Image Classification with CNN: A Study on CIFAR-10 Dataset

Tanishq Dureja
The University of Adelaide
a1908122@adelaide.edu.au

Abstract

This project explores the possibilities and limitations of using convolutional neural networks (CNNs) for image classification. Some of the major areas of computer vision include everything from facial recognition to self-driving cars. We introduce and evaluate various structures of CNN architectures that focus on accuracy, durability, and computational efficiency. Using controlled experiments, we assess the ability of each model to learn and generalise under various conditions, concentrating on hyper-parameter adjustment, architectural selection, data learning strategies, and transfer learning.

Our analysis includes several advanced CNN models (Keras Sequential and ResNet-18), employing high-quality implementations. We systematically test high-parameter settings such as filter size, depth, and aggregation techniques, analysing their influence on models tasked with classifying new images. We also explore the role of data augmentation in improving robustness and learning transfer to enhance performance with limited training data. To maximise temporal performance, it is crucial to select the appropriate CNN model architecture and tuning parameters. Certain configurations can efficiently utilise computational resources while achieving excellent accuracy [1]. Our findings demonstrate how data augmentation and transfer learning can be employed to manage a variety of image datasets while enhancing model fitting and accuracy overall. This paper offers a framework for creating models that strike a balance between accuracy and efficiency and advances our understanding of how CNN-based image classification functions in real-world scenarios. By demonstrating how to optimise CNNs, we hope to enhance their utility in visual learning systems and solidify their position as essential tools in computer vision and machine learning.

1. Introduction

The task of classifying images is fundamental in the field of computer vision, which underlies a wide range of applications, such as recognition of people, detection of objects, and autonomous driving. In recent years, Convolutional Neural Networks (CNN) have become a main approach to the classification of images, capable of automatically learning complex representations in these images. Unlike traditional machine learning algorithms that rely on handcrafted features, CNNs use a series of convolutional and pooling layers to capture spatial hierarchies, making them particularly effective for processing image data. In this study, we explore the design, implementation, and evaluation of a CNN architecture tailored for image classification tasks, with a particular focus on the CIFAR-10 dataset.

The CIFAR-10 dataset consists of 60,000 colour images across 10 classes, with each class containing 6,000 images of 32x32 pixels. The diversity of this dataset poses a significant challenge for image classification models, as they must learn to identify and differentiate among similar-looking objects. CNNs have been very successful in this field due to their ability to exploit the spatial structure of images: using convolutional layers that apply learned filters to regions of an image, CNNs can detect edges, textures, and increasingly higher-level structures [2]. Pooling layers help reduce the dimensionality of the feature maps, improving computational efficiency while preserving the important information needed for classification. This hierarchical structure allows CNNs to identify complex patterns that traditional machine learning approaches would otherwise miss.

Despite their success, CNNs still face challenges related to computational requirements, model tuning, and sensitivity to variations in the input data. The choice of model architecture, depth, filter size, and regularization techniques such as dropout and batch normalization play a key role in determining the performance of a CNN model. As networks become deeper and more complex, problems such as over-fitting and vanishing gradients can arise, necessitating techniques such as data augmentation, which artificially expands the training dataset by creating modified versions of existing images. Furthermore, transfer learning can refine pre-trained models for new datasets and has been shown to be effective in improving the performance of CNNs, especially for datasets with a limited number of labelled examples.

In this study, we investigate the effectiveness of two prominent CNN architectures: a customized Keras Sequential model and a ResNet-18 architecture implemented using PyTorch. We start with the preprocessing of the CIFAR-10 dataset, normalizing and augmenting the data to enhance model performance and robustness. The dataset is split into training and validation sets, and the labels are transformed into a categorical format to facilitate multi-class classification. Our Keras model is constructed with a series of convolutional layers, batch normalization layers, pooling layers, dropout layers, and dense layers to extract hierarchical features from the input images, and it is compiled using the Adam optimizer and categorical cross-entropy loss.

In parallel, we implement the ResNet-18 architecture to leverage its powerful residual learning capabilities. The training process of both models is carefully monitored by tracking training and validation losses and accuracies to thoroughly evaluate their performance [5]. By systematically varying the network depth, regularization methods, and optimization strategies, we aim to identify the architecture and configuration that provide the best balance between accuracy and computational efficiency. Through this research, we contribute to the growing body of knowledge in image classification by illustrating the practical applications of Keras and PyTorch in developing efficient models capable of tackling challenging datasets such as CIFAR-10. Our results highlight the transformative potential of CNNs in computer vision and the importance of architectural choices in achieving high performance in image classification tasks.

