Week 12: Convex Hexagon Classification Using YOLO11n CNN and Transformer Architectures prosepects

Hatem Feckry (120220264) Seifeldeen Mohamed Galal (120220252)
Abdelrahman Ashraf (120220292) Ahmed Elsherbeny (120220271)

May 2, 2025

1 Overview

This week we tried to implement a CNN based classification for sets as **Empty**, with no convex hexagons, or **Non-empty**. We believe this would be a good proof of concept or a baseline that we can improve upon using a transformer architecture. We used a relatively small CNN model, YOLO11n, and were able to achieve 97.5% accuracy during validation and 99.3% in testing. This proved that even though the CNN model isn't the perfect match for the task, it still performed very well assuring us that a transformer with relative positional embedding has a true chance of surpassing our analytical approaches.

2 Motivation

Our long-term goal is to develop a **transformer-based model** for both set generation and enlargement, but we decided to start with a **CNN-based classifier** as a preliminary step. The idea is that if a CNN can handle this task effectively, it gives us confidence that a transformer—especially one that uses set input and relative embedding for a graph like analysis—would perform well, or better.

We are working with images that are entirely **black (blank)** except for a few **points of interest**, which could represent corners or vertices of geometric shapes. One of the key challenges we face is the **sparsity of information**: in a 100×100 image (10,000 pixels), we only have **16 points of interest**. That's a ratio of just 16/10,000, making the input extremely sparse and hard to learn from, again this would be solved with a transformer by only inputting the points of interest.

So, we designed a **simplified classification task** that aims to infer if the image contains an empty set or a non-empty one. This might mirror what a transformer encoder might do by reducing the input to a dense latent space.

The model's job is to classify the image as either:

- **Empty**: has no convex hexagons.
- **Non-empty**: contains at least one convex hexagon.

This task is intentionally minimal: it should be solvable in **O(1)** time in theory, based on global context, which makes it ideal as a **proxy for an encoder's behavior**.

By solving this classification task:

- The model **acts like an encoder**, reducing the image to a meaningful representation.
- We can test if the model can capture the abstract "existence" of structure (polygons), not just local pixel patterns.
- Later, we can extend this to **adding new points** to the same set—similar to how **diffusion models** gradually generate more structure over time.

3 Introduction

Our initial strategy for addressing this problem was to utilize an object detection approach, specifically leveraging the YOLO (You Only Look Once) architecture. Object detection models are designed to perform two tasks simultaneously: **localizing** objects within an image by predicting bounding boxes, and **classifying** the objects contained within those boxes. The YOLO loss function is a combination of the following components:

- **Localization loss**: typically based on Mean Squared Error (MSE), this penalizes inaccuracies in bounding box coordinates.
- **Confidence loss**: evaluates the model's ability to correctly predict whether an object is present in a given bounding box.
- **Classification loss**: often implemented as Cross-Entropy Loss, it penalizes incorrect class predictions for detected objects.

Despite YOLO's effectiveness in many domains, it proved unsuitable for our case. The points of interest in our images were **highly sparse** and had **low contrast**, making it difficult for the model to detect meaningful patterns or define accurate bounding boxes. The absence of clear visual anchors prevented the model from converging to a useful detection behavior.

Recognizing these limitations, we shifted to a **classification-based approach**. Unlike detection, image classification does not require object localization. Instead, it treats the entire image as a single entity and predicts a class label based on global features.

This approach is particularly well-suited for scenarios where visual cues are subtle and dispersed.

We worked with a dataset consisting of approximately **2400 images for training**, along with **400 and 700 images** used for validation and testing, respectively. Initially, we did not expect the model to perform well given the challenging nature of the data. However, the results were surprisingly strong.

Model training was conducted on **Google Colab**. We utilized GPU acceleration as long as free resources were available. Once our GPU runtime quota expired, we continued training on the CPU. Despite these limitations, the model maintained high performance.

