Bug or artistic feature? Scaling assessment and feedback of creative student-authored graphics programs

Miranda Li [*] **Cameron Mohne** [*]
Chris Piech [+]

Abstract

We experiment with machine learning methods for automatically assessing the complexity and creativity of CS1 graphics assignments. We aim to build a tool which can be quickly deployed by educators to provide feedback on student graphics assignments at scale. We have tried framing the prediction of the complexity of a graphics assignment output as a supervised few-shot learning problem as well as performing an unsupervised outlier detection approach to try to pick out creative outputs from the set of all student images.

We also hope to begin a discussion of how we might define and measure creativity in an educational context.

1. Introduction

With this direction in mind we also hope to open discussion on an important sub-question: *how can we distinguish intentionally creative student output from mistakes*? To the best of our knowledge this question has yet to be formally posed. We hypothesize that highly intentionally creative (artistic) outputs are some combination of *novel* and *complex*. Thus as a first step towards creating a generalizable auto-grader we focus on attempting to measure and predict the visual complexity of an output from the student's generated image.

2. Related Work

Auto-Grading Graphics Assignments: Notably, the task of auto-grading graphical assignments within the CS sphere has been explored before (Yan et al., 2019). While revolutionary, the overarching goal of this prior work was close-ended in nature with open-ended pathways. We instead explore methods to move beyond this task and assess a truly open-ended assignment with no ground-truth goal.

Few-Shot Learning: One key constraint to keep in mind due to scale is the amount of data accessible. Requiring too much data hinder users (i.e. teachers) (Sung et al., 2018) when using a generalized model. Few-Shot Learning (FSL) is one type of model architecture purposed for handling this issue. In FSL literature, a vital component of the FSL model's structure is the number of *ways* and *shots* the model uses to train. An N-way, K-shot FSL architecture has N unique classes for training with K labeled instances of each respective class. These examples make up the *support set* which is used to sample from and fine-tune the model to prepare it for various unseen cases from the *query set*. In this implementation we hope that with necessary tweaks the model adequately learns an embedding to classify novel cases which lie outside of any prior collected data as they come.

Transfer Learning: Recent FSL models sometimes utilize transfer learning as a backbone prior to fine-tuning towards a specific task. This methodology generates a metric embedding which is assumed to aid efficacy in generalization when given few examples: a conclusion that is in line with the scope and purpose of transfer learning overall (Medina et al., 2020).

Meta-Learning: While meta-learning techniques are not implemented in this paper, they serve as a next step. The concept of ”learning to learn” can help deepen FSL models to aid in query set generalization through (a learned) more-efficient use of the support set (Ren et al., 2018).

Unsupervised Outlier Detection: Another way to consider the posed problem is to ponder the average case of student flags. We then make an assumption that creativity and/or complexity results in an outlier with respect to other data points. In this assumption, we hope to learn latent features for what defines creativity or complexity across many examples if we correctly learn high-level semantic features (Cheng et al., 2021).

3. Dataset and Features

The flag assignment is a problem students are given for open-ended exploration of graphics during Code in Place (cip, 2023), a 6-week human-centered MOOC which had 9k students from its most recent iteration which our data is from. Our data is student work representing code written with the intention and purpose of generating a flag. There were no hard requirements on what region the flag should represent. We obtained the student code and corresponding images from the Code in Place database. We have 5081 images each with corresponding code. Each example is denoted by a student's unique identifier. We labelled all 5081 images with which country's flag, if any, seemed to be emulated. We labelled 1316 images with a value from 0 - 10 based on how complex the image appeared to be.

In some of our experiments we turned our complexity labels into coarser bins ”low”, ”medium”, and ”high”. We did this binning in two different ways – the first which we will refer to as the ”low_02” scheme was binned as follows:

- Labels 0, 1, and 2 were assigned to the bin ”low_02”.

- Labels 3, 4, 5, 6 were assigned to the bin ”medium_36”.

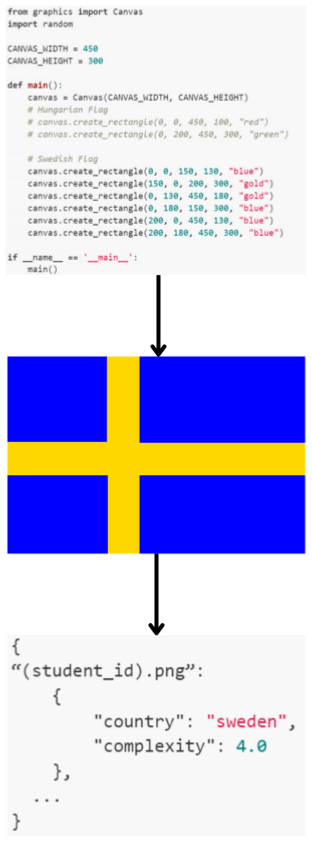
- Labels 7, 8, 9, 10 were assigned to the bin ”high_7”.

