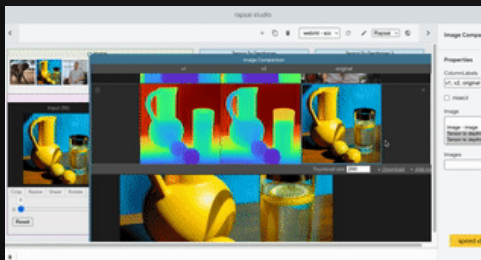




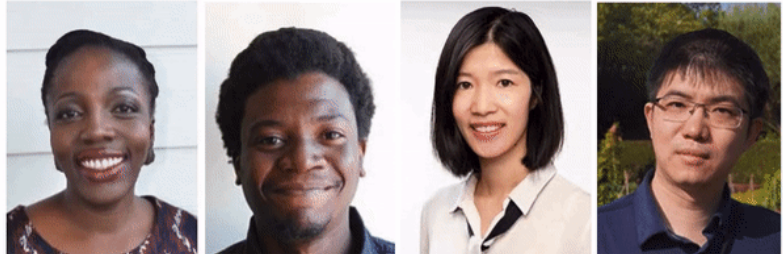
Accelerating Machine Learning Prototyping of Multimedia Applications through Visual Programming



Ruofei Du, Na Li, Jing Jin, Michelle Carney, Scott Miles, Maria Kleiner, Xiuxiu Yuan,
Yinda Zhang, Anuva Kulkarni, Xingyu "Bruce" Liu, Ahmed Sabie, Sergio Escolano, Abhishek Kar,
Ping Yu, Ram Iyengar, Adarsh Kowdle, and Alex Olwal

