optimizer: **SGD**, learning rate: $1e-1$ -> $1e-4$, scheduler: CosineLearningRate, batch: 32
Pretrained Resnet18	optimier: **SGD**, learning rate: $1e-2$ -> learning rate $1e-5$, scheduler: CosineLearningRate, batch: 64
Pretrained Vgg19	optimizer: **SGD**, start learning rate: $1e-2$ -> $1e-5$, scheduler: CosineLearningRate, batch: 64

c. Training
For both approaches, we tuning the hyperparameters and parameters on validation set combine both train and validation:
- For traditional approaches, we only need to fit the model with selected parameters and hyperparameters, and the training time is displayed below:

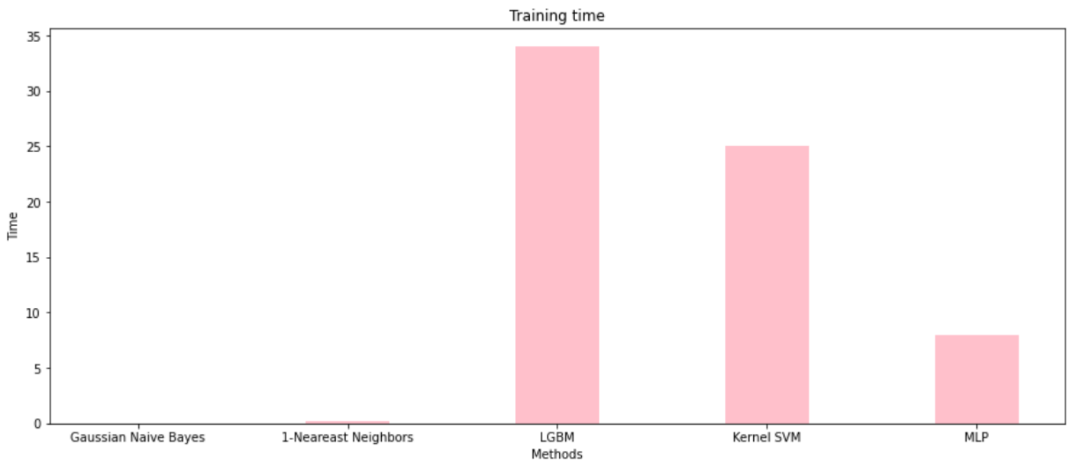
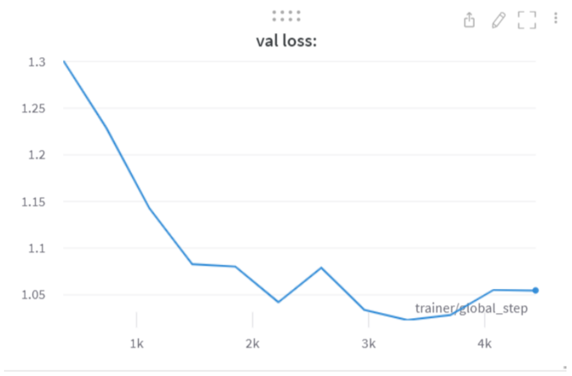
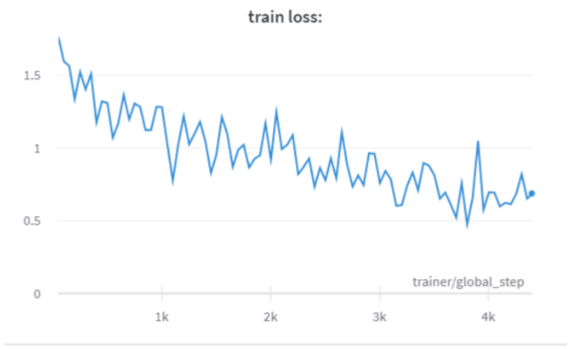


Figure 27: Training time

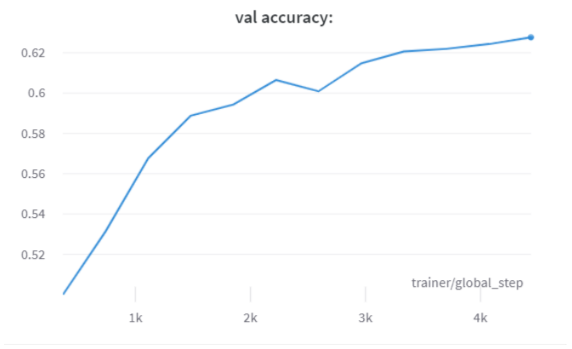
- For CNN models, because of long training time, we fine tune the trained model on the train set and continue training for 20 epochs on the train + validation set for later evaluation with constant learning rate 1e-5. In the training process of train set, we stop "early" after the best validation loss reached, although the validation accuracy keeps increasing.



