Department of Electronic & Telecommunication Engineering
University of Moratuwa

EN3150 - Pattern Recognition

Simple convolutional neural network to perform classification

3rd December 2023

A.T.P. Amarasekara	200023C
D.M.D.V. Bandara	200061N
A.M.P.S. Samarasekera	200558U
K.P. Tharuka	200641T

Contents

Github Repository

1 Introduction

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

Neural Networks (NNs)

Neural Networks, are a computational model inspired by the human brain's neural structure. They consist of interconnected neurons organized in layers. Data flows through these layers where each neuron receives inputs, processes them using weighted connections, and produces an output. NNs can have different architectures with how neurons in adjacent layers are connected to each other.

Multilayer Perceptrons (MLPs)

Multilayer Perceptrons are a specific type of Neural Network architecture consisting of multiple layers of nodes, including an input layer, one or more hidden layers, and an output layer. In an MLP, each neuron in a layer is connected to every neuron in the subsequent layer. They are trained using algorithms like backpropagation to adjust weights and biases, enabling them to learn and approximate complex functions.

Convolutional Neural Networks (CNNs)

Convolutional Neural Networks are a specialized type of neural network designed primarily for processing and classifying visual data such as images. CNNs use convolutional layers to extract features from input images by applying convolution operations with learnable filters. They also incorporate pooling layers to reduce spatial dimensions and learn hierarchical representations of features.

CNNs are specifically designed for image-related tasks, utilizing concepts like parameter sharing, spatial hierarchies, and local connectivity, making them highly efficient and effective for image classification compared to using MLPs or NNs.

- Utilization of spatial hierarchies

  CNNs preserve the spatial relationship between pixels by using convolutional layers, whereas MLPs treat each pixel as an independent feature.

- Parameter sharing

  CNNs use parameter sharing through convolutional kernels, reducing the number of parameters compared to fully connected layers in MLPs. This makes CNNs more efficient to train and less prone to overfitting.

- Translational invariance

  CNNs capture translational invariance within images due to their use of shared weights in convolutional layers, enabling recognition of patterns regardless of position.

- Local connectivity

  CNNs use local connectivity in input data, connecting neurons in convolutional layers to small regions of the input volume, allowing focus on local features.

- Pooling layer implementation

  CNNs use pooling layers to reduce spatial size progressively, lowering computational requirements and managing overfitting.

- Feature hierarchies

  CNNs learn hierarchical representations of features, where lower layers detect edges or simple shapes, and deeper layers learn more complex features.

- Pretrained models and transfer learning

  CNN architectures often trained on large datasets are readily available and can be fine-tuned for specific tasks using transfer learning. This reduces the need for training large models from scratch when working with limited datasets.

2 CNN Architecture and parameters

The convolutional neural network implemented contains two convolutional layers with ReLU activation, each followed by max-pooling layer for downsampling. It is then followed by two fully connected layers. The first fully connected layer has 128 neurons with ReLU activation, batch normalization, and dropout of 50%. Finally, the output layer comprises 10 neurons using softmax activation for classification.

2.1 Convolutional Layers

1. Convolutional Layer 1
   - Filters- 64
   - Kernel Size- 3x3
   - Activation- ReLU
   - Padding- 1
2. Max Pooling Layer 1
   - Kernel Size- 2x2
3. Convolutional Layer 2
   - Filters- 32
   - Kernel Size- 3x3
   - Activation- ReLU
   - Padding- 1
4. Max Pooling Layer 2
   - Kernel Size- 2x2

2.2 Fully Connected Layers

1. Flatten Layer
   - Flattening the output into a 1D array
2. Fully Connected Layer 1
   - Neurons- 128
   - Activation- ReLU
   - Batch Normalization
   - Dropout- 50
3. Fully Connected Layer 2 (Output Layer)
   - Neurons- 10
   - Activation- Softmax