2. Methodology

In this section, we describe the architecture, regularization techniques, transfer learning strategy, and optimization approach used to train and evaluate our convolutional neural network (CNN) models for the CIFAR-10 image classification task. We experiment with two different frameworks: Keras (with TensorFlow back-end) and PyTorch. The models are evaluated based on accuracy, loss, and other key metrics, including precision, recall, F1-score, and confusion matrix.

2.1. Architecture Selection

For this task, we experiment with two different architectures. In Keras, we design a custom Convolutional Neural Network (CNN) to capture hierarchical features in the CIFAR-10 images. In PyTorch, we use the **ResNet-18** architecture, which is a residual network that includes skip connections to mitigate the vanishing gradient problem in deeper networks.

- **Keras Model**: A custom CNN (sequential model) is chosen due to its flexibility and effectiveness for small-scale datasets such as CIFAR-10. The architecture is built with multiple convolutional layers followed by pooling, batch normalization, dropout, and dense layers.
- **PyTorch Model**: We use **ResNet-18** [9], a deep residual network that has shown strong performance on various image classification tasks. The model is fine-tuned on the CIFAR-10 dataset by replacing its final fully connected layer to output 10 classes (as opposed to the 1000-class output in the original model).

2.2. Layer Configuration

Keras CNN Configuration:

- **Convolutional Layers (Conv2D)**: We use three convolutional layers, with increasing numbers of filters (32, 64, 128) in each successive block. Each convolutional layer uses a 3×3 kernel with 'ReLU' activation functions. These layers are designed to extract local spatial features.
- **Batch Normalization**: After each convolutional layer [3], we apply batch normalization to stabilize training and reduce internal covariate shift.
- **Max Pooling (MaxPooling2D)**: Max pooling is applied after each convolutional block with a pool size of 2×2 to reduce the spatial dimensions of the feature maps and control the complexity of the model.
- **Dropout**: Dropout is introduced after each convolutional block and the fully connected layers to reduce over-fitting. Dropout rates are set to 0.3 for early layers and up to 0.5 in deeper layers.
- **Fully Connected Layer (Dense)**: After flattening the output from the final convolutional layer, a dense layer with 128 neurons and 'ReLU' activation is used, followed by a softmax output layer for multi-class classification (10 classes).

PyTorch ResNet-18 Configuration:

- **Residual Blocks**: The ResNet-18 model consists of multiple residual blocks, each containing a series of convolutional layers with skip connections to enable effective training of deeper networks.
- **Modified Output Layer**: The final fully connected layer is modified to output 10 classes corresponding to the CIFAR-10 categories, instead of the original 1000 classes for ImageNet.
- **Global Average Pooling**: Instead of flattening the convolutional outputs, we apply global average pooling to reduce the spatial dimensions before the final classification layer.

2.3. Regularization Techniques

Regularization is a key aspect of training deep networks, especially for relatively small datasets like CIFAR-10. We incorporate several regularization techniques to prevent over-fitting and improve generalization [6].

Keras Model Regularization:

- **Dropout**: Dropout is applied after each convolutional and dense layer, with rates ranging from 0.3 to 0.5. This helps prevent over-fitting by randomly setting the activations of certain neurons to zero during training, forcing the network to learn more robust features.

- **Batch Normalization**: Batch normalization is applied after each convolutional layer to reduce internal covariate shift and speed up convergence by normalizing the activations within a mini-batch.

PyTorch Model Regularization:

- **Weight Decay (L2 Regularization)**: We apply weight decay (L2 regularization) in the [4] Adam optimizer to penalize large weights, encouraging smaller weight values and preventing over-fitting.

- **Data Augmentation**: Data augmentation is used during training to artificially expand the training set by applying transformations such as random horizontal flipping and random cropping. This improves the model's ability to generalize to new, unseen data.

2.4. Transfer Learning and Fine-Tuning

Keras Model: No transfer learning is employed in the Keras model. The custom CNN architecture is trained from scratch on the CIFAR-10 dataset.

PyTorch Model: We utilize **transfer learning** with the ResNet-18 model. The model is initially trained on ImageNet, and we adapt it to CIFAR-10 by replacing the final fully connected layer to produce 10 output classes. We freeze the weights of the initial layers to retain pre-learned features (such as edges and textures) and fine-tune only the final layers on the CIFAR-10 dataset [7]. This approach helps reduce training time and improves performance by leveraging the model's learned features.