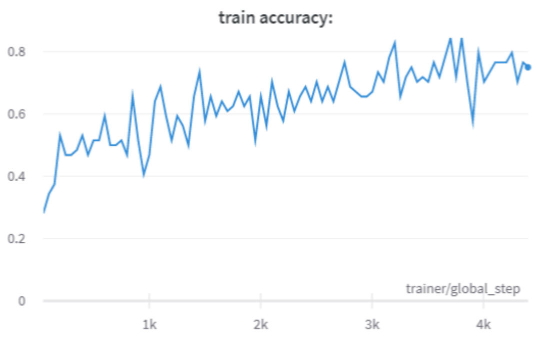
(a) Val loss



(b) Train loss



(c) Val accuracy

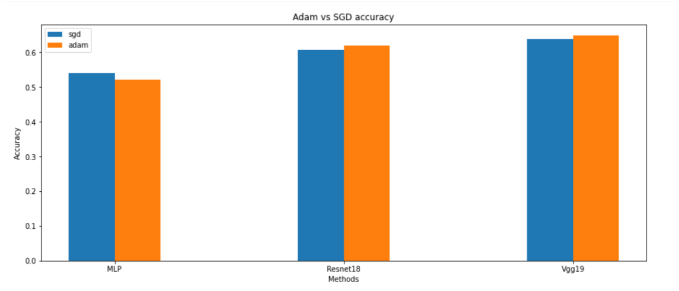


(d) Train accuracy

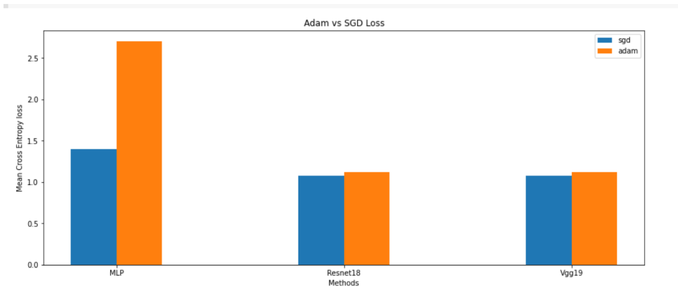
Figure 28: Example of training process

d. Evaluation

- First we reason the fact of using sgd in our neural networks: After training on train set, though reached higher accuracy in CNN models, we opt for SGD with better loss. In Fig. we use the SGD with CosineAnealling scheduler comparing with Adam with constant Lr (most effective LR scheduler for both we observed).



(a) Accuracy



(b) Mean loss

Figure 29: Adam vs SGD

- Pretrained CNN models converge fast in many few epochs, SVM, MLP and Tree ensembling performing decently.

- After finishing training process on the tuned models, we compare methods by different common metrics for classification: Weighted average of **Accuracy, Precision, Recall, F1-score, Support** with 2 different testing methods: Normal test (count number of valid predicted test samples) and 10-crop (Randomly crop image and major vote the result of

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\text{-}score = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Figure 30: Formula for metrics

- **Firstly**, we compare the highest yieled accuracy by models, testing by Normal test methods:

