This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2022.3209466

Visual Concept Programming: A Visual Analytics Approach to Injecting Human Intelligence at Scale

Md Naimul Hoque*, Wenbin He, Arvind Kumar Shekar, Liang Gou, and Liu Ren

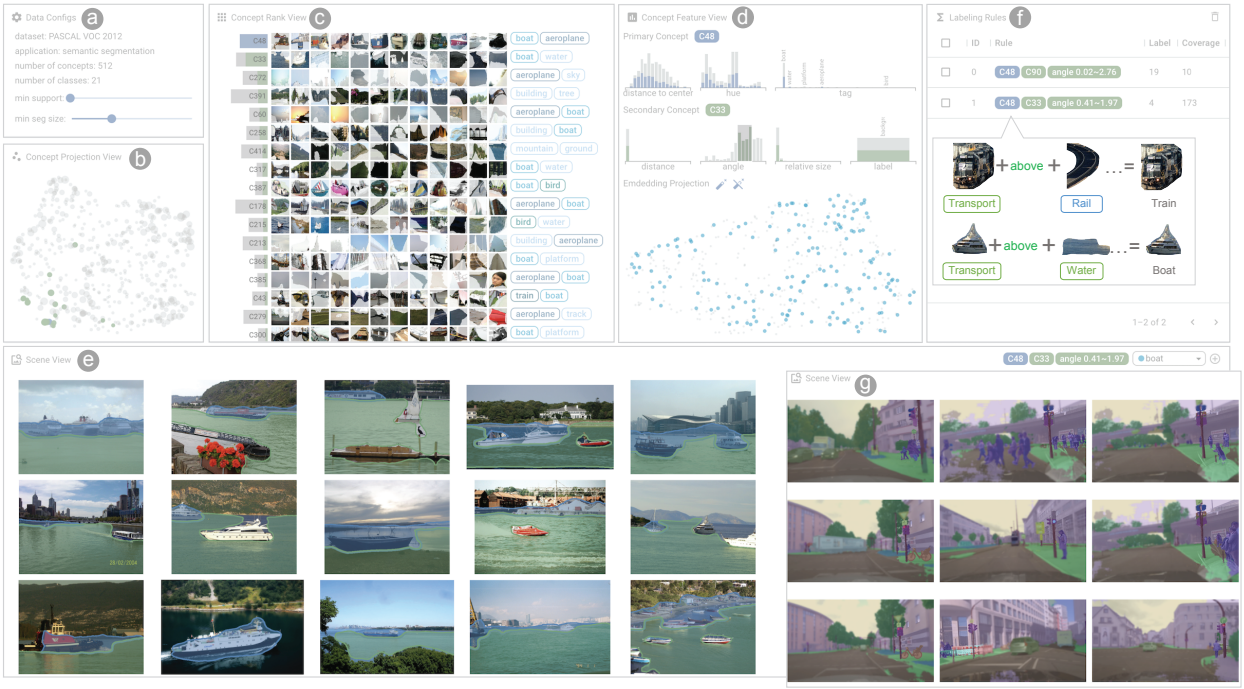

Fig. 1: Visual Concept Programming user interface. (a) Summary statistics and configurations; (b) Projection of concept embeddings in a 2D plane where circle sizes are proportional to the concept sizes; (c) A list of concepts sorted by the number of image segments belonging to these concepts. A user can click on any concept to select it (blue) for further investigation. Upon selecting a concept, the list is updated by positioning the top-k relevant concepts (green) adjacent to the selected concept (blue). (d) Concept feature view shows the distribution of various features (distance to the concept embedding, hue, and semantic tags) of segments assigned to the selected concept (top row). The second row shows relative distances, sizes, and angles of the segments between two relevant concepts. The third row shows the projection of the image segments of the selected concept. (e) Scene view shows retrieved images that match a user-specified labeling function. (f) Shows a history of labeling functions composed by a user. (g) Example images (blurred for privacy issues) from one real-world semantic segmentation application for autonomous driving.

Abstract— Data-centric AI has emerged as a new research area to systematically engineer the data to land AI models for real-world applications. As a core method for data-centric AI, *data programming* helps experts inject domain knowledge into data and label data at scale using carefully designed *labeling functions (e.g., heuristic rules, logistics)*. Though data programming has shown great success in the NLP domain, it is challenging to program image data because of a) the challenge to describe images using visual vocabulary without human annotations and b) lacking efficient tools for data programming of images. We present Visual Concept Programming, a first-of-its-kind visual analytics approach of using visual concepts to program image data at scale while requiring a few human efforts. Our approach is built upon three unique components. It first uses a self-supervised learning approach to learn visual representation at the pixel level and extract a dictionary of visual concepts from images without using any human annotations. The visual concepts serve as building blocks of labeling functions for experts to inject their domain knowledge. We then design interactive visualizations to explore and understand visual concepts and compose labeling functions with concepts *without writing code*. Finally, with the composed labeling functions, users can label the image data at scale and use the labeled data to refine the pixel-wise visual representation and concept quality. We evaluate the learned pixel-wise visual representation for the downstream task of semantic segmentation to show the effectiveness and usefulness of our approach. In addition, we demonstrate how our approach tackles real-world problems of image retrieval for autonomous driving.

Index Terms—Visual concept programming, data-centric AI, data programming, self-supervised learning, semantic segmentation

- *Md Naimul Hoque is with University of Maryland, USA. E-mail: nhoque@umd.edu. *This work was done during his internship at Bosch Research North America.*
- *{Wenbin He, Liang Gou, Liu Ren} are with Bosch Research North America, USA. E-mails: {wenbin.he2, liang.gou, liu.ren}@us.bosch.com.*
- *Arvind Kumar Shekar is with Robert Bosch GmbH, Germany. E-mail: arvindkumar.shekar@de.bosch.com.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2022.3209466

1 Introduction

Data-centric AI [25] has emerged as a new research area to advance machine learning (ML) and land ML models for real-world problems. In the past, the amazing progress of ML mostly followed a model-centric approach. Given a dataset, researchers attempt to introduce human prior knowledge (e.g., model architecture, loss function) into models and iteratively improve the models to achieve the desired results (Figure 2, on the left). The assumption is, given a fixed dataset, a well-designed model can learn useful representation from noisy data and approximate a prediction goal. However, ML models are becoming data hungry in the deep learning era. Right now, creating large-scale annotated datasets is by far a crucial bottleneck to landing deep learning models for real-world problems [28]. Moreover, once assembled, it is difficult to adjust the data to a new domain, task, or objective [30]. For example, integrating newly discovered knowledge from domain experts to a dataset is often desirable, but is not feasible in the existing training mechanism of deep learning. Increasingly, common model architectures are becoming a commodity and started to dominate a wide range of tasks and domains. With model improvement strategies becoming predictable [11, 36], ML practitioners and researchers started focusing more on a data-centric approach that creates, modifies, and evaluates appropriate data for a model to learn (Figure 2, on the right).

Data programming [21,28,29,39] serves as a promising approach for data-centric AI by using human-in-the-loop methods to inject higher-level supervision into the data and generate *weak labels* (i.e., labels that are generated systematically with noisy supervision signals). The weak labels are then used to learn representation for ML tasks. Figure 3 shows an example of applying data programming for an NLP task of sentiment prediction. A sentence can be labeled as with positive sentiment through a labeling function that searches for positives words in a sentence. Many other functions can be defined and applied to a large corpus. These weak labels can be used for large-scale model training. Many data programming approaches [39] have been proposed and successfully applied to different NLP tasks. The most notable one is SnorkelAI from Stanford University [28].

While data programming shows promising results in the NLP domain, there are still two key challenges for engineering image data:

a) It is challenging to *break images into visual vocabularies without human supervision*. Unlike using meaningful words to program text, it is difficult to describe complex scenarios between objects and parts in images to build labeling functions. For example, a person can be defined by a set of body part concepts, including face, hair, hand, etc.

b) We *lack an efficient tool* to program images. Even with visual vocabularies that describe objects and parts in images, it is still challenging to program image data as we need to explore and program an enormous number of objects and parts to define high-level scenarios.

