**DEPARTMENT OF ELECTRONIC AND TELECOMMUNICATION ENGINEERING**

**UNIVERSITY OF MORATUWA**



**EN3150 Assignment 03**

Simple convolutional neural network to perform classification

**ABEYGUNATHILAKA T.L. 200003P**

**MANIMOHAN T. 200377M**

**PUSHPAKUMARA H.M.R.M. 200488E**

**TILAKARATHNA U.A. 200664P**

December 02, 2023

Table of Contents

[Introduction 3](#_Toc152411373)

[CNN Model 4](#_Toc152411374)

[Model Parameters: 6](#_Toc152411375)

[Model Training 7](#_Toc152411376)

[Model evaluation 9](#_Toc152411377)

[For different learning rates 10](#_Toc152411378)

[Learning rate = 0.0001 11](#_Toc152411379)

[Learning rate = 0.001 13](#_Toc152411380)

[Learning rate = 0.01 15](#_Toc152411381)

[Learning rate = 0.1 17](#_Toc152411382)

[Pre-trained Models 19](#_Toc152411383)

[Resnet 19](#_Toc152411384)

[Densenet 20](#_Toc152411385)

[Googlenet 21](#_Toc152411386)

[Model Evaluation 22](#_Toc152411387)

[Comparison 23](#_Toc152411388)

[Discussion 24](#_Toc152411389)

[Custom Models 25](#_Toc152411390)

[Pre-trained Models 26](#_Toc152411391)

# Introduction

Why CNNs preferable for image classification over multilayered perceptrons (MLPs) or simple feedforward neural networks (NNs)?

* Local receptive fields:

CNNs can exploit the local receptive field property of images, which means that they can learn to identify patterns in small patches of an image without having to look at the entire image at once. This is a significant advantage over MLPs, which require the entire image to be flattened into a vector before they can be processed.

* Parameter sharing:

CNNs use parameter sharing, which means that the same weights are used to convolve over different parts of the image. This helps to reduce the number of parameters that the network needs to learn, and it also makes the network more robust to noise and variations in the image.

* Pooling:

CNNs use pooling layers to reduce the dimensionality of the output of the convolutional layers. This helps to prevent overfitting and makes the network more computationally efficient.

* Translation invariance:

CNNs are translation invariant, which means that they can recognize objects regardless of their position in the image. This is a valuable property for image classification tasks, as it means that the network does not need to learn a separate model for each possible position of an object.

* Hierarchical feature extraction:

CNNs can extract features from images at different scales, which allows them to learn more complex features than MLPs. This is important for image classification tasks, as it allows the network to learn to identify the key features that distinguish between different classes of objects.

# CNN Model

The architecture consists of three convolutional layers followed by max-pooling layers, and finally, two fully connected layers. The model also includes batch normalization, ReLU activation functions, and dropout for regularization.

### Convolutional Layers:

1. **Convolutional Layer 1:**
   * Input: 3 channels (for RGB images)
   * Output Channels: 32
   * Kernel Size: 5x5
   * Stride: 1
   * Padding: 2
   * Batch Normalization and ReLU Activation applied.
2. **Max Pooling Layer 1:**
   * Kernel Size: 2x2
   * Stride: 2
3. **Convolutional Layer 2:**
   * Input Channels: 32
   * Output Channels: 64
   * Kernel Size: 5x5
   * Stride: 1
   * Padding: 2
   * Batch Normalization and ReLU Activation applied.
4. **Max Pooling Layer 2:**
   * Kernel Size: 2x2
   * Stride: 2
5. **Convolutional Layer 3:**
   * Input Channels: 64
   * Output Channels: 128
   * Kernel Size: 5x5
   * Stride: 1
   * Padding: 2
   * Batch Normalization and ReLU Activation applied.
6. **Max Pooling Layer 3:**
   * Kernel Size: 2x2
   * Stride: 2

### Fully Connected Layers:

1. **Flatten Layer:**
   * Flattens the output from the last convolutional layer for input to fully connected layers.
2. **Fully Connected Layer 1:**
   * Input Features: 2048 [128 \* (32/8) \* (32/8)]
   * Output Features: 64
   * ReLU Activation applied.
   * Dropout with probability 0.5

### Output Layer:

1. **Fully Connected Layer 2 (Output Layer):**
   * Input Features: 64
   * Output Features: 10 (assuming 10 classes for classification)
   * SoftMax Activation applied for multiclass classification.