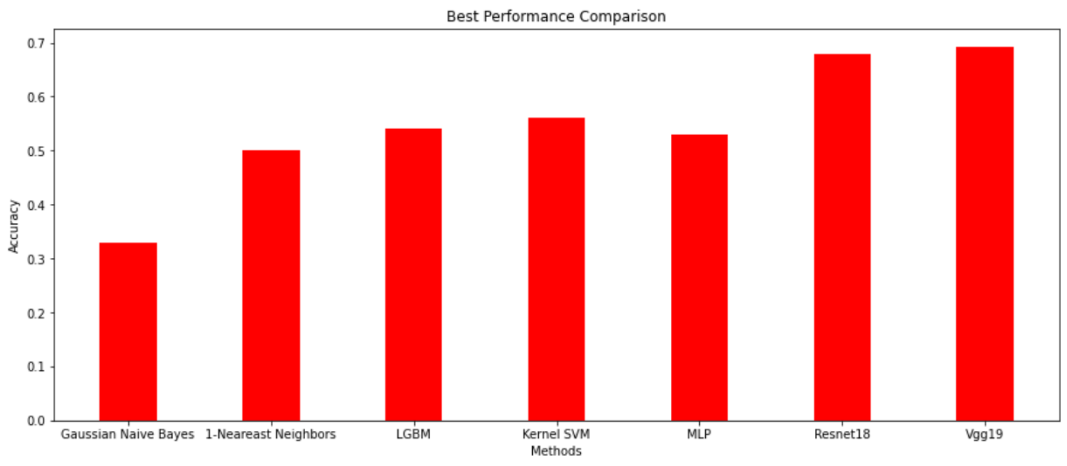


Figure 31: Accuracy of tuned models

- **Secondly**, the 10-crop testing enable us get higher accuracy, we take the model performed best Vgg19 as our instance:

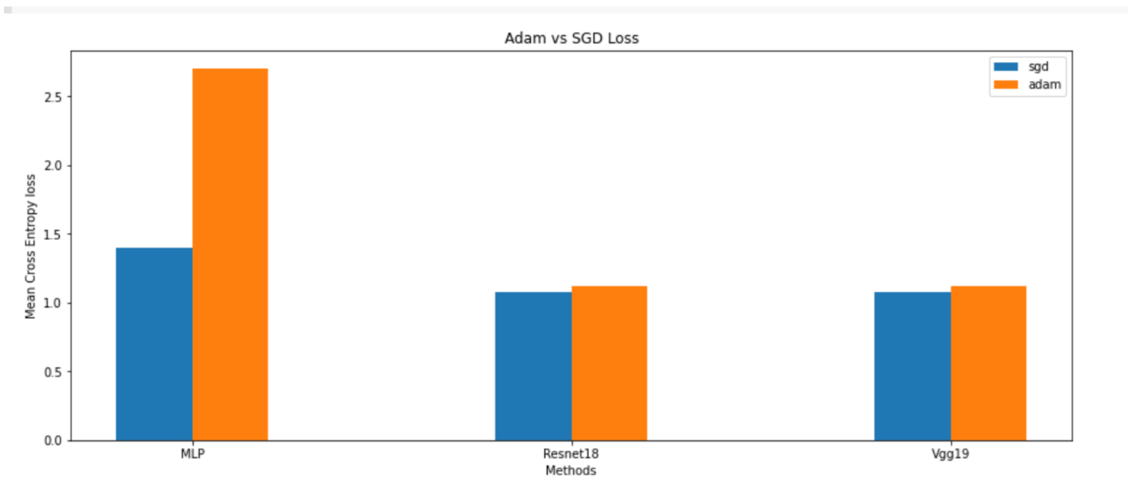
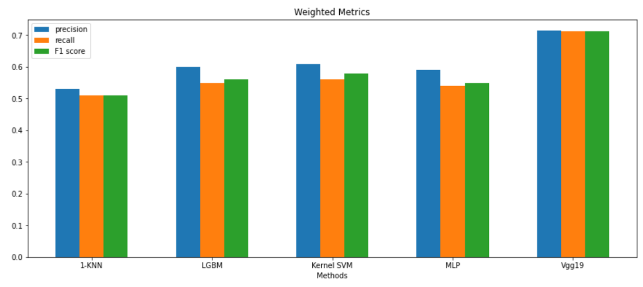
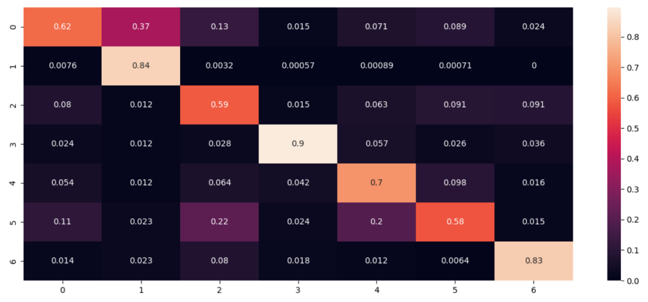