In this paper, we present, **Visual Concept Programming**, the first visual analytics approach of using visual concepts to programming image data at scale while requiring minimal human efforts. Visual concepts can be informally defined as groups of image segments with semantic meanings (e.g., objects and parts). Previous research has shown that visual concepts are useful for interpreting deep neural networks [13, 20, 26, 41]. We steer this line of research to a *new* dimension, from using concepts for model interpretation to utilizing them as a vehicle to inject human intelligence into image data at scale. In turn, the data with human knowledge can improve both the targeting ML model and the visual concepts.

Our approach first generates a set of visual concepts as fundamental building blocks to program images with a self-supervised approach (i.e., **self-supervised visual concept generation**, with no labels required). This generation approach is built upon a series of state-of-the-art ML technologies, including a self-supervised representation learning method [15] to extract groups of image segments, namely concepts, and a zero-shot vision-language model [27] to assign semantic tags to the concepts for easy comprehension.

Then our approach provides visualizations to support both **concept exploration** and **concept programming**. It enables users to explore and understand visual concepts with carefully designed ranking, projection, and relation views. For concept programming, it provides *no-coding* user interfaces to compose, verify, and refine visual labeling functions. For example, a user can create a labeling function like a transportation above water is most likely a boat ($Concept_{Transportation} + above + Concept_{Water} = Class_{Boat}$ in Figure 1f). The user can retrieve all images matching this function to verify and revise it (Figure 1e). These labeling functions are then applied to all images to generate large-scale weak labels (e.g., boat and person).

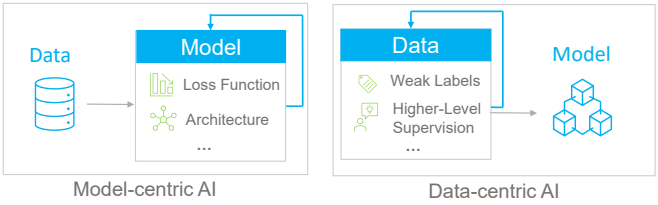

Fig. 2: Model-centric AI vs. data-centric AI. In a model-centric approach, the focus is to iteratively improve the models by introducing human prior knowledge (e.g., model architecture, loss function) in model design. In a data-centric approach, researchers focus on acquiring and improving data for better model performance.

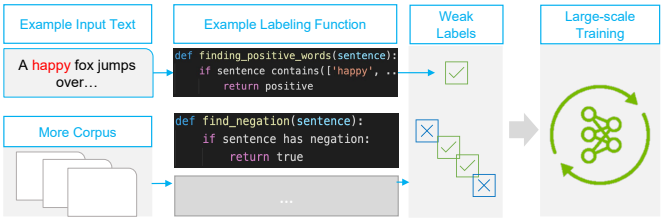

Fig. 3: An example of *Data-Programming* for an NLP task of sentiment classification. A sentence can be programmed as positive sentiment through a labeling function that searches for positives words. Many other functions can be defined and applied to a large corpus to generate weak labels. The weak labels can then be used for model training.

In our experiments, we show that such weak supervision can improve the performance of downstream tasks (e.g., image segmentation) as well as the visual concepts. In summary, our contributions are as follows:

- A novel Visual Concept Programming framework for injecting human intelligence into image data at scale by utilizing self-supervision based visual concepts.
- A visual analytics tool that allows subject-matter experts to explore and understand concepts, interactively compose, verify, and edit labeling functions *without writing code*.
- Case studies and experiments demonstrating the effectiveness and efficiency of Visual Concept Programming for engineering large-scale image data and its applications for real-world problems, such as driving scenario creation, retrieval, and validation for autonomous driving.

To highlight this work, we believe **Visual Concept Programming** is the first visual analytics approach to program image data *at scale*. Its efficiency comes from self-supervision based visual concepts. This work is echoing the current ML research trend of *Data-Centric AI* by iterating data (e.g., quality or higher-level supervision), not models. It also explores new research opportunities of applying visual analytics to inject human knowledge into large-scale data efficiently, by leveraging the powerfulness of representation learning (e.g., visual concept learning) to augment human cognition (e.g., providing higher-level supervision for complicated data).

2 Related Work

In this section, we review related work in data-centric AI, visual analytics for deep learning, and concept extraction and exploration.

2.1 Data-centric AI

Data-centric AI is the discipline of systematically engineering data to build AI systems [1]. Data engineering and feature extraction have

long been integral parts of developing AI systems. However, with the advance of deep learning, the need for feature engineering has rapidly vanished since these models can learn highly discriminative representations from data. While that is true, the ability to engineer existing data to a new task is still challenging for machine learning, especially since obtaining new training data can be a painstaking and costly process. This is evident from the many semi-supervised, self-supervised, and weakly-supervised methods proposed over the years.

There are two main approaches to utilizing the data with limited or no annotations: semi-supervised learning [5] and self-supervised learning [3]. In semi-supervised settings, there is often a small labeled dataset and a large unlabeled dataset. The idea is to utilize domain specific assumptions to exploit unlabeled data that is often cheaply available. In self-supervised settings such as *representation learning* [3], the main objective is to learn useful representation from data without any labeled data. The learned representation can generalize to many downstream applications such as classification, detection, few-shot learning, domain adaptation, reinforcement learning, and generative models. However, these methods often lack a mechanism to inject human intelligence efficiently.

Meanwhile, *data programming* [29, 39], one of the most promising data-centric approaches, is a method where domain experts write *labeling functions*, domain heuristics that generate weak labels for a dataset. This is also referred as a weakly-supervised learning approach. The labels assigned in this way are more scalable than labeling each training sample separately. This approach achieves state-of-the-art performance on many downstream applications. Ratner et al. further proposed Snorkel [28], a system to create labeling functions for large-scale data.

While data programming is shown to be effective for NLP applications, it is still challenging to program image data. First, there is no dictionary of visual vocabulary to inject human knowledge for image data, unlike using meaningful words to program text. In this work, we adapt the mechanism of data programming for image data. Our visual labeling functions are implemented by first training a self-supervised model to extract visual concepts [15], and then using a novel visual interface to write the labeling functions without coding and train the model iteratively. In essence, we present a unique approach to program images built upon self-supervised concept extraction.

2.2 Visual Analytics for Model-centric AI

The majority of visual analytics systems for AI is model-centric. One dominant research direction along this thread is Explainable AI (XAI) [17, 32], emerging as a trending topic for explaining deep learning models. The motivation is to investigate the mechanisms of deep learning models, as it is imperative to interpret or explain the model behaviors in many socially sensitive domains such as autonomous driving [14], healthcare [12], and financing [10] decision-making. Methods such as LIME [31] and SHAP [23] and visual analytics systems such as the What-if Tool [37], LSTMVis [33], SUMMIT [18], VASS [16], and VATLD [14] are examples of solutions that facilitate XAI principles. While these methods and interfaces were shown to be effective for their purposes, they were predominantly developed to interpret and refine task specific models. None of them facilitate data programming for weak supervision such as ours, shifting the focus of current visual analytics research from model-centric AI to data-centric AI.

2.3 Visual Analytics for Data Labelling

Another research direction relevant to our work is visual analytics systems for data labeling [4]. Choi et al. [8] presented AILA, an interactive interface that helps users label documents efficiently by highlighting important words in the document. Desmond et al. [9] proposed an AI-assisted interface that highlights probable labels for users. Zhang et al. [40] proposed OneLabeler, a system that provides common modules and states for data labeling. Interactive systems have also been proposed to verify crowdsourced labeled dataset [22, 38]. While these systems have increased efficiency in data labeling, they still predominantly depend on crowd workers to label data instances one by one. As a result, it is costly for model developers to obtain a new labeled dataset, modify an existing dataset to adjust to a new domain, or explore new ideas without the need to arrange lengthy data labeling process. We present a novel visual analytics framework that aims to solve this problem for image datasets by allowing model developers to write labeling functions that can label thousands of images. We utilize visual concepts as building blocks for writing labeling functions, a unique approach to injecting human intelligence into data labeling.

