Research and implementation of multi-dataset training for image classification with discrepant taxonomies

A master thesis in the field of computer science

by

Björn Buschkämper

1st supervisor: Dr. Petra Bevandic
2nd supervisor: M.Sc. Riza Velioglu

Submitted in the research group “HammerLab”
of the faculty of technology for the degree of
Master of Science
at
Universität Bielefeld

July 30, 2025

Abstract

Scientific documents often use LaTeX for typesetting. While numerous packages and templates exist, it makes sense to create a new one. Just because.

Contents

I Introduction

2 Methodology

Our main goal is to create a universal taxonomy that connects multiple image classification datasets. This taxonomy maps every dataset class to a universal class, which allows us to analyse the relationships and shared concepts between datasets.

In the end, our taxonomy will allow us to train models that can classify images from multiple datasets at once, building a robust and flexible system that can quickly adapt to new domains.

2.1 Formal Definitions

To formalise our algorithm for building a universal taxonomy, we first need to define some terms:

- **Dataset** D: A collection of images and labels written as $D = \{(x_1, c_1), (x_2, c_2), \dots, (x_n, c_n)\}$, where x_i is an image and c_i is its label. Since we are dealing with multiple datasets, we number them as $D_i = \{(x_1^i, c_1^i), (x_2^i, c_2^i), \dots, (x_n^i, c_n^i)\}$, where D_i is the dataset D with index i. In the same way, we denote the set of all classes in a dataset as $C_i = \{c_1^i, c_2^i, \dots, c_k^i\}$.

- **Model** m: A neural network trained on a dataset D_I which maps an image $x \in X$ to a class $c_i^I \in C_I$, denoted as $m_I : X \mapsto C_I$.

- **Domain**: Since both models and classes are dataset-specific, we define the term **domain** as the dataset D_i and its classes C_i that we are working with.

- **Universal Classes**: Our universal taxonomy will contain a set of classes that are not specific to any dataset. We denote these classes as $C_U = \{c_1^U, c_2^U, \dots, c_k^U\}$. A universal class is a concept represented by a set of domain classes that share similar characteristics. We therefore define a function $\text{classes} : C_U \mapsto \mathcal{P}(C)$, where $\mathcal{P}(C)$ is the power set of C, to represent the set of domain classes that belong to a universal class.

- **Graph**: We represent our taxonomy as a directed graph $G = (V, E)$, where V is a set of vertices and E is a set of edges. Each vertex v_i represents a single class or universal class, which we define with $\text{class} : V \mapsto C$. Every edge e_{ij} between two vertices v_i and v_j indicates a relationship $\text{class}(v_i) \rightarrow \text{class}(v_j)$.

- **Probability**: Every edge e_{ij} has a probability associated with it, which indicates the likelihood of classifying an image from class $\text{class}(v_i)$ as class $\text{class}(v_j)$. We denote this as a function $\text{probability} : E \mapsto [0,1]$.

2.2 Cross-Domain Graph Generation

Before building our universal taxonomy, we need to construct our initial graph that captures the relationships between classes across different domains:

1. **Foreign predictions**: For each dataset with its corresponding model, we run the model on all images from all other datasets. This gives us a set of predictions $P_{ab} = \{(x_i^a, c_j^b)\}$, where x_i^a is an image from dataset D_a and c_j^b is the class predicted by model m_b for that image.

2. **Prediction probabilities**: We count the number of times each class c_i^a was predicted as a foreign-domain class c_j^b. We denote this count in a matrix $M_{ab} \in \mathbb{N}^{+|C_a|\times|C_b|}$, where $M_{ab}(i,j)$ is the number of times class c_i^a was predicted as class c_j^b. We then divide each entry in the matrix by its row sum to get the probability of classifying an image from class c_i^a as class c_j^b:

$$P_{ab}(i,j) = \frac{M_{ab}(i,j)}{\sum_{k=1}^{|C_a|} M_{ab}(i,k)}$$

   This gives us a matrix $P_{ab} \in [0,1]^{|C_a|\times|C_b|}$, where $P_{ab}(i,j)$ is the probability of classifying an image from class c_i^a as class c_j^b.

3. **Graph construction**: We now create a directed graph that represents the relationships between classes and datasets by iterating over every dataset D_a with every dataset D_b where $a \neq b$ for cross-predictions:

   a) We want to evaluate different methods for selecting the most relevant relationships, so we formalise a function $\text{select\_relationships}(P_{ab}) : [0,1]^{|C_a|\times|C_b|} \mapsto \mathcal{P}(\mathbb{N}^2)$ that selects a set of relationships from the probability matrix P_{ab}.

   b) For every $(i,j) \in \text{select\_relationships}(P_{ab})$:

      i. We create the vertices v_k and v_l for classes c_i^a and c_j^b respectively if they do not already exist and add them to the graph (otherwise we find the existing vertices for these classes as v_k and v_l).

      ii. We create an edge e_{kl} between the vertices v_k and v_l and add it to the graph.

      iii. We define $\text{probability}(e_{kl}) = P_{ab}(i,j)$.

2.2.1 Selecting Relationships

To now filter relationships from the probability matrix, we define a range of different methods and later evaluate their performance.

Our main challenges are:

- **Unknown number of shared concepts**: We don't know how many concepts two classes from different domains share, so we do not know how high the probability of a relationship should be.
- **Noisy predictions**: A low model accuracy can severely impact the relationship predictions, since - depending on the number of foreign classes that share concepts with the class - the target probabilities can be very low, making even a small number of wrong predictions a huge obstacle.
- **Unbalanced datasets**: Some datasets might have more images for a class than others, which can lead to skewed probabilities. This can be mitigated by preprocessing the datasets to balance the number of images per class, but our goal is to create a methodology that can be applied to any dataset without specific requirements on the datasets.

Naive Thresholding

The most straightforward method is to apply a fixed threshold to the probabilities:

$$\text{select\_relationships}(P_{ab}) = \{(i, j) \mid P_{ab}(i, j) \geq t\}$$

where t is a threshold value between 0 and 1.

