An Introduction to Rule-Based Feature and Object Perception for Middle School Students

Daniella DiPaola[1], Parker Malachowsky[1], Nancye Blair Black[2], Sharifa Alghowinem[1], Xiaoxue Du[1], Cynthia Breazeal[1]

[1] MIT Media Lab, Massachusetts Institute of Technology
[2] Teacher's College, Columbia University
dipaola@media.mit.edu, pmalacho@media.mit.edu, nwb2111@tc.columbia.edu, sharifah@media.mit.edu, xiaoxued@media.mit.edu, cynthiab@media.mit.edu

Abstract

The Feature Detection tool is a web-based activity that allows students to detect features in images and build their own rule-based classification algorithms. In this paper, we introduce the tool and share how it is incorporated into two, 45-minute lessons. The objective of the first lesson is to introduce students to the concept of feature detection, or how a computer can break down visual input into lower-level features. The second lesson aims to show students how these lower-level features can be incorporated into rule-based models to classify higher-order objects. We discuss how this tool can be used as a "first step" to the more complex concept ideas of data representation and neural networks.

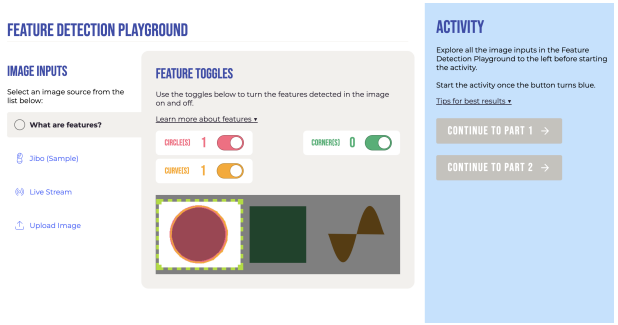

Figure 1: Welcome screen of the Feature Detection Tool

Introduction

Today's students are growing up in a world in which computer vision algorithms are widespread. For instance, popular social media apps such as TikTok and Snapchat use vision algorithms to detect faces for silly face filters; image storing apps such as Google Photos and Dropbox use computer vision to make it easier to sort through your photos; self-driving cars use computer vision to detect road signs; and vision techniques have been applied to medical imaging to diagnose cancers. Today's students will be a part of a generation that will make vital decisions regarding how vision algorithms are used in a wide variety of applications.

Numerous computational approaches have been developed for image recognition. Traditional image recognition systems use feature engineering, which is a process in which engineers hand-select the important lower-level features that make up an object (i.e., circles, curves, and edges). A handful of tools exist to teach students object recognition through deep neural networks (Tang et al. 2019; Carney et al. 2020; Jordan et al. 2021), but there is a lack of tools that allow them to create their own rule-based systems.

Teaching students about rule-based systems provides scaffolding to understand neural networks, as a neural network can be reasonably thought of as a rule-based system designed by a machine learning algorithm based on a training set. More specifically, it is helpful to have a mental model of neural networks as hierarchical, rule-based systems where a machine learning algorithm iterates on feature engineering and rule creation over training samples. By offering students the chance to leverage feature detection in the creation of their own rule-based object detection systems, we lay the ground work for them to intuitively understand how a neural network can accomplish a similar task.

In this paper, we present the *Feature Detection Tool*, an interactive learning tool that introduces middle school students to rule-based image recognition (Figure 1). The tool is divided into two parts: the first allows students to visualize and count features (circles, curves, and edges) in an image or video feed. In the second part of the tool, students create rules based on these features to classify different objects.

Background

Computer Vision Brief Overview

Images, similar to any other data types, are complex. Before the advancement of supervised machine learning algorithms, such as deep learning, mathematical algorithms were used to extract meaningful information about the image, known as feature engineering (see Figure 2). The process of feature engineering starts with expert consultation and brainstorming of the most representative features of each class/object in an image before implementing the actual extraction algorithm. Intuitively, sample features include things such as color, brightness, motion, edges, contours, illumination gradients, and more. This handcrafted process attempts to identify the right set of features for the image recognition task

