AI600 – Deep Learning
Assignment 2: Quick, Draw! Challenge

Namra Nadeem
Roll Number: 25280097

GitHub Repository: `https://github.com/namranadeem-wq/Deep-Learning-Assignment-2`
Leaderboard Score: 78.63

1 Introduction

This assignment studies the width–depth trade-off for image classification using only MLPs (CNNs not allowed). Quick, Draw! sketches are noisy and ambiguous: the same class can appear in many styles, and different classes may share similar shapes. The goal is to design an efficient MLP ($\leq$3M parameters, $\leq$40 epochs) that generalizes well, not merely memorizes training examples.

2 Dataset and Preprocessing

Each sample is a 28×28 grayscale bitmap. Inputs are flattened to 784D vectors and normalized to $[0, 1]$ to stabilize optimization. The provided training set has 60,000 samples and 15 classes; an 80/20 stratified train/validation split (48,000/12,000) is used for model selection and generalization analysis.

3 Models and Results

3.1 Pancake (Width Focus)

Architecture: $784 \to 2048 \to 15$ with ReLU and dropout.
Parameters: 1,638,415.
Observation: Train accuracy rose to 96.47% while validation peaked at $\approx$76.79% (around epoch 17). Validation loss increased after mid training.
Explanation: A very wide hidden layer has strong capacity and quickly fits training data, including noise. The growing train–val gap indicates overfitting: width increases memorization power more than transferable feature learning in this noisy sketch setting.

3.2 Tower (Depth Focus)

Architecture: 6 hidden layers, width 256, with BatchNorm + ReLU + dropout.
Parameters: 536,847.
Observation: Validation improved to 78.66% (epoch 19) with a smaller train–val gap (train $\approx$83.80%).
Explanation: Depth enables hierarchical transformations where intermediate layers can progressively build abstract representations (e.g., strokes $\to$ shapes $\to$ object cues). BatchNorm stabilizes training for deep MLPs, and the smaller gap suggests better generalization than the wide shallow model.

3.3 Champion (Architecture Search)

Architecture (pyramid): 784 → 512 → 512 → 256 → 256 → 128 → 15 with BatchNorm + GELU + dropout.
Training: 14 epochs (checkpoint saved at epoch 14).
Observation: Best validation in experiments ≈79% (peak 79.29% in one run; 78.81% at epoch 14 in the saved run). Leaderboard score: 78.63.
Explanation: A pyramid combines early-layer capacity (to capture diverse drawing styles) with later compression (to keep only salient information). GELU provides smoother gradients than ReLU, improving optimization. Regularization (dropout/weight decay) controls overfitting while retaining enough capacity to outperform Pancake and Tower.

4 Theoretical Analysis

4.1 Why deep models if a single hidden layer is a universal approximator?

The Universal Approximation Theorem guarantees *existence* of a shallow approximator but not *efficiency* or *learnability*. Functions with hierarchical structure can require exponentially many neurons in a single hidden layer, while deep networks reuse intermediate features across layers. Empirically, depth achieved higher validation accuracy with fewer parameters (Tower vs Pancake), indicating better parameter efficiency and generalization.

4.2 Confusion and Uncertainty

A confusion matrix (computed on the validation set) typically shows highest confusion among visually similar classes (e.g., Cookie vs Donut, Pizza vs Wheel). Such errors often reflect *aleatoric uncertainty*: sketches may lack distinguishing details and multiple classes share similar outlines. If confusion persists despite high training accuracy, ambiguity in the data is a major contributor rather than pure underfitting.

5 Model Comparison

Model	Parameters	Epochs	Train Acc	Val Acc (best)
Pancake	1,638,415	20	96.47%	76.79%
Tower	536,847	20	83.80%	78.66%
Champion	∼1.0M	14	86.38%	78.81% (saved run)

Table 1: Comparison of width-focused, depth-focused, and champion architectures.

6 Regularization and Generalization

Overfitting was controlled via dropout, weight decay (AdamW), batch normalization, and selecting epochs based on validation performance. The Pancake model showed the clearest overfitting signature (rising validation loss), while deeper architectures reduced the train–val gap and improved validation accuracy.

7 Generative AI Usage Disclosure

Generative AI was used for conceptual clarification and debugging guidance (e.g., resolving notebook execution-order issues, saving/loading checkpoints, and structuring report text). All

experiments, code execution, and final submission decisions were performed and verified by the author.