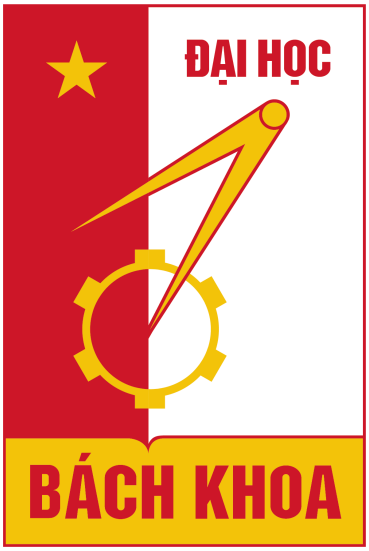
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY
FACULTY OF COMPUTER SCIENCE




# Facial Emotion Recognition

**Course: IT3320E - Deep Learning**
**Class Code: 161861**

**Group's members:**
Nguyen Thai Anh Minh - 20235605
Dang Trung Anh - 20235583
Pham Duc Duy - 20235588
Bui Cong Minh - 20235601

**Supervisor: Ph.D. Than Quang Khoat**

# Table of Contents

# 1 Introduction

Psychologists, neuroscientists, and computer scientists have long been interested in the difficulty of understanding human emotions based on facial traits, which is at the heart of good social interactions and communications. With the introduction of deep learning approaches, there was a breakthrough in the field of emotion identification using face photographs. Deep learning approaches have demonstrated outstanding performance in recognizing emotions from facial characteristics by replicating the structure and function of the human brain. Furthermore, deep learning systems can learn from complicated patterns, extract features from enormous datasets, and apply learning skills to fresh data. Recognizing emotions using face characteristics using deep learning algorithms has emerged as an interesting research topic. Its applications encompass robots, mental health applications, and human-computer interface.

The FER-2013 dataset was introduced in the ICML 2013 Challenges in Representation Learning and has since become a standard benchmark for evaluating facial expression recognition models. The dataset consists of 35,887 grayscale facial images, each with a resolution of $48 \times 48$ pixels. All images are labeled with one of seven basic emotion categories: Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral.

Figure 1: FER-2013

FER-2013 images were automatically collected from the internet using a face detection algorithm, resulting in significant variability in pose, illumination, facial appearance, and image quality. This makes the dataset particularly challenging and representative of real-world conditions. Additionally, the dataset exhibits class imbalance, with some emotions (such as Disgust) being significantly underrepresented compared to others (such as Happy or Neutral).

The dataset is provided in CSV format. Each image is represented as a sequence of pixel intensity

values ranging from 0 to 255. The dataset contains seven emotion categories: Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral. The dataset is divided into three subsets: Training, PublicTest, and PrivateTest, which are commonly used for training, validation, and final evaluation, respectively.

Due to its relatively small image size, grayscale format, and challenging variability, FER-2013 serves as an effective testbed for developing and evaluating both traditional convolutional neural networks and modern transfer learning approaches using pretrained deep models. Despite its simplicity in resolution, achieving high performance on FER-2013 remains non-trivial, making it a valuable dataset for benchmarking facial expression recognition methods.

In this project, we investigate facial emotion recognition on the FER-2013 dataset using two complementary approaches: (1) a custom-designed convolutional neural network trained from scratch, and (2) a transfer learning approach based on ResNet18. We analyze the impact of data preprocessing, class imbalance handling strategies, and optimization techniques, and compare their effects on model performance.

# 2 Theoretical Background

## 2.1 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are deep learning models designed to process data with a grid-like topology such as images. They are the foundation for most modern computer vision applications to detect features within visual data.

Unlike traditional fully connected neural networks, CNNs exploit spatial locality and parameter sharing, which significantly reduce the number of parameters and improve learning efficiency.

**Key Components:**

- Convolutional Layers
- Pooling Layers
- Activation Functions
- Fully Connected Layers

### 2.1.1 Convolutional Layers

Unlike a fully connected neuron, each convolutional neuron(filter) is only locally connected to the input data.

The convolutional neuron slides from left to right and from top to bottom of the input data block and computes togenerate an activation map.

The depth of the convolutional neuron is **equal** to the depth of the input data block.

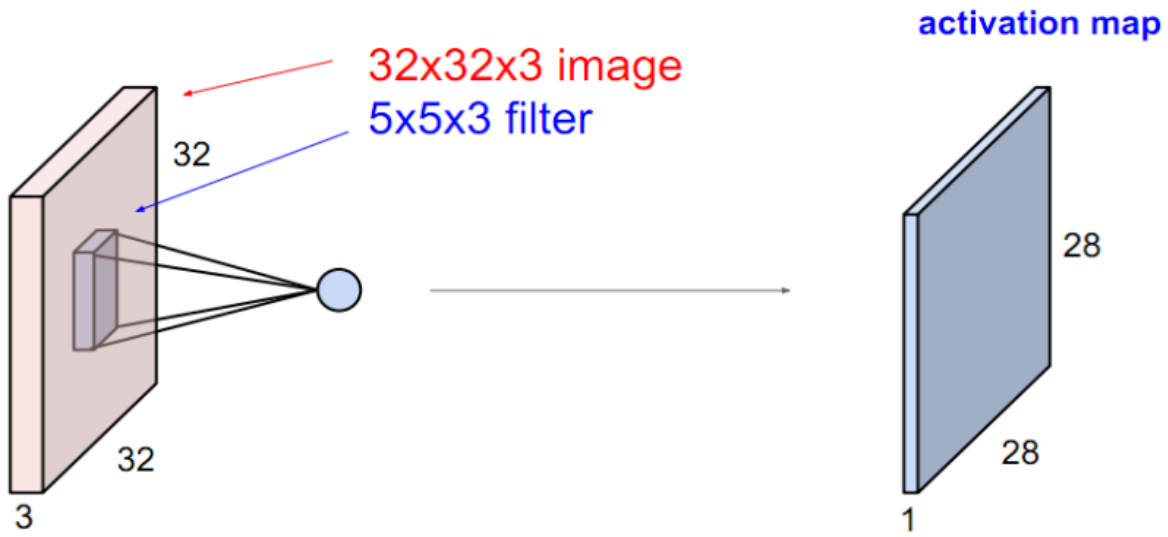



Figure 2: Example of Convolutional layers

$$\text{Output size} = \frac{N - F}{S} + 1$$

where:

- $N$: input size
- $F$: filter (kernel) size
- $S$: stride

To **preserve the output size**,usually add padding borders with zeros (zero padding).
Example: input size 7x7 (N), neuron size 3x3 (F), stride 1, padding width 1.

$\Rightarrow$ The output size is $7 \times 7$

**A convolutional neural network (CNN)** is a sequence of convolutional layers stacked on top of each other, and they are interspersed with activation functions (e.g. ReLU)

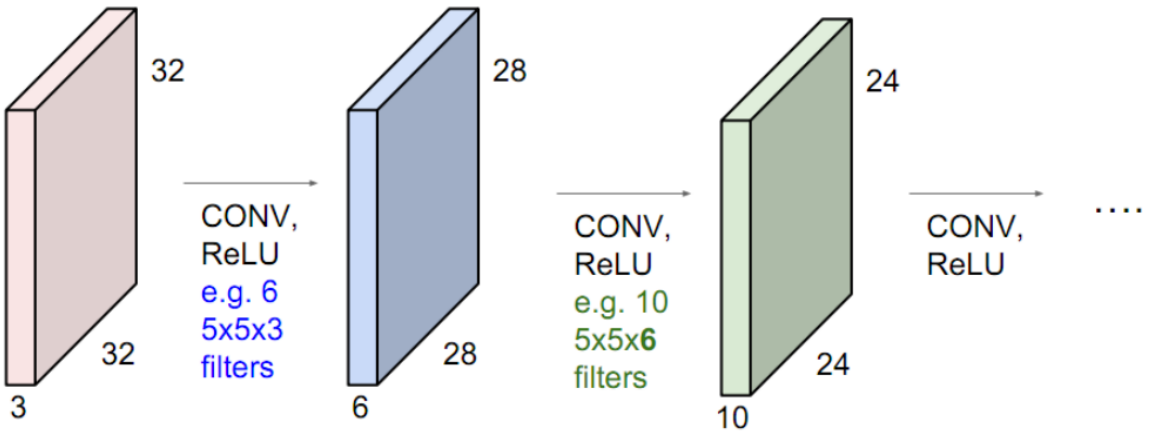



Figure 3: CNN

### 2.1.2 Pooling Layers

Pooling is a down-sampling technique.

- Helps reduce the block data resolution to reduce memory and computation consumptions.
- Work independently on each activation map.
- The max pooling layer helps the network to make the representation become invariant to the small (local) translation, or deformation (deformation-invariant) of the input data.