For training, we used **Cross-Entropy Loss**, which is ideal for multi-class classification problems. The loss function measures the divergence between the predicted probability distribution and the true labels, placing a stronger penalty when the model is confidently incorrect. Given that our dataset is balanced and the classes are mutually exclusive, this loss function aligns well with our needs.

This change in strategy significantly improved model performance. Since the patterns in our images are **global and non-localized**, the classification model was better able to capture the relevant features without being constrained by the need for bounding boxes. Moreover, it reduced model complexity and improved generalization.

In summary:

- **Detection failed** due to its reliance on spatially dense, local features.
- **Classification succeeded** by leveraging global patterns and using an appropriate loss function for our data characteristics.

This strategic pivot was instrumental in achieving our final accuracy of **97.5%**, highlighting the importance of aligning the model architecture with the underlying nature of the data.

4 Loss Functions

4.1 Detection Loss (YOLO)

YOLO uses a compound loss function that combines localization, confidence, and classification components:

$$\mathcal{L}\text{YOLO} = \lambda\text{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right]$$
$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2$$
$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum c \in \text{classes} (p_i(c) - \hat{p}_i(c))^2$$

- (x_i, y_i, w_i, h_i): ground truth bounding box coordinates.
- $\hat{x}_i, \hat{y}_i, \hat{w}_i, \hat{h}_i$: predicted coordinates.
- C_i: confidence score.
- $p_i(c)$: predicted probability for class c.
- $\mathbb{1}_{ij}^{\text{obj}}$: indicator if object exists in cell i, box j.
- $\lambda_{\text{coord}}, \lambda_{\text{noobj}}$: loss weights for localization and background.

This loss function assumes the presence of clearly localizable features, making it less effective in our case due to sparse, low-contrast image regions.

4.2 Classification Loss (Cross-Entropy)

For classification, we used the categorical cross-entropy loss, which is well-suited for multi-class problems with balanced classes:

$$\mathcal{L}\text{CE} = - \sum i = 1^N y_i \log(\hat{y}_i)$$

- y_i: one-hot encoded true label for class i.
- $\hat{y}_i$: predicted softmax probability for class i.
- N: number of classes.

This loss function measures the divergence between the predicted distribution and the actual label, making it ideal for our setup where each image belongs to exactly one class and no spatial localization is needed.

5 Results

The YOLO11n-based CNN classifier delivered unexpectedly strong performance on the convex-hexagon classification task. After training, the model achieved a validation accuracy of 97.5% and a test accuracy of 99.3%. Throughout training, both the train and validation losses steadily decreased, while top-1 accuracy climbed in tandem, indicating minimal overfitting (see Figure 1 in the Figures section).

The normalized confusion matrices (see Figure 2 in the Figures section) quantify this performance in detail. On the test set, the model attains 100% recall and precision for the 'Non-empty' class and 99% for the 'Empty' class, with only 1% false positives. Training-set performance is similarly high (99% empty, 96% non-empty), underscoring excellent generalization.

Figure 3 in the Figures section shows a sample training batch of 16 images, each 100×100 pixels with only 16 white points. Identifying a convex hexagon among such sparse data is non-trivial, yet the CNN reliably learns the pattern.

A qualitative check on validation data (see Figure 4 in the Figures section) shows perfect alignment between true labels and model predictions on a representative batch of 16 images, further demonstrating robust generalization.

6 Analysis

Our dataset presents two main challenges: extreme sparsity (only 16 active pixels out of 10,000) and a label determined by a combinatorial property (existence of any convex hexagon among $\binom{16}{6} = 8008$ subsets). Despite CNNs being designed for local spatial patterns, the YOLO11n architecture succeeded by leveraging deep receptive fields and hierarchical feature extraction. Early convolutional layers detect small point alignments; deeper layers aggregate these signals over the entire image via global pooling and fully-connected classification heads. This enabled the network to implicitly learn geometric heuristics—such as points lying on a convex hull—without explicit localization.

