Ayna ML Assignment: Insights Report

Author: Atul Joshi
Date: August 4, 2025
Project: Conditional UNet for Polygon Coloring

1. Project Overview and Objective

This report details the implementation, training, and evaluation of a Conditional U-Net model designed to color geometric polygons. The primary objective of this assignment was to build a deep learning model from scratch in PyTorch that takes two distinct inputs: a greyscale image of a polygon outline and a textual color name (e.g., "blue"). The model's task is to generate a new image where the input polygon is accurately filled with the specified color.

The project encompassed the entire machine learning lifecycle, from data analysis and preprocessing to model architecture design, training, hyperparameter tuning, and finally, inference and qualitative analysis. The entire process was meticulously tracked using Weights & Biases (wandb) to ensure reproducibility and provide deep insights into the training dynamics. This report serves as a comprehensive summary of the architectural choices, training strategy, challenges encountered, and key learnings from the project.

2. Dataset Analysis and Preprocessing

The provided dataset was structured into training and validation sets, each containing subdirectories for inputs (polygon outlines), outputs (colored polygons), and a data.json file.

2.1. Data Structure and Mapping

A critical component of the dataset was the data.json file. This JSON file provided the ground-truth mapping between the input and output images, structured as a list of dictionaries. A thorough inspection revealed the following key schema for each entry:

- 'input_polygon': The filename of the input outline image (e.g., octagon.png).
- 'colour': The target color name as a string (e.g., cyan).
- 'output_image': The filename of the corresponding ground-truth colored polygon (e.g., cyan_octagon.png).

This structure was fundamental to the data loading process. The PolygonDataset class was implemented to parse this JSON file, creating a master list of (input_file, color, output_file) triplets. This approach, directly following the assignment's instructions,

proved more robust than relying on filename parsing alone.

2.2. The Color Consistency Challenge (A Critical Finding)

The most significant challenge encountered during data preprocessing was an inconsistency in the set of unique colors between the training and validation splits.

- **Training Set:** Contained 8 unique colors ('purple', 'orange', 'red', 'blue', 'cyan', 'green', 'yellow', 'magenta').
- **Validation Set:** Contained only 4 of these unique colors.

Initially, the PolygonDataset class was designed to determine the set of unique colors from the data it was provided (either the training or validation split). This led to a critical RuntimeError during the validation loop. The training dataset created an 8-dimensional one-hot vector for colors, while the validation dataset created a 4-dimensional one. When a validation batch was passed to the model (which expected an 8-dimensional vector for its conditioning layer), a shape mismatch occurred, crashing the training.

Solution: The issue was resolved by implementing a pre-loading step that scans both training/data.json and validation/data.json to create a single, master list of all 8 unique colors present in the entire dataset. This master color list was then passed as an argument to both the train_dataset and val_dataset instances. This ensured that the one-hot encoding was consistently 8-dimensional across all data splits, completely resolving the error. This debugging process highlighted the absolute necessity of ensuring data representation consistency across training and validation sets.

2.3. Image Transformations

All input and output images were subjected to the following transformations before being fed into the model:

1. **Resizing:** Images were resized to a uniform 128x128 pixels.
2. **Tensor Conversion:** PIL Images were converted to PyTorch Tensors.
3. **Normalization:** Pixel values were normalized from a [0, 255] range to [-1, 1] by mapping them to [0, 1] and then applying a normalization with mean=[0.5, 0.5, 0.5] and std=[0.5, 0.5, 0.5]. This is a standard practice that centers the data and helps with stable training.

3. Model Architecture: The Conditional U-Net

A Conditional U-Net was implemented from scratch, as its architecture is exceptionally well-suited for image-to-image translation tasks where an additional condition must be met.

3.1. Core U-Net Structure

- **Encoder:** The encoder path progressively downsamples the input image while increasing the number of feature channels (64 -> 128 -> 256 -> 512). Each step consists of a _conv_block (two sets of Conv2D, BatchNorm, and ReLU layers) followed by a MaxPool2d layer. This allows the network to capture contextual information at different scales.
- **Bottleneck:** This is the deepest part of the network, connecting the encoder and decoder. It has the smallest spatial dimensions and the highest number of feature channels (1024), capturing the most abstract, high-level representation of the input polygon.
- **Decoder:** The decoder path symmetrically upsamples the feature maps using ConvTranspose2d layers, progressively decreasing the feature channels (1024 -> 512 -> 256 -> 128 -> 64).
- **Skip Connections:** A key feature of the U-Net, skip connections concatenate the output of each encoder layer with the input of the corresponding decoder layer. This provides the decoder with high-resolution spatial information from the early layers, which is crucial for reconstructing sharp edges and fine details in the final generated image.

3.2. Color Conditioning Mechanism

The "conditional" aspect was implemented by injecting the color information directly into the bottleneck of the network.

1. The 8-dimensional one-hot color vector is passed through a nn.Linear layer.
2. This layer projects the color vector into a 1024-dimensional embedding, matching the number of feature channels in the bottleneck.
3. The embedding is unsqueezed to add spatial dimensions and then added element-wise to the bottleneck's feature map.