And the ”low_13” scheme was binned as follows:

- Labels 1, 2, and 3 were assigned to the bin ”low_13”. Note that here we exclude all images with label 0, which are all blank.

- Labels 4, 5, 6 were assigned to the bin ”medium_46”.

- Labels 7, 8, 9, 10 were assigned to the bin ”high_7”.



```
from graphics import Canvas
import random

CANVAS_WIDTH = 450
CANVAS_HEIGHT = 300

def main():
    canvas = Canvas(CANVAS_WIDTH, CANVAS_HEIGHT)
    # Hungarian Flag
    # canvas.create_rectangle(0, 0, 450, 100, "red")
    # canvas.create_rectangle(0, 200, 450, 300, "green")

    # Swedish Flag
    canvas.create_rectangle(0, 0, 150, 130, "blue")
    canvas.create_rectangle(150, 0, 200, 300, "gold")
    canvas.create_rectangle(0, 130, 450, 180, "gold")
    canvas.create_rectangle(0, 180, 150, 300, "blue")
    canvas.create_rectangle(200, 0, 450, 130, "blue")
    canvas.create_rectangle(200, 180, 450, 300, "blue")

if __name__ == '__main__':
    main()
```

```
{
"(student_id).png":
    {
        "country": "sweden",
        "complexity": 4.0
    },
  ...
}
```

Figure 1. Data retrieval pipeline. From top to bottom we have: the student code, the corresponding flag, the labelled data entry.

4. Methods

4.1. HDBScan

As a baseline we implemented unsupervised clustering over the flag images to see if simple algorithms could recognize any meaningful structure to the data (ideally clustering flags of the same country or similar complexity near to each other); we examined the cluster outputs by hand. For this we used HDBScan (Malzer & Baum, 2019), a hierarchical clustering algorithm which attempts to infer the number of clusters by pruning a tree of generated clusters until all remaining clusters are stable. We preferred this method because most other methods require an a priori specification of how many clusters are in the data, which in this case we did not know — though there are 193 national flags in the world, not all of them were represented in the dataset, and some students created flags which were not national flags (intentionally or not).

4.2. K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a nonparametric supervised learning algorithm which is a common baseline in supervised computer vision tasks. We used KNN for both classification and regression on complexity values as a baseline.

When a prediction is to be made for an unseen sample, KNN examines the k closest points in the dataset to that point and returns the most common label (in the case of classification) or the average label (in the case of regression) as the prediction on that sample. k is a parameter which we specify.

4.3. Prototypical Networks

We utilized a Prototypical Network (ProtoNet) model, a few-shot learning approach, as one of our methodologies. We went with this approach because we ultimately hope to build a tool which can be easily applied by educators to their graphics assignments with minimal overhead (e.g. easier than requiring them to label a large dataset and train their own model). ProtoNet intends to learn a metric space in which classification can be performed by performing a distance computation from *prototype representations* of the various classes (Snell et al., 2017).

Our implementation closely follows (Snell et al., 2017) where we take a support set $S \in \mathbb{R}^{N \times D}$ and train our model by building *prototypes* using S_k — the subset of the N D-dimensional examples having class k. The prototypes, c_k, are then built on the binned complexities by taking an embedding function with learnable parameters ϕ, $f_\phi : \mathbb{R}^D \to \mathbb{R}^M$, and finding the mean vector of the respective classes:

$$c_k = \frac{1}{|S_k|} \sum_{(x_i),(y_i) \in S_k} f_\phi(x_i)$$

This result is then used alongside a distance function d to create a distribution over the classes based on the softmax over distance to each prototype in the embedding space:

$$p_\phi(y = k|x) = \frac{\exp(-d(f_\phi(x), c_k))}{\sum_{k'} \exp(-d(f_\phi(x), c_{k'}))}$$

We take this distribution and minimize its negative log-probability as our cost function

$$J(\phi) = -\log p_\phi(y = k|x)$$

with regard to the true classes k via SGD.