For a feature map with dimensions $n_h \times n_w \times n_c$, the dimensions of the output after a pooling layer are:

$$\left(\left\lfloor \frac{n_h - f}{s} \right\rfloor + 1\right) \times \left(\left\lfloor \frac{n_w - f}{s} \right\rfloor + 1\right) \times n_c$$

where:

- $n_h$: height of the feature map
- $n_w$: width of the feature map
- $n_c$: number of channels in the feature map
- $f$: size of the pooling filter
- $s$: stride length

#### 2.1.2.1 Max Pooling Layers

Max pooling selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map. Max pooling layer preserves the **most important features** (edges, textures, etc.) and provides better performance in most cases

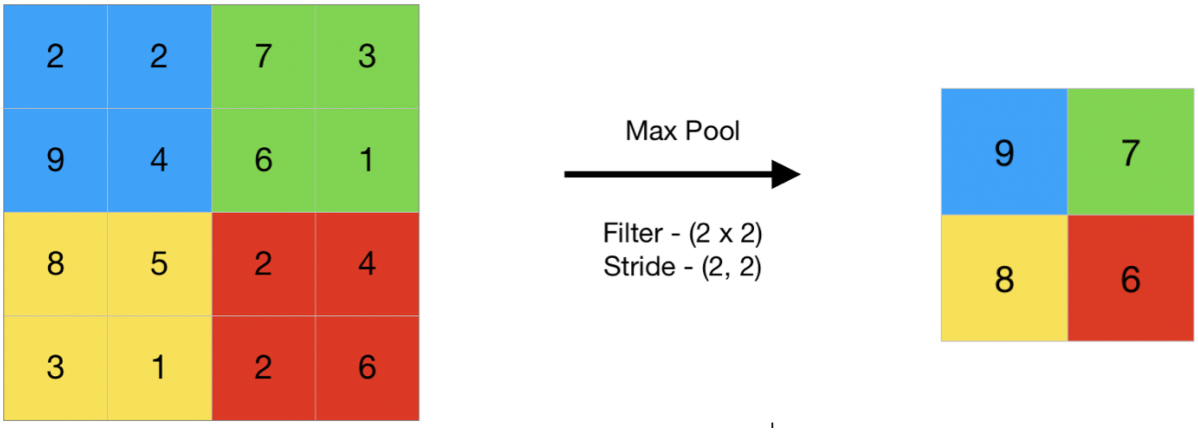



Figure 4: Max Pooling Layers

#### 2.1.2.2 Average Pooling layers

Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.

Average pooling provides a more **generalized representation** of the input. It is useful in the cases where preserving the overall context is important.

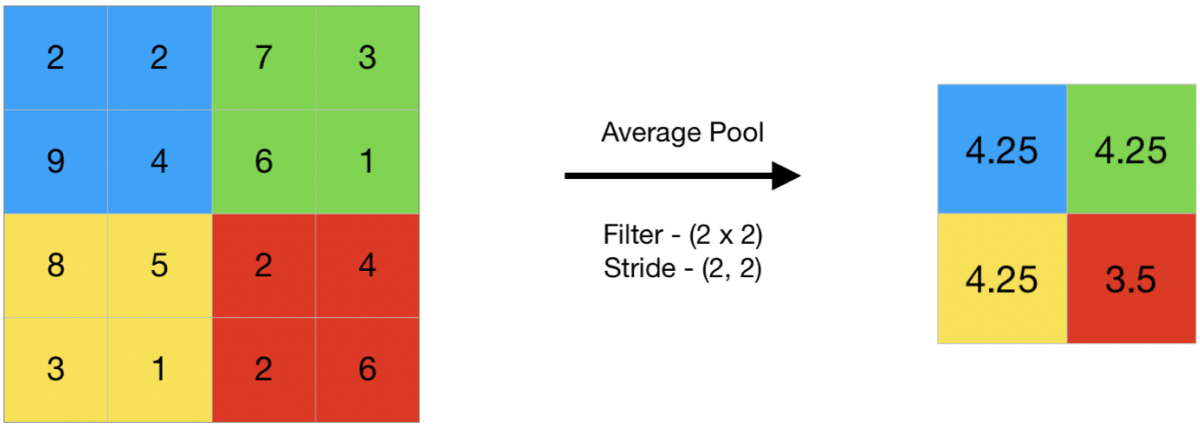



Figure 5: Average Pooling Layers

### 2.1.3 Activation Functions

An activation function in a neural network is a mathematical function applied to the output of a neuron. It introduces non-linearity, enabling the model to learn and represent complex data patterns. Without it, even a deep neural network would behave like a simple linear regression model.

Some activation functions

- Sigmoid Function
- Tanh Function
- ReLU Function
- Leaky ReLU
- Exponential Linear Units (ELU)
- Maxout

#### 2.1.3.1 Sigmoid Function

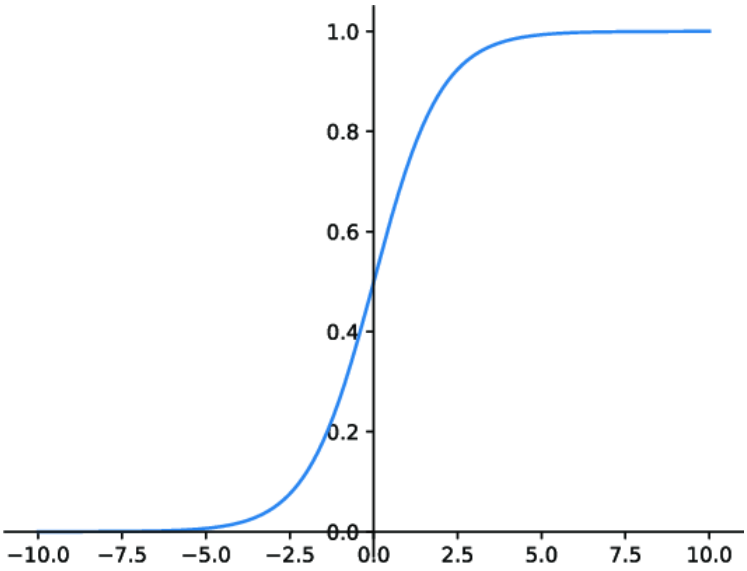



Figure 6: Sigmoid Function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

This activation function squashes input values into the range $[0, 1]$. It has been widely used in the history of neural networks because it simulates the firing rate of biological neurons.

However, it has several disadvantages:

- Vanishing gradient due to saturated neurons
- Non-zero mean output
- Computationally expensive due to the exponential function $\exp(\cdot)$

#### 2.1.3.2 Tanh Function

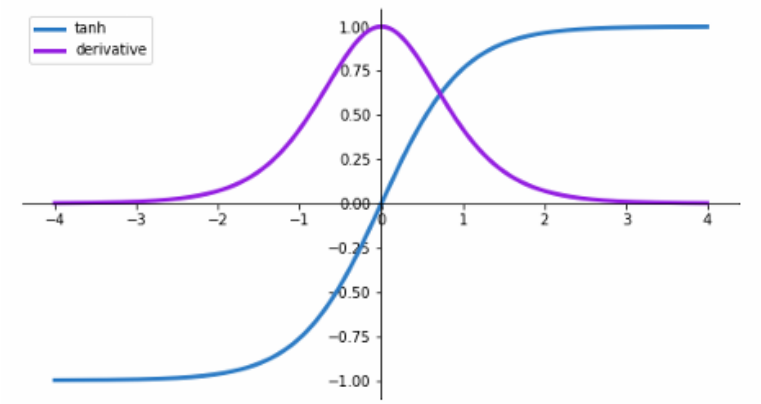



Figure 7: Tanh Function

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

The hyperbolic tangent activation function squashes input values into the range $[-1, 1]$.

- The average output value is 0
- It still suffers from saturated neurons and the vanishing gradient problem

#### 2.1.3.3 ReLU Function

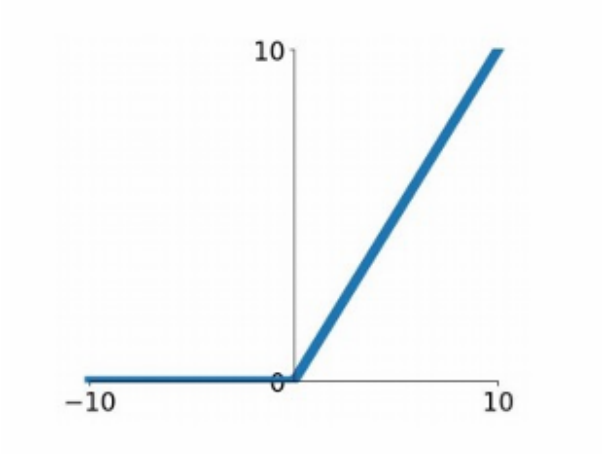



Figure 8: ReLU Function

$$f(x) = \max(0, x)$$

- **Value range:** $[0, \infty)$, meaning the function only outputs non-negative values.
- ReLU is less computationally expensive than *tanh* and *sigmoid*.
- In practice, convergence is faster than sigmoid/tanh (about 6 times).
- **Downsides:**
  1. Non-zero average output.
  2. **Dying ReLU problem:** some ReLU neurons may be permanently deactivated when the input data causes the output to always be negative, resulting in zero gradients and no further weight updates.

