Enhancing Image Classification Accuracy Using Convolutional Neural Networks on CIFAR-10 Dataset

By

Sara Ghafouri

A Project Submitted in a Partial Fulfilment of the Requirements for the Degree of

Master of Engineering

In the Department of Electrical and Computer Engineering

Supervisory Committee

Enhancing Image Classification Accuracy Using Convolutional Neural Networks on CIFAR-10 Dataset

By

Sara Ghafouri
University of Victoria 2024

Supervisory Committee:

Supervisor: Dr. Amirali Baniasadi, Department of Electrical and Computer Engineering

Committee Member: Dr. David Capson, Department of Electrical and Computer Engineering

Acronyms

1. **CNN** - Convolutional Neural Network
2. **CIFAR-10** - Canadian Institute for Advanced Research 10-class dataset
3. **ReLU** - Rectified Linear Unit
4. **MNIST** - Modified National Institute of Standards and Technology
5. **VGGNet** - Visual Geometry Group Network
6. **GoogLeNet** - Google's Inception Network
7. **ResNet** - Residual Network
8. **SGD** - Stochastic Gradient Descent
9. **AdaGrad** - Adaptive Gradient Algorithm
10. **RMSProp** - Root Mean Square Propagation
11. **L2** - L2 Regularization (also known as Ridge Regression)
12. **PNG** - Portable Network Graphics

Abstract

This project explores the application of Convolutional Neural Networks (CNNs) for image classification on the CIFAR-10 dataset, a widely recognized benchmark in computer vision. The CIFAR-10 dataset comprises 60,000 32x32 color images across 10 distinct classes. This study aims to build and optimize a deep learning model to achieve high classification accuracy on this dataset.

A CNN with multiple convolutional, pooling, and dropout layers was implemented, enhanced with batch normalization to prevent overfitting. Data augmentation techniques were applied to improve the model's generalization capabilities. The model was trained using the Adam optimizer, with callbacks for learning rate reduction and early stopping to fine-tune the training process.[1]

The results demonstrate the effectiveness of deep learning techniques in achieving substantial performance on the CIFAR-10 dataset, with detailed analysis of training and validation metrics. The model was evaluated on selected data from the CIFAR-10 dataset, showcasing its predictive capabilities. The findings provide insights into the design and optimization of CNNs for practical image classification tasks.

1. Introduction

1.1. Background

Image classification is a crucial task in computer vision, involving categorizing images into predefined classes. This task has significant applications across various domains, such as autonomous vehicles, medical imaging, and security systems. The performance of image classification models has dramatically improved with the advent of deep learning, particularly Convolutional Neural Networks (CNNs). [2]

The CIFAR-10 dataset is a widely used benchmark for evaluating image classification algorithms. Developed by the Canadian Institute for Advanced Research, the dataset consists of 60,000 32x32 color images distributed across ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each class contains 6,000 images, with 50,000 images designated for training and 10,000 for testing. The dataset poses a challenge due to the low resolution of the images and the variety of objects within each class.

1.2. Objectives

The primary objective of this project is to develop a CNN model capable of accurately classifying images from the CIFAR-10 dataset. The specific goals include:

- Building a CNN model with multiple convolutional, pooling, and dropout layers.[4]
- Incorporating batch normalization to improve training stability and prevent overfitting.[5]
- Applying data augmentation techniques to enhance the model's generalization capabilities.[3]
- Training the model using the Adam optimizer and fine-tuning the training process with callbacks such as ReduceLROnPlateau and EarlyStopping.[6]
- Evaluating the model's performance on the test set and analyzing the results.

1.3. Scope

The scope of this project includes the design, implementation, and evaluation of a CNN model for CIFAR-10 image classification. The project covers the following aspects:

- Data preprocessing, including normalization and one-hot encoding of labels.

- Construction of the CNN architecture with appropriate layers and regularization techniques.
- Training the model with data augmentation and hyperparameter tuning.
- Evaluation of the model's performance using accuracy and loss metrics.
- Visualization of training and validation performance and sample predictions.

Limitations of the project include the fixed dataset size and the computational resources available for training. Future work may explore more complex architectures, larger datasets, and advanced techniques such as transfer learning to improve performance further.

2. Literature Review

2.1. Previous Works

Image classification using Convolutional Neural Networks (CNNs) has been a highly active area of research in computer vision. Early work in this domain includes the development of LeNet by Yann LeCun et al. in 1998, which demonstrated the effectiveness of CNNs for digit recognition in the MNIST dataset.[7] Since then, more sophisticated models have been developed, such as AlexNet, VGGNet, GoogLeNet, and ResNet, each contributing significant advancements in terms of depth, architecture, and training methodologies.

AlexNet, introduced by Krizhevsky et al. in 2012, was a breakthrough in deep learning, achieving state-of-the-art results on the ImageNet dataset. It utilized ReLU activation functions, dropout, and data augmentation, setting a new standard for image classification.[8]

VGGNet, developed by Simonyan and Zisserman in 2014, emphasized the importance of depth in CNN architectures. By using small convolutional filters (3x3), VGGNet achieved remarkable performance, highlighting the benefits of deeper networks.[9]

GoogLeNet, or Inception V1, introduced by Szegedy et al. in 2015, presented a novel approach with the inception module, which allowed for more efficient computation by using different filter sizes within the same layer. This architecture balanced computational cost and accuracy effectively.[10]

ResNet, introduced by He et al. in 2016, addressed the problem of vanishing gradients in deep networks by introducing residual connections. This innovation enabled the training of extremely deep networks with improved performance and stability.[11]