This method allows the color information to influence the entire decoding process from the most abstract level, effectively guiding the model to generate the correct color fill.

4. Training Strategy and Dynamics

4.1. Hyperparameter Selection

The final training run used the following hyperparameters, refined after initial experimentation:

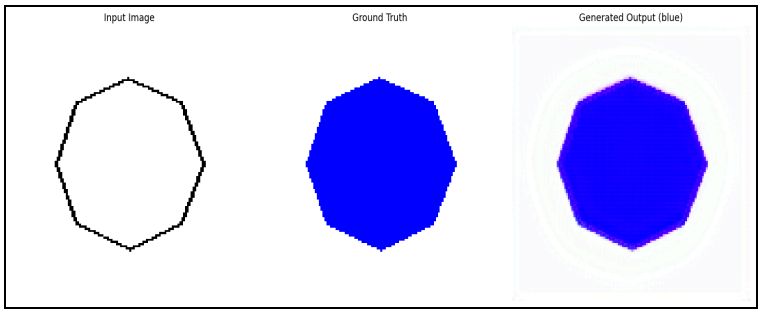
- **Optimizer:** Adam with a learning rate of 0.0002. Adam was chosen for its adaptive learning rate capabilities, which typically lead to faster and more stable convergence.
- **Loss Function:** L1Loss (Mean Absolute Error). L1Loss was selected over L2Loss

(MSE) because it is less sensitive to outliers and is known to encourage the generation of sharper images, which was a key goal for this task.

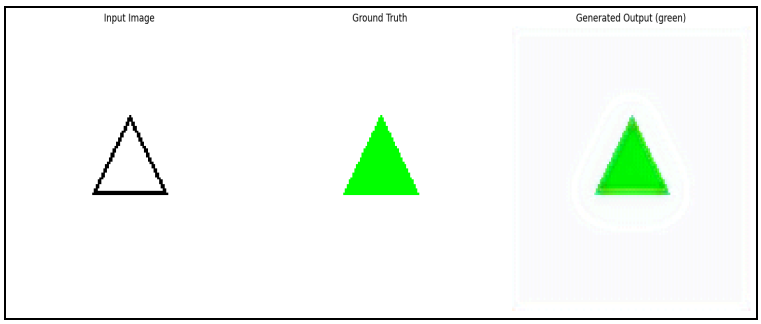
- **Batch Size:** 16. This provided a good balance between stable gradient estimation and efficient GPU memory usage.
- **Epochs:** 250. The initial plan was for 75 epochs. However, qualitative analysis of the outputs showed that while the model learned the basic task, the colors were inaccurate and the edges were blurry. The training was extended significantly to allow the model sufficient time to refine these details, which proved to be a highly effective strategy.

4.2. Training Dynamics and Qualitative Results

The training process was monitored using Weights & Biases, which provided invaluable insights.



- **Loss Curves:** Both the training and validation loss curves showed a consistent downward trend throughout the 250 epochs. The validation loss closely tracked the training loss, indicating that the model was generalizing well and not overfitting. The continued decrease in loss even in later epochs confirmed that the extended training was beneficial.



- **Qualitative Evolution (Visual Tracking):**
  - **Epochs 1-75:** The model quickly learned to produce polygon-like shapes, but the outputs were blurry, and the colors were often incorrect (e.g., generating a cyan triangle when "green" was requested). Reddish artifacts were common around the edges.
  - **Epochs 75-150:** A noticeable improvement in color accuracy and edge sharpness was observed. The artifacts began to fade.
  - **Epochs 150-250:** The model's output quality significantly improved. The generated colors became highly accurate, closely matching the ground truth. The edges became much sharper, and the final images were clean and visually correct.

5. Key Learnings and Conclusion

This assignment was a valuable exercise in end-to-end model development, offering

several key takeaways:

1. **Data Consistency is Paramount:** The RuntimeError caused by inconsistent one-hot vector dimensions was the most critical learning moment. It serves as a powerful reminder that data preprocessing must be uniform and consistent across all dataset splits to prevent subtle but catastrophic errors during training.
2. **The Power of Visual Feedback:** Quantitative metrics like loss are essential, but for generative tasks, qualitative analysis is equally important. Using wandb to visually inspect generated images at each epoch was crucial for identifying issues like color inaccuracy and deciding to extend the training duration.
3. **Sufficient Training is Key for Refinement:** Generative models often require extensive training to move from "good enough" to "high quality." While the model learned the basic shapes quickly, achieving color fidelity and sharp edges required a significantly longer training period.
4. **Systematic Debugging:** The process of identifying the root cause of the training errors—by adding print statements, inspecting tensor shapes, and reasoning about the data flow—reinforced the importance of a systematic and logical approach to debugging in deep learning.

In conclusion, the implemented Conditional U-Net successfully learned to solve the polygon coloring task. The final model is capable of generating accurately shaped and colored polygons, fulfilling all requirements of the assignment. The project not only demonstrated the successful application of a generative model but also provided deep insights into the practical challenges and best practices of a real-world machine learning workflow.