

**VIETNAM GENERAL CONFEDERATION OF LABOR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY**



**PHAN ANH KIET - 523H0046
NGUYEN QUANG HUY - 523H0140**

FINAL REPORT

INTRODUCTION TO MACHINE LEARNING

HO CHI MINH CITY, 2025

**VIETNAM GENERAL CONFEDERATION OF LABOR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY**



**PHAN ANH KIET - 523H0046
NGUYEN QUANG HUY - 523H0140**

FINAL REPORT

INTRODUCTION TO MACHINE LEARNING

Advised by
Dr. Tran Luong Quoc Dai

HO CHI MINH CITY, 2025

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to Dr. Tran Luong Quoc Dai, our instructor and mentor, for his valuable guidance and support throughout the report. He has been very helpful and patient in providing us with constructive feedback and suggestions to improve our work. He has also encouraged us to explore new technologies and techniques to enhance our system's functionality and performance. We have learned a lot from his expertise and experience in web development and software engineering. We are honored and privileged to have him as our teacher and supervisor.

Ho Chi Minh city, 12th December 2025.

Author

(Signature and full name)

Kiet

Phan Anh Kiet

Huy

Nguyen Quang Huy

DECLARATION OF AUTHORSHIP

We hereby declare that this is our own project and is guided by Dr. Tran Luong Quoc Dai. The content research and results contained herein are central and have not been published in any form before. The data in the tables for analysis, comments and evaluation are collected by the main author from different sources, which are clearly stated in the reference section.

In addition, the project also uses some comments, assessments as well as data of other authors, other organizations with citations and annotated sources.

If something wrong happens, we'll take full responsibility for the content of my project. Ton Duc Thang University is not related to the infringing rights, the copyrights that We give during the implementation process (if any).

Ho Chi Minh city, 12th December 2025.

Author

(Signature and full name)

Kiet

Phan Anh Kiet

Huy

Nguyen Quang Huy

TABLE OF CONTENTS

TABLE OF CONTENTS.....	3
LIST OF FIGURES.....	4
LIST OF TABLES.....	5
CHAPTER 1. FEEDFORWARD NEURAL NETWORKS.....	6
1.1 Theoretical Basis.....	6
1.1.1 Feedforward Neural Networks Architecture.....	6
1.1.2 Mean Squared Error.....	9
1.1.3 Back Propagation Mechanism.....	10
1.2 Python Implementation.....	12
1.2.1 Data Preparation.....	12
1.2.2 Model Architecture.....	13
1.2.3 Training Configuration.....	13
1.2.4 Results.....	13
1.3 Early Stopping and Overfitting Solving Method.....	14
1.3.1 Overfitting.....	14
1.3.2 Early Stopping.....	14
CHAPTER 2. CONVOLUTIONAL NEURAL NETWORK.....	16
2.1 Theoretical Basis.....	16
2.1.1 Convolutional Layer.....	16
2.1.2 Pooling Layer.....	18
2.1.3 Activation Function.....	19
2.1.4 Fully Connected and Output Layer.....	20
2.2 Comparison between CNN and FNN.....	21
2.3 Experiments and Analysis.....	22
2.3.1 Model Architecture.....	22
2.3.2 Overfitting Solving Techniques.....	23
2.3.3 Comparative Analysis of Regularization Effects.....	24
2.3.4 Conclusion.....	26
2.4 Expand Research.....	27
2.4.1 Methodology.....	27
2.4.2 Experimental Results.....	28
2.4.3 Comparision.....	28

LIST OF FIGURES

Figure 1.1.1. Feed-forward Neural Network.....	6
Figure 1.1.2. 2D Image Flattening Visualization.....	7
Figure 1.1.3. Computation of Hidden Layer.....	7
Figure 1.1.4. Softmax Activation.....	8
Figure 1.2.2. Validation Loss (MSE) - Left and Validation (MAE) - right.....	14
Figure 1.3.1. Early Stopping Visualization.....	15
Figure 2.1.1. Convolution Neural Network Architecture.....	16
Figure 2.1.2. Kernel (Padding = 0) Visualization.....	17
Figure 2.1.3. Kernel (Padding = 0, Stride =1) Visualization.....	18
Figure 2.1.4. Pooling Visualization.....	19
Figure 2.1.5. ReLU Activation Function.....	20
Figure 2.3.1. Training and Validation Accuracy of Simple CNN and CNN with L2, Dropout.....	24
Figure 2.3.2. Validation Accuracy of CNN with different Overfitting Solving Techniques.....	25
Figure 2.3.3. Confusion Matrix of CNN with Dropout.....	25
Figure 2.4.1. VGG19 Transfer Learning.....	28

LIST OF TABLES

Table 1.2.1. FNN Architecture.....	13
Table 2.2.1. Comparison between CNN and FNN.....	21
Table 2.3.1. CNN Architecture.....	22
Table 2.3.2. Results On CIFAR-10 Test Dataset.....	24

CHAPTER 1. FEEDFORWARD NEURAL NETWORKS

1.1 Theoretical Basis

The Feed-forward Neural Network (FNN), frequently identified as the Multi-Layer Perceptron (MLP), represents a class of biologically inspired mathematical models designed to approximate complex, non-linear functions by mapping input vectors to output vectors through a series of intermediate computational layers.

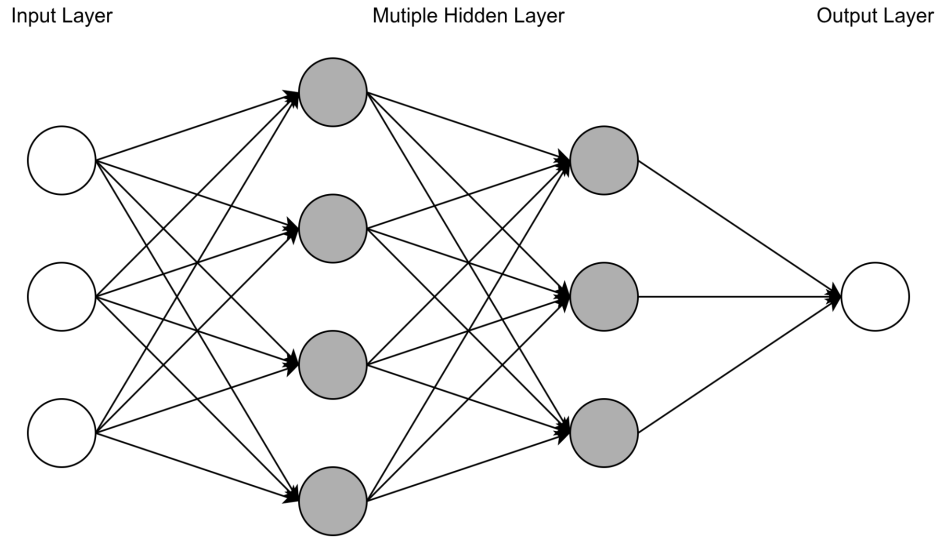


Figure 1.1.1. Feed-forward Neural Network

1.1.1 Feedforward Neural Networks Architecture

a. Input Layer

The input layer consists of neurons that receive the input data. Each neuron in the input layer represents a feature of the input data. For example, in Digital Image Processing, the input layer flattens the 2D grid of pixels into a 1D vector

$$X \in \mathbb{R}^{32 \times 32 \times 3}$$
$$x = \text{vec}(X) \in \mathbb{R}^{32 \times 32 \times 3 = 3072}$$