3 Training the Model

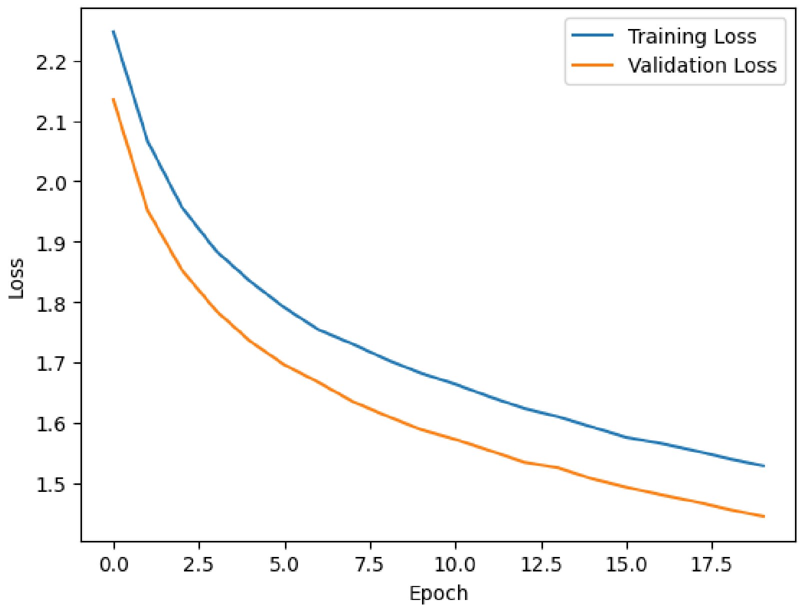


Figure 1: The training and validation loss with each epoch

4 Chosen Optimizer - Adam Optimizer

The Adam optimizer offers several advantages over Stochastic Gradient Descent (SGD). One key advantage is its adaptive learning rate mechanism. Unlike SGD, Adam computes individual learning rates for each parameter during training. This adaptability allows it to adjust learning rates based on the historical gradients, enabling faster convergence in various scenarios. Moreover, Adam includes momentum optimization similar to SGD with momentum, facilitating accelerated gradients in the correct directions while damping oscillations. This feature proves beneficial in high-dimensional datasets and complex models. The Adam optimizer is effective in handling noisy gradients and sparse data. By maintaining separate learning rates for each parameter, it handles noisy data and sparse gradients well, making it suitable for challenging datasets. Moreover, Adam optimizer is less sensitive to hyperparameters than SGD, with default settings that often perform well across various domains. Therefore, the Adam optimizer is used in place of SGD.

5 Chosen Loss function - Sparse categorical cross-entropy

The utilization of sparse categorical cross-entropy as the chosen loss function offers several advantages in the context of multi-class classification tasks. The loss function is specifically designed for scenarios where each sample belongs to a single class among multiple classes, it efficiently handles integer-labeled data without the need for encoding. By directly using integer labels to represent class membership, it significantly reduces both memory consumption and computational overhead compared to methods that require one-hot encoded labels. Additionally, sparse categorical cross-entropy is efficient when dealing with a substantial number of classes, as it avoids the potential memory-intensive nature of one-hot encoding, making it particularly efficient when a large and diverse range of classes are required to be classified.

6 Model Evaluation

The following results were obtained for the evaluation.
Test accuracy: 0.4966000020503998
Precision: 0.4901497614770582
Recall: 0.4966