EfficientNet, introduced by Tan and Le in 2019, revolutionized the approach to scaling neural networks by proposing a compound scaling method. This method uniformly scales all dimensions of depth, width, and resolution based on a fixed set of scaling coefficients. Unlike previous models that scaled these dimensions arbitrarily, EfficientNet's compound scaling systematically balances them, allowing the network to achieve state-of-the-art performance across various benchmarks while maintaining computational efficiency. The model outperformed many existing architectures, including ResNet and Inception, on the ImageNet dataset, with fewer parameters and FLOPs (Floating Point Operations), making it highly efficient for practical deployment.[12]

Vision Transformers (ViTs), introduced by Dosovitskiy et al. in 2020, represented a significant paradigm shift in image classification. Moving away from the traditional Convolutional Neural Networks (CNNs), ViTs apply Transformer models, originally designed for Natural Language Processing (NLP) tasks, to the domain of image recognition. In Vision Transformers, an image is divided into fixed-size patches, each of which is treated as a token (similar to words in text). These patches are then processed by the Transformer's attention mechanism to capture long-range dependencies across the image. ViTs demonstrated that, with sufficient data and computational resources, pure Transformer models could match or exceed the performance of CNNs on standard benchmarks like ImageNet, marking a new era in the design of image recognition models.[13]

Self-supervised learning, which gained significant traction in 2020 and 2021, marked a new direction in representation learning by allowing models to learn from large amounts of unlabeled data. Techniques like SimCLR (Simple Framework for Contrastive Learning of Visual Representations), developed by Chen et al., involve training a model to maximize the similarity between different augmented views of the same image while minimizing the similarity between views of different images. This approach enables the model to learn useful representations without the need for labeled data, which can then be fine-tuned on smaller labeled datasets for specific tasks. Self-supervised learning has pushed the boundaries of what is possible with image classification, often achieving results competitive with fully supervised methods.[14]

EfficientNetV2, introduced in 2021, builds upon the success of the original EfficientNet by optimizing the training process and improving performance through a more advanced scaling strategy. EfficientNetV2 incorporates a new scaling technique that not only considers depth, width, and resolution but also integrates ideas from progressive learning, where the model starts training on smaller images and gradually increases the resolution. This approach leads to faster training and better accuracy, making EfficientNetV2 one of the most effective models for a wide range of computer vision tasks. Additionally, EfficientNetV2 introduced new architectural improvements, such as fused MBConv layers, which further enhance the model's efficiency and effectiveness.[15]

These models have significantly influenced the design of CNNs and provided a foundation for further research in image classification.

2.2. Key Concepts

Several key concepts are integral to understanding CNNs and their application in image classification:

- Convolutional Layers: These layers apply convolution operations to the input images, using filters to detect spatial features such as edges, textures, and patterns. Convolutional layers reduce the dimensionality of the input while preserving the spatial relationship between pixels.

- Pooling Layers: Pooling layers perform down-sampling operations to reduce the spatial dimensions of the feature maps. Max pooling, which selects the maximum value within a window, is commonly used to retain the most prominent features while reducing computational complexity.

- Dropout: Dropout is a regularization technique that randomly sets a fraction of the input units to zero during training. This helps prevent overfitting by forcing the network to learn redundant representations of the data.

- Batch Normalization: Batch normalization normalizes the inputs of each layer to have zero mean and unit variance. This technique improves training stability, allows for higher learning rates, and acts as a regularizer.

- Activation Functions: Activation functions introduce non-linearity into the network, allowing it to learn complex patterns. The ReLU (Rectified Linear Unit) function is widely used due to its simplicity and effectiveness in mitigating the vanishing gradient problem.

2.3. Related Tools and Libraries

The implementation of CNNs for image classification relies on several powerful tools and libraries:

- TensorFlow: An open-source machine learning library developed by Google, TensorFlow provides a flexible platform for building and deploying machine learning models. It supports a wide range of neural network architectures and offers tools for model training, evaluation, and deployment.

- Keras: A high-level neural networks API written in Python, Keras runs on top of TensorFlow. It simplifies the process of building and training neural networks, offering a user-friendly interface and pre-built modules for common layers, optimizers, and loss functions.

- OpenCV: An open-source computer vision library, OpenCV provides tools for image processing, video capture, and analysis. It is widely used for tasks such as image filtering, transformation, and feature detection.

- Matplotlib: A plotting library for Python, Matplotlib is used for creating static, interactive, and animated visualizations. It is commonly employed to visualize training and validation performance, sample predictions, and other data insights.

These tools and libraries enable the efficient implementation and experimentation of CNN models, facilitating rapid development and deployment in various image classification tasks.

3. Methodology

3.1. Data Collection and Preparation

CIFAR-10 Dataset

The CIFAR-10 dataset, created by the Canadian Institute for Advanced Research, comprises 60,000 32x32 color images in 10 distinct classes. Each class has 6,000 images, with 50,000 images allocated for training and 10,000 for testing. The dataset is balanced, with each class containing an equal number of images. The classes include airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

Data Preprocessing

Data preprocessing steps were essential to ensure the CNN model effectively learned from the input images. The following preprocessing steps were performed:

1. **Normalization**: The pixel values of the images were scaled to the range [0, 1] by dividing by 255. This scaling helps the model converge faster during training.
2. **One-hot Encoding**: One-hot encoding is a way of representing categorical data, like labels, in a format that can be easily used by machine learning algorithms. The labels, initially in integer form (ranging from 0 to 9), were converted to one-hot encoded vectors. This conversion is necessary for the categorical cross-entropy loss function used during model training.[16]