2.4 Visual Concept Extraction and Exploration

Visual concepts provide a meaningful way to interpret and steer deep neural networks for vision tasks. Typically, deep neural networks operate on low level features such as pixel-values and neural activation. In contrast, humans learn and reason from perceptual groups and structures (i.e., visual concepts) [2, 6, 13]. We can unravel complex correspondence between concepts, generalize knowledge, and understand unseen concepts. Similar reasoning power for a vision model can help it mimic human learning and generalize to multiple downstream applications. Additionally, visual concepts are human-understandable and provide a way to quantify the interpretability of a model [13, 20, 26, 41].

Several methods have been proposed for quantifying the interpretability of a neural network based on visual concepts. Bau et al. [2] proposed Network Dissection, a general framework for scoring interpretability of any hidden layer of a Convolutional Neural Networks (CNN) with respect to a wide range of human-interpretable visual concepts. Similarly, Kim et al. [20] used directional derivatives to quantify the sensitivity of ML model predictions for different user-defined concepts.

The above methods are effective in interpreting internal states of a CNN. However, they require user-defined concepts, which are expensive to annotate. Further, the space of possible concepts to query or define can be unlimited, or in some settings be unclear. To address these limitations, researchers have proposed methods to automatically extract concepts from a neural network [6, 13]. However, these methods are still only applicable in supervised settings only.

Finally, researchers have also used human-in-the-loop approaches to extract and interpret visual concepts. Zhao et al. [41] proposed ConceptExtract, a human-in-the-loop active learning system to extract visual concepts. Park et al. [26] on the other hand, proposed an automated and scalable method to extract and visually summarize concepts.

Our work further advances how visual concepts can be extracted not only to interpret models, but also as the building blocks to program data and inject human intelligence in a scalable way.

3 Method

Figure 4 presents the overview of our visual concept programming approach. Given an image dataset, we first train a self-supervised representation learning model to extract a set of visual concepts (Section 3.1.1). Each concept is a group of image segments with similar semantic meanings such as human face and body. Meanwhile, we use a vision-language model named CLIP [27] to assign tags to each concept (Section 3.1.2). Then, we build an interactive visual interface to help users explore and program the visual concepts (Section 3.2). By programming the visual concepts, users can generate weak labels for the corresponding image segments (Section 3.3) with concept-based labeling functions. In the end, with weak supervision, users can retrain the representation learning model to improve the performance on downstream tasks (Section 3.4) and the quality of visual concepts. Users can again apply visual concept programming to the refined model to improve both data annotations and model performance iteratively.

3.1 Visual Concept Extraction and Tagging

3.1.1 Self-supervised Visual Concept Extraction

Our approach starts with extracting a set of visual concepts from the input image dataset without using any human annotations. To this end, we utilized a self-supervised representation learning approach [15], which learns the embedding (i.e., a high-dimensional feature vector) of each pixel to segment images into semantically meaningful regions. Meanwhile, the image segments are grouped into visual concepts where each concept captures image segments with similar semantic meanings such as human face, hair, and body.

This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2022.3209466

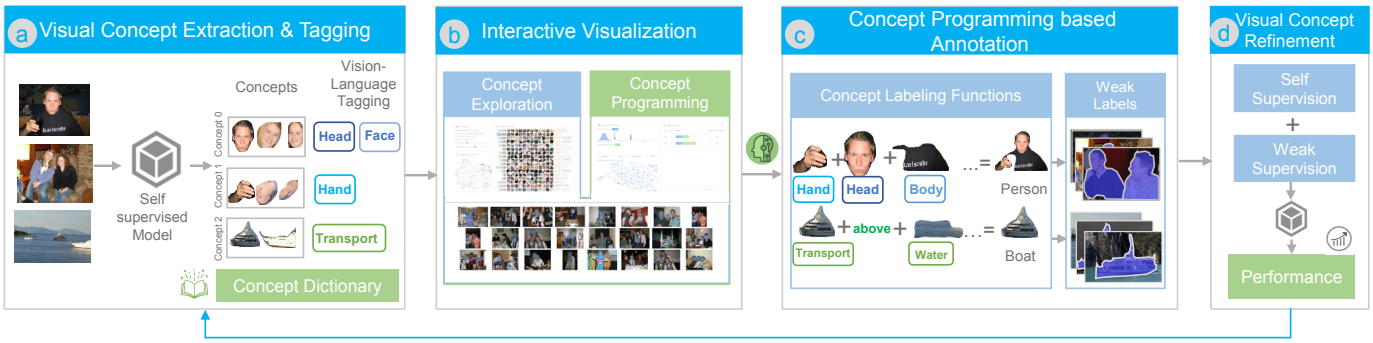

Fig. 4: The framework of visual concept programming. a) Visual concept extraction and tagging component encode large-scale images into a dictionary of visual concepts through a self-supervised representation learning model. Each visual concept is associated with a group of image segments that share similar semantic meanings such as human head, hand, etc. The visual concepts are then tagged via a zero-shot vision-language model, CLIP [27]; b) Interactive visualizations are provided to help users explore and understand the concepts, and also enable users to program concepts by supporting to compose, verify, and refine labeling functions with visual concepts; c) User generated labeling functions are applied to large-scale images and generate weak labels for image segments; d) Finally, weak labels are used to retrain the representation learning model to improve image segments and concepts.

The main idea of the self-supervised learning approach are shown in Figure 5. Starting from images with pseudo segments generated by edge detection, different augmentations are generated and fed into a CNN to produce pixel embeddings. The CNN is then trained to generate similar pixel embeddings for regions with similar semantic meanings (please refer to [15] for details). During training, a set of visual concepts are also updated by clustering the embeddings. After training, images are segmented by clustering the pixel embeddings using k-means. For each image segment, we calculate the mean vector of the pixel embeddings and use it as the embedding of the segment. Each image segment is then assigned into one visual concept based on the distance between the embeddings of the segment and the visual concepts. After the concept extraction process, we have a set of visual concepts, where each concept contains a group of image segments with similar semantic meaning and embeddings. Note that while the model learns pixel-wise embeddings during training, the pixel-wise embeddings are aggregated into embeddings of segments and concepts for our visual analytics approach. The embedding of a concept is the cluster center of embeddings of segments for that concept.

3.1.2 Zero-shot Vision-Language Tagging for Visual Concepts

We used CLIP [27] to tag the extracted visual concepts. The motivation is to provide high level overview of the visual concepts' semantic meaning, in addition to showing image segments of each concept.

CLIP is a vision-language model pre-trained on a large number of text-image pairs, and its goal is to generate appropriate description for a given image. CLIP uses a text encoder and an image encoder to get the embeddings of texts and images. The encoders are trained by attracting the embeddings of texts and images that are paired and repeling the embeddings of texts and images that are not.

A pre-trained CLIP model is used to tag the visual concepts as shown in Figure 5c. Given a concept, we feed the corresponding image segments into the image encoder and compare their embeddings with the embeddings of a set of predefined tags. Then we assign the tag with the closest distance to each image segment in the embedding space. All predicted tags are aggregated over all image segments within this concept, and the tag distribution is then computed. The top-k tags (e.g., $k = 2$ in our case) with high frequency are used to describe the concept.

3.2 Data Transformation for Concept Exploration and Programming

To develop a visual analytics tool to explore and program the extracted visual concepts effectively, we need to prepare and transform relevant data from visual concepts. In this section, we describe data preparation methods to facilitate both concept exploration and programming.

3.2.1 Visual Concept Exploration

To identify interesting visual concepts, we design two mechanisms to preprocess the visual concepts: ranking and projection.

Concept Ranking We rank the visual concepts at two levels: global and local.

a). First, we rank the concepts globally based on the number of image segments within each concept. With the global ranking, users can quickly identify visual concepts that cover more data.

b). With a selected concept, we also rank the concepts locally based on the frequency of cooccurrence with the selected concept. With the local ranking, users can identify relevant visual concepts and program the visual concepts based on their relations.

