KENTECH
Korea Institute of Energy Technology

Visionary Course – Energy AI
Week 12

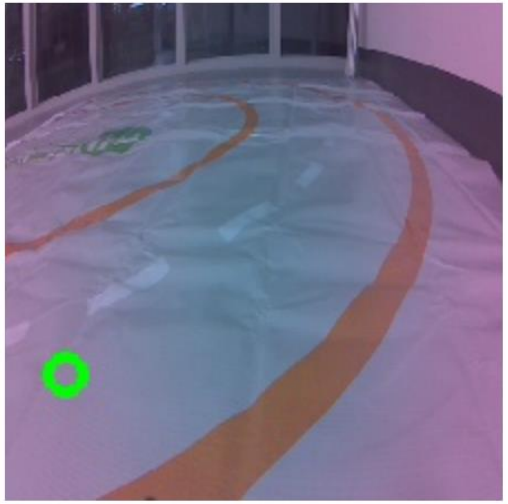
Nov 17, 2022
Sohee Kim, Giwon Sur

KENTECH
Korea Institute of Energy Technology

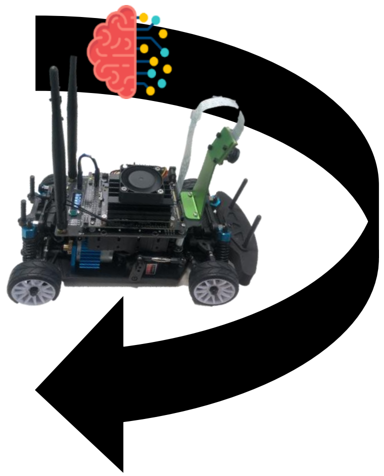
Week 12b – Interactive regression for road following on Jetson Nano

JetRacer: Let's continue autonomous driving!

➡ Today we will train model on the road with 2D-lined track.



Train model!



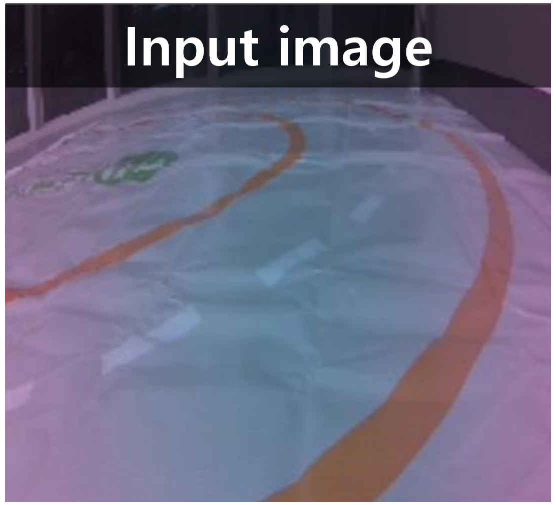
Green point
= Human's annotation
= Ground Truth (GT)

Blue point
= Model's output
= Prediction

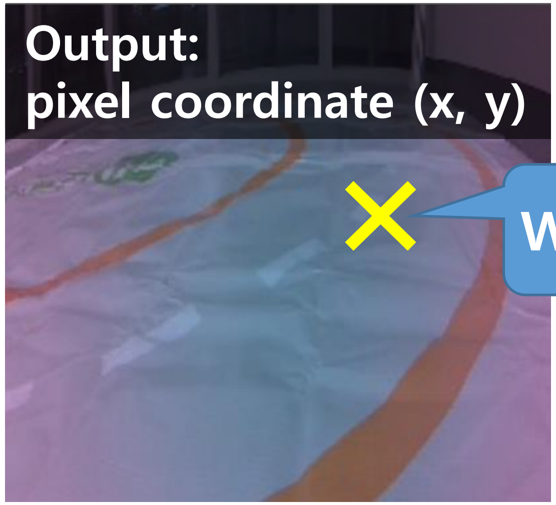
Workspace:

"localhost:8888/lab/tree/jetracer/notebooks/interactive_regression.ipynb"

JetRacer: Let's continue autonomous driving!