Figure 32: Normal vs 10-crop testing

- Lastly, we display the comparison in different metrics and the confusion matrix, we take the Vgg19 as representative for CNN models (slightly better performance than Resnet18):



(a) Common metrics



(b) Confusion Matrix

Figure 33: Metrics

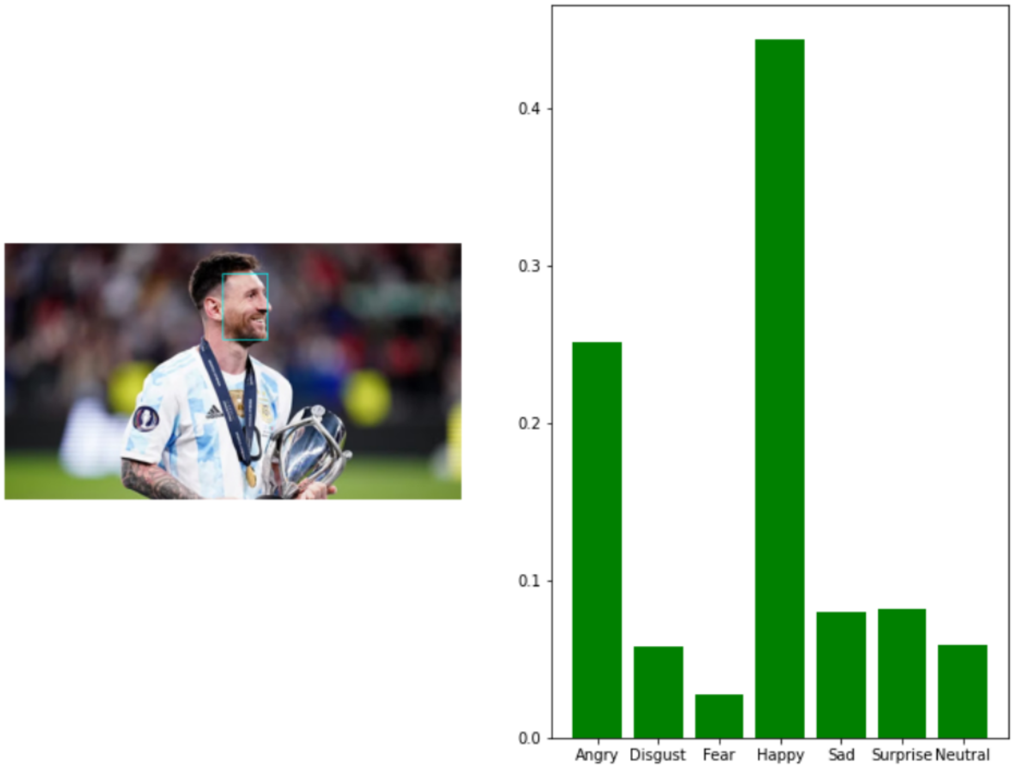
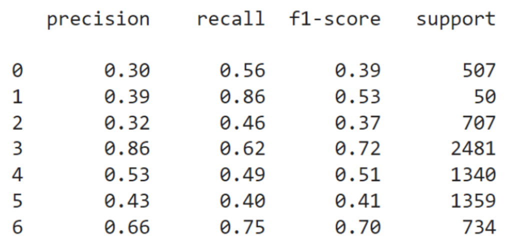


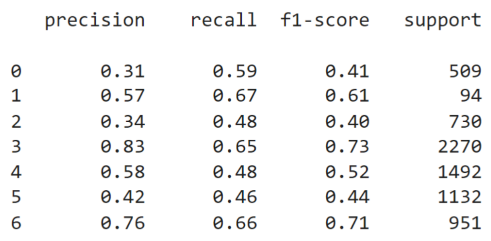
Figure 34: Model inference



```
     precision    recall  f1-score   support

0         0.30      0.56      0.39       507
1         0.39      0.86      0.53        50
2         0.32      0.46      0.37       707
3         0.86      0.62      0.72      2481
4         0.53      0.49      0.51      1340
5         0.43      0.40      0.41      1359
6         0.66      0.75      0.70       734
```