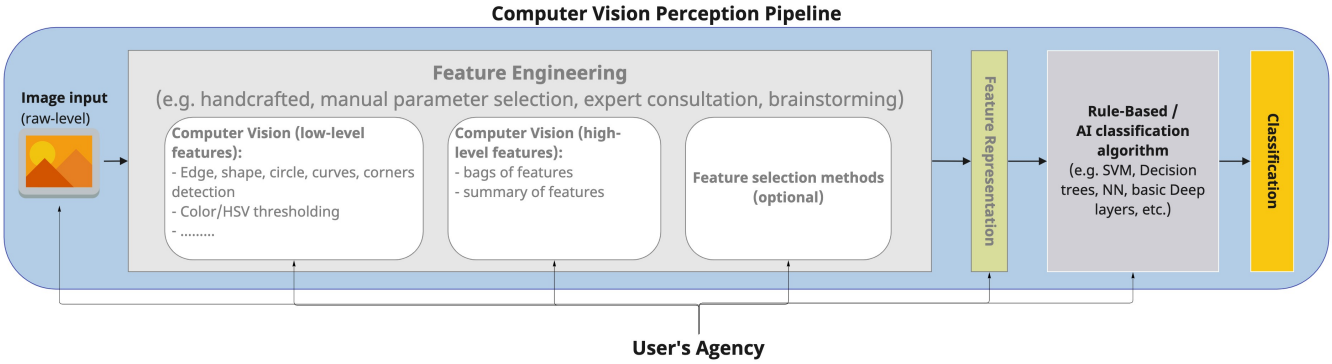

Figure 2: Computer Vision Perception for Object Detection using Feature Engineering Methods, Highlighting User Agency

that have the potential to generalize to recognize similar objects in different images.

Simple features such as edges, curves, corners, etc. are detected by using different thresholds in color spaces (see section on Technical Details of Feature Detection Interface). More advanced, feature detection algorithms can be used, such as bag-of-visual-words algorithms (Yang and Newsam 2010).

Once these features are extracted, they are combined to recognize more complex objects in the image and the relationship between them. Knowledge-based (or rule-based methods) translate human knowledge about an object's features into rules that can can be applied by an algorithm to classify instances of that object in an image (Yang, Kriegman, and Ahuja 2002). For instance, one might create a rule that a face must have two symmetric eyes, a nose, and a mouth. Another rule might incorporate the relationship between features and the relative distances and position. A disadvantage of this method is that the rules might be too specific or too general. The system may fail to detect the object, or may result in a false positive detection (such as mistaking a cloud pattern for a face).

Older supervised learning algorithms (e.g., Support Vector Machines, or "shallow" Neural Networks), use these engineered features as input (a feature vector) coupled with their labels to learn patterns about an object in the image (i.e., a class). In this method, similar to any supervised training of a model, the more comprehensive and diverse the training data set is (i.e., inclusion of different lighting conditions, object occlusions, sizes, orientations, etc.), the more robust the results will be. The user's understanding of the image data and the visual features that are relevant for object detection are key to these algorithms' success.

Modern supervised machine learning methods, such as deep learning applied to computer vision tasks, require far less human insight about visual features, etc. In deep learning, the algorithm takes the raw image as input (rather than engineered features). Visual features are learned and extracted through the first few layers in the architecture. Even though the layers use kernels to extract features, the extracted features are also influenced (weighted) by the data and the classes to find the best representative features for each class (Nixon and Aguado 2019), which are then used by deeper layers to make a decision about the presence of an object in the image. The person training the deep network has little control of what or how the model is learning, except by providing more, correctly labelled training data, varying the number of layers, etc. This is in contrast with rule-based models, where the creator of the algorithm plays a significant role in how the vision algorithm is implemented.

Children's Usage of Computer Vision Tools

The AI4K12 Initiative - a partnership between the Association for the Advancement of Artificial Intelligence and the Computer Science Teachers Association - has been developing national guidelines for AI education in grades K-12, as summarized through the "Five Big Ideas in AI" (Touretzky et al. 2019). Big Idea #1 focuses on machine perception, the idea that "computers perceive the world using sensors" (Touretzky et al. 2019).

Computer vision is an important way that machines try to interpret and interact with the world using cameras and other types of image sensors. In the "Draft Big Idea 1 - Progression Chart,"[1] AI4K12 details several related concepts and learning objectives that build K-12 students' understanding of how AI extracts meaning from sensory data through computer vision. These include understanding the process of feature extraction and how extraction proceeds from lower- to higher-level feature detection, as well as the exploration of the differences between computer and human sensing and of specialized AI algorithms for tasks like computer vision.