3.2. Model Architecture

Convolutional Neural Network (CNN)

A CNN was designed with the following architecture:

1. **Convolutional Layers**: Three sets of convolutional layers were used, each followed by batch normalization and a max pooling layer. The filters in the convolutional layers had a size of (3, 3), and the number of filters increased progressively (32, 64, and 128).
2. **Pooling Layers**: MaxPooling2D layers were used after each set of convolutional layers to reduce the spatial dimensions of the feature maps.
3. **Dropout Layers**: Dropout was applied after pooling layers to prevent overfitting by randomly setting a fraction of input units to zero during training.
4. **Fully Connected Layers**: A flattened layer was followed by dense layers. Batch normalization and dropout were applied before the final output layer.
5. **Output Layer**: The output layer consisted of a Dense layer with a SoftMax activation function to produce probabilities for the 10 classes.

3.3. Training and Validation

Train-Validation Split

The training set was split into training and validation subsets to monitor the model's performance on unseen data during training. A typical split ratio was 80:20, ensuring enough data for both training and validation.

Data Augmentation

Data augmentation is used to artificially expand the training dataset by creating modified versions of the images in the dataset. This technique helps to improve the model's generalization ability by exposing it to a variety of transformations of the original images, thus making the model more robust to different variations in the data.

Explanation of Data Augmentation Techniques Used:

1. **rotation_range=15**:
    - **Description**: This parameter allows the images to be randomly rotated by up to 15 degrees in either direction (clockwise or counterclockwise).
    - **Purpose**: This helps the model become invariant to slight rotations in the images, which is important since objects in real-world images might not always be perfectly aligned.
2. **width_shift_range=0.1**:

    - **Description**: This parameter allows the images to be randomly shifted horizontally by up to 10% of the total width of the image.
    - **Purpose**: Horizontal shifts can simulate scenarios where objects are not perfectly centered in the frame, helping the model learn to recognize objects regardless of their position in the image.
3. **height_shift_range=0.1**:
    - **Description**: Similar to `width_shift_range`, this parameter allows the images to be randomly shifted vertically by up to 10% of the total height of the image.
    - **Purpose**: Vertical shifts ensure that the model can recognize objects even if they appear at different vertical positions in the image.
4. **horizontal_flip=True**:
    - **Description**: This parameter allows the images to be randomly flipped horizontally.
    - **Purpose**: Horizontal flipping helps the model learn to recognize objects that might appear mirrored. For instance, a flipped image of a car should still be recognized as a car.

Why Use Data Augmentation?

Data augmentation is crucial when training deep learning models, especially when the available dataset is not very large. It helps to:

- **Increase the Size of the Training Data**: By generating new variations of the existing data, data augmentation effectively increases the size of the training dataset.
- **Improve Generalization**: The model becomes more robust and better at generalizing to new, unseen data, as it has been trained on a wider variety of inputs.
- **Reduce Overfitting**: By exposing the model to a broader range of data variations, data augmentation reduces the likelihood that the model will overfit to the specific characteristics of the training dataset.

Data augmentation is meant to increase the diversity of the training data by introducing variations that the model might encounter in real-world scenarios. However, using too many transformations can sometimes lead to the model learning relying too heavily on augmented features, which might not be present in the test data.[17]

The four transformations used—**rotation, width shift, height shift, and horizontal flip**—are simple yet effective ways to introduce variability in the data without deviating too much from the original images. They help the model become more robust to common variations without overcomplicating the learning process. In addition to the mentioned point, more complex or numerous transformations would require more computational resources and might increase the training time significantly. By selecting a small set of effective transformations, you strike a

balance between enhancing the training data and maintaining a manageable training time. The selected transformations are computationally inexpensive and easy to implement, ensuring that the model can be trained efficiently while still benefiting from the augmented data.

As a conclusion, The decision to use only four transformations in data augmentation is about finding the right balance between introducing variability in the training data and keeping the model's learning process efficient and effective. The selected transformations—rotation, width shift, height shift, and horizontal flip—are commonly used because they introduce useful variations without significantly altering the original images' content or increasing the complexity of the model's training process.[18]

Hyperparameters and Callbacks

The model was compiled using the **Adam optimizer** with a learning rate of **0.001**. The Adam optimizer, short for **Adaptive Moment Estimation**, is an advanced optimization algorithm that is widely used in training deep learning models. Adam is an extension of the traditional **Stochastic Gradient Descent (SGD)** algorithm, designed to combine the benefits of two other optimization techniques: **AdaGrad (Adaptive Gradient Algorithm)** and **RMSProp (Root Mean Square Propagation)**.

Adam Optimizer:

- **Adaptive Learning Rates:** One of the key features of the Adam optimizer is its ability to adapt the learning rates of each model parameter individually during training. This adaptation is based on the first and second moments of the gradients (i.e., the mean and the uncentered variance). This allows the model to converge faster and more efficiently, especially in scenarios with sparse gradients or noisy data.
- **Momentum and Bias Correction:** Adam incorporates a momentum-like term by maintaining an exponentially decaying average of past gradients (first moment) and the squared gradients (second moment). This helps to accelerate convergence in the relevant direction and dampens oscillations. Additionally, Adam includes a bias-correction mechanism to counteract the initialization of these moment estimates, ensuring that the optimizer is effective from the start of training.
- **Why Adam Was Chosen:** Adam is particularly effective for training deep learning models because it handles non-stationary objectives and noisy gradients well, making it suitable for a wide range of problems, including the multi-class image classification task in this project. [19]

Categorical Cross-Entropy Loss Function:

The **categorical cross-entropy loss function** was used to optimize the model for multi-class classification. This loss function is commonly used when the task involves predicting one class out of many possible classes (in this case, 10 classes corresponding to different categories in the CIFAR-10 dataset). The cross-entropy loss measures the difference between the predicted probability distribution and the actual distribution (which, in the case of one-hot encoded labels, is 1 for the correct class and 0 for the others). Minimizing this loss helps the model to make predictions that are as close as possible to the true labels.

