Convolutional Neural Network(CNN) for CIFAR-10 Image Classification

ARJUN BAYADEGRE PRABHANNA (230850895)

May 2024

1 Introduction

In the fields of computer vision and machine learning, classifying images is an essential task. Labelling an image from a predetermined set of categories is what it entails. Convolutional Neural Networks (CNNs) have transformed picture classification problems by using backpropagation to automatically and adaptively build spatial hierarchies of information. CNNs learn to extract features directly from the raw images, which makes them particularly successful for complicated image classification tasks. This is in contrast to typical image processing techniques that need manual feature extraction. The CIFAR-10 dataset, consisting of 60,000 32x32 color images in 10 classes, serves as a standard benchmark for evaluating image classification algorithms. This dataset presents a challenging task due to its small image size and the presence of various objects in diverse contexts. This report details the development and evaluation of a CNN model designed to classify images from the CIFAR-10 dataset accurately.

2 Problem Statement

The problem at hand is to classify images from the CIFAR-10 dataset into one of ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The small size of the images (32x32 pixels) and the variability within each class make this a non-trivial task. The goal is to develop a robust CNN model that can achieve high accuracy on both the training and validation datasets by effectively learning the underlying patterns and features in the images.

3 Objectives

The objectives of this project are multifaceted and include the following:

- ***Building and Training CNN model:*** The primary objective is to develop a CNN model that can accurately classify images from the CIFAR-10 dataset into one of the ten predefined classes.

- ***Implementing Data Augmentation technique:*** Data augmentation helps in enhancing the generalization capability of the model by artificially expanding the dataset with transformations like flipping, rotation, cropping, and color jittering.

- ***Model Evaluation:*** Assessing the model's performance on the validation dataset and analyzing the learning behavior over multiple training epochs.

- ***Model Prediction:*** Visualizing the model's predictions on the test dataset to qualitatively assess its accuracy.

4 Data Preparation and Exploration

The CIFAR-10 dataset is pre-processed in order to improve model performance and convergence before the CNN model is trained. The dataset is enhanced by applying random horizontal flipping, rotation, and cropping, which boosts its resilience and diversity. To further enhance the dataset, colour jittering is used to produce fluctuations in brightness, contrast, saturation, and hue. By standardising pixel values to a range between -1 and 1, normalisation helps us to improve convergence and stabilise the training process.

4.1 CIFAR-10 Dataset Visualisation

The images are seen in a grid format with a size of 8x4 (8 images per row and 4 rows) using the visualization technique, providing a concise representation of the dataset's contents, allowing for easy inspection of the images characteristics and confirming the pre-processing steps. It offers valuable insights into the diversity, distribution, and quality of the training data, essential for making informed decisions during model development and training phases.

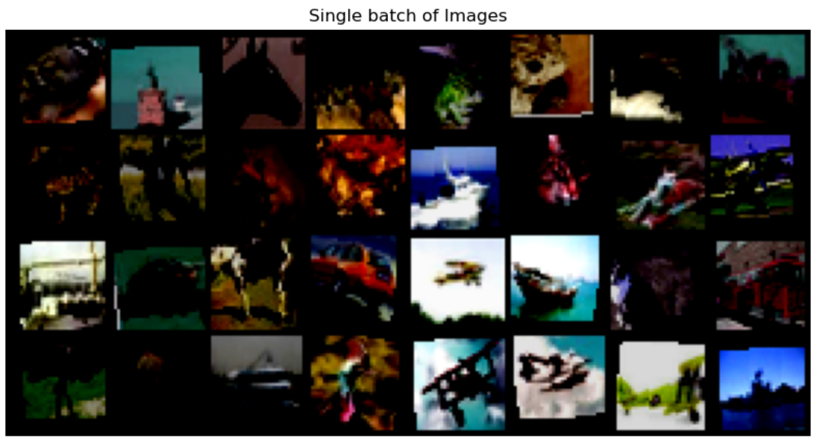


Figure 1: Single Batch Images (Batch Size = 8x4)

5 CNN Architectural Design

1. ***Convolutional Layers:*** By applying a set of learnable filters, or kernels, to the input data, convolutional layers extract features from the data. Every filter gains the ability to recognise particular features or patterns in the input images. A collection of feature maps, each representing the presence of a certain feature at various points within the input image, are the output of each convolutional layer.