## Model Parameters:

The model includes several parameters, and the total number of learnable parameters can be calculated for each layer.

Convolutional Layer 1:

* Parameters: (Input Channels \* Output Channels \* Kernel Size \* Kernel Size) + Output Channels + Output Channels (for batch normalization)
* Total Parameters: (3 \* 32 \* 5 \* 5) + 32 + 32 = 2464

Convolutional Layer 2:

* Parameters: (32 \* 64 \* 5 \* 5) + 64 + 64 (for batch normalization)
* Total Parameters: (32 \* 64 \* 5 \* 5) + 64 + 64 = 51328

Convolutional Layer 3:

* Parameters: (64 \* 128 \* 5 \* 5) + 128 + 128 (for batch normalization)
* Total Parameters: (64 \* 128 \* 5 \* 5) + 128 + 128 = 205056

Fully Connected Layer 1:

* Parameters: (128 \* (32/8) \* (32/8) \* 64) + 64
* Total Parameters: (128 \* 4 \* 4 \* 64) + 64 = 131136

Output Layer:

* Parameters: (64 \* 10) + 10
* Total Parameters: (64 \* 10) + 10 = 650

1. Convolutional Layer 1: 2464 parameters
2. Convolutional Layer 2: 51328 parameters
3. Convolutional Layer 3: 205056 parameters
4. Fully Connected Layer 1: 131136 parameters
5. Output Layer: 650 parameters

The total number of learnable parameters in the entire CNN model is the sum of parameters from all layers:

2464+51328+205056+131136+650 = 390634

So, the CNN model has a total of **390,630** learnable parameters.

## Model Training

**Learning rate = 0.00025**

A screen shot of a black screen

Description automatically generated

A graph of a graph

Description automatically generated

**Why have we chosen Adam Optimizer over SGD?**

Adam and SGD are both optimization algorithms used in training deep learning models. Adam is an adaptive learning rate algorithm, which means it adjusts the learning rate for each parameter during training. This can be beneficial for models with sparse gradients or noisy data. SGD uses a fixed learning rate for all parameters, which can be more difficult to tune but can be more robust to noisy gradients. Adam generally converges faster than SGD, but SGD may perform better in the long run. Adam is also more sensitive to hyperparameters than SGD.

**Why have we chosen sparse categorical crossentropy as the loss function?**

* Single-Label Classification:

CIFAR-10 is a single-label classification problem, meaning that each image belongs to one and only one of ten possible classes (e.g., airplane, car, bird, etc.). Sparse categorical crossentropy is a loss function specifically designed for this type of problem, where class labels are represented as integers.

* Integer Labels:

CIFAR-10 uses integer labels (0 to 9) to represent the ten classes. Sparse categorical crossentropy directly handles these integer labels, eliminating the need for conversion to another format.

* No One-Hot Encoding Required:

Unlike categorical crossentropy, which requires labels to be one-hot encoded vectors, sparse categorical crossentropy allows you to provide class indices directly. This simplifies label representation and reduces computational overhead.

* Computational Efficiency:

Sparse categorical crossentropy is more computationally efficient than converting labels to one-hot encoded vectors. It avoids the need for extra memory and computation associated with one-hot encoding.