4.4. Outlier Detection

Converse to the other non-baseline methodologies, Outlier Detection is a completely unsupervised learning algorithm. The specific architecture we used was that of a single-stack *autoencoder* resembling the model of (Wan et al., 2019). The overarching pipeline of this model takes ground truth images and proceeds to reconstruct the image via an encoder

$$y = a(Wx + b)$$

(where a is an activation function); and a decoder

$$z = a(W'y + b')$$

In this case, y is an encoded representation of x, and z is a reconstructed version of x. Notably, (Wan et al., 2019)'s architecture uses the sigmoid activation function for both the encoder and decoder activation functions. Ours instead uses the ReLU activation function for all of the encoding layers and most of the decoding layers. We retain the sigmoid only in the last step of reconstruction. This choice was informed by: gradients being less likely to vanish for deeper encodings, overall quicker convergence (Krizhevsky et al., 2012), and hopes to learn more distinct and disentangled latent features through sparsity. Thus, we reserve usage of the sigmoid only for the purposes of a smoother output in the reconstruction process. We then ultimately take this reconstruction and use it to optimize reconstruction error. We do this by training and updating perceived reconstructions using the MSE a reconstruction has from its ground truth as our loss function.

This process is depicted in Figure 2.

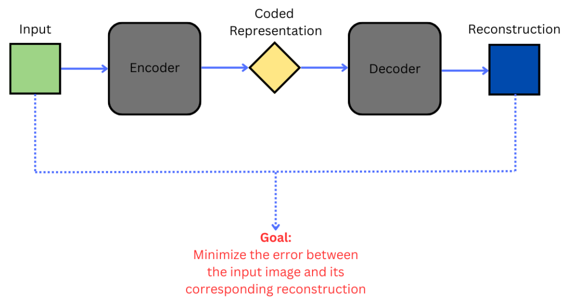

Input
Encoder
Coded Representation
Decoder
Reconstruction
Goal:
Minimize the error between the input image and its corresponding reconstruction

Figure 2. Process of reconstruction and the corresponding training goal

With this approach, the outlier detector learns how to extract features from our data which would lead to the ability to classify outliers through the severity of difference an image has in regard to the perceived reconstruction in the mind of the model. In doing so, this also provides interpretable results as to where and what is causing outlier classification. This explainability will be fundamental when we discuss creativity else it will be impossible to deduce how novelty was perceived in the scope of our classification process.

5. Experiments/Results/Discussion

5.1. Baseline results

5.1.1. HDBScan

We ran a few different configurations of HDBScan, all of which produced very unsatisfying results upon visual examination. We tried running HDBScan with:

- a minimum cluster size of 5 on the embeddings (length 1000) directly
- a minimum cluster size of 5 on the embeddings after reducing their dimension to 50 using PCA — we chose 50 because it seems to explain a reasonable amount of the variance (from a cumulative explained variance plot) while being a significantly smaller number of dimensions than the full embedding

5.1.2. KNN

We ran KNN with varying values of k. We treated our complexity labels as a classification as well as a regression task.

On classification (treating the complexity scores 0 - 10 as eleven different bins) we were able to achieve a maximum accuracy of 37.08% with $k = 32$ after trying KNN with all values of k from 1 to 40.

On regression (treating the complexity score as continuous) we were able to achieve a minimum MSE of 3.45 with $k = 35$ after trying KNN with all values of k from 1 to 40.

Under the ”low_02” scheme we achieved 69.00% accuracy (notably on the coarser bins ”low_02”, ”med_36” and ”high_7”) with $k = 7$ after trying KNN with all values of k from 1 to 40.

5.2. ProtoNet results

Before running ProtoNet we binned our complexity values into ”low”, ”medium” and ”high” bins since some of our classes 0-10 had extremely few examples. To understand the effect that binning into these more coarse labels would have, we attempted ProtoNet twice; we will describe below how we performed the binning and what the results were. Note that ProtoNet is a classification algorithm, but since we are ultimately hoping to treat complexity as continuous, we also report mean squared error (which we calculate by assuming classification to a certain bin represents a prediction of the average complexity value of that bin) with regard to the 0-10 complexity labels in addition to accuracy with regard to the bins.

We ran ProtoNet with the following parameters:

- 2 classes per task (N_WAY)
- 5 images per class in the support set (N_SHOT)
- 10 images per class in the query set (N_QUERY)

Under the ”low_02” scheme our evaluation of 100 tasks achieved 73.05% accuracy and MSE of 8.60 without finetuning ProtoNet, and 73.45% accuracy and MSE of 8.05 after finetuning ProtoNet for 40k iterations. The confusion matrix of the finetuned model is presented in Figure 3.