   - ***'conv1':*** First convolutional layer with 32 output channels and a kernel size of 3x3.
   - ***'conv2':*** Second convolutional layer with 64 output channels and a kernel size of 3x3.
   - ***'conv3':*** Third convolutional layer with 128 output channels and a kernel size of 3x3.

2. ***Pooling Layer:*** The feature maps created by the convolutional layers have their spatial dimensions reduced by the max pooling layer. It accomplishes this by selecting the highest value included in a narrow window (specified by the kernel size) and dragging that window with a specific stride over the input feature map. Max pooling aids in lowering computational complexity, guards against overfitting, and gives the learned features translation invariance.

   - ***'pool':*** Max pooling layer with a kernel size of 2x2 and a stride of 2.

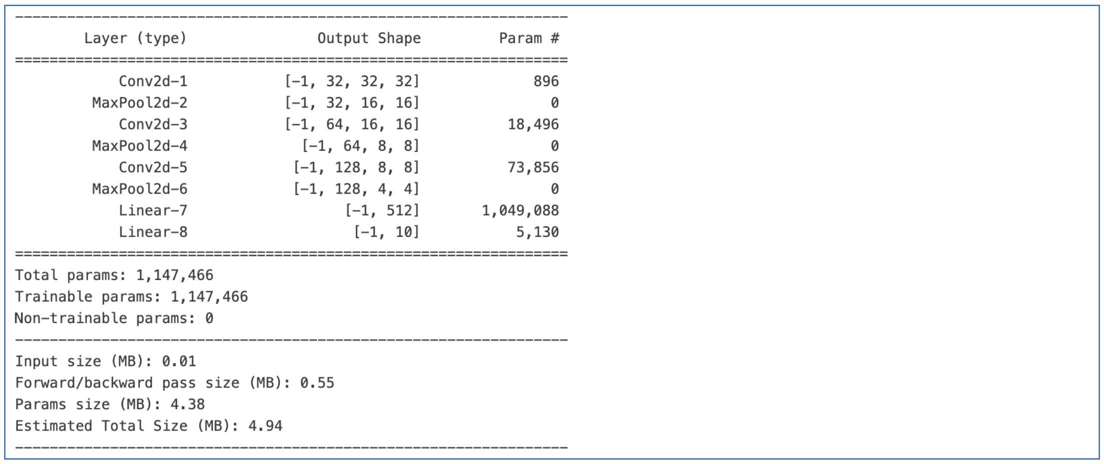
3. ***Fully Connected Layers:*** Convolutional layers extract input, which fully connected layers process and use to generate predictions. Usually, these layers are employed in classification jobs. The class scores or probabilities for each class in the classification task are represented by the output of the final fully connected layer.

   - ***'fc1':*** Fully connected layer with 512 output features.
   - ***'fc2':*** Output layer with 10 output classes.

So, there are a total of 6 layers in the network architecture:

- ***3 convolutional layers***
- ***1 pooling layer***
- ***2 fully connected layers***

