HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

FACIAL EXPRESSION RECOGNITION

WITH CNN APPROACH

Course : Machine Learning
Course ID : IT 3190E
Instructor : Prof. Than Quang Khoat
Team Members :

No.	Name	Student ID	Email
1	Bui Minh Quang	20214925	Quang.BM214925@sis.hust.edu.vn
2	Le Tuan Anh	20214874	Anh.LT214874@sis.hust.edu.vn
3	Vu Tuan Minh	20210597	Minh.VT210597@sis.hust.edu.vn
4	Dinh Nguyen Cong Quy	20214927	Quy.DNC214927@sis.hust.edu.vn
5	Phan Dinh Truong	20214937	Truong.pd214937@sis.hust.edu.vn

Ha Noi, June 2023

TABLE OF CONTENTS

I. INTRODUCTION

In various fields, there is a growing need to analyze and understand human emotions for various applications such as human-computer interaction, market research, and psychological studies. One approach to address this demand is **Facial Expression Recognition** (FER), which involves automatically identifying and categorizing facial expressions from images or video sequences. In this project, we aim to contribute to the field of FER by applying **Convolutional Neural Network** (CNN) techniques.

This project serves two purposes: to build a robust CNN model which can recognize human expression reliably and to conduct various experiments for the properties of CNN (our main goal). Our main contribution is as following:

- A completed version of CNN that can recognize human emotions by images.
- Optimization of this CNN with grid search on different Learning Rates.
- Experiments on Fine-tuning CNN model.
- Experiments with Data Augmentation and Normalization.
- Implement Focal Loss for more balance classification.

To accomplish this task, we leverage Python modules and libraries that provide pre-trained models and tools for facial expression recognition. Instead of building a CNN from scratch, we will utilize these existing resources, enabling us to focus on analyzing and fine-tuning the results in different settings. By employing these pre-trained models, we can benefit from the wealth of knowledge and expertise captured during their training on large-scale facial expression datasets.

II. DATASET AND RELATED WORKS

Our solution consists of a trained CNN (which is constructed using a part of VGG19) that can take in an input grayscale image whose dimension is 48x48 pixels. Images contain one of 7 types of emotions, namely: ***Anger, Disgust, Fear, Happiness, Sadness, Surprise, and Neutral***. CNN will output a probability distribution and the emotion with the highest distribution will be the answer. This answer can be easily converted to document formats for further usage. This section only goes into details of the solution without explanation, further discussion will be in the following chapters.

2.1. Dataset and Existing Works

2.1.1. Dataset

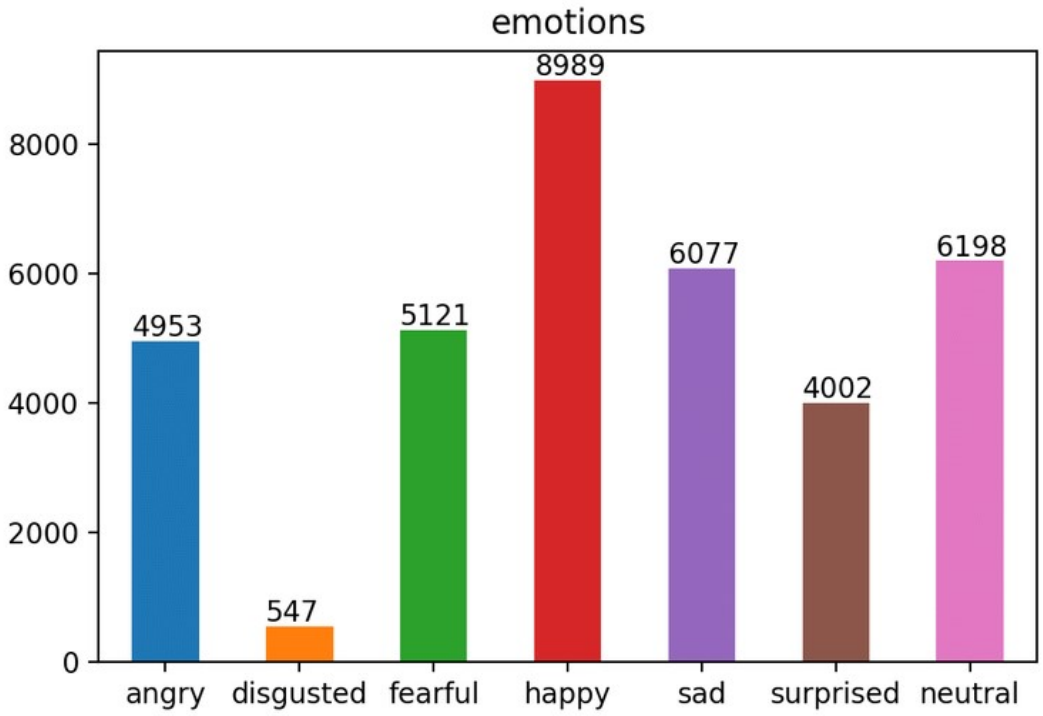
Since its first introduction in ICML 2013, the FER-2013 dataset has been widely used in this field. The FER-2013 dataset consists of approximately 28,700

images (80%) in the training set, nearly 3,600 images (10%) in the public test (can be used for validation) set, and the remaining images (10%) in the private test set (for final evaluation).

Each image in FER-2013 is labeled as one of seven emotions: ***Happy, Sad, Angry, Afraid, Surprise, Disgust, And Neutral***, with "***Happy"*** being the most prevalent emotion (25% labels in the total dataset). The images in FER-2013 consist of both posed and unposed headshots, which are in grayscale and 48x48 pixels.