visualblocksforml.github.io





3D Photos



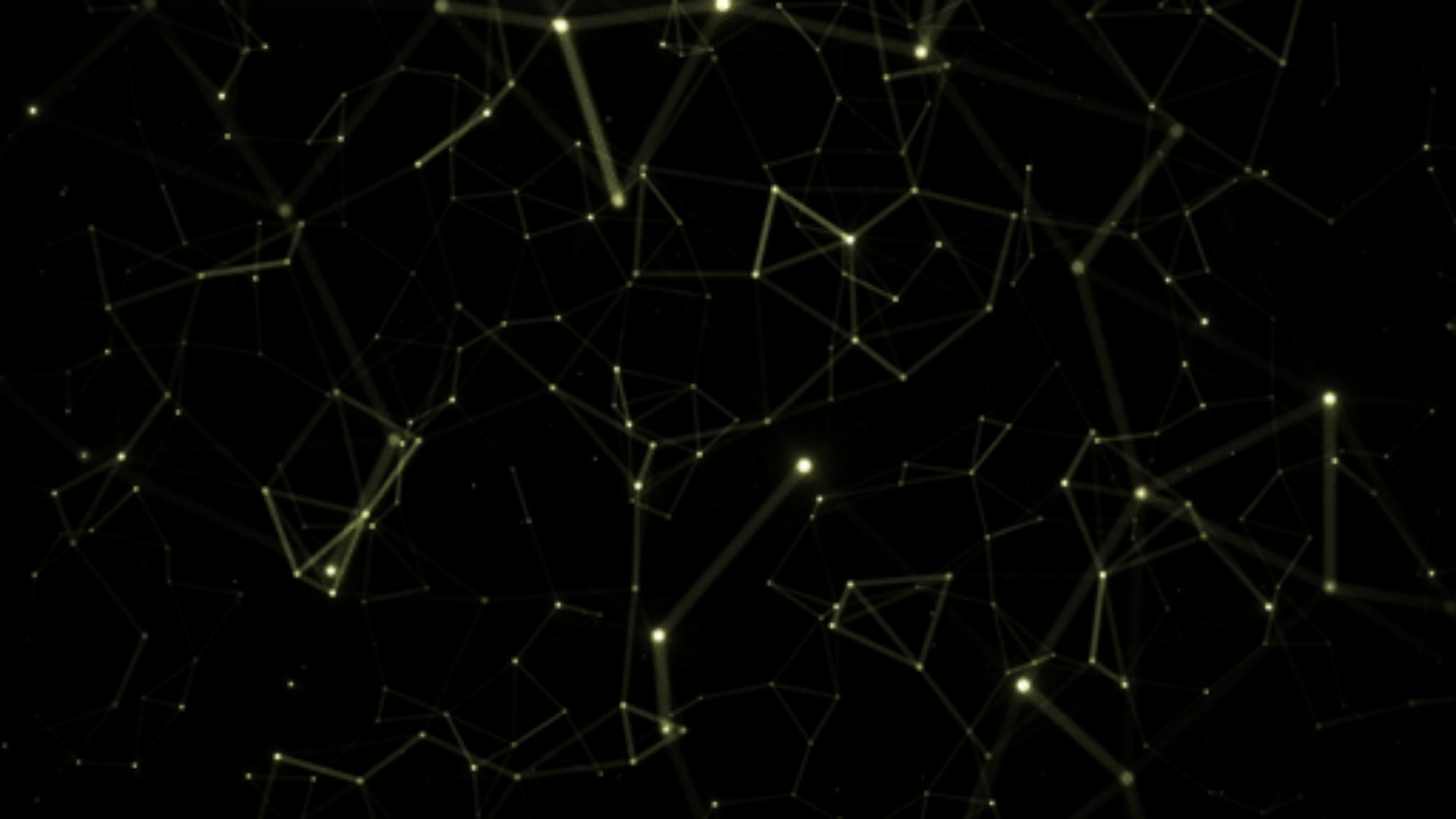
Portrait Relighting

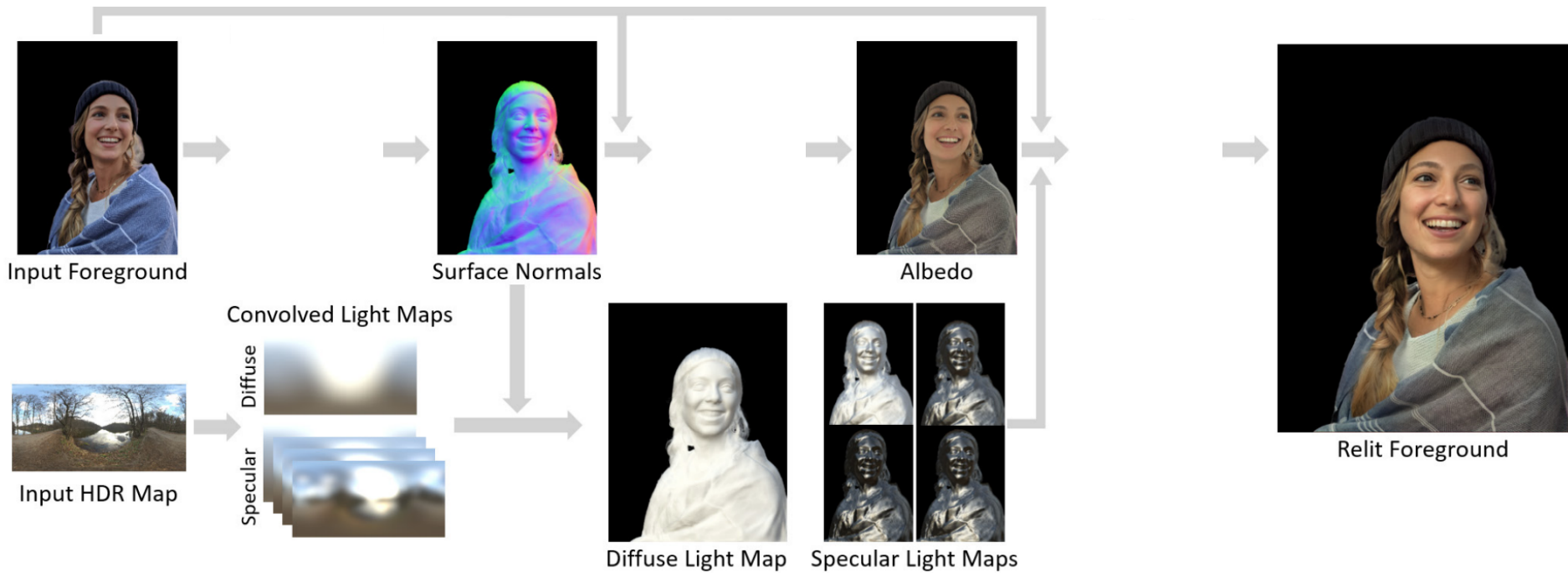


Virtual Background



MediaPipe





THE UNIVERSITY OF CHICAGO
 DIVISION OF THE PHYSICAL SCIENCES
 DEPARTMENT OF CHEMISTRY

REPORT OF THE RESEARCH GROUP ON
 THE CHEMISTRY OF ORGANIC
 COMPOUNDS

FOR THE YEAR 1945-1946

EDITED BY R. M. BUNN

CHICAGO, ILLINOIS, 1946

PRINTED BY THE UNIVERSITY OF CHICAGO PRESS

ALL RIGHTS RESERVED

LIBRARY OF THE UNIVERSITY OF CHICAGO

520 EAST 57TH STREET, CHICAGO, ILL.

1946

1945-1946

1945-1946

1945-1946

1945-1946

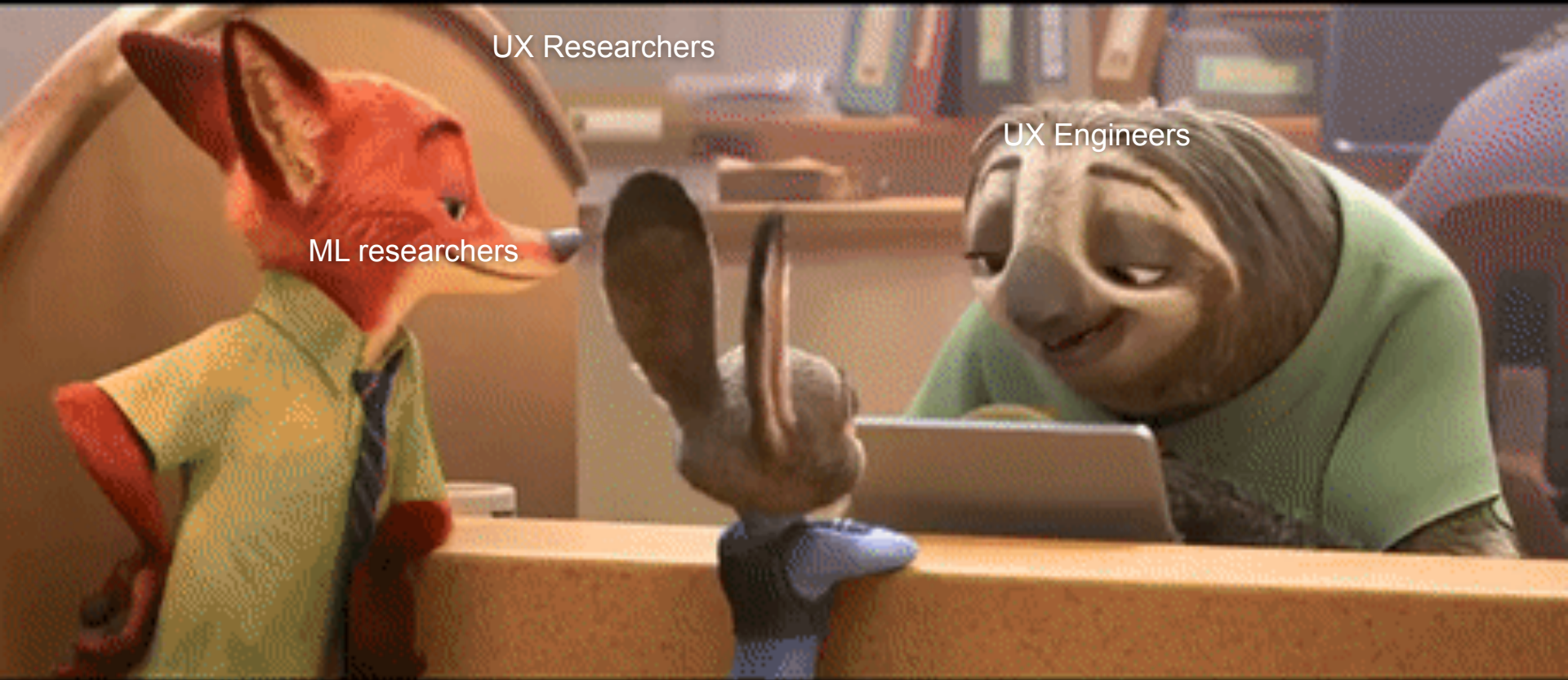
1945-1946

1945-1946

1945-1946

1945-1946

1945-1946



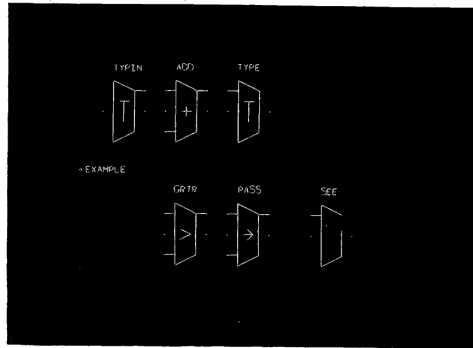
UX Researchers

UX Engineers

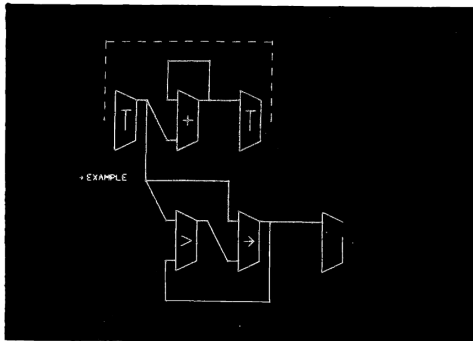
ML researchers

ML researchers

Related Work
Visual programming
in graphics



Basic Symbols
Figure 1.1



Connected Program
Figure 1.2

MASS. INST. OF TECHNOLOGY
APR 5 1966
LIBRARIES

THE ON-LINE GRAPHICAL SPECIFICATION OF COMPUTER PROCEDURES

by

WILLIAM ROBERT SUTHERLAND

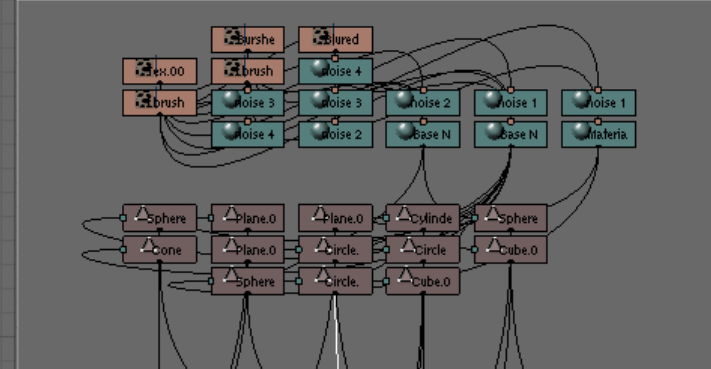
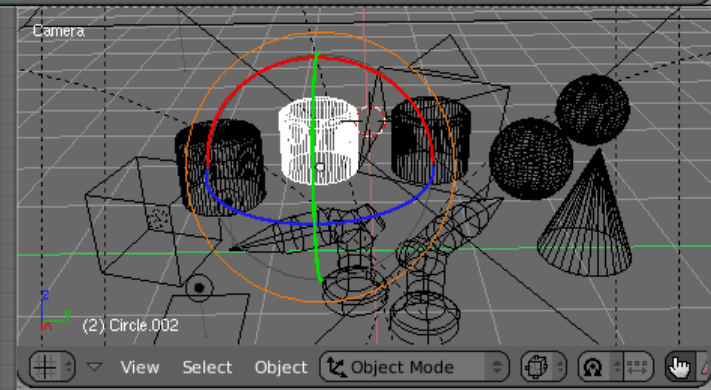
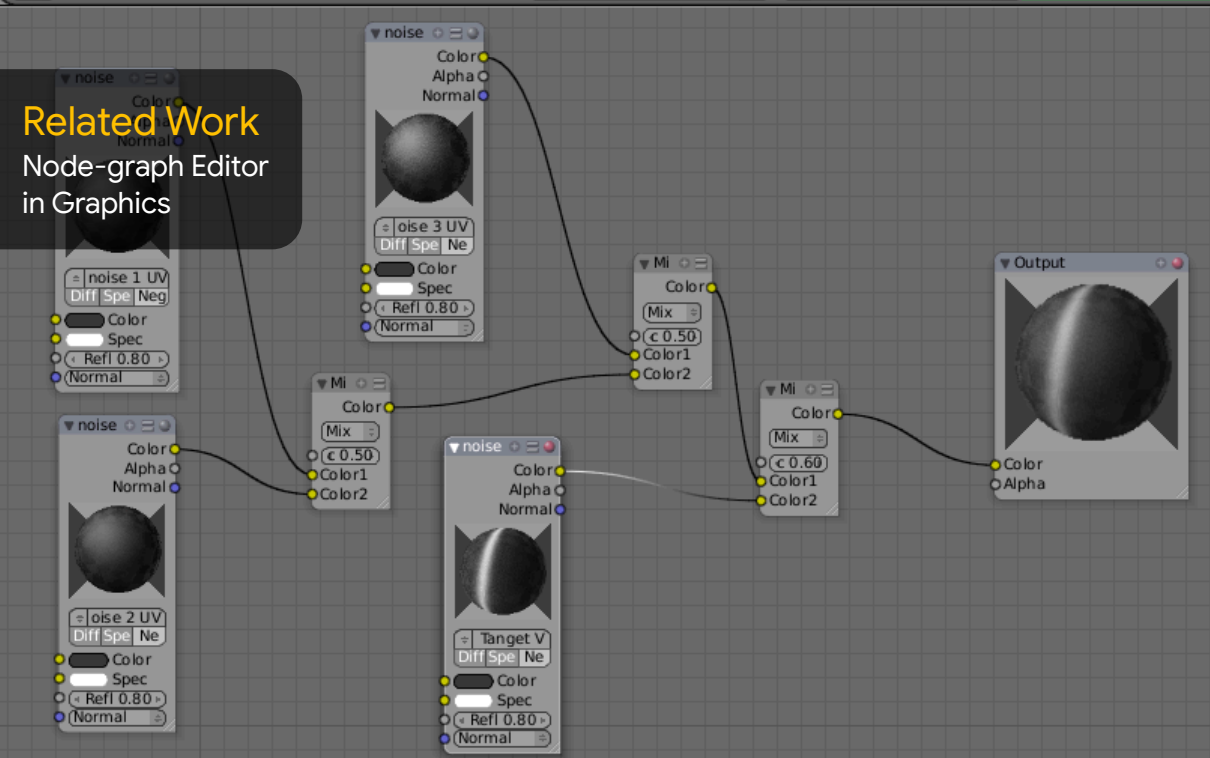
B.E.E., Rensselaer Polytechnic Institute
(1957)
M.S., Massachusetts Institute of Technology
(1963)

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
January, 1966

Signature of Author.....
Department of Electrical Engineering, January 10, 1966
Certified by..... Thesis Supervisor
Accepted by.....
Chairman, Departmental Committee on Graduate Students

Related Work

Node-graph Editor in Graphics



Preview: A sphere rendered with the material.

Material: VCol Lig VCol Pai | TexFace | ShadeLe Env

Shaders: **Lambe** (Ref: 0.8) | **Tangen**

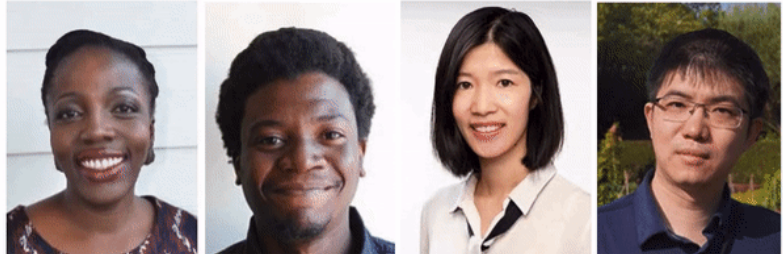
Mirror Transp: RayMir 0.2 | Ray Mirror | Depth: 4 | Fresnel 2 | Fac: 1.2

Texture: Col | Nor | Csp | Cmir | Ref | Spec | Am | Hard | RayM | Alph | Emi | Transl | Disp

Map Input: Glob | Object | UV | Orco | Stick | Win | Nor | Refl | Stress | Tangent

Ramps: Link to Object | 1: Base Node UV | 2 | X | F | (Nod) | E: Circle.004 | OB ME | 1 Mat 1 | Active Material Node | MA: noise 4 UV Target V

Render Pipeline: Halo | Z/Transp | offs: 0.000 | Full Osa | Wire | Strands | ZInvert | Radio | OnlyCas | Traceabi | Shadbuf



3D Photos



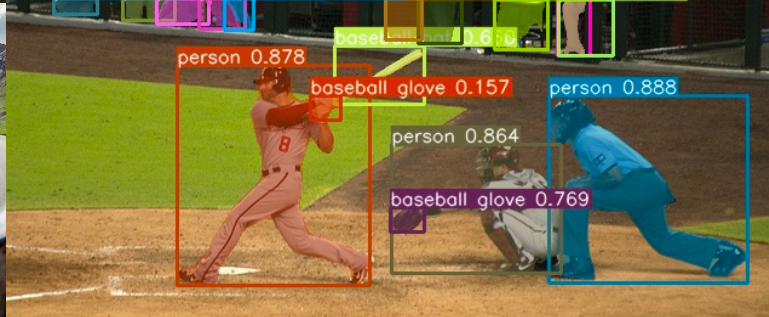
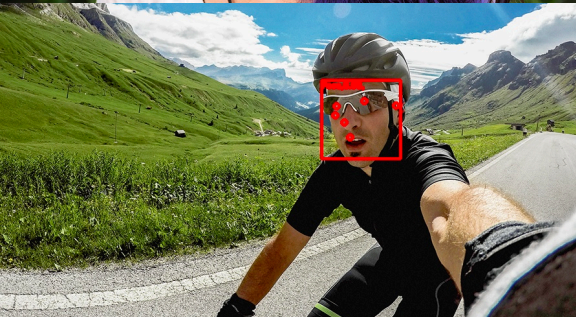
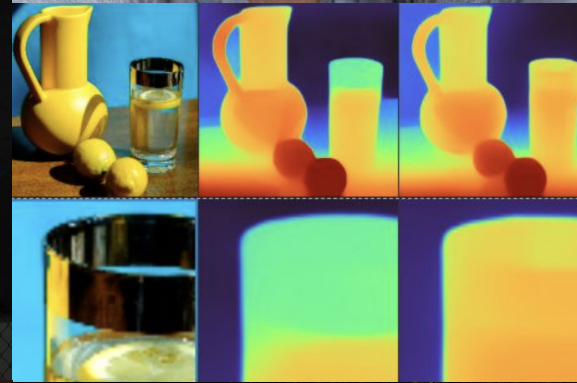
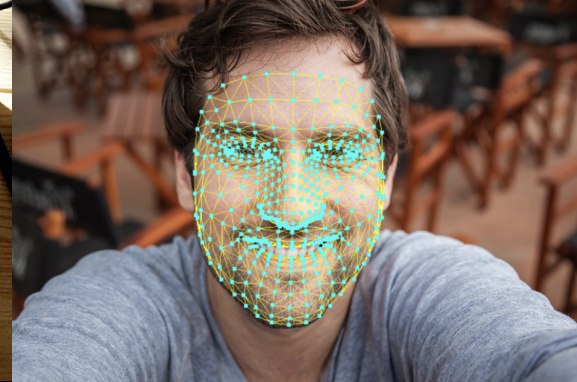
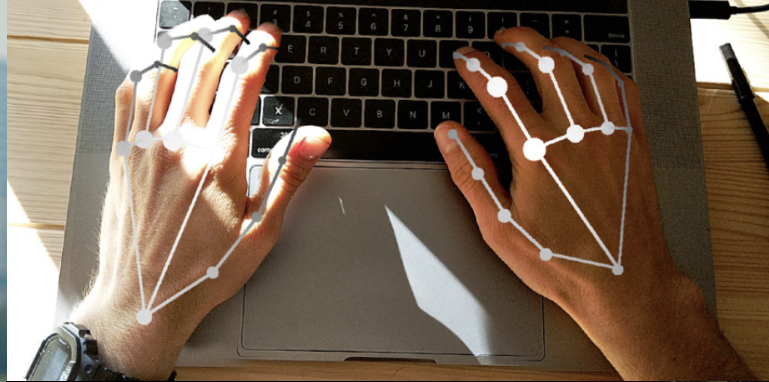
Portrait Relighting



Virtual Background



MediaPipe



 TensorFlow

 PyTorch

 Keras

Model Development







**Video / Graphics /
Audio Processing**



Multimedia Applications



Formative
Study (N=7)



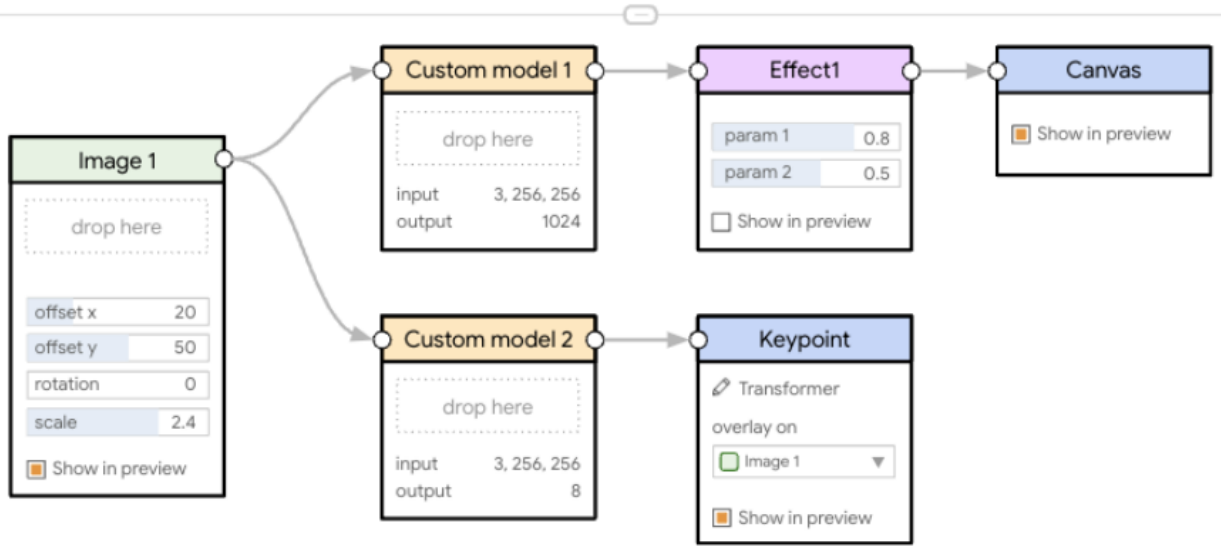
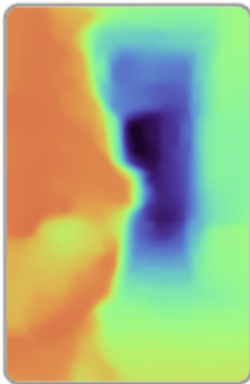
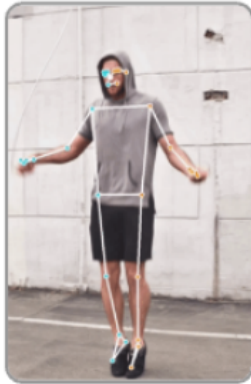
Jing Jin

Na Li



Mockup Sketch

- INPUT**
 - Image
 - Camera
 - Raw
- MODEL**
 - Pre-trained model ▶
 - Custom model
- EFFECT**
 - Effect1
 - Effect2
- OUTPUT**
 - Canvas
 - Key points
 - Raw
- MISC**
 - Performance



Lack of **data processing** for input in-the-wild

The complexity of processing of input formats yields longer cycles between finding failure cases and fine-tuning new models.

Lack of **interactive** data and model tuning

2

"models can be affected by over-fitting, so need to test with a large variety of image augmentations" (12)

Loss of application context

3

“Metric doesn’t help <in my depth models>, it’s always good for all the models, so it’s no use. They need human eyes to evaluate.” (16)

Lack of **direct comparison** and **sharing**

4

"I want to isolate bad examples of a specific error pattern to discuss with stakeholders." (I1)

Slow iterations

5

"A lot of time goes into visualization of challenge sets, benchmarking, and metrics. Usually takes weeks." (I2)

Insufficient **controllability**

6

"We need to integrate the model with other modules — (to evaluate whether) can we improve the higher-level model?" (17)



Design goals

1. **Visual programming** for rapid prototyping
2. Run **real-time** ML pipelines
3. Input **in-the-wild**
4. **Interactive** data augmentation
5. **Side-by-side** comparison
6. **Off-the-shelf** & customize models



rapsai →

Visual Blocks for ML

A Visual Programming Platform for Rapid and Iterative
Development of End-to-end ML-based Applications

Search nodes



Audio

Input

Image

Live Camera



Effect



Model



Output



Tensor



Misc

WebML1 Rapsai

Inputs: url [ImageURL]
image Image
Outputs: image Image

Nothing to inspect

Rapsai





































System Overview

The screenshot displays the Rapsai Studio interface, which is divided into several functional panels:

- (a) Nodes Library:** A sidebar on the left containing categories such as Audio, Input, Effect, Model, Output, Tensor, and Misc. The 'Input' category is currently selected.
- (c1) interactive data augmentation:** A panel titled 'Image Processor' showing 'Input (fit)' and 'Output (fit)' images of a yellow pitcher and glass. Below the images are sliders for 'Crop', 'Resize', 'Shear', 'Rotate', 'Brightness' (set to 18), 'Contrast', 'Blur', and 'Noise'. A 'Reset' button is located at the bottom.
- (c2) model comparison:** A central panel displaying a grid of images. The top row shows the 'input' image, the 'processed image', and two model outputs ('model1' and 'model2'). The bottom row shows zoomed-in views of the glass from the input and processed images, along with their corresponding depth maps. A 'Thumbnail size: 160' and 'Download + Add row' options are visible.
- (d) Node Inspector:** A panel on the right showing the 'Properties' of the selected 'Image comparison' node. It lists 'Inputs: image, Image (multiple) [Image], columnLabels String' and 'ColumnLabels: input, processed image, model1, model2'. There is also a 'HideUI' checkbox and an 'Images' dropdown menu.
- (b) Node-graph Editor:** A bottom panel showing a visual programming graph. The flow starts with an 'Image' node, followed by an 'Image processor' node, then a 'Preprocess image' node. This leads to two parallel paths: one for 'Scene Depth v1' and another for 'Scene Depth v2'. Each path includes a 'Tensor picker' node, a 'Remap value range' node, and a 'Depth Map Vis' node. The final outputs are connected to an 'Image comparison' node.