Most Common Foreign Predictions

As in the paper that provided the ground work for our methodology [**bevandic_automatic_2022**], we can also select the single most common foreign prediction for each class:

$$\text{select\_relationships}(P_{ab}) = \{(i, j) \mid j = \text{argmax}_{j'} P_{ab}(i, j')\}$$

Density Thresholding

Another approach is to use the least amount of relationships whose summed probabilities cover a certain percentage of the total probability mass. This can be done by sorting the probabilities in descending order and then selecting the smallest set of relationships that covers at least p percent of the total probability mass:

1. We define $R = \emptyset$ as the set of relationships to select.
2. For every $i \in \{1, \dots, |C_a|\}$:
   a) Let X_i be the list of all probabilities in row i of P_{ab} sorted in descending order.
   b) We find the smallest k such that $\sum_{j=1}^{k} X_i(j) \geq p$.
   c) We add the first k relationships of the sorted list X_i to R.
3. We return R for the function $\text{select\_relationships}(P_{ab})$.

Relationship Hypothesis

Let us naively assume that every relationship between two classes is based on a single shared concept. In this case, the probability of every outgoing edge from a class c_i^a should be roughly equal.

We can therefore hypothesise the probability distribution based on the number of relationships and compare this hypothesis against the actual probabilities in the matrix P_{ab}:

1. We define $R = \emptyset$ as the set of relationships to select.
2. For every $i \in \{1, \dots, |C_a|\}$:
   a) Let X_i be the list of all probabilities in row i of P_{ab} sorted in descending order.
   b) We find the $k \in \{1, \dots, n\}$ such that we minimise:
   $$\sum_{j=1}^{k} \left| X_i(j) - \frac{1}{k} \right| + \sum_{j=k+1}^{|C_b|} X_i(j)$$
   c) We add the first k relationships of the sorted list X_i to R.
3. We return R for the function $\text{select\_relationships}(P_{ab})$.

In this equation, n is the upper bound of the number of relationships for a class c_i^a that we want to test against.

2.3 Synthetic Taxonomy Generation

2.3.1 The Need for a Controlled Ground Truth

To evaluate our taxonomy generation methods, we need a reliable ground truth with known relationships between datasets. This presents a challenge, as most existing image classification datasets lack clear inter-dataset relationships:

- **ImageNet** [**deng_imagenet_2009**, **russakovsky_imagenet_2015**] uses WordNet's [**fellbaum_wordnet_1998**] hierarchical structure to organize classes. However, this strict hierarchy doesn't match our use case where we need to connect datasets with different class structures and partial overlaps.
- **Open Images** [**kuznetsova_open_2020**] contains approximately 9 million images with multiple labels per image generated by Google's Cloud Vision API[1]. This multi-label approach makes it difficult to determine a single class for each image, which is required for our evaluation. Additionally, since most labels were automatically generated, it doesn't provide the verified ground truth we need.
- **iNaturalist** [**horn_inaturalist_2018**] offers a detailed taxonomy of plant and animal species, but its domain-specific nature makes it unsuitable for developing a general-purpose evaluation framework.

2.3.2 Our Approach: Building Synthetic Datasets

Instead of relying on existing taxonomies, we developed a method to generate synthetic datasets with controlled relationships. Our approach:

1. Define a set of "atomic concepts" that serve as building blocks for classes
2. Create multiple domains by sampling these concepts to form classes
3. Calculate inter-domain relationships based on shared concepts

This method allows us to precisely control the taxonomy structure while creating realistic relationships between domains. To generate images for these synthetic classes, we can leverage existing datasets by treating each original class as an atomic concept.

2.3.3 Formal Definitions

We define our synthetic taxonomy framework on top of the definitions from section 2.1:

- **Atomic Concepts** $\mathcal{U} = \{1, 2, \dots, n\}$: A set of atomic concepts will be a universe of concepts that make up the basis for our synthetic class generation.
- **Synthetic Class**: A class c^i_j will contain a subset of the atomic concepts from our universe: $c^i_j \subseteq \mathcal{U}$. To maintain disjoint class definitions, we ensure that $c^i_j \cap c^i_k = \emptyset$ for all $j \neq k$.

[1] https://cloud.google.com/vision

2.3.4 Randomized Domain Generation

To create realistic domains, we use normal distributions to sample the number of classes and concepts per class. This allows us to generate domains with varying sizes and complexities, mimicking different real-world datasets.

Parameter Sampling

We sample the number of classes per domain and the number of concepts per class from truncated normal distributions to ensure realistic variation while maintaining control. Since normal distributions are unbounded, we use a truncated version:

$$f(x|\mu, \sigma, a, b) = \begin{cases} \frac{\phi\left(\frac{x-\mu}{\sigma}\right)}{\sigma\left[\Phi\left(\frac{b-\mu}{\sigma}\right)-\Phi\left(\frac{a-\mu}{\sigma}\right)\right]} & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

Where:

- ϕ is the standard normal PDF
- Φ is the standard normal CDF
- a and b are lower and upper bounds

We implement this using SciPy's `truncnorm` module[2], handling SciPy's standardization of bounds internally:

$$X \sim \text{TruncNorm}(\mu, \sigma^2, a, b)$$

Domain Generation Algorithm

To generate a domain C_i, we follow these steps:

1. **Sample set size**: Determine how many concepts l to use for the domain:

$$l \sim \lfloor \text{TruncNorm}(\mu_{\text{concepts}}, \sigma^2_{\text{concepts}}, 1, n) \rfloor$$

2. **Sample concept pool**: Randomly select l concepts from the universe $\mathcal{U}$:

$$P = \{a, b, c, \ldots\} \quad \text{where } a, b, c, \ldots \text{ are sampled without replacement from } \mathcal{U}$$

[2] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.truncnorm.html

3. **Initialise domain**: $C_i = \{\}$

4. **Generate classes**: While concepts remain in the pool ($P \neq \emptyset$):

    a) Sample class size s_j:

    $$s_j \sim \lfloor \text{TruncNorm}(\mu_{\text{classes}}, \sigma^2_{\text{classes}}, 1, |P|) \rfloor$$

    b) Form class c^i_j by selecting s_j concepts randomly from P

    c) Remove selected concepts: $P = P \setminus c^i_j$

    d) Add class to domain: $C_i = C_i \cup \{c^i_j\}$

This algorithm ensures that each concept is assigned to exactly one class within the domain, maintaining our disjointness constraint.

2.3.5 Modeling Cross-Domain Relationships

Once we've generated multiple domains, we need to model the relationships between them to create our ground truth.

Simulating Neural Network Predictions

Our taxonomy generation method assumes that neural network classifiers will predict related classes across domains with certain probabilities. To simulate this, we create ”perfect” synthetic probabilities based on concept overlap.