Concept Projection We visually group the concepts by projecting them into a 2D space at two levels.

a). First, we project the embeddings of the concepts into a 2D space with UMAP [24]. With the projection, users can locate concepts with similar semantic meanings, such as the humans head and hair.

b). While exploring a concept of interest, we project the associated image segments into a 2D space with UMAP. Users can explore subsets of the concept with the segments projection.

3.2.2 Visual Concept Programming

Using the visual concepts as building blocks, users can define *labeling functions* to program the image segments with heuristics and prior knowledge. For example, if an image contains a *transport* above *water*, the *transport* is most likely a boat. Hence, the label "boat" can be assigned to all *transport* above *water*. Inspired by previous work [7, 35], we derived several features from the concepts to enrich labeling functions with our domain experts. The features can be summarized into two groups - characteristics of concepts and their relations.

First is the concept's own features, includes:

- *Color statistics* of each image segment is used to help users form heuristics with respect to object types and colors.
- *Tags* of the image segments generated by CLIP are used to help users understand each concept and provide hints on the class labels of the segments. Note that we only use tags to provide hints on concept programming because the tags on individual image segments are noisy.
- *Embedding* of each image segment extracted from the self-supervised model is projected into a 2D space to help users identify groups of similar segments to assign labels. Meanwhile, their distance to the concept's embedding can be used to identify outlier image segments, which have a higher chance to belong to a different type of object.

Second, the characteristics of relations between two concepts are also important to define labeling functions such as the relative position of objects [7]. For a specified concept, users can consider the following relations between this concept with other concepts:

- *Geometric properties (e.g., distance, angle, relative size)* are used to define labeling functions by considering image segments' relative position and size. For example, the knowledge *monitors are*

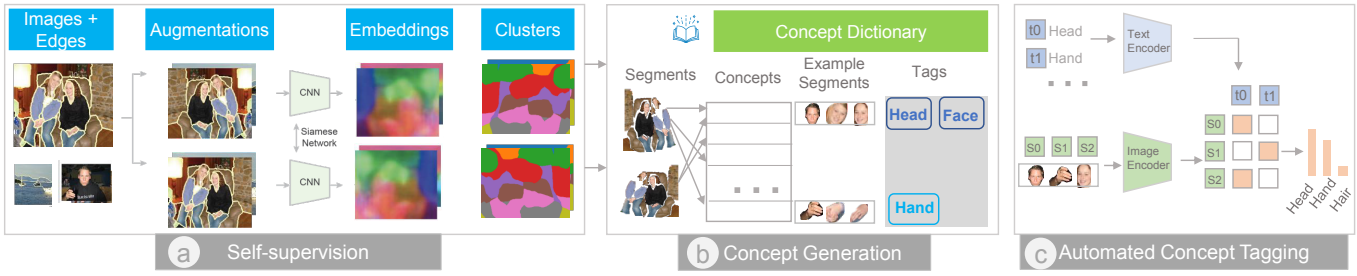

Fig. 5: Visual concept extraction and tagging module. (a) A self-supervised model is trained on input images that are roughly segmented with edge detection to generate pixel embeddings that capture the semantic meaning of each pixel. The pixel embeddings are used to segment images into meaningful regions (e.g., hair, head). (b) Meanwhile, the image segments are grouped into a dictionary of visual concepts. Each visual concept is then tagged with a vision-language model, CLIP [27], to provide a high-level overview of the concepts. (c) Details on tagging a visual concept using CLIP. All image segments (S_i) assigned to a visual concept are fed into the CLIP image encoder. For each segment, the embedding similarity between this segment and a set of pre-defined tags (T_i) is used for tag assignment for this segment. The distribution of the tags for this concept is then computed and the tags with high frequency are used to tag the concept.

often placed on desks can be injected into the data by considering the relative position of monitors and desks.

- *CLIP tags/Weak labels* of one concept can be used to infer the class labels of the other concept's image segments. For example, given a concept of tires, it is difficult to tell each tire belongs to a bus or car. In this case, users can use the CLIP tags/weak labels (if available) of vehicles' body to infer the class label of tires.

3.3 Concept Programming based Annotation

When a labeling function is defined by a user, it will be applied to all image segments, and the segments that satisfy the condition will be labeled automatically. For example, as shown in Figure 4c, there is one example labeling function:

$Concept_{transport} + above + Concept_{water} = Class_{boat}$

As this labeling function is applied, all *transports* that above *water* will be labeled as "boat". Note our method does not require the labeling functions to produce 100% accurate labels and the labeling functions may disagree with each other on certain image segments. To resolve the conflicts, our method analyzes the labeling functions and uses labels generated by a more precise labeling function (i.e., a labeling function that involves more concepts and constraints, and thus covers fewer segments). Meanwhile, users can also visualize the image segments with conflicted labels and resolve the conflicts manually.

3.4 Visual Concept Refinement

With the weak labels generated by the labeling functions, users can retrain the representation learning model [15] with the combination of self- and weak-supervision.

For the self-supervision, data augmentation and contrastive learning [15] are used to calculate the *self-supervision loss*.

The contrastive learning is also extended to weak labels as follows. Given a pixel p with label c, all image segments of label c are considered as its positive pairs C^+ and the rest of the segments are considered as its negative pairs C^-. Then the *weak-supervision loss* is defined as:

$$\mathscr{L}_{weak}(p) = -log\frac{\sum_{s\in\mathscr{C}^+} exp(sim(\mathbf{z}_p,\mathbf{z}_s)\kappa)}{\sum_{s\in\mathscr{C}^+\cup\mathscr{C}^-} exp(sim(\mathbf{z}_p,\mathbf{z}_s)\kappa)}, \quad (1)$$

where $\mathbf{z}_p$ and $\mathbf{z}_s$ are the embeddings of a pixel p and segment s, respectively. With the *weak-supervision loss*, image segments with the same label are forced to have similar embeddings. The distance between the embeddings $sim(\mathbf{z}_p,\mathbf{z}_s)$ is measured by cosine distance. The concentration of the embeddings is controlled by a constant κ. In the end, the self- and weak-supervision losses are combined and used to train the representation learning model.

4 Design Requirements

In this section, we outline the design requirements of an interactive visual interface for programming visual concepts, guided by our framework discussed above. The requirements broadly fall into two categories of **concept exploration** and **concept programming**. First, we identify three design requirements for concept exploration:

- *R1.1. summarize and explore individual concepts.* We need a starting point for the exploration process. A natural choice is to design the UI so that the prominent concepts are evident from a glance. This requires us to sort, filter, and search concepts based on some predefined metrics.
- *R1.2. visualize and understand relations between concepts.* We will need to write labeling functions that outline relationships between concepts. Thus, once we find a concept of interest from R1.1, we need to first find concepts that potentially have relationships with that concept. Secondly, we need evidence so that we can pinpoint how the two concepts are related to each other. We will need visualizations that appropriately summarize the numerical evidence (e.g., distance, co-occurrences). We will also need to examine how the concepts are encoded in the input images for understanding the semantic relations.

Our second set of requirements relates to composing, verifying, and refining labeling functions based on visual concepts.

- *R2.1. interactively compose labelling functions without writing code.* In current data programming approaches, domain experts manually write labeling functions [28, 29]. In our case, the labeling functions should be generated automatically as users interact with the concepts in the system.
- *R2.2. verify labeling functions.* After defining a labelling function, users should be able to apply it to relevant images and verify the correctness of a labelling function.
- *R2.3. efficiently refine labeling functions.* We anticipate that the labeling functions may need to be refined as new evidences emerge from the the exploration process. Thus, our system should provide functionalities to edit the history of labeling functions.

5 The Visual Concept Programming User Interface

The visual interface of Visual Concept Programming is divided into five coordinated views (Figure 1). We describe each view in detail below.

5.1 Concept Projection and Rank Views

Concept Projection View The view visualizes the concepts' embeddings in a 2D plane (Figure 6). The radius of the circles are set to be proportional to the number of segments in the concepts. This helps in finding prominent concepts quickly (**R1.1**).

Concept Rank View The view visualizes the concepts in a list, where each row in this list represents one concept. The list is sorted by concept size to show the prominent concepts at the top (**R1.1**). To summarize the concepts effectively, we encoded this view with three types of information.