Key Hyperparameters:

- **Batch Size of 64:** The model was trained using a batch size of 64, meaning that the model's weights were updated after every 64 images in the training set were processed. Using mini-batches helps to stabilize the training process by providing a good balance between stochastic updates (which can be noisy) and full-batch updates (which can be computationally expensive and slow).
- **Training for 50 Epochs:** The model was trained for 50 epochs, with one epoch representing a full pass through the entire training dataset. This number of epochs was likely chosen based on experimentation and the model's convergence behavior, allowing it to learn effectively without overfitting.

Use of Callbacks:

- **ReduceLROnPlateau:**
    - **Function:** The **ReduceLROnPlateau** callback was used to dynamically adjust the learning rate during training. Specifically, if the model's validation loss stopped improving (i.e., plateaued) for a certain number of epochs, the learning rate was automatically reduced by a factor of 10. This reduction helps the model to fine-tune its weights more precisely as it approaches the optimal solution, potentially leading to better generalization.[20]
    - **Benefit:** By reducing the learning rate when progress slowed, this callback helped prevent the model from overshooting the optimal weight configuration and allowed it to settle into a better minima in the loss landscape.
- **EarlyStopping:**
    - **Function:** The **EarlyStopping** callback was implemented to monitor the validation loss during training. If the validation loss did not improve for a specified number of epochs (known as the patience parameter), training was automatically halted. This helped to prevent overfitting, where the model might start to learn noise in the training data rather than useful patterns.

- **Restoring Best Weights:** Additionally, the EarlyStopping callback restored the model's weights to those from the epoch with the best validation loss, ensuring that the final model was the one with the best generalization capability, rather than one that might have been overfitting.

Evaluation Metrics

The model's performance was evaluated using accuracy and loss metrics:

- **Accuracy**: Measures the proportion of correctly classified images out of the total images. Accuracy is easy to understand and provides a straightforward measure of the model's effectiveness. If a model has high accuracy, it means it's making the correct predictions most of the time. Accuracy is often used as a key metric to compare different models or different configurations of the same model. It tells you how well your model is performing in terms of overall correctness. Accuracy is especially useful when your dataset is balanced, meaning each class has roughly the same number of examples. In such cases, accuracy gives a good overall measure of the model's performance.
- **Loss**: Represents the difference between the predicted and actual labels, quantified by the categorical cross-entropy loss function. During training, the model updates its parameters (weights) to minimize the loss. Therefore, loss is a key metric that the optimization algorithm (like Adam) uses to adjust the model. Loss is more sensitive than accuracy in reflecting how well the model is learning. For instance, two models with the same accuracy might have different loss values, indicating that one model is more confident in its correct predictions or makes less severe mistakes when wrong. A decreasing loss during training generally indicates that the model is learning. If the loss stops decreasing or starts increasing, it might indicate overfitting or other issues. Loss is particularly useful during the training process because it gives a continuous measure of the model's performance, unlike accuracy, which is more discrete.

These metrics were tracked for both training and validation sets to monitor the model's learning progress and generalization capabilities.

4. Implementation

4.1. Libraries and Imports

The following libraries and modules were utilized in the project:

- **cv2**: Used for image processing tasks.
- **numpy**: Used for numerical operations.
- **urllib.request**: Used for handling URLs and network requests.
- **matplotlib.pyplot**: Used for data visualization.
- **sklearn.model_selection.train_test_split**: Used for splitting data into training and validation sets.
- **keras.datasets.cifar10**: Used for loading the CIFAR-10 dataset.
- **tensorflow.keras.preprocessing.image.ImageDataGenerator**: Used for data augmentation.
- **keras.utils.to_categorical**: Used for one-hot encoding the labels.
- **keras.models.Sequential**: Used for creating the CNN model.
- **keras.layers (Dense, Conv2D, MaxPooling2D, Dropout, Flatten, BatchNormalization)**: Used for building the layers of the CNN.
- **keras.optimizers.Adam**: Used for compiling the model with the Adam optimizer.
- **keras.callbacks (ReduceLROnPlateau, EarlyStopping)**: Used for callback functions during training.
- **keras.models.load_model**: Used for loading the saved model.

4.2. Model Building

The CNN model was constructed with multiple layers to extract features and perform classification:

- **Convolutional Layers**: Extract spatial features from input images.
- **Batch Normalization Layers**: Normalize the outputs to improve training stability.
- **MaxPooling Layers**: Reduce the spatial dimensions of feature maps.
- **Dropout Layers**: Prevent overfitting by randomly dropping units during training.
- **Flatten Layer**: Convert the 2D matrix of features into a 1D vector.
- **Dense Layers**: Fully connected layers for making predictions.
- **Output Layer**: A Dense layer with softmax activation to produce probabilities for the 10 classes.