Note that the images in FER-2013 are both posed and unposed emotions. This makes the model more robust to different situations but also creates a challenge because the data is already limited.

A visual breakdown of FER-2013 dataset:



Data distribution of FER-2013. Image from ResearchGate

For more details on this dataset:

Challenges in Representation Learning: A report on three machine learning contests
Ian J. Goodfellow, Dumitru Erhan, Pierre Luc Carrier, Aaron Courville, et al.

This is a challenging dataset. ***"Happy"*** is the label with the highest number of samples and **"Disgust"** is the class with the lowest number of samples. Also, the ***"Angry"***, ***"Fearful"***, **"Sad"**, and ***"Neutral"*** classes highly resemble each other,

which makes these classes generally hard to distinguish and have low metrics (precision, recall). Human performance on this dataset is estimated at around 65.8%.

Four images from ***"Angry"***, ***"Fearful"***, ***"Neutral"***, and ***"Sad"*** labels private test sets respectively. Even humans have trouble defining the correct emotion.

2.1.2. Existing Works

To our best knowledge, the highest accuracy that can be achieved on the official FER-2013 dataset (without extra training data) is 75.2% with Ensemble CNNs [1]. The top highest accuracy on the test set by using a single Neural Network is around 73%.

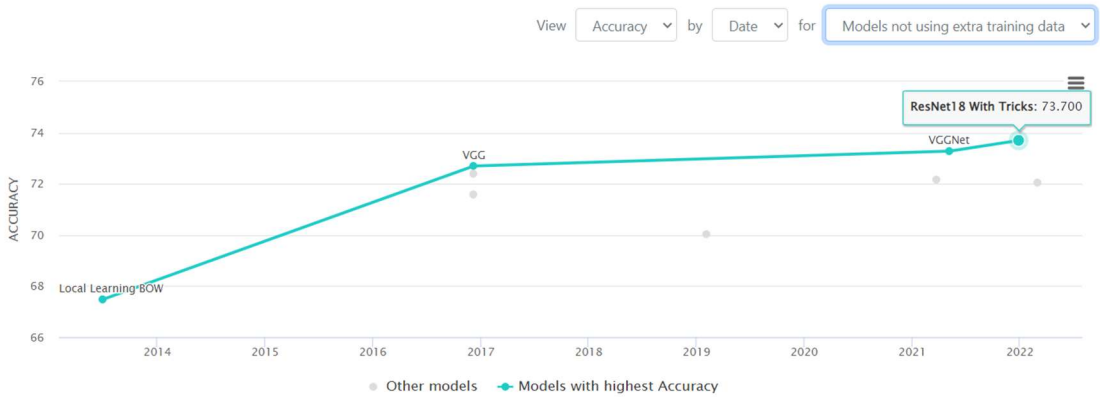


Image from Paper with code.

Generally, the majority of the models for this dataset follow the structure of: CNN blocks (which includes pooling and sometimes batch normalization) and then a number of Fully Connected (FC) layers for classification. Various techniques have been employed such as Discriminative Learning Rate (for Fine-tuning), Additional Batch Normalization after each CNN block and FC layer, ...

Although we are aware that there are existing techniques for improving CNN models. To implement them successfully and more importantly, to explain them thoroughly would require enormous efforts. Therefore, within the scope of this project, we decided to ***build a simple CNN model utilizing Transfer Learning from VGG 19 and perform various experiments on it.***

2.2. Data Preparation

2.2.1. Training, Validation, and Testing

The original FER-2013 dataset has a total of **35,887 samples**. 80% of which are used for Training, 10% are used for Public Test or Validation and 10% final are used for Private Test or Testing. We keep the same proportion for training, testing, and validation.

In all our experiments, we perform stratified hold-out sampling.

2.2.2. Scaling Data and Augmentation

From experimental results, we also implement Min Max Scaler, further discussion will be in the **"Discussion"** section.

In this problem, this dataset does not have any missing labels or data, so it is not necessary to perform any missing value handling.

Since the images are flattened, we need to reshape them back to 48x48 (2D) for further usage of CNN. Also, since VGG 19 only accepts RGB input, we need to convert the dataset from grayscale form to RGB form by cloning the last color dimension, which adds no new information to the datasets.

Data augmentation is also very important. By examining the dataset, we have concluded that the images come in various angles, and zoom settings. Therefore, some additional augmentation to make the limited amount of training data become more diverse is the key factor to increase the training effectiveness.

Four images from the training set for the "Happy" label

The leftmost image is slightly rotated to the right, the second one is horizontally flipped (compared to the first one), the third is without any modification meanwhile the final is zoomed in.

Note that not only do the images from the training set have various properties but the ones from the test set has this variety. To address this problem, we decided to implement the following augmentations: horizontal flip, rotation, shifting (sometimes the images are not in the center), and shearing (this is optional).

2.3. Background

2.3.1. Convolution Layer – The Core Component

In this section, we will present the reason we choose a CNN model for solving this problem – **Facial Expression Recognition**. Recently, the Convolutional Neural Network (CNN) has achieved great success in large-scale image and video recognition. This can be illustrated by Ciresan et al. [2] shows state-of-the-art performance on NORB and CIFAR-10 datasets.

Regarding architecture, not only does CNN learn image feature representations efficiently but it also has a superior performance compared to several handcrafted feature approaches [3]. Models of neural networks show features of a hierarchical representation of data and are based on the calculation of layers with a sequential implementation, with the result of the previous layer being put to the input for the subsequent levels. Each layer is considered to be a single representation level. In addition, the layers are specified by a set of weights.