First, on the left side of the view, we encode the total number of segments belonging to each concept using bars encompassing the concept IDs. Second, in the center, we show sample segments belonging to each concept. To select the samples for a concept, we sort the segments based on their distance to the concept's embedding and then select

This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2022.3209466

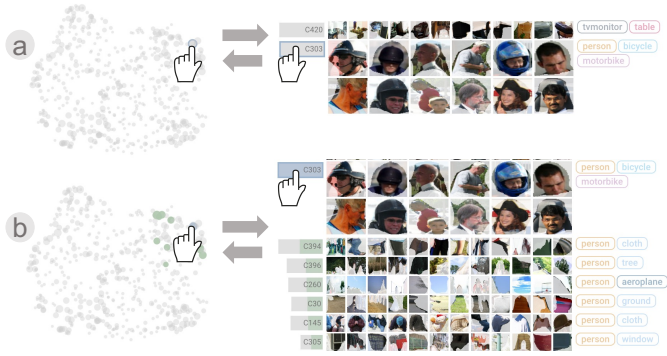

Fig. 6: Interacting with the Concept Projection and Rank View. The two views are synchronized. a) On hovering over a concept in the projection/rank view, the sample segments from the concept is enlarged and highlighted in the rank view. b) A subsequent click on the concept will update the rank view by positioning the 16 most relevant concepts adjacent to the clicked concept.

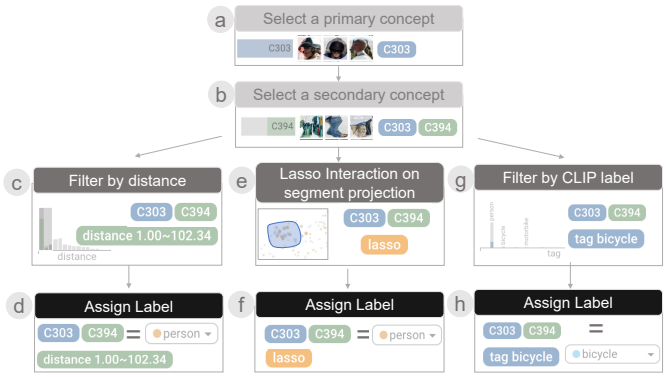

Fig. 7: Interaction to write labeling functions. A user starts with selecting (a) primary and (b) secondary concept from the projection or rank view. The user can then add different types of optional filters in the feature view to write the labeling functions. For example, the user can select segments of interest by applying a distance filter (c) and then assign a label to the selected segments (d). The user can apply a lasso selection on the projection of the segments belonging to the primary concept in the feature view (e) and then assign a label to the selected segments (f). The user can also use the CLIP tags to filter the data (g) and (h).

$n = 16$ segments out of all segments in equal intervals. Figure 6a shows sample images from Concept 303. Finally, on the right side of the segments, we show the CLIP tags.

Interactions between the Projection and Rank Views The two views are synchronized as follows. While a user hovers over a circle in the concept projection view, the corresponding concept will be highlighted in the concept rank view and the segments will be enlarged (Figure 6a). Similarly, a user can hover over a concept in the concept rank view to enlarge the segments and highlight the concept in the concept projection view.

A subsequent click on either view will select a concept to focus on, which is named as the *primary concept* in this work. Upon selection, we highlight the primary concept with a blue color in both views. We also highlight the most relevant (co-occurring) concepts (in green) to the primary concept in both views (**R1.2**). In addition, we reorder the list and position the relevant concepts adjacent to the primary concept in the concept rank view. For example, Figure 6b shows the list with $C303$ as the selected primary concept and $C394$, $C396$, and $C260$ as the top 3 relevant concepts. Note that each concept shows the number of co-occurrence with the primary concept using green bars. Users can select one of the relevant concepts as the *secondary concept* to explore its relations to the primary concept.

5.2 Concept Feature View

The Concept Feature View (Figure 1d) visualizes the details of the primary concept and its relation with an optional secondary concept (**R1.2**). Both primary and secondary concept can be selected from the Concept Projection or Rank View. This feature view is divided into three rows. In the top row, we show the distribution of segments belonging to the primary concept across three different features: *distances to concept's embedding, hue,* and *tags*. If a secondary concept is selected, the bar charts highlight the distribution of the segments that co-occur with the secondary concept. Users can brush any parts of the three distributions to select segments of interest.

Similar to the top row, the middle row visualizes distribution of the secondary concept. By default, the bar charts in this row show distribution of segments that co-occur with the primary concept. It visualizes four relative features of the co-occurred segments: *distance*, *angle, relative size*, and *tags/weak labels*. Users can again brush any parts of the four distributions to select segments of interest.

Finally, the bottom row shows a projection of the segments belonging to the primary concept (Figure 1d, bottom row). Different from the concept projection view that provides the global view of all concepts, this view provides a zoomed-in view of a specific concept by visualizing the projection of image segments within the concept as a scatter plot. This helps users to identify groups and outliers in the concept. Moreover, the labels of the segments are color encoded in the scatter plot to help users identify unlabeled data or data with conflicting labels.

Interacting with the visualizations in this view helps users write labeling functions (**R2.1**). A labeling function is automatically updated in the scene view when users interact with the feature view. For example, Figure 7 shows how a user can iteratively update a labeling function by interacting with the feature view.

5.3 Scene View

The scene view (Figure 1e) is designed to show segments and concepts in the context of the whole images. When a primary concept is selected, the scene view is updated with the images containing the segments from primary concept. The segments are highlighted with a blue mask in the images. When a secondary concept is selected, the scene view is updated with the images that contain both primary and secondary concept together (**R2.2**). Segments from the secondary concept is highlighted using a green mask. When users updating the labeling function, the images that satisfies the function are automatically retrieved and visualized in the scene view.

5.4 Labeling Rules View

The final output of Visual Concept Programming is a set of rules outlined by the labeling functions. The labeling rules appear in two places in the interface. First, when users interact with the various views in the interface, a labeling rule is automatically created and visualized in the scene view (Figure 1e). Users can choose to save this rule in the labeling rules view and later refine and export all the rules to label the data for retraining the representation learning model (**R2.3**). The labeling rules view shows the history of rules (Figure 1f). Each rule here is assigned an ID and shows the rule and the number of segments covered by the rule. Users can use this list to go back to prior rules and refine them as required (**R2.3**).

6 Case Studies and Evaluation

We worked closely with three MLOps developers from the product team of autonomous driving and evaluated the usefulness and effectiveness of our approach. We worked on two use cases: 1) A *public* semantic segmentation benchmark dataset, PASCAL VOC, to validate this approach as a Proof of Concept (PoC) engagement; 2) An *internal* dataset showing cases of applying this approach to MLOps of perception tasks for autonomous driving.

6.1 Improving the Performance of Semantic Segmentation

In this use case, we worked with the developers on a standard semantic segmentation task to validate and benchmark our approach. The goal

Fig. 8: (a) Visual concept (C176) of ground transportation's body. (b-e) Sample images of C176, which include body of bus, train, and car (highlighted in blue).

is to use visual concept programming to improve the performance of semantic segmentation.

6.1.1 Experiment Setup

Dataset We use the Pascal VOC 2012 dataset for this use case, which contains 20 object classes and one background class. We use the *train_aug* set with 10,582 images for the visual concept programming and evaluate model performance on the *val* set with 1,449 images. For self-supervised pre-training, we follow this work [15] with pseudo segmentations generated by HED-owt-ucm [19].

Training and testing The hyper-parameters for training are set as follow. The embedding dimension is set to 32 and the size of the concept dictionary is set to 512. We train the model on the *train_aug* set with pseudo segments (and weak labels in the refinement stage) for $5k$ iterations with a batch size of 8. The learning rate is initialized to 0.001 and decayed with a poly learning rate policy.

The semantic segmentation is then generated on the *val* set using k-means clustering proposed in [19]. For each image, we first cluster the pixels into segments based on their embeddings. Then each segment is assigned a class label based on the majority vote of its nearest neighbors in the training set. Note that here we use the labels in the training set only for the purpose of evaluation. The labels are not used during training. We cluster each image into 25 segments and use 15 nearest neighbors in the training set to get the class label of the segments. The performance of the segmentation results are then evaluated using *Intersection-over-Union (IoU)*, which measures the ratio of the intersection and union of the prediction and ground truth segments.