Several tools are already used to teach K-12 students about various aspects of computer vision. For example, Teachable Machine allows K-12 students to create their own image classification models using supervised machine learning (Carney et al. 2020). While users are involved at the image labeling stage, Teachable Machine does not allow for any degree of feature engineering to tweak the accuracy or see the inner workings of the algorithm. Students can similarly use an end-user tool, like Google Lens, to see a pretrained image classifier in action and explore the benefits and

[1] https://ai4k12.org/big-idea-1-overview/

limitations of the technology [2].

While both of these resources demonstrate object recognition, they do not reveal the feature patterns detected, or rules for classification established by the model, or the location of recognized object in the image (with bounding boxes or otherwise). In fact, when using Teachable Machine, it's possible for students to believe they are training the AI to recognize one thing, such as the hand gesture for peace, when the AI is technically recognizing something else, such as an accompanying smile. This can also lead to a common misconception that machine vision systems observe the same types of high-level visual features that humans do, rather than pixels and lower-lever features. To better understand AI perception, students need a peek inside a simpler black box to scaffold their understanding of more modern machine vision algorithms. Tools exist to make these models more transparent, but are typically developed for older audiences (Wang et al. 2020; Karpathy 2016; Makwana et al. 2020).

Other computer vision tools have limitations in illuminating for students how AI perception works. Seek by iNaturalist[3] or Google Translate [4] demonstrate exciting real-world applications of computer vision. However, they do not give students access to training data or the agency to intervene in the training process. Google's QuickDraw[5] AI Experiment reveals its vast training image dataset, but does not illuminate the specific feature patterns the model found in its images.

Some *Machine Learning as a Service (MLaaS)* tools also hold promise for teaching about computer vision (Black and Brooks-Young 2020). For example, the Google Could Vision API[6] has a free demo that uses machine learning to analyze uploaded photos with face detection, emotion recognition, object recognition, and more. The interface displays bounding boxes to show where these objects are detected in the image. However, for students it is a black box. Likewise, the free online MLaaS tool, Microsoft Azure Custom Vision API, has an easy-to-use interface for uploading images, creating classes, drawing bounding boxes to train a model, and exporting the model for use in other applications, but it does not support a live feed from a webcam. It also requires signing up with a credit card, a clear barrier to K-12 students.

Finally, two free tools give students the agency of developing block-based programs that include computer vision models. Poseblocks builds on the open source Scratch platform to provide models for computer vision applications like hand tracking and emotion recognition (Jordan et al. 2021). Similarly, an extension for MIT's App Inventor allows students to integrate image classification models they trained with webcam images (Tang et al. 2019). While these are powerful tools for understanding the potential of AI perception, neither provide the additional layer of insight into feature detection and the development of rules for object recognition. Given the limitations of all of the previously available tools, there are many learning objectives around perception that could be best illuminated through the development of additional tools and simulations, specifically those that allow students to design the vision algorithms themselves.

[2]https://lens.google/

[3]https://www.inaturalist.org/

[4]https://translate.google.com/

[5]https://quickdraw.withgoogle.com/

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

Feature Detection Tool

Please navigate to the following link to see the tool in action: https://tinyurl.com/featuredetectiontool

Tool Summary

The Feature Detection Tool is a two-part, web-based tool that allows the user to (manually) use feature perception to solve an object detection task. The activity centers around the concept that objects can be identified by a unique signature of their feature counts (i.e., how many circles, corners, and curves are detected in images of them). This is certainly not a perfect assumption, which leads to weaknesses in the object detection algorithms the students develop using this tool. The tool intentionally trades off the accuracy and robustness of more sophisticated objected detection models (e.g., neural networks) in order to offer students more agency in the development and tuning of the features and rules of their computer vision object recognition algorithms.

The Feature Detection Tool serves as the interface for this development, first educating students on what feature perception is, and then offering tooling to configure feature count signatures for different objects. Finally, the Feature Detection Tool allows students to test out their algorithms and, when a bug arises, to continually adapt and improve them.

