CNN Design Challenge — Build Your Own Image Classifier

Overview

This assignment is a **mini AI competition**. Each group will design and train a **custom Convolutional Neural Network (CNN)** to classify images from the subset of **Tiny ImageNet dataset**. Your model will be trained and validated using the data provided. The instructor will test your model on a **hidden (unseen) 20% test set** that you will not have access to. Your final **score and ranking** will depend on your model's accuracy on this unseen test set.

Dataset Details

- **Dataset name:** Tiny ImageNet (Subset)
- **Number of classes:** 15
- **Image size:** 64 × 64 pixels (RGB)
- **Total images:** 8250
- **Provided data split:**
    - **Training:** 70% (5775) (download from here)
    - **Validation:** 10% (825) (download from here)
    - **Hidden test (kept by instructor):** 20% (1650)

Your goal is to build a model that generalizes well to the unseen 20% test split.

Objective

Design and train a **Sequential CNN (from scratch)** that performs accurate classification on Tiny ImageNet data. You will submit your model, and the instructor will evaluate it on the hidden set.

Rules and Constraints

1. Architecture Design

- You must **build your own CNN (deep structure with sequential layers)** using standard layers only:
  `Conv2D`, `Pooling`, `BatchNorm`, `Dropout`, `Flatten`, and `Dense`.

- No limit of no. of Layers.
- You can set your maximum no. of epochs and patiences (early stopping) by yourself.

- You **cannot** use:

    - Any **pre-trained model** (e.g., ResNet, VGG, EfficientNet)
    - Any **transformer**, **attention-based**, or **residual/skip connection/denseblock** architectures
    - Any **Hyperparameter Optimization (HPO)** or **Neural Architecture Search (NAS)** algorithm (e.g., Optuna, Ray Tune, DARTS, etc.)

2. Model

- Must be built using **PyTorch** or **TensorFlow/Keras** (**GPU version**)

3. Data Augmentation

- You are allowed to use reasonable augmentations such as rotation, flipping, color jitter, cropping, etc.

4. Training Rules

- You do not need to split your provided data further into **train/validation** as it is already provided.
- You must **not** have access to the hidden test set.
- Use consistent random seeds for reproducibility (always choose the best one).

5. Submission Files

Submit Two Files

1. **Your code file** (.py or .ipynb) with model architecture
2. **model.pth** file with trained weights

Required Components:

- **Model class** - Your exact neural network architecture
- **Load function** - Code to load weights from model.pth
- **Predict function** - Takes test data, returns predictions

Important:

- Your model class must exactly match your training architecture
- We will call your load function to get your model
- We will call your predict function with our test data

How We Test

1. We import your model class
2. We load your weights using your load function
3. We provide our test data to your predict function
4. We calculate accuracy from your predictions

Before Submitting

- Test that your code loads model.pth successfully
- Ensure predict function works with sample data
- Verify architecture matches your trained model

6. Team Rules

- It should be the same group that you submitted for the project.
- Only **one submission per group.**
- After Submission please leave **a comment** with your **teammate's name and ID.**

Bonus:

Top 3 groups with the highest accuracy will receive **bonus marks.**

Submission Details

- **Deadline:** 17 November 2025, 11.59pm
- **Submission format:** `GroupX_Assignment.zip` (replace X with group number)

Helpful Tips

- Before you start, first load the dataset and make sure everything is okay.
- Start with a small CNN (3–4 conv layers) and build complexity step-by-step.
- Add **BatchNorm** and **Dropout** for stability and regularization.
- Experiment with **learning rate schedules** and **data augmentation**.
- Keep your validation accuracy close to your training accuracy — avoid overfitting.
- Explore different regularization techniques.
- Save your best model using `ModelCheckpoint` or manual save.