


DSA5205 Project 2: Convolutional Neural Networks

Hangxiao Yang¹ 

¹Faculty of Science, National University of Singapore, 597159, Singapore.

Contributing authors: E1351047@u.nus.edu;

Abstract

This report explores the principles and methodologies of Artificial Neural Networks (ANNs) and Convolutional Neural Networks (CNNs), emphasizing their architectures, training mechanisms, and applications. The study reviews various CNN models, including AlexNet, GoogLeNet, ResNet, DenseNet, and U-Net, highlighting their innovations and performance in image processing tasks. Experimental evaluations were conducted on the CIFAR-10 dataset, comparing the efficiency and accuracy of traditional machine learning models with advanced neural networks such as ResNet-18 and Vision Transformers (ViTs). The report also implements the U-Net model for image generation using Stable Diffusion, demonstrating its efficacy in generating CIFAR-10 images. Experiments indicate the superior performance of CNNs and ViTs in image classification tasks. The related code is open-sourced at: <https://github.com/yhx30/Project-5205>.

Keywords: BP, Artificial Neural Networks, Convolutional Neural Networks, ResNet, UNet

1 Introduction

Artificial Neural Networks (ANNs) are computational models inspired by the human brain's neural architecture, designed to recognize patterns and solve complex problems. Among these, Convolutional Neural Networks (CNNs) have gained prominence, especially in image and video analysis. CNNs utilize specialized layers, such as convolutional and pooling layers, to effectively capture spatial hierarchies in data, enhancing their ability to identify intricate patterns. This report delves into the foundational principles of ANNs and CNNs, exploring their architectures, learning mechanisms, and diverse applications across various domains.

2 Methodologies

2.1 Artificial Neural Networks and Single Neuron Model

The fundamental building block of an ANN is the neuron, which mathematically models the behavior of a biological neuron. The output of a single neuron can be expressed as Equation 1:

$$net_i = (\sum_{j=1}^d w_{ij} o_j + b_i), \quad (1)$$

where w_{ij} represents the connection weights, b_i is the bias term, and σ is the activation function. A commonly used activation function is ReLU (Rectified Linear Unit), defined as Equation 2:

$$\sigma(x) = \max(0, x). \quad (2)$$

Its derivative is given by:

$$\frac{\partial \sigma(x)}{\partial x} = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

The nonlinearity introduced by the activation function is essential for the learning and adaptive capabilities of the neural network.

2.2 Feedforward Multilayer ANN

A multilayer neural network connects multiple neurons to realize complex functions. For the $l - th$ layer, the output is computed as Equation 4:

$$o^{(l)} = \sigma(W^{(l)} o^{(l-1)} + b^{(l)}), \quad (4)$$

where $W^{(l)}$ is the weight matrix, $b^{(l)}$ is the bias vector, and $o^{(l-1)}$ is the output of the previous layer. The final output of the network is obtained through forward propagation across all layers.

2.3 Neural Network Training and Optimization

The training process aims to minimize the loss function, optimizing parameters θ to achieve:

$$\theta^* = \arg \min_{\theta} c(\theta) = \arg \min_{\theta} \sum_{n=1}^N L(y_n, f_{\theta}(x_n)), \quad (5)$$

where L is the loss function (e.g., Mean Squared Error or Cross-Entropy). Optimization methods such as Gradient Descent and Stochastic Gradient Descent (SGD) are commonly employed to iteratively update parameters.

In Gradient Descent, parameters are updated as:

$$\theta(t+1) = \theta(t) - \eta \Delta_c(\theta), \quad (6)$$

where η is the learning rate. However, gradient descent is susceptible to local minima, especially in high-dimensional, non-convex loss functions.

Chapter 6 introduces two main optimizers:

- **Stochastic Gradient Descent:**

Stochastic Gradient Descent (SGD) estimates gradients using a subset of data, introducing randomness to escape local minima. The update rule is:

$$\theta(t+1) = \theta(t) - \eta \nabla c_{mini-batch}(\theta). \quad (7)$$

- **Momentum Method:**

The Momentum Method accelerates convergence by incorporating historical gradients:

$$\begin{aligned} V(t) &= \beta V(t-1) - \eta \nabla c(\theta(t)), \\ \theta(t+1) &= \theta(t) + V(t), \end{aligned} \quad (8)$$

where β is the momentum coefficient controlling the influence of previous gradients.