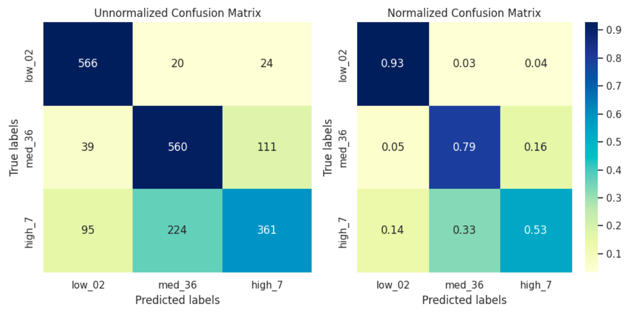


Figure 3. Confusion matrices for ProtoNet with labels ”low_02”, ”medium_36”, ”high_7”

We make two observations: first, that finetuning ProtoNet doesn’t seem to really improve the accuracy, and second, ProtoNet does not do much better than our KNN baseline on the same dataset which has an accuracy of 68.9%. It remains to be investigated why finetuning ProtoNet does not seem to improve the accuracy.

Under the ”low_13” scheme our evaluation of 100 tasks achieved 72.25% accuracy and MSE of 6.40 without finetuning ProtoNet, and 68.85% accuracy and MSE of 6.61 after finetuning ProtoNet for 40k iterations. The confusion matrix of the finetuned model is presented in Figure 4.

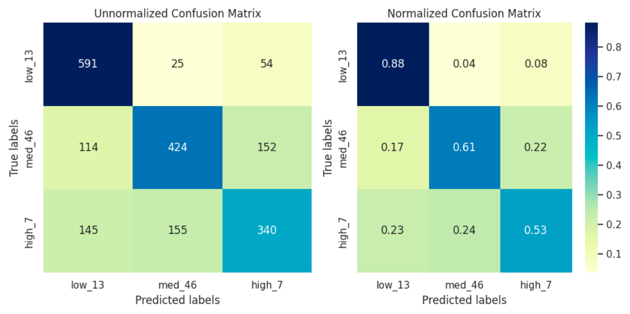


Figure 4. Confusion matrices for ProtoNet with labels ”low_13”, ”medium_46”, ”high_7”

Similarly we observe here that finetuning does not have a positive effect, and that we once again do not improve from the baseline We also note that since our results seem to differ between our two binning strategies, the choice of bins has an important effect on the efficacy of the final model.

Across both experiments we observe from the confusion matrix that the low bin is always the easiest to correctly classify, and the highest is always the most difficult. Also, the prediction seems to more frequently be a lower bin than the true label, since the lower left portion of the matrices (under the diagonal) seem to be generally more likely than the upper right. We hypothesize that this could be due to class imbalance, where since we have a disproportionately large number of samples in the lowest bins the model is able to learn a better representation of those samples, but further exploration is required.

5.3. Outlier detection results

After training and testing the model on the same dataset as other implementations, we found that the model had a fair interpretation of visual complexity; but poor discernment of countries versus edge cases overall. This is best seen in outlier classification with varying selections of thresholds.

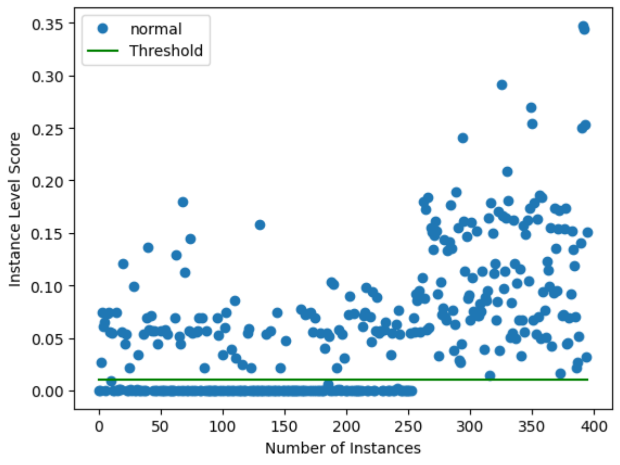


Figure 5. Example threshold graph with threshold = 0.01

As we can see in Figure 5, the bound for the threshold dictates which *instance scores* (how much an image deviates from the dataset) are considered abnormal. This directly impacts the sets of perceived novelty and mundanity and which images can thus be deemed complex or simple respectively. The performance derived from these different thresholds provide valuable insight into what the model finds novel upon reconstruction. Please note that *novelty* in this context refers to the metric of belief that a given case is an outlier within the dataset. Novelty is related to image complexity as we will find, but they are not equivalents.

To validate the first point—that the model has a fair interpretation of visual complexity—we investigate Figure 6.