Furthermore, the input units are linked to the output units by weight as well as a set of biases. Weights are shared locally in convolutional neural networks (CNN), which implies that the weight of each input location is the same. This feature shows some benefits, such as reducing the number of parameters, and time for calculating. Filter form using the weight associated with comparable output.

A convolutional neural network is made up of layers of convolutions. Each convolutional layer has several convolution kernels. A convolution kernel, for example, is used to scan the whole picture horizontally, vertically, and diagonally to generate a feature map (map). In addition, each filter shows effectiveness in detecting some location in images. As the picture is processed, a restricted receptive field is employed for each pixel in the output image, which means that each pixel in the input image uses just a tiny portion of the input image. When we have an insight into CNN architecture, we can see that one of the comparative features of CNN is processing images efficiently, because of applying a filter to an image to extract features.

2.3.2. VGG 19 and VGGNet Model Type

VGG stands for **Visual Geometry Group** (this name is derived from Visual Geometry Group at the University of Oxford where this model is developed). VGGNet was published in 2014 and achieved state-of-the-art in ILSVRC (ImageNet Large Scale Visual Recognition Challenge). Because of the small size of filters (3x3), researchers were able to add many convolutional layers to create a very deep model.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Architectures of different VGGNet models

VGGNet is used to refer to 5 architectures of models which share the same idea of using several convolution blocks before a stack of fully connected layers. Out of 5 network models, VGG 16 (model D) and VGG 19 (model E) are the most common models. Both models achieved decent results.

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

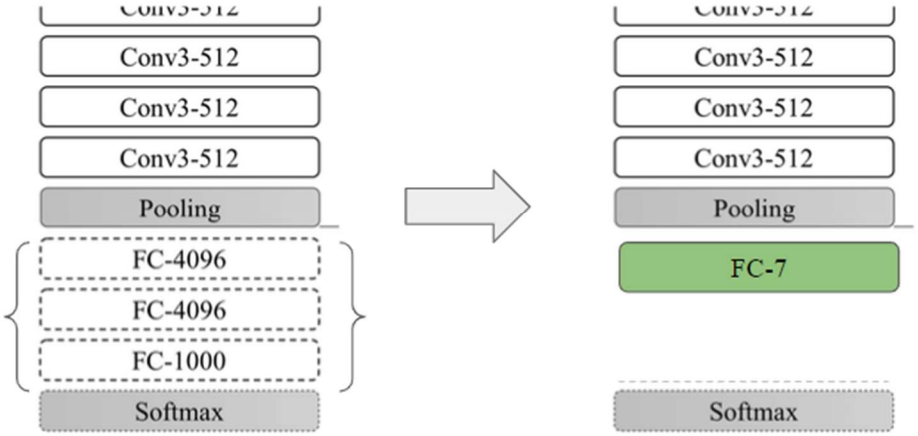
Results of different VGGNet models on ILSVRC 2012 dataset (No scale jittering test set)

The number "19" refers to the number of weight layers (16 convolutional and 3 fully connected layers). Original VGG19 has about 144 million parameters (the majority are on 3 final fully connected layers), which makes training it require enormous computation resources. Therefore, in our project, we experimented with different numbers of fully connected layers and ended at only a fully connected one of 7 neurons for classification.
For more on VGGNet, please refer to [4].

III. PROPOSED CNN ARCHITECTURE

3.1 Architecture

In our proposed approach for **Facial Emotion Recognition**, we leverage the **VGG19 model** as the base architecture. However, we made some modifications to adapt it to our specific task.



Our modification to VGG 19 architecture

Firstly, we **remove the last 3 Fully Connected layers** of the VGG19 model because these layers are designed for classifying 1,000 categories in the ImageNet dataset, which is different from our emotion recognition task. By removing these layers, we can reduce the complexity of the model and make it more suitable for our specific problem.

Next, we **keep the GlobalMaxPooling2D layer** to reduce the dimensionality of the output of the convolutional layers. This makes the model more efficient and easier to train. Next, we added the Dense layer to learn more complex features from the output of the GlobalMaxPooling2D layer. This helps to improve the accuracy of the model. Finally, we **added the Output layer** with 7 neurons to classify the input image into one of the 7 emotions.

By inheriting the VGG19 model and replacing the last three fully connected layers with one fully connected output layer of 7 neurons we create a network architecture that is simpler and more suitable for our specific dataset.

3.2. Optimizer Adam

In this project, we utilized a method named ***Adaptive Moment Estimation*** **or** ***Adam***, which is an extension of the famous ***Stochastic Gradient Descent*** method. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients; the name Adam is derived from adaptive moment estimation.

The method is designed to combine the advantages of two recently popular methods: AdaGrad, which works well with sparse gradients, and RMSProp, which works well in on-line and non-stationary settings.

The Adam Optimizer maintains adaptive learning rates for each parameter by estimating both the first and second moments of the gradients. It is well-suited for training deep neural networks and has become a popular choice in various applications, including computer vision tasks such as facial expression recognition.

$$For\ each\ Parameter\ w^j$$

$$(j\ subscript\ dropped\ for\ clarity)$$

$$\nu_t = \beta_1 * \nu_{t-1} - (1 - \beta_1) * g_t$$

$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2$$

$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

$$\eta : Initial\ Learning\ rate$$

$$g_t : Gradient\ at\ time\ t\ along\ \omega^j$$

$$\nu_t : Exponential\ Average\ of\ gradients\ along\ \omega_j$$

$$s_t : Exponential\ Average\ of\ squares\ of\ gradients\ along\ \omega_j$$

$$\beta_1, \beta_2 : Hyperparameters$$

The key components of the Adam optimizer are as follows:

Learning Rate: Adam adjusts the learning rate adaptively for each parameter. The initial learning rate is set based on domain knowledge or heuristics and can be modified during training. The learning rate determines the step size taken during gradient descent.