AI Model



Where to go?

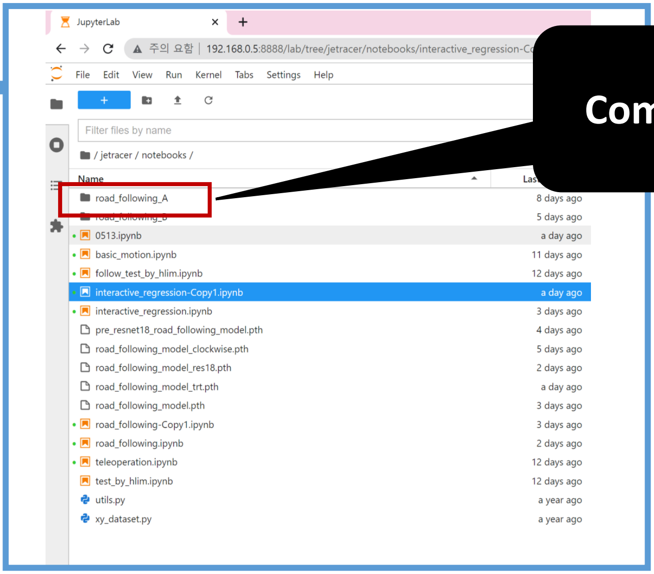
Deep Learning!

Controller

Steering!

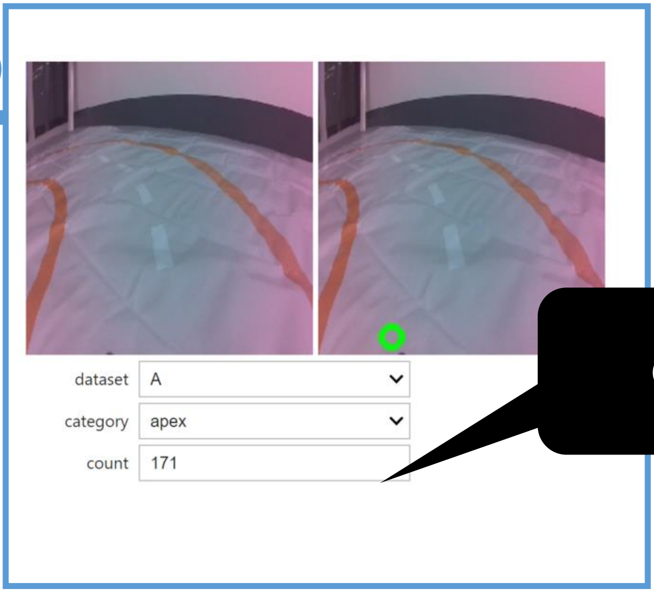
Overview of interactive_regression.ipynb