2.4 Backpropagation Algorithm

Backpropagation (BP) is a core algorithm for efficiently computing gradients, essential for training neural networks. The key idea is to compute gradients layer by layer, using the chain rule to propagate errors from the output layer back to the input layer.

For the l -th layer, the error term is given by:

$$\delta^{(l)} = \sigma'(g^{(l)})(W^{(l+1)} \delta^{(l+1)}), \quad (9)$$

where $\sigma'(g^{(l)})$ is the derivative of the activation function, and $\delta^{(l+1)}$ is the error from the next layer. The gradient of the weights is computed as:

$$\frac{\partial c}{\partial W^{(l)}} = \delta^{(l)} (o^{(l-1)})^{\top}. \quad (10)$$

Forward propagation is used to compute layer outputs, followed by backpropagation to calculate gradients.

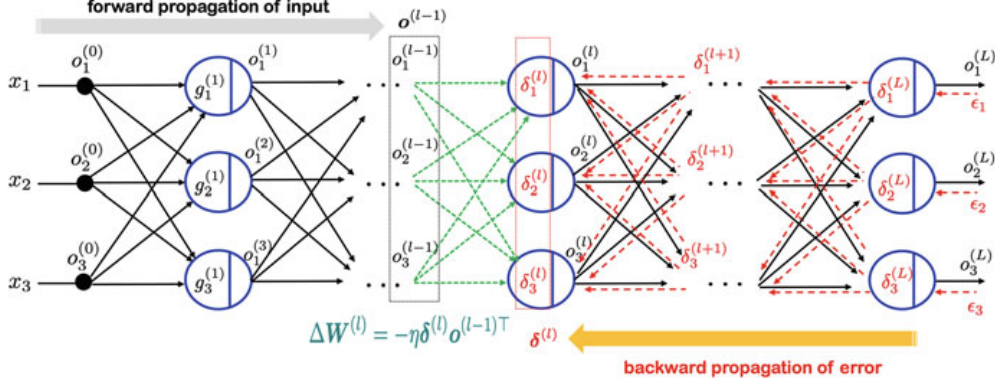


Fig. 1

2.5 Geometrical Interpretation and Gradient Updates

As shown in Figure 1, backpropagation can be interpreted geometrically as the outer product of the forward-propagated input signal $o^{(l-1)}$ and the backward-propagated error signal $\delta^{(l)}$. The weight update is performed using:

$$\Delta W^{(l)} = -\eta \delta^{(l)} (o^{(l-1)})^\top. \quad (11)$$

This update rule iteratively adjusts weights to minimize the loss function.

2.6 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specialized type of deep neural network widely used in image analysis and processing. Unlike the fully connected architecture of Multilayer Perceptrons (MLPs), CNNs utilize shared weights and convolution operations to significantly reduce the number of parameters while preserving the spatial features of images. The earliest CNN model, LeNet, was designed by Yann LeCun for handwritten digit recognition. Despite early limitations in computation power and optimization challenges (e.g., vanishing gradient problems), advances in hardware and algorithms have propelled CNNs to excel in image classification and other tasks.

2.7 Different CNN Models

Modern CNN models have undergone many variations, which shown as Table 1.

Table 1: Different variations of CNN

Model	Description
AlexNet	Proposed by Krizhevsky et al. for the 2012 ImageNet Challenge, AlexNet consists of five convolutional layers and three fully connected layers, with a key innovation being the use of the ReLU activation function. A classic model with high computational costs.

Model	Description
GoogLeNet	The winner of the 2014 ImageNet Challenge introduced the "Inception Module," which allows the network to extract features at multiple scales. This "network within a network" design enhanced feature representation while reducing parameters. Achieved the first breakthrough in large-scale deep learning applications.
VGGNet	VGGNet made a lasting impact by replacing large convolution kernels with consecutive small 3×3 kernels, significantly improving feature extraction capabilities. Its modular structure remains widely used for pre-trained models. Improved performance with smaller convolution kernels.
ResNet	ResNet introduced skip connections, enabling identity mapping that alleviates vanishing gradient problems in deep networks. This architecture allowed for the training of networks with hundreds or even thousands of layers. Extracted multi-scale features. Solved training challenges for deep networks with skip connections.
DenseNet	DenseNet employed an extreme form of skip connections, where each layer receives inputs from all preceding layers. This approach enhanced feature reuse and reduced the number of parameters. Maximized feature reuse.
U-Net	Originally designed for biomedical image segmentation, U-Net features a symmetric encoder-decoder architecture combined with skip connections. It is highly effective for tasks requiring global contextual information, such as image reconstruction and segmentation. Designed for segmentation and reconstruction tasks.

