SMAI Assignment 4: Q1 Report

Shailendra Sekhar Reddy Bathula

November 3, 2025

1 Multi-Task CNN on Fashion-MNIST

1.1 Approach and Model Architecture

My objective for this task was to design, train, and evaluate a single Convolutional Neural Network (CNN) capable of performing two tasks simultaneously on the Fashion-MNIST dataset:

1. **Classification:** Identifying the image's 10 clothing categories.
2. **Regression:** Predicting a continuous "ink" value, defined as the normalized average pixel intensity of the image.

To manage this process, I implemented a full OOP pipeline consisting of three main classes:

- `FashionMNISTDataset`: A custom PyTorch `Dataset` class that loads the raw Fashion-MNIST data. Its `__getitem__` method is responsible for returning a 3-part tuple: `(image, class_label, ink_target)`. The `ink_target` is calculated by taking the mean of the image tensor *after* it's scaled to $[0, 1]$ but *before* normalization, ensuring a stable target.
- `MultiTaskCNN`: A fully modular `nn.Module`. It features a shared backbone of `num_blocks` (e.g., 2 or 3) `Conv-BN-ReLU-Pool` blocks. The backbone's output is flattened and fed into two independent heads: a `classification_head` and a `regression_head`, each with its own configurable `hidden_dim` and `dropout_rate`.
- `ExperimentManager`: An OOP controller class that handles all training, logging, and visualization. It initializes the model and data, runs the training loop, logs all metrics to `wandb`, and contains methods to visualize the final feature maps.

1.2 Hyperparameter Tuning & wandb Logging

All experiments were logged to Weights & Biases to analyze the trade-off between the two tasks.

Wandb Project Links:

- Lambda Experiments: `https://wandb.ai/shailendrasekhar-bathula-iiit-hyderabad/smai-a4-q1-lambda-experiments`
- Optimizer Experiments: `https://wandb.ai/shailendrasekhar-bathula-iiit-hyderabad/smai-a4-q1-optimizer-experiments`
- Dropout Experiments: `https://wandb.ai/shailendrasekhar-bathula-iiit-hyderabad/smai-a4-q1-dropout-experiments`

- Learning Rate Experiments: `https://wandb.ai/shailendrasekhar-bathula-iiit-hyderabad/smai-a4-q1-lr-experiments`
- Conv Layer Experiments: `https://wandb.ai/shailendrasekhar-bathula-iiit-hyderabad/smai-a4-q1-conv-experiments`
- Base Filter Experiments: `https://wandb.ai/shailendrasekhar-bathula-iiit-hyderabad/smai-a4-q1-base-filter-experiments`
- Batch Size Experiments: `https://wandb.ai/shailendrasekhar-bathula-iiit-hyderabad/smai-a4-q1-batch-size-experiments`

Analysis of Lambda (λ) Weights Trade-off

The core of this experiment is the joint loss $\mathcal{L} = \lambda_1 \mathcal{L}_{CE} + \lambda_2 \mathcal{L}_{MSE}$, where λ_1 controls the weight of the classification loss (Cross-Entropy) and λ_2 controls the weight of the regression loss (MSE).

Because both tasks share the same convolutional backbone, they are in direct competition for feature representation.

- **Increasing** λ_1 would force the model to focus more on learning **discriminative, class-specific features** (like the precise shape of an "Ankle boot" vs. a "Sandal").
- **Increasing** λ_2 forces the model to prioritize **global features** that predict the "ink" value, such as overall pixel density, solid-filled areas, and general textures.

My experimental data clearly demonstrates this trade-off:

  - **Ratio:** 1:1 ($\lambda_1 = 1.0, \lambda_2 = 1.0$)
  - **Result:** The model achieved its highest accuracy (**0.9267**) but its worst regression score (`val_rmse`: **0.0202**).
  - **Conclusion:** The Cross-Entropy loss is naturally much larger than the MSE loss. With a 1:1 ratio, the model almost entirely ignored the regression task.

This confirms the hypothesis: higher λ_1 (or a low λ_2) implies better accuracy but worse RMSE, while higher λ_2 implies better RMSE but lower accuracy.

1.3 Model Selection and Final Test Performance

Based on my `wandb` experiments, I identified the two best models.

