Assignment 6: Image Classification & Deep Learning in PyTorch

1 Instructions

In this assignment, you will build, train, and compare several neural network architectures for image classification using **PyTorch**. The goal is to deepen your understanding of architectural design tradeoffs, inductive bias, and how different components impact model performance.

You will implement and evaluate models on the **CIFAR-100** dataset, which consists of 60,000 32x32 color images across 100 classes. Moving beyond simple linear models, you will leverage PyTorch's automatic differentiation and hardware acceleration to train high-capacity deep learning models.

1.1 Submission Requirements

You must submit two components to Gradescope:

1. **The Report:** A plaintext Markdown document containing the 5 sections described in Part 2.
    - **No Images:** Describe your findings with text and data tables.
    - **Character Limit:** 7,500 characters.
2. **Supplemental Material:** Your `.ipynb` notebook containing all code, plots, and raw results.

---

2 Part 1: Implementation (The Notebook)

You will author a Jupyter Notebook (`.ipynb`) to perform the following experiments. It is **highly recommended** that you use **Google Colab** with a GPU runtime to ensure your models train in a reasonable amount of time.

Global Setup:

- Set random seeds for reproducibility (e.g., `torch.manual_seed(0)`).

- Configure your device dynamically to use a GPU if available: `device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")`.
- Load the **CIFAR-100** dataset using `torchvision.datasets`, applying necessary standardizations via `torchvision.transforms`.
- Create `DataLoader` instances for the train and test sets.

2.1 Experiment 1: The Multi-Layer Perceptron (MLP)

Concept: Multi-Layer Perceptrons (MLPs) or fully connected deep networks treat inputs as flat, 1D vectors. We use this as a baseline to see how a model *without* spatial awareness performs on visual data. It highlights the difficulty linear models and standard MLPs face when modeling complex, high-dimensional datasets like images without the help of hand-crafted features.

2.1.1 Task 1.1: Training and Testing Framework

Design modular `train_model` and `test_model` functions.

- Your training loop must iterate through batches, move data/labels to the `device`, compute the forward pass, compute the loss (e.g., `nn.CrossEntropyLoss`), perform the backward pass (`loss.backward()`), and step the optimizer.
- Make sure your `test_model` function uses `model.eval()` and `with torch.no_grad():` to disable dropout and save memory.
- Print the running training loss periodically.
- Both functions should calculate and print the total number of trainable parameters in the model.

2.1.2 Task 1.2: MLP Implementation

- Implement an MLP using `torch.nn.Module`. You may only use linear layers (`nn.Linear`), normalization layers, dropout, and activation functions (e.g., ReLU).
- **Constraint:** Convolutional layers are not allowed.
- **Constraint:** Your model must have $\leq$ **5 million** trainable parameters.
- *Hint:* You will need to flatten the 3x32x32 image tensor into a 1D vector. To stay within the parameter limit, consider making your first hidden layer smaller than the input dimension to act as a bottleneck.
- Train the model and evaluate it on the test set.

Notebook Output:

- Printout of the MLP's parameter count.
- Final Test Accuracy of the MLP.

2.2 Experiment 2: The Convolutional Neural Network (CNN)

Concept: Convolutional Neural Networks (CNNs) introduce an "inductive bias" tailored specifically for images. By using learnable filters that slide across the spatial dimensions of the image, CNNs leverage spatial locality and parameter sharing. This drastically reduces the number of parameters compared to an MLP while often improving accuracy, as the network automatically learns hierarchical features (like edges, then shapes, then objects) directly from the pixel data.