Relationship Calculation

For any two domains C_A and C_B, we calculate the probability of classifying an instance of class c^A_i as class c^B_j using:

$$\begin{aligned} \text{NaiveProbability}(i, j) &= \frac{|c^A_i \cap c^B_j|}{|c^A_i|} \\ P_{i,j} &= \text{NaiveProbability}(i, j) + \frac{1 - \text{NaiveProbability}(i, j)}{|C_B|} \end{aligned} \tag{2.1}$$

Where:

- NaiveProbability(i, j) is the proportion of concepts in class c^A_i that also appear in class c^B_j

- The second term distributes remaining probability mass evenly across all classes in domain C_B, simulating the behavior of a neural network when encountering concepts it hasn't seen before

A Concrete Example

To illustrate this approach, consider two domains:

- Domain A: $C_A = \{c_1^A = \{1,2\}, c_2^A = \{3,4\}\}$
- Domain B: $C_B = \{c_1^B = \{1,2,4\}, c_2^B = \{5,6\}\}$

For the relationship $c_1^A \rightarrow c_1^B$:

- $\text{NaiveProbability}(1,1) = \frac{|\{1,2\} \cap \{1,2,4\}|}{|\{1,2\}|} = \frac{2}{2} = 1$
- $P_{1,1} = 1 + \frac{1-1}{2} = 1$

For the relationship $c_2^A \rightarrow c_1^B$:

- $\text{NaiveProbability}(2,1) = \frac{|\{3,4\} \cap \{1,2,4\}|}{|\{3,4\}|} = \frac{1}{2} = 0.5$
- $P_{2,1} = 0.5 + \frac{1-0.5}{2} = 0.5 + 0.25 = 0.75$

This example shows how our framework captures partial relationships between classes and how it simulates a perfect neural network classifier's behavior. The resulting probability prediction matrix between two domains can then be used to build a graph of relationships between classes, which can then be turned into a universal taxonomy using the methods described in section 2.2.

No-Prediction Classes

Some datasets have a special class that indicates that the model could not classify the image. For these "no-prediction" classes, we need to adapt the relationship probability calculation: Instead of distributing the remaining probability mass evenly across all classes, we simply ignore it and therefore only have the probability of the overlapping concepts.

2.4 Universal Taxonomy Algorithm

After constructing our initial graph structure from cross-domain predictions (as described in section 2.2), we now need to transform it into a universal taxonomy that merges classes from different datasets into universal classes where they share similar concepts.

2.4.1 Taxonomy Building Rules

1. **Isolated Node Rule**: For any domain class A that has no relationships (neither incoming nor outgoing edges), create a new universal class B and add the relationship $A \to B$. We also define the probability of the relationship's edge as 1 and the classes of the universal class as $\{A\}$.

   This ensures that all domain classes without relationships (which can be created by later rules) are still represented in the universal taxonomy.

2. **Bidirectional Relationship Rule**: When two classes have bidirectional relationships (A → B and B → A), they likely represent the same concept. We resolve this by creating a new universal class C and adding relationships $A \to C$ and $B \to C$ to the graph. The probability of the new relationships is set to the average of the bidirectional relationships and the classes of the universal class will be the two classes that were merged (or, if the two classes are universal classes themselves, the union of their classes).

3. **Transitive Cycle Rule**: If we have relationships A → B → C where A and C are in the same domain, we have a problem since classes within a domain are disjoint, which means that one of the relationships must be incorrect. We solve this by removing the relationship with the lower probability, thus breaking the cycle.

4. **Unilateral Relationship Rule**: A uniliteral relationship A → B indicates that the concepts of class A are a subset of the concepts of class B. We therefore create two new universal classes:

   - Class C, which contains both classes A and B and has incoming relationships from both classes with the probability of the uniliteral relationship. This universal class represents the union of the two classes.
   - Class D, which contains only class B and has a relationship from class B with a probability 1. This universal class represents the concepts of class B that are not in class A.

2.5 Taxonomy Difference Metrics

Now that we have methods for generating a ground truth synthetic taxonomy, we need to define metrics to compare the predicted taxonomy against the ground truth. Comparison can happen at two points in our pipeline:

- **Universal Taxonomy Comparison**: Comparing the predicted universal taxonomy against the ground truth universal taxonomy. This is done after applying our universal taxonomy generation algorithm and allows us to evaluate the quality of the final taxonomy. However, the algorithm might change the scale of differences between our predicted and ground truth taxonomies (e.g. a unilateral vs. bidirectional relationship would be a small difference before the algorithm, but would result in a subset hypothesis with two universal classes vs. one universal class after the algorithm).

- **Relationship Graph Comparison**: Comparing the predicted graph of relationships between classes against the ground truth graph. This is done before converting the relationship graph into a universal taxonomy and allows us to evaluate the quality of the relationships between classes.

2.5.1 Constructing Adjacency Matrices

For our metrics, we first need to represent our intra-domain relationships as adjacency matrices. We concatinate every class from every domain into a single set of classes $C = \bigcup_{i=1}^{n} C_i$, where n is the number of domains. We then create an adjacency matrix $A \in [0,1]^{|C|\times|C|}$, where $A(i,j)$ is the relationship probability between classes c_i and c_j.

Additionally, we need to handle the case where a class has no relationships at all: Since these classes will later become a single universal class, we additionally create a self-loop for every class c_i without relationships, which is defined as $A(i,i) = 1$.

2.5.2 Edge Difference Ratio

Our first metric is the edge difference ratio (EDR), which measures the difference in edge weights between two relationship graphs G_1 and G_2. The metric is bounded between 0 and 1, where 0 indicates that the two graphs are identical and 1 indicates that the two graphs have no edges in common.

For two adjacency matrices A_1 and A_2 of graphs G_1 and G_2, we define the edge difference ratio as follows:

$$\mathrm{EDR}(G_1, G_2) = \frac{\sum_{i,j} |A_1(i,j) - A_2(i,j)|}{\sum_{i,j} \max(A_1(i,j), A_2(i,j))} \tag{2.2}$$

This definition captures the difference in edge weights between the two graphs, while normalizing it by the total edge weights in both graphs (without double counting edges).

Our EDR metric is similar to the Jaccard index [**jaccard_distribution_1912**] as well as the Tanimoto coefficient [**tanimoto_elementary_1958**] when we consider the adjacency matrices as sets of edges. In contrast to these metrics, however, our EDR metric supports weighted edges, which allows us to respect the probabilities of relationships between classes.

2.5.3 Precision, Recall, and F1 Score

While the edge weights in our relationship graphs are important, every single edge (even with a very low probability) can create a new universal class and therefore change the universal taxonomy.