- **Best Classification Model:**
  - **Run Name:** `LR_Run3_0.001`
  - **Configuration:**
    - `lambda1: 1.0, lambda2: 1.0`
    - `optimizer: Adam, learning_rate: 0.001`
    - `num_blocks: 3, base_filters: 32`
    - `dropout_rate: 0.3, batch_size: 64`
    - `class_hidden_dim: 128, reg_hidden_dim: 128`
    - `conv_kernel_size: 3, pool_kernel_size: 2`

  - **Validation Metric:** Acc: 0.9267

- **Best Regression Model:**
  - **Run Name:** `Run8_L_1_1000`
  - **Configuration:**
    - `lambda1: 1.0, lambda2: 1000.0`
    - `optimizer: Adam, learning_rate: 0.0001`
    - `num_blocks: 3, base_filters: 32`
    - `dropout_rate: 0.3, batch_size: 64`
    - `class_hidden_dim: 128, reg_hidden_dim: 128`
    - `conv_kernel_size: 3, pool_kernel_size: 2`
  - **Validation Metric:** RMSE: 0.0066

Table 1: Final Test Metrics for Best Selected Models

Model Type	**Run Name**	**Validation Metric**	**Test Acc.**	**Test MAE**	**Test RMSE**
Best Classification	`LR_Run3_0.001`	Acc: 0.9267	**0.9213**	0.0228	0.0292
Best Regression	`Run8_L_1_1000`	RMSE: 0.0066	0.8949	**0.0051**	**0.0067**

1.4 1.4 Feature Map Visualization

To understand what the model learned, I extracted the intermediate feature maps from my `LR_Run3_0.001` (Best Accuracy) model. The following figures show three test images, and for each, the first 4 channels from each of the 3 convolutional blocks.