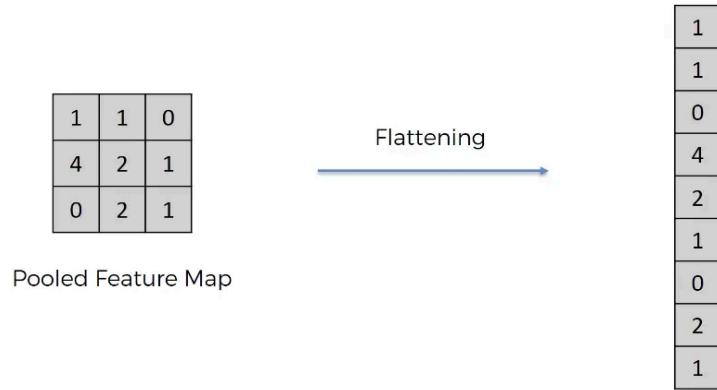


Figure 1.1.2. 2D Image Flattening Visualization

b. Hidden Layers

A Hidden Layer used for feature extraction and non-linear transformation. A network may possess multiple hidden layers. For a given layer l , the computation consists of two distinct steps (Linear Aggregation and Non-Linear Activation).

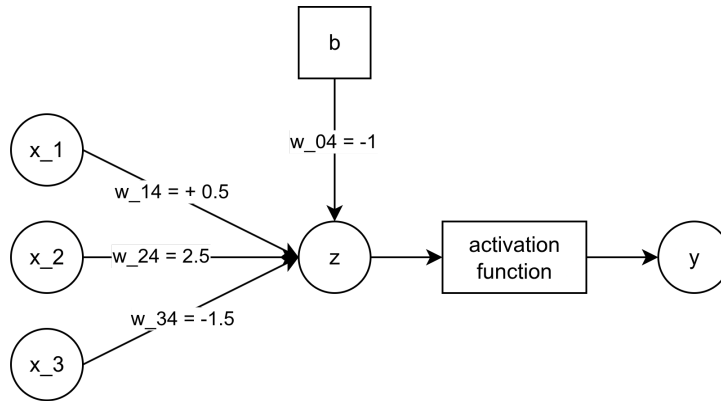


Figure 1.1.3. Computation of Hidden Layer

1. **Linear Aggregation:** The pre-activation value z is computed as a weighted sum of the inputs from the previous layer a plus a bias term b . The bias vector b allows the activation function to be shifted left or right, essential for fitting data that does not pass through the origin

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]} \text{ with } W^{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}$$

2. Non-Linear Activation: The linear output is passed through a non-linear activation function g to produce the activation a

$$a^{[l]} = g(x^{[l]})$$

Common activation functions include the Sigmoid, Hyperbolic Tangent (tanh) and the Rectified Linear Unit (ReLU)

$$\text{sigmoid} = \frac{1}{1 + e^{-x}}$$

$$\text{tanh} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\text{ReLU}(x) = \max(0, x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

Without non-linearity, the entire network, regardless of depth, would mathematically collapse into a single linear transformation stripping the model of its ability to learn complex patterns

$$Y = W_{total}X + B$$

c. Output Layer

The final layer returns the network's prediction. The choice of activation function depends on the task.

1. Regression: Linear activation allows for unbounded continuous output.
2. Binary Classification: Sigmoid, Tanh, ReLU, ... activation maps outputs to a probability range.
3. Multi-class Classification: Softmax activation produces a probability distribution over C classes such that total probability equal to 1.

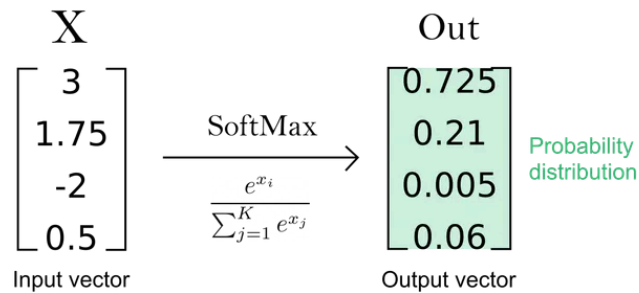


Figure 1.1.4. Softmax Activation

1.1.2 Mean Squared Error

To optimize the parameters, we must quantify the differences between the network's predictions and the ground truth. For regression tasks, the Mean Squared Error (MSE) is the common function.

a. Derivation from Maximum Likelihood Estimation (MLE)

Assume that the relationship between the input \mathbf{x} and the target \mathbf{y} is governed by a deterministic function, corrupted by irreducible Gaussian noise

$$y^{(i)} = f(x^{(i)}; \theta) + \epsilon^{(i)}, \quad \text{where } \epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$$

The probability density function for a single observation

$$p(y^{(i)}|x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y^{(i)} - \hat{y}^{(i)})^2}{2\sigma^2}\right)$$

The likelihood of the entire dataset

$$\mathcal{D} = \left\{ \left(x^{(i)}, y^{(i)} \right) \right\}_{i=1}^m$$

assuming independent and identically distributed (i.i.d.) samples, is the product of individual probabilities

$$\mathcal{L}(\theta) = \prod_{i=1}^m p\left(y^{(i)}|x^{(i)}; \theta\right)$$

Maximizing the likelihood is equivalent to maximizing the log-likelihood

$$l(\theta) = \sum_{i=1}^m \ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \sum_{i=1}^m \frac{(y^{(i)} - \hat{y}^{(i)})^2}{2\sigma^2}$$

Since the first term is constant with respect to θ , maximizing the log-likelihood is equivalent to minimizing the negative of the second term. Removing the scaling factor

$$\frac{1}{2\sigma^2}$$

which does not affect the location of the minimum yields, the Sum of Squared Errors. Averaging over m examples gives the Mean Squared Error cost function:

$$J(W, b) = \frac{1}{2m} \sum_{i=1}^m \|\hat{y}^{(i)} - y^{(i)}\|^2$$

the factor 0.5 is often included to simplify the derivative, as:

$$\frac{d}{dx} x^2 = 2x$$

which cancels the fraction.

b. Properties of MSE:

1. Convexity: In linear regression, the MSE surface is strictly convex, guaranteeing a single global minimum. However, in neural networks with non-linear activations, the MSE landscape becomes non-convex, containing multiple local minima and saddle points.
2. Sensitivity to Outliers: Because errors are squared, large deviations are penalized heavily. This is beneficial when large errors are unacceptable but can lead to model instability if the data contains significant outliers.