To account for this, we also define precision, recall, and F1 score metrics for the relationship graphs.

For two adjacency matrices A_1 and A_2 of graphs G_1 and G_2, we first create binarised versions of the matrices as B_1 and B_2, where $B_1(i,j) = 1$ if $A_1(i,j) > 0$ and $B_2(i,j) = 1$ if $A_2(i,j) > 0$.

Next, we compute the true positives, false positives, and false negatives as follows:

- **True Positives (TP)**: The number of edges that are present in both B_1 and B_2.
- **False Positives (FP)**: The number of edges that are present in B_1 but not in B_2.
- **False Negatives (FN)**: The number of edges that are present in B_2 but not in B_1.

Using these counts, we can then compute the precision, recall, and F1 score as follows:

- **Precision**: The ratio of true positives to the sum of true positives and false positives.
- **Recall**: The ratio of true positives to the sum of true positives and false negatives.
- **F1 Score**: The harmonic mean of precision and recall.

3 Results

3.1 Domain-Model Training

3.1.1 Datasets

To start with our taxonomy generation, we first need a set of datasets that we can use to train and evaluate our domain models:

- **Caltech-101 and Caltech-256** [**li_caltech_2022**, **griffin_caltech_2022**]: The Caltech-101 dataset contains 101 general object categories with 40 to 800 images per category, while the Caltech-256 dataset extends this to 256 categories with at least 80 images per category. Both datasets have been widely used for image classification tasks[1]. The images are roughly 300x200 pixels in size and contain annotated outlines for each object in the image, which we will not need for our purposes. The dataset has no predefined train/test split, so we will use a 80/10/10 split for training, validation, and testing.

- **CIFAR-100** [**krizhevsky_learning_2009**]: The CIFAR-100 datasets contains 100 classes grouped into 20 superclasses, with 600 images per class. Each image is 32x32 pixels in size, which is significantly smaller than the Caltech datasets. The dataset is one of the most popular datasets for image classification tasks[2]. The dataset has a train/test split of 50000 training images and 10000 test images, which we will further split by dividing the training set into 80% for training and 20% for validation.

- **Synthetic Datasets**: To have a ground truth for our taxonomy generation, we will also create synthetic datasets based on the Caltech-101 and CIFAR-100 datasets. These datasets will be used to evaluate our cross-domain relationship graph generation methods (see Section 2.2). We will create synthetic datasets of varying sizes and complexity and evaluate how well our methods perform for different challenges.

[1] Over 500 open-access papers have cited the datasets, according to Papers with Code: `https://paperswithcode.com/dataset/caltech-101` and `https://paperswithcode.com/dataset/caltech-256`

[2] Over 5000 open-access papers have cited the dataset, according to Papers with Code: `https://paperswithcode.com/dataset/cifar-100`

3.1.2 Training Domain Models

Model Architecture

To now start our training of domain models, we first need to define the architecture of our models. The ResNet architecture[**he_deep_2015**, **he_identity_2016**] is a popular choice for image classification tasks and has been shown to perform well on a variety of datasets. It also has the advantage of being pre-trained on the ImageNet dataset [**deng_imagenet_2009**, **russakovsky_imagenet_2015**], which will save us the effort of training a model from scratch. From the available ResNet architecture sizes, we decide for the leaner ResNet-50 architecture to meet our resource constraints.

We adapt the ResNet-50 architecture to our datasets by switching out the final fully connected layer with a funnel architecture that ends in an output layer matching the number of classes per dataset (see Figure 3.1).

Training Procedure

In our initial training runs, we observe severe overfitting on the training data as can be seen in Figure 3.2. To mitigate this, we apply several regularisation techniques:

- **Dropout** [**hinton_improving_2012**]: As can be seen in Figure 3.1, our fully connected layers contain dropout layers between them at rates of 0.5 and 0.2. Dropout is a regularisation technique that randomly sets a fraction of the input units to zero during training, which reinforces the model to learn more robust features and thereby reduces overfitting.
- **Data Augmentation**: We apply data augmentation techniques to our training data, such as random cropping, horizontal flipping, random erasing, and color jittering. These techniques artificially increase the size of our training dataset and help the model to generalise better by exposing it to a wider variety of input data.

We now train our models on the Caltech-101, Caltech-256, and CIFAR-100 datasets (i.e. their synthetic variants).

For the easier Caltech-101 and Caltech-256 datasets, we use the SGD optimiser [**sutskever_importance_2013**] with a learning rate of 0.01, a Nesterov momentum of 0.9, and a weight decay of 0.0001 and train all variants for 50 epochs. We also use a batch size of 64 for the datasets.

For our more complex CIFAR-100 dataset, we use the AdamW optimiser [**loshchilov_decoupled_2017**] with an initial learning rate of 0.001, a weight decay of 0.001. We train for 100 epochs with a multistep learning rate scheduler that reduces the learning rate by a factor of 0.1 at epochs 30, 60 and 80. For the smaller CIFAR-100 images we use a batch size of 256.

The training is performed on a single NVIDIA RTX 3070 GPU with 8GB of VRAM using the PyTorch Lightning framework [**falcon_pytorch_2019**]. The training process takes approximately

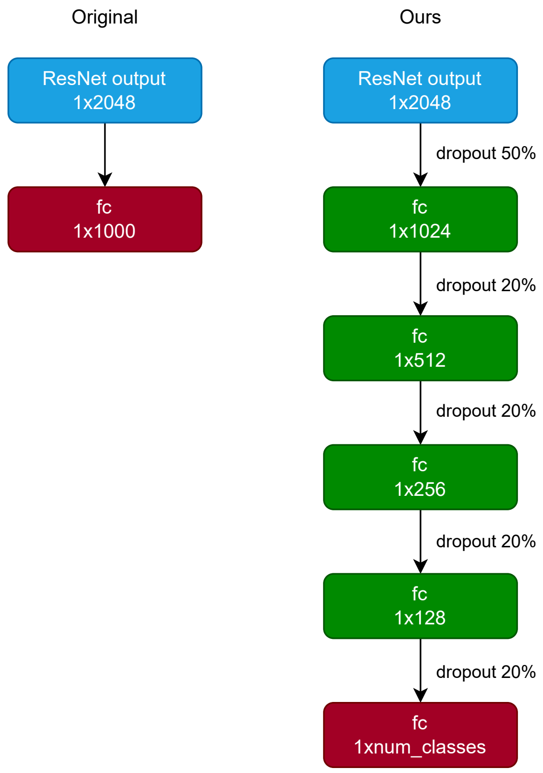


Figure 3.1: Our ResNet-50 architecture with a funnel layer for classification. Blue blocks represent the input from the ResNet-50 architecture, red blocks represent the final output layer, and green blocks represent our new funnel layers.