#### 2.1.3.4 Leaky ReLU

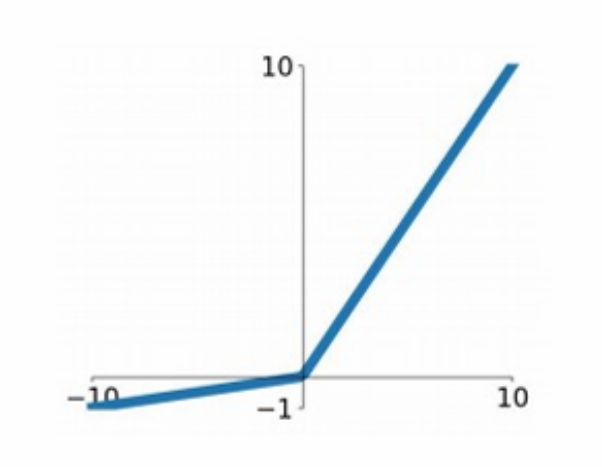



Figure 9: Leaky ReLU

$$f(x) = \max(\alpha x, x)$$

- Leaky ReLU is similar to ReLU but allows a small negative slope ( $\alpha$ , e.g., 0.01) instead of zero.
- **Value range:** $(-\infty, \infty)$
- Solves the "dying ReLU" problem, where neurons get stuck with zero outputs
- Preferred in some cases for better gradient flow

#### 2.1.3.5 Exponential Linear Units (ELU)

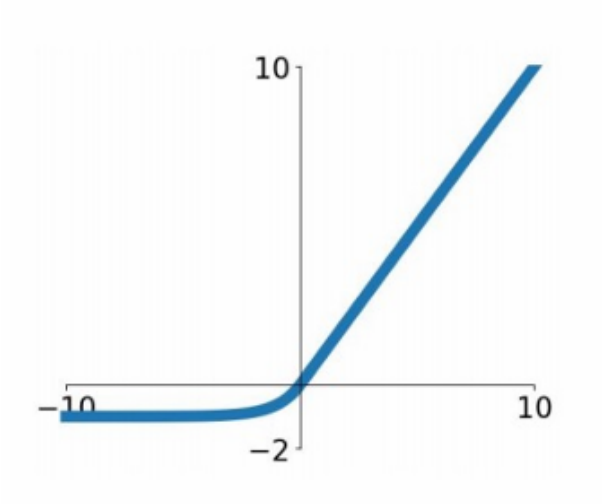



Figure 10: ELU

The Exponential Linear Unit (ELU) is defined as:

$$f(x) = \begin{cases} x, & \text{if } x > 0, \\ \alpha\left(e^x - 1\right), & \text{if } x \leq 0 \end{cases}$$

**Advantages:**

- Retains all benefits of ReLU
- Outputs are closer to zero mean
- Negative saturation compared with Leaky ReLU, adding robustness to noise

**Disadvantages:**

- Computation requires evaluating the exponential function $\exp(x)$

#### 2.1.3.6 Maxout function

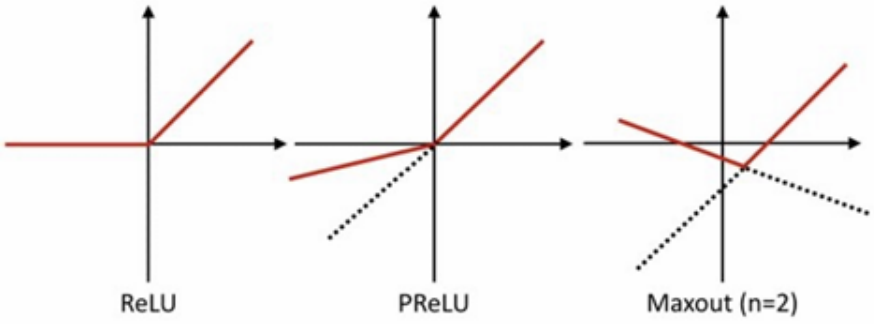



Figure 11: Maxout

$$f(x) = \max\left(\mathbf{w}_1^T\mathbf{x} + b_1,\ \mathbf{w}_2^T\mathbf{x} + b_2\right)$$

- Does not have the basic form of a dot product → nonlinearity
- Generalizes ReLU and Leaky ReLU
- Linear regime! Does not saturate! Does not die!
- Doubles the number of parameters per neuron

### 2.1.4 Fully Connected Layers

Fully Connected (FC) layers are also known as dense layers which are used in neural networks especially in of deep learning. They are a type of neural network layer where every neuron in the layer is connected to every neuron in the previous and subsequent layers. The "fully connected" descriptor comes from the fact that each of the neurons in these layers is connected to every activation in the previous layer creating a highly interconnected network.

- In CNNs, fully connected layers often follow convolutional and pooling layers used to interpret the feature maps generated by these layers into the final output categories or predictions.
- In fully connected feedforward networks these layers are the main building blocks that directly process the input data into outputs.

The structure of FC layers is one of the most significant factors that define how it works in a neural network. This structure involves the fact that every neuron in one layer will interconnect with every neuron in the subsequent layer.

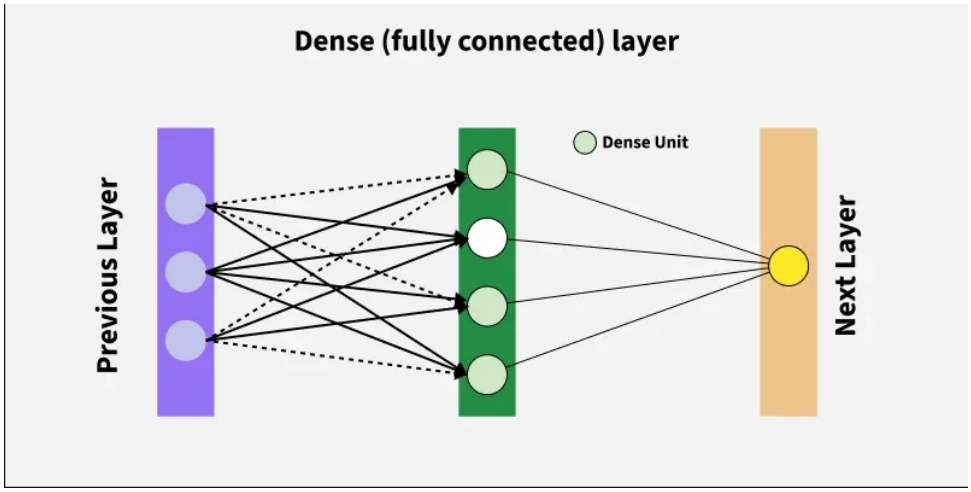



Figure 12: Fully Connected Layers

**Advantages:**

- **Integration of Features:** Fully connected (FC) layers are capable of combining all extracted features before making predictions, which is essential for complex pattern recognition tasks.
- **Flexibility:** FC layers can be incorporated into various network architectures and can handle any form of input data, provided it is suitably reshaped.
- **Simplicity:** These layers are straightforward to implement and are supported by all major deep learning frameworks.

**Drawbacks:**

- **High Computational Cost:** The dense connections lead to a large number of parameters, increasing both computational complexity and memory usage.
- **Prone to Overfitting:** Due to the high number of parameters, FC layers can easily overfit on smaller datasets unless regularization techniques such as dropout or weight decay are applied.
- **Inefficiency with Spatial Data:** Unlike convolutional layers, FC layers do not exploit the spatial hierarchy of images or other structured data, which can result in less effective learning.

## 2.2 Optimizer

In machine learning and deep learning, optimizers and loss functions are two fundamental components that help improve a model's performance.

- A loss function evaluates a model's effectiveness by computing the difference between expected and actual outputs. Common loss functions include log loss, hinge loss and mean square loss.
- An optimizer improves the model by adjusting its parameters (weights and biases) to minimize the loss function value. Examples include RMSProp, ADAM and SGD (Stochastic Gradient Descent).

The optimizer's role is to find the **best combination** of weights and biases that leads to the most accurate predictions.

**Some optimizers:**

- Gradient Descent
- AdaGrad
- RMSProp
- Adam
- AdamW
- ...

### 2.2.1 Gradient Descent

Gradient Descent is a popular optimization method for training machine learning models. It works by iteratively adjusting the model parameters in the direction that minimizes the loss function.

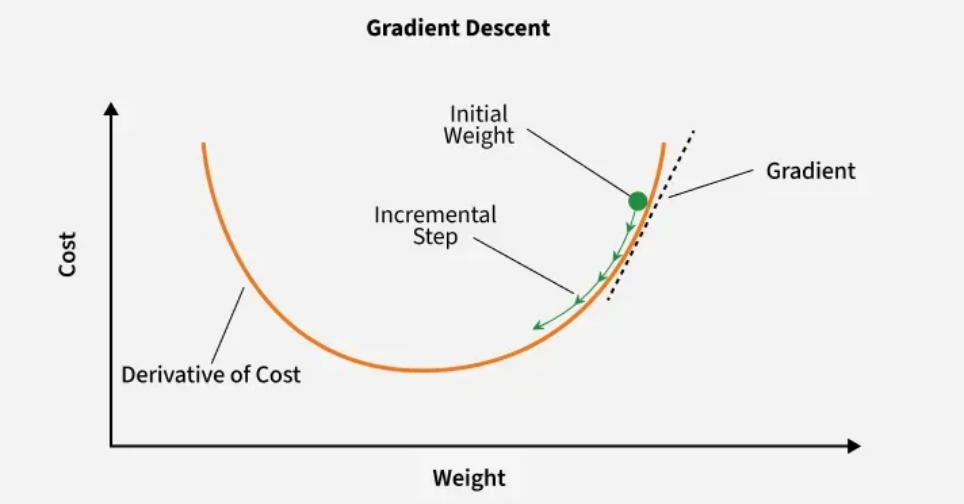



Figure 13: Gradient Descent

Step of Gradient Descent

- **Initialize parameters:** Randomly initialize the model parameters.
- **Compute the gradient:** Calculate the gradient (derivative) of the loss function with respect to the parameters.
- **Update parameters:** Adjust the parameters by moving in the opposite direction of the gradient, scaled by the learning rate.

**Formula:**

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \nabla J(\boldsymbol{\theta}_k)$$

**where:**

- $\boldsymbol{\theta}_{k+1}$ is the updated parameter vector at the $(k+1)^{\text{th}}$ iteration.
- $\boldsymbol{\theta}_k$ is the current parameter vector at the $k^{\text{th}}$ iteration.
- $\alpha$ is the learning rate, which is a positive scalar that determines the step size for each iteration.
- $\nabla J(\boldsymbol{\theta}_k)$ is the gradient of the cost (loss) function $J$ with respect to the parameters $\boldsymbol{\theta}_k$.

#### 2.2.1.1 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) updates the model parameters after each training example, making it more efficient for large datasets compared to traditional Gradient Descent, which uses the entire dataset for each update.