1



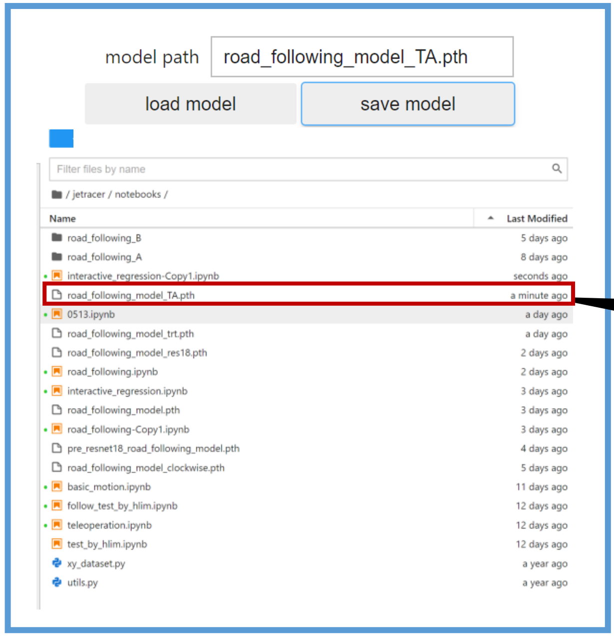
Composite dataset

2



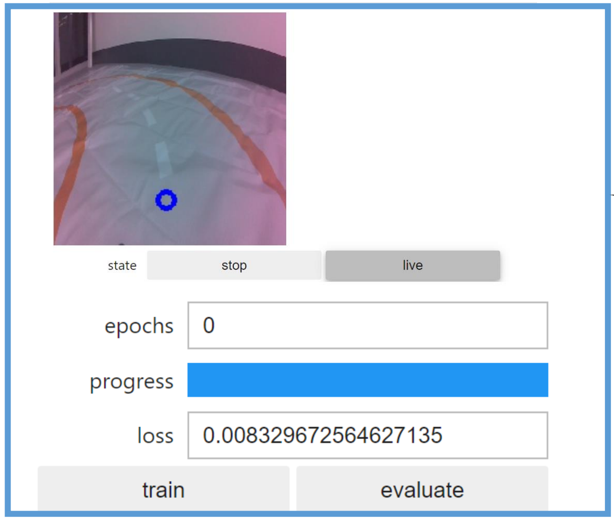
Collect data

3



Make model

4



Train & evaluate

1. Composite dataset

```python
import torchvision.transforms as transforms
from xy_dataset import XYDataset

TASK = 'road_following'

CATEGORIES = ['apex']

DATASETS = ['A', 'B']

TRANSFORMS = transforms.Compose([
    transforms.ColorJitter(0.2, 0.2, 0.2, 0.2),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

datasets = {}
for name in DATASETS:
    datasets[name] = XYDataset(TASK + '_' + name, CATEGORIES, TRANSFORMS, random_hflip=True)
```