Through shared convolutional operations, these models demonstrate the powerful feature-learning capabilities of CNNs, achieving remarkable success across various domains.

2.8 Basic Components of CNNs

The basic CNN architecture includes the following components:

- **Convolution Operation:**

Convolution is the core operation in CNNs for extracting spatial features. Its general form is:

$$y[m, n] = \sum_{p, q} h[p, q] x[m - p, n - q], \quad (12)$$

where h is the convolution kernel and x is the input feature map. **Multi-input multi-output (MIMO)** convolutions and 1×1 convolutions are used for multi-channel feature extraction and dimensionality reduction, respectively.

- **Pooling and Unpooling:**

Pooling reduces dimensions, with common methods like max pooling and average pooling extracting spatially invariant features. Unpooling (e.g., transposed convolution) performs upsampling to increase spatial resolution.

- **Skip Connections:**

Skip connections facilitate identity mapping, improving gradient backpropagation and enhancing feature representation capabilities.

2.9 Training CNNs

2.9.1 Loss Functions

Training CNNs involves minimizing a loss function. Common examples include:

- **Softmax Loss:** For multi-class classification, it computes classification error using cross-entropy.
- **L1/L2 Loss:** For regression tasks, it measures the difference between predicted and target values.

2.9.2 Data Splitting

Training data is divided into training, validation, and test sets to monitor overfitting during training and evaluate model generalization.

2.9.3 Regularization

- **Data Augmentation:** New training samples are generated by transformations like rotation and flipping.
- **Parameter Regularization:** A penalty term is added to the loss to prevent overfitting.
- **Dropout:** Randomly disabling neurons at each iteration enhances robustness.

2.10 Visualizing CNNs

Visualization allows researchers to observe the evolution of features across layers. For example, early layers of VGGNet extract edge information, while deeper layers capture higher-order semantic features. This hierarchical feature extraction is analogous to biological visual processing.

3 Experiments

In this section, we implement the CNN models introduced in the book, along with some of their variations, using code.

3.1 Datasets

We conduct experiments on a large public image dataset, CIFAR-10¹.

The CIFAR-10 dataset is a widely used image dataset in the fields of machine learning and computer vision. It was collected and organized by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton at the University of Toronto. The dataset was created to provide a standard benchmark platform for image classification tasks, assisting researchers in evaluating and comparing the performance of different algorithms. Due to its moderate size and diverse categories, CIFAR-10 is extensively used to assess the performance of image classification algorithms, particularly in the training and testing of deep learning models.

CIFAR-10 consists of 60,000 color images that are evenly divided into 10 classes, each representing a distinct category. These classes are: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship and Truck. Each image in the dataset is of size 32×32 pixels, and the dataset includes 50,000 images in the training set and 10,000 images in the test set.

3.2 Results

On the CIFAR-10 dataset, we compare different models (Random Forest, PCA + SVM, ResNet-18, CNN, Vision Transformer), and the results are shown in Table 2.

Table 2: Compare 5 models.

Model Name	Loss	Accuracy (%)	Time per epoch (secs)
Random Forest	-	46.53	0.67
SVM	-	52.36	0.24
ResNet-18	0.0012	84.76	27.86
CNN	0.0131	86.90	16.18
ViT	1.4755	69.01	79.87

From the Table 2, it is evident that lightweight traditional machine learning models (Random Forest and PCA + SVM) have a significant advantage in training speed; however, their performance is comparatively poor. In contrast, neural network models with more parameters (ResNet-18, CNN, and ViT) require longer training times but achieve better peak performance. Among these, ResNet-18 incorporates a residual structure, significantly enhancing model performance. The deeper CNN model, which introduces receptive fields akin to attention mechanisms, is better suited for image recognition tasks and thus shows slightly superior performance at comparable parameter levels.

Due to computational constraints, we trained the more complex Transformer-based ViT model for only 50 epochs. Below, we present a visualization (as Figure 2) of the training processes for the three neural network models to analyze ViT’s advantages.