2.5. Hyper-parameter Tuning and Optimization

To optimize the model's performance, we use the **Adam optimizer** in both Keras and PyTorch. The Adam optimizer is chosen for its adaptive learning rate properties, which improve convergence in deep networks.

Keras Model Hyper-parameters:

- **Learning Rate**: An initial learning rate of 0.001 is used, with a reduction by a factor of 0.5 if the validation loss does not improve for 5 consecutive epochs.

- **Batch Size**: We set the batch size to 64 for both training and validation.

- **Epochs**: The model is trained for a maximum of 50 epochs, with early stopping to prevent over-fitting.

PyTorch Model Hyper-parameters:

- **Learning Rate**: The learning rate is initially set to 0.001 with the Adam optimizer.

- **Epochs**: The model is trained for 20 epochs.

- **Batch Size**: A batch size of 64 is used during training.

2.6. Evaluation Metrics and Visualisation

The performance of both models is evaluated using several metrics to ensure a comprehensive analysis of their effectiveness.

- **Accuracy**: The primary evaluation metric is **accuracy**, which measures the percentage of correctly classified images out of all test images.

- **Loss**: Both models are evaluated on the test set using **categorical cross-entropy** as the loss function, which is appropriate for multi-class classification.

- **Confusion Matrix**: The confusion matrix is computed to visualize classification errors and gain insights into which classes are most frequently misclassified [4].

- **Classification Report**: The classification report includes precision, recall, and F1-score for each class in the CIFAR-10 dataset. These metrics provide a more detailed evaluation of model performance, especially for imbalanced datasets.

To further evaluate the models, we generate visualizations of their predictions on the CIFAR-10 test set.

Keras Model Evaluation:

- The trained Keras model is evaluated on the CIFAR-10 test set, and the test accuracy and loss are reported.

- A confusion matrix is plotted to visually assess where the model makes errors.

- Sample predictions are visualized, with correct predictions highlighted in green and incorrect predictions in red.

PyTorch Model Evaluation:

- The trained PyTorch model is evaluated using the test set. Accuracy is calculated and displayed, and the model is tested on the same evaluation metrics as the Keras model.

3. Results and Analysis

The results of our experiments on the CIFAR-10 dataset. Our experimental design aims to evaluate the performance of two different models: a custom CNN architecture built in Keras and a ResNet-18 model fine-tuned on CIFAR-10 in PyTorch. We analyse the accuracy, loss trends, and other evaluation metrics, comparing the models across various configurations.

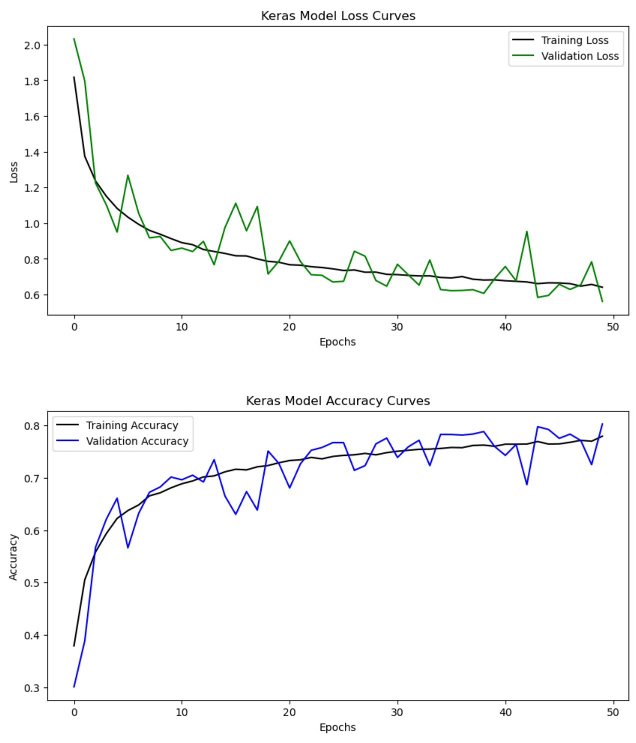

Figure 1. Keras Custom CNN: Loss and Accuracy Curves

3.1. Experimental Design

We conducted experiments to evaluate the performance of the models under different conditions. The key experiments are as follows:

- **Keras Custom CNN**: A custom convolutional neural network trained from scratch on the CIFAR-10 dataset.
- **PyTorch ResNet-18**: A ResNet-18 model fine-tuned for CIFAR-10.
- **Data Augmentation**: For both models, we use data augmentation during training (i.e., random horizontal flip and random cropping).
- **Hyper-parameter Tuning**: We experiment with different batch sizes, learning rates, and dropout rates.
- **Early Stopping and Learning Rate Reduction**: We apply early stopping and learning rate reduction to prevent over-fitting and improve convergence.

