Multitask Learning for Geometric Shape Classification and Counting

Introduction

In this project I explore multitask learning on the Geometric Shape Numbers (GSN) dataset, where each 28×28 image contains exactly 10 simple geometric shapes drawn from six types (square, circle, and four oriented triangles). The model must solve two tasks at once: (1) **135-way classification** of the exact shape configuration (unordered pair of shape types and their split of 10 objects, e.g. “3 circles + 7 squares”), and (2) **6-dimensional regression** of the counts of each shape type.

The main goal is to study how sharing a convolutional backbone between these two related tasks affects performance and training dynamics. To this end, I use the fixed backbone architecture from the assignment, attach two task-specific heads, and train three variants of the model with different regression loss weights λ_cnt: classification-only (λ_cnt = 0), regression-only, and a multitask setting with λ_cnt = λ*. These settings are then compared using accuracy, F1, RMSE/MAE, and confusion-matrix-based error analysis on the validation set.

Data augmentation

I implemented several augmentations tailored to the GSN dataset, with label-aware updates:

- RandomHorizontalFlip - Flips the image horizontally. Swaps counts of right- and left-pointing triangles.
- RandomVerticalFlip - Flips vertically. Swaps counts of up- and down-pointing triangles.
- RandomRotate90 - Rotates the image by 90°, 180°, or 270° clockwise, updating the labels accordingly
- RandomBrightnessContrast - Mild per-image brightness and contrast jitter.
- RandomGaussianNoise - Adds low-variance Gaussian noise to the image.

For convenience, these are combined into a generic RandomAugmentation class, which applies each transform independently with its own probability.

Evaluation metrics

Classification (135-way)

For the classification task, I consider:

1. **Top-1 accuracy**
2. **Per-pair accuracy**
    - Each label encodes a pair of shapes and the split of 10 items.
    - Using PairLabelEncoder.decode(label), I can extract the unordered pair of shape types.
    - Per-pair accuracy is computed by aggregating over the 15 unordered pairs (e.g., {circle, up}):
        - A prediction is counted as correct at the pair-level if the predicted pair matches the true pair, regardless of the exact 1–9 split.

Regression (6-D counts)

For regression I compute:

- **Per-class RMSE**
- **Per-class MAE**
- **Overall RMSE/MAE**

Confusion matrix and error analysis

I implemented a reusable ConfusionMatrix class that:

- Accumulates counts of (true_class, predicted_class) pairs across batches,
- Exposes:
    - The full matrix as a tensor or NumPy array,
    - Global accuracy from the diagonal,
    - Per-class accuracy,
    - top_k_best_worst(k) to list the best- and worst-classified classes.

A ConfusionMatrixPlotter then visualizes the matrix as a heatmap

Architecture used

Additionaly to the provided backbone, the **classification head** consists of:

- Dropout(0.6)
- Linear(256, 135)

The **regression head** consists of:

- Dropout(0.5)
- Linear(256, 192)
- ReLU()
- Dropout(0.4)
- Linear(128, 6)

To mitigate overfitting I applied relatively strong label-consistent augmentations. On the model side, I used substantial dropout in both heads (0.6 in the classification head and 0.5/0.4 in the regression head).