Learning Objectives

Students should gain the following understandings by using the Feature Detection Tool:

1. Students understand that machines can perceive features from visual input. (Part I)
2. Students understand that machines can use the combination of features to detect more complex objects. (Part II)
3. Students build their own rule-based models using features to detect objects. (Part II)

These learning outcomes are aligned with AI4K12's suggested learning objectives (as of September 2022) for grades 6-8. They suggest that 6-8 students should understand that *"the transformation from signal to meaning takes place in stages, with increasingly abstract features and higher level knowledge applied at each stage"*. Specifically, this activity addresses the following 6-8 learning objectives from AI4K12:

1. 1-B-ii - Illustrate the concept of feature extraction from images by simulating an edge detector.
2. 1-B-iv - Describe how edge detectors can be composed to form more complex feature detectors, e.g., for letters and shapes.

Target Age Group

Middle School Students (Grades 6-8)

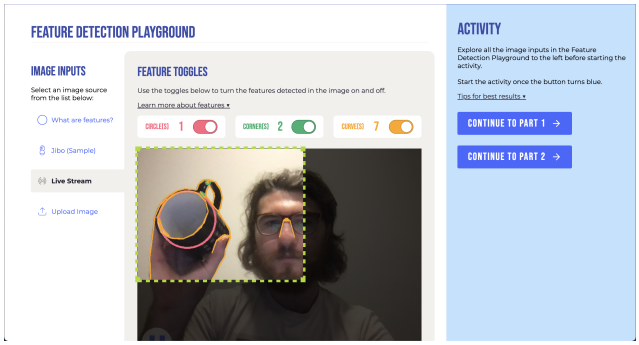

Figure 3: In Part I, users manipulate the window view and get a count of circles, corners, and curves.

Time Needed

90 Minutes

Materials Needed

To use the tool, students must have access to a computer that has web access and a video camera. Students will also need 3-5 examples of common objects (i.e. three mugs, three backpacks, three books).

Prerequisite Knowledge

No prerequisite knowledge is needed to use this tool. The lesson can be done on its own, but we suggest including it in a progression after learning about data and before learning about machine learning.

Part I: Introduction to Features

In Part I, students are introduced to the idea of *features*, or characteristics of data. The Part I interface allows students to get real-time feedback on the amount of circles, curves, and corners in an image or video feed. This feedback enables students to experiment with different visual stimuli in order to understand how the tool detects different features.

Once they are comfortable with the concept of feature detection, students are required to identify three object classes — these should be common objects (ideally from the classroom), like backpacks, books, water bottles, etc. They should find 3-5 examples of each object. The student will then place each example inside of the feature detection window (discussed below) and record how many of each feature is detected on a sheet of paper.

Part I Interface

The feature detection interface is designed to allow the student to explore the detection and visualization of features at their own pace before starting the activity. This is accomplished through several interface components, which can be seen in Figure 3.

First, the student is given four choices of image sources to use as input to the feature detection algorithm. After processing a frame from the input source, the detected features are overlaid directly on top of the input for ease of understanding. The first source (and the one visible on page load) is a simple 2D rendering of three basic shapes: a circle, a square, and a shaded sine wave. This source offers a quick intuition about where and when certain features will be detected. The second source a digital photograph, it offers students a glimpse into how feature detection can be less predictable when given a complex image. Next, the student can use their live webcam feed as an input source (with the feature detection algorithm running roughly every ten frames). This source is intended to be used throughout the activity. With their learning from the first two sources, students should be prepared to make sense of the dynamic results of running feature detection on their webcam video feed. If a student cannot use a webcam, they can use the fourth and final input source and upload their own image.

In addition to the feature overlay directly on the input source, students are always able to see a count of exactly how many of each feature is being detected. The visualization of each feature can also be toggled on and off, allowing a student to probe features one by one.

A key aspect of the feature detection experience is leveraging the Feature Detection Window. This offers students a draggable and resizable frame in which features are detected. The feature detection window allows students to hone in on certain areas of the frame to better understand how they are perceived by the feature detection algorithm. Additionally, it allows them to block out areas of the input source that are not relevant to the object detection task (which is especially necessary when using the webcam input source where it is difficult to control the background).

Technical Details of Feature Detection Interface

In order to create a responsive experience, the tool was built using the Svelte web framework. Circle, corner, and curve (i.e., contour) detection is accomplished entirely in the browser using OpenCV.js. To avoid freezing the user's window during the heavy processing of feature detection, we isolate each feature detection task into its own thread using Web Workers.