Data augmentation

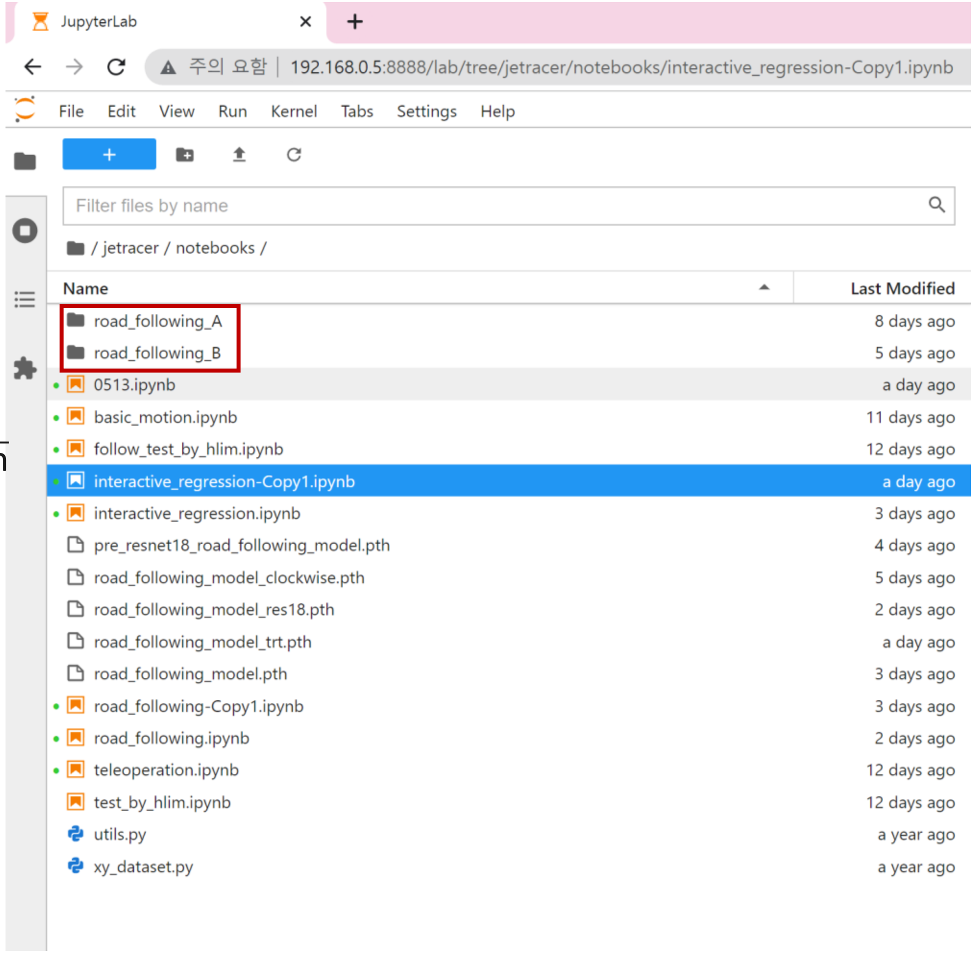
brightness, contrast, saturation, hue

mean

standard deviation

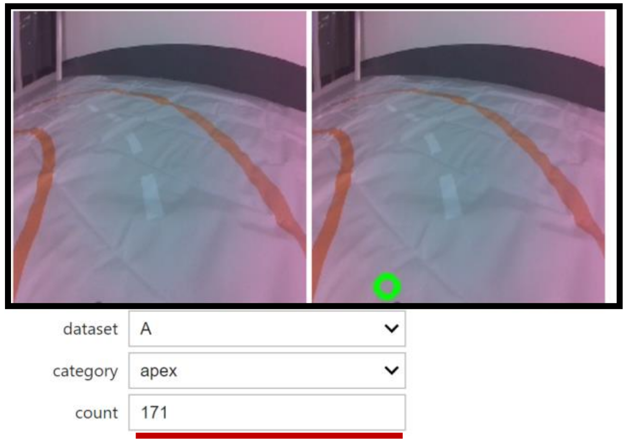
$$\frac{\text{data} - \text{mean}}{\text{Standard deviation}}$$

Load data if it exists



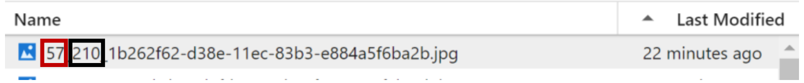
2. Collect data

```python
import cv2
import ipywidgets
import traitlets
from IPython.display import display
from jetcam.utils import bgr8_to_jpeg
from jupyter_clickable_image_widget import ClickableImageWidget


# initialize active dataset
dataset = datasets[DATASETS[0]]

# unobserve all callbacks from camera in case we are running this cell for second time
camera.unobserve_all()

# create image preview
camera_widget = ClickableImageWidget(width=camera.width, height=camera.height)
snapshot_widget = ipywidgets.Image(width=camera.width, height=camera.height)
traitlets.dlink((camera, 'value'), (camera_widget, 'value'), transform=bgr8_to_jpeg)

# create widgets
dataset_widget = ipywidgets.Dropdown(options=DATASETS, description='dataset')
category_widget = ipywidgets.Dropdown(options=dataset.categories, description='category')
count_widget = ipywidgets.IntText(description='count')

# manually update counts at initialization
count_widget.value = dataset.get_count(category_widget.value)

# sets the active dataset
def set_dataset(change):
    global dataset
    dataset = datasets[change['new']]
    count_widget.value = dataset.get_count(category_widget.value)
dataset_widget.observe(set_dataset, names='value')

# update counts when we select a new category
def update_counts(change):
    count_widget.value = dataset.get_count(change['new'])
category_widget.observe(update_counts, names='value')


def save_snapshot(_, content, msg):
    if content['event'] == 'click':
        data = content['eventData']
        x = data['offsetX']
        y = data['offsetY']

        # save to disk
        dataset.save_entry(category_widget.value, camera.value, x, y)

        # display saved snapshot
        snapshot = camera.value.copy()
        snapshot = cv2.circle(snapshot, (x, y), 8, (0, 255, 0), 3)
        snapshot_widget.value = bgr8_to_jpeg(snapshot)
        count_widget.value = dataset.get_count(category_widget.value)

camera_widget.on_msg(save_snapshot)
```