Results

Classification metrics

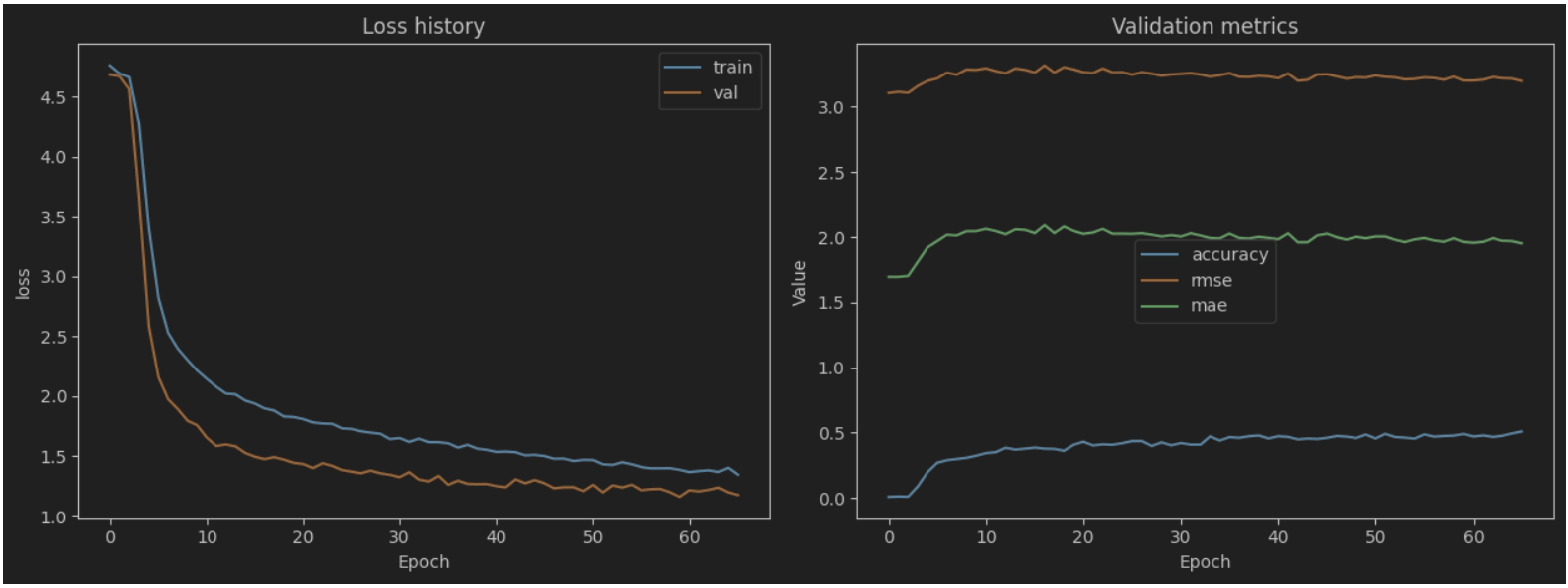
Setting	λ_cnt	**Top-1 Acc**	**Per-pair Acc**
Classification only	0.0	1.0	0.509
Regression only		0.1	
Multitask (λ = 1.4)	1.4	1.0	0.5

Regression metrics

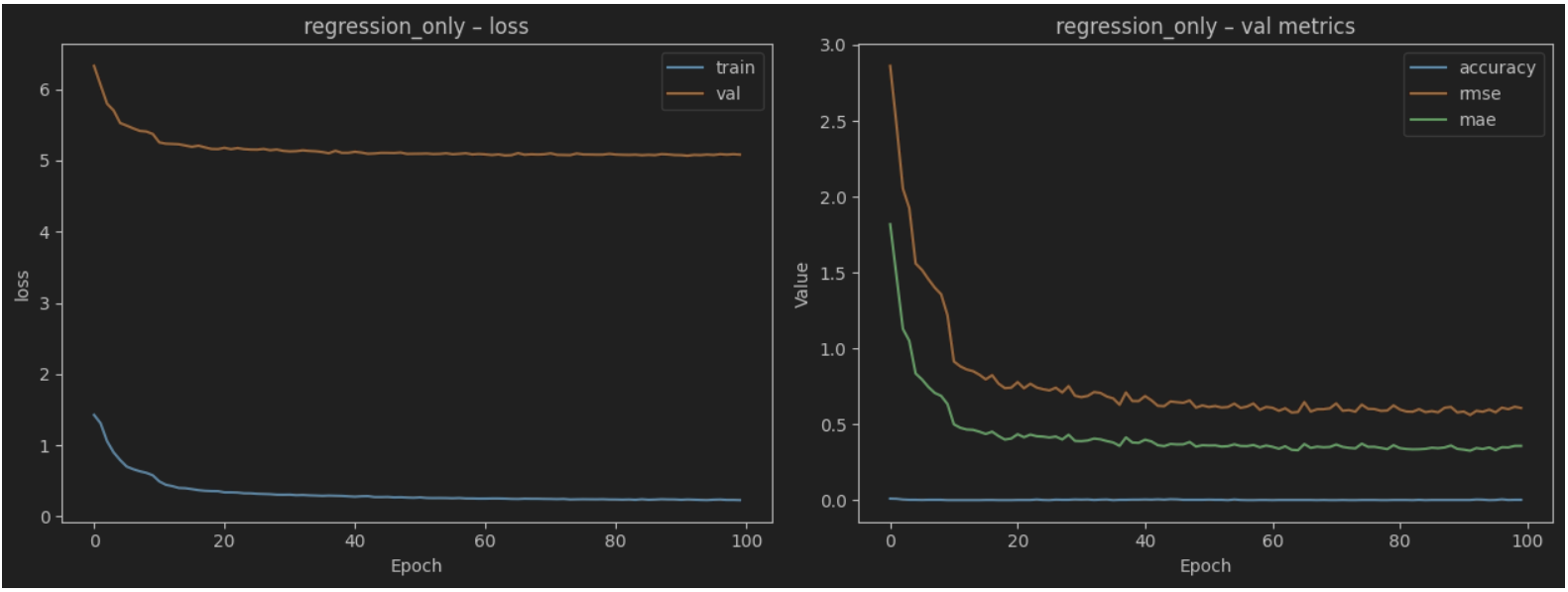
Setting	λ_cnt	**Overall RMSE**	**Overall MAE**
Classification only	0.0	3.2	2.0
Regression only			
Multitask (λ = 1.4)	1.4	1.0	0.6