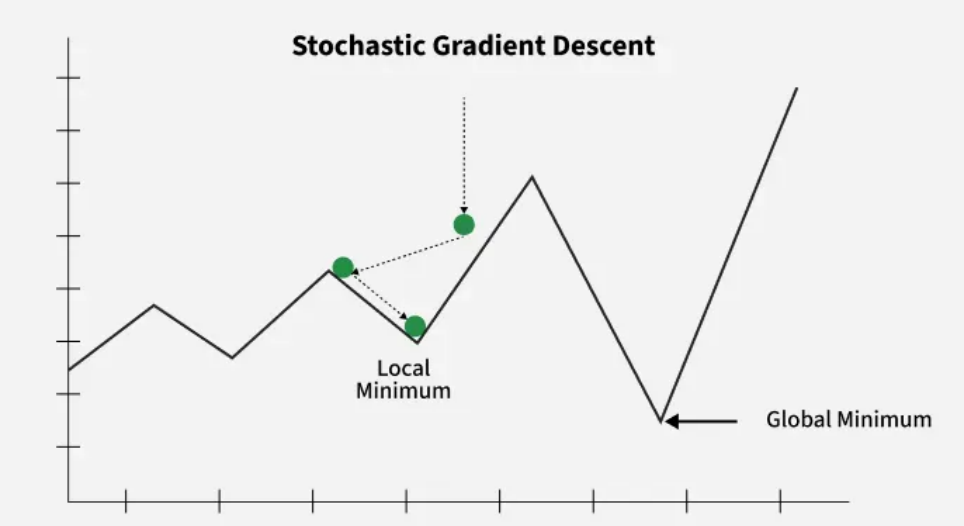



Figure 14: SGD

**Advantages**: Requires less memory and may find new minima.

**Disadvantages**: Noisier, requiring more iterations to converge.

#### 2.2.1.2 SGD with momentum

Momentum helps accelerate convergence by smoothing out the noisy gradients of SGD, thus reducing fluctuations and improving the speed of convergence.

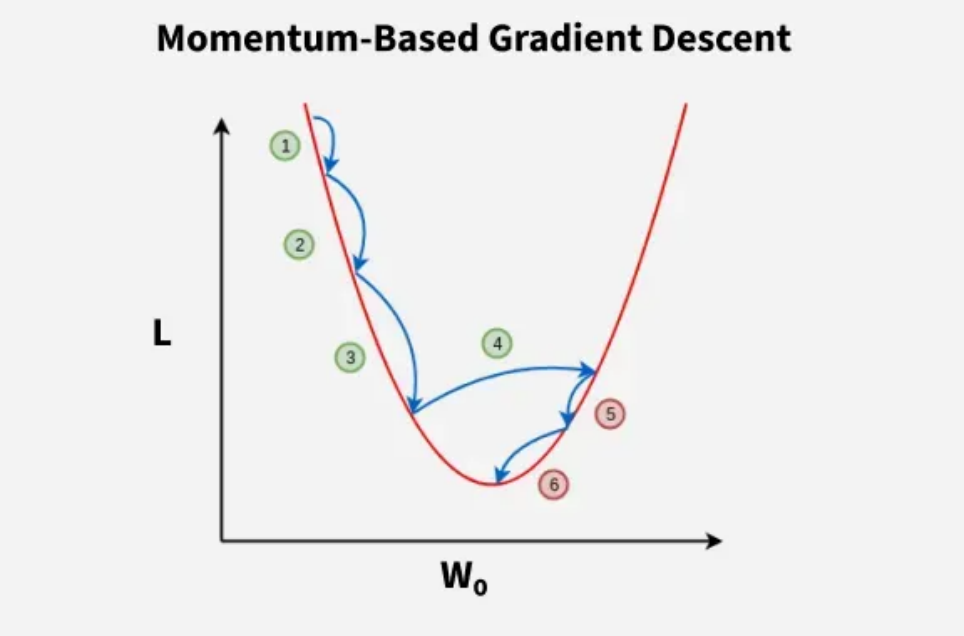



Figure 15: SGD with momentum

**Formula:**

$$\boldsymbol{v}_{k+1} = \boldsymbol{\beta} * v_t + (1 - \beta) * \nabla J(\boldsymbol{\theta}_t)$$

**where:**

- $\boldsymbol{v_t}$ is the momentum at time t
- $\boldsymbol{\beta}$ is the momentum coefficient.
- $\nabla J(\boldsymbol{\theta}_t)$ is the gradient at time t

### 2.2.2 AdaGrad

AdaGrad adapts the learning rate for each parameter based on the historical gradient information. The learning rate decreases over time, making AdaGrad effective for sparse features.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{G_t + \varepsilon}} \nabla J(\boldsymbol{\theta}_t)$$

**Where:**

- $G_t$ is the sum of squared gradients.
- $\varepsilon$ is a small constant to avoid division by zero.

**Advantages:** Adapts the learning rate, improving training efficiency.

**Disadvantages:** Learning rate decays too quickly, causing slow convergence.

#### 2.2.3 RMSProp

RMSProp improves upon AdaGrad by introducing a decay factor to prevent the learning rate from decreasing too rapidly.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)(\nabla J(\boldsymbol{\theta}_t))^2$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{E[g^2]_t + \varepsilon}} \nabla J(\boldsymbol{\theta}_t)$$

**Where:**

- $\gamma$ is the decay rate.
- $E[g^2]_t$ is the exponentially moving average of squared gradients.

**Advantages:** Prevents excessive decay of learning rates.

**Disadvantages:** Computationally expensive due to the additional parameter.

#### 2.2.4 Adam Optimizers

Adam combines the advantages of Momentum and RMSProp. It uses both the first moment (mean) and second moment (variance) of gradients to adapt the learning rate for each parameter.

1. **Update the first moment:** $m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla J(\boldsymbol{\theta}_t)$
2. **Update the second moment:** $v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla J(\boldsymbol{\theta}_t))^2$
3. **Bias correction:** $\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$
4. **Update parameters:** $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{\hat{v}_t + \varepsilon}} \hat{m}_t$

**Where:**

- $\beta_1$ and $\beta_2$ are the decay rates for the first and second moments.
- $\varepsilon$ is a small constant to prevent division by zero.

**Advantages:** Fast convergence.

**Disadvantages:** Requires significant memory due to the need to store first and second moment estimates.

##### 2.2.4.1 AdamW Optimizers

AdamW (Adaptive Moment Estimation with Decoupled Weight Decay) is an improved variant of the Adam optimizer. Unlike Adam, AdamW decouples weight decay from the gradient-based update rule, which leads to better regularization and improved generalization performance in deep neural networks.

Given the learning rate $\alpha$, exponential decay rates $\beta_1, \beta_2$, a small constant $\epsilon$, and weight decay coefficient $\lambda$, AdamW updates parameters as follows:

$$g_t = \nabla_\theta L(\theta_t) \tag{1}$$
$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{2}$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{3}$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{4}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{5}$$

The parameter update with decoupled weight decay is given by:

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} - \alpha \lambda \theta_t \tag{6}$$

**Advantages**

- **Decoupled weight decay:** Weight decay is applied independently from the gradient update, ensuring correct regularization.
- **Better generalization:** Often achieves higher validation accuracy and reduced overfitting compared to Adam.
- **Fast and stable convergence:** Inherits adaptive learning rates and momentum from Adam.
- **Widely used in practice:** Commonly adopted in state-of-the-art models such as Transformers.

**Disadvantages**

- **More hyperparameters:** Requires tuning both learning rate and weight decay.
- **Higher memory cost:** Stores first and second moment estimates for each parameter.
- **Not always optimal:** In some cases, SGD with momentum may provide better generalization.

## 2.3 Learning Rate

The learning rate is a key hyperparameter in neural networks that controls how quickly the model learns during training. It determines the size of the steps taken to minimize the loss function. It controls how much change is made in response to the error encountered, each time the model weights are updated. It determines the size of the steps taken towards a minimum of the loss function during optimization.

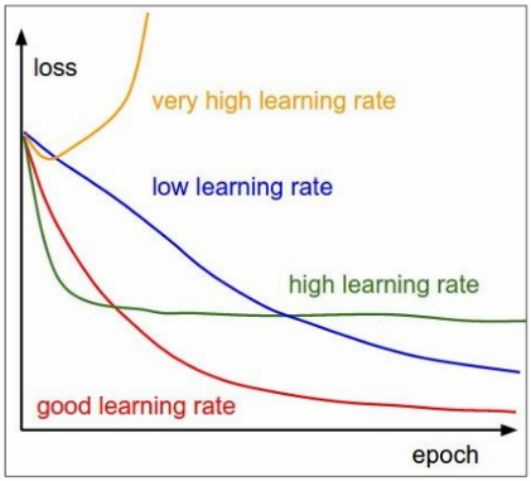



Figure 16: Learning Rate

**Low Learning Rate**:

- Leads to slow convergence
- Requires more training epochs
- Can improve accuracy but increases computation time

**High Learning Rate**:

- Speeds up training
- Risks of overshooting optimal weights
- May cause instability or divergence of the loss function

**Optimal Learning Rate**:

- Balances training speed and model accuracy
- Ensures stable convergence without excessive training time

In practice, we decay learning rate over time by reduce learning rate at some fixed points