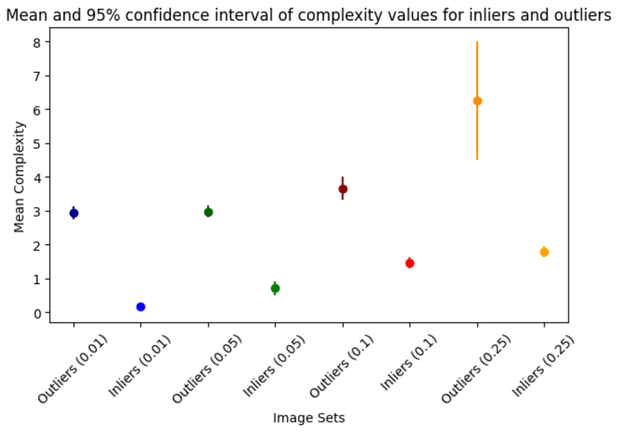


Figure 6. Threshold impact on mean complexity of the outlier and inlier image sets

From Figure 6 it is visually apparent that as we increase the threshold for novelty, complexity in both sets indeed rise. This is expected as we naturally assume novel cases to be more complex on average. Increasingly difficult restrictions on novelty would thus lead us to assume that the average complexity will increase given our prior assumption—which holds true. However, the new influx of previously novel cases (for a prior threshold) means that the set of inliers now absorbs cases generally above the mean complexity of what it was previously; thereby increasing it. Overall, we find this result to conclude that visual complexity is a valid derivative in regards to outliers. This isn't without faults, though.

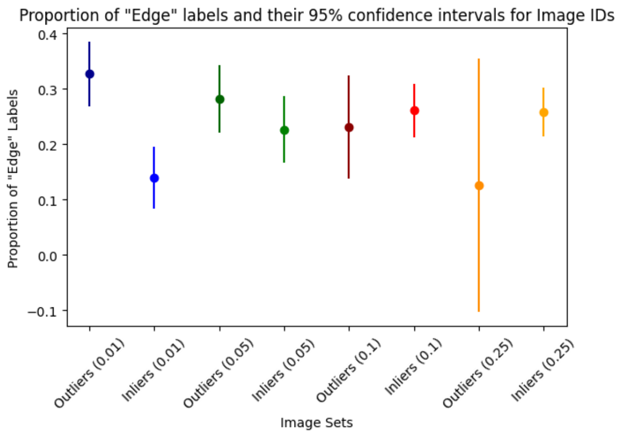


Figure 7. Threshold impact on the proportion of edge cases in outlier and inlier image sets

We see in Figure 7 that outside of when the threshold is set to 0.01, novelty thresholds have little impact on the proportion of edge cases in outlier/inlier sets of images. This is vital knowledge given that edge cases are cases where a country is not classified; a result implying that we fail to capture a true classification on the countries' flags. It is entirely possible for this to be due to the type of outliers present in the dataset—an explanation hypothesized prior in the paper for the ProtoNet methodology. Overall, more research needs to be done.

Adjacently, verification was done to ensure that this model is indeed trying to perceive and classify novelty. To do this, we investigated random samples of both outlier and inlier sets of data for each given threshold. We compared the ground truth flags to heat maps of what the model perceives as novelty—that is, parts of the image which diverges significantly from the reconstruction the model was expecting. Overall, we find that as the threshold increases, there is higher semantic meaning within the heat maps. See appendix for these findings.

Lastly are two key weaknesses in this model as well as why we decided to include it. Namely, this implementation in its current state fails to adhere to scalability and ease of generalizability.

This is counteracted, though, by the belief that this work may instead spark new conversation regarding interpretations in defining a metric for creativity within open-ended assignments. It also has potential to serve as a baseline consideration for adequate assessment moving forward.

6. Conclusion / future work

This work represents a first step towards creating a generalizable auto-grader for open-ended, creative assignments at scale.

We revisit our hypothesis that intentional creativity is captured by both novelty and complexity of an output.

From unsupervised outlier detection we found that complexity could be captured well (with the caveat that this is certainly affected by the fact that the majority of our dataset were images of low complexity – it remains to be seen if this trend is as clear for datasets where the images are more similar in complexity or higher in complexity overall), but the difference between a mistake and an intentional choice was not captured particularly well from our qualitative observations of the results.

We also tried few-shot learning on our hand-labeled complexity labels to see if we could create a system which is more easily deployable (i.e. requires less data labelling and less training time), but unfortunately we were not able to do much better than the baseline KNN approach. We will continue to investigate how we might improve our methods.

We will continue to work on this and try a few-shot regression model instead of a few-shot classification model. We are also investigating implementations of meta-learning.

Simultaneously, we are looking into how, philosophically, creativity should be not only be defined, but also measured. We hope to explore a rich qualitative discussion about what constitutes a high-quality creative student assignment as we move forward with this work.

Contributions

Miranda Li (mirandal@stanford.edu)

Responsible for retrieval of raw data (code); half of labelled images; implementations of HDBScan, KNN, and ProtoNet FSL models.

Responsible for half of the write-up including: majority of introduction; majority of dataset and features; parts of methodology; HBDScan, KNN, and ProtoNet results; conclusion.