## Model evaluation

**Test Accuracy: 74.14%**

**Weighted Precision: 0.7443**

**Weighted Recall: 0.7414**

A graph with numbers and a chart

Description automatically generated with medium confidence

# For different learning rates

## Learning rate = 0.0001

A screenshot of a computer screen

Description automatically generated

A graph of a graph with blue lines

Description automatically generated with medium confidence

Test Accuracy: 61.56%

Weighted Precision: 0.5488

Weighted Recall: 0.6156

A diagram of a confusion matrix

Description automatically generated

## Learning rate = 0.001

A screenshot of a computer

Description automatically generated

A graph of a graph with blue and orange lines

Description automatically generated

Test Accuracy: 66.19%

Weighted Precision: 0.6897

Weighted Recall: 0.6619

A graph of a confused matrix

Description automatically generated with medium confidence

## Learning rate = 0.01

A screenshot of a computer screen

Description automatically generated

A graph of a graph with blue and orange lines

Description automatically generated

Test Accuracy: 21.49%

Weighted Precision: 0.1335

Weighted Recall: 0.2149

A diagram of a confused matrix

Description automatically generated

## Learning rate = 0.1

A screenshot of a computer screen

Description automatically generated

A graph with numbers and lines

Description automatically generated

Test Accuracy: 10.54%

Weighted Precision: 0.0111

Weighted Recall: 0.1054

A diagram of a confusion matrix

Description automatically generated

# Pre-trained Models

Choosed state-of-the-art pre-trained models

* 1. **ResNet**
  2. **Densenet**
  3. **Googlenet**

We used only 10 epochs when fine-tuning these models on the CIFAR-10 dataset because using more epochs led to overfitting. Overfitting is when a model learns the training data too well and is unable to generalize to new data. By limiting the training to 10 epochs, we were able to prevent overfitting and improve the models' performance on the test dataset.

## Resnet

training and validation loss values for each epoch

A screenshot of a computer

Description automatically generated

A graph of a graph with blue and orange lines

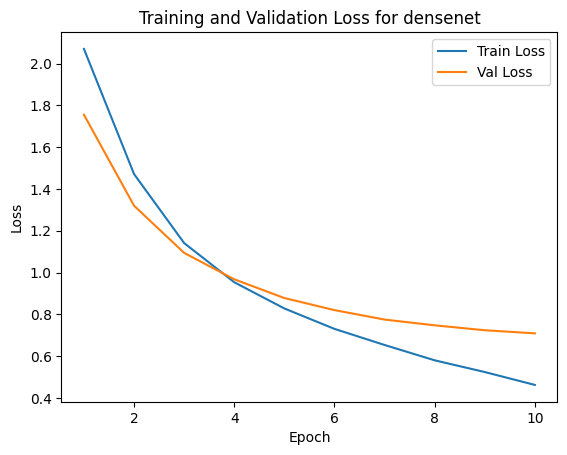
Description automatically generated

## Densenet

training and validation loss values for each epoch

A screenshot of a computer

Description automatically generated



## Googlenet

training and validation loss values for each epoch

A screenshot of a computer

Description automatically generated

A graph of a training and validation loss

Description automatically generated

## Model Evaluation

ResNet test accuracy: 73.05%

Densenet test accuracy: 76.18%

GoogleNet test accuracy: 66.47%

A black background with white text

Description automatically generated

# Comparison

Compare the test accuracy of your custom CNN model with that of the fine-tuned state-of-the-art models.

1. **Custom CNN Model (Test Accuracy: 74.14%):**
   * Simplicity: Our custom CNN model is relatively simple, which might be beneficial for the CIFAR-10 dataset.
   * Customization: We have control over the architecture, allowing us to tailor it to the specifics of the dataset.
   * **Possible Limitations:**
     + Limited Complexity: The model might not capture intricate patterns in the data compared to more complex architectures.
     + Data Augmentation: Further augmentation techniques could potentially improve performance.
2. **ResNet (Test Accuracy: 73.05%):**
   * Residual Connections: Help mitigate vanishing gradient problems, facilitating training of deep networks.
   * Feature Reuse: Dense connectivity allows features to be reused across layers.
   * **Possible Limitations:**
     + Complexity: ResNet may be too complex for the relatively small CIFAR-10 dataset, *leading to overfitting or suboptimal generalization*.
3. **DenseNet (Test Accuracy: 76.18%):**
   * Dense Connectivity: Facilitates feature reuse, enhancing gradient flow and learning.
   * Parameter Efficiency: It tends to be parameter-efficient due to the sharing of features.
   * **Possible Reasons for Higher Accuracy:**
     + Dense Connectivity: CIFAR-10 benefits from the enhanced feature reuse provided by DenseNet.
     + Efficient Parameter Usage: DenseNet might be utilizing parameters more effectively for this task.
4. **GoogleNet (Test Accuracy: 66.47%):**
   * Inception Modules: Capture multi-scale features effectively.
   * Auxiliary Classifiers: Aid in mitigating the vanishing gradient problem.
   * **Possible Limitations:**
     + Architecture Complexity: GoogleNet's intricate architecture may not be well-suited for the simplicity of CIFAR-10.