1.1.3 Back Propagation Mechanism

Back Propagation is the algorithm of deep learning. It is a specific implementation of the chain rule of calculus used to compute the gradient of the loss function J with respect to every weight and bias in the network. These gradients indicate the direction and magnitude of the update required for each parameter to reduce the error. Consider a network with L layers. We aim to compute

$$\frac{\partial J}{\partial W^{[l]}} \text{ and } \frac{\partial J}{\partial b^{[l]}} \text{ for } l = 1 \dots L$$

The process involves two passes: a forward pass to compute activations and a backward pass to compute gradients.

a. The Gradient at the Output Layer

Let the loss for a single example be

$$L(\hat{y}, y) = \frac{1}{2} \left(a^{[L]} - y \right)^2$$

We define the error term as the partial derivative of the cost with respect to the pre-activation

$$\delta^{[L]} = \frac{\partial J}{\partial z^{[L]}}$$

Using the chain rule

$$\delta^{[L]} = \frac{\partial J}{\partial a^{[L]}} \cdot \frac{\partial a^{[L]}}{\partial z^{[L]}}$$

For MSE, if the output activation is Sigmoid

$$g'(z^{[L]}) = a^{[L]} (1 - a^{[L]})$$

if Linear

$$g'(z^{[L]}) = 1$$

Thus, for a standard regression scenario (Linear output)

$$\delta^{[L]} = (a^{[L]} - y)$$

b. Back propagating the Error to Hidden Layers

To find the error at layer l , we propagate backwards

$$\delta^{[l]} = \frac{\partial J}{\partial z^{[l]}} = \sum_j \frac{\partial J}{\partial z_j^{[l+1]}} \frac{\partial z_j^{[l+1]}}{\partial z^{[l]}}$$

In vectorized form, the recurrence relation is

$$\delta^{[l]} = ((W^{[l+1]})^T \delta^{[l+1]}) \odot g'^{[l]}(z^{[l]})$$

where \odot denotes the Hadamard (element-wise) product.

c. Computing Gradients for Parameters

Once $\delta^{[l]}$ is known, the gradients for weights and biases are straightforward:

$$\frac{\partial J}{\partial W^{[l]}} = \delta^{[l]} (a^{[l-1]})^T$$

For a batch of m examples, we average these gradients:

$$dW^{[l]} = \frac{1}{m} \delta^{[l]} (A^{[l-1]})^T$$
$$db^{[l]} = \frac{1}{m} \sum_{i=1}^m \delta^{[l]}(i)$$

d. Parameter Update (Gradient Descent)

$$W^{[l]} := W^{[l]} - \alpha dW^{[l]}$$
$$b^{[l]} := b^{[l]} - \alpha db^{[l]}$$

where *alpha* is the learning rate.

1.2 Python Implementation

The following section details the implementation of the Feedforward Neural Network (FNN) using TensorFlow/Keras, applied to the California Housing Price Prediction task.

1.2.1 Data Preparation

The experiment utilizes the California Housing Price Dataset for regression, where 8 demographic and housing features are used to predict the median house value. The data pipeline was implemented as follows:

1. Dataset Splitting: The data was systematically split for robust evaluation:
 - The original training set was split into a Training Set and a Validation Set (17.6% of the full set) for monitoring model performance during training.
 - A final Test Set (15% of the full set) was held out for the final, unbiased performance assessment.
2. Feature Scaling: All 8 input features were scaled using the StandardScaler to standardize their distribution (zero mean, unit variance). This is crucial for optimizing the convergence rate of gradient descent.

1.2.2 Model Architecture

The experimental FNN model is a Multi-Layer Perceptron (MLP) defined by the following sequential architecture:

Layer Type	Output Shape	Activation	Regularization	Purpose
Dense	64	ReLU	None	Feature transformation.
Dropout	64	(None)	Rate 0.2	Standard regularization.
Dense	32	ReLU	None	Secondary feature transformation.
Dense (Output)	1	Linear	None	Final continuous output.

Table 1.2.1. FNN Architecture

1.2.3 Training Configuration

The model compilation and training parameters were set as follows:

1. Optimizer: Adam was used for its adaptive learning rate capabilities.
2. Loss Function: Mean Squared Error (MSE) was chosen, as is standard for regression tasks.
3. Metrics: Mean Absolute Error (MAE) was monitored for a more interpretable measure of the average prediction error.
4. Regularization: The primary strategy was Early Stopping (monitor='val_loss', patience=10), which stops training when the validation error plateaus, restoring the best weights to prevent overfitting.

