Assignment 2

Student: Emanuele Bellini; Student's email: belliem@usi.ch

November 4, 2024

Image Classification using CNNs

2 **Dataset Analysis** We start by loading the CIFAR-10 dataset using PyTorch's built-in functionality:
`torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)`
where train is true if we want to load the train set and `transform = transforms.ToTensor()` converts the PIL image into a PyTorch tensor and scales the pixel values of the image from the range [0, 255] to [0, 1].
We then analyze the dataset and we can observe the following:

- Dataset Composition:
  - Training set: 50,000 images
  - Test set: 10,000 images
  - Perfect class balance: 5,000 images per class in training
- Class Distribution:
  - 10 distinct classes with equal representation (10% each)
  - Classes: planes, cars, birds, cats, deer, dogs, frogs, horses, ships, trucks
- Image Properties:
  - RGB color format (3 channels)
  - Fixed resolution: 32×32 pixels
  - High variability of sample images within the same classes

Figure 1 and 2 show one example image for each class in both train and test set, while Figure 3 and 4 show the class distribution.

Figure 1: Sample images from training set

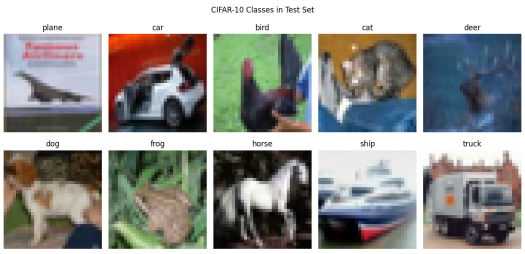

Figure 2: Sample images from test set

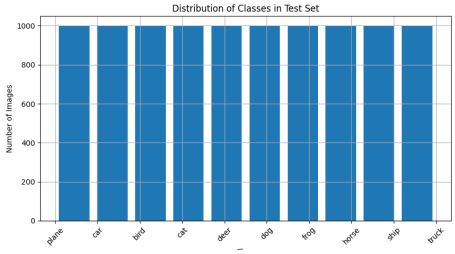

Figure 3: Class distribution in test set showing balanced classes

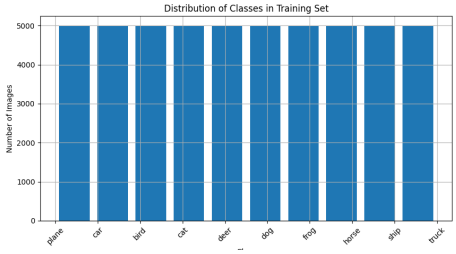

Figure 4: Class distribution in training set showing balanced classes

3 **Data Format Analysis**
The analysis of dataset format suggest a transformation for PyTorch compatibility: The original format is `<class 'PIL.Image.Image'>`, while the format after the transformation is `<class 'torch.Tensor'>`.
As previously mentioned, the necessary conversion is implemented using `transforms.ToTensor()`. This function performs a series of transformations:

- PIL Image → `torch.float32` tensor
- Pixel value scaling: [0, 255] → [0, 1]
- Channel reordering: (H,W,C) → (C,H,W)

The shape of the image as `torch.Tensor` Object is:

- Shape: `torch.Size([3, 32, 32])`
- Total elements per image: $3 \times 32 \times 32 = 3{,}072$

The dimensions can be interpreted as follows:

- First dimension (C = 3):
  - Represents color channels (RGB)
  - Index 0: Red channel
  - Index 1: Green channel
  - Index 2: Blue channel
- Second dimension (H = 32):
  - Image height in pixels
  - Spatial dimension for vertical axis
- Third dimension (W = 32):
  - Image width in pixels
  - Spatial dimension for horizontal axis

4 **Dataset Normalization**
To normalize the dataset, we first calculate the dataset statistics, i.e. mean and variance for each channel.
Then, we use `transforms.Compose` to chain transformations:
we apply `transforms.ToTensor()` followed by `transforms.Normalize(mean=means, std=stds)`, where means and stds are the computed dataset statistics. Doing that, we have implemented a precise channel-wise normalization that guarantees to have mean=0 and variance=1 for each channel, as shown in these results:

- Original Statistics:
  - Red channel - Mean: 0.4914, std: 0.2470
  - Green channel - Mean: 0.4822, std: 0.2435
  - Blue channel - Mean: 0.4465, std: 0.2616
- Normalized image statistics::
  - Red channel - Mean: 0.0000, Std: 1.0000
  - Green channel - Mean: -0.0000, Std: 1.0000
  - Blue channel - Mean: -0.0000, Std: 1.0000

5 **Validation Set Creation**
We split the test set into validation set and a new test set to improve model generalization capabilities. To do so, we use `torch.utils.data.random_split` function and a fixed seed to force reproducibility throughout the experiments.