Exponential Moving Averages: Adam utilizes exponential moving averages of the gradient and its squared gradient. This helps in maintaining both the first and second moments of the gradients.

Bias Correction: Adam performs bias correction to compensate for the fact that the moving averages initialized at zero are biased towards zero in the early stages of training. This bias correction is crucial for the optimizer to perform accurately.

Momentum: Adam incorporates momentum, which helps accelerate the convergence process. It introduces a new parameter called β1 (typically set to 0.9) that controls the momentum effect. This allows the optimizer to build up velocity in directions with consistent gradients and dampen oscillations in directions with varying gradients.

RMSprop: Adam employs the concepts of RMSprop to normalize the gradients. It divides the first moment of the gradient (mean) by the second moment (uncentered variance) to achieve this normalization. This helps to scale the learning rates for different parameters appropriately.

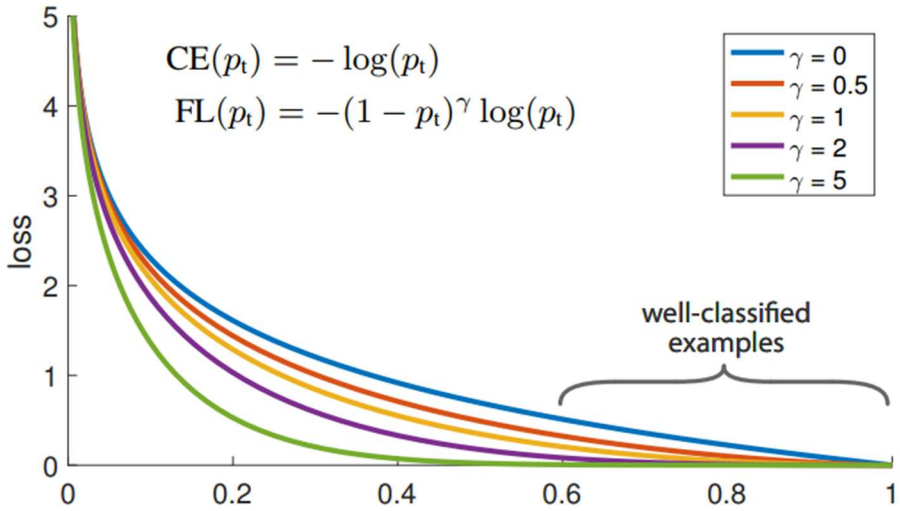
The algorithm updates exponential moving averages of the gradient (v_t) and the squared gradient (s_t) where the hyper-parameters $\beta 1, \beta 2 \in [0, 1)$ control the exponential decay rates of these moving averages. The moving averages themselves are estimates of the 1st moment (the mean) and the 2nd raw moment (the uncentered variance) of the gradient.

3.3. Loss function

There are 2 loss functions we used in our project, namely **"Categorical Cross Entropy"** and **"Focal Cross Entropy".**

Categorical Cross Entropy is a classical choice for multi-classification problems and it is still popular now. This loss function punishes the model equally whenever the probability distribution output by the model is not the same as the one-hot encoded vector of the corresponding sample.

Focal Cross Entropy is a form of Focal Loss function (whose paper was published in 2017, please visit [5])



Focal Loss formula, $(1-p_t)^{\gamma}$ is used to decrease the weight update from samples with high pt value

The Focal Loss function introduces a new term $(1-p_t)$. The p_t closer is to 1 or the bigger the γ becomes, the smaller the first term is and thus the smaller the absolute value of the loss function will become. This new weight term is aimed to reduce the absolute value of loss function with already well-recognized labels (those often score high p_t) and thus reduce the impact of these well- recognized samples on the update of weight.

We used both of them to build different models focusing on overall accuracy and every label accuracy respectively.

3.4. Learning Rate Schedule

In this project, we applied the **Reduce Learning Rate on Plateau** (RLRP), which will reduce the learning rate the longer we train the model and monitor the model's current performance before making a learning rate update. The initial learning rate is set to the value of 0.0001 and for every 7 epochs with no improvement, the learning rate will be reduced by a factor of 0.5. The minimum learning rate, i.e., the lower bound for the update process, is set to be 0.0000001 (1e-7). All other parameters are assigned with their default values.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

All the results, graphs used in this section as well as the notebook where these results, graphs come from are stored in the "assets" folder.

Methodology

During the training process, there are several factors such as the randomness in the flow of training data, weight initialization of fully connected layers, data

augmentation, … Therefore, there is no guarantee that the result will be reproduced exactly.

To mitigate the impact of randomness on our result, we conduct each experiment setting 5 times and calculate the average and other metrics such as variance, min, and max to deliver a final reliable report.

Why not K-fold cross validation?

Training a deep neural network is very costly. Thus, in many works, in model evaluation, authors do not use K - fold cross validation.

Moreover, we have added a fixed random state in our code when splitting data which makes the model more reproducible.

Lastly, repeated experiments are aligned with the principle of statistics for unbiased results.

Experiment Setting

All the experiments follow the mentioned architecture, hyperparameter settings except those that are specifically specified. To be more precise, all models will have **5 unfrozen convolutional blocks** with **weights inherited from VGG 19** and **one fully connected layer**. Optimizers Adam with **learning rate = 0.0001, beta_1 = 0.9, and beta_2 = 0.999**, batch size of 32 samples, same early stopping mechanic are applied to all models in all settings. Data normalization and augmentation are also applied. All of the settings mentioned above are the ones that contribute to our final optimal settings.