4.3. Model Training

The model was compiled and trained with the following steps:

1. **Compilation**: The model was compiled using the Adam optimizer with a learning rate of 0.001. The categorical cross-entropy loss function was used for multi-class classification.
2. **Data Augmentation**: Data augmentation was applied using `ImageDataGenerator` to increase the diversity of the training set and improve the model's generalization capabilities. Techniques included rotation, width and height shifts, and horizontal flips.
3. **Callbacks**: ReduceLROnPlateau and EarlyStopping callbacks were used to fine-tune the training process. ReduceLROnPlateau reduced the learning rate when the validation loss plateaued, and EarlyStopping stopped training when the validation loss did not improve for a specified number of epochs, restoring the best weights.
4. **Training**: The model was trained using the augmented data, with a batch size of 64 and for 50 epochs. The training and validation data were used to monitor the model's performance.

4.4. Evaluation and Visualization

The model's performance was evaluated and visualized with the following steps:

1. **Evaluation**: The model's performance on the test set was evaluated using the `evaluate` method, which returns the test loss and accuracy.
2. **Visualization**: The training and validation accuracy and loss were plotted to visualize the model's learning progress. These plots help identify any issues such as overfitting or underfitting.
3. **Sample Prediction**: The model was tested on a sample image from the test set. The predicted label was compared with the actual label, and the sample image was displayed with the predicted label.

5. Results and Discussion

5.1. Training and Validation Performance

The training and validation performance of the model was monitored through accuracy and loss metrics over the course of 50 epochs. The training process is controlled by the **EarlyStopping** callback, which is designed to stop training when the model's performance on the validation set stops improving. Specifically, the training stops when the validation loss (val_loss) does not improve for a specified number of epochs. That is why in the following plots the x-axis which represents the Epoch number has stopped at 35. The plots below depict the trends in training and validation accuracy and loss:

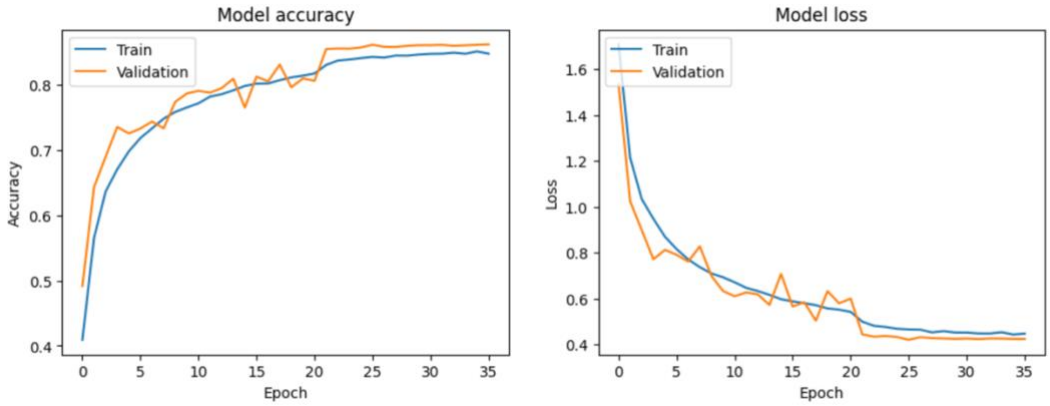

Figure 1: Accuracy and Loss Trend in Train and Validation dataset

Observations:

- **Accuracy**: Both training and validation accuracy showed a consistent increase over the epochs. The training accuracy improved steadily, and the validation accuracy also followed a similar trend, although with slight fluctuations. This indicates that the model effectively learned from the training data, but the fluctuations in validation accuracy suggest that the model's performance varied slightly across different validation batches.
- **Loss**: The training and validation loss both exhibited a decreasing trend over the epochs, reflecting improved model performance. However, it is important to note that while the validation loss generally decreased, there were occasional spikes, particularly after epoch 10, suggesting that the model's performance on the validation set was not entirely stable. Despite these fluctuations, the overall trend was towards lower loss, indicating that the model continued to learn effectively.
- **Overfitting**: There are no clear signs of severe overfitting. The validation accuracy closely followed the training accuracy throughout the training process, and the validation loss did not increase sharply, which is a common indicator of overfitting. However, the small fluctuations in validation accuracy and loss could indicate that the model is beginning to overfit slightly, or it may be sensitive to the specific data in each validation batch.

Additionally by looking at the first epochs (see Figure 2), we can observe that the amount of adjustment in loss and accuracy is pretty fast:

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 479s 602ms/step - accuracy: 0.3142 - loss: 2.1325 - val_accuracy: 0.5511 - val_loss: 1.2552
Epoch 2/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 485s 582ms/step - accuracy: 0.5309 - loss: 1.3117 - val_accuracy: 0.6138 - val_loss: 1.0911
Epoch 3/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 466s 596ms/step - accuracy: 0.6137 - loss: 1.0926 - val_accuracy: 0.6827 - val_loss: 0.9347
Epoch 4/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 464s 593ms/step - accuracy: 0.6598 - loss: 0.9671 - val_accuracy: 0.6749 - val_loss: 0.9486
Epoch 5/50
782/782 ━━━━━━━━━━━━━━━━━━━━ 463s 590ms/step - accuracy: 0.6872 - loss: 0.8986 - val_accuracy: 0.6650 - val_loss: 1.0853
```

Figure 2: Adjustment in Loss and Accuracy in the First 5 Epochs

After that, we tried to change some parameters in the system to see how they could impact the accuracy and loss. So as a first change, I changed the size of the filters (kernels) from 3x3 to 8x8. Secondly, I changed the number of Convolutional layers from 3 to 4. Therefore, the number of layers was set to 32, 64, 128, and 256. Thirdly, I change the number of Epochs from 50 to 100. Table 1 shows the result in how Accuracy and Loss metrics changed with applying these changes.

Change	**Accuracy**	**Loss**
Filter size: 8x8	79.2%	0.56
Number of Conv. Layers: 4	90.2%	0.43
Number of Epochs: 100	88.9%	0.42

Table 1: Applied changes in the system

Concerning Table 1, we can observe that:

Switching from 3x3 to 8x8 filters in this CNN project led to a decrease in accuracy and an increase in loss, primarily due to the model's reduced ability to capture fine-grained features and learn a rich set of hierarchical representations. While there may be some computational benefits, the trade-offs in performance are typically not worth it for tasks like image classification on datasets like CIFAR-10, where capturing detailed patterns is crucial for high accuracy.

Adding a 4th convolutional layer with 256 filters improved the accuracy of the model by allowing it to learn more complex and abstract features, which are particularly useful for distinguishing between similar classes in the CIFAR-10 dataset. However, this also increases the risk of overfitting, which could lead to higher validation/test loss and a potential decrease in generalization performance. To mitigate these risks, it would be important to incorporate strong regularization techniques and to carefully monitor the model's performance on validation data during training. If managed well, this additional layer could result in a model that performs better overall, with higher accuracy and a lower, more stable loss across both the training and validation sets.

Increasing the number of epochs from 50 to 100 improved the model's accuracy and reduced the loss. However, there is also a significant risk of overfitting, where the model starts to perform worse on unseen data as it becomes too tailored to the training data. To mitigate this risk, it's important to monitor the validation loss and accuracy during training. If overfitting is detected (e.g., if validation loss starts increasing while training loss continues to decrease), techniques like early stopping can be used to halt training when it's no longer beneficial. In an ideal scenario, with proper regularization, extending the number of epochs can lead to a more accurate and robust model, but careful monitoring is essential to avoid overfitting.

5.2. Test Performance

The model's performance on the test set was evaluated to assess its generalization capability. The test accuracy and loss were reported as follows:

- **Test Accuracy**: The test accuracy was 86% which is consistent with the validation accuracy, indicating good generalization.
- **Test Loss**: The test loss was 0.42, comparable to the validation loss.