Cameron Mohne (mohnec1@stanford.edu)

Responsible for: code-to-image pipeline; image labelling software; half of labelled images; implementation of outlier detection and an omitted alternative FSL architecture.

Responsible for half of the write-up including: formatting; parts of introduction; related work; parts of dataset and features; parts of methodology; outlier detection results.

Chris Piech (cpiech@stanford.edu)

Provided the following: project ideas; means of access to the raw data (code) used; project direction; revision suggestions; and unwavering morale boost.

Appendix

A. Outlier Detection

The next two pages are dedicated towards outlier detection. In these two pages, we have the 4 thresholds, $[0.01, 0.05, 0.1, 0.25]$, organized into columns.

The top graph in the column showcases the threshold in relation to instance level scores.

The middle graph in the column showcases the mapping from the inlier images to the corresponding heat map for the given threshold.

The bottom graph in the column showcases the mapping from the outlier images to the corresponding heat map for the given threshold.

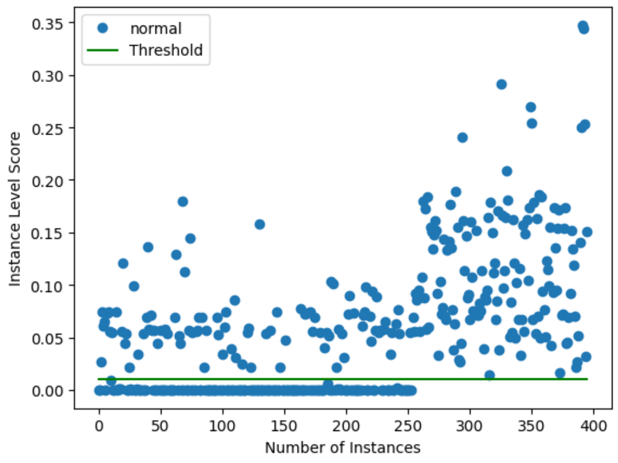


Figure 8. Threshold graph with threshold = 0.01

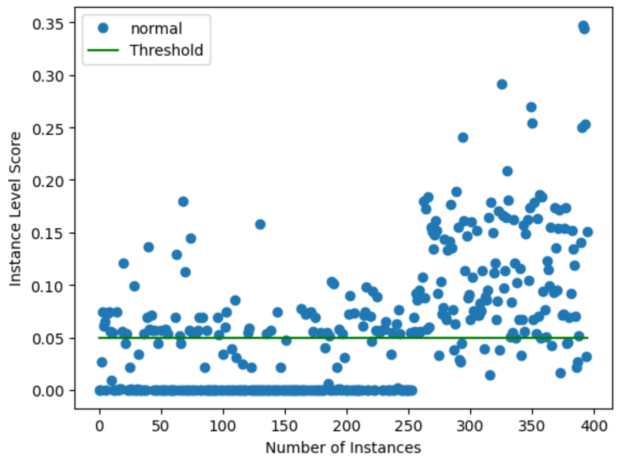


Figure 11. Threshold graph with threshold = 0.05

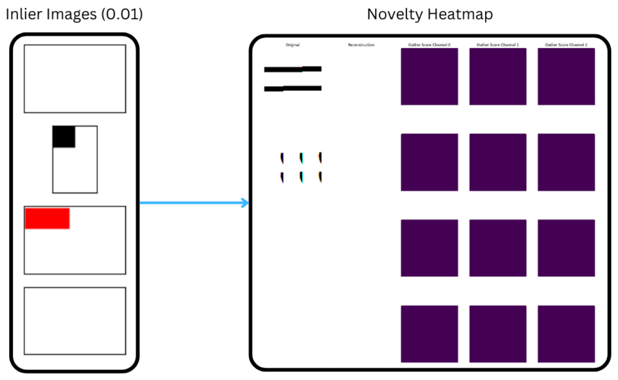


