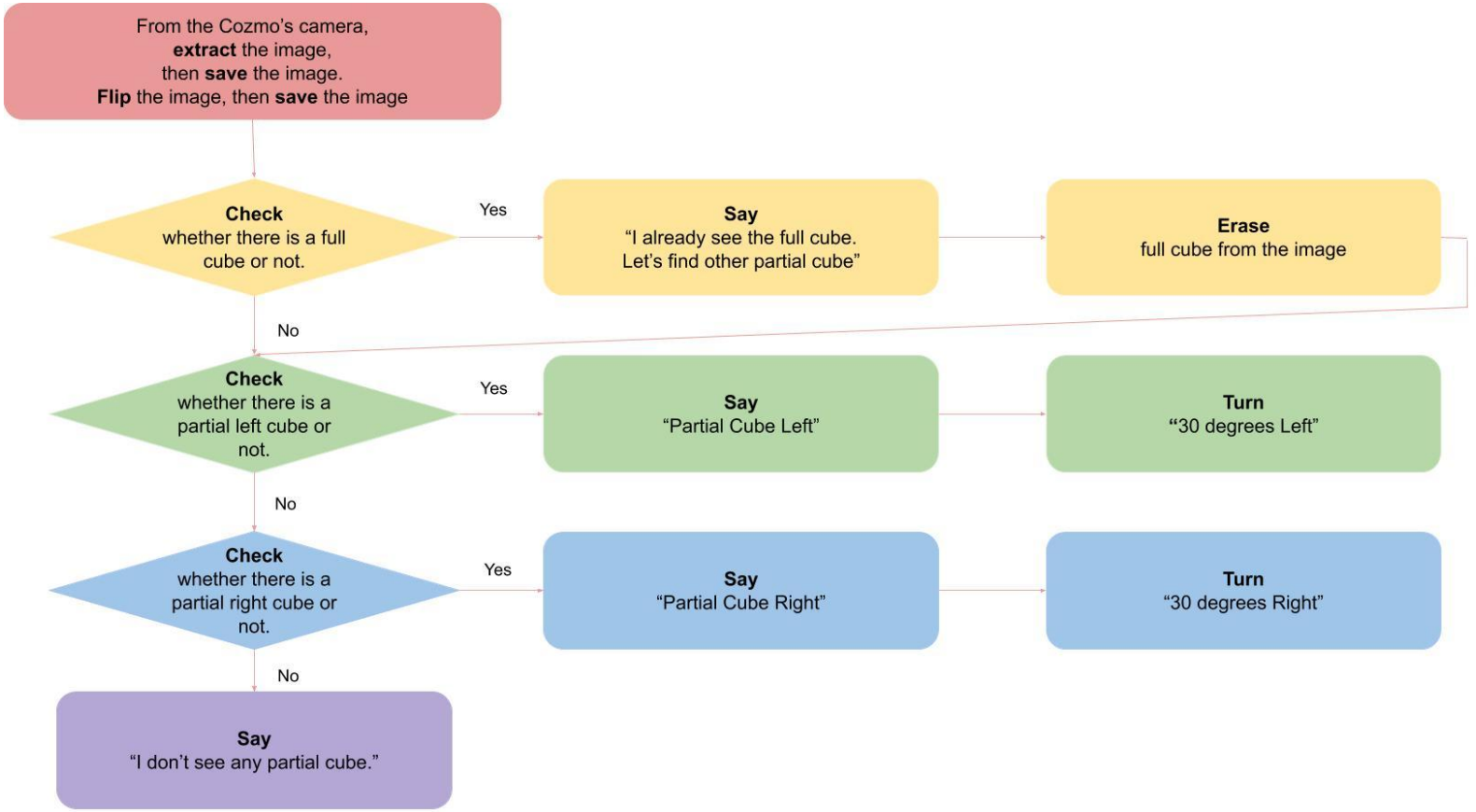
# 15-694 Cognitive Robotics Final Project

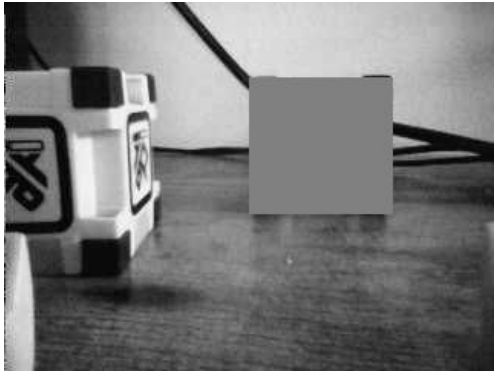
Partial Cube Detection

JeongKeun (Jake) Shin

# Workflow


From the Cozmo's camera,
**extract** the image,
then **save** the image.
**Flip** the image, then **save** the image
**Check**
whether there is a full
cube or not.
Yes
**Say**
"I already see the full cube.
Let's find other partial cube"
**Erase**
full cube from the image
No
**Check**
whether there is a
partial left cube or
not.
Yes
**Say**
"Partial Cube Left"
**Turn**
"30 degrees Left"
No
**Check**
whether there is a
partial right cube or
not.
Yes
**Say**
"Partial Cube Right"
**Turn**
"30 degrees Right"
No
**Say**
"I don't see any partial cube."