Rapsai

Nodes Library

Search nodes		Search nodes		Search nodes		Search nodes		Search nodes		Search nodes	
	Audio		Audio Processor		Body Segmentation		3D Model Viewer		Binary Op		Get Image Size
Input	Image	Input	Fragment Shader	Input	CCA Denoise	Input	Audio Comparison	Input	Clip By Value	Input	Get Size From Rect
	Input Stream		Image Processor		Custom Model Runner		Audio Player		Const Tensor		Lobby
Effect	Live Camera	Effect	Images Mixer	Effect	Mobilenet	Effect	Bar Viewer	Effect	Crop And Resize	Effect	Webpage
	Simple Audio		Shader Library				Device Image		Image To Tensor		
Model	Video	Model	Shader Processing	Model		Model	Image Comparison	Model	Postprocess Depth Model	Model	
							Image Viewer		Preprocess Image		
Output		Output		Output		Output	Json Viewer	Output	Remap Value Range	Output	
							Output Stream		Tensor Picker		
Tensor		Tensor		Tensor		Tensor	Tensor To Depthmap	Tensor	Tensor To ClassifierResults	Tensor	
							Tensor To Image				
Misc		Misc		Misc		Misc	Tensor Viewer	Misc		Misc	

D FULL LIST OF NODES IN RAPSAI

We list all the supported nodes in Rapsai before the case study. Note that Rapsai is extendable for expert users by adding new nodes in JavaScript.

D.1 Input nodes

- (1) **image**: capture a photo from webcam, upload from hard drive, or fetch from a list of remote URLs.
- (2) **video**: record a video with external webcam or upload a video from disk or YouTube.
- (3) **audio**: record sounds from microphone, or upload audio files from disk or Internet
- (4) **live camera**: use live camera stream, similar for the live audio node.
- (5) **remote stream**: stream input from another device (e.g., mobile phone) via WebRTC, by opening a URL of the page with a lobby node.
- (6) **lobby**: create a WebRTC server to accept video streaming from other devices. Remote stream nodes that connect to the lobby node with the same name are sharing output via WebRTC streaming.

D.2 Effect nodes

- (7) **image processor**: crop and translate a region of interest in the input to verify an image model's invariance to translation; rotate, shear, resize an image to examine potential biasing issues in the training sets; apply blur and noise to test a model's robustness.
- (8) **image mixer**: mix images with GPU-based blending modes¹⁰
- (9) **audio processor**: trim the audio, change volume, and add background noise from a collection of 17 presets.
- (10) **fragment shader**: program and apply a screen-space graphical shader effect.
- (11) **shader library**: offers a pre-defined list of shader code to avoid coding into the "fragment shader".

D.3 Model nodes

- (12) **custom model runner**: enter the URL of an TensorFlow.js model or upload a TensorFlow model into the pipeline.
- (13) **body segmentation**: run a deployed MediaPipe body segmentation model.

- (14) **audio denoising**: run either of two deployed audio denoising models.
- (15) **MobileNet**: run a deployed MobileNet model for image classification.

D.4 Output nodes

- (16) **image viewer** node displays an image.
- (17) **audio player**: play a single audio output.
- (18) **image comparison**: qualitatively compare output from multiple models with zoom-in tools.
- (19) **audio comparison**: qualitatively compare output from multiple models with automatic track switching.
- (20) **JSON viewer**: read the raw output from a model for debugging.
- (21) **bar viewer**: view the classification results from a model such as MobileNet.
- (22) **3D model viewer**: view 3D models from an URL or tensor.
- (23) **tensor to image**: view a tensor as images.
- (24) **tensor to depthmap**: view a tensor as depthmaps with different transfer functions.
- (25) **output stream**: see remote video streams from the **lobby node** input.

D.5 Tensor nodes

- (26) **preprocess image**: converts an input image to a 4D tensor as an input that is required by most image models.
- (27) **tensor picker**: select a tensor from an array of output tensors.
- (28) **tensor postprocess**: convert a tensor to an image and apply normalization calculators.
- (29) **binary op**: apply "and", "or", "xor", and "not" operations between two input tensors.
- (30) **clip by value**: clamp the values of an input tensor.
- (31) **crop and resize**: crop and resize a two-dimensional tensor.
- (32) **preprocess tensor**: normalize a tensor, expand dimensions, and optionally convert to grayscale image for many generative models.
- (33) **postprocess tensor**: normalize and resize a tensor for image output.
- (34) **tensor picker**: select a certain tensor from an array of model output. Note that most models only have one output so it is optional.
- (35) **remap value range**: select an input and an output ranges to remap tensor values.

D.6 Miscellaneous nodes

- (36) **webpage**: append a Google Form or a custom webpage for filling in surveys.
- (37) **image size**: obtain image size for outputting to some models.

Search: image

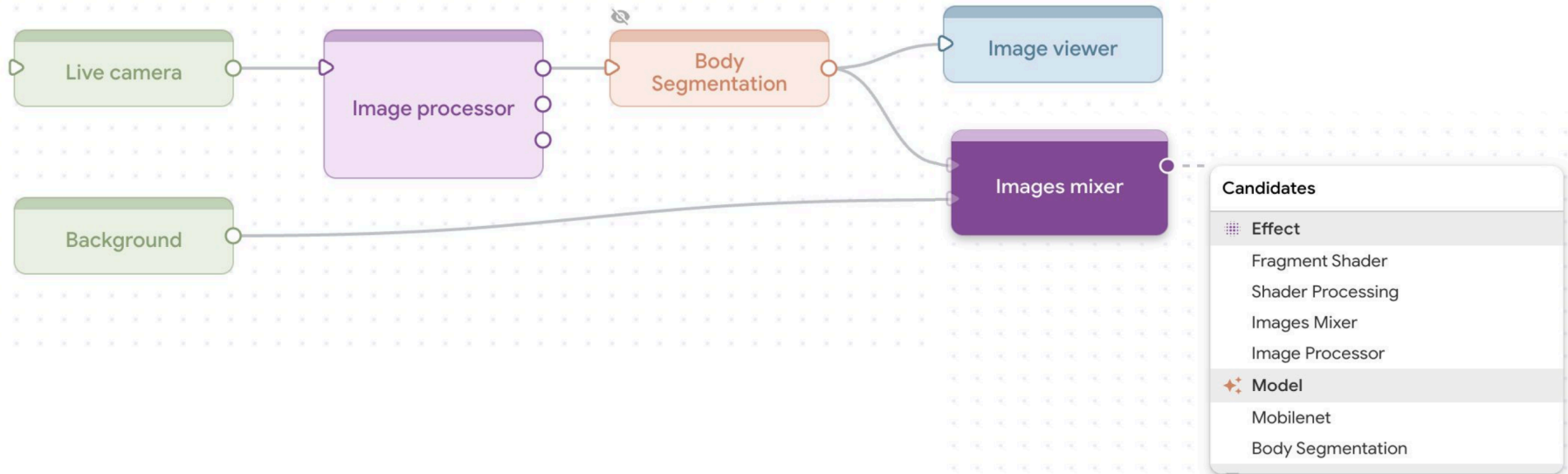
- Input** (1)
 - Image Processor
 - Images Mixer
- Effect** (2)
- Model**
- Output** (4)
- Tensor** (2)
- Misc** (1)

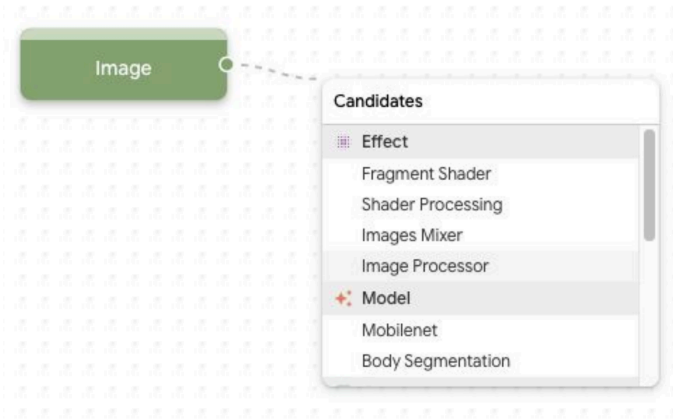
Search: audio

- Input** (2)
 - Audio Comparison
 - Audio Player
- Effect** (1)
- Model**
- Output** (2)
- Tensor**
- Misc**

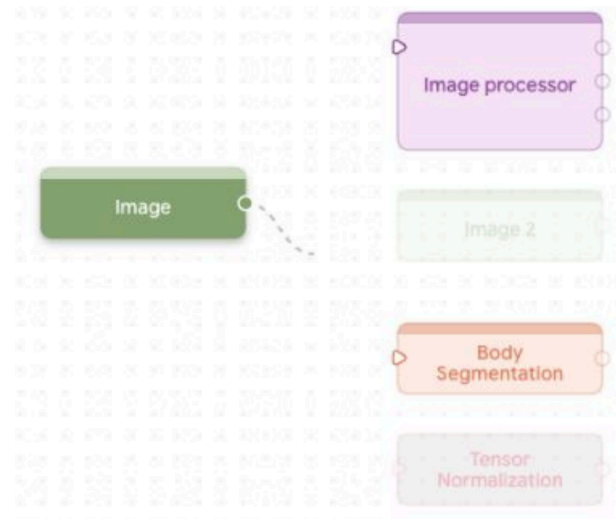
Rapsai

Node-graph Editor





Node Suggestion



Link Suggestion

Rapsai

Preview Panel

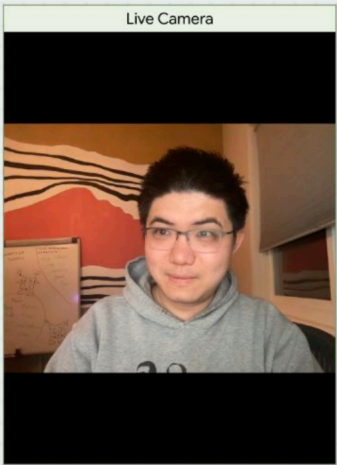
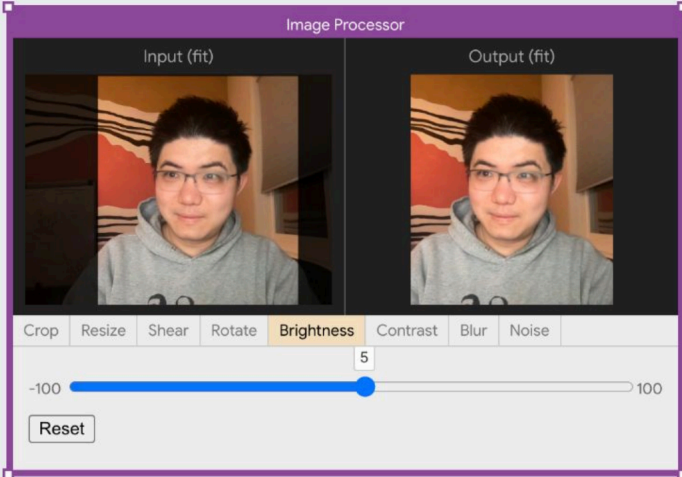


Image Processor

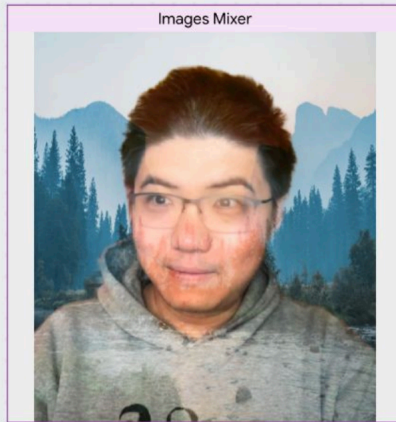
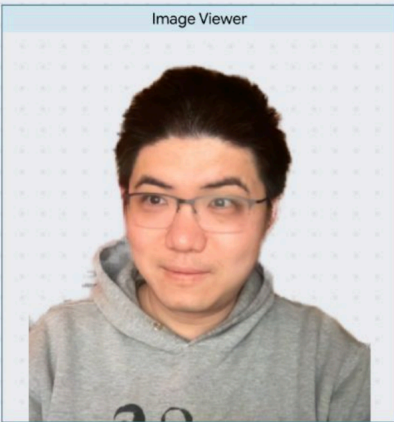
Input (fit) Output (fit)

The Image Processor panel is divided into two side-by-side image windows. The left window, labeled 'Input (fit)', shows the original video frame. The right window, labeled 'Output (fit)', shows the same frame with a slight brightness increase. Below the windows is a control bar with buttons for 'Crop', 'Resize', 'Shear', 'Rotate', 'Brightness', 'Contrast', 'Blur', and 'Noise'. The 'Brightness' button is highlighted in yellow. Below the buttons is a horizontal slider with a blue bar and a blue knob. The slider is labeled with '-100' on the left, '5' in the middle, and '100' on the right. A 'Reset' button is located at the bottom left of the panel.

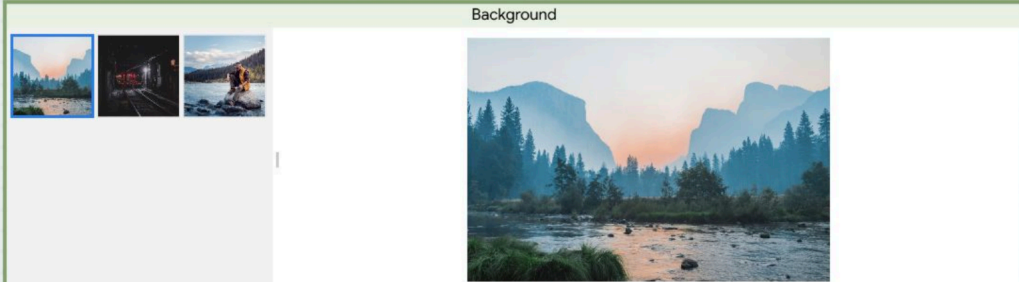
Crop Resize Shear Rotate **Brightness** Contrast Blur Noise

-100 5 100

Reset

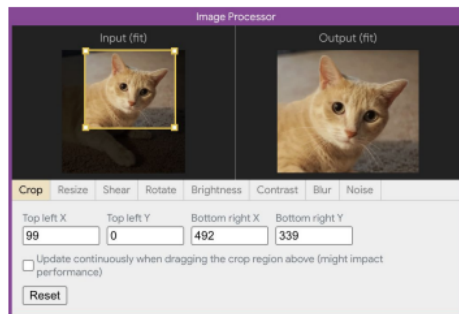


Background

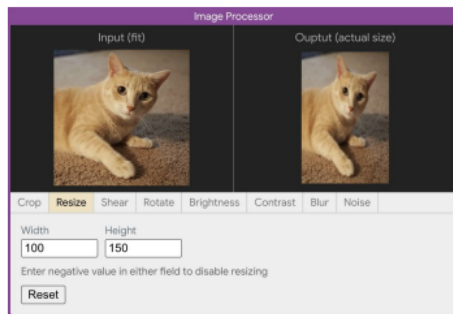
The Background panel features a horizontal row of three small thumbnail images on the left: a mountain landscape, a street at night, and a person on a boat. To the right of these thumbnails is a large, full-sized image of a mountain landscape with a river, which is currently selected and displayed in the main area of the panel.