### 2.3.1 Cosine rate decay

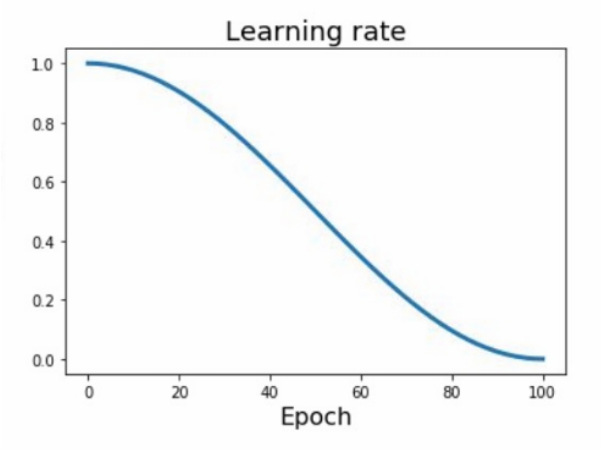



Figure 17: Cosine rate Decay

**Formula:**

$$\alpha_t = \frac{1}{2}\alpha_0 \left(1 + \cos\left(\frac{t\pi}{T}\right)\right)$$

**where:**

- $\alpha_0$ : Initial learning rate
- $\alpha_t$ : Learning rate at epoch t
- T : Total number of epochs

### 2.3.2 Linear rate decay

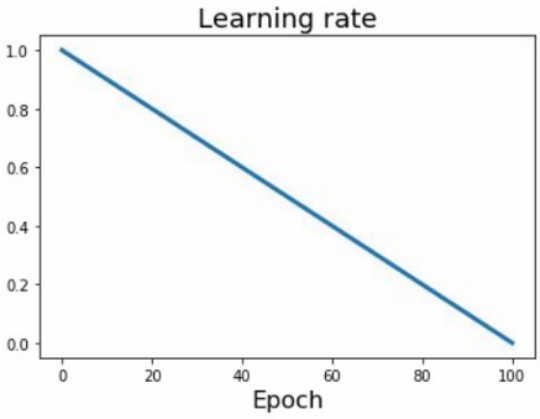



Figure 18: Linear rate Decay

**Formula:**

$$\alpha_t = \alpha_0(1 - t/T)$$

**where:**

- $\alpha_0$ : Initial learning rate
- $\alpha_t$ : Learning rate at epoch t
- $T$ : Total number of epochs

### 2.3.3 Inverse square root of total number of epochs

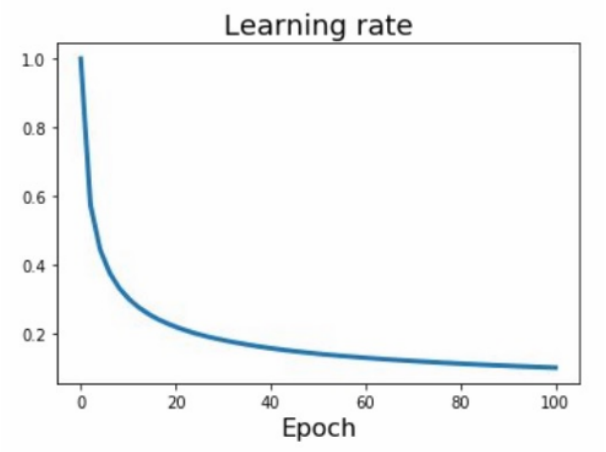



Figure 19: Inverse Square-root of total number of epochs

**Formula:**

$$\alpha_t = \alpha_0 / \sqrt{t}$$

**where:**

- $\alpha_0$ : Initial learning rate
- $\alpha_t$ : Learning rate at epoch t
- $T$ : Total number of epochs

### 2.3.4 Linear Warmup

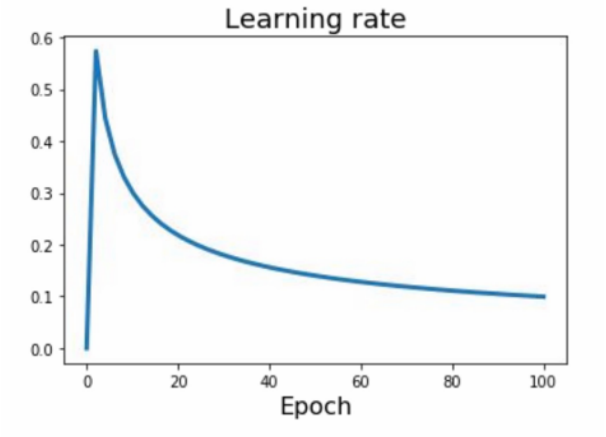



Figure 20: Linear Warmup

High initial learning rates can make loss explode; linearly increasing learning rate from 0 over the first 5000 iterations can prevent this

Empirical rule of thumb: If you increase the batch size by N, also scale the initial learning rate by N

## 2.4 Regularization

### 2.4.1 Dropout

Dropout is a regularization technique which involves randomly ignoring or "dropping out" some layer outputs during training, used in deep neural networks to prevent overfitting.

Dropout is implemented per-layer in various types of layers like dense fully connected, convolutional, and recurrent layers, excluding the output layer. The dropout probability specifies the chance of dropping outputs, with different probabilities for input and hidden layers that prevents any one neuron from becoming too specialized or overly dependent on the presence of specific features in the training data.

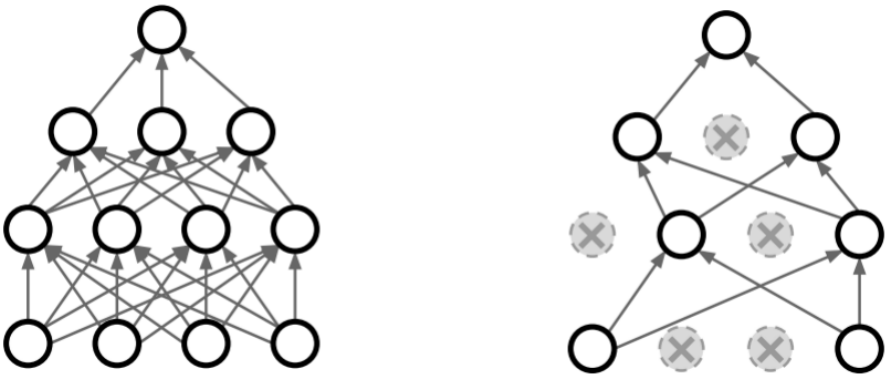


Figure 21: Dropout

**Advantages**

- **Prevent Overfitting**: By randomly disabling neurons, the network cannot overly rely on the specific connections between them.
- **Ensemble Effect**: Dropout acts like training an ensemble of smaller neural networks with varying structures during each iteration. This ensemble effect improves the model's ability to generalize to unseen data.
- **Enhancing Data Representation**: Dropout methods are used to enhance data representation by introducing noise, generating additional training samples, and improving the effectiveness of the model during training.

**Drawbacks**

- **Longer Training Times**: Dropout increases training duration due to random dropout of units in hidden layers. To address this, consider powerful computing resources or parallelize training where possible
- **Optimization Complexity**: Understanding why dropout works is unclear, making optimization challenging. Experiment with dropout rates on a smaller scale before full implementation to fine-tune model performance.

- **Hyperparameter Tuning**: Dropout adds hyperparameters like dropout chance and learning rate, requiring careful tuning. Use techniques such as grid search or random search to systematically find optimal combinations.
- **Redundancy with Batch Normalization**: Batch normalization can sometimes replace dropout effects. Evaluate model performance with and without dropout when using batch normalization to determine its necessity.
- **Model Complexity**: Dropout layers add complexity. Simplify the model architecture where possible, ensuring each dropout layer is justified by performance gains in validation.

### 2.4.2 Early Stopping

Early stopping is a regularization technique that stops model training when overfitting signs appear. It prevents the model from performing well on the training set but underperforming on unseen data i.e validation set. Training stops when performance improves on the training set but degrades on the validation set, promoting better generalization while saving time and resources.

The technique monitors the model's performance on both the training and validation sets. If the validation performance worsens, training stops and the model retains the best weights from the period of optimal validation performance.

Early stopping is an efficient method when training data is limited as it typically requires fewer epochs than other techniques. However, overusing early stopping can lead to overfitting the validation set itself, similar to overfitting the training set.

The number of training epochs is a hyperparameter that can be optimized for better performance through hyperparameter tuning.

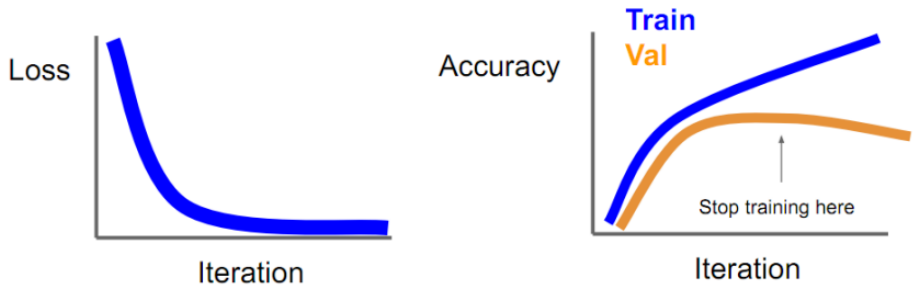



Figure 22: Early Stopping

**Benefits of Early Stopping**

- **Reduces Overfitting**: By stopping training when overfitting starts, early stopping improves generalization to unseen data.
- **Saves Computational Resources**: Training for too many epochs can be time-consuming and costly. It stops training once further improvement is unlikely, saving time and resources.
- **Improves Model Efficiency**: It leads to better-performing models in fewer epochs making the process more efficient.
- **Simple to Implement:** It is a straightforward technique that can be easily applied using built-in tools in most machine learning libraries.

**Limitations of Early Stopping**

- **Risk of Underfitting:** Stopping too early may lead to underfitting where the model doesn't fully learn the patterns in the data. This results in poor generalization.
- **Not Suitable for All Models:** Early stopping is not beneficial for every type of model. Complex models may require more extensive training to achieve optimal performance, making early stopping less effective.
- **Dependency on the Validation Set:** A poorly chosen or small validation set may fail to accurately indicate when to stop, leading to suboptimal performance.
- **Computational Overhead:** Validation checks for early stopping can still incur computational costs especially with large models or datasets, despite saving resources overall.