# ‘Erase the full cube’ process

- Sometimes, Cozmo can see the full cube and the partial cube together.
- At this case, as Cozmo automatically detects the full cube, then apply the bounding box on the full cube. By using this bounding box, Cozmo erase the full cube from the image by changing all pixels in the bounding box as 0
- Then, save the modified image, and horizontally flipped image.
- With these modified images, now without the full cube, Cozmo starts to detects the partial cube.

# CNN Architecture

| Input (320 width x 240 height image) |
| --- |
| Conv5-32 |
| BatchNorm2d-32 |
| ReLU |
| MaxPool 4x4 |
| Conv5-32 |
| ReLU |
| MaxPool 4x4 |
| Flatten |
| Linear-10 |
| Linear-10 |
| Linear-2 |

```
class CubeDetector2(nn.Module):
  def __init__(self, in_dim, nkernels1=32, nkernels2=32, pool1=4, pool2=4):
    super(CubeDetector2, self).__init__()
    self.network = nn.Sequential(
      nn.Conv2d(in_channels=1, out_channels=nkernels1, kernel_size=5, stride=1, padding=2),
      nn.BatchNorm2d(nkernels1),
      nn.ReLU(),
      nn.MaxPool2d(kernel_size=pool1),
      nn.Conv2d(in_channels=nkernels1, out_channels=nkernels2, kernel_size=5, stride=1, padding=2),
      nn.BatchNorm2d(nkernels2),
      nn.ReLU(),
      nn.MaxPool2d(kernel_size=pool2),
      nn.Flatten(),
      nn.Linear(int(in_dim[0]*in_dim[1]/(pool1**2*pool2**2))*nkernels2, 10),
      nn.Linear(10, 10),
      nn.Linear(10, 2)
    )

  def forward(self,input):
    return self.network(input)
```

Result

Training Set Accuracy: 99% (1323 / 1324)
Test Set Accuracy: 92% (48 / 52)

# Deep Learning Experiments & Analysis

All outcomes are the average of 3 experiments results

# Optimizer: SGD vs Adam

| torch.optim.SGD | | | |
|---|---|---|---|
| **Epochs** | **Learning Rate** | **Momentum** | **Testset Accuracy** |
| 250 epochs | 0.0001 | 0.9 | 65% |
| **250 epochs** | **0.0005** | **0.9** | **89%** |
| 150 epochs | 0.001 | 0.9 | 82% |

| torch.optim.Adam | | |
|---|---|---|
| **Epochs** | **Learning Rate** | **Testset Accuracy** |
| 150 epochs | 0.0001 | 80% |
| 250 epochs | 0.0001 | 82% |
| **250 epochs** | **0.0005** | **91%** |
| 150 epochs | 0.001 | 85% |

- Compared the performance of SGD optimizer and Adam optimizer in same condition
- Adam optimizer tend to show the better performance when other factors stay fixed.
- Both SGD Optimizer and Adam optimizer shows the best performance when learning rate is ***0.0005*** and the number of epochs are ***250***

# Average Pooling vs Max Pooling

```
self.network = nn.Sequential(
  nn.Conv2d(in_channels=1, out_channels=nkernels1, kernel_size=5, stride=1, padding=2),
  nn.BatchNorm2d(nkernels1),
  nn.ReLU(),
  nn.MaxPool2d(kernel_size=pool1),
  nn.Conv2d(in_channels=nkernels1, out_channels=nkernels2, kernel_size=5, stride=1, padding=2),
  nn.BatchNorm2d(nkernels2),
  nn.ReLU(),
  nn.MaxPool2d(kernel_size=pool2),
  nn.Flatten(),
  nn.Linear(int(in_dim[0]*in_dim[1]/(pool1**2*pool2**2))*nkernels2, 10),
  nn.Linear(10, 10),
  nn.Linear(10, 2)
)
```

- Max Pooling is used in the optimal network twice, I switched them to Average Pooling to check how that makes a difference
- Ran 250 epochs, and used Adam optimizer.
- Result: Max Pooling shows better performance than Average Pooling in both training set and test set

| | Training Set Accuracy | Test Set Accuracy |
|---|---|---|
| **Average Pooling** | 99% (1318 / 1324) | 75% (42 / 52) |
| **Max Pooling** | 99% (1323 / 1324) | 92% (48 / 52) |

# Sigmoid, ReLU, and Linear Layers

```
nn.Linear(int(in_dim[0]*in_dim[1]/(pool1**2*pool2**2))*nkernels2, 10), #1st Layer
nn.Linear(10, 10), #2nd Layer
nn.Linear(10, 2) #3rd Layer
```