Count total snapshot

```python
def save_entry(self, category, image, x, y):
    category_dir = os.path.join(self.directory, category)
    if not os.path.exists(category_dir):
        subprocess.call(['mkdir', '-p', category_dir])

    filename = '%d_%d_%s.jpg' % (x, y, str(uuid.uuid1()))

    image_path = os.path.join(category_dir, filename)
    cv2.imwrite(image_path, image)
    self.refresh()
```



x y

Saved snapshot get coordination in the file name

3. Define model

```python
import torch
import torchvision

device = torch.device('cuda')
output_dim = 2 * len(dataset.categories)  # x, y coordinate for each category

# ALEXNET
# model = torchvision.models.alexnet(pretrained=True)
# model.classifier[-1] = torch.nn.Linear(4096, output_dim)

# SQUEEZENET
# model = torchvision.models.squeezenet1_1(pretrained=True)
# model.classifier[1] = torch.nn.Conv2d(512, output_dim, kernel_size=1)
# model.num_classes = len(dataset.categories)

# RESNET 18
model = torchvision.models.resnet18(pretrained=True)
model.fc = torch.nn.Linear(512, output_dim)

# RESNET 34
# model = torchvision.models.resnet34(pretrained=True)
# model.fc = torch.nn.Linear(512, output_dim)

# DENSENET 121
# model = torchvision.models.densenet121(pretrained=True)
# model.classifier = torch.nn.Linear(model.num_features, output_dim)

model = model.to(device)

model_save_button = ipywidgets.Button(description='save model')
model_load_button = ipywidgets.Button(description='load model')
model_path_widget = ipywidgets.Text(description='model path', value='road_following_model.pth')

def load_model(c):
    model.load_state_dict(torch.load(model_path_widget.value))
model_load_button.on_click(load_model)

def save_model(c):
    torch.save(model.state_dict(), model_path_widget.value)
model_save_button.on_click(save_model)

model_widget = ipywidgets.VBox([
    model_path_widget,
    ipywidgets.HBox([model_load_button, model_save_button])
])


display(model_widget)
```

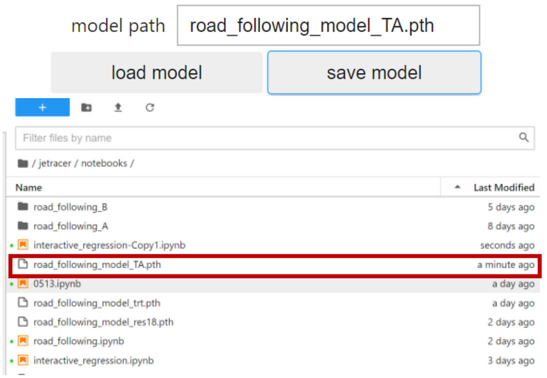
Declare specific model (resnet18 here)