The model's near-perfect generalization to unseen point configurations indicates it learned true relational features rather than memorizing particular layouts. Translation invariance, inherent in convolution and pooling, allowed the CNN to recognize convex polygons regardless of their position. However, the CNN's inductive bias for locality means that capturing long-range interactions was an implicit consequence of depth and pooling layers, not an architectural guarantee. Moreover, the CNN lacks a native mechanism for permutation invariance over points; it treats the data as a fixed grid rather than a set of tokens.

Transformers, by contrast, employ self-attention to model *all* pairwise (and higher-order) interactions explicitly, making them naturally suited to set-based geometric reasoning. A transformer encoder could ingest the raw point coordinates (or learned embeddings) with relative positional encodings that preserve translation invariance, and directly learn to attend to the six points forming a convex polygon. This explicit modeling of relationships

would likely require fewer examples to learn the same abstract property and would scale more gracefully to larger point sets or higher polygon orders.

7 Conclusion

Repurposing YOLO11n as a pure classifier for the Happy Ending convex-hexagon problem produced a powerful baseline: 97.5% validation and 99.3% test accuracy on extremely sparse, combinatorial data. The CNN's success—despite its local inductive biases—demonstrates that deep architectures can learn abstract geometric relations from images.

These findings strongly motivate transitioning to Transformer-based architectures. Transformers' self-attention mechanism aligns perfectly with the set nature of the task, offering direct modeling of point-to-point interactions and true permutation invariance via relative positional encodings. We anticipate that a carefully designed transformer encoder will match or exceed the CNN's performance, generalize more efficiently to larger or more complex point sets, and provide interpretable attention patterns highlighting the convex hexagon's vertices.

Future work will implement and evaluate a transformer-based classifier on this task, extend the dataset to more points and higher polygon orders, and explore generative applications—such as constructing point sets with or without convex polygons—leveraging the rich representational power of attention-based models. This research paves the way for deep learning approaches to combinatorial geometry problems, uniting theoretical mathematics with modern neural architectures.

Figures

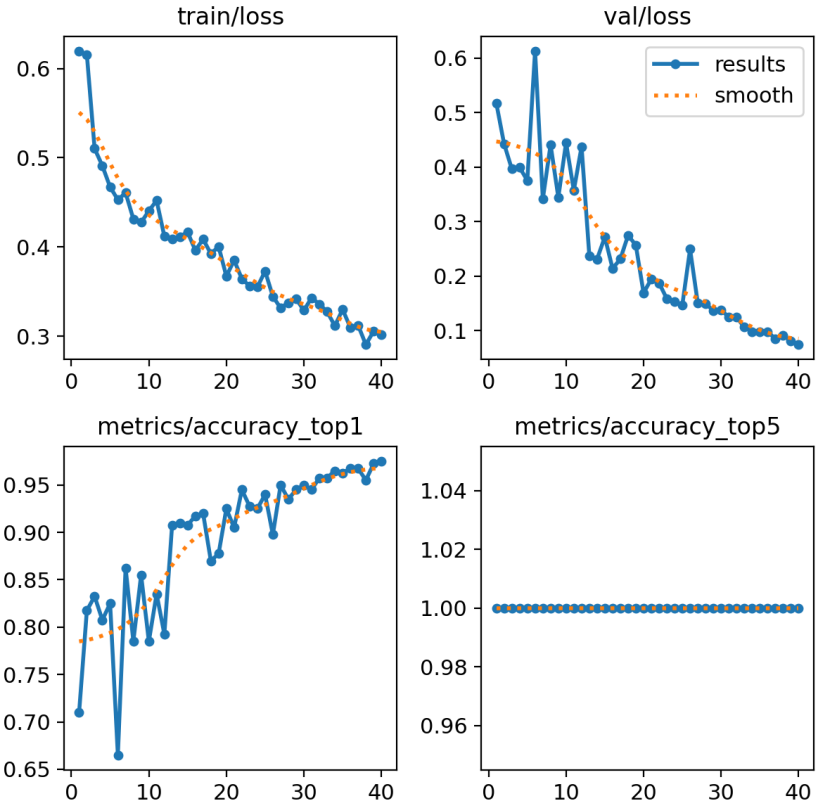
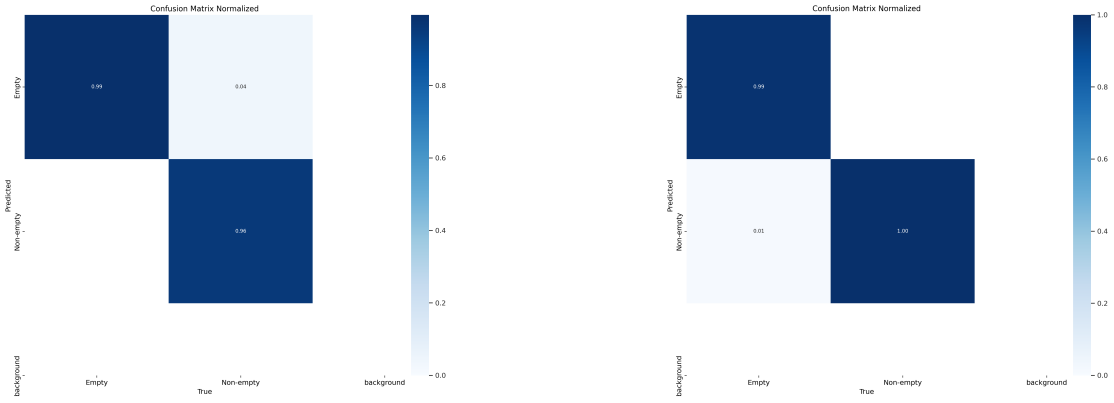