1.2.4 Results

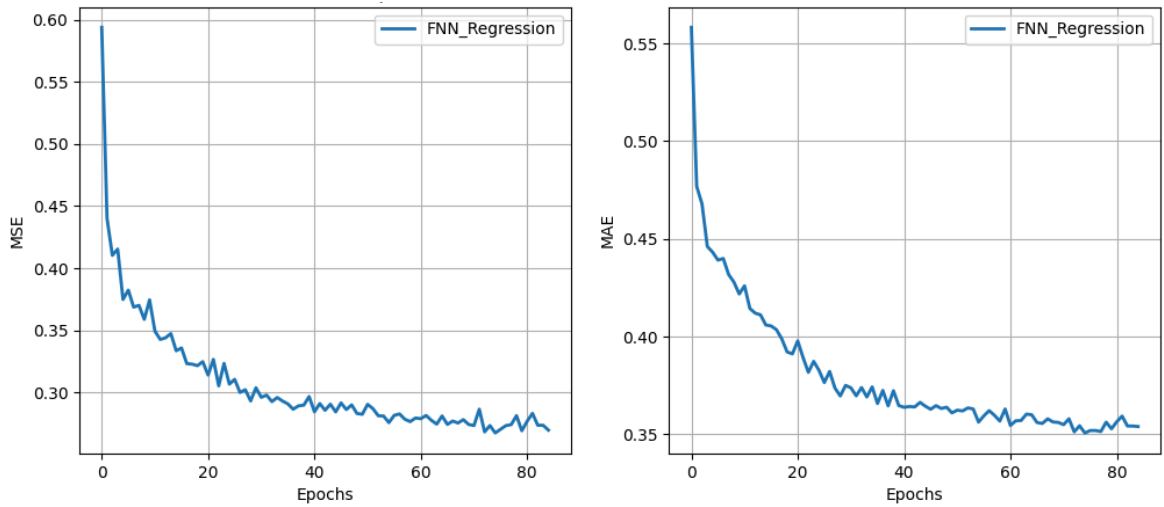


Figure 1.2.2. Validation Loss (MSE) - Left and Validation (MAE) - right

1.3 Early Stopping and Overfitting Solving Method

1.3.1 Overfitting

In the training of neural networks, a challenge is overfitting, where the model learns not only the underlying data distribution but also the stochastic noise inherent in the training set. This results in a model with low error on training data but poor generalization to unseen data. Mathematically, this corresponds to high variance in the bias-variance tradeoff. As training progresses, the model effectively "memorizes" the training examples, mapping specific input noise patterns to outputs rather than learning robust features.

1.3.2 Early Stopping

Early Stopping is a widely adopted regularization technique that mitigates overfitting by treating the number of training epochs as a tunable hyperparameter. It operates on the premise that the error on a held-out validation set will initially decrease (learning phase) but will eventually plateau and begin to increase (overfitting phase) as the model fits the training noise.



Figure 1.3.1. Early Stopping Visualization

Mechanism and Algorithm:

1. **Data Partitioning:** The available data is split into three subsets: Training (for gradient updates), Validation (for monitoring), and Test (for final evaluation).
2. **Monitoring:** At the end of each epoch, the model is evaluated on the validation set.
3. **Patience:** To account for stochastic fluctuations in the validation loss (which may be jagged), a patience parameter is introduced. This integer defines the number of epochs the system will wait for an improvement before terminating.
4. **Checkpointing:** The algorithm maintains a record of the model parameters that yielded the lowest validation loss observed so far.
5. **Termination:** If the validation loss does not improve after patience epochs, training is halted, and the model weights are effectively rolled back to best THETA.

CHAPTER 2. CONVOLUTIONAL NEURAL NETWORK

2.1 Theoretical Basis

A standard 224×224 color image contains over 150,000 pixels. In a fully connected network, a single neuron in the first hidden layer would require 150,000 weights. This curse of dimensionality leads to massive computational costs and extreme susceptibility to overfitting. Convolutional Neural Networks (CNNs) address this by embedding two distinct inductive biases into the architecture: local connectivity and translation invariance.

2.1.1 Convolutional Layer

The convolutional layer is the fundamental building block of a CNN, deriving its name from the mathematical operation of convolution.

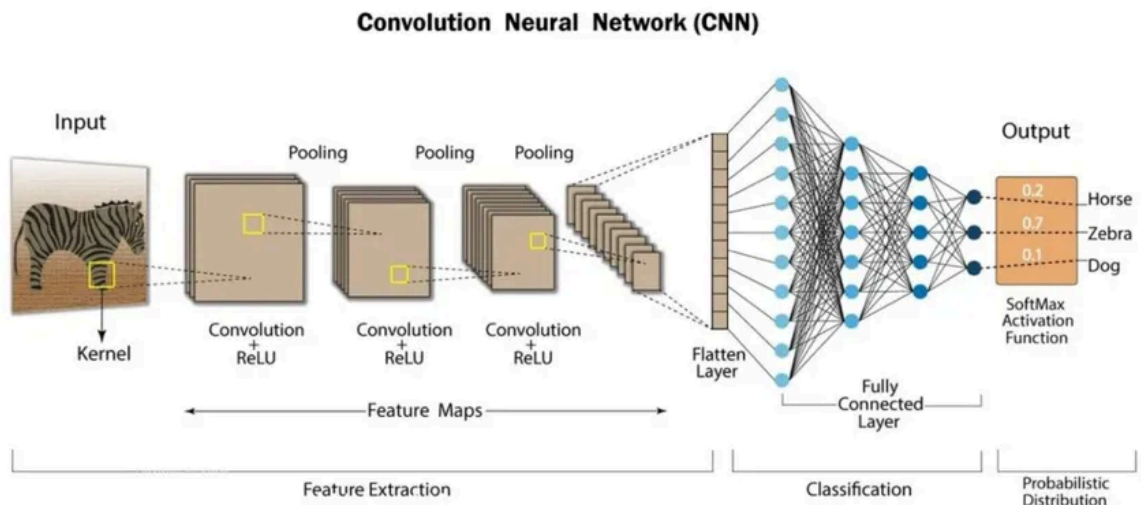


Figure 2.1.1. Convolution Neural Network Architecture

a. Mathematical Definition:

In the context of image processing, the operation is technically a discrete cross-correlation, though convention labels it convolution.

$$I \in \mathbb{R}^{H \times W}, K \in \mathbb{R}^{k_h \times k_w}$$

$$S(i, j) = (I * K)(i, j) = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} I(i+m, j+n) \cdot K(m, n)$$

The filter K slides (convolves) across the input image, computing the dot product between the filter weights and the local patch of input pixels at each position (i, j) .

b. Key Structural Parameters

1. Filters (Kernels): Unlike FNN neurons that look at the whole input, a filter looks at a small local region (receptive field), typically 3×3 or 5×5 . A filter is designed to detect specific features, such as vertical edges, color transitions, or corners.