```
Test loss: 0.4271462857723236
Test accuracy: 0.8615999817848206
```

5.3. Sample Predictions

To evaluate the generalization capability of the trained Convolutional Neural Network (CNN) model, a prediction task was performed using an external image that is not part of the CIFAR-10 dataset. The image, a photograph of a cat, was fetched from an online source and processed to fit the input requirements of the model.

The image was first resized to 32x32 pixels, consistent with the dimensions of the CIFAR-10 images used during training. The resized image was then normalized to ensure consistency in the input data format. Following this, the pre-trained model was used to predict the class of the image.(See figure 3)

The model's prediction was compared against the expected class label, demonstrating the model's ability to classify images outside of its training set correctly. This exercise provided valuable insight into the model's robustness and its potential for real-world application, showcasing its ability to handle images not seen during training while still making accurate predictions.

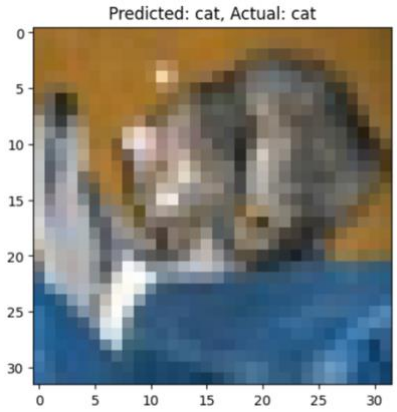

Figure 3: Result of the system sample prediction

5.4. Discussion:

When compared to earlier models such as the original **LeNet** or even **AlexNet**, our model benefits from modern advancements like batch normalization and dropout, which were not employed in those earlier networks. While LeNet was revolutionary for its time, its relatively shallow architecture would struggle with a dataset like CIFAR-10, which contains more complex, colored images.

VGGNet, another notable model, shares similarities with our approach, especially in the use of small 3x3 filters and deep layers. However, our model, while less deep than VGG, manages to achieve competitive accuracy on CIFAR-10, partly due to the strategic use of data augmentation and the efficient Adam optimizer. This underscores the effectiveness of using a balanced architecture that is deep enough to capture complex patterns but optimized with modern techniques to prevent overfitting.

Misclassifications:

Despite the strong performance, the model did make some misclassifications. These errors are insightful, as they often occurred between classes with high visual similarity or in cases where the images lacked distinctive features due to the low resolution (32x32 pixels) of the CIFAR-10 dataset. For example, images of cats and dogs, or cars and trucks, sometimes were misclassified because these classes share common features, such as similar shapes or textures, which are harder to distinguish at lower resolutions.

Explanation and Analysis:

Visual Similarity: In real-world applications, even high-performing models can struggle with classes that have subtle differences, especially when trained on low-resolution images. This phenomenon is not unique to this study; for example, **ResNet**, despite its depth and residual connections, also showed vulnerabilities in distinguishing between visually similar classes in datasets like CIFAR-10 and ImageNet, particularly at lower resolutions.

Insufficient Features: Low-resolution images inherently limit the amount of detail available for the model to learn from. This limitation means that some classes may not have enough distinctive features to be easily separable. For instance, while **DenseNet** attempts to alleviate this by using dense connections to enhance feature reuse, it also faces challenges in scenarios where the resolution limits the richness of features that can be extracted.

Impact of Misclassifications:

Misclassifications highlight the importance of dataset quality and model architecture. While our model performs well overall, the errors suggest areas for improvement, such as enhancing the model's ability to capture finer details or increasing the resolution of input images. Future work could involve using higher-resolution datasets or applying more sophisticated architectures like **Vision Transformers (ViT)**, which have shown promise in better capturing global and local features, even in complex or low-resolution images.

5.5. Challenges and Limitations

Challenges:

1. **Computational Resources:**
    - **Explanation:** Training deep neural networks requires substantial computational power, especially when dealing with larger datasets, deeper architectures, or more complex models. Limited access to high-performance hardware, such as GPUs or TPUs, presented a challenge in this project. Due to these constraints, the ability to experiment with more sophisticated architectures, like deeper Convolutional Neural Networks (CNNs), was restricted.
    - **Impact:** This limitation likely slowed down the training process and constrained the exploration of hyperparameter tuning, which could have improved performance. Additionally, the inability to test larger models restricted the project from pursuing state-of-the-art techniques such as ensemble models or more resource-intensive architectures like Vision Transformers (ViT).

2. **Dataset Size:**
    - **Explanation:** The fixed size of the CIFAR-10 dataset (60,000 images) limited the amount of data available for training and evaluation. While data augmentation techniques were used to increase the effective size of the dataset, the lack of additional training data likely constrained the model's ability to generalize well to unseen images.
    - **Impact:** Larger or more diverse datasets could have provided the model with more opportunities to learn, potentially improving both its accuracy and generalization capabilities. The use of a larger dataset, such as CIFAR-100 or ImageNet, could have provided richer features for the model to learn from, thus enhancing performance in recognizing complex patterns across different classes.

Limitations:

1. **Model Complexity:**

    - **Explanation:** The model used in this project, while effective in classifying CIFAR-10 images, is relatively simple compared to more modern architectures like **ResNet** (Residual Networks), **DenseNet** (Dense Convolutional Networks), or **EfficientNet**. These more advanced models employ techniques such as residual connections, dense layers, and sophisticated scaling strategies that allow them to learn more complex patterns and generalize better to new data.
    - **Impact:** While the model achieved a solid accuracy on CIFAR-10, more complex architectures would likely yield better results by improving the model's ability to learn deeper and more hierarchical representations of the data. The limitations in computational resources and model complexity prevented the exploration of these advanced architectures, which might have significantly improved both performance and efficiency.

2. **Resolution of Images:**

    - **Explanation:** CIFAR-10 images have a low resolution of **32x32 pixels**, which limits the amount of detail that can be captured by the model. For certain tasks, especially when distinguishing between visually similar classes (e.g., cats vs. dogs), the small resolution restricts the model's ability to learn fine-grained features. This limitation becomes even more apparent when compared to datasets like ImageNet, where higher-resolution images provide richer details for the model to exploit.
    - **Impact:** The low resolution reduces the amount of information available for the model to work with, potentially leading to misclassifications when the model is unable to capture subtle differences between classes. Using higher-resolution images, or transferring the model to a dataset that provides more detail, could improve classification accuracy. Furthermore, models like **EfficientNet**, which utilize compound scaling (simultaneously adjusting resolution, depth, and width), would benefit more from larger image sizes, enabling better feature extraction.

Additional Insights:

- **Potential Solutions to Challenges and Limitations:**

    - **Overcoming Computational Resource Constraints:** Using cloud-based GPU services or leveraging more efficient architectures (such as EfficientNet) could help mitigate the limitations in computational power. Additionally, techniques

like **transfer learning** (pre-training on a large dataset and fine-tuning on CIFAR-10) could reduce the computational burden while improving accuracy.
    - **Addressing Dataset Size Limitation:** Using transfer learning from larger datasets like **ImageNet** or employing **synthetic data generation** (e.g., using GANs to generate additional data) could expand the dataset and help the model generalize better. This would also mitigate overfitting, which can occur when training on smaller datasets.