Circle Detection To detect circles in an image (or a selected part of an image) a pre-processing step is performed to reduce the noise in the image, and subsequently reduce the false positive circles. For pre-processing in this tool, we convert the color space to gray, and even though it is recommended to use a Gaussian blur to smooth the image, our experiments with different conditions did not generalize well and therefore we did not apply it. Other adaptive threshold, erode, and dilate methods were also explored, but we chose to focus on a generalized and stable pre-processing method that would work in any classroom and lighting condition. We used the Hough Gradient Method (Yuen et al. 1990) for circle detection. The Hough method first detects edges in an image and then tries to fit a circle by finding a center with a specific radius that connects most detected edges (even if they are broken or jagged).

Corner Detection Similar to circle detection, the pre-processing step in corner detection includes converting the color space to gray, as well as median blur which was more stable than other methods for our generalization purposes.

Median blur methods were used to smooth the image be applying a median filter in each pixel based on the neighboring pixels to reduce the noise, which in return refines the actual corners. We used Harris corner detector (Harris, Stephens et al. 1988), where it finds the local maxima of eigenvalues and eigenvectors from each specified block of pixels. A corner is detected when the force (based on the eigenvalues and eigenvectors) in a pixel change is coming from three different directions. Similar to circle detection, this methodology is sensitive and requires tuning to avoid false positives/negatives, so we employ a ”default” profile of generalized parameters.

Curve Detection For pre-processing step in curve detection, we started similar to the corner detection by converting the color space to gray, followed by a median blur, then we added a Canny edge detection to retain the edges only for curve calculations. We used contours finding algorithm (Suzuki et al. 1985), that uses binary borders to follow curves to detect shapes and analyze images. Finally, we did a post-processing step, where we found that closed curves are counted twice, while open curves are not. Therefore, we corrected the count by detecting the closed curves by comparing the curve length with its area. Similar to circle and corner features, the parameters we selected were the most generalized to the context it would apply to. With some profile adjustments, we anticipate a higher customization for a specific environment.

All the above features are sensitive to the details of the image during detection, for example, a mug with some drawing would yield more curves than a mug without any designs.

Part II: Rule-based Object Detection

Once a student has a grasp on feature detection and visualization, the web tool invites them to apply this knowledge to accomplish an object detection task. In the second part of the activity, students configure rules to identify objects based on how many of each feature is detected. For example, a student might establish that all images of backpacks have 1-3 circles, 4-8 corners, and 10-20 curves. During this process they are effectively creating a regression tree for object detection.

Object Detection Interface

After collecting feature counts for their example objects in Part I, students will continue to the second part of the activity and use the data they collected to form feature ranges that uniquely identify an object class. For example, if a student found that amongst their 5 examples of books the minimum number of detected curves was 5 and the maximum was 10. then they would conclude images of books should have 5-10 detected curves. The object detection interface (Figure 4) allows students to easily configure these ranges for each object category across each feature (using both sliders and input fields).

Once a student is satisfied with their range configurations, they can select to ‘apply’ their rule set. This will then add a text field above the input source that displays a prediction based on the students set ranges (Figure 5). If the current input satisfies each of an object’s feature criteria (meaning all currently detected feature counts fall within the configured range), then the prediction will include that object. It is therefore possible for multiple objects to be included in the predicted value, provided that multiple objects have their feature criteria met.

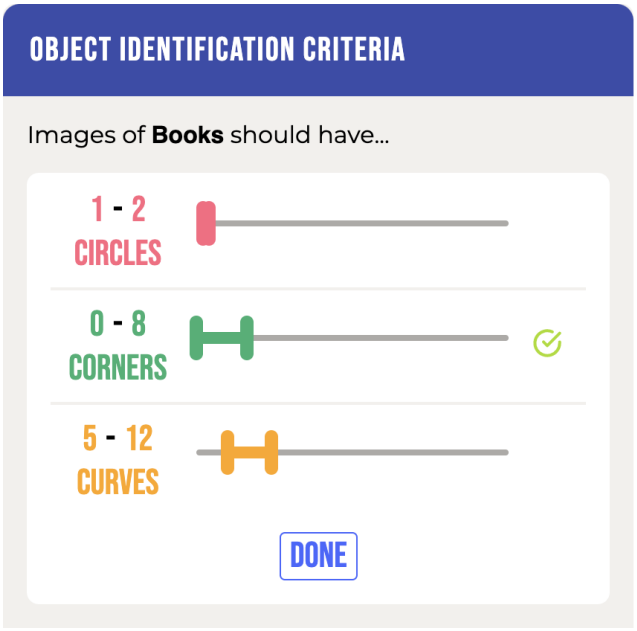

Figure 4: In Part II, users create their own feature-based rules to detect objects.