Iterative Concept Programming We worked with the model developers to program visual concepts through an iterative process. In each iteration, developers perform the following steps:

- Select one type of object by comparing model performance with the previous iteration.
- Find the concepts related to the selected type of object based on the tags.
- Define labeling functions to program the filtered concepts and generate weak labels.
- Retrain the model with the weak labels and evaluate the model's performance.

Developers will end the process when they are satisfied with the performance improvement.

6.1.2 Evaluation Results

Iteration 1: Programming for *Bus*

The developers first check the model's performance as shown in Table 1, where the first row is the performance of the state-of-art (SOTA) unsupervised approach [34] and the second row is our baseline model with only self-supervision. They find that the performance of ground transportation such as bus and train doesn't improve much compared with SOTA. Hence, they decide to start with a dominant category of *Bus*. Here, concept $C176$ (Figure 8a) is used as an example, which mostly captures body of a bus and is tagged as bus and truck (Figure 8b-c). However, by examining the image examples of $C176$, developers find this concept also contains the body of trains (Figure 8d) and cars (Figure 8e). This finding triggers them to build labeling functions to separate the three types of ground transportation in $C176$.

To build labeling functions for trains, developers examine the concepts that relate to $C176$ and identify $C90$ is corresponding to *rails*. Then, they build a labeling function indicating that a transportation running above rail is more likely to be a train instead of a bus:

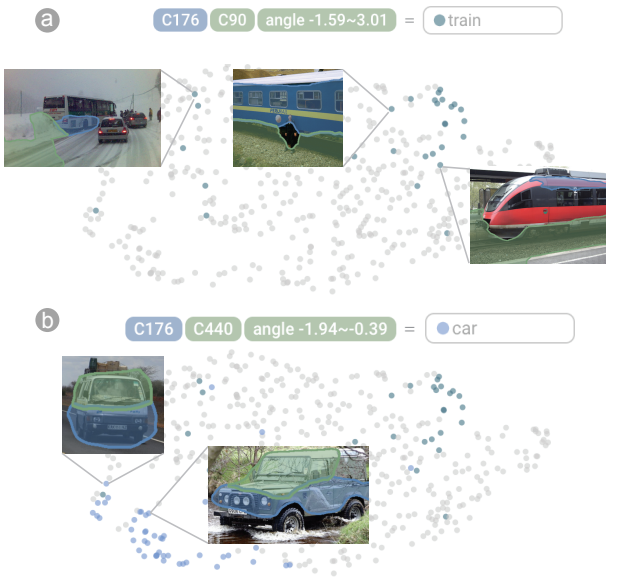

Fig. 9: Labeling functions to annotate (a) trains and (b) cars in Concept C176 and related sample images.

$$C176(GroundTransportation) + angle[-0.06, 30.1](above) + C90(Rail) = train$$

The image segments that matches this labeling function are then labeled as *train* automatically. The annotation results are shown in the scatter plot in Figure 9a, where the dark blue dots are image segments that satisfies the labeling function. We can see that most of the trains are in the top right of the projection space. Meanwhile, there are a few outliers in other regions of the projection space, such as the bus driving in snow, whose label can be corrected by lasso selection of those data points. Similarly, labeling functions can be used to annotate cars in $C176$. For example, developers find a concept of car window ($C440$) and use it to build a labeling function:

$$C176(GroundTransportation) + angle[-3.11, 0.39](below) + CC440(CarWindow) = car$$

that annotates the cars as shown in Figure 9b. After identifying trains and cars, the rest segments are labeled as bus.

Iteration 1: Performance after *Bus* Programmed

After programming the concepts related to buses, the model is trained on the generated labels and tested on the validation dataset, which takes around 2 and 1.5 hours, respectively. The results are shown in the third row of Table 1. We can see that as the developers inject knowledge about classes of bus, car, and train, the performance of the three classes improved by a large margin comparing with the strong baseline ($+1.17$, $+1.62$, and $+2.17$ for bus, car, and train, respectively). However, we also observe that the overall performance drops. This is mainly because the noise introduced by the weak labels affects the model's performance on classes without annotations. Hence, the developers decide to program other classes to alleviate the influence of the noise by adding more supervision to guide the training of other classes.

By comparing the performance of the current model with the baseline, the developers find that the performance of farm animals such as cow, horse, and sheep drops significantly. Hence, they decide to program the concepts related to one type of farm animal such as cow in the second iteration.

Iteration 2: Programming for *Cow*

After examining the concepts relate to cow (Figure 10a), developers find that different concepts capture different parts of animals such as head, body, and foot. However, within each concept, there could be parts from different types of animals such as cow, horse, sheep, and dog. Hence, different labeling functions are built to annotate each type of these animals. For animals head and body, developers find that tags and embedding projections can be used to classify different types of animals. For animals foot, it is a bit challenging to use the tags or

Table 1: Performance improvement with visual concept programming on the PASCOL VOC 2012 dataset, measured by *IoU* (larger is better).

	background	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	all
SOTA [34]	86.00	62.99	23.63	56.08	45.28	61.31	74.68	71.33	64.89	6.70	46.55	35.21	63.80	46.67	58.83	58.03	32.53	60.07	29.69	67.17	47.95	52.35
baseline	91.02	80.47	33.93	80.42	59.93	67.15	81.94	74.00	83.25	19.00	52.45	50.85	76.12	54.41	66.89	75.63	37.97	65.57	34.47	70.07	58.34	62.56
ours (bus)	90.98	77.36	32.03	77.65	58.58	67.73	**83.11**	**75.62**	79.68	18.45	50.02	50.08	69.61	54.63	63.07	76.71	31.5	58.17	34.00	**72.24**	57.73	60.91
ours (bus + cow)	91.01	77.77	32.78	78.48	56.69	68.46	83.93	75.9	82.29	18.99	**59.81**	47.81	74.24	**58.87**	64.06	75.58	31.9	63.07	34.53	70.83	59.71	62.22
ours (bus + cow + plant)	91.05	76.8	33.69	79.99	57.94	67.12	82.4	76.61	81.99	17.71	58.67	49.94	74.6	58.48	64.74	76.63	**41.6**	64.93	35.1	71.31	60.56	62.95
ours (all)	91.32	79.45	33.8	82.44	62.68	70.09	84.82	76.2	84.88	18.58	54.12	53.6	78.79	57.15	68.31	76.09	44.4	63.31	40.58	76.23	64.21	64.81
with ground truth	91.50	79.93	33.83	82.09	63.32	69.57	85.80	76.21	86.34	22.3	58.88	55.29	80.8	59.14	69.03	77.54	46.52	65.94	39.03	76.25	62.55	65.80

Table 2: Number of concepts, labeling functions, and coverage for bus, cow, plant, and all classes. The ratio between the number of labeling functions and the segments it can label is used to measure the relative effort of creating the labeling functions.

	#concepts	#labeling functions / relative effort	coverage (#segments / %)
bus	17	137 / 0.020	6965 / 7%
cow	25	229 / 0.023	9573 / 9%
plant	15	65 / 0.017	3752 / 4%
all	512	1363 / 0.013	102445 / 100%

Fig. 10: (a) Example of concepts related to cow. (b)-(d) Example labeling functions to label different body parts of animals.

the embedding projections of it's own because feet of cow, horse, and sheep are hard to differentiate. Hence, developers combine feet with head/body and use the features (e.g., tags) of head/body to label them (examples are shown in Figure 10b-d).

Iteration 2: Performance after *Cow* Programmed

After training the model on the new labels, the results are shown in the 4th row of Table 1. Developers find that the performance of cow and horse are improved significantly comparing with the baseline (+7.36 and +4.46 for cow and horse respectively). The performance of sheep is also improved comparing with the previous iteration.