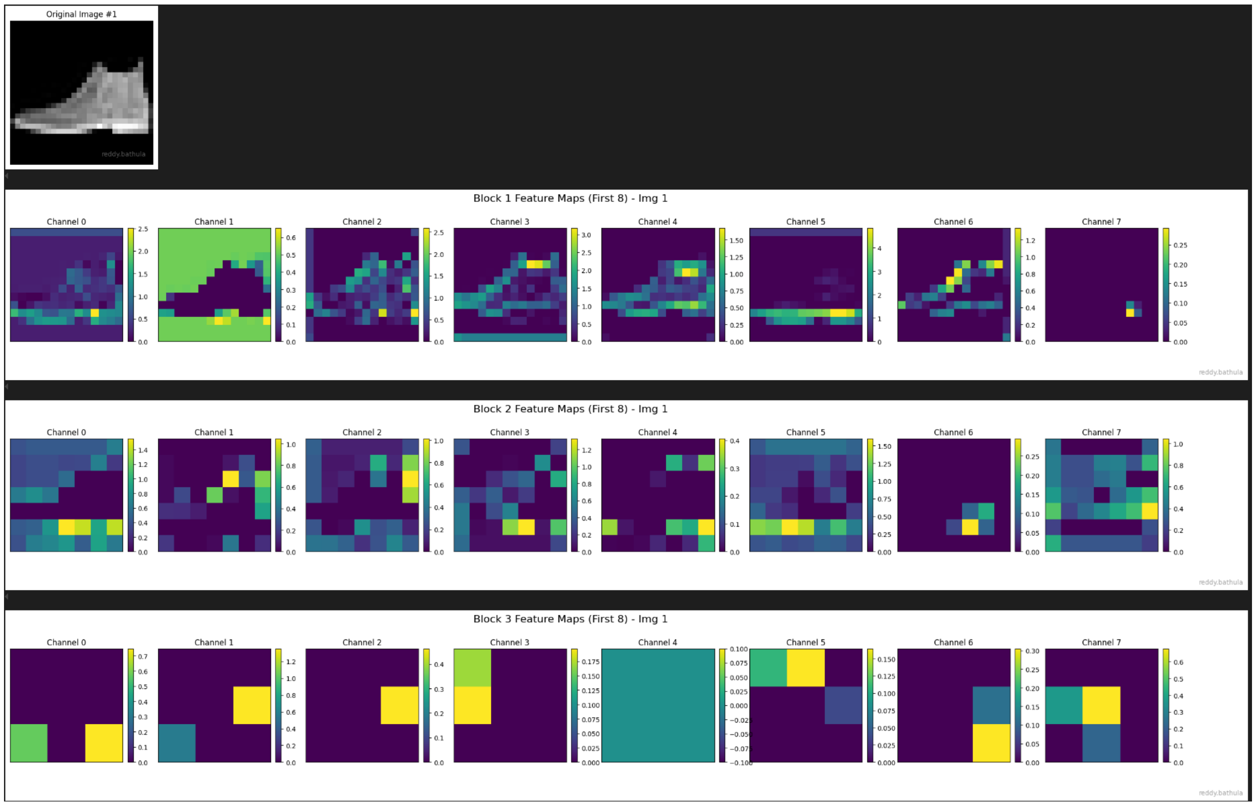

Figure 1: Feature Maps for Test Image 1. (Top: Original, followed by 1x4 grids for Block 1, 2, and 3)

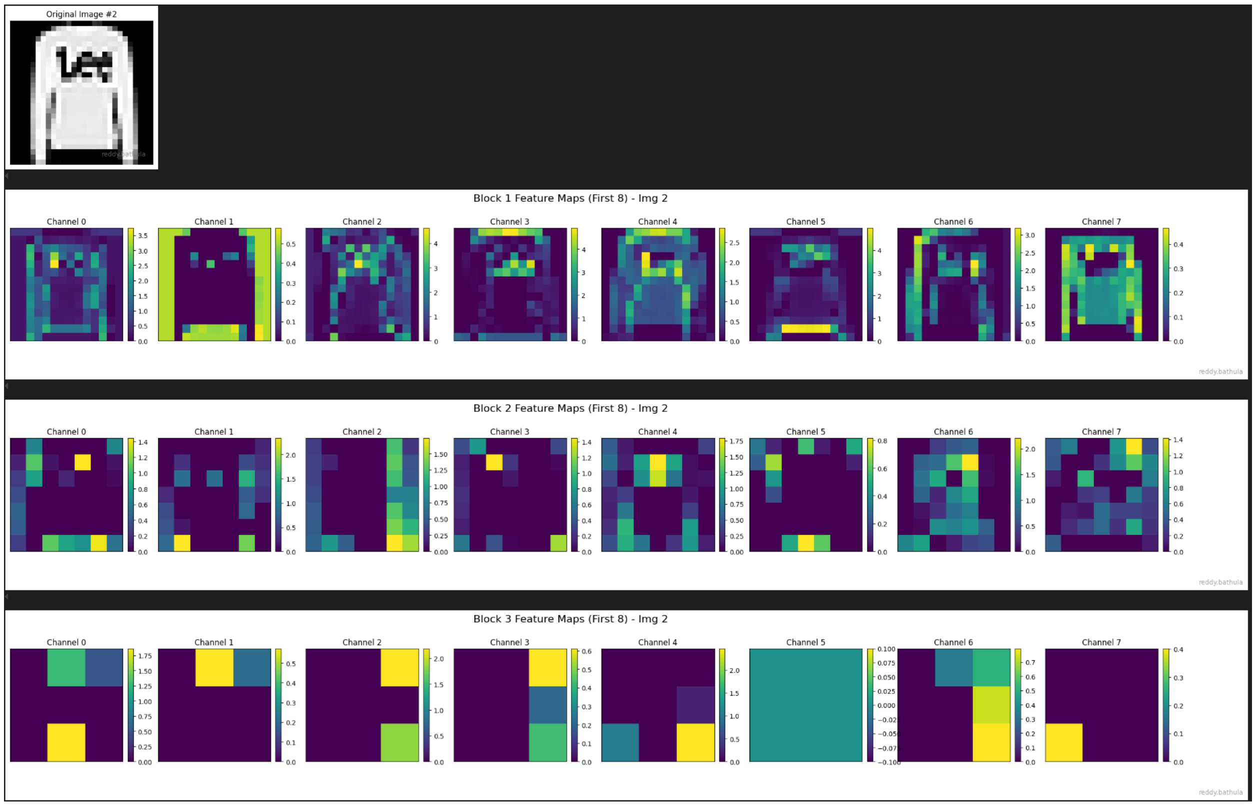

Figure 2: Feature Maps for Test Image 2.