Set output dim's as 2

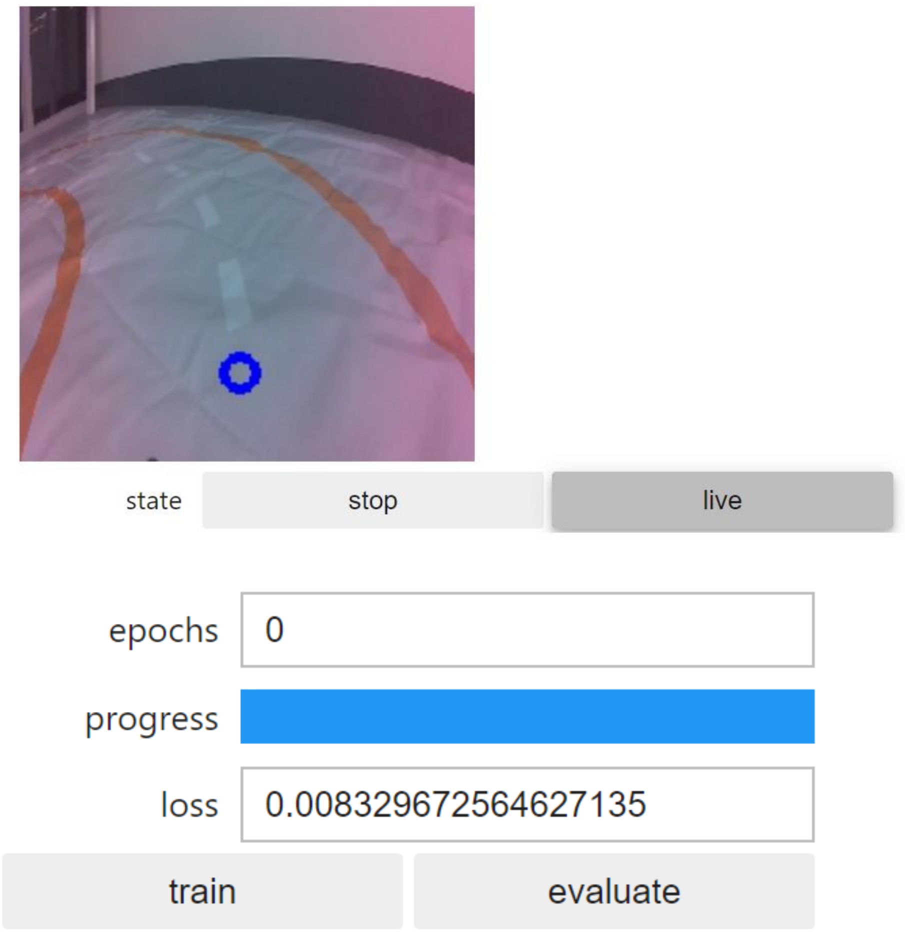
Load model to GPU

Make widgets for loading and saving model

Model's name



4. Train & evaluation




```python
BATCH_SIZE = 8

optimizer = torch.optim.Adam(model.parameters())
# optimizer = torch.optim.SGD(model.parameters(), lr=1e-3, momentum=0.9)

epochs_widget = ipywidgets.IntText(description='epochs', value=1)
eval_button = ipywidgets.Button(description='evaluate')
train_button = ipywidgets.Button(description='train')
loss_widget = ipywidgets.FloatText(description='loss')
progress_widget = ipywidgets.FloatProgress(min=0.0, max=1.0, description='progress')
```

4. Train & evaluation

```python
def train_eval(is_training):
    global BATCH_SIZE, LEARNING_RATE, MOMENTUM, model, dataset, optimizer, eval_button, train_button, accuracy_widget, loss_widget, progress_widget, state_

    try:
        train_loader = torch.utils.data.DataLoader(
            dataset,
            batch_size=BATCH_SIZE,
            shuffle=True
        )

        state_widget.value = 'stop'
        train_button.disabled = True
        eval_button.disabled = True
        time.sleep(1)

        if is_training:
            model = model.train()
        else:
            model = model.eval()

        while epochs_widget.value > 0:
            i = 0
            sum_loss = 0.0
            error_count = 0.0
            for images, category_idx, xy in iter(train_loader):
                # send data to device
                images = images.to(device)
                xy = xy.to(device)

                if is_training:
                    # zero gradients of parameters
                    optimizer.zero_grad()

                # execute model to get outputs
                outputs = model(images)

                # compute MSE loss over x, y coordinates for associated categories
                loss = 0.0
                for batch_idx, cat_idx in enumerate(list(category_idx.flatten())):
                    loss += torch.mean((outputs[batch_idx][2 * cat_idx:2 * cat_idx+2] - xy[batch_idx])**2)
                loss /= len(category_idx)

                if is_training:
                    # run backpropogation to accumulate gradients
                    loss.backward()

                    # step optimizer to adjust parameters
                    optimizer.step()

                # increment progress
                count = len(category_idx.flatten())
                i += count
                sum_loss += float(loss)
                progress_widget.value = i / len(dataset)
                loss_widget.value = sum_loss / i

            if is_training:
                epochs_widget.value = epochs_widget.value - 1
            else:
                break
    except e:
        pass
```