2.2.1 Task 2.1: CNN Implementation

- Implement a CNN using `torch.nn.Module`. You should use convolutional (`nn.Conv2d`), pooling (`nn.MaxPool2d` or `nn.AvgPool2d`), and linear layers.
- **Constraint:** Your CNN architecture must include at least one "advanced" architectural component to serve as the subject of your ablation study later. You must include at least one of the following: Batch Normalization (`nn.BatchNorm2d`), Dropout (`nn.Dropout2d`), or a Residual/Skip Connection.
- **Constraint:** Your model must have $\leq$ **5 million** trainable parameters.
- **Tensor Shape Math:** In your notebook comments, explicitly calculate the spatial dimensions (H_{out}, W_{out}) of the tensor just before it is flattened for the final linear layers, using the formula:
$$H_{out} = \lfloor \frac{H_{in} + 2P - K}{S} + 1 \rfloor$$
- Train the model and evaluate it on the test set.

2.2.2 Task 2.2: Visualizing the Learned Filters

Extract the weights from your **first** convolutional layer. Plot a grid of these filters (e.g., as grayscale or RGB images depending on your channel setup) to see what low-level features the network is learning to detect.

Notebook Output:

- Printout of the CNN's parameter count.
- Final Test Accuracy of the CNN.
- A plot showing the learned visual filters from the first `Conv2d` layer.

2.3 Experiment 3: Ablation Study

Concept: In deep learning, architectures consist of many interacting components (activations, normalization, skip connections, pooling). An ablation study systematically removes or alters one of these components to isolate its specific contribution to the model's overall performance. This teaches you how to empirically validate whether a specific design choice is actually helping your model learn.

Identify a component that you believe is essential to your CNN's success and remove or alter it.

2.3.1 Task 3.1: The Ablation

- Create a modified version of your CNN model class.
- Examples of valid ablations: Removing all BatchNorm layers, swapping ReLU for a linear activation, removing skip/residual connections, or drastically reducing the channel width.
- Train this ablated model using the exact same hyperparameters and procedure as Experiment 2.

Notebook Output:

- Final Test Accuracy of the ablated model.

2.4 Experiment 4: Zero-Shot Classification with CLIP

Concept: Traditional supervised models are limited to a fixed set of labels they were trained on. **CLIP (Contrastive Language-Image Pre-training)**, developed by OpenAI, learns to align images and text in a shared embedding space. This enables **Zero-shot classification**: the ability to classify images into arbitrary categories (like the 100 classes of CIFAR-100) by simply comparing the image's visual features to the text features of a natural language prompt, without any further training.

2.4.1 Task 4.1: CLIP Zero-Shot Evaluation

- **Model Selection:** Navigate to the Hugging Face Models hub and use the `openai/clip-vit-base-patch32` model from the Hugging Face `transformers` library.
- **Preprocessing:** Use `CLIPProcessor` to prepare the CIFAR-100 test images. Note that CLIP typically expects 224x224 images; the processor will handle resizing and specific normalization.
- **Prompt Engineering:** Create a list of 100 text prompts corresponding to the CIFAR-100 classes (e.g., `f"a photo of a {class_name}"`).
- **Zero-Shot Inference:**
  1. Extract image embeddings using `model.get_image_features()`.
  2. Extract text embeddings for all 100 class prompts using `model.get_text_features()`.
  3. Normalize the embeddings and compute the **Cosine Similarity** between the image and all 100 text candidates.
  4. The predicted class is the one with the highest similarity score.
- **Constraint: DO NOT** perform any training or fine-tuning. This is a pure zero-shot evaluation.

Notebook Output:

- The CLIP model name and total parameter count.

- Final Zero-shot Test Accuracy on CIFAR-100.

---

3 Part 2: Content Requirements (The Report)

Your report must be organized into **exactly five sections** with Markdown headers.

3.1 Section 1: Executive Summary

- **One-Sentence Takeaway:** A single sentence summarizing the core finding regarding architectural design (e.g., contrasting the efficiency of CNNs vs MLPs).
- **Summary Paragraph:** Briefly outline the progression of the assignment, highlighting the accuracy differences between the fully connected baseline, the CNN, and the pre-trained model.