Rapsai

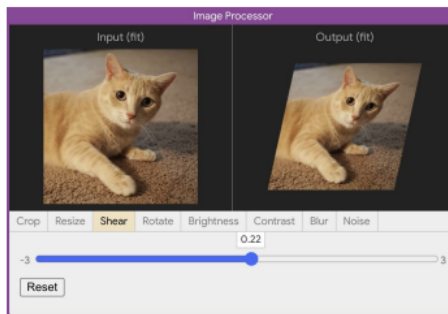
Interactive Data Augmentation



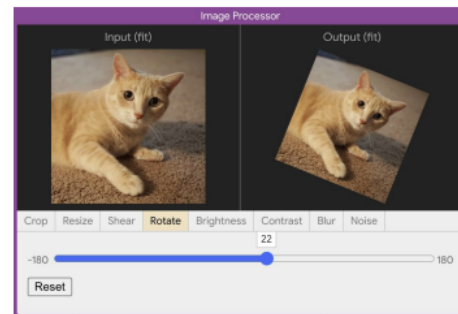
(a) crop



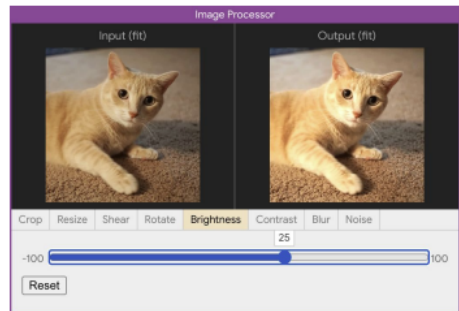
(b) resize



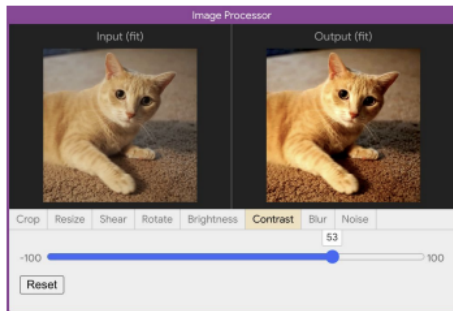
(c) shear



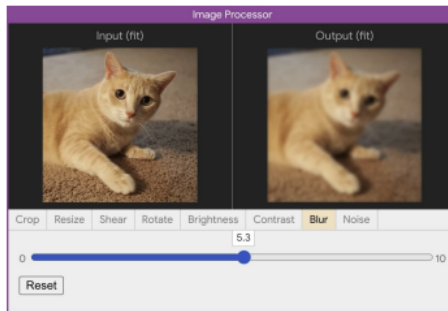
(d) rotate



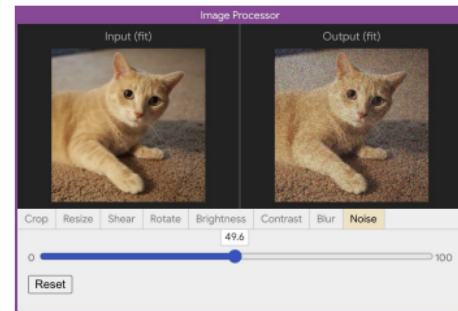
(e) brightness



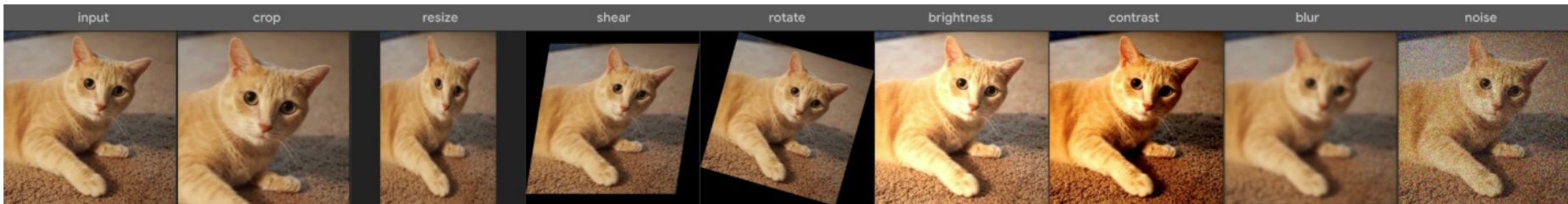
(f) contrast



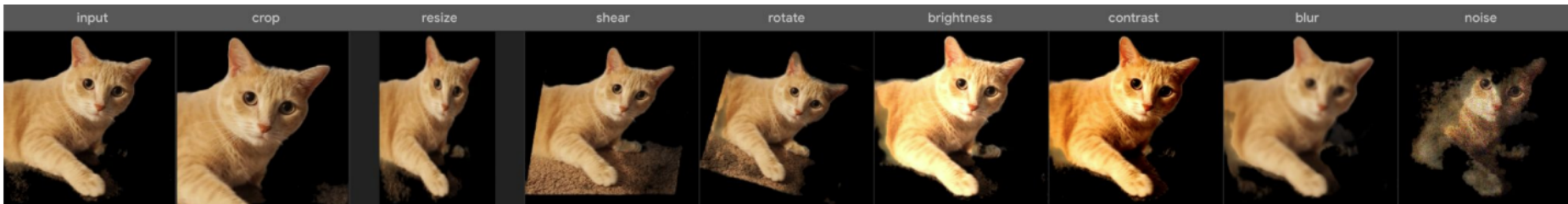
(g) blur



(h) noise



(a) intermediate results of interactive data augmentation applied onto a single input image

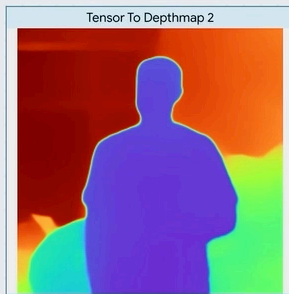
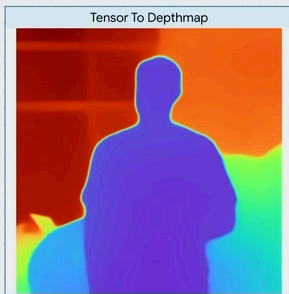
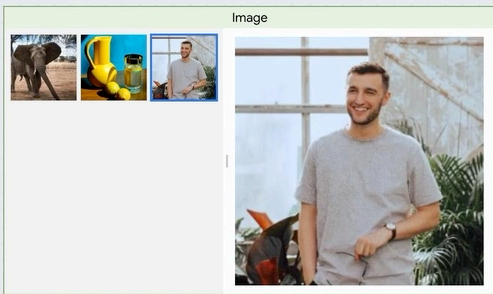


(b) side-by-side comparison of segmentation results with different augmentation techniques

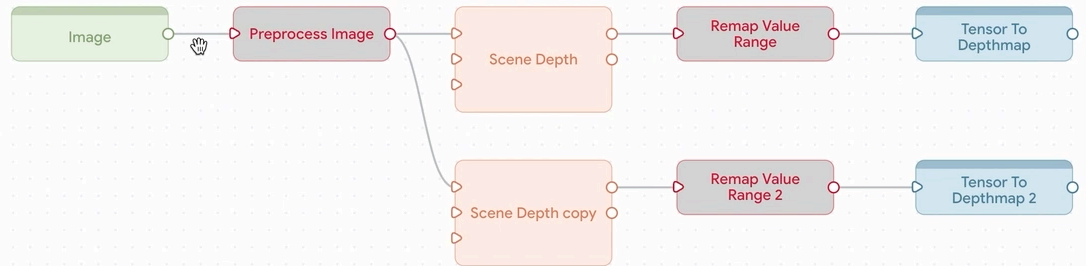
Search nodes

+ webml - sce Rapsai

- Audio
- Input
- Image
- Live Camera
- Effect
- Model
- Output
- Tensor
- Misc



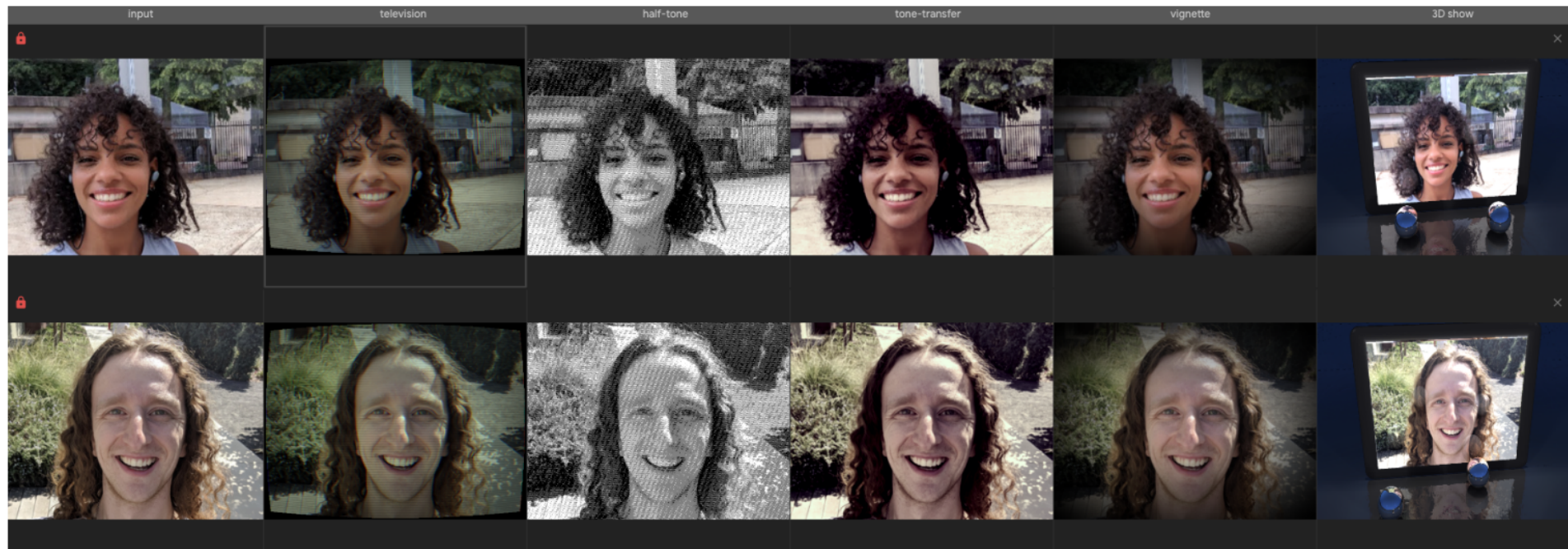
Nothing to inspect

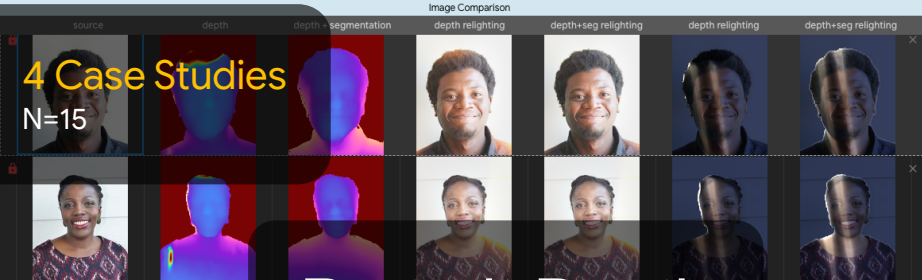


speed x8

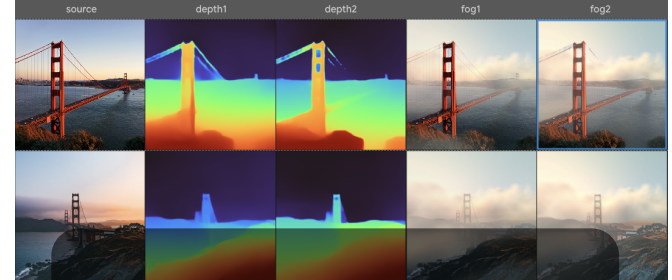
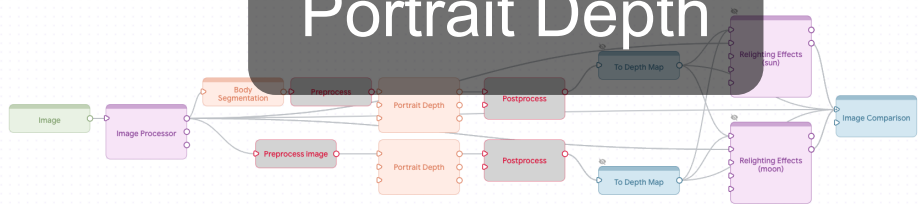
Rapsai

Shader Library

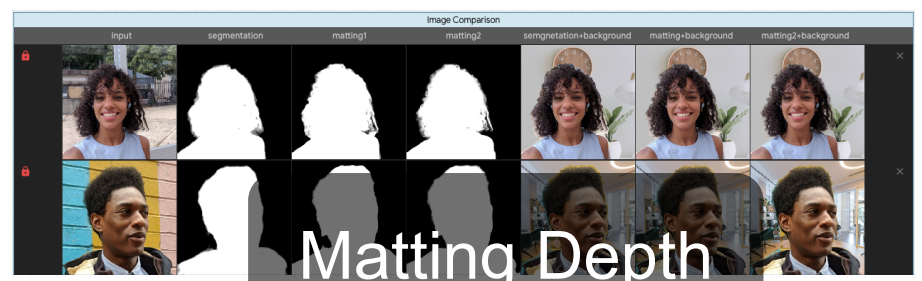
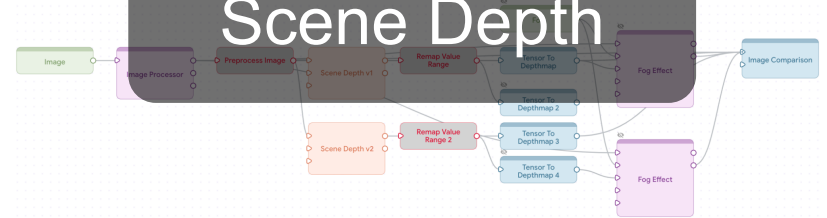