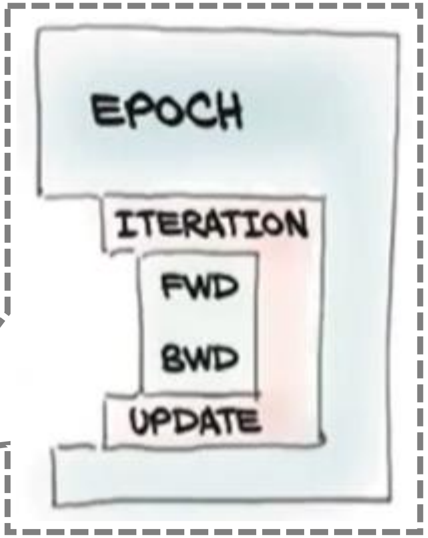
Epoch → Number of iterations over the entire dataset

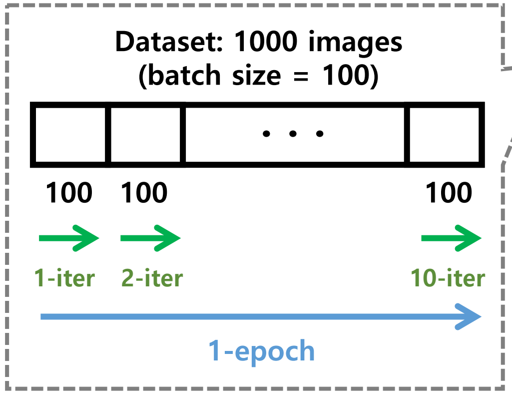
Iteration → Mini-batch iterations

Forward → Predict output

→ Calculate loss

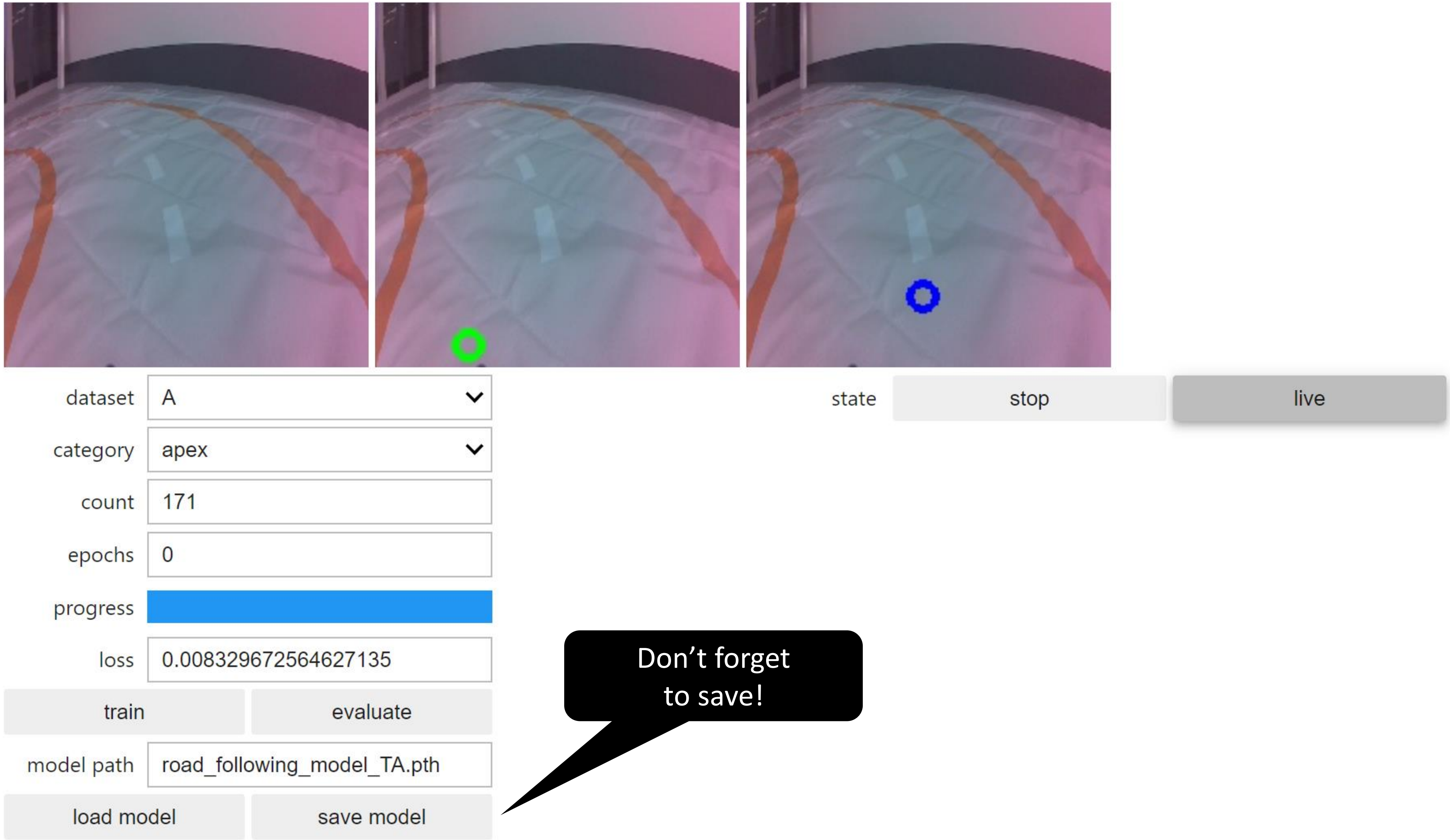
Backward → Calculate gradients and update the model (back-propagation)





Epoch:
One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE.
Batch size:
Total number of training examples present in a single batch.
Iteration:
The number of passes to complete one epoch.

4. Train & evaluation



Don’t forget
to save!

Experiments (Submit the report by 11/24 11:59PM)

1. Run interactive regression.

Q.1.1. Collect data in ‘A’ and train the model.
Q.1.2. Repeat collection and training and evaluation alternatively until the count no larger than 30.
Q.1.3. Save the model.

2. Run interactive regression with more data in another dataset with a different model.

Q.2.1. Collect data in ‘B’ and train the model with a different named model from 1.
Q.2.2. Repeat collect data and training and evaluation alternatively until the count becomes 150~200.
Q.2.3. Save the model.

3. Change the batch size and epoch and train the model

Q.3.1. Set batch size (original 8) to 4 or 16. and train the model. Can you observe the training time is different from 2?
Q.3.2. Set epoch size 5 and train. Observe the loss changes and evaluation result in the blue circle.
Q.3.3. Again, set epoch size 10 and train. Observe the loss changes and evaluation results in the blue circle. Compared to Q.3.2. how the result change?

Experiments (Submit the report by 11/24 11:59PM)

4. Run the trained models.

Q.4.1. With the trained models, run the road following.
Q.4.2. When observing the performance, which one is better? (trained with 30 or 150~200)
Q.4.3. Run the better model, and compare the model with "road_following_model_gwsur.pth"

5. Train the model on different environment

Q.5.1. Place the car outside of the lane but head to the track.
Q.5.2. With the trained model, run the road following. Does the car get into the lane?
Q.5.3. If the car keeps moving outside, please train the model.
Q.5.4. Place the track in diverse places under different illumination conditions.
Does the car drive well?
Q.5.5. If the car doesn't work well, please train the model again.
Q.5.6. Measure the lab time of your model. Make it faster using y axis! Do your best.