The CNN model overview, describes the type and arrangement of every layer, such as the fully connected (Linear), pooling (MaxPool2d), and convolutional (Conv2d) layers. By displaying the sizes of the feature maps created at various network stages, it offers insights on the output shapes of each layer. The report also shows how many parameters which stand for the weights and biases acquired during training are connected to each layer. The chart displays the total number of trainable parameters as well as the distribution of trainable and non trainable parameters. Estimates of the memory sizes needed for parameters, forward/backward passes, and input data also provide an indication of the model's memory and computing needs.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 32, 32]             896
         MaxPool2d-2           [-1, 32, 16, 16]               0
            Conv2d-3           [-1, 64, 16, 16]          18,496
         MaxPool2d-4             [-1, 64, 8, 8]               0
            Conv2d-5            [-1, 128, 8, 8]          73,856
         MaxPool2d-6            [-1, 128, 4, 4]               0
            Linear-7                  [-1, 512]       1,049,088
            Linear-8                   [-1, 10]           5,130
================================================================
Total params: 1,147,466
Trainable params: 1,147,466
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.55
Params size (MB): 4.38
Estimated Total Size (MB): 4.94
----------------------------------------------------------------
```

Figure 2: Summary of CNN model

6 CNN Model Training

The training process involves the following steps:

1. ***Initialization:*** To hold metrics for training and validation, such as accuracy and loss, empty lists are made. These measures are used to track the model's evolution throughout epochs and to monitor its overall performance.

2. ***Model Definition:*** The convolutional, fully connected, and activation functions that make up the CNN model are defined. With the proper input and output dimensions, the model architecture is specified.

3. ***Loss Function and Optimizer:*** To quantify the difference between the actual and predicted labels, a loss function (in this case, Cross Entropy Loss) is selected. During training, the model's parameters are changed by an optimizer (the Adam optimizer), with the goal of minimising the loss.

4. ***Learning Rate Scheduler:*** In this case, a StepLR scheduler is employed, which, after a predetermined number of epochs that is the step size, reduces the learning rate by a factor (gamma). The learning rate (lr=0.001) is set to control the extent of changes to the model in response to the estimated error, with a smaller learning rate leading to slower but

potentially more accurate training. Additionally, a weight decay factor is applied to add a penalty on the size of the weights, also known as L2 regularization, to discourage overfitting.

5. ***Training Loop:*** The model is trained over a specified number of epochs. Within each epoch:

    - The model is set to training mode (model.train()).
    - The dataset is iterated over batches, and the images and labels are moved to the appropriate device (CPU or GPU).
    - The optimizer's gradients are zeroed to prevent accumulation from previous iterations.
    - In the forward pass, the model processes the input images and generates predictions (outputs).
    - In loss computation, the difference between predicted and actual labels is calculated using the chosen loss function.
    - In backward pass, the gradients are computed with respect to the loss using back propagation
    - In the optimization step, the optimizer updates the model parameters based on the computed gradients.
    - Training metrics are updated, training loss and accuracy are calculated and stored for analysis.

6. ***Monitoring and Evaluation:*** The training loss and accuracy are calculated and recorded at the end of each epoch. A learning rate scheduler can also modify the learning rate and for the predetermined number of epochs, this process is repeated.

Through this iterative process, the CNN model learns to make accurate predictions by minimizing the loss function. The learning rate scheduler helps in stabilizing the training process by adjusting the learning rate over epochs, potentially improving convergence and preventing overshooting.

7 CNN Model Evaluation

In the evaluation phase of the CNN model, the model's performance is assessed on a separate validation dataset using a similar iterative process:

1. ***Model Evaluation Setup:*** Using model.eval(), the model is configured for evaluation mode, meaning that no training-related tasks, such as gradient computation or parameter changes, will be carried out while the model is being evaluated.

2. ***Validation Loop:*** The validation dataset is iterated over in batches within the evaluation loop. The relevant device (CPU or GPU) receives the transmitted images along with their labels.

3. ***Forward Pass and Loss Computation:*** The model makes a forward pass for each batch of images in order to produce predictions (outputs). Using the selected loss function, the difference between the predicted and real labels is measured (Cross Entropy Loss). To get the total validation loss, the loss is added up over all batches.