Figure 1: Training and validation loss (top) and top-1/top-5 accuracy (bottom) vs. epoch for the YOLO11n classifier. Validation performance closely follows training, reaching 97.5% top-1 accuracy without overfitting; top-5 accuracy remains 100%, trivial for 2-class system.



(a) Training set (b) Test set

Figure 2: Normalized confusion matrices for the CNN classifier. Both sets show very high true-positive and true-negative rates for ‘Empty’ and ‘Non-empty’ classes.

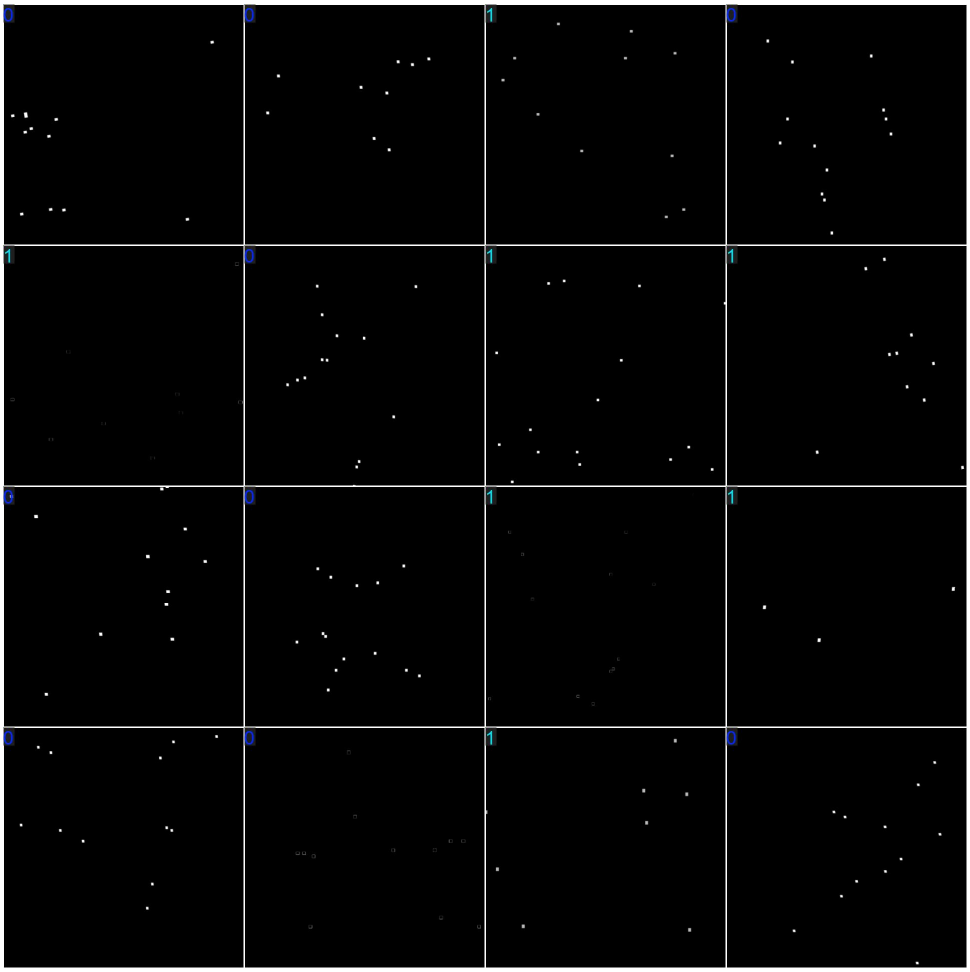