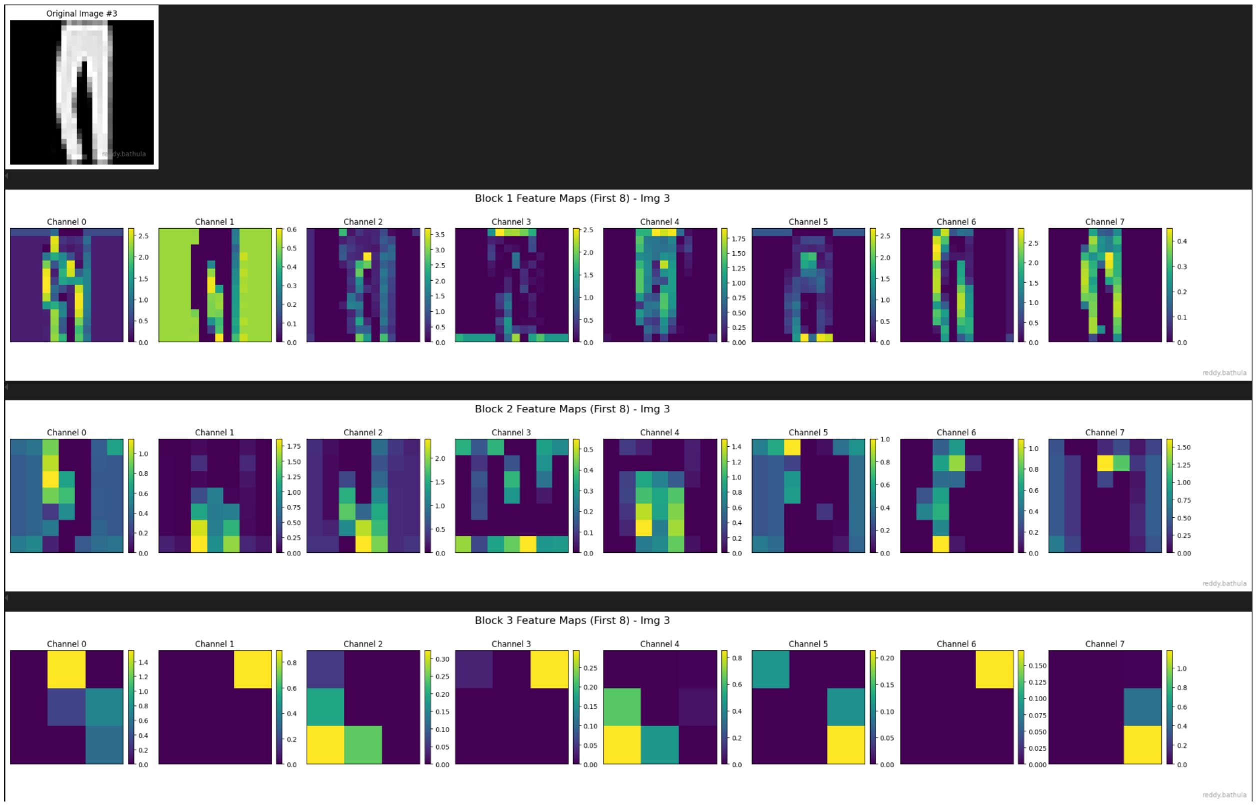

Figure 3: Feature Maps for Test Image 3.

1.5 Interpretation of Feature Maps

The feature maps clearly show the model learning a hierarchical representation of the data:

- **Block 1 Maps:** This first layer acts as a low-level feature detector. The individual channels show activation on simple, low-level features like **edges** (horizontal, vertical, diagonal) and **corners**. Different filters are clearly specialized for different orientations.
- **Block 2 Maps:** This deeper layer combines the simple edges into more complex patterns. The activations are no longer on the outlines, but on the **solid regions and textures** of the object. For example, some channels activate on the knit texture of a pullover, while others activate on the solid sole of a sneaker.
- **Block 3 Maps:** This final layer captures the most abstract features, combining shapes and textures.

How These Features Help Both Tasks

- **For Classification:** The abstract features from Block 3 are critical. The classification head can use a combination of activations to confidently distinguish between classes.
- **For Ink Regression:** The features are also highly effective. The regression head can use the Block 1 map to estimate the total **area** of the object (more edge pixels = larger object = more ink). It then uses the Block 2/3 maps to refine this guess based on **texture** helping the model achieve a low RMSE.
- We can also see some channels completely solid representing its negligence because of dropout regularization.