Discussion

Key Design Features

Some of the design features of the feature perception tool may be helpful for the future design of AI literacy tools.

Student Agency AI and specifically deep neural networks are often called ”black boxes” because their learnt features and internal connections are difficult to parse. This quality makes interpreting the behavior of individual neural networks difficult, but should not limit students in understanding how learning and compositing features is the source of neural networks’ predictive power. AI literacy tools should support this understanding, while also teaching about the challenges that come along with the opacity of ”black box” technology. Though our feature perception tool does not directly involve neural networks, it does offer students a hands-on experience with feature-based object detection, which can then be used as a basis for understanding the inner workings of neural networks applied to machine vision tasks (especially their early layers). The way students combine feature detectors into a higher level decision unit is a close approximation to the relationship between the nodes of two layers of a neural network. Future work should focus on demonstrating this connection.

By testing and changing their algorithms, students can make their own hypotheses and get real-time feedback on

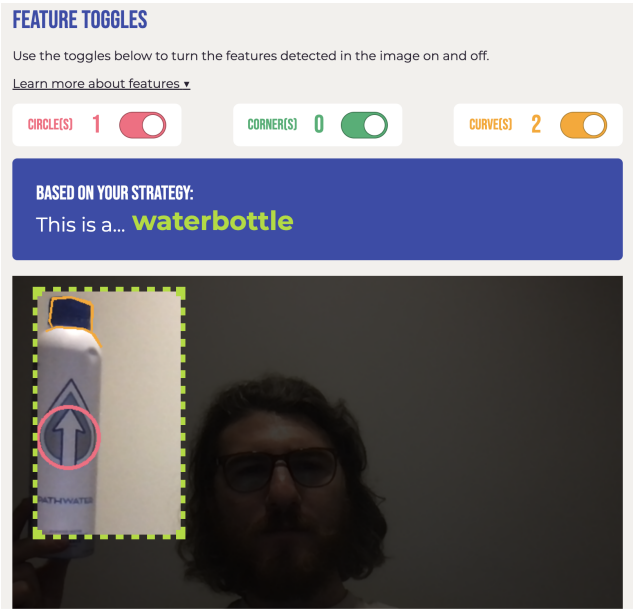

Figure 5: In Part II, users can test out their object recognition algorithms by viewing the number of features detected as well as the object classification.

how the design of their feature criteria affects the outcome of their system. As they test their classifier, they can see the detected features and the predicted objects, which gives them a direct understanding of why their classification algorithm is making a particular decision. This practical understanding can then be used to help students intuitively understand how neural networks use features in data to make decisions as well as how neural networks learn through adjusting their decision-making criteria based on examples.

Importantly, students will also be able to contrast the feature perception tool and neural networks in terms of the agency and control they offer to the user. In a tool like Teachable Machine, the user's only recourse when their trained neural network makes a wrong prediction is to add more labeled data. With the feature perception tool, a student can actually probe and adjust the decision-making process of their algorithm. Therefore, the feature perception tool not only offers students a glimpse into *conceptually* what an object recognition neural network is trying to learn from data, but also sets students up to understand that it is difficult to understand how a *specific* neural network makes a decision.

Computer Feature Feedback Without visualization, learners often rely on their own intuition and perception of features in order to make sense of the feature patterns picked up by computers. For example, learners might focus on abstract features of an object and overlook details that the algorithm considers in its decision. This is especially true with neural networks, where their success in classifying complex objects can mislead us to think their decision making is similar to humans, when it reality they often identify commonalities in data that we as humans completely overlook (like a system that differentiates wolves and huskies based on the presence of snow in the background (Ribeiro, Singh, and Guestrin 2016)). This tool gives visual feedback for where a circle, corner, and curve are detected, allowing students to see the direct features the algorithm is picking up instead of relying on their own assumptions. This design feature allows students to differentiate their understanding of features from the algorithm's understanding of a feature.

Suggested Progression: Scaffolding for Neural Networks

As alluded to in the Student Agency section, this tool is designed to offer students a hands-on introduction to the mechanisms that enable neural networks, especially convolutional neural networks (CNNs). Following this learning experience, students would be poised to explore more complex concepts found in the AI4K12 Big Ideas and guidelines, such as data representation, reasoning, and machine learning. For example, by uncovering AI perception through feature detection and rule-based classification, students are able to transfer this prior knowledge and language into further lessons on data representation and reasoning algorithms, such as the use of decision trees for classification.