## 2.5 Data Augmentation

Data augmentation is a technique used to increase diversity of a dataset without actually collecting new data. It works by applying various transformations to the existing data to create new, modified versions of data that helps the model generalize better.

These transformations are applied in a way that maintains the original label of the data while creating augmented data for training.

### 2.5.1 Geometrics Transformation

Geometric transformations alter the spatial properties of an image

1. **Rotation**: It rotate the image to a certain angle like 90° or 180°.
2. **Flipping**: It flips the image horizontally or vertically.
3. **Scaling**: Helps in zooming in or out in image.
4. **Translation**: Shifting the image along the x or y axis.
5. **Shearing**: Slanting the shape of the image

### 2.5.2 Color Space Augmentations

Color space augmentations modify the color properties of an image

1. **Brightness Adjustment**: We can increase or decrease the brightness of the image.
2. **Contrast Adjustment**: It change the contrast of image.
3. **Saturation Adjustment**: It modify intensity of colors in the image.
4. **Hue Adjustment**: Shifting the colors by changing the hue.

### 2.5.3 Kernel Filters

Kernel filters apply convolutional operations to enhance or suppress specific features in the image

1. **Blurring**: Applying Gaussian blur to smooth the image.

2. **Sharpening**: Enhancing the edges to make the image sharper.
3. **Edge Detection**: Highlighting the edges in the image using filters like Sobel or Laplacian.

#### 2.5.4 Random Erasing

Random erasing involves randomly masking out a rectangular region of the image. This helps the model become invariant to occlusions and improves its ability to handle missing parts of objects.

### 2.6 Transfer Learning

Transfer learning is a machine learning technique where a model trained on one task is repurposed as the foundation for a second task. This approach is beneficial when the second task is related to the first or when data for the second task is limited.

Using learned features from the initial task, the model can adapt more efficiently to the new task, accelerating learning and improving performance. Transfer learning also reduces the risk of overfitting, as the model already incorporates generalizable features useful for the second task.

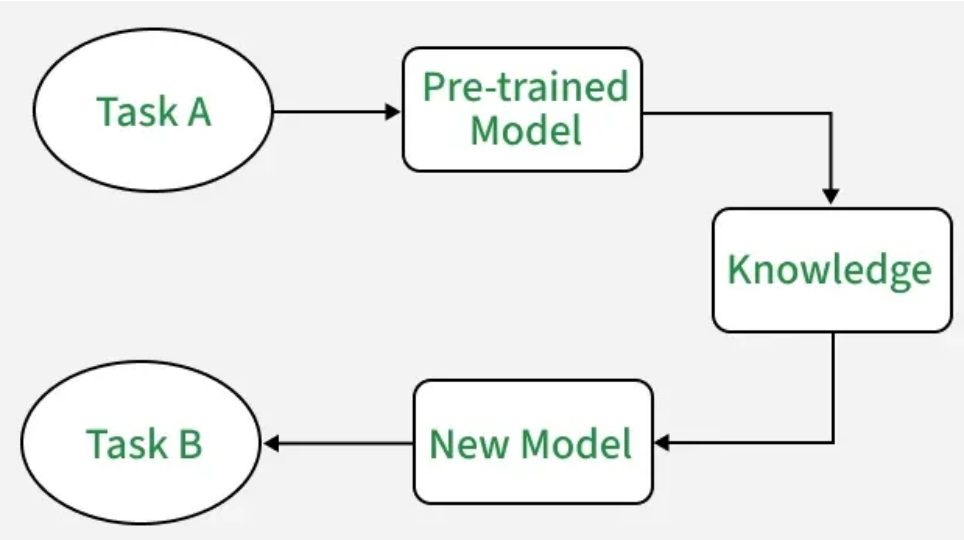



Figure 23: Transfer Learning

**Advantages**

- **Speed up the training process**: By using a pre-trained model the model can learn more quickly and effectively on the second task, as it already has a good understanding of the features and patterns in the data.
- **Better performance**: Transfer learning can lead to better performance on the second task, as the model can use the knowledge it has gained from the first task.
- **Handling small datasets**: When there is limited data available for the second task, transfer learning can help to prevent overfitting as the model will have already learned general features that are likely to be useful in the second task.

**Disadvantages**

- **Domain mismatch**: The pre-trained model may not be well-suited to the second task if the two tasks are vastly different or the data distribution between the two tasks is very different.

- **Overfitting**: Transfer learning can lead to overfitting if the model is fine-tuned too much on the second task, as it may learn task-specific features that do not generalize well to new data.

- **Complexity**: The pre-trained model and the fine-tuning process can be computationally expensive and may require specialized hardware.

In our project, we employ ResNet18 as our transfer learning approach.

#### 2.6.1 ResNet18

The ResNet18 model consists of 18 layers and is a variant of the Residual Network (ResNet) architecture. The residual blocks are the core building blocks of ResNet and include skip connections that bypass one or more layers. Residual blocks help in mitigating the vanishing gradient problem by allowing the network to learn residual mappings instead of the original mappings. This makes gradient flow easier during backpropagation.

A residual block in ResNet18 consists of:

- Two convolutional layers
- Batch normalization
- ReLU activation

The skip connection adds the input of the block to the output of the second convolutional layer. This helps the network to learn residual features, improving convergence and overall performance.

The implementation of ResNet18 consists of several steps:

1. Install and Import Libraries
2. Load and Preprocess Dataset
3. Define the Residual Block
4. Define ResNet18 Architecture
5. Define Loss Function and Optimizer
6. Train the Model
7. Plot Training Loss and Accuracy

### 2.7 Evaluation Metrics

To comprehensively assess model performance on the FER-2013 dataset, multiple evaluation metrics are employed beyond overall accuracy. Given the inherent class imbalance in the dataset, relying solely on accuracy may provide a misleading representation of model effectiveness, as high performance on majority classes can mask poor recognition of minority emotions. Therefore, a combination of class-wise and aggregate metrics is used to ensure a fair and informative evaluation.

Overall classification **accuracy** is reported to provide a general measure of correct predictions across all emotion categories. In addition, **precision, recall, and F1-score** are computed for each class. Precision measures the reliability of positive predictions for a given emotion, while recall reflects the model's ability to correctly identify all instances of that emotion. The F1-score, defined as the harmonic

mean of precision and recall, provides a balanced metric that accounts for both false positives and false negatives.

To further mitigate the effects of class imbalance, macro-averaged metrics are reported. Macro-averaging assigns equal weight to each emotion class, regardless of its frequency, thereby emphasizing performance on underrepresented categories such as Disgust and Fear. This approach ensures that improvements in minority class recognition are appropriately reflected in the evaluation.

Additionally, a confusion matrix is used to visualize prediction errors and inter-class confusion patterns. This analysis highlights systematic misclassifications between visually similar emotions, such as Fear and Surprise, and provides qualitative insight into the strengths and limitations of the model.

Together, these evaluation metrics offer a robust and comprehensive assessment of model performance, enabling meaningful comparisons across different training strategies and ensuring balanced recognition across all emotion categories.

# 3 Group's CNN model

## 3.1 Data Preprocessing

The FER-2013 dataset consists of grayscale facial images of size 48×48 pixels, stored in CSV format. Each image is represented as a sequence of pixel intensity values ranging from 0 to 255. The dataset contains seven emotion categories: Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral. The dataset is pre-split into training, validation, and test sets.

The dataset is loaded from the CSV file, where pixel values stored as space-separated strings are converted into two-dimensional arrays and reshaped into 48×48 grayscale images. Emotion labels are mapped to integer class indices.

To stabilize training and accelerate convergence, pixel values are normalized to the range [-1, 1] using mean 0.5 and standard deviation 0.5. This ensures consistent input distribution across batches.

$$x' = \frac{x - 0.5}{0.5}$$

To improve generalization and reduce overfitting, data augmentation is applied exclusively to the training set. Augmentations include **random horizontal flipping, small rotations, and random cropping.** These transformations simulate real-world variations in facial expressions while preserving emotion semantics.

The FER-2013 dataset exhibits a pronounced class imbalance across emotion categories. In particular, emotions such as Disgust and Fear are significantly underrepresented compared to dominant classes such as Happy and Neutral. This imbalance can lead to biased learning, where the model disproportionately favors majority classes, resulting in poor recognition performance for minority emotions.

To address this issue, two complementary imbalance mitigation strategies are considered during training: class-weighted loss functions and weighted random sampling.

### 3.1.1 Class-Weighted Loss Function

In the class-weighted loss approach, higher importance is assigned to underrepresented emotion classes by scaling the contribution of each class in the loss function according to its inverse frequency in the training set. As a result, misclassification errors on minority classes incur a larger penalty during optimization, encouraging the model to learn more discriminative features for rare emotions.

This method directly influences gradient updates and is particularly effective when class imbalance is moderate. However, excessive weighting can lead to training instability or overfitting to minority classes if not carefully controlled.

#### 3.1.2 Weighted Random Sampling

As an alternative strategy, a WeightedRandomSampler is employed during training to rebalance the class distribution at the data sampling level. Instead of modifying the loss function, this approach adjusts the probability of selecting training samples such that images from minority classes are sampled more frequently. Consequently, each training batch contains a more balanced representation of emotion categories.

This sampling-based approach ensures that the model is exposed uniformly to all classes throughout training, reducing bias toward majority classes while preserving the original data distribution during evaluation.

While class-weighted loss modifies the optimization objective directly, weighted random sampling addresses imbalance at the data level by ensuring balanced exposure during training. In our experiments, this resulted in more stable gradients and improved validation performance.