5 hours for the Caltech-101 and Caltech-256 synthetic dataset variants and approximately 3 hours for the CIFAR-100 synthetic dataset variants.

3.1.3 Model Performance

Synthetic Variants

For our evaluation of relationship selection methods (see Section 3.2.1), we need synthetic dataset variants to calculate evaluation metrics on.

We select the general-purpose Caltech-256 and CIFAR-100 datasets and create synthetic variants of these datasets:

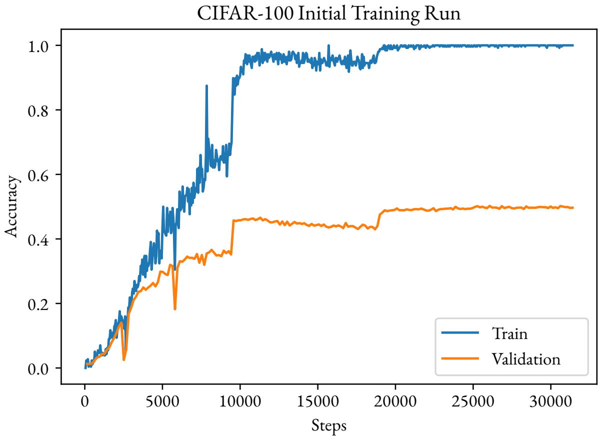


Figure 3.2: Overfitting on the CIFAR-100 dataset during training. The blue line represents the training accuracy, while the orange line represents the validation accuracy. The model overfits on the training data, resulting in a significant gap between the training and validation accuracy.

- **Caltech-256 2-Domain Variant 1**: We create a basic 2-domain variant of the Caltech-256 dataset with parameters $\mu_{\text{concepts}} = 180$, $\sigma^2_{\text{concepts}} = 10$, $\mu_{\text{classes}} = 3$, and $\sigma^2_{\text{classes}} = 1$. The resulting relationship graph (before applying universal taxonomy algorithms) is shown in Figure 3.3a.

- **Caltech-256 2-Domain Variant 2**: We create a simpler 2-domain variant of the Caltech-256 dataset with parameters $\mu_{\text{concepts}} = 200$, $\sigma^2_{\text{concepts}} = 10$, $\mu_{\text{classes}} = 2$, and $\sigma^2_{\text{classes}} = 1$. This variant has fewer concepts per class and therefore fewer relationships between the classes, which might be more similar to simple real-world datasets. The resulting relationship graph (before applying universal taxonomy algorithms) is shown in Figure 3.3b.

- **Caltech-256 3-Domain Variant**: We create a more complex 3-domain variant of the Caltech-256 dataset with parameters $\mu_{\text{concepts}} = 180$, $\sigma^2_{\text{concepts}} = 10$, $\mu_{\text{classes}} = 5$, and $\sigma^2_{\text{classes}} = 1$. This variant has more concepts per class and therefore more relationships between the classes, which makes it more challenging for our relationship selection methods. These extreme numbers should be seen less as a realistic dataset and more as a stress test for our methods. The resulting relationship graph (before applying universal taxonomy algorithms) is shown in Figure 3.3c.

- **CIFAR-100 2-Domain Variant**: For the CIFAR-100 dataset, our 2-domain variant has parameters $\mu_{\text{concepts}} = 50$, $\sigma^2_{\text{concepts}} = 5$, $\mu_{\text{classes}} = 3$, and $\sigma^2_{\text{classes}} = 1$. In the Caltech-256 dataset variants we have used approximately 70% of the classes as concepts, while in the CIFAR-100 dataset variants we use approximately 50% of the classes as concepts. This results in a smaller, more manageable relationship graph that can be better manually inspected. The resulting relationship graph (before applying universal taxonomy algorithms) is shown in Figure 3.3d.

Model Accuracy

Let us now take a look at the accuracy of our models trained on the synthetic dataset variants. We use checkpoints to save the model after each epoch and pick the model checkpoint with the lowest validation loss for our final evaluation.

We can see our final training runs in Figure 3.4. It can be observed that our overfitting mitigation techniques have worked sufficiently well, as our training and validation accuracy curves do not diverge significantly. We present the final model accuracies on the test sets in Table 3.1. Multiple things can be observed:

- All the models achieve an accuracy of around 0.8 on the test set, which is an average performance for these datasets. Our focus is not on achieving state-of-the-art performance, but rather on creating models suitable for our cross-domain prediction task. It should be noted that a lower model accuracy will lead to worse performance in our relationship selection methods, but since we will compare the methods against each other using the same models, this should not be a problem.

- The CIFAR-100 variants have a slightly lower accuracy than the Caltech-256 variants, which is expected since the CIFAR-100 dataset has closely related classes categorised into superclasses, which make it harder for a model to distinguish between them.

- The number of concepts (i.e. classes) in the original dataset that get merged into a new class in the synthetic dataset variants does not seem to have a significant impact on the model accuracy: The Caltech-256 2-domain variant 2 has a $\mu_{\text{classes}} = 2$, while the Caltech-256 3-domain variant has a $\mu_{\text{classes}} = 5$, but the deviation in accuracy is negligible (≤ 0.05).

Now that we have sufficiently good domain models, we can use them to evaluate our relationship selection methods and generate taxonomies from the relationship graphs we create.

3.2 Taxonomy Generation

3.2.1 Relationship Selection Methods

Synthetic Dataset Variants

Now that we have trained our domain models on the synthetic dataset variants, we can use them to evaluate our different relationship selection methods.

Table 3.1: Evaluation results on test sets. Models were checkpointed after every epoch and evaluated on the validation loss. The model with the lowest validation loss was selected for evaluation on the test set.

Dataset Variant	Domain	Steps	Accuracy
Caltech-256 2-Domain Variant 1	A	5520	0.83
Caltech-256 2-Domain Variant 1	B	5220	0.81
Caltech-256 2-Domain Variant 2	A	14450	0.77
Caltech-256 2-Domain Variant 2	B	14700	0.80
Caltech-256 3-Domain Variant	A	5520	0.84
Caltech-256 3-Domain Variant	B	5500	0.81
Caltech-256 3-Domain Variant	C	4940	0.81
CIFAR-100 2-Domain Variant	A	8200	0.77
CIFAR-100 2-Domain Variant	B	7900	0.75

A good relationship selection method needs to be versatile enough to work on a range of different datasets with different characteristics:

- The number of relationships between classes can vary significantly, depending on the number of concepts and classes in the dataset. We therefore need a method that adapts to the probability distribution of the relationships instead of picking a fixed number of relationships.