¹<https://www.cs.toronto.edu/~kriz/cifar.html>

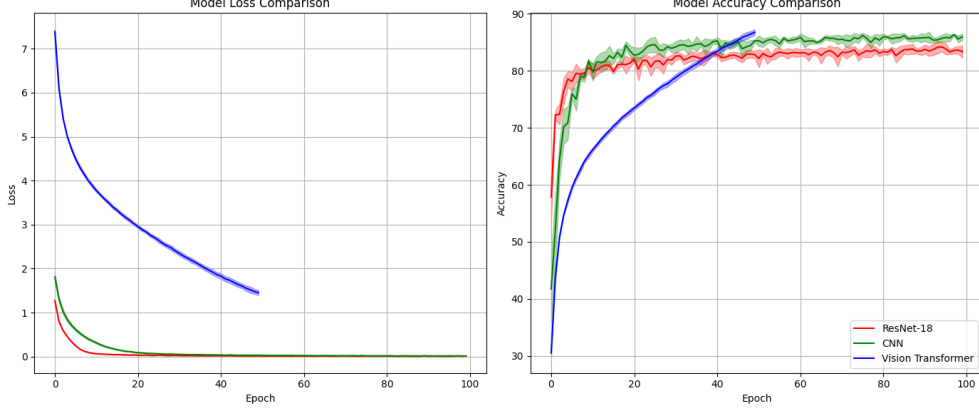


Fig. 2

From the Figure 2, it can be observed that ViT's learning process is more stable and quickly surpasses the other models, suggesting that, with sufficient computational resources, it could become the top-performing model.

3.3 UNet for Image Generation

For the UNet model introduced in the book, we use it to implement the Stable Diffusion algorithm for generating images in CIFAR-10.

First, we want to define the step that adds a little bit of noise $q(x_t|x_{t-1})$ (as shown as Figure 3).

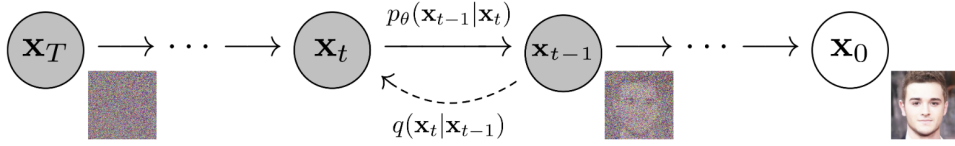


Fig. 3

We set up a 'variance schedule' β , where β_t specifies how much noise we want to add at that step. You get fancy schedules but we'll stick with a linear one for now. The formula you'll see for this single noise step is:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1 - \beta_t}x_{t-1}; \beta_t\mathbf{I}),$$

$$q(x_{1:T}|x_0) = \prod_{t=1}^T q(x_t|x_{t-1}). \quad (13)$$

We mix x_t with some gaussian noise, with how much noise decided by β_t .

Now, we want to train our model at different time steps and we don't particularly want to iteratively add little bits of noise a bunch of times just to train one sample from $t = 37$.

Luckily, some smart people did some fancy maths using something called the reparameterization trick that lets us get x_t for any t given x_0 , as shown in Equation 14.

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0; (1 - \bar{\alpha}_t)\mathbf{I}), \text{ where } \bar{\alpha}_t = \prod_{i=1}^T \alpha_i \quad (14)$$

Let's try applying it to an image (shown in Figure 4):

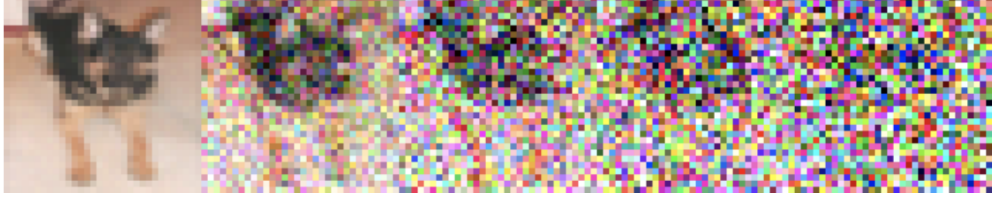


Fig. 4

Finally, we implemented a simple Stable Diffusion Model based on UNet and used it to generate a series of CIFAR-10 images(as shown as Figure 5). For detailed implementation, please refer to our repository².



Fig. 5

²<https://github.com/yhx30/Project-5205/blob/main/Project-5205-2/main.ipynb>