Figure 9. Mapping of inlier samples to their heat maps when threshold = 0.01

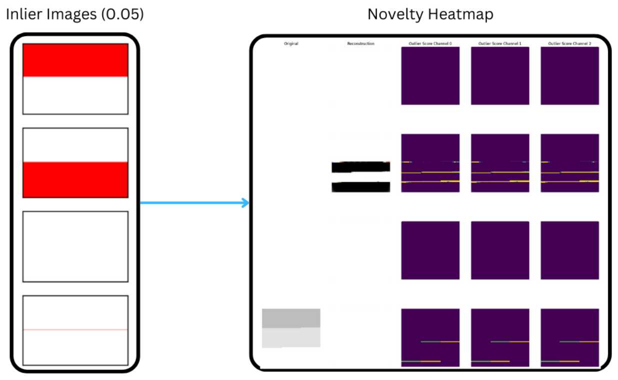


Figure 12. Mapping of inlier samples to their heat maps when threshold = 0.05

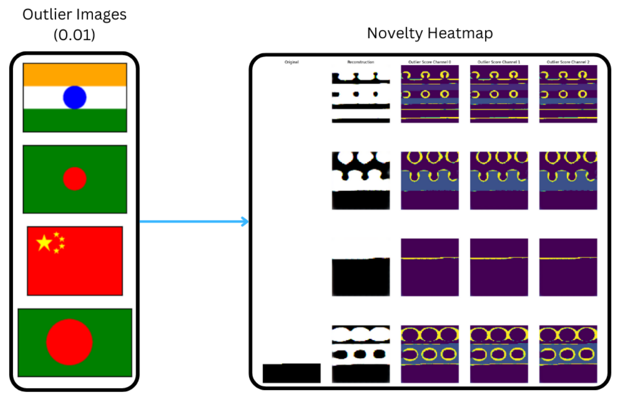


Figure 10. Mapping of outlier samples to their heat maps when threshold = 0.01

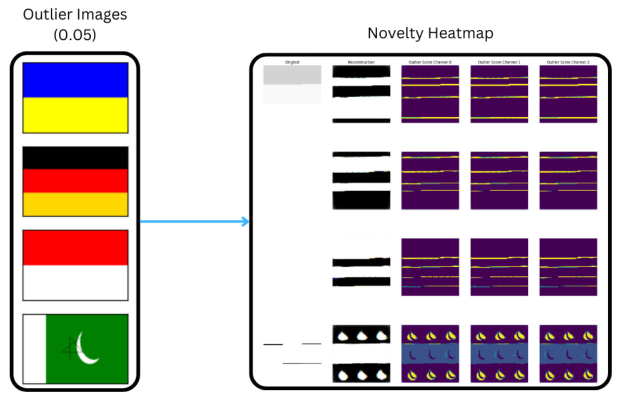


Figure 13. Mapping of outlier samples to their heat maps when threshold = 0.05

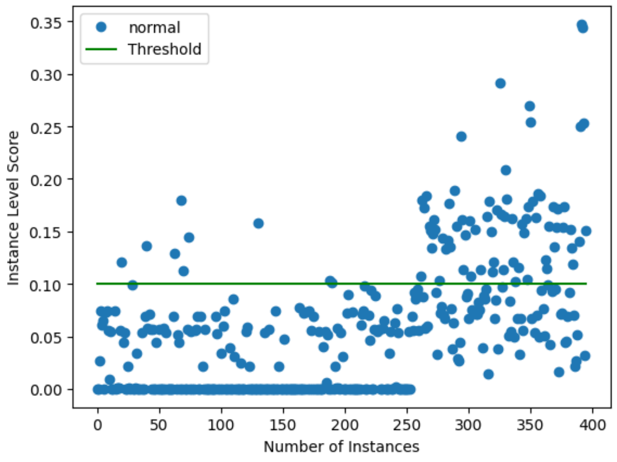


Figure 14. Threshold graph with threshold = 0.1

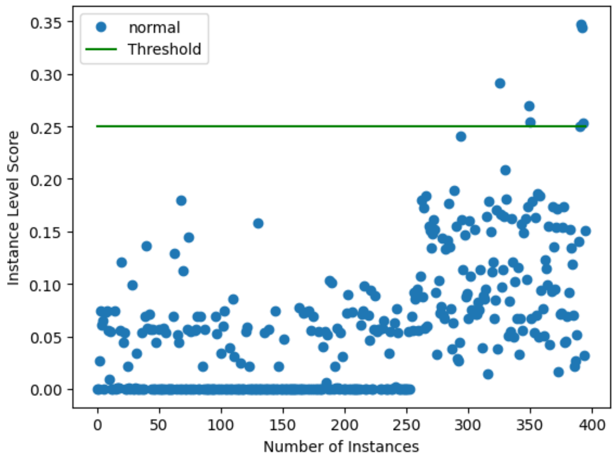


Figure 17. Threshold graph with threshold = 0.25

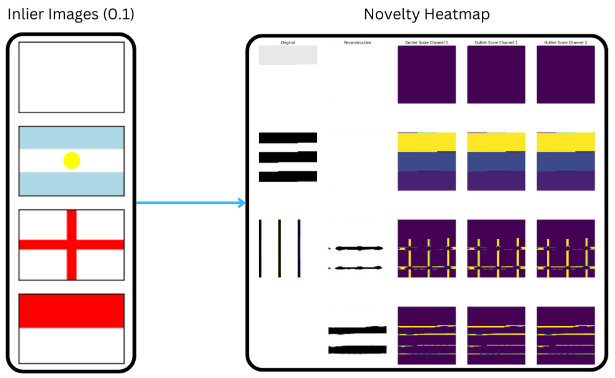


Figure 15. Mapping of inlier samples to their heat maps when threshold = 0.1

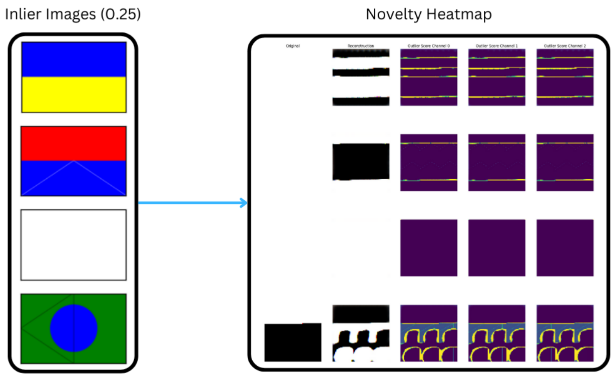


Figure 18. Mapping of inlier samples to their heat maps when threshold = 0.25

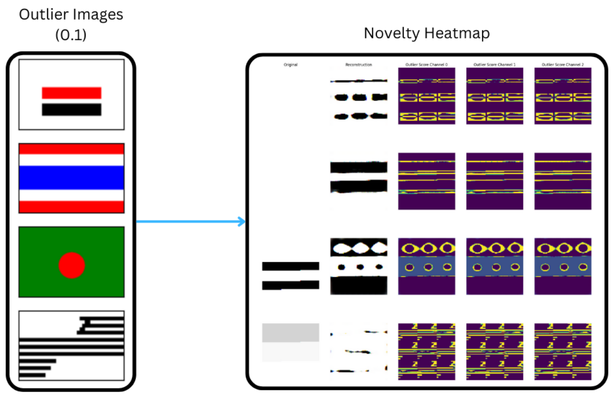


Figure 16. Mapping of outlier samples to their heat maps when threshold = 0.1

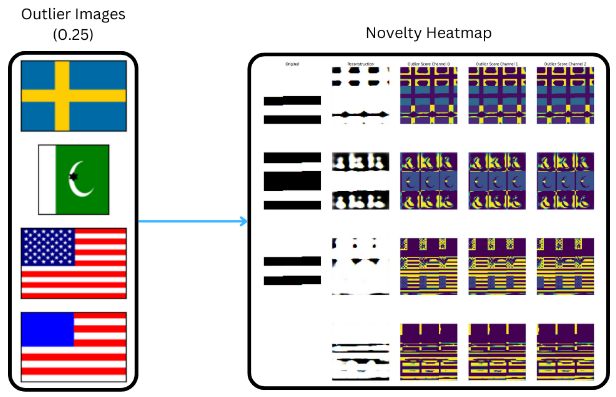


Figure 19. Mapping of outlier samples to their heat maps when threshold = 0.25

References

Code In Place, December 2023. URL https://codeinplace.stanford.edu. [Online; accessed 6. Dec. 2023].