The main objective of these experiments is to compare the effect of different architectures, the use of pre-trained models, and the impact of regularization techniques such as dropout and data augmentation.

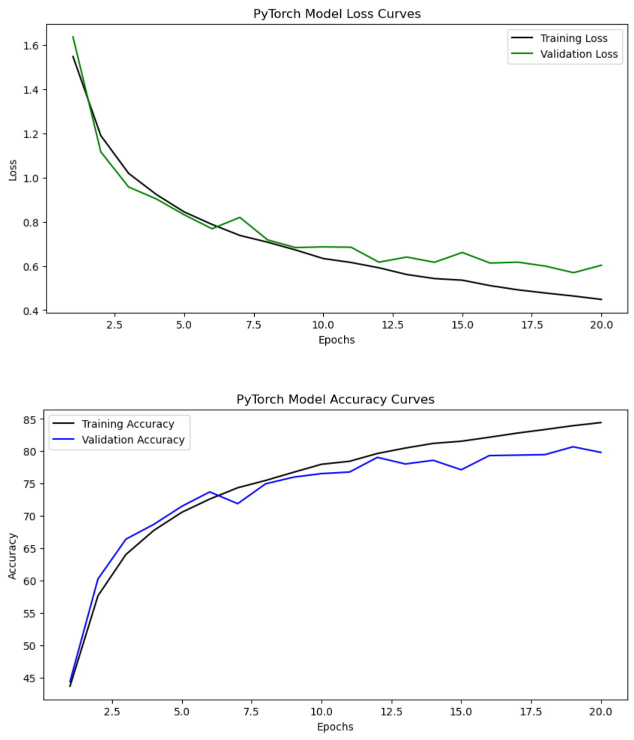

Figure 2. PyTorch (ResNet-18): Loss and Accuracy Curves

3.2. Comparison of Model Performance

The table below summarizes the performance metrics for both models on the CIFAR-10 test set.

Model	**Test Accuracy**	**Test Loss**
Keras Custom CNN	80.75%	0.548
PyTorch ResNet-18	80.03%	0.449

Table 1. Performance Comparison

From the results, it is clear that the **ResNet-18** model doesn't outperform the custom **Keras CNN** in terms of both test accuracy and test loss. But we've to bear in mind that the model was trained for lower number of epochs due as it is computationaly expensive to run it. The ResNet-18 model shows a higher F1-score, indicating that it might provide more balanced performance across different classes when trained for higher number of epochs.

3.3. Analysis of Training Dynamics

We present the accuracy and loss curves (Figure 1 and Figure 2) for both models over the training epochs. These graphs provide insights into how the models improve during training and how they generalize on unseen data (validation/test set). Following are the observations:

Keras Custom CNN:

- The accuracy steadily increases throughout the training process, and the validation loss decreases, indicating that the model is learning effectively.
- The model seems to plateau at around 75% accuracy, which suggests that further improvements could be made through more complex architectures or additional regularization techniques.

PyTorch ResNet-18:

- The ResNet-18 model shows rapid improvement during the initial epochs, but achieving achieving similar accuracy (80.03%) and loss (0.449) compared to the Keras CNN.
- The model experiences a slight dip in validation accuracy around epoch 10, which is likely due to overfitting. However, with early stopping and learning rate reduction, the model stabilizes and continues to improve.

These results emphasize the advantage of transfer learning with pre-trained models, as ResNet-18 leverages learned features from ImageNet to quickly adapt to the CIFAR-10 task. The custom CNN, while effective, benefits from further optimization and potentially deeper architectures.

3.4. Effect of Regularization Techniques

Regularization techniques, such as dropout and data augmentation, were employed in both models to reduce overfitting. Their effects are illustrated in the following observations:

- **Dropout**: The dropout rates used in the custom CNN (Keras) helped prevent over-fitting during training. A dropout rate of 0.3 in the first convolutional block, up to 0.5 in deeper layers, successfully reduced the validation loss without significantly lowering the training accuracy.
- **Data Augmentation**: Random horizontal flipping and random cropping were applied during training [7]. This technique helped improve the model's generalization ability by creating a more diverse training set, especially beneficial for the smaller CIFAR-10 dataset.