6 **CNN Architecture Design**
We implement the CNN starting from lecture code as suggested, chosing the Conv-Conv-Activ-Pool-Conv-Conv-Activ-Pool-FC pattern, allowing for progressive feature extraction through multiple conv layers.
No padding (padding=0) and stride=1 as recommended.
The layer configuration is the following:

- Input: 3×32×32 RGB images
- First block: Conv(3→32), Conv(32→64), ReLU, MaxPool
- Second block: Conv(64→128), Conv(128→128), ReLU, MaxPool
- Output: FC layer (3200→10)

7 **Training Implementation & Analysis**
Here we show the implementation details and the motivations behind them, based on hempirical results and code analysis.

- **Learning Rate Analysis (0.03)**: By testing the models with different value of learning rate, we can observe:
  - lr=0.01: Good enough for the base CNN implementation, but shows a too slow convergence for the improved CNN implementation, decreasing accuracy by 5%.
  - lr=0.05: Training instability, fluctuating loss and poor performance
  - lr=0.03: We reach the best accuracy for both base and improved CNN implementation, showing the best balance between stability and convergence speed

  Analyzing the output produced by the code with `lr=0.03`, we can see a steady loss decrease from 1.9648 to 0.7381. Moreover, we see smooth accuracy improvement without oscillations and no signs of learning rate-related instability. This confirms our choice for `lr=0.03`.

- **Epoch Selection (4 epochs)**:
  - Convergence evidence on train set:
    - Epoch 1: Rapid improvement (54.81%)
    - Epoch 2: Strong gains (68.57%)
    - Epoch 3: Continued improvement (72.52%)
    - Epoch 4: Final refinement (75.00%)

- Validation stability:
  - Initial jump to 66.86%
  - Steady improvement to 72.84%
  - No overfitting signs after epoch 4
- Training Progress Details:
  - Per-epoch metrics:
    - Epoch 1: 54.81% train, 66.86% validation
    - Epoch 2: 68.57% train, 66.96% validation
    - Epoch 3: 72.52% train, 72.44% validation
    - Epoch 4: 75.00% train, 72.84% validation

We can conclude that utilizing less than 4 epochs will decrease the model accuracy. Also, utilizing more epochs than 4 will bring no additional improvements, because the model will start to overfit as shown in Figure 5: the accuracy of the train set increases, while the validation set accuracy starts to oscillate.

- **Architectural Analysis and Justification.**
  We opted for a 2 Conv Blocks design as follows:
  - First block (3→32→64 channels):
    - First layer identifies basic visual elements (edges, colors, textures)
    - Second layer combines these into simple patterns (corners, curves)
    - MaxPooling reduces spatial dimensions while keeping dominant features
  - Second block (64→128→128 channels):
    - Builds higher-level abstractions (object parts, shapes)
    - Maintains same channel size (128) to consolidate features
    - Final pooling provides translation invariance
  - Why not deeper? The main reasons are the constraints on the input size: since eace image is composed by 32x32 pixels, 2 conv-pool blocks already reduce the image dimensions to 8x8, since each one halves the spatial dimentions. So, adding another block will results in 4x4 features, that is too small for meaningful patterns. Moreover, additional depth will risk to extraxt noise rather than useful features.
  - **Design Principles Applied**:
    In the first block we prevent information bottleneck by apply- ing a gradual reduction: 32×32 → 28×28 → 14×14. Here, no padding preserves edge information and MaxPool after ReLU mantains the strongest activations. Still, 14x14 preserves the ini- tial datail for low-level features.
    In the second block we have: 14×14 → 10×10 → 5×5 obtaining a 5×5 final size, that is 25 spacial locations for decision making. 25 positions × 128 channels = 3200 features for classification.
    ReLU positioned after double convolutions allows for feature combination before non-linearity and reduces total number of operations. Pooling after activation downsamples only meaningful features, while it helps prevent overfitting by reducing parameters.

    Steady convergence during training, good generalization (similar train/val accuracy) and no signs of underfitting or overfitting clearly support the design choices, obtaining a final test accuracy of 72.48 %.