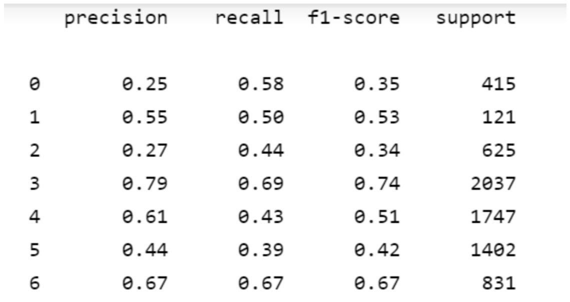
(a) LGBM



```
     precision    recall  f1-score   support

0         0.31      0.59      0.41       509
1         0.57      0.67      0.61        94
2         0.34      0.48      0.40       730
3         0.83      0.65      0.73      2270
4         0.58      0.48      0.52      1492
5         0.42      0.46      0.44      1132
6         0.76      0.66      0.71       951
```

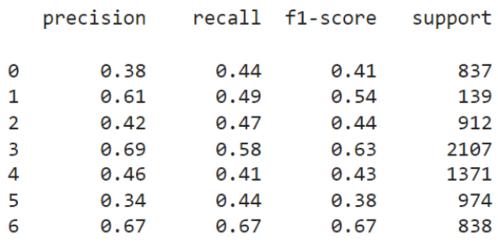
(b) SVM



```
     precision    recall  f1-score   support

0         0.25      0.58      0.35       415
1         0.55      0.50      0.53       121
2         0.27      0.44      0.34       625
3         0.79      0.69      0.74      2037
4         0.61      0.43      0.51      1747
5         0.44      0.39      0.42      1402
6         0.67      0.67      0.67       831
```

(c) MLP



```
     precision    recall  f1-score   support

0         0.38      0.44      0.41       837
1         0.61      0.49      0.54       139
2         0.42      0.47      0.44       912
3         0.69      0.58      0.63      2107
4         0.46      0.41      0.43      1371
5         0.34      0.44      0.38       974
6         0.67      0.67      0.67       838
```

(d) KNN

Figure 35: Details for metrics

- We can observe that the accuracy as well as micro metrics for CNN models better than traditional methods, meanwhile SVM slightly outdoes 2 counterparts Tree Gradient Boosting and MLP. Above is our example for model inference (Fig.34)

4 Conclusions and further work

- With human accuracy around 0.65%, while traditional methods underperform, CNN models surpass human ability. In details, for traditional ML methods, Kernel SVM, MLP, In contrast to simple method like Naive Bayes performing worst (33.3%, Tree Gradient Boosting stand out as powerful methods, and SVM performs best amongst them (56.2%, 2 comparable methods follow (54%), 1-KNN surprisingly gain decent accuracy (50.69%). As expected, CNN models outdo all the traditional methods, with good performance in 2 models Resnet18 () and Vgg19().
- Regularization methods: Augmentation surely benifit models performance,weight decay makes models converge harder, dropout make training slower, but robust to overfitting, and tuning process with validation set help us to gain the models with their best capacity. Besides, the accuracy can increase as we overfit model for a while, which can be comprehended as a random search in weight space in the proximity of the local minimum reached by optimizer.
- The accuracy in test set can be affected by the resemblance between classes and the imbalance of test set - For further experiments, we will try out different methods, including augmentation, models (we only use simple method in this report for introduction), loss function (Triplet loss, Focal loss, Weighted Binary Cross Entropy), tuning methods (Bayesian Optimization), ensembling different methods, etc. In the scope of ML introduction, we believe that our work in this report is sufficiently informative.

References

[1] https://users.soict.hust.edu.vn/khoattq/ml-dm-course/

[2] https://www.kaggle.com/datasets/msambare/fer2013