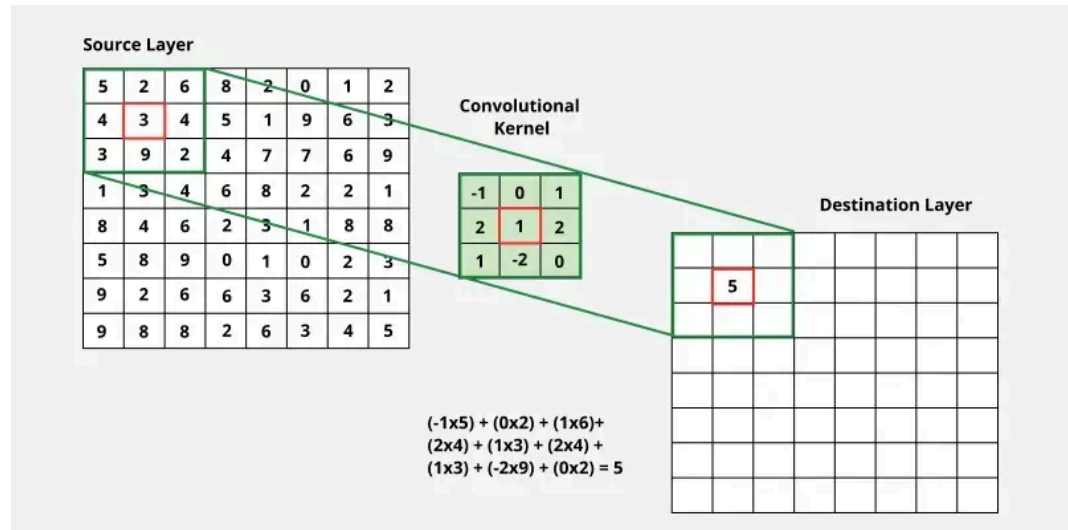


Figure 2.1.2. Kernel ($Padding = 0$) Visualization

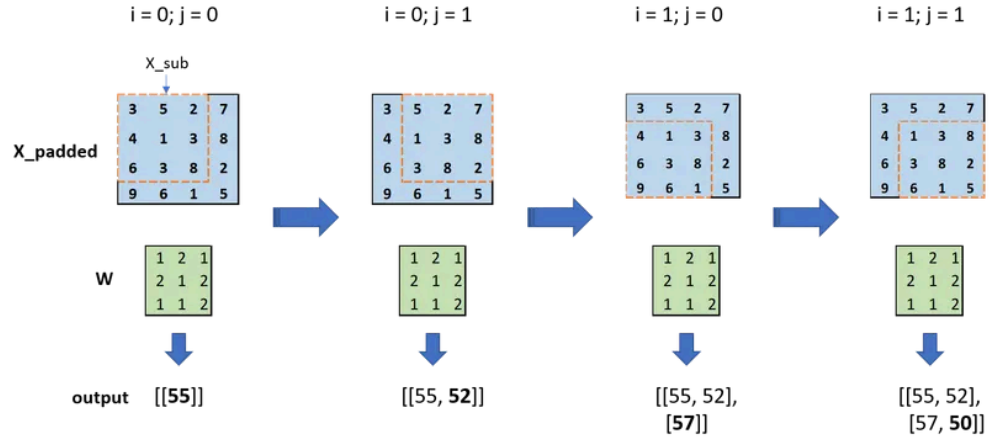


Figure 2.1.3. Kernel ($\text{Padding} = 0$, $\text{Stride} = 1$) Visualization

2. Stride (s): The step size by which the filter moves. A stride of $s=1$ means the filter shifts one pixel at a time. A stride of $s=2$ results in downsampling, effectively halving the spatial dimensions of the output.
3. Padding (p): To prevent the spatial dimensions from shrinking with every layer and to allow the filter to process edge pixels, zero-padding is often applied. "Same" padding ensures the output dimension matches the input dimension.

c. Parameter Sharing

The most significant theoretical innovation of the CNN is parameter sharing. A feature (e.g., an edge) is useful regardless of its location in the image. Therefore, the same filter K is used across the entire image. If a layer has F filters of size $k \times k$ connecting to C input channels, the total number of parameters is $(k \times k \times C + 1) \times F$. This is invariant to the image size, drastically reducing model complexity compared to FNNs.

2.1.2 Pooling Layer

Pooling layers typically follow convolutional layers and serve to progressively reduce the spatial size of the representation. This process is known as subsampling or downsampling.

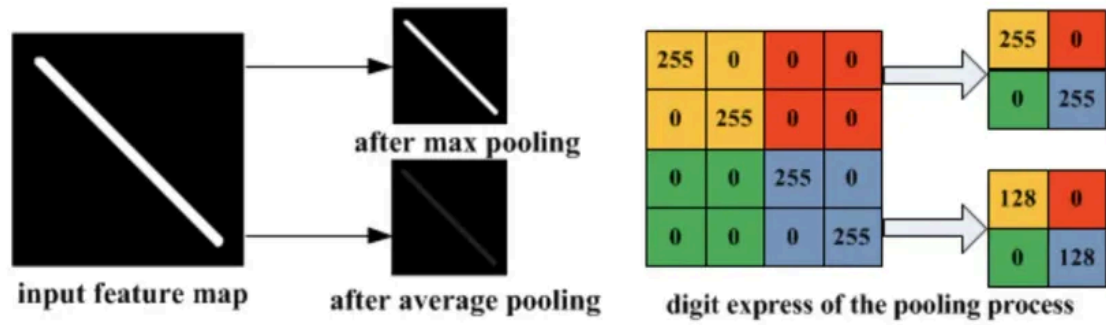


Figure 2.1.4. Pooling Visualization

a. Max Pooling

This operation selects the maximum pixel value within a defined window (e.g., 2×2).

$$y_{i,j} = \max_{(m,n) \in \mathcal{R}_{i,j}} x_{m,n}$$

Max pooling acts as a feature detector: if a feature (like an edge) creates a high activation anywhere in the window, the pooling layer preserves it. It is the most common pooling method as it introduces non-linearity and discards noisy background information.

b. Average Pooling

Computes the arithmetic mean of the window. While historically relevant (e.g., in LeNet-5), it has largely been superseded by Max Pooling, although Global Average Pooling is frequently used in the final layers of modern architectures like ResNet.

2.1.3 Activation Function

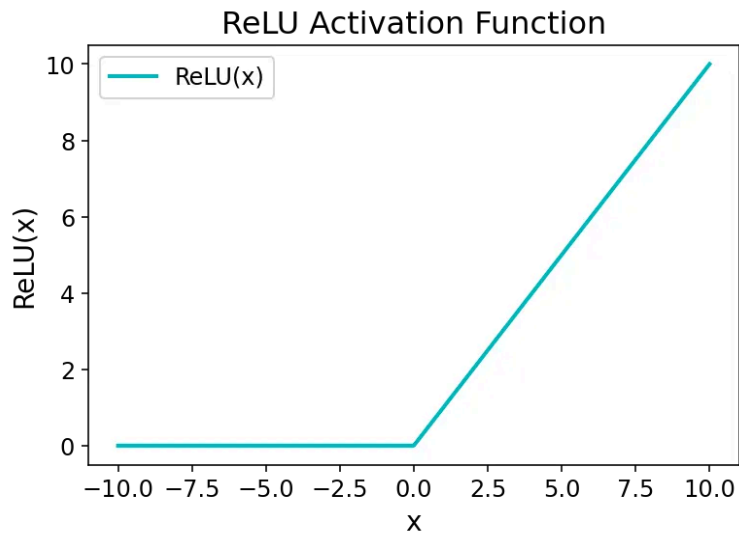


Figure 2.1.5. ReLU Activation Function

While the convolution operation is linear (a dot product), neural networks require non-linearity to model complex data. The Rectified Linear Unit (ReLU) is the standard activation function applied after each convolutional operation.