**Conclusion:**

* DenseNet achieved the highest accuracy (76.18%), suggesting that the dense connectivity and parameter efficiency are well-suited for CIFAR-10.
* Our custom CNN model performed competitively, likely benefiting from its simplicity.
* GoogleNet's lower accuracy could be due to its complex architecture, which might not be necessary for this dataset.

# Discussion

The decision of whether to use a custom model or a pre-trained model depends on several factors, including the specific task, the available data, the computational resources, and the expertise available.

## Custom Models

### Advantages:

* **Domain Specificity:** Custom models can be tailored specifically for the task or domain at hand, ensuring they are optimized for the specific requirements of the problem we are trying to solve.
* **Control over Architecture:** We have full control over the architecture of the model. This allows us to design a network structure that is well-suited for the intricacies of our data.
* **Performance Optimization:** Fine-tuning hyperparameters and optimizing the model's performance on our specific dataset can lead to better results compared to a pre-trained model that might have been trained on a different distribution of data.
* **Data Efficiency:** If we have a sufficiently large and representative dataset, training a custom model from scratch can lead to better generalization and performance.

### Limitations:

* **Data Dependency:** Training a custom model requires a substantial amount of labeled data. In domains where obtaining labeled data is difficult or expensive, pre-trained models might be a more practical choice.
* **Computational Resources:** Training a deep learning model from scratch demands significant computational resources and time, especially for large datasets and complex architectures.
* **Expertise Required:** Developing and training a custom model demands expertise in machine learning and deep learning.

## Pre-trained Models

### Advantages:

* **Transfer Learning:** Pre-trained models, especially those trained on large and diverse datasets, can be fine-tuned on smaller datasets for specific tasks. This often results in faster convergence and improved performance.
* **Time and Resource Efficiency:** Training a model from scratch can be computationally expensive and time-consuming. Pre-trained models save time and computational resources, as they have already learned useful features from extensive datasets.
* **Generalization:** Pre-trained models have often learned generic features from a broad range of data, which can lead to good generalization across various tasks.
* **Community Support:** Popular pre-trained models often have strong community support, with many resources, tutorials, and tools available for implementation and fine-tuning.

### Limitations:

* **Domain Mismatch:** Pre-trained models might not perform well if there's a significant mismatch between the data they were trained on and our specific task. This is particularly relevant when dealing with highly specialized or domain-specific tasks.
* **Limited Flexibility:** Pre-trained models have a fixed architecture, limiting our ability to customize the model for specific requirements. If our task demands a unique network structure, a custom model might be more appropriate.
* **Overfitting Risk:** Fine-tuning a pre-trained model on a small dataset carries the risk of overfitting, especially if the pre-trained model has already captured a broad set of features that may not be relevant to our specific task.

The choice between a custom model and a pre-trained model is a complex one, and there is no one-size-fits-all answer. The best approach will depend on the specific requirements of our task, the available data, and the resources at our disposal.

In general, custom models are a good choice when we have a lot of relevant data and the expertise to train a model, while pre-trained models are a good choice when we need a model quickly and easily or when we do not have a lot of data or expertise.

Here is a table summarizing the key trade-offs between custom and pre-trained models:

|  |  |  |
| --- | --- | --- |
| Feature | Custom Model | Pre-trained Model |
| Domain Specificity | High | Medium – Low |
| Control over Architecture | High | Low |
| Data Requirements | High | Low |
| Computational Requirements | High | Low |
| Expertise Requirements | High | Low |
| Ease of Use | Low | High |
| Time Efficiency | Low | High |
| Generalization | Medium – High | Medium |
| Interpretability | Medium – High | Medium – Low |
| Scalability | Medium – High | Medium – Low |

# GitHub

GitHub Link:

<https://github.com/RavinduMPK/EN3150-Pattern-Recognition/tree/main/Assignment%2003>