8 **Training Analysis**

From the loss curves shown in Figure 6, we can observe the following training evolution:

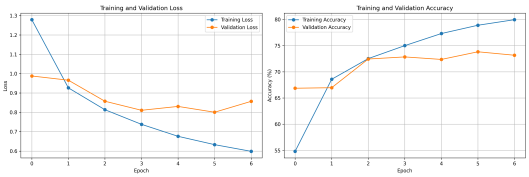

Figure 5: Validation loss increases at epoch 5

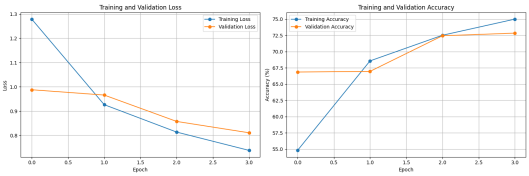

Figure 6: Training and Validation Metrics for Base Model

– **Loss Evolution**:
  * Initial phase (Epoch 0-1):
    · Training loss starts at 1.28, showing rapid decrease
    · Validation loss starts at 0.98, decreasing more gradually
    · A higher initial training loss is expected, due to random initialization
  * Middle phase (Epoch 1-2):
    · Training loss continues steady decline
    · Validation loss follows similar trend but with slight gap
    · Both curves maintain smooth, parallel trajectories
  * Final phase (Epoch 2-3):
    · Training loss stabilizes around 0.74
    · Validation loss stabilizes around 0.81
    · Small, consistent gap (0.07) between curves

  In conclusion, validation loss continues decreasing until final epoch, with a smooth decrease from 0.99 to 0.81. There is no divergence between training and validation losses, in particular the final loss gap remains small and stable. In addition, the accuracy plot also shows a similar behaviour. So, the plot clearly shows no sign of overfitting and good generalization capabilities of the model.

9 **Enhanced Architecture**

  Starting from the base CNN that achieved 72.48% accuracy, we have implemented an improved CNN architecture that boosts the performance to 82.12% (+9.64%). This enhanced architecture combines modern CNN practices with the consideration of the CIFAR-10 dataset characteristics:

  * The architecture progressively processes input images (32×32×3) through three main blocks:
    · First block: 3→64→128 channels, padding=1 preserves spatial dimensions
    · Second block: 128→256→256 channels, maintains feature richness
    · Third block: 256→512 channels, final feature extraction
    · Each block followed by MaxPool(2×2) reducing spatial dimensions by half
    · Final feature map (512×4×4 = 8192) fed to FC layers (8192→1024→10)

* **Key improvements and motivations**:
  - BatchNorm after each conv layer: Stabilizes training by normalizing feature distributions, enabling 0.1 learning rate for faster convergence
  - Dropout (p=0.2): Applied after pooling and FC layers, prevents overfitting despite more than 3× learning rate increase. Rate chosen empirically as balance between regularization and information preservation
  - GELU activation: Replaces ReLU for smoother gradients, improving feature extraction especially with BatchNorm
  - Padding=1: Preserves spatial information early in network, crucial for the small 32×32 images. Allows deeper architecture while maintaining feature map sizes
  - Kaiming initialization: Properly scales initial weights for GELU activation, prevents vanishing/exploding gradients in deeper network

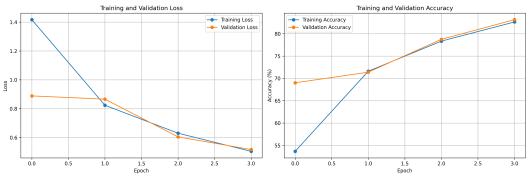

Figure 7: Train and validation loss for the improved CNN architechtures

10 **Seed Sensitivity Analysis**

We trained the base CNN model with 5 different seeds (5-9) while keeping all other hyperparameters constant. The results provide insights into the model's robustness:

* **Test Accuracy Distribution**:
  - Mean accuracy: 72.15%
  - Standard deviation: 0.48%

  As shown in Figure 8, we observe:
  - Performance variation range: 1.44% (between 71.44% and 72.88%)
  - All seeds maintain performance above 70%, indicating stable learning
  - Seed 6 shows notably higher performance, suggesting favorable initial conditions

  These results demonstrate that while random initialization does impact final performance, the model maintains consistent learning capability across different seeds. The relatively small standard deviation (0.48%) suggests that our architecture and training process are robust.

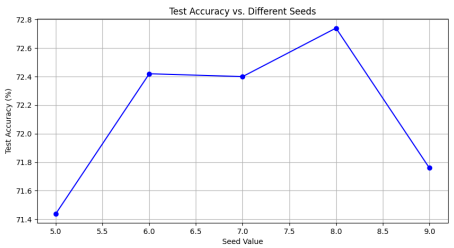

Figure 8: Test loss with different seeds