Based on comparative experiments, weighted random sampling is chosen as the primary method for addressing class imbalance. Compared to class-weighted loss, this approach provides more stable training dynamics and yields improved validation performance by ensuring balanced class exposure during training. Therefore, weighted random sampling is adopted in the final model to mitigate class imbalance in the FER-2013 dataset.

### 3.2 Model Architecture

This model is a custom Convolutional Neural Network (CNN) designed for facial emotion recognition on the FER-2013 dataset. It takes grayscale 48×48 images as input and outputs probabilities for 7 emotion classes.

The architecture follows a progressive feature extraction strategy:

- Spatial size ↓ ($48 \to 24 \to 12 \to 6$)
- Channel depth ↑ ($1 \to 32 \to 64 \to 128 \to 256$)
- Final classification via Global Average Pooling + Fully Connected layer

| **Block** | **Layers** | **Output Shape** | **Params** |
|---|---|---|---|
| Input | – | $1 \times 48 \times 48$ | – |
| Block 1 | 2×Conv + BN + ReLU + MaxPool | $32 \times 24 \times 24$ | 9.7K |
| Block 2 | 2×Conv + BN + ReLU + MaxPool | $64 \times 12 \times 12$ | 55.7K |
| Block 3 | 2×Conv + BN + ReLU + MaxPool | $128 \times 6 \times 6$ | 221.7K |
| Block 4 | Conv + BN + ReLU + Dropout | $256 \times 6 \times 6$ | 295.7K |
| GAP + FC | Global Avg Pool + Linear | 7 | 1.8K |
| **Total** | | | **584,807** |

Table 1: Architecture of the proposed CNN for FER-2013

The proposed CNN contains approximately 0.58 million trainable parameters, which is significantly smaller than standard deep architectures such as ResNet18 (11M parameters). This compact design reduces the risk of overfitting on the relatively small FER-2013 dataset while maintaining sufficient representational capacity for facial emotion recognition.

**Detailed Layer Explanation**

- Input: Shape: (B, 1, 48, 48)
  - B: batch size
  - 1: grayscale channel
  - 48×48: facial image resolution
- Block 1 — Low-level feature

```
Conv2d(1 → 32) → BatchNorm → ReLU
Conv2d(32 → 32) → BatchNorm → ReLU
MaxPool2d(2)
```

  - Learns basic visual features: edges, corners, simple textures
  - Two convolutions improve feature richness
  - Max pooling reduces spatial size and adds translation invariance
  - Shape transformation: 1×48×48 → 32×48×48 → 32×24×24
- Block 2 — Mid-level feature extraction

```
Conv2d(32 → 64) → BatchNorm → ReLU
Conv2d(64 → 64) → BatchNorm → ReLU
MaxPool2d(2)
```

  - Extracts combinations of low-level features: eye regions, mouth curves, facial contours
  - Doubles channel depth to increase representational capacity
  - Shape transformation: 32×24×24 → 64×24×24 → 64×12×12
- Block 3 — High-level facial patterns

```
Conv2d(64 → 128) → BatchNorm → ReLU
Conv2d(128 → 128) → BatchNorm → ReLU
MaxPool2d(2)
```

  - Captures emotion-related structures: smiles vs frowns, raised eyebrows, eye openness
  - Further reduces spatial size while increasing semantic abstraction
  - Shape transformation: 64×12×12 → 128×12×12 → 128×6×6
- Block 4 — Deep semantic features + regularization

```
Conv2d(128 → 256) → BatchNorm → ReLU
Dropout2d
```

  - Learns high-level emotional representations
  - Channel depth increases to 256 for strong discriminative power
  - Dropout2d randomly drops entire feature maps during training → reduces overfitting, and encourages robust feature learning
  - Shape transformation: 128×6×6 → 256×6×6
- Global Average Pooling (GAP)

```
x = x.mean(dim=(2, 3))
```

  - Averages each feature map across spatial dimensions
  - Converts (B, 256, 6, 6) → (B, 256)

  Instead of using fully connected layers after flattening, the model applies Global Average Pooling to aggregate spatial information into a 256-dimensional feature vector. This design significantly reduces the number of parameters and enforces channel-wise semantic learning, which is particularly suitable for emotion recognition tasks.
- Fully Connected Classification Head

```
Linear(256 → 7)
```

  - Maps the 256-dimensional feature vector to 7 emotion logits
  - Each output corresponds to one emotion class
  - Final output: (B, 7)

### 3.3 Optimizer Selection

In this project, the AdamW optimizer is selected for model training due to its effectiveness and stability in deep neural networks. AdamW combines the adaptive learning rate benefits of the Adam optimizer with a decoupled weight decay mechanism, which allows regularization to be applied independently from gradient-based parameter updates. This separation results in more consistent and theoretically sound regularization compared to the standard Adam optimizer, where weight decay is implicitly coupled with gradient updates.

The use of AdamW is particularly advantageous for training convolutional neural networks on moderately sized and noisy datasets such as FER-2013, where overfitting is a common concern. By applying weight decay explicitly, AdamW helps control model complexity and improves generalization performance. Additionally, its adaptive learning rate strategy enables faster convergence and reduces sensitivity to initial learning rate selection, leading to stable and efficient optimization throughout training.

Compared to SGD with momentum, AdamW converges faster in early training and is less sensitive to learning rate tuning, which is advantageous given limited computational resources.

Empirical results further confirm that AdamW provides smoother training dynamics and superior validation performance compared to conventional stochastic gradient descent and standard Adam optimization. As the result, AdamW is adopted as the optimizer in the final training configuration.

## 4 Transfer learning with ResNet18

### 4.1 Input Normalization and Data Preparation

In this project, transfer learning is applied using a ResNet18 model pre-trained on the ImageNet dataset. Since ImageNet models are trained on RGB images of size 224 × 224 224×224 with specific normalization statistics, the FER-2013 dataset must be adapted accordingly.

Originally, FER-2013 images are grayscale facial images with resolution 48×48. To match the input requirements of the pre-trained network, each image is first resized to 224×224 using bicubic interpolation. The grayscale image is then converted into a 3-channel image by duplicating the single channel across RGB channels. This transformation allows the model to reuse the learned low-level filters from ImageNet.

All images are normalized using the standard ImageNet normalization parameters:

$$mean = (0.485, 0.456, 0.406), std = (0.229, 0.224, 0.225)$$

This normalization step is critical, as it ensures that the input distribution aligns with the data distribution seen during pre-training, enabling stable convergence and effective feature reuse.

To improve generalization, several data augmentation techniques are applied during training, including random resized cropping, horizontal flipping, small rotations, color jittering, and random erasing. These augmentations help the model become more robust to variations in facial appearance, pose, and illumination.

### 4.2 Model Architecture for Transfer Learning

The backbone of the transfer learning model is ResNet18, a deep convolutional neural network consisting of residual blocks that mitigate the vanishing gradient problem and enable efficient training of deep architectures.

The model architecture can be divided into two main components:

Frozen Backbone (Feature Extractor) The convolutional layers of ResNet18, pre-trained on ImageNet, are initially frozen. These layers are responsible for extracting generic visual features such as edges, textures, and facial structures, which are transferable across different vision tasks.

Task-Specific Classification Head The original fully connected layer of ResNet18 is replaced with a new classification head tailored to the FER-2013 task. This head consists of:

An optional dropout layer to reduce overfitting

A linear layer mapping from the backbone feature dimension (512) to 7 emotion classes

This design allows the model to leverage powerful pre-trained representations while adapting the final decision layer to the emotion recognition task.

## 4.3 Two-Stage Training Strategy

To effectively apply transfer learning, a two-stage training strategy is employed:

Stage 1: Frozen Backbone Training

In the first stage, all backbone parameters are frozen, and only the classification head is trained. This allows the model to quickly adapt high-level features to the FER-2013 emotion categories without disrupting the pre-trained representations. A relatively higher learning rate is used for the head during this stage to ensure fast convergence.

Stage 2: Fine-Tuning the Backbone

After a fixed number of epochs, the backbone is unfrozen, and the entire network is trained end-to-end. During this fine-tuning phase, a much smaller learning rate is assigned to the backbone layers, while the classification head continues to use a higher learning rate. This prevents catastrophic forgetting while allowing the model to adjust deeper features to domain-specific facial expression patterns.

## 4.4 Handling Class Imbalance

The FER-2013 dataset is highly imbalanced, with certain emotion classes (e.g., disgust) appearing far less frequently than others. To address this issue, WeightedRandomSampler is employed during training.

WeightedRandomSampler ensures that each mini-batch contains a more balanced representation of classes by sampling minority classes more frequently. This approach avoids bias toward majority classes and improves recall on underrepresented emotions. When using the sampler, class weights are deliberately disabled in the loss function to prevent double compensation.

## 4.5 Additional Considerations

- Label Smoothing is applied to the cross-entropy loss to reduce overconfidence and improve generalization.
- Test-Time Augmentation (TTA) is used during validation by averaging predictions from original and horizontally flipped images, leading to more stable and robust performance.
- Cosine Annealing Learning Rate Scheduler is adopted to gradually reduce the learning rate, allowing smoother convergence during long training runs.