The obtained confusion matrix is as follows,

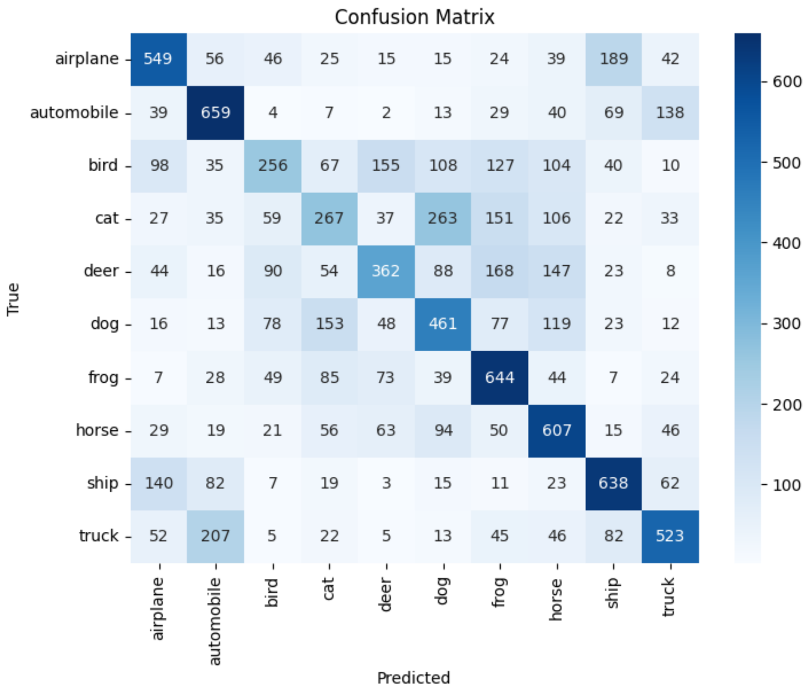


Figure 2: Confusion matrix

7 Learning Rate variation

The variation of the training and validation loss with the learning rate was observed as follows.

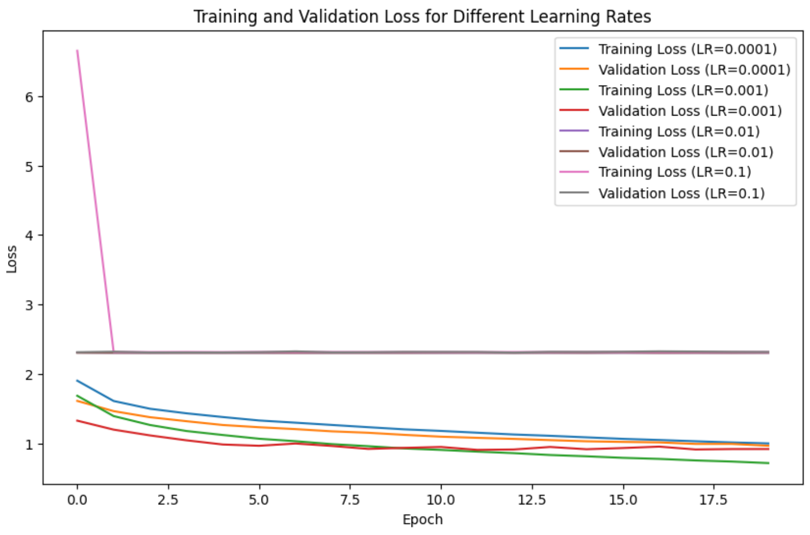


Figure 3: Loss with Learning rates

8 Comparison with State of the Art Models

The two chosen models are ResNet50 and VGG16.