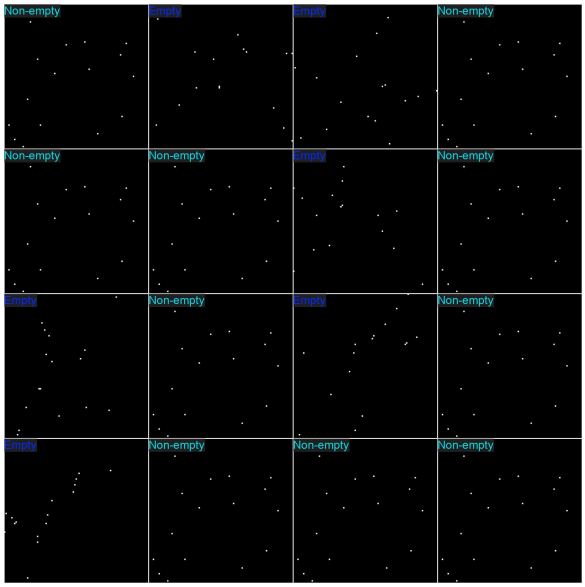
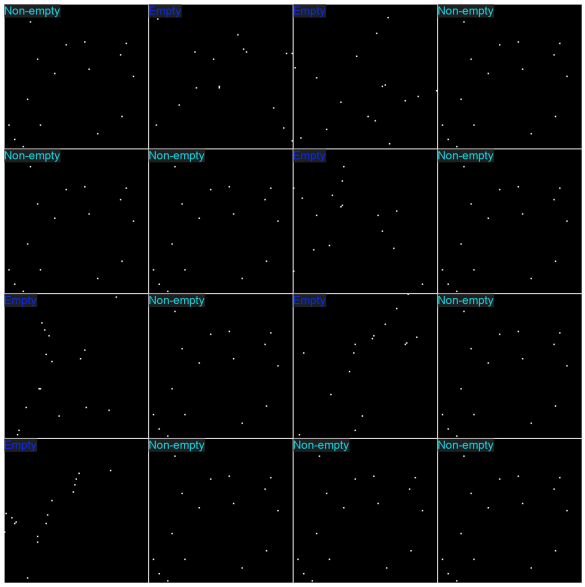


Figure 3: Example training batch: 16 point-set images with ground-truth labels (‘0’: Empty, ‘1’: Non-empty). Each image contains only 16 points among 10,000 pixels.



(a) Validation batch (ground-truth labels)



(b) Validation batch (model predictions)

Figure 4: Comparison of true labels vs. model predictions on a validation batch. All 16 images are classified correctly (‘Empty’ in blue, ‘Non-empty’ in green).