Reliability

All results mentioned in this report are in the "assets" folder (which is included in our final project folder).

Our experiments are performed repeatedly and independently to align with the principle for unbiased estimation. Although the sample size is not large enough to satisfy the condition for Central Limit Theorem in statistics, the sample variance is also very low. Therefore, the result provided in this "experiment" subsection can still be used as a reliable estimation for the true value of the population mean and the mean of sample means (in statistics theory, three values are supposed to be equal).

4.1. Accuracy – Optimal Settings

Our model achieves a decent overall accuracy of 69.3% on testing data with a standard deviation of approximately 0.5% (results are shown below), and standard error of mean approx. 0.22%.

Label - precision	Max	Min	Stadard Deviation	Variance	Mean
0	0.6435	0.5514	0.03682	0.0013558	0.60682
1	0.9	0.6939	0.0973	0.0094665	0.78948
2	0.6324	0.5649	0.02711	0.0007347	0.5915
3	0.9045	0.8675	0.01541	0.0002373	0.8857
4	0.5892	0.5628	0.01004	0.0001007	0.5796
5	0.8189	0.7921	0.00973	0.0000947	0.80606
6	0.6266	0.6096	0.00668	0.0000447	0.61646
Label - recall	**Max**	**Min**	**Stadard Deviation**	**Variance**	**Mean**
0	0.6909	0.598	0.0362	0.0013107	0.6408
1	0.6727	0.4909	0.0673	0.0045289	0.59636
2	0.5781	0.457	0.05174	0.0026766	0.52342
3	0.881	0.8432	0.01445	0.0002089	0.85718
4	0.5674	0.5214	0.01931	0.0003728	0.54078
5	0.8075	0.78	0.01165	0.0001356	0.793
6	0.7435	0.7177	0.01047	0.0001097	0.7303
Label - f1-score	**Max**	**Min**	**Stadard Deviation**	**Variance**	**Mean**
0	0.6417	0.5968	0.01624	0.0002637	0.62196
1	0.7033	0.6353	0.02889	0.0008345	0.67276
2	0.572	0.5306	0.02046	0.0004186	0.5532
3	0.8746	0.8618	0.00528	0.0000279	0.87104
4	0.5671	0.5489	0.00795	0.0000631	0.55926
5	0.8075	0.7954	0.00483	0.0000234	0.79938
6	0.673	0.6593	0.00552	0.0000305	0.66852
Accuracy	**Max**	**Min**	**Stadard Deviation**	**Variance**	**Mean**
	0.699	0.685	0.00512	0.0000262	0.6932

*Label 3 or class **"Happy"** is the one with highest **F1-Score, Precision and Recall** due to its large number of samples and also distinguishable features of a Happy person*

The number of samples is not the only factor that affects metrics of a label. Despite the extremely ratio of number of samples compared to label **"1" ("Disgust")**, all the metrics on label **"0" ("Anger"), "2" ("Fearful"), "4" ("Sadness")** and **"6" ("Neutral")** are lower than those of label **"1"**. The reason has been given in the **"Dataset"** section: All 4 later classes share very common elements in the faces that even humans find it hard to classify them correctly.

Another noticeable feature from this report is that due to the very low proportion of data within the dataset, the **"Disgust"** label often has the highest variance, meanwhile, the **"Happy"** label with its highest proportion in the dataset (25.05%) has considerably lower variance compared to other labels.

4.2. Without Data Augmentation

As mentioned in the **"Dataset"** section, data augmentation is a crucial part in improving the quality of the training process. In our project, we decided to perform additional experiments on the model without data augmentation.

As mentioned in the **"Methodology"** subsection, apart from data augmentation, all other settings such as network architecture, hyperparameter settings will remain as specified in the **"Methodology"** subsection.

Generally, we observe a significant decrease in overall accuracy (from 69.3% to 65.9%). The standard deviation also slightly increases from 0.5% to 0.7%, which means the result is still quite stable.

Label - precision	Max	Min	Stadard Deviation	Variance	Mean
0	0.6325	0.57	0.023	0.00055	0.59662
1	0.78	0.6279	0.057	0.00324	0.70858
2	0.5945	0.53	0.025	0.00063	0.56032
3	0.85	0.7978	0.022	0.00049	0.82762
4	0.56	0.5067	0.021	0.00043	0.53736
5	0.8	0.7725	0.012	0.00014	0.7845
6	0.61	0.5447	0.029	0.00086	0.57614
Label - recall	Max	Min	Stadard Deviation	Variance	Mean
0	0.6	0.5071	0.038	0.00147	0.56086
1	0.5636	0.4909	0.026	0.00066	0.52836
2	0.543	0.4238	0.046	0.00216	0.49112
3	0.8776	0.84	0.015	0.00023	0.85902
4	0.5576	0.53	0.011	0.00012	0.5466
5	0.8	0.76	0.015	0.00023	0.774
6	0.66	0.6016	0.024	0.00059	0.63296
Label - f1-score	Max	Min	Stadard Deviation	Variance	Mean
0	0.59	0.5486	0.016	0.00025	0.57618
1	0.63	0.551	0.033	0.00107	0.60504
2	0.5472	0.4949	0.019	0.00037	0.52176
3	0.85	0.8358	0.007	0.00004	0.8447
4	0.5529	0.5309	0.008	0.00006	0.54072
5	0.79	0.77	0.009	0.00008	0.78018
6	0.64	0.5781	0.025	0.00064	0.60438
Accuracy	Max	Min	Stadard Deviation	Variance	Mean
	0.665	0.651	0.007	0.00005	0.6592

*All **F1-Scores** for different labels also decrease from 0.02 to 0.06*

4.3. Without Data Normalization

It is a common practice when training neural networks to normalize data for faster convergence speed and better performance. In some scenarios, Normalizing data even helps reduce the impact of outliers on the data.