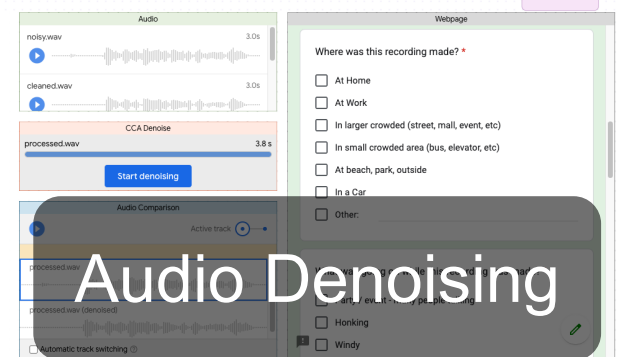
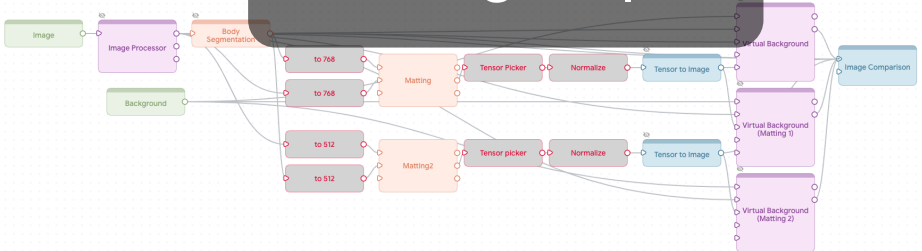
Portrait Depth



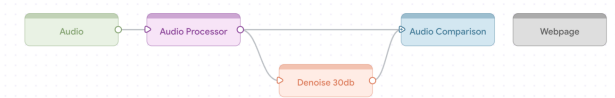
Scene Depth



Matting Depth

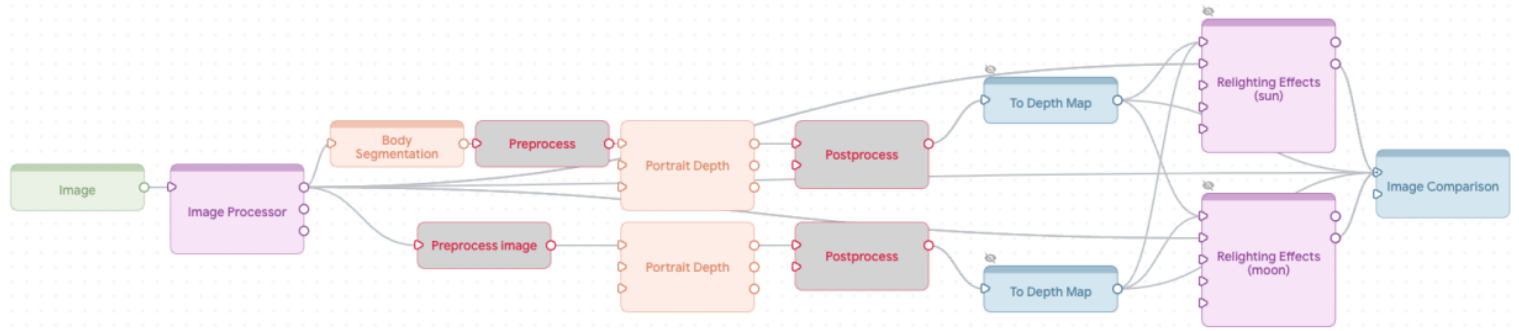
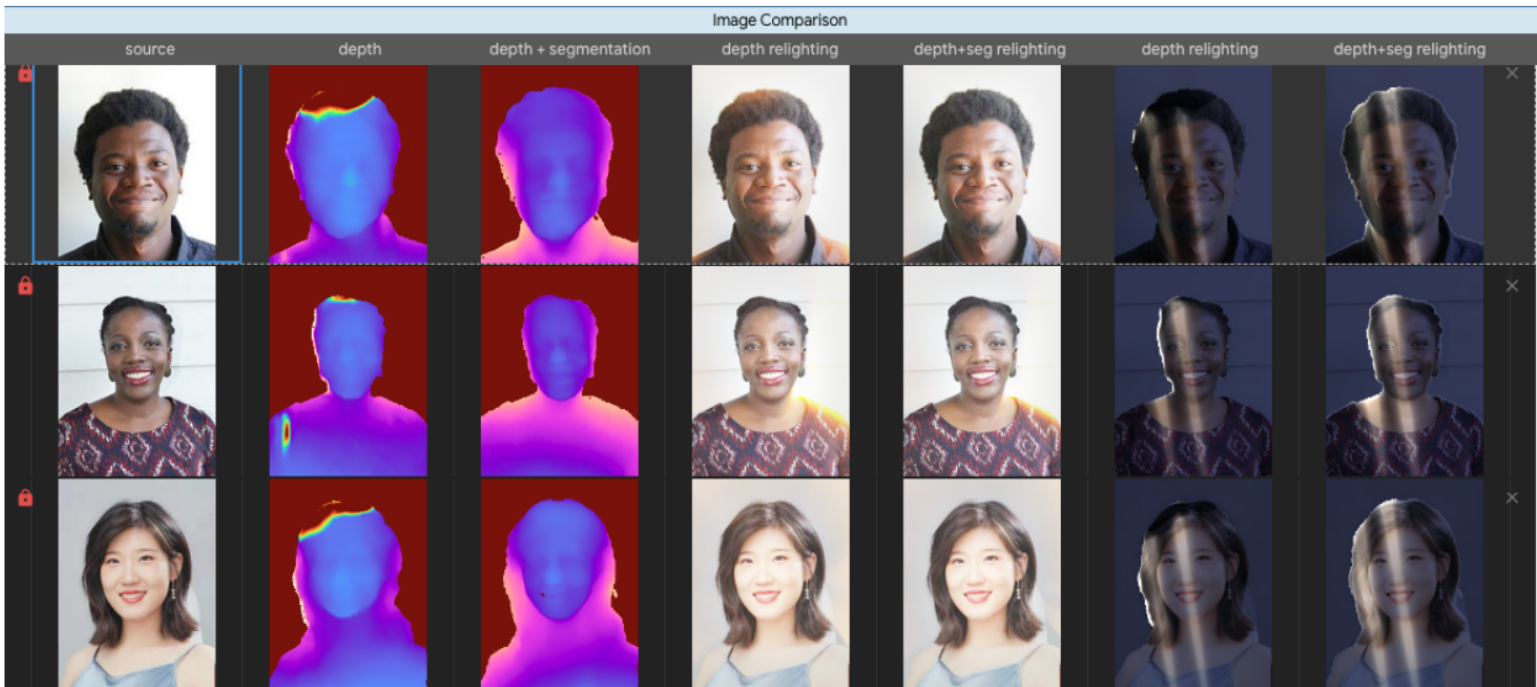


Audio Denoising



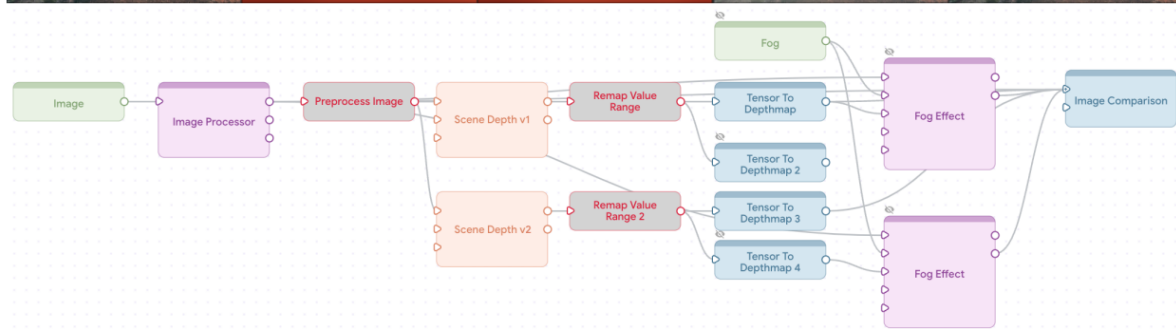
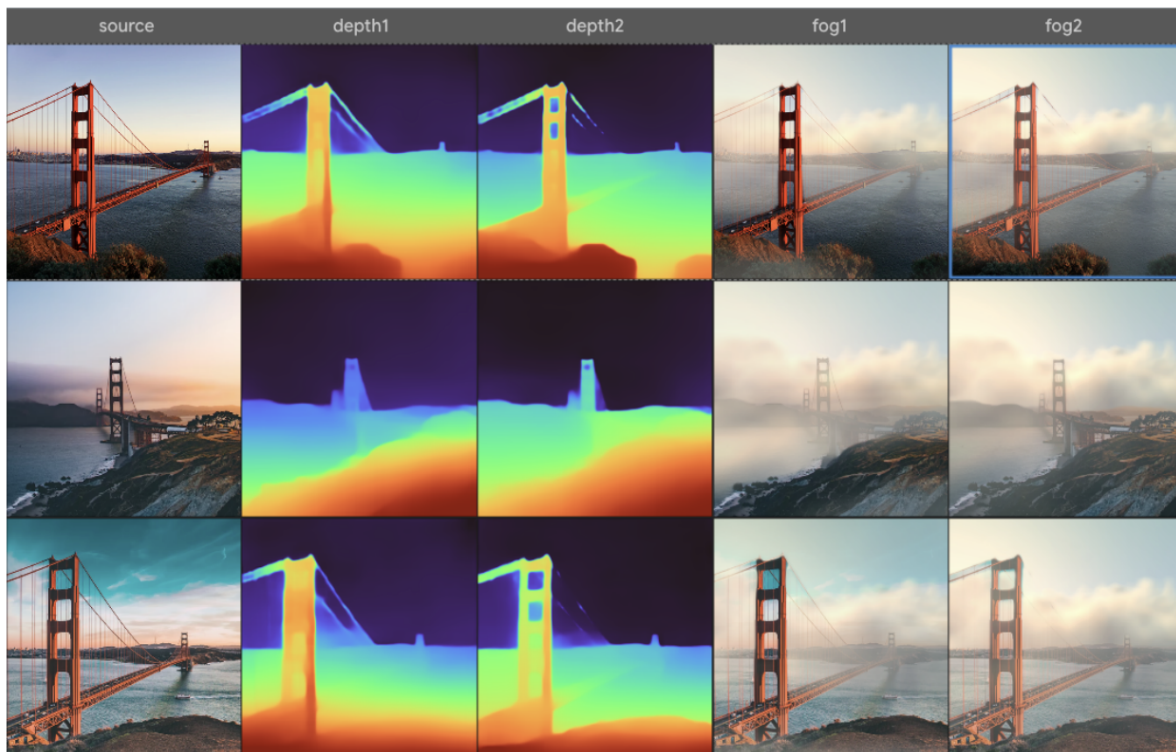
Case Study 1

Portrait Depth



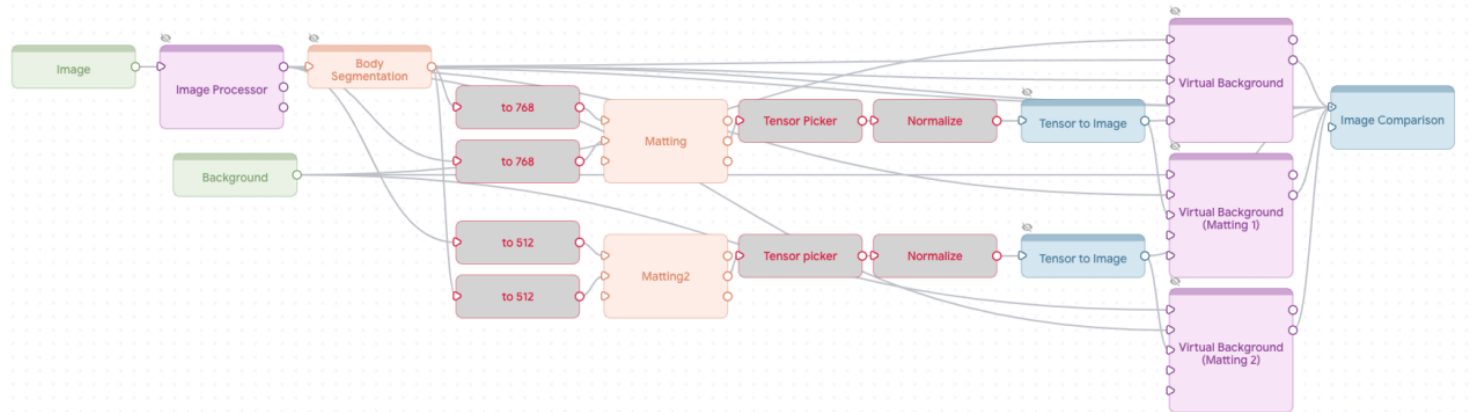
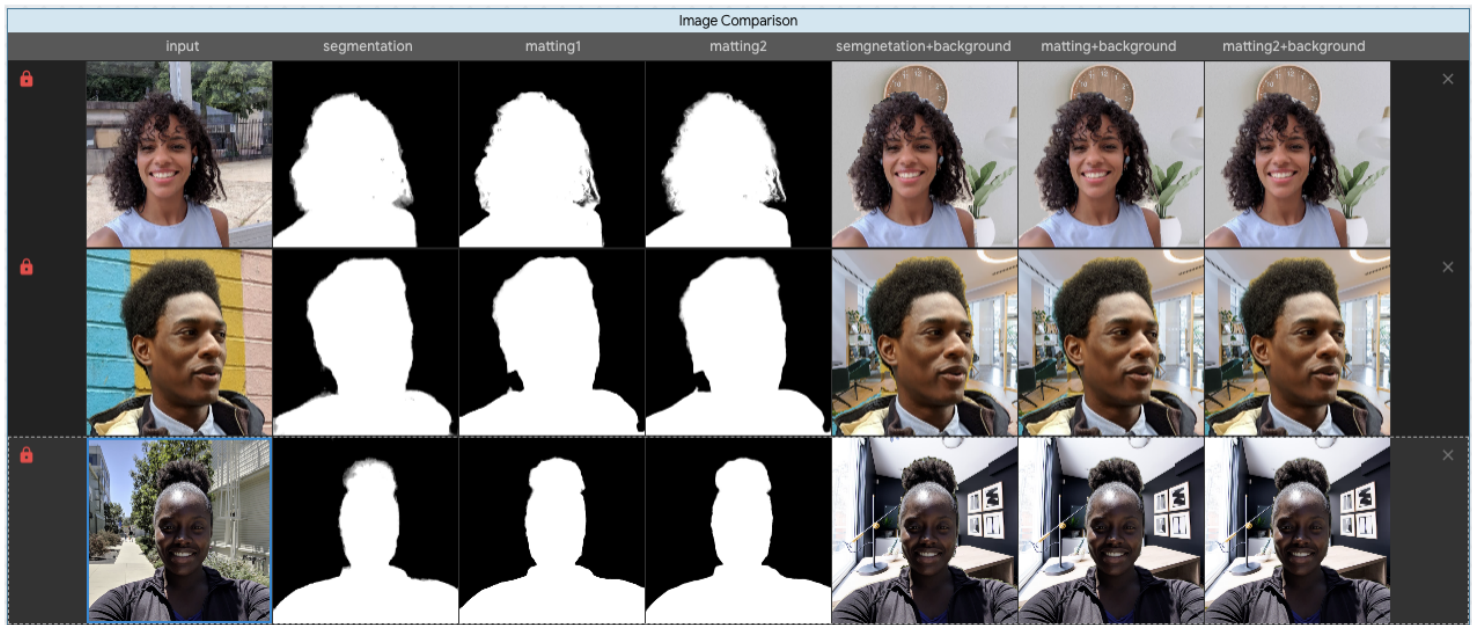
Case Study 2

Scene Depth



Case Study 3

Alpha Matte



Case Study 4

Audio Denoising

Audio

noisy.wav 3.0s

cleaned.wav 3.0s

CCA Denoise

processed.wav 3.8 s

Start denoising

Audio Comparison

Active track

processed.wav

processed.wav (denoised)

Automatic track switching ?