Iteration 3: Programming and Improving *Plant*

As the performance of *plant* drops the most in the previous iteration, developers program concepts related to *plant* in the 3rd iteration. By retraining the model with the annotations on *plant*, they observe that the performance of *plant* improved by a large margin comparing with previous interactions. More importantly, the overall performance of the model starts to outperform the baseline in this interaction (5th row of Table 1).

Iteration N: Final Results

After around 10 iterations, all visual concepts have been labeled and the final results are shown in the 6th row of Table 1. We can see that the overall performance of the refined model outperforms the baseline more than 2.3% in terms of IoU. For comparison, we also annotate all image segments using the ground truth label and train a model on the annotated data. The results are shown in the 7th row of Table 1. We can see that our method performs only slightly worse than the model trained on ground truth annotations.

The developers are also interested in the quality of the visual concepts with respect to the downstream task after programming. To this

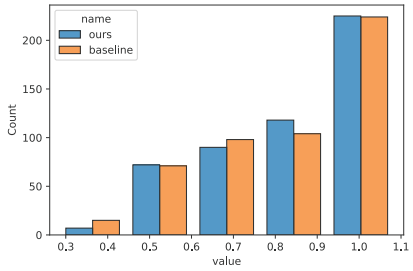

Fig. 11: Comparison of purity distribution between the baseline and our approach. The number of concepts with purity less than 0.7 is decreased significantly using our approach.

end, we measure the *purity* of each visual concept as follows. We find the major class of each visual concept based on the ground truth and calculate the *purity* as the percentage of image segments that belong to the major class. Figure 11 compares the purity distribution between the baseline and our approach. The developers find that the number of concepts with low purity (i.e., purity less than 0.7) is decreased by 7% with our approach.

In addition, Table 2 shows the statistics about the labeling functions. Comparing with drawing and labeling the image segments manually, annotating the image segments with labeling functions is more efficient (e.g., the ratio between the number of labeling functions and the image segments it can label is around 0.02).

Model Iteration Summary

With visual concept programming, developers are able to improve: a) *model performance* on downstream semantic segmentation task by over 2.3% and b) *concept quality with respect to the downstream task* by reducing the number of low purity concepts by 7%. Moreover, the effort to get the performance gain is much lower than manually annotating the images.

6.2 Scenario Creation and Model Validation

Our approach can be applied on a broader range of applications beyond data labeling. In this case study, we demonstrate how to create test scenarios and validate a semantic segmentation model using our tool.

Dataset We worked with the developers on an internal autonomous driving dataset, which contains tens of thousands of driving scene images captured from fleets. There are no semantic segmentation labels created for the dataset because of the cost. Hence, the developers cannot use the ground truth labels to evaluate the model and exam the failure cases.

Scenario Creation To help developers understand the data, we apply our method on it and use visual concepts to create scenarios and retrieve images of their interest. Figure 12 shows one example of scenario creation. In this example, developers want to create scenarios that relates to person and traffic light. To this end, developers first identify the person and traffic light concepts in the concept rank view (Figure 12a). Then they combine the two concepts and add more constrains to create specific scenarios. For example, developers limit the distance between a person and traffic light such that they can retrieve images that contains a person standing very close to a traffic light (Figure 12b). Many other scenarios can be created with different combinations of visual concepts.

Model Validation With the ability of creating different scenarios and retrieve the corresponding images, developers can use the scenar-

This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2022.3209466

Fig. 12: Example of scenario creation and image retrieval. (a) Concepts of person and traffic light are identified. (b) and (c) Constrain the distance between the two concepts to create scenarios such as person standing under a traffic light.

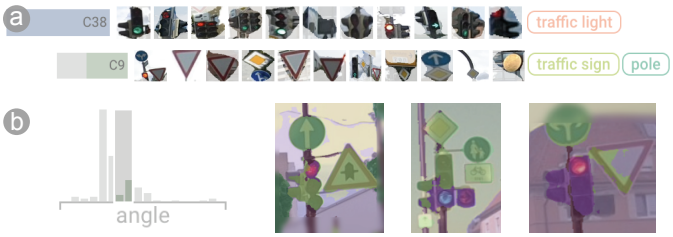

Fig. 13: Example on model validation. (a) Concepts of traffic light and traffic sign are selected to test a semantic segmentation model. (b) The performance of the model is poor while a traffic sign (green) is located on the top right of a traffic light (purple).

ios to test the model's performance. To efficiently evaluate model's performance, we combine the model's prediction results with the retrieved images. Figure 13 shows an example of model validation results. Developers examined the combination of traffic light and traffic sign to test the model. They found that while a traffic sign is located on the top right of a traffic light, the model is not able to segment traffic lights (purple) and traffic signs (green) accurately. While the model performs well as a traffic sign is located on the top of a traffic light. The developers found the results very interesting as they never thought that the relative position between the traffic light and traffic signs would affect model's performance dramatically.

6.3 Domain Experts Feedback

We conduct two case studies (Section 6.1 and 6.2) with the three MLOps developers introduced in 6, who have over 5 years of experience in operating ML models for autonomous driving. We work with the developers to come up with labeling functions using our tool and refine models based on the weak labels. We conclude the studies with open-ended interviews and discussions, and summarize the developers' feedback from the following perspectives.

Efficient Concept Exploration Overall, the developers feel our tool "*is really needed*" for exploring large scale image data and creating scenarios of interest. They liked the idea of summarizing image data with a concept dictionary, which is "*straightforward and explainable*". In particular, they found "*the tags are important to help them understand and explore the concepts*". In addition, the developers liked the way of combining concepts to create specific scenarios and they found it is "*quite beneficial*" for retrieving images from large scale data without labels. Moreover, they also thought using the created scenarios for model validation is "*very helpful*".

Scalable Data Annotation with Concept Programming Overall, developers "*liked*" the idea of annotating data by programming visual concepts. As the developers often work with large amounts of unlabelled data, having a tool to inject their knowledge into data is "*needed*" for model development. Even the automatically generated labels are not 100% accurate, it is already "*very useful*" to test developers' hypothesis and help them quick identify data that require labels. The developers were also "*impressed*" by the performance improvement.

Improvement The developers also suggested improvements in terms of the visual analytics system. Overall, they feel "*the system takes a few moments to digest*". In particular, they thought "*it is not immediately understandable how the concepts are ranked*". It would be better to provide some hints on it and give more options on how to rank the concepts. They also would like to have the functionality that let them "*rank the labeling functions*".

7 Discussion, Limitations, and Future Work

How to select the most valuable data to label Although visual concept programming has reduced the effort to label data, developers still need to explore and define labeling functions for hundreds of concepts. Hence, it is an important and interesting question to identify the visual concepts that influence model performance the most. This can be linked to another thread of work in Data-centric AI called data valuation, which aims to identify data points that contribute the most to the accuracy/mistakes of AI models. In the future, we would like to quantify the value of each visual concept such that developers can focus on high value concepts to get performance gain with less effort.

How to provide interactive feedback The current approach lacks interactive feedback on how the labels will affect model performance as it often takes hours to retrain a model. In the future, we would like to explore methods for approximating the performance change after data labeling. We believe it could all benefit concept and data selection.

How to select proper labeling functions The quality of labeling functions has high impact on the accuracy of weak labels and hence the model's performance. In this work, developers need to define labeling functions by examining different combinations of concepts as well as features such as tags and embedding projections. In the future, we would like to study the importance of the features for concepts and give experts suggestions/templates on how to compose a labeling function.

How to balance the self- and weak-supervision As weak labels generated by visual concept programming are often noisy and only cover a subset of data, self-supervision still plays an important role in model training. Hence, it worth to explore that how the self- and weak-supervision affect each other and how to fuse them during training.

Improving model performance on downstream tasks We notice that the proposed approach depends on a series of upstream models such as concept extraction and tagging models, which may introduce noise to the data. Although we have demonstrated performance improvement using the noise labels, there are several directions we want to explore to further improve our approach. On one hand, we would like to improve the performance of upstream models, such as more accurate image segments for concepts. On the other hand, we would like to explore training methods that take label noises into account.

Other future work We would like to improve the visual analytics system based on the developers' feedback, including providing more options to rank the concepts, enabling them to compare and rank different labeling functions, and analyze the relationships with more than two concepts.