8.1 ResNet50 implementation

8.1.1 Model Training

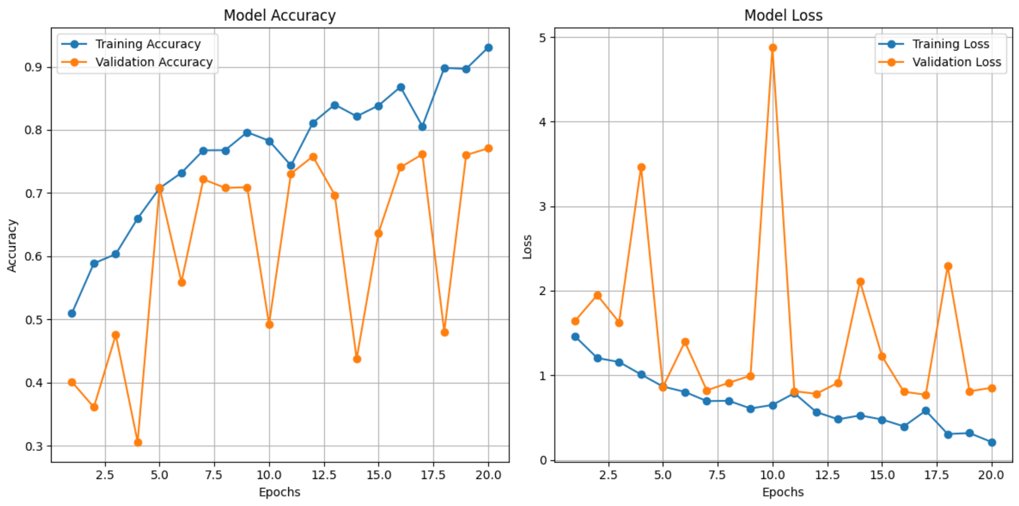


Figure 4: Training Loss and Accuracy of ResNet50 Model

8.1.2 Model Evaluation

The evaluation metrics are as follows;
Test accuracy: 0.7649000287055969

Precision: 0.7710694254347904
Recall: 0.7649

The obtained confusion matrix is,

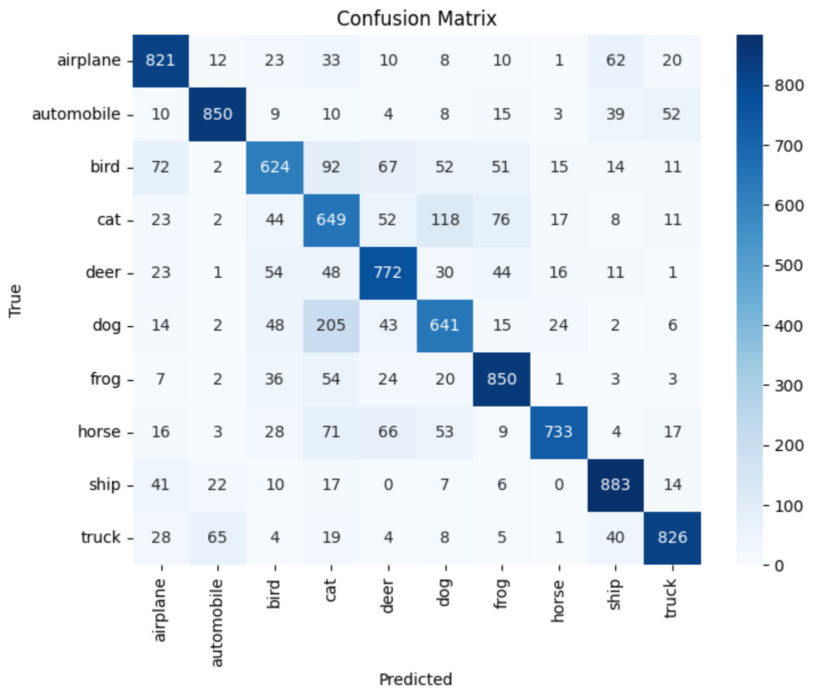


Figure 5: Confusion Matrix for ResNet model

8.2 VGG16 implementation

8.2.1 Model Training

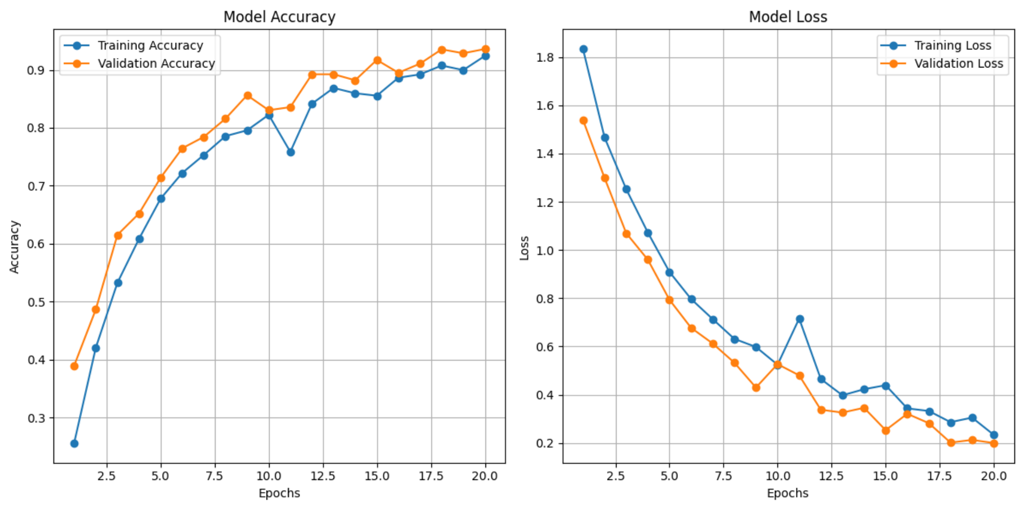
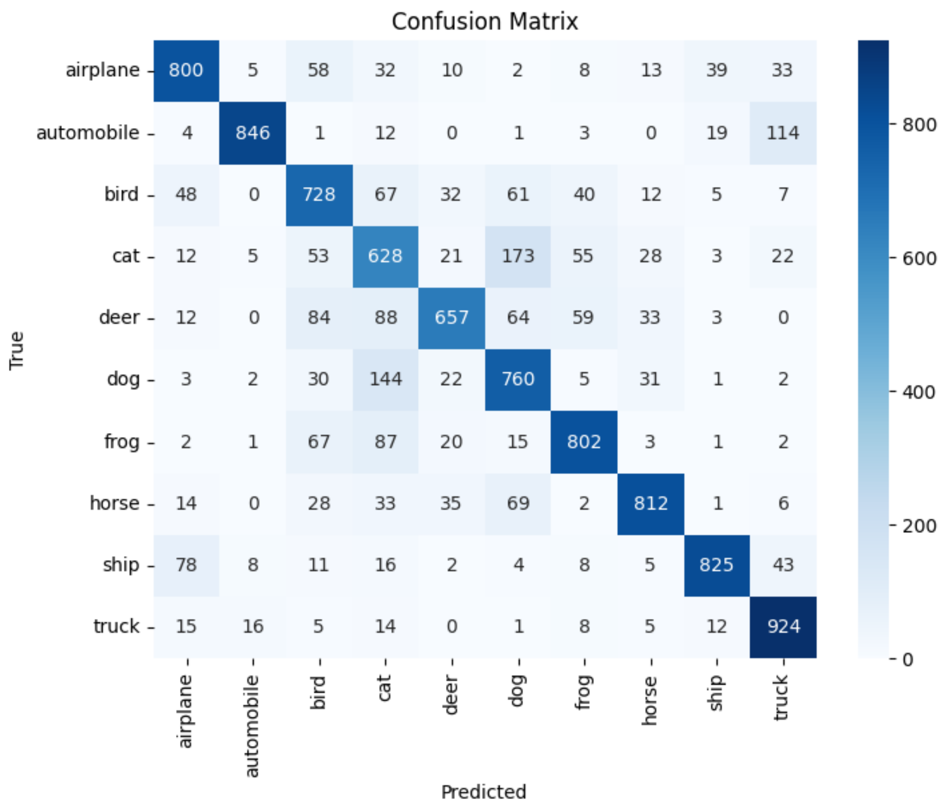


Figure 6: Training loss and accuracy for VGG Model

8.2.2 Model Evaluation

The evaluation metrics are as follows;
Test accuracy: 0.7781999707221985

Precision: 0.78758187783830338
Recall: 0.7782



True \ Predicted	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	800	5	58	32	10	2	8	13	39	33
automobile	4	846	1	12	0	1	3	0	19	114
bird	48	0	728	67	32	61	40	12	5	7
cat	12	5	53	628	21	173	55	28	3	22
deer	12	0	84	88	657	64	59	33	3	0
dog	3	2	30	144	22	760	5	31	1	2
frog	2	1	67	87	20	15	802	3	1	2
horse	14	0	28	33	35	69	2	812	1	6
ship	78	8	11	16	2	4	8	5	825	43
truck	15	16	5	14	0	1	8	5	12	924