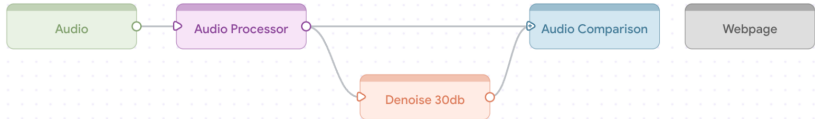
Webpage

Where was this recording made? *

- At Home
- At Work
- In larger crowded (street, mall, event, etc)
- In small crowded area (bus, elevator, etc)
- At beach, park, outside
- In a Car
- Other: _____

What was going on while this recording was made? *

- Party / event - many people talking
- Honking
- Windy



Case Study

Five Stages

Background **interview** (6.1 ± 0.8 min)

Video **tutorial** (4 min),

Visual analytics procedure using Rapsai (39.4 ± 4.6 min)

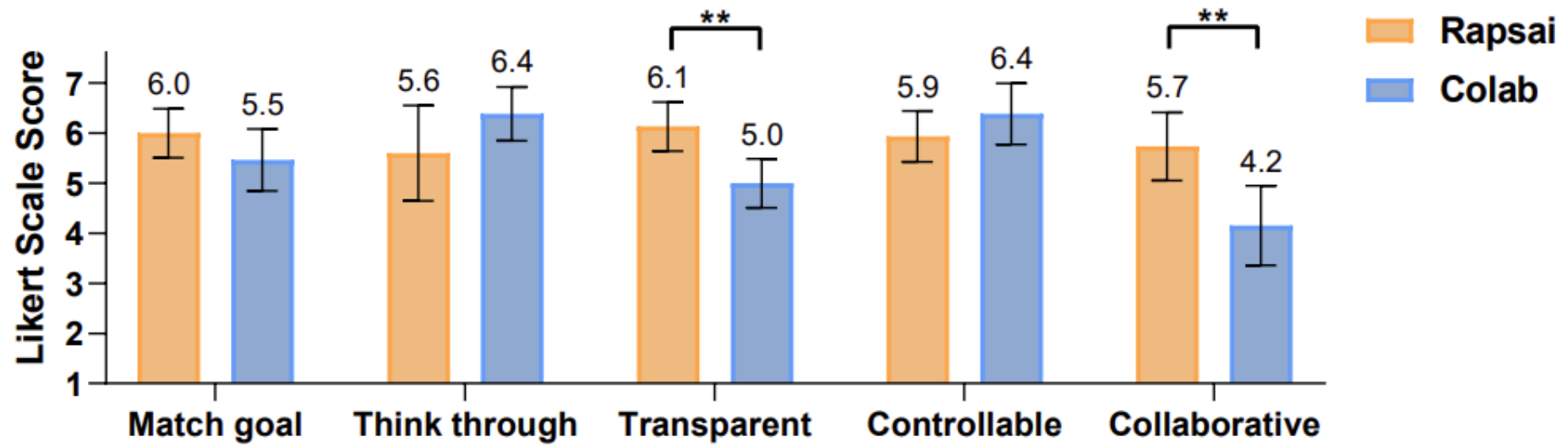
Discussion of Rapsai and perception prototyping in future (10.2 ± 2.0 min)

Post-hoc **exit survey** to use Rapsai and compare with Colab

Findings 1

Rapsai vs Colab

Less Control but More Transparent and Collaborative



“

*I spent about **half a minute** to create an image classification pipeline, and I spent **2–3 minutes** to build a depth estimation pipeline from scratch, since it took some time to figure out how to preprocess the input and visualize the output... while Colab is more flexible for different tasks, I guess it could range from **1 hour** to **a day or two**.*

”

“

In my case, I started from an existing template but overall it was quite fast, I'd say **less than 5 min.**

”

Findings 2

Assist in *Identifying Issues* with ML Models and Training Sets

The screenshot displays a software interface for a neural network pipeline, titled "rapai studio". The interface is divided into several sections:

- INPUT:** Includes "Camera", "Audio Recorder", "Webpage", and "Batch Image Upload".
- EFFECT:** Includes "Shader Toy", "Shader Toy Library", "Images Mixer", and "Image Processor".
- MODEL:** Includes "MultiNet", "Cuda Device", "Body Segmentation", "Portrait Depth", and "Custom Model Editor".
- OVERLAY:** Includes "Habitat" and a "5.5%" indicator.
- OUTPUT:** Includes "Image", "Media Image", "Json Viewer", "Bar Viewer", "Audio Visualizer", "Image Comparison", "Depthmap Visualizer", "Tensor To Image", and "Tensor To Denoising".
- TENSOR:** Includes "Crop And Resize", "Image To Tensor", "Range Value Range", "Connet Tensor", and "Resize To".

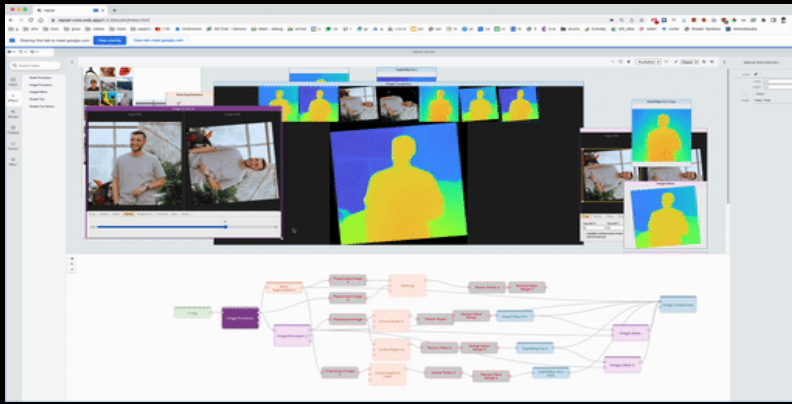
The main workspace shows a video feed of a woman's face. The "Input (img)" and "Output (actual view)" windows display the original video. Below the video, there are controls for "Crop", "Resize", "Brightness", "Contrast", and "Blur". To the right of the video, there are several small preview windows showing depth maps (purple and blue) and the original video with depth overlays. A large window on the right shows a "DepthPipeline" with a grid of images: two depth maps, the original video, and the original video with depth overlays.

Below the main workspace is a node-based graph showing the pipeline's logic. The graph starts with "Range" nodes leading to "Custom Model Runner" nodes. These connect to "Binary Clip" nodes, which then connect to "Range Value Range" nodes. These nodes connect to "Clip By Value" nodes, which then connect to "Crop And Resize" nodes. The "Crop And Resize" nodes connect to "Tensor To Denoising" nodes, which then connect to "DepthMap Visualizer" nodes. Finally, the "DepthMap Visualizer" nodes connect to "Image Comparison" and "Image Mixer" nodes.

“

It can help me understand **how I should change the model architecture** and what training examples to add.

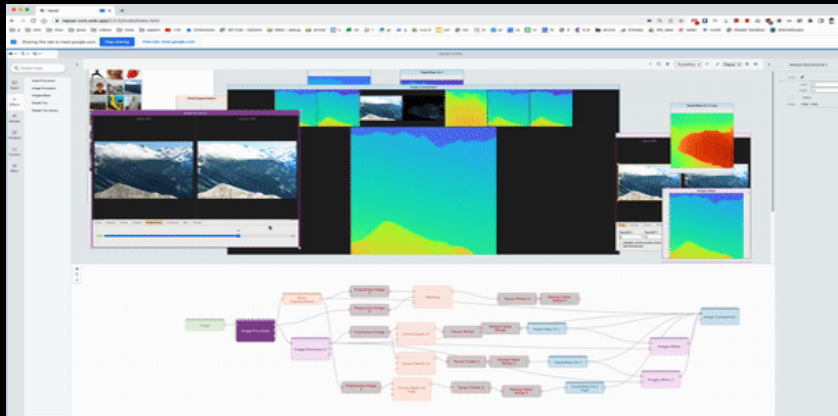
”



“

I can manipulate the brightness to see when the model fails.

”

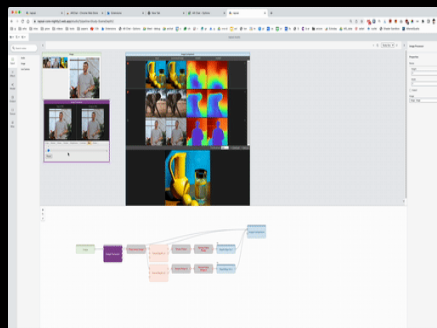


P2

“

It gives me an intuition about **which data augmentation operations that my model is more sensitive**, then I can go back to my training pipeline, maybe increase the amount of data augmentation for those specific steps that are making my model more sensitive.

”



“

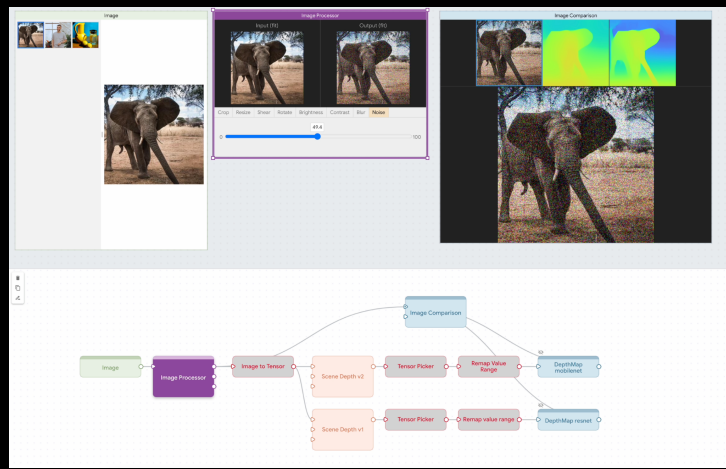
Using a video <as input> helps me get **a cross-time feel** of how the model performance varies, which is hard to capture with metrics.

”

“

Comparing various noise parameters in the input to a model is useful to identify augmentation bias.

”



Findings 3

Rapsai helps Model Selection, Learning From Pipelines, Study deployment

The screenshot displays the Rapsai Studio interface for a pipeline named 'DEPTHMAP-V1'. The interface includes a search bar, a sidebar with tool categories (Model, Input, Effect, Device, Image, Image Comparison, Image Viewer, Join Viewer, Tensor To Dashmap, Tensor To Image, Tensor), and a main workspace. The workspace contains several visual components: an 'Image Comparison' window showing a person's face and its depth map; an 'Image Processor' window with a 'Crop' tool and a 'DepthMap V1' window showing a depth map. Below these are 'Segmented Image' and 'Tensor To Dashmap V1' windows. The main workspace also features a grid of six depth maps and one body segmentation image. At the bottom, a complex flowchart diagram illustrates the pipeline's logic, starting with an 'Image' input, passing through an 'Image Processor' to a 'DepthMap V1' node, which then branches into multiple 'DepthMap V1' nodes, each leading to a 'Body Segmentation' node. The pipeline concludes with an 'Image Comparison' node.

“

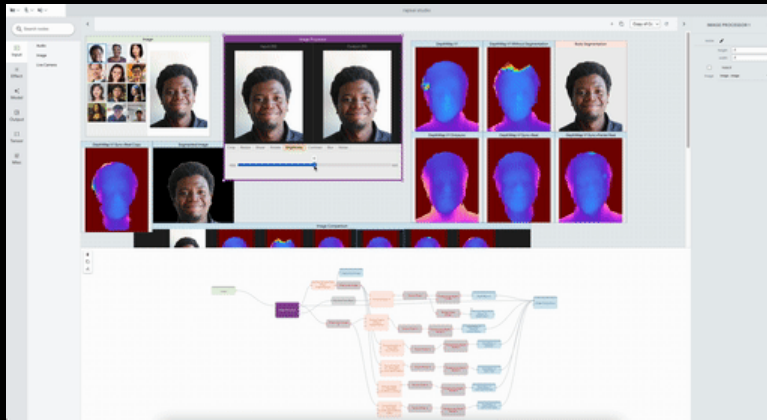
Building a custom webpage as debugging tool [by coding], cost <a junior engineer> over a month to build. This [Rapsai] **is easy to distribute and try it immediately**. It helps debug the pipeline.

”

“

It can help me understand how
should I **change the model
architecture** and **add what
training examples.**

”



P1



VisualBlocks for ML

1. **Lowers the barriers** for ML prototyping
2. Empowers users to **experiment with no/low-code environment**
3. **Facilitates collaboration** between designers and developers

[BLOG](#) ›

Visual Blocks for ML: Accelerating machine learning prototyping with interactive tools

FRIDAY, APRIL 21, 2023

Posted by Ruofei Du, Interactive Perception & Graphics Lead, Google Augmented Reality, and Na Li, Tech Lead Manager, Google CoreML

Recent deep learning advances have enabled a plethora of high-performance, real-time multimedia applications based on machine learning (ML), such as [human body segmentation](#) for video and teleconferencing, [depth estimation](#) for 3D reconstruction, [hand and body tracking](#) for interaction, and [audio processing](#) for remote communication.

However, developing and iterating on these ML-based multimedia prototypes can be challenging and costly. It usually involves a cross-functional team of ML practitioners who fine-tune the models, evaluate robustness, characterize strengths and weaknesses, inspect performance in the end-use context, and develop the applications. Moreover, models are frequently updated and require repeated integration efforts before evaluation can occur, which makes the workflow ill-suited to design and experiment.