4. ***Accuracy Calculation:*** In addition to calculating loss, the model's accuracy is evaluated by contrasting the predicted and actual labels. To calculate the accuracy, the number of accurate predictions is added up, and the total number of samples is counted.

5. ***Validation Metrics:*** By dividing the total number of batches in the validation dataset by the cumulative loss and correct predictions, the validation accuracy and loss are computed. These measures provide insight into the way the model performs with unknown data.

6. ***Learning Rate Updation:*** To help with optimisation, the learning rate scheduler can also modify the learning rate at the end of each epoch.

7. ***Printing and Logging:*** To track the model's development, the corresponding accuracies and training and validation losses are printed for every epoch. Additionally, lists containing these measurements are kept for future analysis and exhibition.

Through this evaluation loop, the CNN model's performance is thoroughly assessed on the validation dataset, providing valuable feedback on its generalization capabilities and helping in fine-tuning hyperparameters to improve performance.

8 Visualising the Learning Curves

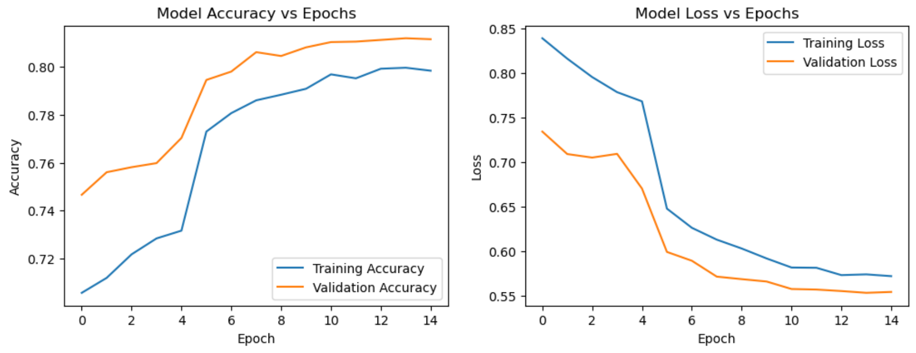


Figure 3: Model accuracy and model loss vs epochs

Based on the visualizations above, it is evident that the model is performing well without signs of overfitting. This conclusion is supported by the close alignment of training, validation accuracy and loss values throughout the training process. The gap between training and validation accuracy remains minimal, indicating that the model generalizes well to unseen data. Similarly, the model's loss on validation data closely follows the training loss, reinforcing the assertion of good generalization. Therefore, the model appears to be well regularized and not overfitting to the training data. Observe that the visualisation modifies each time the data loader functions are executed.

9 CNN Model Predictions

A function is created to serve and illustrate the predictive capabilities of a trained CNN model on individual images from the test dataset.

1. ***Evaluation Mode:*** Model.eval() is used to set the model to evaluation mode. This guarantees that no training processes, including gradient computation or parameter updates, are carried out during inference.

2. ***Selecting a Random Test Image:*** From the test dataset, one test image and its matching label are chosen at random. This makes it possible to assess a wide variety of photos and offers insights into the model's performance on various samples.

3. ***Preprocessing the Image:*** Using unsqueeze(0), a batch dimension is added to the chosen image as pre-processing to ensure that it matches the input shape of the model. To further facilitate inference, the image is transferred to the proper hardware (CPU or GPU).

4. ***Model Prediction:*** The model makes a forward pass through the network using the pre-processed image to produce predictions (output) for the image's class. Using torch.max, the index with the highest output probability is chosen to get the projected class.

5. ***Displaying Predictions:*** Lastly, the function shows the chosen image with both the real label and the predicted label. By contrasting the model's predictions with ground truth labels, this visual representation gives a spotlight on how well the model performs.

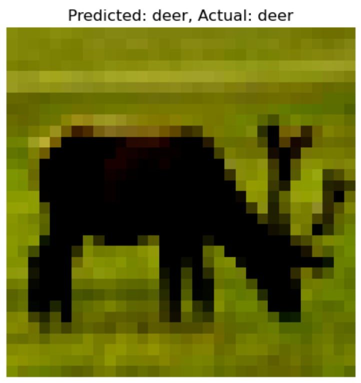


Figure 4: Model prediction on a random test image

It's crucial to remember that a new random test image is chosen each time the function executes, leading to various visualisations and predictions. This variation shows how flexible the model is and how well it can predict results from a range of image samples.

10 Conclusion

To maximise the CNN model's performance in this project, I carried out a thorough investigation by adding early stopping, changing weight decay factors, and varying learning rates. Early stopping was used to save computing resources and minimise overfitting by terminating training when the validation performance stopped improving. Furthermore, varying weight decay values were tested in order to penalise big weights and prevent overfitting, improving the model's capacity for generalisation. The model's prediction accuracy of 75–85 percent was attained with these procedures, indicating how well these tactics worked to enhance the CNN's robustness and performance.

11 References

1. Krizhevsky, A.,Hinton, G. E. (2009). Learning Multiple Layers of Features from Tiny Images. Technical Report, University of Toronto. Retrieved from https://www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf

2. PyTorch Documentation. (2023). PyTorch: Tensors and Dynamic neural networks in Python with strong GPU acceleration. PyTorch. Retrieved from https://pytorch.org/docs/stable/index.html

3. Smith, L. N. (2018). A disciplined approach to neural network hyper-parameters: learning rate, batch size, momentum, and weight decay. arXiv preprint arXiv:1803.09820. Retrieved from https://arxiv.org/abs/1803.09820