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

a. Advantages over Sigmoid/Tanh:

Sparsity: ReLU outputs zero for all negative inputs. In a randomly initialized network, about 50% of hidden units are active, leading to sparse representations that are computationally efficient.

Mitigation of Vanishing Gradient: Sigmoid and Tanh functions saturate at their extremes (derivatives approach zero), killing gradients in deep networks. ReLU has a constant gradient of 1 for positive inputs, allowing effective backpropagation through many layers.

2.1.4 Fully Connected and Output Layer

Following the extraction of spatial features by the convolutional and pooling blocks, the data is typically flattened into a 1D vector and passed through one or

more Fully Connected (Dense) layers. These layers function identically to the MLP layers described in Chapter 1.

- Role: They integrate the local, high-level features extracted by the convolutional layers (e.g., "eye", "nose", "ear") to perform global reasoning and classification (e.g., "cat").
- Output: For classification, the final layer uses the Softmax function to output a probability distribution over the classes.

$$P(y = j|x) = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$

This ensures the output values sum to 1, representing a valid probability distribution.

2.2 Comparison between CNN and FNN

Feature	Feedforward Neural Network (FNN)	Convolutional Neural Network (CNN)
Input Handling	Flattens input into 1D vector. Loses all spatial topology (pixel (x,y) is just index i).	Preserves 3D structure (Height, Width, Depth). Respects spatial relationships between pixels.
Connectivity	Dense (Global): Every neuron connects to all neurons in the previous layer.	Sparse (Local): Neurons connect only to a local receptive field defined by filter size.
Parameters	Extremely High: Scales with $H \times W$. For a 200×200 image and 100 hidden units: approx 12 million weights.	Low: Independent of image size due to sharing. Only depends on filter size and count.
Inductive Bias	Weak inductive bias. Must learn spatial invariance from scratch (requires massive data).	Strong spatial inductive bias: assumes locality and translation invariance.
Performance	Prone to overfitting on images. Cannot handle shifts/distortions well.	State-of-the-art for vision. Robust to translation, scaling, and distortion.

Table 2.2.1. Comparison between CNN and FNN

Consider an input image of $32 \times 32 \times 3$, and a first 32 neurons layer.

- FNN: $\text{Weights} = (3072 \times 32) + 32 = 98,336$ parameters.
- CNN: (Assuming 32 filters of size 3×3). $\text{Weights} = (3 \times 3 \times 3 + 1) \times 32 = 28 \times 32 = 896$ parameters. The CNN achieves the same depth of feature extraction with $< 1\%$ of the parameters, demonstrating superior efficiency.

2.3 Experiments and Analysis

2.3.1 Model Architecture

The CNN was built as a sequential model, consisting of three convolutional blocks followed by a classification head. The architecture is as follows:

Layer Type	Filters/ Neurons	Activation	Kernel/ Pool Size	Regularization	Purpose
Conv2D	32	ReLU	(3, 3)	None	Learn low-level features.
MaxPooling2D	-	(None)	(2, 2)	None	Downsamples feature map.
Dropout	32	(None)	-	Rate 0.2	Prevents co-adaptation of features.
Conv2D	64	ReLU	(3, 3)	None	Learn complex, higher-level features.
MaxPooling2D	-	(None)	(2, 2)	None	Further downsamples the feature map.
Dropout	64	(None)	-	Rate 0.2	Prevents co-adaptation of features.
Conv2D	64	ReLU	(3, 3)	L2 (1e-4)	Feature extraction with weight penalty.
Flatten	-	(None)	-	None	Prepares feature vectors for classification.
Dense	64	ReLU	-	L2 (1e-4)	Fully Connected Layer with weight penalty.
Dropout	64	(None)	-	Rate 0.5	Heavy regularization before the output.
Dense (Output)	10	Softmax	-	None	Final classification layer, providing probability distribution over 10 classes.

Table 2.3.1. CNN Architecture

The core principle of this design is that different layers are optimized to learn features at different levels of abstraction:

1. Early Layers (Low Filter Count - 32):

The initial Conv layers (with 32 filters) are used to identify low-level, universal features in the input image, such as edges, gradients, color blobs, and basic textures.

2. Deeper Layers (Increased Filter Count - 64):

As the network depth increases, the layers shift their focus to synthesizing high-level, abstract features (e.g., complex shapes, object parts like eyes, wheels, or specific patterns).

3. The Role of Pooling and Dimensionality:

Each Conv block is immediately followed by a MaxPooling2D layer, which aggressively reduces the spatial dimensions of the feature map (e.g., halving the width and height).

2.3.2 Overfitting Solving Techniques

1. Dropout: Multiple Dropout layers (with rates of 0.2 and 0.5) were placed to randomly deactivate neurons during training. This prevented the co-adaptation of features and forced the model to learn more distributed, independent representations.
2. L2 Regularization: L2 Weight Regularization (`L2_REG_STRENGTH = 0.0001`) is applied to the kernel weights of the third Conv2D layer and the first Dense layer in the classification head. This added a penalty to the loss function proportional to the square of the weight magnitudes, discouraging large weights and simplifying the decision boundary.
3. Early Stopping: This mechanism globally monitored the `val_loss` and restored the best weights, preventing the model from over-training on the noise within the training set.

2.3.3 Comparative Analysis of Regularization Effects

The initial comparison between the FNN and the Simple CNN demonstrated the foundational advantage of the CNN architecture. The Simple CNN achieved a 17.92% improvement in accuracy (0.6512 vs. 0.4720) because its use of Conv2D layers preserved the crucial spatial relationships in the images, unlike the FNN which lost this structure via Flatten.

- Dropout: The CNN with Dropout model achieved the highest Test Accuracy of 0.7368 and the lowest Test Loss.
- L2 Regularization: Applying L2 alone provided only a minimal uplift (0.6642) compared to the Simple CNN, indicating that weight magnitude was a less critical factor in overfitting than was the co-adaptation of features.
- Combined Regularization (L2, Dropout): The full model, combining both L2 and Dropout, maintained a high accuracy of 0.7267. While slightly lower than the Dropout-only model, this architecture is considered the most resilient and stable against potential overfitting spikes.