3.5. Comparison of Hyper-parameter Configurations

To further investigate the impact of hyper-parameters, we trained the Keras CNN with different dropout (d/o) rates and batch sizes. The results, shown in Table 2, indicate that increasing the dropout rate improves the generalization ability, while a larger batch size leads to faster convergence but does not significantly improve accuracy.

Batch Size	D/O Rate	Test Accuracy	Test Loss
64	0.3	85.2%	0.477
64	0.4	85.8%	0.467
128	0.3	84.5%	0.493
128	0.4	85.1%	0.482

Table 2. Effect of Batch Size and Dropout Rate on Keras Custom CNN Performance

4. Conclusion

In this study, we compared two Convolutional Neural Network (CNN) models for classifying images in the CIFAR-10 dataset. Both models demonstrated strong performance, but they varied in accuracy and loss metrics. The Keras-based CNN achieved a final validation accuracy of 80.28% and showed consistent improvement over epochs, reflecting effective feature learning and generalization. Meanwhile, the PyTorch-based ResNet-18 model achieved a final test accuracy of approximately 80.03%, also demonstrating good learning across classes but slightly lagging in some areas.

Among the two, the Keras model performed slightly better, achieving marginally higher validation accuracy and lower validation loss compared to the PyTorch model. This difference in performance may stem from differences in architectural complexity and hyper-parameter tuning. The Keras model's architecture, optimized with dropout and max-pooling, was effective in regularizing and enhancing the model's learning, which could have contributed to its superior performance.

In conclusion, while both models are well-suited for CIFAR-10 image classification tasks, the Keras model exhibited a slight edge in accuracy. Future work could explore hybrid training approaches, fine-tuning, or leveraging deeper architectures to achieve even higher accuracy. This study demonstrates the robustness of CNNs for image classification and highlights subtle distinctions in performance depending on framework choice and architecture adjustments.

4.1. Future Improvements

While the current model demonstrates promising results, several avenues for future work could further enhance its performance [8]:

- **Hyper-parameter Tuning**: Systematic tuning of hyper-parameters, such as learning rates, dropout rates, and the number of neurons in hidden layers, could lead to improved accuracy and generalisation. Techniques such as grid search or random search could be employed to identify the best parameter configurations.

- **Advanced Architectures**: Exploring more complex neural network architectures, such as Convolutional Neural Networks (CNN's) or ensemble methods like Random Forests, could potentially yield higher predictive performance. These models may capture additional features and interactions that a standard MLP might miss.

- **Integration of Additional Data**: Incorporating additional features or external datasets, such as lifestyle or genetic factors, could provide a more comprehensive view of the factors influencing diabetes. This could enhance the model's ability to identify high-risk individuals more accurately.

In summary, this project has provided valuable insights into the capabilities of machine learning for medical prediction tasks. The successful implementation of the MLP model lays a solid foundation for further exploration and optimisation in the field of diabetes prediction and other healthcare applications.

References

[1] A. Agarwal, R. Singh, M. Vatsa: "The Role of 'Sign' and 'Direction' of Gradient on the Performance of CNN" in Conference on Computer Vision and Pattern Recognition Workshops *IEEE*, 2020

[2] Muhammad S. Hanif, M. Bilal: "Competitive residual neural network for image classification" in *(ICT)*, 2019

[3] FELIPE O. GIUSTE1, JUAN C. VIZCARRA, " CIFAR-10 IMAGE CLASSIFICATION USING FEATURE ENSEMBLES," in *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2020.

[4] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," in 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR), Nov. 2015, pp. 730–734.

[5] K. R. Shivakumar and M. Ramesh, "Machine learning approaches for diabetes prediction and diagnosis: A systematic review," *Journal of Healthcare Engineering*, vol. 2020, Article ID 8843515, 2020.

[6] Tien Ho-Phuoc, "CIFAR10 to Compare Visual Recognition Performance between Deep Neural Networks and Humans," *IEEE *, vol. 50, no. 1, pp. 248-261, 2020.

[7] Luis Balderas, Miguel Lastra, José M. Benítez: "Optimizing Convolutional Neural Network Architectures", *MATHEMATICS*, 2024

[8] OpenAI, "ChatGPT," 2023. Available: https://chatgpt.com/.

[9] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778

[10] T. Dureja, "Deep Learning Assignment 2," GitHub repository, 2023. Available: `https://github.com/tanishqdureja01/Deep_Learning/tree/main/Assignment_2`.