As mentioned in the **"Methodology"** subsection, apart from data normalization, all other settings such as network architecture, hyperparameter, and even data augmentation settings will remain as specified in the **"Methodology"** subsection.

In this experimental setting, we decided to test whether data normalization using **MinMaxScaler** help with training our model.

The accuracy metric is 68.7% with a 1% standard deviation. This testing result is not enough to conclude decisively whether the model performs worse on unnormalized data however, the standard deviation is larger, which indicates that the result when data is unnormalized may be more unstable.

4.4. Focal Loss

In this experiment, we changed the loss function from categorical cross entropy to its focal version.

As mentioned in the **"Methodology"** subsection, apart from data normalization, all other settings such as network architecture, hyperparameter and even data augmentation settings will remain as specified in the **"Methodology"** subsection.

We experimented with the case $\gamma=2$ and $\gamma=5$. Two models achieved mean accuracy of 68.7%, 67.0%, and standard deviation of 1% and 0.7% respectively.

In both cases, we do not observe any consistent improvement in metrics (precision, recall) for 4 hard labels **"0", "2", "4",** and **"6"**. This can be explained as our model does not overfit to easy class (**"3" – "Happy"** or **"5" – "Surprise"**). In other words, the four classes are inherently difficult, and this challenge cannot be overcome by **Focal Loss**.

Also, higher gamma values may result in overemphasis on difficult labels. In the FER-2013 dataset, easy labels ("**3" – "Happy"** or **"5" – "Surprise"**) still contribute to approximately 42% of the dataset. In the case of $\gamma=5$, f-1 score for label **"5"** decreases from 0.79938 to 0.75678.

4.5. Optimizer and batch size experiments

All results in this section are from the validation set (the selection of hyperparameters cannot be based on the result of the test set because the selected final model may be overfitting to the test data).

All the settings which are not specified in this section follow the configuration mentioned in the **"Methodology"** subsection.

Batch size

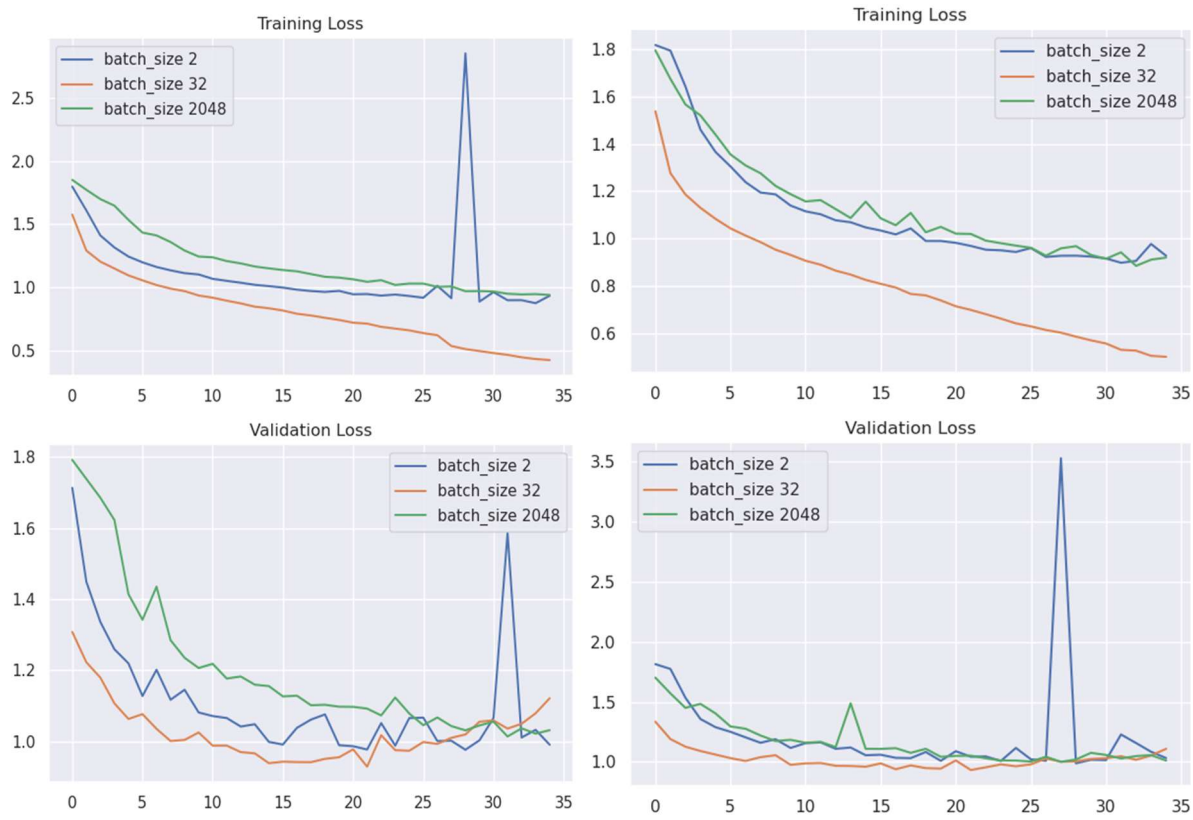
In our experiments, we perform training and validation on 3 batch sizes - 2, 32, and 2048. Generally, large batch sizes can take advantage of GPU multi-processing for faster parallel computation.