8 Conclusion

This is a first-of-its-kind work that combines visual anlytics with data-centric AI to inject human knowledge into image data at scale. The introduced Visual Concept Programming approach uses self-supervision to learn pixel-wise embeddings and extract visual vocabulary, namely a dictionary of visual concepts, from unlabeled image data. Then a visual analytics tool is designed to explore the concepts and create labeling functions to program the concepts. With the labeling functions, human knowledge can be injected into data at scale by generating weak labels for pixels automatically. The weak labels are then used to improve the pixel embeddings for downstream tasks. We hope this work can trigger some new research questions of applying visual analytics to inject human knowledge into large-scale data efficiently with the help of powerful representation learning (e.g., self-supervised visual concept extraction) for human cognition augmentation.

REFERENCES

[1] Data-centric ai resource hub. https://datacentricai.org/. Accessed on 25 March, 2022.

[2] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6541–6549, 2017.

[3] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[4] J. Bernard, M. Hutter, M. Zeppelzauer, D. Fellner, and M. Sedlmair. Comparing visual-interactive labeling with active learning: An experimental study. *IEEE transactions on visualization and computer graphics*, 24(1):298–308, 2017.

[5] O. Chapelle, B. Scholkopf, and A. Zien. Semi-supervised learning. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

[6] C. Chen, O. Li, D. Tao, A. Barnett, C. Rudin, and J. K. Su. This looks like that: deep learning for interpretable image recognition. *Advances in neural information processing systems*, 32, 2019.

[7] V. S. Chen, P. Varma, R. Krishna, M. Bernstein, C. Re, and L. Fei-Fei. Scene graph prediction with limited labels. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2580–2590, 2019.

[8] M. Choi, C. Park, S. Yang, Y. Kim, J. Choo, and S. R. Hong. Aila: Attentive interactive labeling assistant for document classification through attention-based deep neural networks. In *Proceedings of the 2019 CHI conference on human factors in computing systems*, pp. 1–12, 2019.

[9] M. Desmond, M. Muller, Z. Ashktorab, C. Dugan, E. Duesterwald, K. Brimijoin, C. Finegan-Dollak, M. Brachman, A. Sharma, N. N. Joshi, et al. Increasing the speed and accuracy of data labeling through an ai assisted interface. In *26th International Conference on Intelligent User Interfaces*, pp. 392–401, 2021.

[10] X. Ding, Y. Zhang, T. Liu, and J. Duan. Deep learning for event-driven stock prediction. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[11] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[12] A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, and J. Dean. A guide to deep learning in healthcare. *Nature medicine*, 25(1):24–29, 2019.

[13] A. Ghorbani, J. Wexler, J. Y. Zou, and B. Kim. Towards automatic concept-based explanations. *Advances in Neural Information Processing Systems*, 32, 2019.

[14] L. Gou, L. Zou, N. Li, M. Hofmann, A. K. Shekar, A. Wendt, and L. Ren. Vatld: a visual analytics system to assess, understand and improve traffic light detection. *IEEE transactions on visualization and computer graphics*, 27(2):261–271, 2020.

[15] W. He, W. Surmeier, A. K. Shekar, L. Gou, and L. Ren. Self-supervised semantic segmentation grounded in visual concepts. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 949–955, 2022.

[16] W. He, L. Zou, A. K. Shekar, L. Gou, and L. Ren. Where can we help? a visual analytics approach to diagnosing and improving semantic segmentation of movable objects. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):1040–1050, 2021.

[17] F. Hohman, M. Kahng, R. Pienta, and D. H. Chau. Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE transactions on visualization and computer graphics*, 25(8):2674–2693, 2018.

[18] F. Hohman, H. Park, C. Robinson, and D. H. P. Chau. Summit: Scaling deep learning interpretability by visualizing activation and attribution summarizations. *IEEE transactions on visualization and computer graphics*, 26(1):1096–1106, 2019.

[19] J.-J. Hwang, S. X. Yu, J. Shi, M. D. Collins, T.-J. Yang, X. Zhang, and L.-C. Chen. Segsort: Segmentation by discriminative sorting of segments. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7334–7344, 2019.

[20] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, et al. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International conference on machine learning*, pp. 2668–2677. PMLR, 2018.

[21] J. Li, H. Ding, J. Shang, J. McAuley, and Z. Feng. Weakly supervised named entity tagging with learnable logical rules. In *Proceedings of the Association for Computational Linguistics*, pp. 4568–4581, 2021.

[22] S. Liu, C. Chen, Y. Lu, F. Ouyang, and B. Wang. An interactive method to improve crowdsourced annotations. *IEEE transactions on visualization and computer graphics*, 25(1):235–245, 2018.

[23] S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., *Advances in Neural Information Processing Systems 30*, pp. 4765–4774. Curran Associates, Inc., 2017.

[24] L. McInnes, J. Healy, N. Saul, and L. Grossberger. Umap: Uniform manifold approximation and projection. *The Journal of Open Source Software*, 3(29):861, 2018.

[25] A. Ng, D. Laird, and L. He. Data-centric ai competition. 2021. Available online: https://https-deeplearning-ai.github.io/data-centric-comp. Accessed on 9 December, 2021.

[26] H. Park, N. Das, R. Duggal, A. P. Wright, O. Shaikh, F. Hohman, and D. H. Chau. Neurocartography: Scalable automatic visual summarization of concepts in deep neural networks. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 2022. doi: 10.1109/TVCG.2021.3114858

[27] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision, 2021. doi: 10.48550/ARXIV.2103.00020

[28] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, vol. 11, p. 269. NIH Public Access, 2017.

[29] A. J. Ratner, C. M. De Sa, S. Wu, D. Selsam, and C. Ré. Data programming: Creating large training sets, quickly. *Advances in neural information processing systems*, 29, 2016.

[30] I. Redko, E. Morvant, A. Habrard, M. Sebban, and y. Bennani. *Advances in Domain Adaptation Theory*. Elsevier, 2019.

[31] M. T. Ribeiro, S. Singh, and C. Guestrin. "why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016.

[32] W. Samek, G. Montavon, A. Vedaldi, L. K. Hansen, and K.-R. Müller. *Explainable AI: interpreting, explaining and visualizing deep learning*, vol. 11700. Springer Nature, 2019.

[33] H. Strobelt, S. Gehrmann, H. Pfister, and A. M. Rush. Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE transactions on visualization and computer graphics*, 24(1):667–676, 2017.

[34] W. Van Gansbeke, S. Vandenhende, S. Georgoulis, and L. Van Gool. Unsupervised semantic segmentation by contrasting object mask proposals. *arXiv:2102.06191*, 2021.

[35] P. Varma, B. He, P. Bajaj, N. Khandwala, I. Banerjee, D. Rubin, and C. Ré. Inferring generative model structure with static analysis. In *Proceedings of Advances in Neural Information Processing Systems*, p. 239–249, 2017.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of Advances in Neural Information Processing Systems*, p. 6000–6010, 2017.

[37] J. Wexler, M. Pushkarna, T. Bolukbasi, M. Wattenberg, F. Viégas, and J. Wilson. The what-if tool: Interactive probing of machine learning models. *IEEE transactions on visualization and computer graphics*, 26(1):56–65, 2019.

[38] S. Xiang, X. Ye, J. Xia, J. Wu, Y. Chen, and S. Liu. Interactive correction of mislabeled training data. In *2019 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 57–68. IEEE, 2019.

[39] J. Zhang, C.-Y. Hsieh, Y. Yu, C. Zhang, and A. Ratner. A survey on programmatic weak supervision, 2022. doi: 10.48550/ARXIV.2202.05433

[40] Y. Zhang, Y. Wang, H. Zhang, B. Zhu, S. Chen, and D. Zhang. Onelabeler: A flexible system for building data labeling tools. In *CHI Conference on Human Factors in Computing Systems*, pp. 1–22, 2022.

[41] Z. Zhao, P. Xu, C. Scheidegger, and L. Ren. Human-in-the-loop extraction of interpretable concepts in deep learning models. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):780–790, 2021.