Model name	Test Accuracy	Test Loss
Simple CNN	0.6512	1.0042
CNN with Dropout	0.7368	0.7854
CNN with L2	0.6642	1.0084
CNN with L2, Dropout	0.7267	0.8647

Table 2.3.2. Results On CIFAR-10 Test Dataset

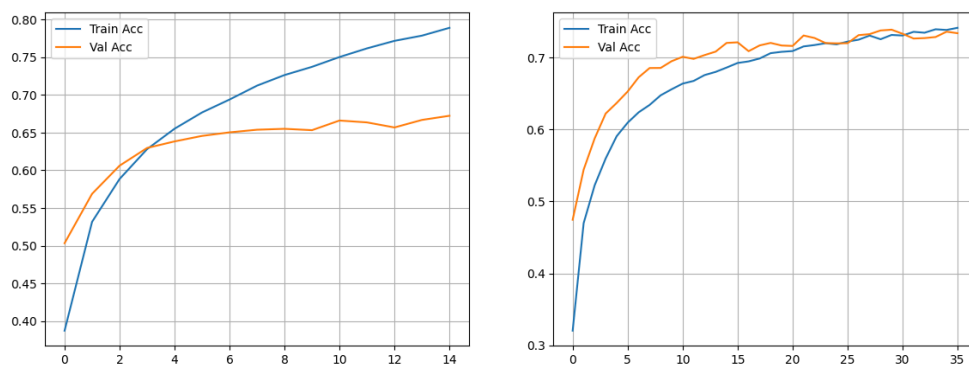


Figure 2.3.1. Training and Validation Accuracy of Simple CNN and CNN with L2, Dropout

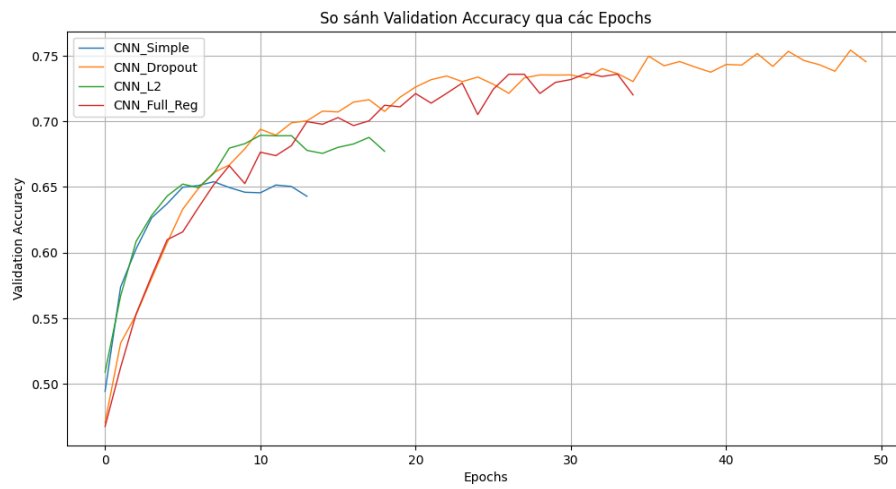


Figure 2.3.2. Validation Accuracy of CNN with differents Overfitting Solving Techniques

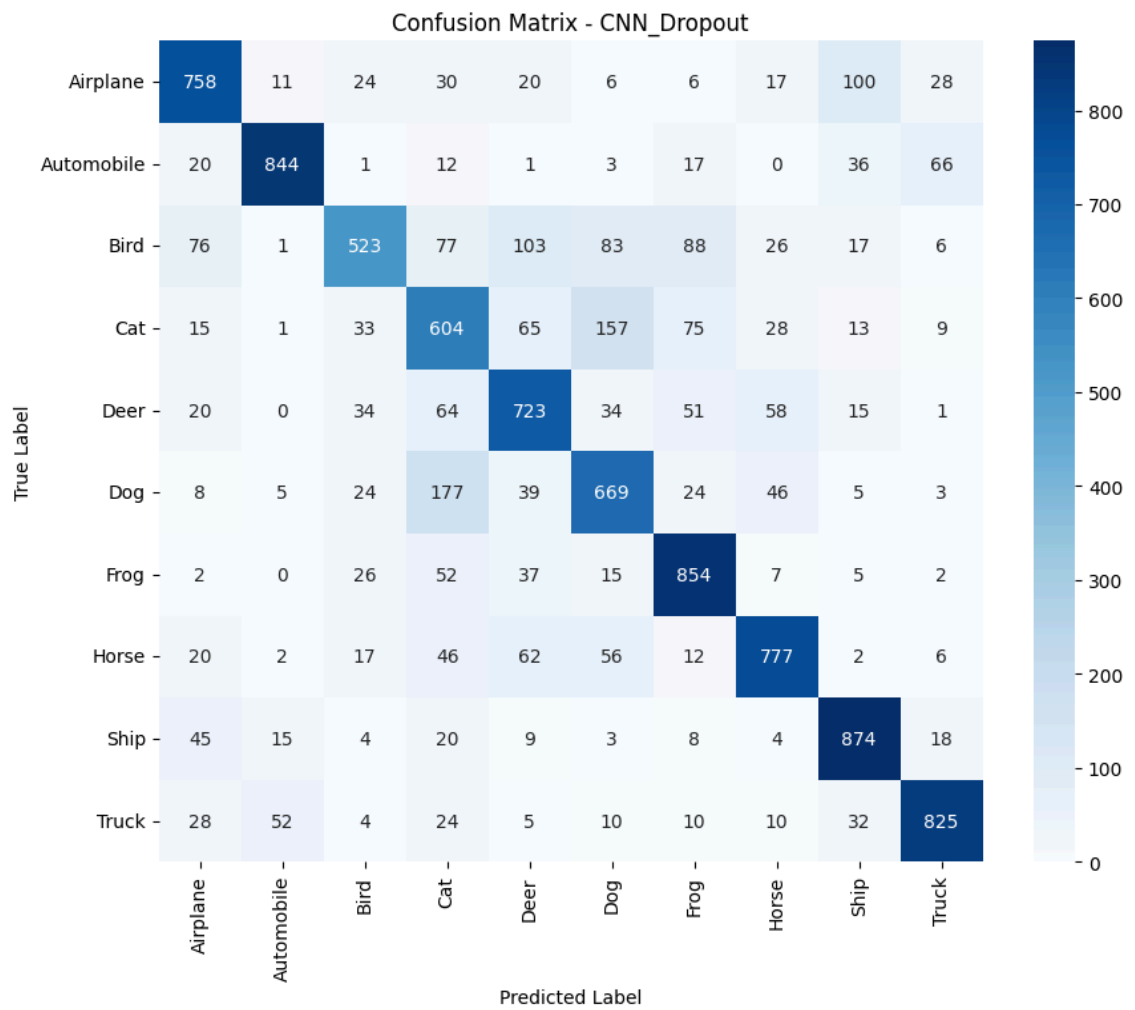


Figure 2.3.3. Confusion Matrix of CNN with Dropout

The Confusion Matrix (Figure 2.3.3) provides a granular view of the model's performance beyond a simple accuracy score, highlighting specific classification errors and which classes the model consistently misclassifies.