# 5 Results

## 5.1 Group's CNN Model result

Table 2: Classification Report

| Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| Angry | 0.6021 | 0.5947 | 0.5984 | 491 |
| Disgust | **0.8182** | 0.6545 | 0.7273 | 55 |
| Fear | 0.5474 | 0.4261 | 0.4792 | 528 |
| Happy | **0.8657** | **0.8874** | **0.8764** | 879 |
| Sad | 0.5292 | 0.5640 | 0.5460 | 594 |
| Surprise | **0.8038** | **0.8077** | **0.8058** | 416 |
| Neutral | 0.6385 | 0.7109 | 0.6727 | 626 |
| **Accuracy** | | | **0.6824** | 3589 |
| **Macro Avg** | 0.6864 | 0.6636 | 0.6723 | 3589 |
| **Weighted Avg** | 0.6796 | 0.6824 | 0.6793 | 3589 |

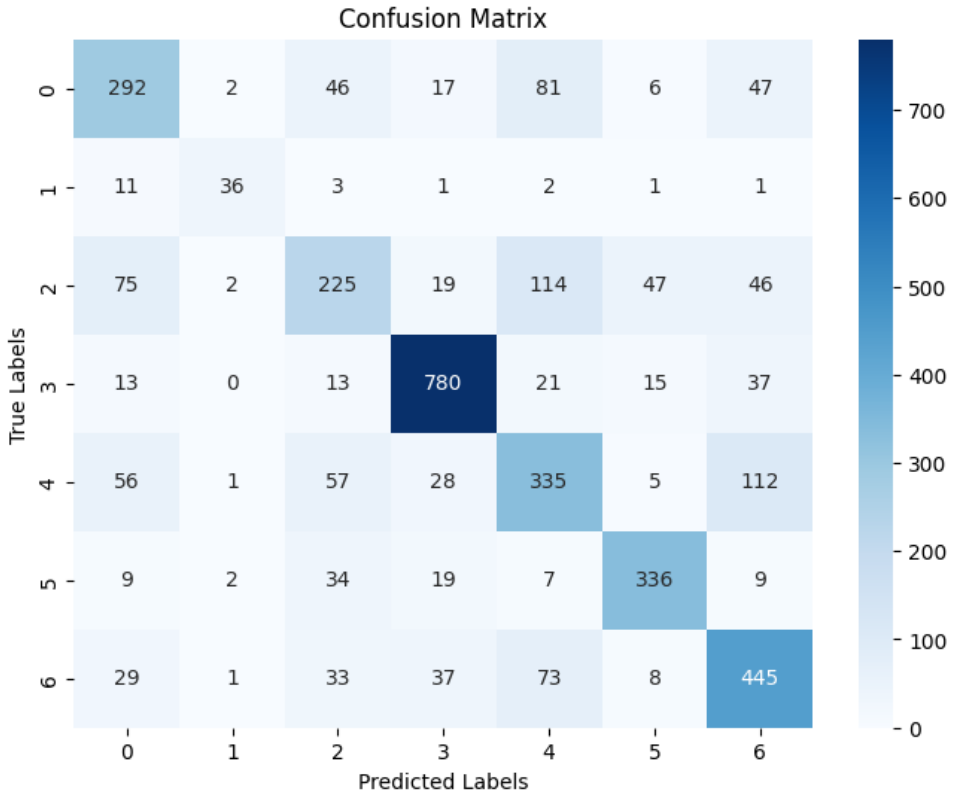



Figure 24: Confusion Matrix of group's model

### 5.2 ResNet18's result

Table 3: Classification Report

| Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| Angry | 0.6267 | 0.6210 | 0.6223 | 467 |
| Disgust | **0.8780** | 0.6429 | 0.7423 | 56 |
| Fear | 0.6202 | 0.5202 | 0.5658 | 496 |
| Happy | **0.8588** | **0.9106** | **0.8839** | 895 |
| Sad | 0.6063 | 0.5896 | 0.5978 | 653 |
| Surprise | **0.8163** | **0.8458** | **0.8308** | 415 |
| Neutral | 0.6172 | 0.6639 | 0.6397 | 607 |
| **Accuracy** | | | **0.7072** | 3589 |
| **Macro Avg** | 0.7172 | 0.6848 | 0.6975 | 3589 |
| **Weighted Avg** | 0.7038 | 0.7072 | 0.7042 | 3589 |

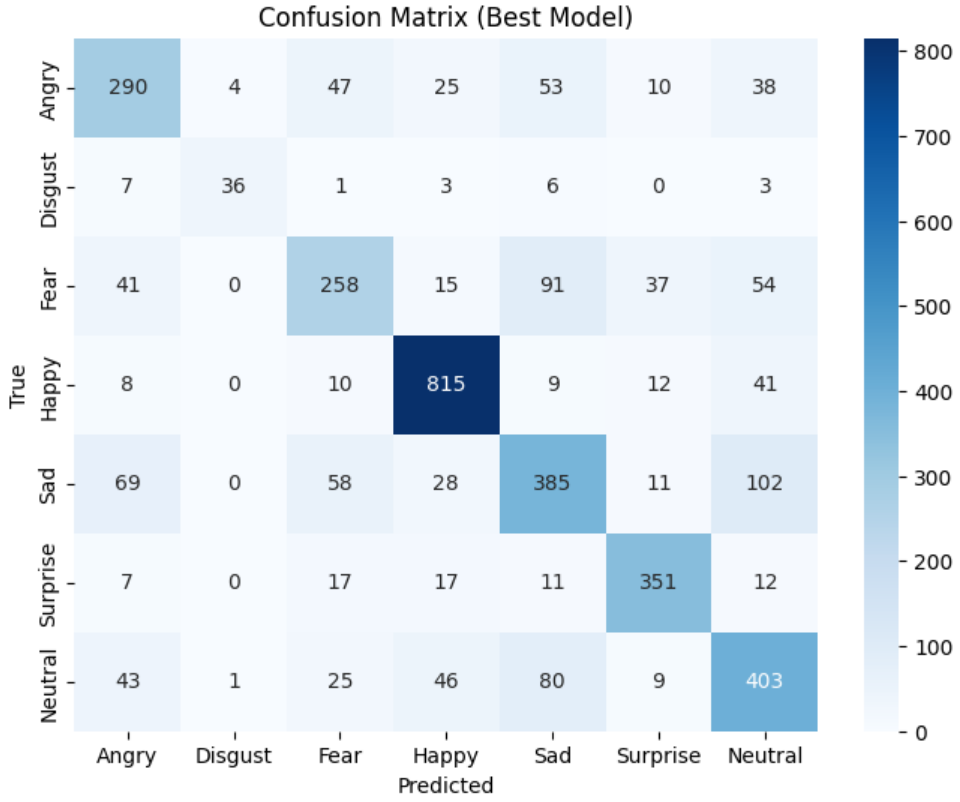



Figure 25: Confusion Matrix of ResNet18

## 6 Conclusion

In this project, two approaches for facial emotion detection on the FER2013 dataset were evaluated: a custom Convolutional Neural Network (CNN) trained from scratch and a transfer learning approach using ResNet18. The experimental results indicate a clear performance difference between the two models.

The CNN model achieved an overall accuracy of **68.24%**, whereas the ResNet18-based model improved the accuracy to **70.72%**. In addition, ResNet18 outperformed the CNN model in terms of Macro Average F1-score (0.6975 compared to 0.6723) and Weighted Average F1-score (0.7042 compared to

0.6793). These results demonstrate that transfer learning provides better generalization across emotion classes, particularly under class imbalance conditions.

From a class-wise perspective, both models achieved their highest performance on the *Happy* and *Surprise* emotions. However, ResNet18 slightly improved the recognition of the *Happy* class, achieving an F1-score of 0.8839 compared to 0.8764 for the CNN model. The *Disgust* class, which contains the fewest samples, also benefited from transfer learning, with its F1-score increasing from 0.7273 to 0.7423.

Despite these improvements, *Fear* and *Sad* remain challenging emotions for both models and are frequently confused with *Neutral* and *Angry*. Nevertheless, ResNet18 consistently improved recall and precision for these classes. The *Neutral* class continues to exhibit confusion with negative emotions; however, the ResNet18 model reduces this misclassification, as reflected in the confusion matrix.

Analysis of the confusion matrices shows that the CNN model produces more scattered misclassifications, particularly among the *Fear–Sad–Neutral* and *Angry–Neutral* emotion groups. In contrast, the ResNet18 model exhibits stronger diagonal dominance, indicating improved class separability and more confident predictions.

In conclusion, the results confirm that transfer learning using ResNet18 is more effective than training a CNN from scratch for facial emotion recognition on the FER2013 dataset. By leveraging pretrained deep feature representations, the ResNet18 model achieves higher accuracy, better F1-scores, and improved robustness across emotion categories. Consequently, ResNet18 is a more suitable choice for practical facial emotion detection systems, especially when the available training data is limited or imbalanced.

Future work may explore deeper architectures, fine-grained attention mechanisms, self-supervised pretraining on facial datasets, or multi-task learning approaches to further improve recognition of subtle and ambiguous emotions.

# 7 Work Assignment

| Name | Student ID | Work assignment |
|---|---|---|
| Nguyen Thai Anh Minh | 20235605 | CNN model, report writing |
| Dang Trung Anh | 20235583 | CNN model, report writing |
| Pham Duc Duy | 20235588 | CNN model, slide presentation |
| Bui Cong Minh | 20235601 | Transfer Learning using Resnet18, slide presentation |

Our source code is available at this GitHub link:

**https://github.com/ImmortalZeus/IT3320E_IntroductionToDeepLearning_Group4**

# References

[1] PyTorch: An Open-Source Deep Learning Framework. https://pytorch.org/, 2025. Accessed: Nov 2025.

[2] GeeksforGeeks. Deep Learning Tutorial. https://www.geeksforgeeks.org/deep-learning/deep-learning-tutorial/, 2025. Last updated: 23 Jul 2025, Accessed: Nov 2025.

[3] Than Quang Khoat. IT3320E - Deep Learning Lecture Slides. Hanoi University of Science and Technology, 2025.

[4] M. Sambare. FER2013: Facial Expression Recognition 2013 Dataset. https://www.kaggle.com/datasets/msambare/fer2013. Accessed: Nov 2025.