Cheng, Z., Zhu, E., Wang, S., Zhang, P., and Li, W. Unsupervised outlier detection via transformation invariant autoencoder. *IEEE Access*, 9:43991–44002, 2021.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Malzer, C. and Baum, M. A hybrid approach to hierarchical density-based cluster selection. *CoRR*, abs/1911.02282, 2019. URL http://arxiv.org/abs/1911.02282.

Medina, C., Devos, A., and Grossglauser, M. Self-supervised prototypical transfer learning for few-shot classification. *arXiv preprint arXiv:2006.11325*, 2020.

Ren, M., Triantafillou, E., Ravi, S., Snell, J., Swersky, K., Tenenbaum, J. B., Larochelle, H., and Zemel, R. S. Meta-learning for semi-supervised few-shot classification. *arXiv preprint arXiv:1803.00676*, 2018.

Snell, J., Swersky, K., and Zemel, R. S. Prototypical networks for few-shot learning. *CoRR*, abs/1703.05175, 2017. URL http://arxiv.org/abs/1703.05175.

Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P. H., and Hospedales, T. M. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Wan, F., Guo, G., Zhang, C., Guo, Q., and Liu, J. Outlier detection for monitoring data using stacked autoencoder. *IEEE Access*, 7:173827–173837, 2019. doi: 10.1109/ACCESS.2019.2956494.

Yan, L., McKeown, N., and Piech, C. The pyramidsnapshot challenge: Understanding student process from visual output of programs. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pp. 119–125, 2019.