Moreover, the mirroring of algorithmic mechanisms for learning and reasoning by neural networks includes: the formulation of a detection rule based on feature perception (analogous to applying a convolution to an input and passing the result through an activation function); the fine-tuning of feature criteria based on examples (analogous to a training algorithm); and the combination of feature criteria to form a higher order representation, in this case an object classification (analogous to the hierarchical structure of neural networks that combine the output of many nodes in a layer to form either an output or an input to the next layer). In this way, the feature perception tool can act as a first step in understanding the inner-workings of neural networks. Further work can focus on either making the tool more advanced to clearly map to neural networks or developing a curriculum that reframes a student's learning about the tool in terms of data representation, reasoning, neural networks, and the relevant terminology.

Conclusion

In this paper, we present the *Feature Detection Tool*, a web-based, two-part activity that introduces middle school students to features and allows them to build object-detection rules based on the combination of features. This tool builds on existing work on teaching computer perception in K-12 by giving students the ability to visualize low-level feature detection and to have more control over the design of the algorithms. We discuss how this tool can be used as a "first step" to the more complex concept ideas of data representation and neural networks. We hope that this tool is helpful to the K-12 AI Education community.

Acknowledgments

The authors would like to thank Project STEM and Amazon Future Engineer for supporting this work. Thank you to Christie Ha for feedback on the interface design of this tool.

References

Black, N. B.; and Brooks-Young, S. 2020. *Hands-On AI Projects for the Classroom: A Guide for Computer Science Teachers*. ISTE.

Carney, M.; Webster, B.; Alvarado, I.; Phillips, K.; Howell, N.; Griffith, J.; Jongejan, J.; Pitaru, A.; and Chen, A. 2020. Teachable machine: Approachable Web-based tool for exploring machine learning classification. In *Extended abstracts of the 2020 CHI conference on human factors in computing systems*, 1–8.

Harris, C.; Stephens, M.; et al. 1988. A combined corner and edge detector. In *Alvey vision conference*, volume 15, 10–5244. Citeseer.

Jordan, B.; Devasia, N.; Hong, J.; Williams, R.; and Breazeal, C. 2021. PoseBlocks: A toolkit for creating (and dancing) with AI. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 15551–15559.

Karpathy, A. 2016. ConvNetJS MNIST demo. *External Links: Link*.

Makwana, J.; Wolff, M.; Ratin, B.; and Touretsky, D. S. 2020. TinyYoloV2 Face Detection. https://www.cs.cmu.edu/~dst/FaceDemo/. Accessed: 2022-01-14.

Nixon, M.; and Aguado, A. 2019. *Feature extraction and image processing for computer vision*. Academic press.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. "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*, KDD '16, 1135–1144. New York, NY, USA: Association for Computing Machinery. ISBN 9781450342322.

Suzuki, S.; et al. 1985. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1): 32–46.

Tang, D.; et al. 2019. *Empowering novices to understand and use machine learning with personalized image classification models, intuitive analysis tools, and MIT App Inventor*. Ph.D. thesis, Massachusetts Institute of Technology.

Touretzky, D.; Gardner-McCune, C.; Martin, F.; and Seehorn, D. 2019. Envisioning AI for K-12: What Should Every Child Know about AI? In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press. ISBN 978-1-57735-809-1.

Wang, Z. J.; Turko, R.; Shaikh, O.; Park, H.; Das, N.; Hohman, F.; Kahng, M.; and Chau, D. H. P. 2020. CNN explainer: learning convolutional neural networks with interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 27(2): 1396–1406.

Yang, M.-H.; Kriegman, D. J.; and Ahuja, N. 2002. Detecting Faces in Images: A Survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(1): 34–58.

Yang, Y.; and Newsam, S. 2010. Bag-of-visual-words and spatial extensions for land-use classification. In *Proceedings of the 18th SIGSPATIAL international conference on advances in geographic information systems*, 270–279.

Yuen, H.; Princen, J.; Illingworth, J.; and Kittler, J. 1990. Comparative study of Hough transform methods for circle finding. *Image and vision computing*, 8(1): 71–77.