- The number of classes in the dataset can also vary significantly, which means that some datasets might have classes with few, high probability relationships, while others might have classes with many, low probability relationships. Our method needs to be able to handle both cases and not be biased towards either.

Our edge difference ratio metric (see Section 2.5.2) is a good indicator of the overall accuracy of the relationship selection methods since it measures not only the number of relationships selected, but also the correctness (i.e. the edge weights) of the relationships.

However, since every selected relationship will later result in a node in the universal model, we need to give special attention to the correctness of the relationships selected. Therefore, we will also evaluate the precision and recall of the selected relationships for each method on the synthetic dataset variants.

We will evalute every relationship selection method on every dataset variant by selecting the method parameters that yield the best edge difference ratio on that dataset variant.

When looking at the results in Table 3.2, we can observe two things:

- The naive thresholding method consistently performs best on all dataset variants, closely followed by the relationship hypothesis method. However, the threshold parameter for

Table 3.2: Best EDR results for relationship discovery methods. For each dataset variant and method, the parameter values that yielded the lowest Edge Difference Ratio (EDR) are shown along with the corresponding F1-score.

Dataset Variant	Method	EDR	F1-score	Parameter
Caltech-256 2-Domain Variant 1	MCFP	0.670	0.526	N/A
Caltech-256 2-Domain Variant 1	Naive Thresholding	**0.459**	**0.798**	0.15
Caltech-256 2-Domain Variant 1	Density Thresholding	0.508	0.662	0.70
Caltech-256 2-Domain Variant 1	Relationship Hypothesis	0.482	0.737	5
Caltech-256 2-Domain Variant 2	MCFP	0.557	0.625	N/A
Caltech-256 2-Domain Variant 2	Naive Thresholding	**0.402**	**0.834**	0.15
Caltech-256 2-Domain Variant 2	Density Thresholding	0.451	0.668	0.70
Caltech-256 2-Domain Variant 2	Relationship Hypothesis	0.429	0.774	4
Caltech-256 3-Domain Variant	MCFP	0.707	0.443	N/A
Caltech-256 3-Domain Variant	Naive Thresholding	**0.377**	**0.864**	0.10
Caltech-256 3-Domain Variant	Density Thresholding	0.420	0.740	0.75
Caltech-256 3-Domain Variant	Relationship Hypothesis	0.391	0.830	6
CIFAR-100 2-Domain Variant	MCFP	0.634	0.636	N/A
CIFAR-100 2-Domain Variant	Naive Thresholding	**0.525**	**0.770**	0.15
CIFAR-100 2-Domain Variant	Density Thresholding	0.562	0.688	0.40
CIFAR-100 2-Domain Variant	Relationship Hypothesis	0.534	0.595	4

the naive thresholding method stays relatively consistent across the dataset variants (0.10 – 0.15), while the relationship hypothesis method has a much wider range of upper bounds (4 – 6).

- The original most common foreign prediction method from the Bevandic et al. paper [**bevandic_automatic_2022**] performs significantly worse than the other methods with an edge difference ratio of over 0.5 on all dataset variants. This was to be expected since our synthetic dataset variants rarely have classes with only a single outgoing relationship.

Let us take a deeper look at each of the methods and their performance on the synthetic dataset variants using precision-recall curves for different parameters:

- The **naive thresholding method** (see Figure 3.5) performs well on all dataset variants, achieving a high precision and recall for thresholds between 0.10 and 0.15. The displayed precision-recall curves stay consistent across the dataset variants, which indicates that the method is robust for different dataset characteristics.

- The **density thresholding method** (see Figure 3.6) performs worse than the naive thresholding method, but still achieves good performance. Similar to the naive thresholding method, the CIFAR-100 dataset variant has a slightly lower precision and recall than the Caltech-256 dataset variants, which is expected since the CIFAR-100 dataset has more closely related classes that are difficult to distinguish.

- The **relationship hypothesis method** (see Figure 3.7) performs similar to the density thresholding method, but has a conspicuous drop for the CIFAR-100 dataset variant (prominently more so than the other methods). This is likely due to the fact that this method assumes that every true relationship has an equal probability, which is not the case for the CIFAR-100 dataset variant (where some merged classes, due to having similar concepts that are hard to distinguish, have a much higher probability than others).

When using our method on real-world datasets, we do not have a ground truth to adjust the parameters to. Therefore, we need a single parameter configuration for every method that works universally across all datasets.

We create a new evaluation on the dataset variants using the parameters that yield the best edge difference ratio averaged across all dataset variants. The results are shown in Table 3.3.

The best performing naive thresholding method achieves an edge difference ratio of 0.463 with a noticably higher recall (0.889) than precision (0.675). This means that while we hit almost 90% of the true relationships, we also select a lot of false relationships.

It can be observed that, for precsion, the mcfp method performs best with a precision of 0.865, which is expected since it only selects relationships that has the highest probability. However, this comes at the cost of a very low recall of 0.421 which ultimately results in a lower F1 score compared to the naive thresholding method.

We can conclude that, if you want to create singular high-confidence relationships, the mcfp method is the best choice (especially for single-hierarchy taxonomies). However, if you want to capture as many shared concepts as possible, the naive thresholding method is the best choice, closely followed by the relationship hypothesis method (maybe as a possible alternative if the naive thresholding method does not yield good results).

However, these synthetic dataset variants are not representative of real-world datasets, since the underlying images between images are the same (from the same real-world dataset). Therefore, we will redo the relationship selection method evaluation on real-world datasets to conclude how they perform in different environments and (hopefully) concur on a single parameter configuration that works well across all scenarios.