3.2 Section 2: Inductive Bias & Parameter Efficiency (MLP vs CNN)

- **Results Table:** A Markdown table comparing the Parameter Count and Final Test Accuracy of your MLP and CNN.
- **Analysis:** Explain *why* the CNN outperformed (or matched) the MLP despite having fewer or similar parameters. Use the concept of **Inductive Bias**. How do the properties of convolution (parameter sharing and sparse connectivity) better align with the geometric nature of image data?

3.3 Section 3: Ablation Study Analysis

- **The Hypothesis:** What architectural change did you make, and why did you hypothesize it would impact performance?
- **The Result:** Report the drop in accuracy.
- **Interpretation:** Explain the mechanical reason for the performance drop. For example, if you removed Batch Normalization, discuss internal covariate shift or gradient flow.

3.4 Section 4: The Paradigm Shift (CLIP Zero-Shot)

- **Results Comparison:** Report the accuracy gap. Discuss how the **CLIP model** achieved its results without being trained on the CIFAR-100 training set, compared to your from-scratch CNN. Use the concept of **Semantic Alignment**.
- **Preprocessing Impact:** Briefly explain why the CLIP model required a different preprocessing step (e.g., resizing to 224x224, specific normalization) than your scratch models. What happens to the **Vision Transformer (ViT)** patches and positional embeddings if you ignore this?

3.5 Section 5: Reflection

- **Debugging Tensor Shapes:** Reflect on the difficulty of tracking tensor shapes (Batch, Channels, Height, Width) through convolutional and pooling layers. Did explicitly calculating H_{out} and W_{out} help?
- **Hardware Acceleration:** Discuss your experience using the GPU (`.to(device)`). Why are GPUs essential for modern deep learning compared to the pure NumPy CPU calculations from previous assignments?

---

4 Grading Rubric

Each of the five sections will be weighted equally (**20% each**).

Criterion	Excellent (5)	Good (4)	Satisfactory (3)	Okay (2)	Poor (1)
Section 1: Executive Summary	Takeaway is insightful and quantitatively grounded. Summary concisely captures the core results and architectural comparisons.	Takeaway is clear. Summary covers the main arc of the assignment.	Takeaway is generic. Summary is present but vague.	Summary misses key elements of the analysis (e.g., ignores the pre-trained aspect).	Missing or unintelligible.
Section 2: Inductive Bias (MLP vs CNN)	Correctly compares models and deeply explains performance gaps using the concepts of inductive bias, parameter sharing, and sparse connectivity.	Table is present. Explanations are correct but slightly mechanical or surface-level.	Compares models but misses the connection to parameter sharing or inductive bias.	Data table missing or accuracy values are incorrect.	Missing or fails to implement the models correctly.

Criterion	Excellent (5)	Good (4)	Satisfactory (3)	Okay (2)	Poor (1)
Section 3: Ablation Study	The ablation hypothesis is insightful. The mechanical explanation for the performance drop demonstrates a deep understanding of the network's internals.	A valid hypothesis and accurate performance drop are reported. The interpretation is logical.	Describes results but the "why" behind the performance drop is vague.	The implemented change is trivial or results are not compared to the baseline.	Missing or no ablation study performed.
Section 4: Paradigm Shift	Accurately compares scale/performance and provides a clear, technically sound explanation of why strict preprocessing is required for pre-trained weights.	Good comparison. Mentions preprocessing but lacks specific technical reasoning for why it matters.	Compares accuracies but ignores the preprocessing discussion entirely.	Accuracies reported are implausible due to failed implementation.	Missing or no pre-trained model evaluated.
Section 5: Reflection	Thoughtful reflection on PyTorch mechanics (tensor shapes, auto-grad) and a clear understanding of the necessity of hardware acceleration (GPUs).	Good reflection on debugging and libraries.	Generic reflection (e.g., "PyTorch is hard but GPUs are fast").	Minimal effort.	Missing.