Smaller batch size causes more instability during the training process.

A spike in loss graph in training or validation loss is not uncommon for the very small batch size of 2 meanwhile, the largest batch size of 2048 has a smoother curve. In theory, the largest batch size of 2048 should give the smoothest curve of training loss and the smallest batch size of 2 should give the most fluctuating curve.

However, in our project, there are a variety of random factors mostly caused by the random data augmentation - which can make the global optimal point for the training dataset unstable. Therefore, in the right graph, the training loss function for the largest batch size seems to fluctuate more compared to the one from the batch size of 32.

The final choice of 32 batch size comes from various sources and our own experiments with validation loss (which is included in the asset folder).

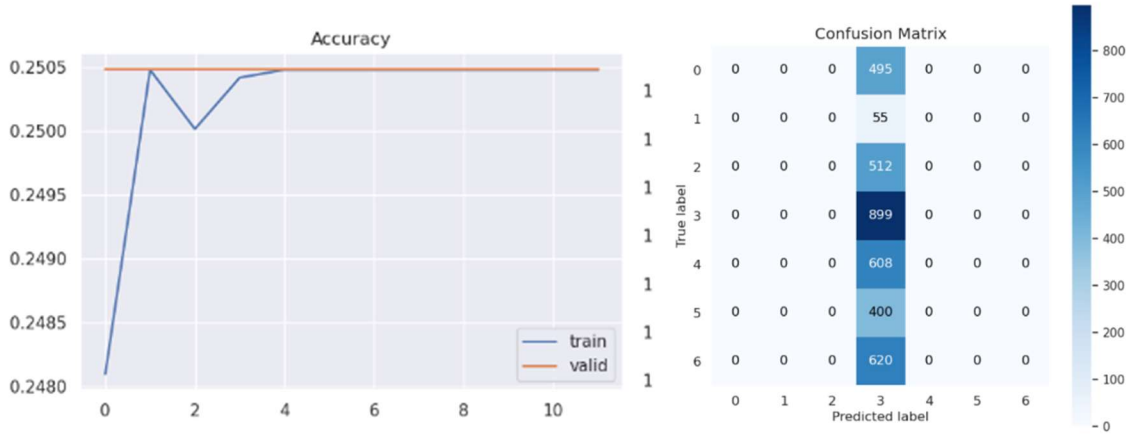


Result from 2 runs. A smaller batch size may cause unstable learning

Learning Rate

From our experiments, the optimal Learning Rate is around 0.0001. A learning rate that is around 0.001 (10 times larger) will lead the model to overfit to **"Happy"** labels (stuck at 0.2505 validation accuracy, which is the proportion of **"Happy"** labels).

The following graph is from the model with learning rate = 0.001.

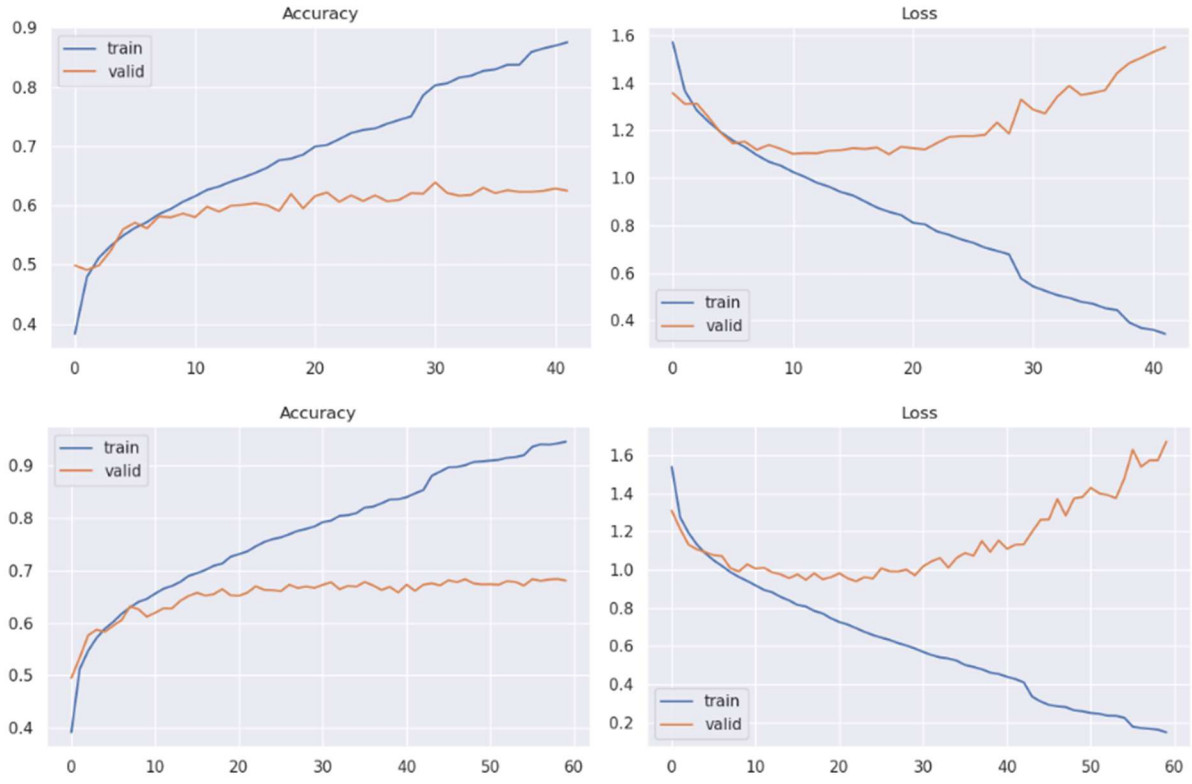


Such a high learning rate makes the weights in the model keep fluctuating around in the parameter space, overshooting the optimal point and to the point that is overfitting to the **"Happy"** label.