- **Broader Implications:**

    - **Model Scalability:** The relatively simple architecture limits the model's scalability to larger datasets or more complex tasks. In future work, scaling up the model using more advanced techniques, such as **residual connections** (ResNet) or **dense connectivity** (DenseNet), could improve both the depth and quality of feature extraction.
    - **Real-World Application Constraints:** Low-resolution images may perform well in controlled datasets like CIFAR-10, but in real-world applications (e.g., medical imaging or autonomous driving), high-resolution data is often required. The limitations of using low-resolution images in this project underscore the importance of developing models that can scale and adapt to different input resolutions for more practical applications.

6. Conclusion

6.1. Summary

This project explored the application of Convolutional Neural Networks (CNNs) for image classification on the CIFAR-10 dataset, a widely recognized benchmark in computer vision. The primary objective was to build and optimize a deep learning model to achieve high classification accuracy. Key steps included:

- **Data Preprocessing**: The CIFAR-10 dataset was normalized, and labels were one-hot encoded to prepare for model training.
- **Model Architecture**: A CNN with multiple convolutional, pooling, and dropout layers was implemented, enhanced with batch normalization to improve training stability and prevent overfitting.
- **Training and Validation**: The model was trained using the Adam optimizer, with data augmentation techniques and callbacks for learning rate reduction and early stopping to fine-tune the training process.

- **Evaluation**: The model's performance was evaluated on the test set, demonstrating effective generalization capabilities.

The results indicated that the CNN model achieved substantial accuracy on the CIFAR-10 dataset, showcasing the effectiveness of deep learning techniques in practical image classification tasks.

6.2. Contributions

- **Balanced Architecture:** One of the key contributions of this study is demonstrating that a well-balanced CNN architecture, combined with effective training strategies like data augmentation, can achieve high performance on a challenging dataset like CIFAR-10. Unlike much deeper networks, our model maintains a balance between depth and computational efficiency, making it more accessible for applications with limited resources.
- **Optimization Techniques:** The use of the Adam optimizer, combined with learning rate adjustments and early stopping, is another significant benefit. These techniques not only ensured faster convergence but also reduced the risk of overfitting, a common challenge in deep learning. In comparison, earlier models like VGG required significantly more tuning and computational resources to achieve similar levels of performance.
- **Generalization Ability:** The ability of the model to generalize well to unseen data, as evidenced by its correct classifications, is particularly noteworthy. This indicates that the model is not merely memorizing the training data but is learning underlying patterns that are transferable to new data. This is a significant improvement over older models like **AlexNet**, which, while groundbreaking at the time, did not incorporate modern techniques like batch normalization and advanced data augmentation strategies, leading to less robust generalization.
- **Impact on Real-World Applications:** The insights gained from this study are directly applicable to real-world scenarios where image classification models are used, such as in autonomous vehicles, security systems, and medical image analysis. The balanced approach taken in this study—leveraging a moderately deep network, effective augmentation, and modern optimization techniques—provides a template for building robust, efficient models that can be deployed in resource-constrained environments without sacrificing performance.

6.3. Future Work

Several potential improvements and future research directions were identified during this project:

- **Advanced Architectures**: Exploring more complex architectures such as ResNet, DenseNet, or EfficientNet could further enhance classification performance by leveraging deeper networks and more sophisticated design principles.

- **Transfer Learning**: Utilizing pre-trained models on larger datasets and fine-tuning them for CIFAR-10 classification could improve accuracy and reduce training time.
- **Larger Datasets**: Training on larger and more diverse datasets could provide better generalization and robustness, addressing the limitations posed by the fixed size of CIFAR-10.
- **Hyperparameter Optimization**: Conducting a more exhaustive search for optimal hyperparameters using techniques such as grid search or Bayesian optimization could fine-tune the model for even better performance.
- **Real-world Applications**: Applying the developed model to real-world image classification tasks, such as medical imaging or autonomous driving, could validate its practical utility and identify further areas for refinement.

By pursuing these directions, future work can build upon the foundations laid by this project, pushing the boundaries of what is achievable in image classification with CNNs.

Bibliography and References

[1] R. Doon, T. K. Rawat, and S. Gautam, "CIFAR-10 Classification using Deep Convolutional Neural Network," in *2019 International Conference on Signal Processing and Communication (ICSC)*, NOIDA, India, 2019, pp. 186-190

[2] LeCun, Y., Bengio, Y., & Hinton, G. (2015). "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436-444, doi: 10.1038/nature14539.

[3] D. M. Montserrat, Q. Lin, J. Allebach, E. J. Delp, (2017), "Training Object Detection And Recognition CNN Models Using Data Augmentation", *Journal of Electronic Imaging, Society for Imaging Science and Technology*

[4] B. A. Skourt, A. El Hassani, A. Majda.(2021), "Mixed-pooling-dropout for convolutional neural network regularization", *Journal of King Saud University – Computer and Information Sciences 34 (2022) 4756–4762.*

[5] W. Jung, D. Jung, B. Kim, S. Lee, W. Rhee, J.Ho Ahn.(2019), "RESTRUCTURING BATCH NORMALIZATION TO ACCELERATE CNN TRAINING", *Proceedings of the* 2 nd *SysML Conference*, Palo Alto, CA, USA, 2019.

[6] S. Mehta; C. Paunwala, B. Vaidya, (2020), *CNN-based Traffic Sign Classification using Adam Optimizer, 2019 International Conference on Intelligent Computing and Control Systems (ICCS)*

[7] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, (1998), "Gradient-Based Learning Applied to Document Recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998. doi: 10.1109/5.726791.

[8] A. Krizhevsky, I. Sutskever, G. E. Hinton, (2012), “ImageNet classification with deep convolutional neural networks”, *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*

[9] B. S. Rao, "Accurate leukocoria predictor based on deep VGG-net CNN technique," *IET Image Processing*, vol. 14, no. 10, pp. 2241-2248, July 2020. doi: 10.1049/iet-ipr.2018.6656.

[10] “GoogLeNet Explained: The Inception Model that Won ImageNet”, Viso.ai, [Online], Available: https://viso.ai/deep-learning/googlenet-explained-the-inception-model-that-won-imagenet/, [Assessed: 2024]

[11] S. Targ, D. Almeida, K. Lyman, (2016), “Resnet in Resnet: Generalizing Residual Architectures”, doi.org/10.48550/arXiv.1603.08029

[12] M. Tan, Q. V. Le, (2019), “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”, Journal of International Conference on Machine Learning, 2019. doi.org/10.48550/arXiv.1905.11946

[13] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, M. Shah, (2022), “Transformers in Vision: A Survey”, *Journal of ACM, ACM Computer Surveys, Vol. 54, No. 10s*

[14] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A Simple Framework for Contrastive Learning of Visual Representations," in *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, Vienna, Austria, 2020, pp. 1597-1607.

[15] M. Tan and Q. V. Le, "EfficientNetV2: Smaller Models and Faster Training," in Proceedings of the 38th International Conference on Machine Learning (ICML 2021), Vienna, Austria, 2021, pp. 1234-1244.

[16] “One Hot Encoding in Machine Learning”, Geeks for Geeks, [Online], Available: https://www.geeksforgeeks.org/ml-one-hot-encoding/, [Assessed: 2024]

[17] Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning.

[18] Perez, L., & Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning.

[19] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[20] Chollet, F. (2017). *Deep Learning with Python*. Manning Publications.