Learning curves

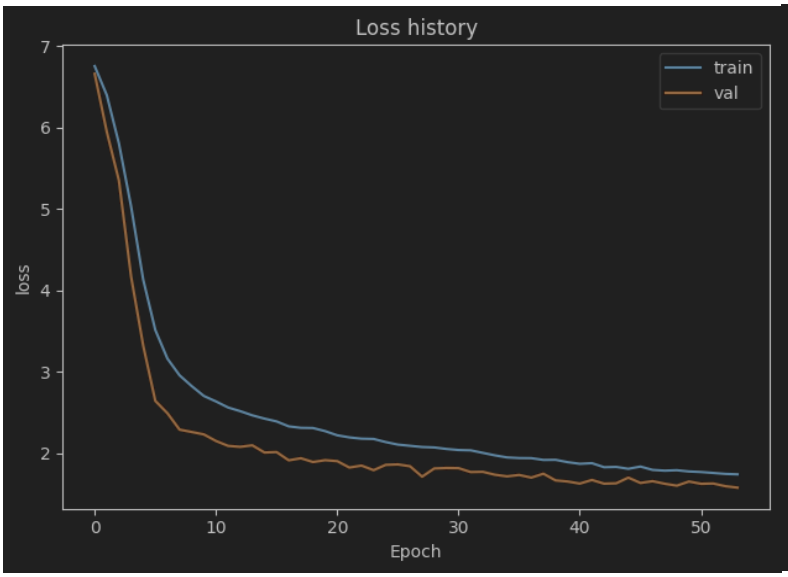
Classification only

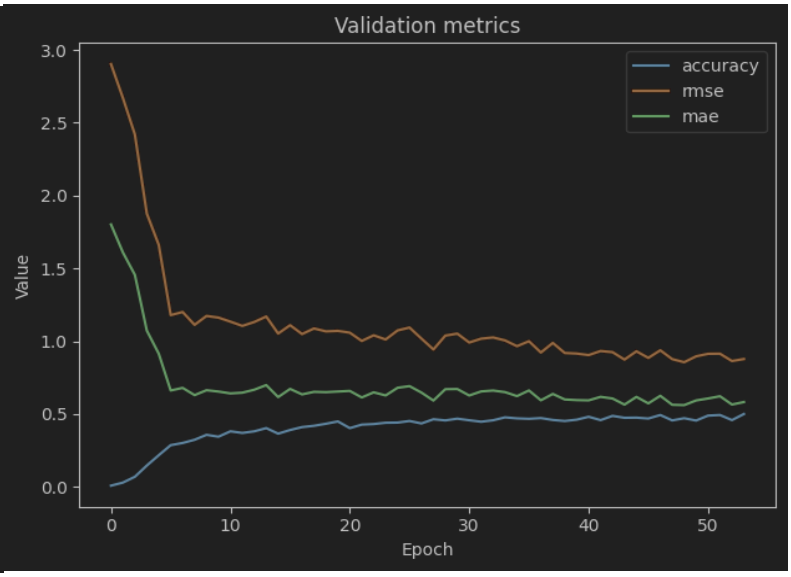


Regression only



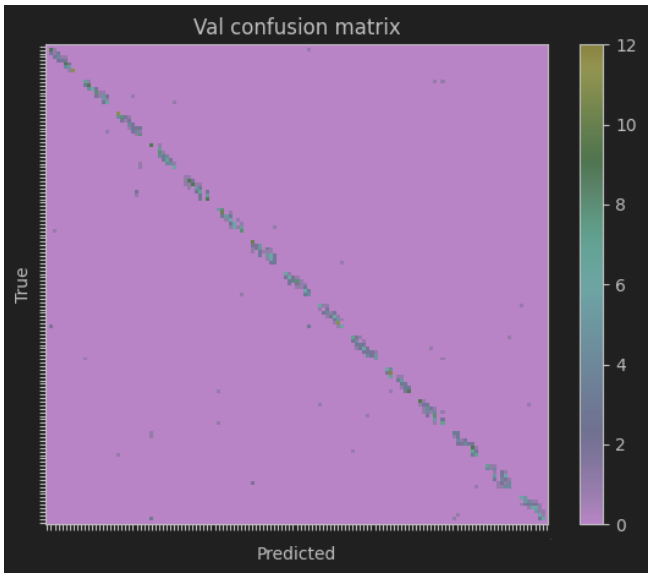
Multitask (λ = 1.4)





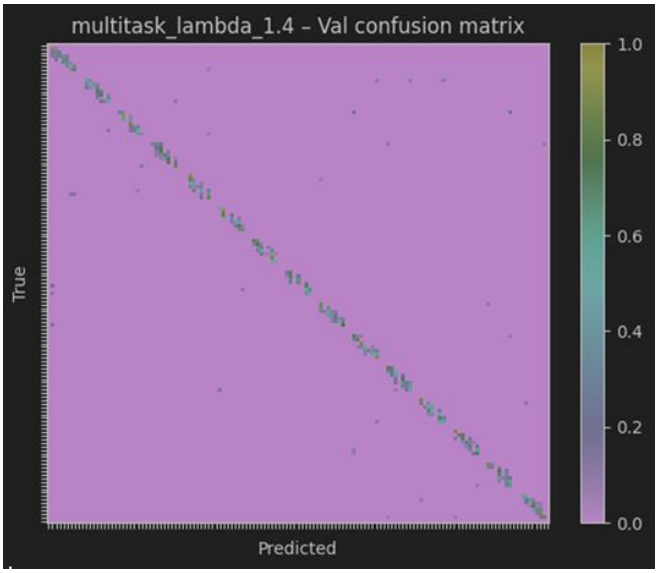
Confusion matrices and error analysis

Classification-only



The matrix suggests, that the classification is quite solid, there aren't many large error spots.

Multitask



The multi-purpose nature of the model did not hinder the quality of the classification, the majority od the labels are predicted quite correctly.

Multitask (λ = 1.4)

Achieved the best performance for:

- 5 circle + 5 triangle up
- 2 circle + 8 triangle up
- 5 square + 5 triangle right
- 3 square + 7 triangle down
- 8 triangle down + 2 triangle left
- 2 circle + 8 triangle right
- 7 square + 3 triangle down

Classification-only

Achieved the best performance for:

- 2 square + 8 triangle right
- 2 circle + 8 triangle right
- 7 circle + 3 triangle left
- 8 square + 2 circle
- 3 circle + 7 triangle up
- 2 triangle up + 8 triangle left
- 2 square + 8 triangle down

Contrary to what I had been expecting, the top-classified classes were different.

Comparison

Multitask stopped early after 50 epochs, finished in 5m 14s.

Classification only stopped early after 66 epochs, finished in 7m 38s.

Regression only did not stop early, finished in 10m 13s

Overall, the **classification-only** and **multitask (λ_cnt = 1.4)** models achieved very similar classification performance - regression loss did **not hurt** the main classification task.

However, the training dynamics and runtime differ across settings. The multitask model reaches its early-stopping criterion after **50 epochs** in about **5m 14s**, while the classification-only model requires **66 epochs** and about **7m 38s** to converge, so the multitask approach provides a substantial increase in learning speed.
Regression-only model runs for the full 100 epochs, taking roughly **10m 13s**.
Combined with the confusion-matrix analysis (which shows similar patterns of best- and worst-classified configurations for both multitask and classification-only models), these results suggest that the multitask formulation is a strictly better trade-off: it provides strong regression performance at essentially no cost to classification accuracy and with even shorter training time.

Conclusion

The conducted experiment shows, that training a multi-purpose model instead of two separate ones may not only be much more time and cost efficient, but also provide the same or even better performance. This effect is in part achieved by natural prevention of overfitting to provided data – the model cannot focus on minimizing loss for only one task, since it has to efficiently serve two distinct purposes.