Figure 7: Confusion Matrix for VGG Model

8.3 Model comparisons

Model	Accuracy	Precision	Recall	F1-score
Initial Model	0.4966	0.4901	0.4966	0.4933
ResNet50	0.7782	0.7877	0.7782	0.7829
VGG16	0.7782	0.7876	0.7782	0.7829

Table 1: Comparison of CNN Models Implemented

The initial model performs significantly lower compared to the other models, with accuracy around 49.66% across all metrics. Both state of the art models used, ResNet50 and VGG16 exhibit significantly better performance across all metrics compared to the Initial Model, with almost identical results in terms of accuracy, precision, recall, and F1-score. These models showcase approximately 28-30% improvement over the Initial Model implemented.

8.4 Discussion

8.4.1 Custom Model

Advantages

- Tailored to specific task

  Custom models can be designed specifically for the task, allowing customization of architecture and parameters as required to the task.

- Flexibility and control

  Full control over network architecture enables experimentation and customization of different features and parameters.

- Potential for better performance

  Custom models might perform well on datasets significantly different from pre-trained models due to the possibility of optimization to specific tasks.

Limitations

- Require more data resources

  Custom models need substantial data to avoid overfitting and generalize well.

- Higher computational resource intensive

  Training custom models from scratch requires extensive computational resources.

- Possibility of overfitting

  Custom models have a possibility to overfit, especially with complex architectures.

8.4.2 Pre-trained Model

Advantages

- Transfer learning

  Pre-trained models leverage learned features, reducing data requirements and computational resources.

- Time and resource efficiency

  Using pre-trained models saves time and resources by leveraging existing learned representations.

- Good generalization

  Pre-trained models often generalize well without overfitting to new tasks due to learned representations.

Limitations

- Limited flexibility

  Fixed architectures and learned representations might not perfectly fit new tasks or datasets.

- Domain adaptation issues

  Differences between pre-trained and target domain datasets can lead to performance issues.

- Challenging when fine-tuning

  Fine-tuning pre-trained models requires careful adjustments for effective adaptation and could be challenging.

8.5 Trade-offs

- Performance and Training time

  Custom models might offer better performance but require longer training times.

- Data Availability

  Custom models need more data compared to pre-trained models that work well with less data.

- Domain Specificity

  Pre-trained models capture general features, while custom models can be tailored to specific domain knowledge.

The choice between implementing a custom model and a pre-trained model depends on factors like data availability, computational resources, domain similarity between datasets, and specific task requirements.

9 References

[1] A. Krizhevsky, "CIFAR-10 and CIFAR-100 datasets," Toronto.edu, 2009. `https://www.cs.toronto.edu/~kriz/cifar.html`

[2] "Keras documentation: Adam," keras.io. `https://keras.io/api/optimizers/adam/`

[3] "tf.keras.losses.SparseCategoricalCrossentropy | TensorFlow Core r2.0," TensorFlow, 2019. `https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy`

[4] "Understanding VGG16: Concepts, Architecture, and Performance," Datagen. `https://datagen.tech/guides/computer-vision/vgg16/#`

[5] "ResNet-50: The Basics and a Quick Tutorial," Datagen. `https://datagen.tech/guides/computer-vision/resnet-50/`

[6] Vincent, "Analysis of Fine-Tuned vs.," Medium, Jul. 26, 2023. `https://medium.com/@schu1678/analysis-of-fine-tuned-vs-29948766094a(accessedDec.03,2023).`

[7] A. Fakhry, "The Applications and Benefits of a PreTrained Model — Kaggle's DogsVSCats," Medium, Nov. 05, 2020. `https://towardsdatascience.com/the-applications-and-benefits-of-a-pretrained-model-kaggles-dogsvscats-50221902c696`