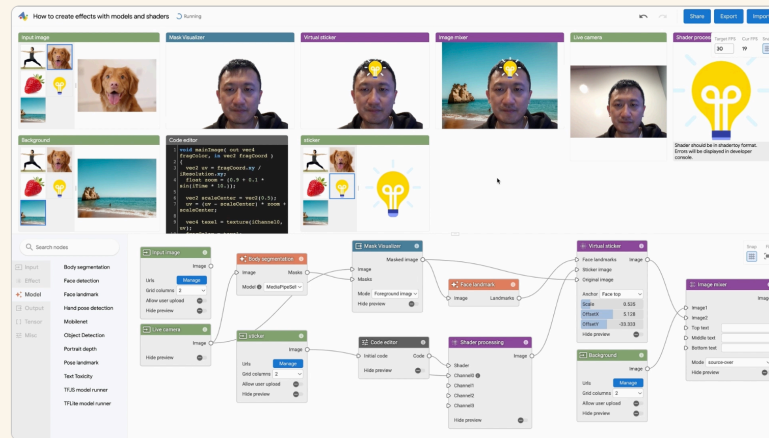
In "[Rapsai: Accelerating Machine Learning Prototyping of Multimedia Applications through Visual Programming](#)", presented at [CHI 2023](#), we describe a visual programming platform for rapid and iterative development of end-to-end ML-based multimedia applications. Visual Blocks for ML, formerly called Rapsai, provides a no-code graph building experience through its [node-graph editor](#). Users can create and connect different components (nodes) to rapidly build an ML pipeline, and see the results in real-time without writing any code. We demonstrate how this platform enables a better model evaluation experience through interactive characterization and visualization of ML model performance and interactive data augmentation and comparison. [Sign up](#) to be notified when Visual Blocks for ML is publicly available.

Visual Blocks for ML

Unleash your creativity

Visual Blocks for ML is an experimental JavaScript framework from Google that helps you add drag-and-drop machine learning blocks to your platform. Only your imagination limits the blocks you give your users. Off the shelf blocks include models, user inputs, processors and visualizations.

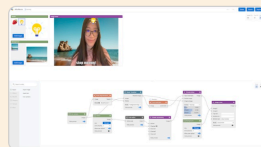
Colab experience and open source library will come soon.



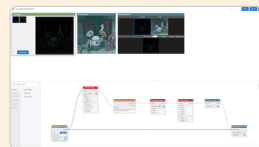
With Visual Blocks for ML, you can drag and drop to make your ML no coding required

Experiment with Visual Blocks

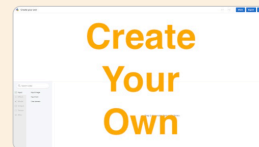
Click on any of the demos to pull up its graph in the interaction editor below



AR Effects



Low Light Enhancement

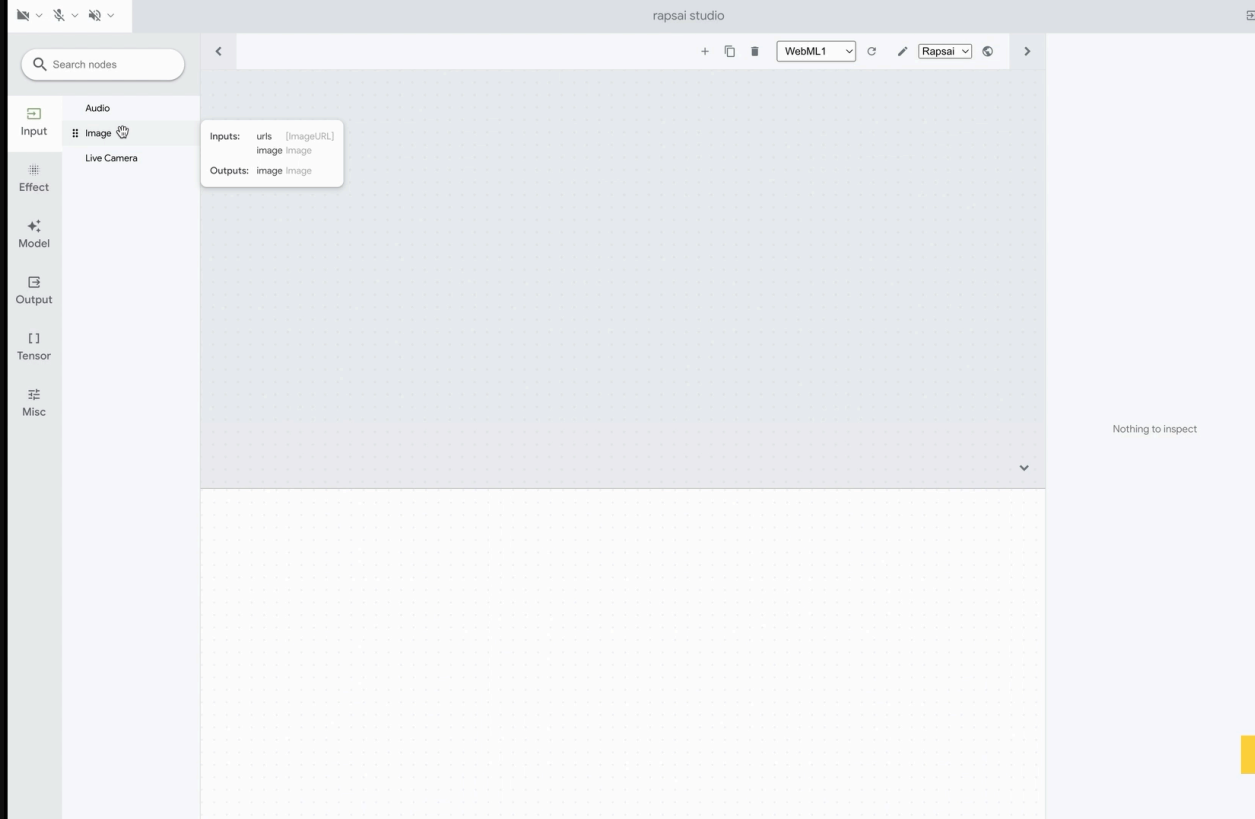
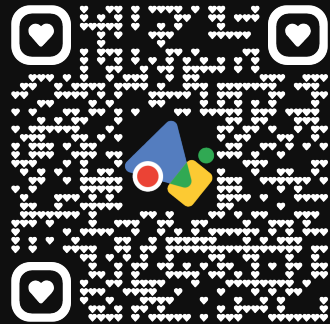


Create Your Own



With the right tools,
everyone can unleash
your inner **creativity**.

Yet another
20% project



Honorable
Mention

visualblocksforml.github.io



Ruofei Du, Na Li, Jing Jin, Michelle Carney, Scott Miles, Maria Kleiner, Xiuxiu Yuan, Yinda Zhang, Anuva Kulkarni, Xingyu "Bruce" Liu, Ahmed Sabie, Sergio Escolano, Abhishek Kar, Ping Yu, Ram Iyengar, Adarsh Kowdle, and Alex Olwal



Search nodes


- Audio
- Image
- Live Camera
- Effect
- Model
- Output
- Tensor
- Misc

webml - aux

Rapsai

Audio

cleaned.wav 3.0s



Denoise - 10db

processed.wav

Start denoising

Denoise - 30db


processed.wav

Start denoising

Audio Comparison

Active track

processed.wav



Automatic track switching

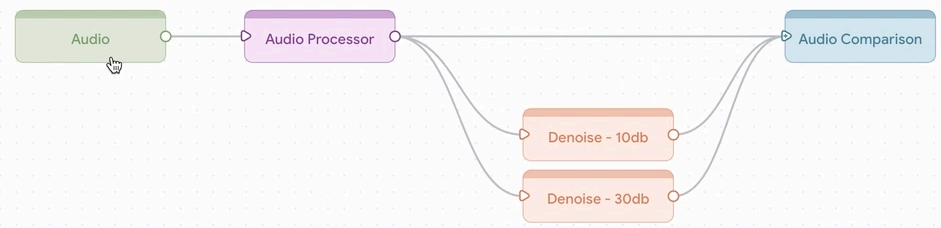
Audio Processor

Input	Output
cleaned.wav 3.0s	processed.wav 3.0s

Trim | Volume | Background Noise

Start (seconds)	End (seconds)
0.00	3.00

Nothing to inspect



“

There are some times when you don't want all noise cancellation. People sometimes prefer audio with less noise cancellation because you want some **context**.

”

Search nodes

aggrating recommendation

Repeat



Audio



Image



Input Stream



Input Text



Use Camera



Depth Audio



Video



Output



Tensor



Misc



Misc

Question

write a **shader/frag** shader to show edges in an image

Use GPU

```

void main() {
  vec2 ip = texture(iChannel0, uv).rgb;
  float g = length(ip) + length(ip);

  // calculate the magnitude of the gradient
  float g = length(ip) + length(ip);

  // use the output color to show gradient magnitude
  float ip = length(ip) + length(ip);
}

```

This shader calculates the gradient of the image in the x and y direction using neighboring pixels. The gradient magnitude is then calculated and used as the output color for each pixel. This will result in an image with strong edges being shown in white and smooth areas being shown in black.

Hope this helped let me know if you have any questions.

Program Shader

code

```

void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
  vec2 uv = fragCoord.xy / (iResolution.xy);
  vec3 c = texture(iChannel0, uv).rgb;

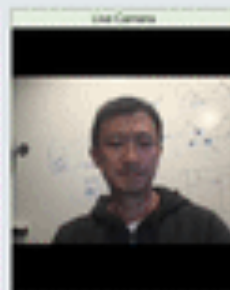
  // calculate the gradient in the x and y direction
  vec2 gx = texture(iChannel0, uv + vec2(1, 0) / iResolution).rgb - c;
  vec2 gy = texture(iChannel0, uv + vec2(0, 1) / iResolution).rgb - c;

  // calculate the magnitude of the gradient
  float g = length(gx) + length(gy);

  // use the output color to show gradient magnitude
  fragColor = vec4(c, g, g, 1);
}

```

output



Question

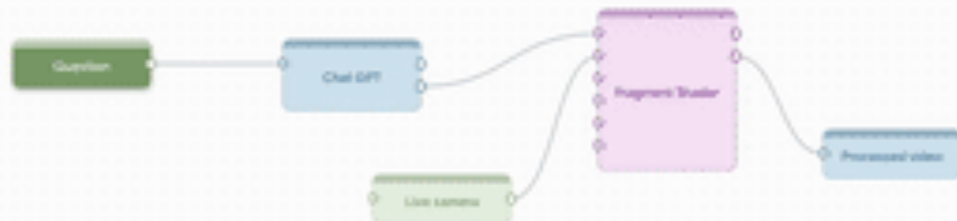
Name	Size	Owner
Avatar	Avatar	Avatar

Outputs: text, multi-output

Properties

 Show

Function

 Hide


Search nodes

- Audio
- Image
- Live Camera
- Effect
- Model
- Output
- Texture
- Misc

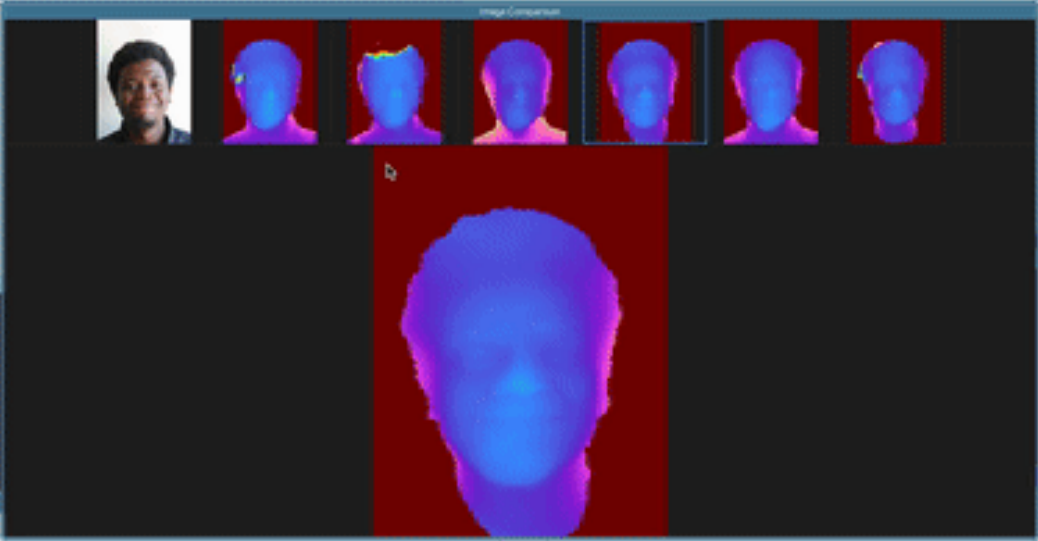
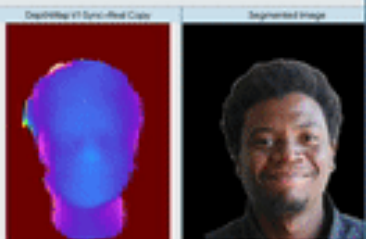


IMAGE COMPRI...

Image: Image
Image: Image
Image: Image
Image: Image

Nodes

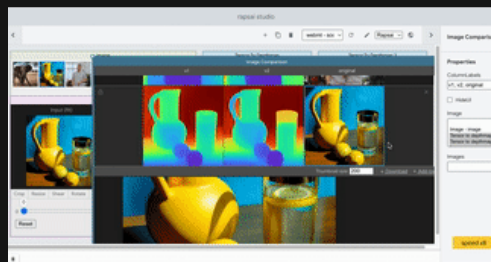


yet another
20% project

Honorable
Mention

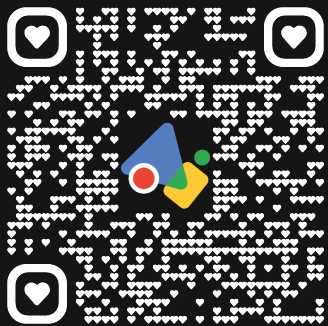


Accelerating Machine Learning Prototyping of Multimedia Applications through Visual Programming



Ruofei Du, Na Li, Jing Jin, Michelle Carney, Scott Miles, Maria Kleiner, Xiuxiu Yuan,
Yinda Zhang, Anuva Kulkarni, Xingyu "Bruce" Liu, Ahmed Sabie, Sergio Escolano,
Abhishek Kar, Ping Yu, Ram Iyengar, Adarsh Kowdle, and Alex Olwal

visualblocksforml.github.io



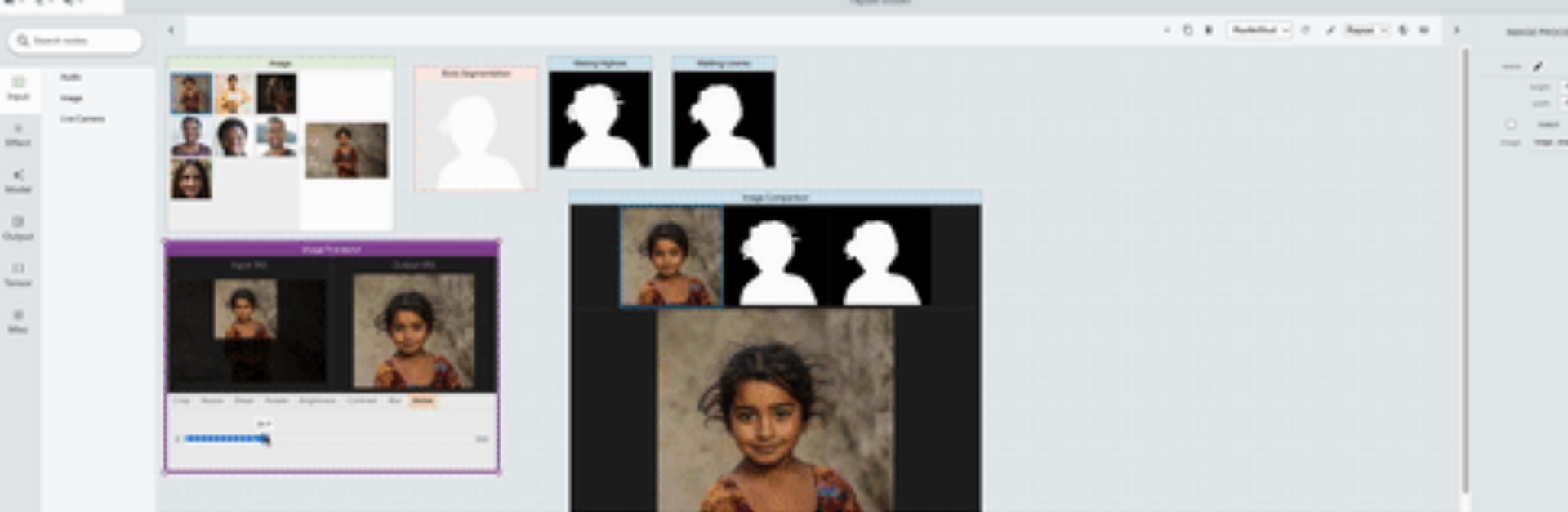


Accelerating Machine Learning Prototyping of Multimedia Applications through Visual Programming