4.6. Fine-tuning experiments

It is common practice when inheriting pre-trained CNN model to freeze the first few CNN blocks and train on the rest of CNN. In our experiments, the accuracy of the model in both training and testing increases with the number of unfrozen blocks. This trend is consistent among all 10 experiment trials (5 trials for the first 3 frozen blocks and 5 for only the first CNN block).

While more trials should be performed to conclude whether the first block of convolutional layers should be frozen or not, consistency in test results indicates that freezing 3 or more (up to 5) blocks will make the model underfitting to our specific dataset set.



Top images are from 3 frozen blocks and bottom images are from only 1 frozen block

4.7. Weight Initialization

This is also an important aspect of Transfer Learning. Based on the result of various settings, we discovered that weight initialization plays a crucial role in our network to achieve a decent performance.

In this experiment, as specified in the previous subsection, except for randomly initiated weight, all other hyperparameters are as specified in the "Experimental Setting" subsection.

The mean accuracy is around 68% with a standard deviation of 0.6%. This is lower compared to the optimal setting (with weight initialization) with an accuracy of 69.3%. This can be explained as the parameters from the VGG 19 model are well trained on various images from the ImageNet dataset and can act as a good initial starting point so that the parameters in our final models can move toward a better optimal convergence point.

V. UNSOLVED PROBLEM AND FUTURE WORK

5.1. Overall accuracy can be improved more

One of the major challenges in facial expression recognition using Convolutional Neural Networks (CNN) is the overall accuracy, which tends to be low when the amount of available data is limited. The performance of CNN models heavily relies on the availability and quality of the training data. Insufficient data poses a significant obstacle to achieving high accuracy in facial expression recognition systems.

Just adding more data samples will not necessarily solve the problem. As mentioned in the **"Dataset"** section**, "Anger", "Fear", "Sad", and "Neutral"** labels have really similar face features (especially those from unposed images). We suggest more features included and even more complex models to solve this problem. For example, in practice, many images will come with a description of the situation. We can use the output probability distribution of this CNN model combined with weighted votes from a large language model to correctly identify the emotion.

5.2. More work on Fine-tuning

Fine-tuning is a crucial aspect of training Convolutional Neural Networks (CNNs) for facial expression recognition. It involves adapting a pre-trained model to perform a specific task by adjusting its parameters based on a new dataset. One emerging technique used in fine-tuning is discriminative learning rate, which allows for differential learning rates across different layers of the network.

Within the scope of this project, we have done only a few experiments involving freezing CNN blocks. We would suggest more thorough investigation through different layer settings.

5.3. More work on Tuning Hyperparameter

Due to the limited computational resource, we have only experimented with a limited range of combinations of hyperparameter settings that is based on our specific domain knowledge. We would suggest more experiments with different configurations of fully connected layers. Also, the hyperparameter values of the optimizer and focal loss function are tunable and more work should be done on tuning those hyperparameters.

VI. CONTRIBUTION

Our group consists of 5 members. The detailed contribution is as follows:

- Quang: Task distribution, code Skeleton Modification, Focal Loss Implementation and Experiments.
- Quy: Code for extracting mode, research about Adam Optimizer, Data Insights.
- Minh: Research about CNN, Statistical Work, VGG19 Theory.
- Truong: Code for Grid Search, Grid Search Experiments, Report.
- Anh: Related Work, Fine-Tuning Experiments, Data Experiments.

VI. SOFTWARE AND PACKAGE UTILIZATION

6.1. Scikit-Learn Module (Data Preprocessing)

This module assists us greatly in data preprocessing. The main methods used are splitting dataset into training and test dataset and classification report which helps getting accuracy for every label. For more details of this module, please visit: https://scikit-learn.org/stable/getting_started.html

6.2. Keras (CNN)

This module is used for building ANN, it offers users an easy way to build and adjust ANN. For more details of this module, please visit: https://keras.io/getting_started/

6.3. Other Modules

Numpy: a Python module that is good for working with matrices . For more details of this module, please visit: https://numpy.org/

Matplotlib, Seaborn: a useful Python module for drawing graph results. For more details of this module, please visit:
https://matplotlib.org/
https://seaborn.pydata.org/

6.4. Code Skeleton

Kaggle: In this project, we import the source code of the VGG19 model from Kaggle with some modifications including: Grid Search; Re-Splitting the data; Performing some experiments with various settings, which will be discussed in the later section.

For more details about this original source code, please visit: https://www.kaggle.com/code/enesztrk/facial-emotion-recognition-vgg19-fer2013

REFERENCES

[1] Christopher Pramerdorfer, Martin Kampel (2016), "Facial Expression Recognition using Convolutional Neural Networks: State of the Art", arXiv:1612.02903.

[2] Dan Cireşan, Ueli Meier, Juergen Schmidhuber (2012), "Multi-column Deep Neural Networks for Image Classification", arXiv:1202.2745.

[3] K. Nguyen, C. Fookes, A. Ross, and S. Sridharan, "Iris Recognition with Off-the-Shelf CNN Features: A Deep Learning Perspective," IEEE Access, vol. 6, pp. 18848 -18855, 2017.

[4] Karen Simonyan, Andrew Zisserman (2014), "Very Deep Convolutional Networks for Large-Scale Image Recognition", arXiv:1409.1556.

[5] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár (2017), “Focal Loss for Dense Object Detection”, arXiv:1708.02002.