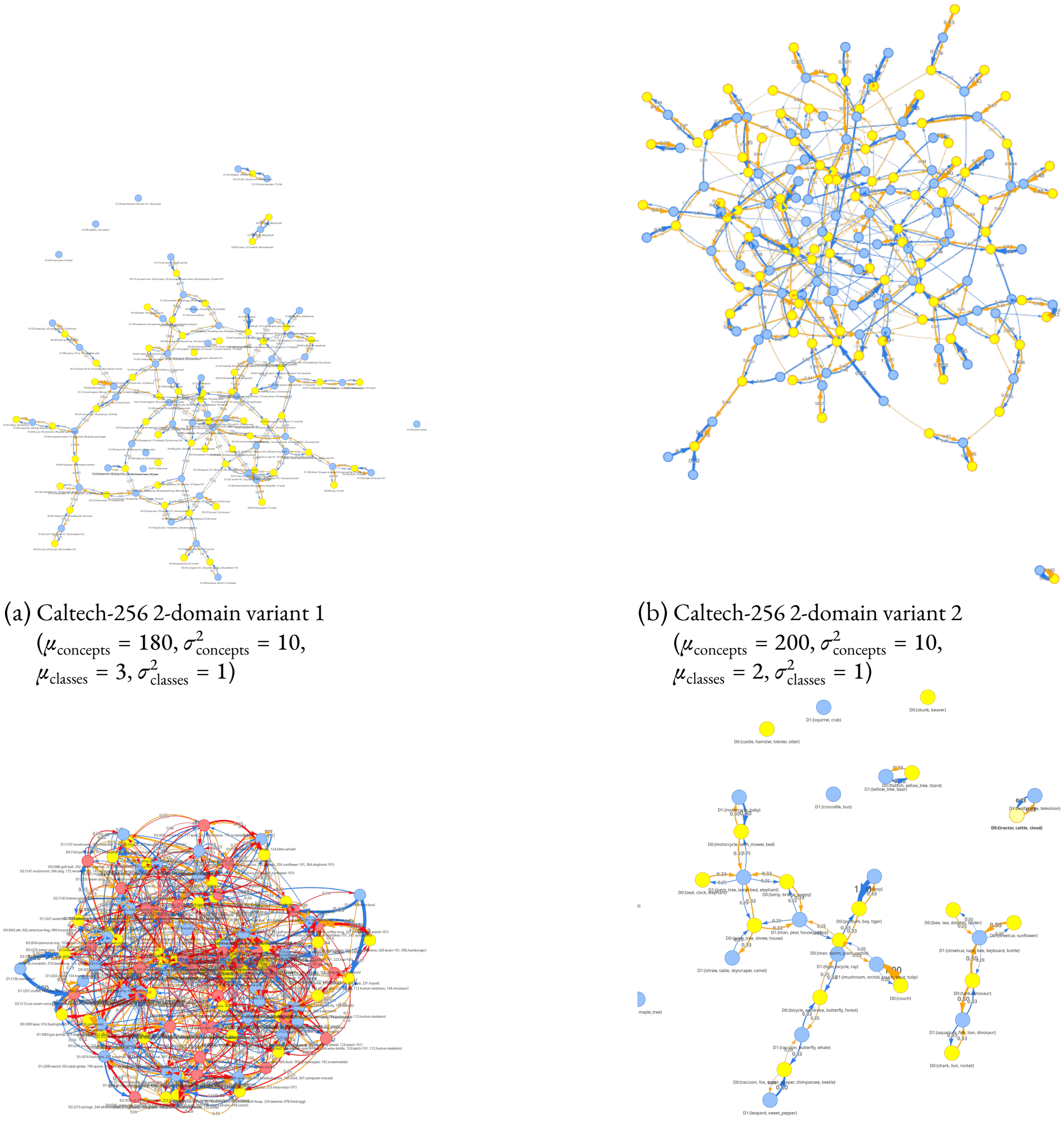
Table 3.3: Average performance metrics for relationship discovery methods with globally optimal parameters. Each method uses the parameter value that minimizes the average EDR across all dataset variants. Performance metrics are then averaged across all dataset variants using these optimal parameters.

Method	Parameter	EDR	Precision	Recall	F1-score
MCFP	N/A	0.642	**0.865**	0.421	0.557
Naive Thresholding	0.10	**0.463**	0.675	**0.889**	**0.760**
Density Thresholding	0.70	0.502	0.514	0.876	0.636
Relationship Hypothesis	4	0.472	0.714	0.750	0.727

Real-World Datasets

3.2.2 Universal Taxonomy Generation

3.3 Universal Models



(a) Caltech-256 2-domain variant 1 ($\mu_{\text{concepts}} = 180, \sigma^2_{\text{concepts}} = 10, \mu_{\text{classes}} = 3, \sigma^2_{\text{classes}} = 1$)

(b) Caltech-256 2-domain variant 2 ($\mu_{\text{concepts}} = 200, \sigma^2_{\text{concepts}} = 10, \mu_{\text{classes}} = 2, \sigma^2_{\text{classes}} = 1$)

(c) Caltech-256 3-domain variant ($\mu_{\text{concepts}} = 180, \sigma^2_{\text{concepts}} = 10, \mu_{\text{classes}} = 5, \sigma^2_{\text{classes}} = 1$)

(d) CIFAR-100 2-domain variant ($\mu_{\text{concepts}} = 50, \sigma^2_{\text{concepts}} = 5, \mu_{\text{classes}} = 3, \sigma^2_{\text{classes}} = 1$)

Figure 3.3: Synthetic dataset variants showing their relationship graphs before applying universal taxonomy algorithms. The number of concepts and classes per concept are sampled from truncated normal distributions with the parameters shown in each subfigure caption.

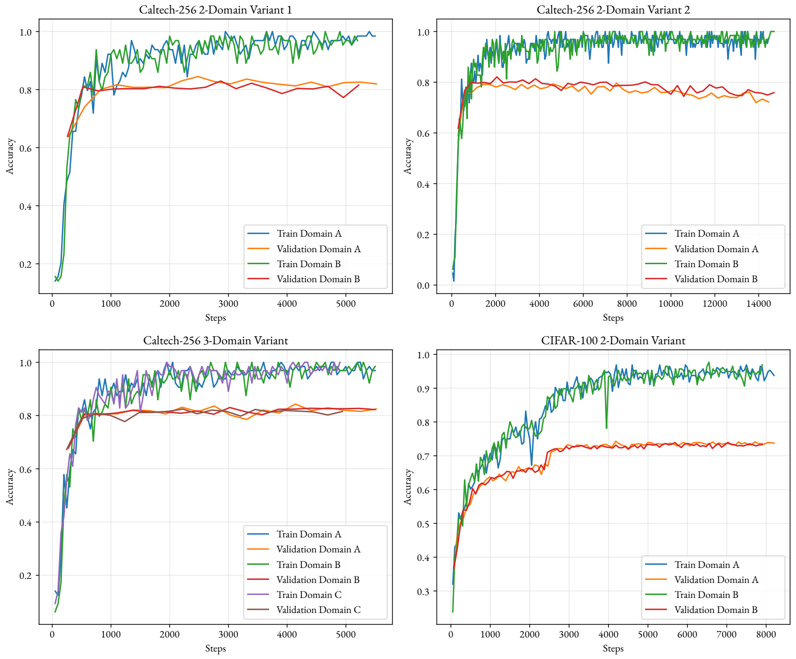


Figure 3.4: Accuracy curves for all synthetic dataset variants. Each subplot shows training and validation accuracy over training steps for all domains in that variant. The models achieve final training accuracies of approximately 0.96-0.98 and validation accuracies of approximately 0.73-0.83.