Ruofei Du, Na Li, Jing Jin, Michelle Carney, Scott Miles, Maria Kleiner, Xiuxiu Yuan, Yinda Zhang,
Anuva Kulkarni, Xingyu "Bruce" Liu, Ahmed Sabie, Sergio Escolano, Abhishek Kar,
Ping Yu, Ram Iyengar, Adarsh Kowdle, and Alex Olwal

visualblocksforml.github.io





Provide a visual programming platform for
rapidly building ML prototypes

1

Support **real-time** multimedia
user input **in-the-wild**

2

Provide **interactive** data augmentation

3

Compare model outputs and
render results directly **side-by-side**

4

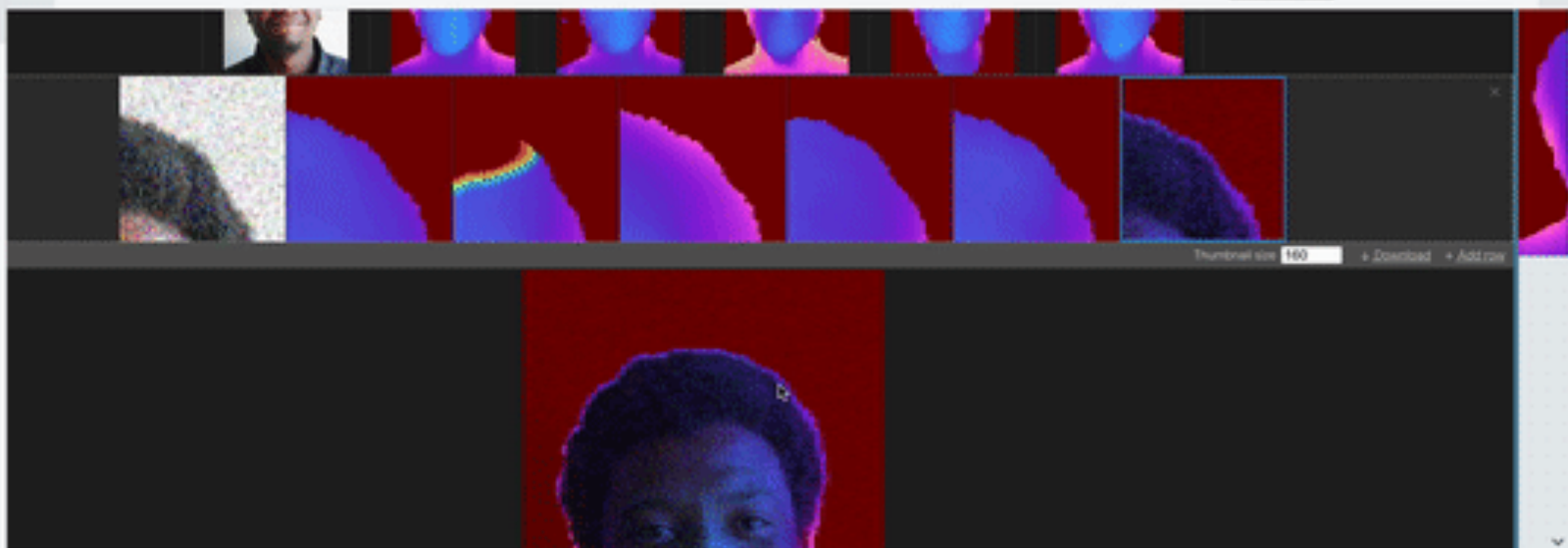
Share visualization with minimum efforts

5

Provide **off-the-shelf** models and datasets

6

There are some times when you don't want all noise cancellation. People sometimes prefer audio with less noise cancellation because you want some **context**.



Search nodes



Audio



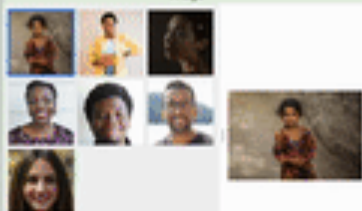
Image



Live Camera



Image



Body Segmentation



Tensor To Image



Image Processor

Input (It)



Output (It)



Crop Resize Shear Rotate Brightness Contrast Blur Noise

Top left X	Top left Y	Bottom right X	Bottom right Y
<input type="text" value="247"/>	<input type="text" value="11"/>	<input type="text" value="474"/>	<input type="text" value="234"/>

 Update continuously when dragging the crop region above (might impact performance)

rapai - Chrome Web Store | Extensions

rapai-core-nightly.web.app/studio/pipeline?groupId=...

who misc grow videos tools papers 1.5h Extensions AI Chat - Options Meet - debug archat

Sharing this tab to meet.google.com [View sharing](#) [View tab: meet.google.com](#)

rapai studio

NO SELECTED NODE

Search nodes

- Audio
- Input
- Image
- Use Camera
- Effect
- Model
- Output
- Tensor
- Misc

Image Processor

Input (0) Output (1) (Full view)

Clear Reset Share Filter **Brightness** Contrast Sat Noise

0.00 1.00

Reset

Body Segmentation

Shader Processing

Attributes (0) (Full view)

Use Camera

The screenshot displays the rapai studio interface. At the top, there's a browser address bar and various extension icons. Below that, a sharing link for a Google Meet tab is visible. The main workspace is titled 'rapai studio' and shows a pipeline of four nodes: 'Live Camera' (green), 'Image Processor' (purple), 'Body Segmentation' (orange), and 'Shader Processing' (purple). The 'Image Processor' node is expanded, showing a 'Brightness' slider and other controls. The 'Body Segmentation' node shows a person's face with a white mask. The 'Shader Processing' node shows the same person with a more complex, colorful mask. A 'Shader Library' node is also connected to the 'Shader Processing' node. On the left, a sidebar lists various node categories like Audio, Input, Image, etc. At the bottom, a search bar and a list of node categories are visible.

Google.com - Calendar - Tue... x Formal Case Study of Rap... x Rapnai Data Collection Cons... x rapnai x Page Not Found x x rapnai x Meet - [Rapnai] Onsite Co... x x

rapnai-core-nightly-web-app/studio/

who miss grow videos tools papers 1.5h Extensions All Chat - Options Meet - debug archat ch f gr b vns-d pdf ton m pf cs cr Ts S b ar score Ertoday

Sharing this tab to meet.google.com Stop sharing View tab: meet.google.com

rapnai studio

Search nodes

Audio
Input
Image
Live Camera

Effect
Model
Output
Tensor
Misc

Image

Body Segmentation

Mating Highres

Mating Lowres

Highres O

Image Processor

Input (10)

Output (10)

Crop Resize Shear Rotate Brightness Contrast Blur Noise

Top-left X: 236 Top-left Y: 0 Bottom-right X: 488 Bottom-right Y: 227

Update continuously when dragging the crop region above (might impact performance)

Image Comparison

source mating1 mating2 mating1 low res

IMAGE COMPARISON 1

columnsLabels: source, mating1, ma

mask

Image: Image - image
Body Segmentation - outputImage
Image processor - outputImage
Resize to image - outputImage

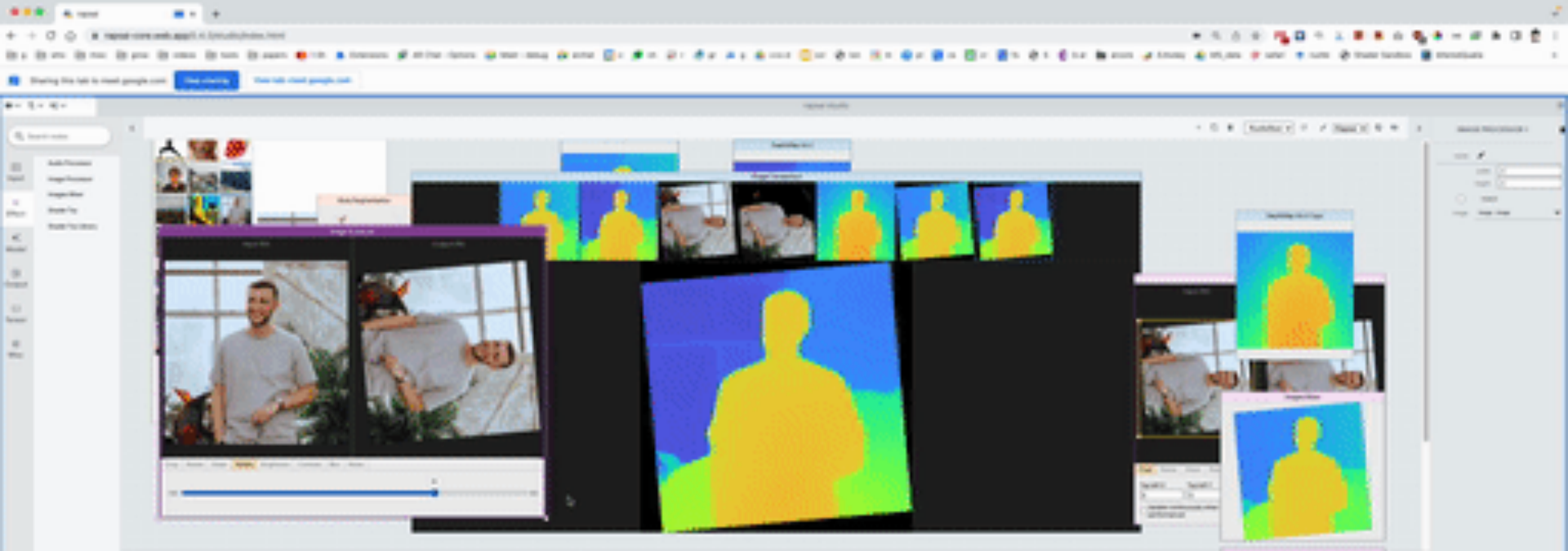
Images

- Audio
- Input
 - Image
 - Live Camera
- Effect
- Model
- Output
- Tensor
- Misc

+ WebML1 Rapisai

Nothing to inspect





This image shows a screenshot of a software interface, likely a video analysis tool, displaying a sequence of frames and their corresponding feature maps. The interface includes a search bar, a sidebar with navigation options, and a main workspace.

The main workspace is divided into two main sections. The top section displays a sequence of frames and their corresponding feature maps. The frames are arranged in a grid, showing a person standing in a room and a yellow pitcher on a table. The feature maps are color-coded, with red and yellow indicating high activation and blue indicating low activation. The feature maps for the person frames show a clear silhouette of the person, while the feature maps for the pitcher frames show a clear silhouette of the pitcher.



The bottom section displays a flowchart diagram illustrating the processing pipeline. The flow starts with an input frame, which is processed by a "Convolutional Layer". The output of this layer is then processed by two parallel "Feature Maps". Each feature map is processed by a "Max Pooling" layer, followed by a "Softmax" layer. The outputs of the two Softmax layers are then combined to produce the final output.

The flowchart diagram is as follows:

```

    graph LR
      Input[Input] --> Conv[Convolutional Layer]
      Conv --> FM1[Feature Map 1]
      Conv --> FM2[Feature Map 2]
      FM1 --> MP1[Max Pooling]
      FM2 --> MP2[Max Pooling]
      MP1 --> S1[Softmax]
      MP2 --> S2[Softmax]
      S1 --> Output[Output]
      S2 --> Output
  
```

Boxer: Interactive Comparison of Classifier Results

Michael Gleicher , Aditya Barve, Xinyi Yu, and Florian Heimert 

University of Wisconsin - Madison

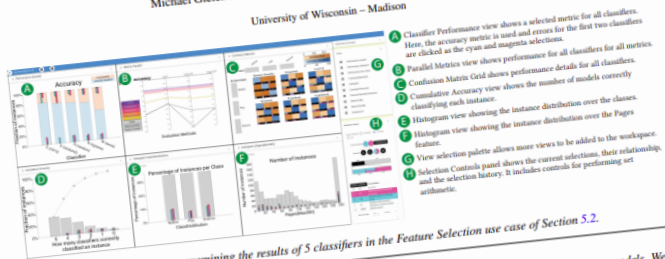


Figure 1: Boxer examining the results of 5 classifiers in the Feature Selection use case of Section 5.2.

Abstract
Machine learning practitioners often compare the results of different classifiers to help select, diagnose and tune models. We present Boxer, a system to enable such comparison. Our system facilitates interactive exploration of the experimental results obtained by applying multiple classifiers to a common set of model inputs. The approach focuses on allowing the user to identify interesting subsets of training and testing instances and comparing performance of the classifiers on these subsets. The system couples standard visual designs with set algebra interactions and comparative elements. This allows the user to compose and coordinate views to specify subsets and assess classifier performance on them. The flexibility of these compositions allow the user to address a wide range of scenarios in developing and assessing classifiers. We demonstrate Boxer in use cases including model selection, tuning, fairness assessment, and data quality diagnosis.

CCS Concepts

• Human-centered computing → Visualization; Visual analytics; Information visualization;

1. Introduction

Machine learning practitioners often perform experiments that compare classification results. Users gather the results of different classifiers or data perturbations on a collection of testing examples. Results are stored and analyzed for tasks such as model selection, hyper-parameter tuning, data quality assessment, fairness testing, and to gain insight about the underlying data. Classifier comparison, however, is not a standard part of the machine learning workflow. Summary statistics

miss important aspects of classifier performance. For closer examination, practitioners rely on scripting in their standard workflows. The lack of specific tooling makes the process laborious and comparisons challenging, limiting how often experiments are examined in detail.

This paper presents Boxer (Figure 1), a system for the detailed examination of classifier comparison experiments. Our approach allows a user to explore a collection of classifier results to identify interesting subsets of the data and compare performance across them. Boxer enhances standard views with interactions for set arithmetic and set algebra. It provides a uniform mechanism