1. **Most Commonly Confused Classes:** A close examination of the matrix reveals that the highest misclassification rates occur predominantly among semantically similar animal classes, with the confusion between Cat and Dog being the most pronounced. This high confusion rate is typical for the CIFAR-10 dataset due to the significant visual overlap, such as similar textures, colors, and general structural features (e.g., fur, posture) shared between various images of cats and dogs. This indicates that the model struggles to learn the subtle, fine-grained features necessary to reliably distinguish between these two categories.
2. **Analysis of Frog and Cat Misclassification:** The matrix also shows a notable level of misclassification where images of Frog are incorrectly predicted as Cat (and vice versa). While these classes are semantically distinct (amphibian vs. mammal), this specific confusion points to a limitation in the model's feature extraction capabilities. It suggests that the convolutional layers may be over-relying on non-distinctive, low-level visual cues like certain green or brown textures, background patterns, or circular/blob-like shapes that happen to be common in some images of both frogs and cats. The model's inability to extract and utilize robust, high-level features such (e.g., distinctive body shape, leg structure) indicates a weakness in creating a truly discriminative feature hierarchy for all 10 classes.

2.3.4 Conclusion

The experimental results conclusively validate the design choices. The intrinsic architectural benefits of the CNN are a prerequisite for image classification, and the strategic integration of Dropout proved essential for

optimizing model generalization, resulting in the final high-performing model capable of accurate classification on the unseen CIFAR-10 test data.

2.4 Expand Research

2.4.1 Methodology

VGG19 is a deep convolutional network (19 layers) pre-trained on the ImageNet dataset, which contains over 14 million images. The intuition is that the early layers of VGG19 have already learned robust, generic low-level features (edges, textures, curves) that are transferable to the CIFAR-10 classification task.

Our experiment was conducted in two distinct phases:

1. Phase 1: Feature Extraction (Frozen Backbone)

We instantiated the VGG19 model without its top classification layer (`include_top=False`). The pre-trained weights were "frozen" (`trainable = False`) to prevent them from being destroyed by the large gradient updates of the randomly initialized classification head.

- Preprocessing: Input images were rescaled to the range $[-1, 1]$ to match the VGG19 requirements.
- Classification Head: We added a `GlobalAveragePooling2D` layer to reduce spatial dimensions, followed by a `Dropout (0.5)` layer for regularization, and a final `Dense` layer with `Softmax` activation for the 10 classes.
- Training: The model was trained for 15 epochs with a standard learning rate ($\alpha = 1e-3$).

2. Phase 2: Fine-Tuning

After the classification head converged, we "unfroze" the VGG19 backbone to allow the weights to be updated.

- Configuration: We utilized a very low learning rate ($\alpha = 1e-5$) to make minute adjustments to the pre-trained features without causing catastrophic forgetting.

- **Early Stopping:** To prevent overfitting during this sensitive phase, Early Stopping was employed with a patience of 5 epochs.

2.4.2 Experimental Results

The application of Transfer Learning yielded a significant improvement in performance compared to the custom CNN models.

- **Training Stability:** As shown in Figure 2.4.1, the accuracy exhibits a distinct jump after epoch 15, marking the transition from Feature Extraction to Fine-Tuning. This demonstrates the model's ability to adapt the deep features of VGG19 specifically for the CIFAR-10 dataset.
- **Final Accuracy:** The VGG19 Fine-tuned model achieved a Test Accuracy of 82.13%.

2.4.3 Comparision

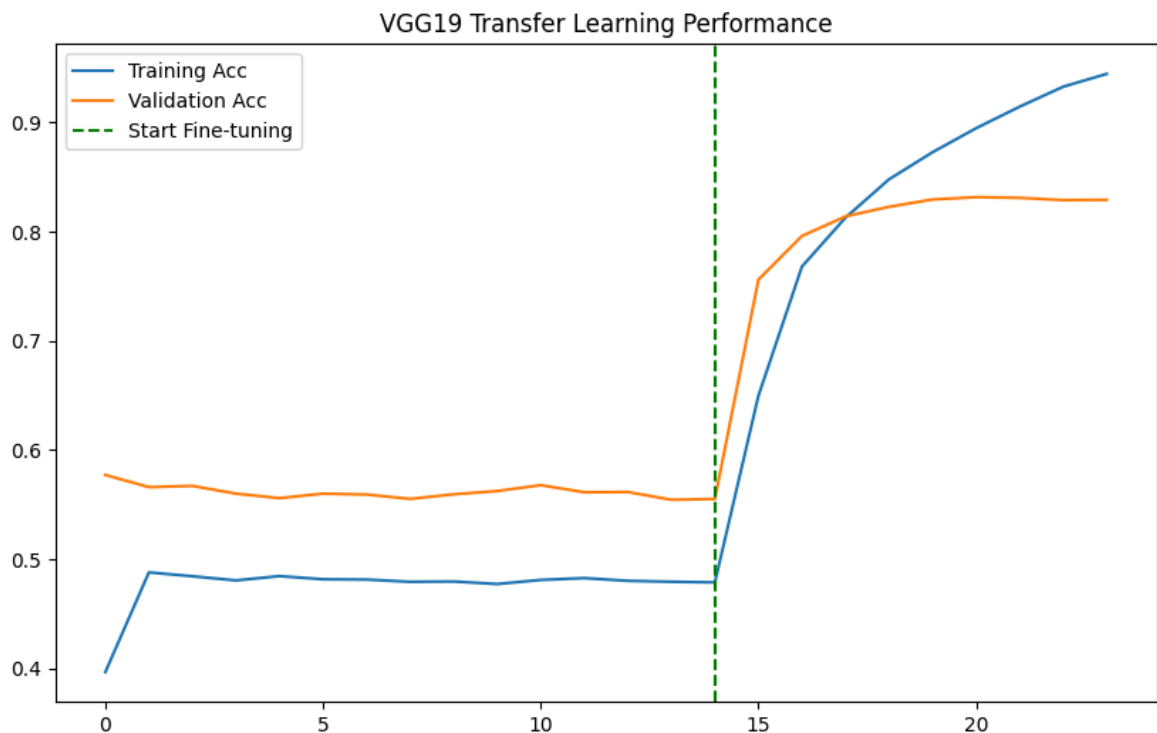


Figure 2.4.1. VGG19 Transfer Learning

Comparing this result to our previous best model (CNN with Dropout, 73.68% accuracy), the Transfer Learning approach provided an absolute improvement of approximately 8.5%. This confirms that leveraging pre-trained weights from large-scale datasets is a highly effective strategy for boosting model performance on complex image classification tasks, surpassing what is typically achievable with shallow, custom-built networks given limited training resources.