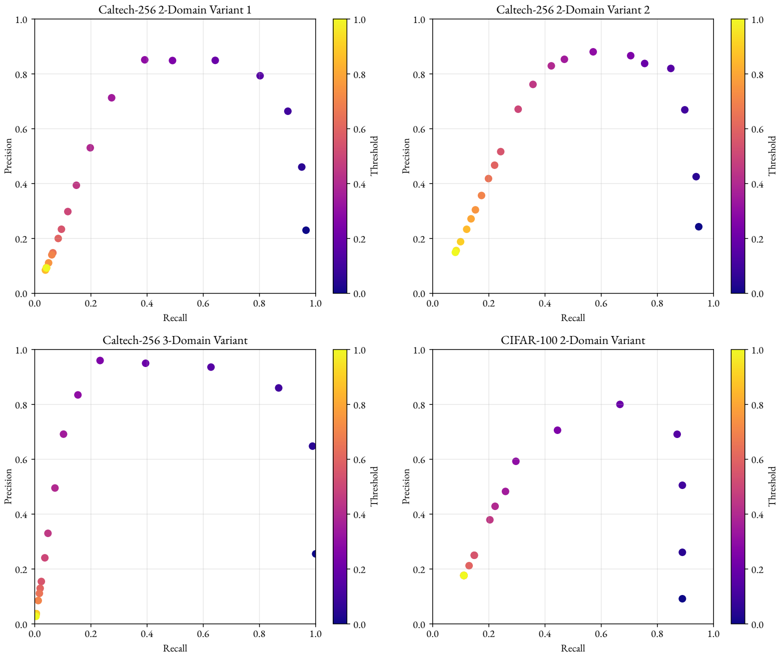


Figure 3.5: Precision and recall plot of the **naive thresholding method** for different thresholds on the Caltech-256 2-domain, Caltech-256 3-domain, and CIFAR-100 2-domain synthetic dataset variants.

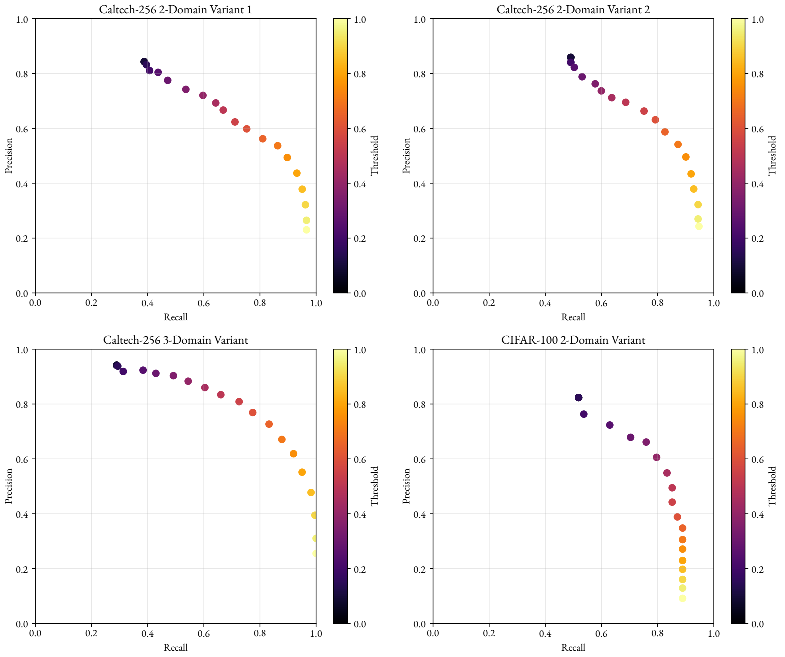


Figure 3.6: Precision and recall plot of the **density thresholding method** for different thresholds on the Caltech-256 2-domain, Caltech-256 3-domain, and CIFAR-100 2-domain synthetic dataset variants.

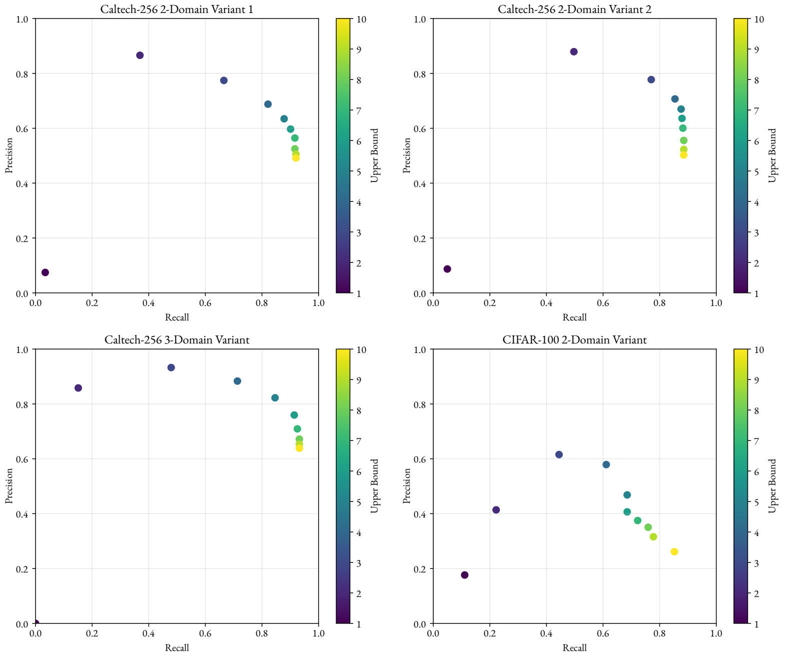


Figure 3.7: Precision and recall plot of the **hypothesis method** for different upper bounds on the Caltech-256 2-domain, Caltech-256 3-domain, and CIFAR-100 2-domain synthetic dataset variants.

4 Discussion

5 Conclusion