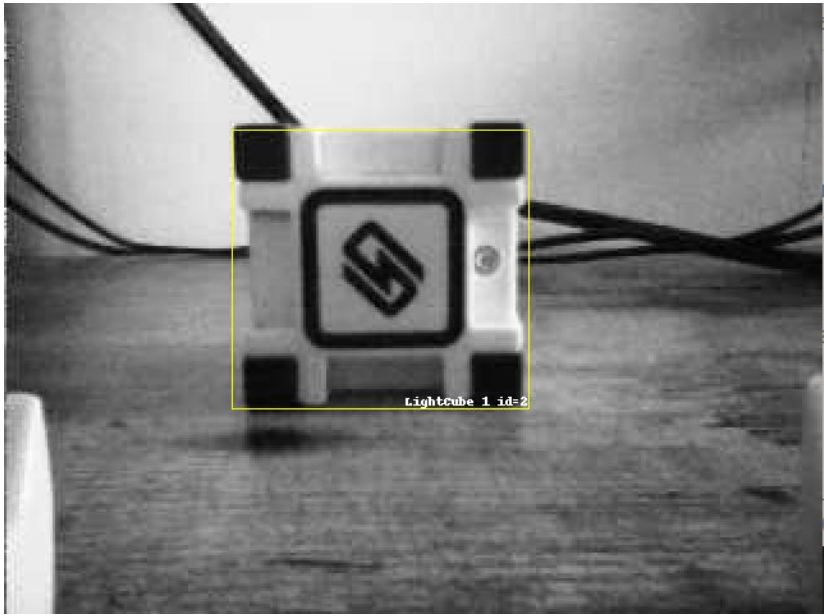
| When Sigmoid activation function was used | | |
|---|---|---|
| | **Training Set Accuracy** | **Test Set Accuracy** |
| **1 Linear Layer** | 90% | 80% |
| **2 Linear Layers** | 90% | 69% |
| **3 Linear Layers** | 85% | 73% |

| When ReLU activation function was used | | |
|---|---|---|
| | **Training Set Accuracy** | **Test Set Accuracy** |
| **1 Linear Layer** | 99% | 84% |
| **2 Linear Layers** | 99% | 84% |
| **3 Linear Layers** | 99% | 92% |

- Compared the performance of the ReLU and Sigmoid
- Also, experimented how performance changes as more linear layers are added at the end
- As a result, ReLU tends to show better performance than Sigmoid
- As more linear layers are added with Sigmoid activation functions, the performance tends to get worse.
- As more linear layers are added with ReLU activation functions, the performance tends to get better.

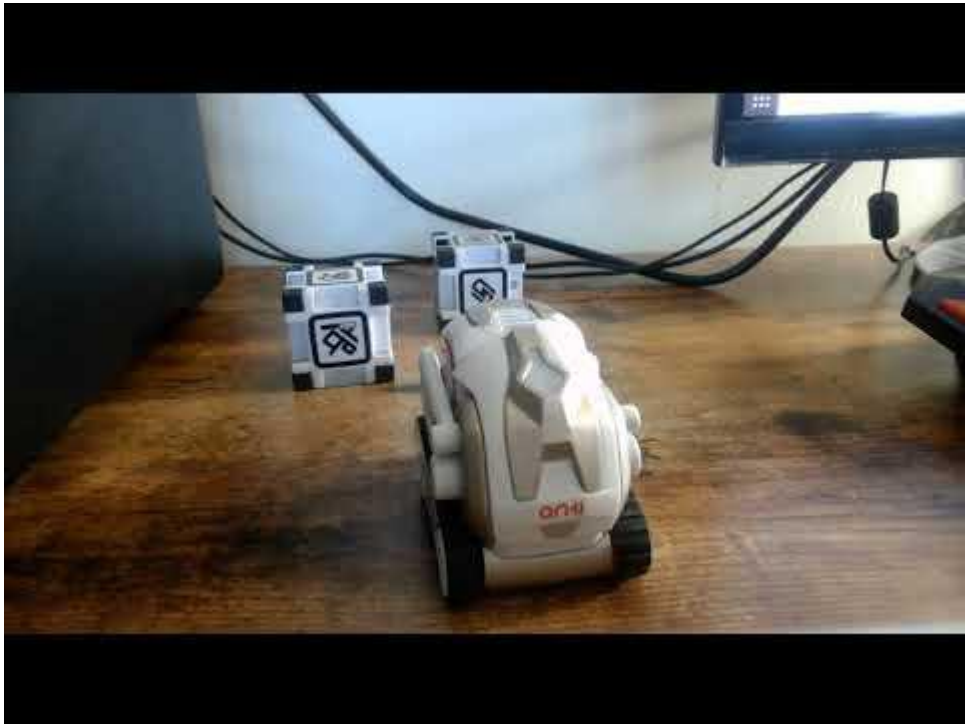
# Demo

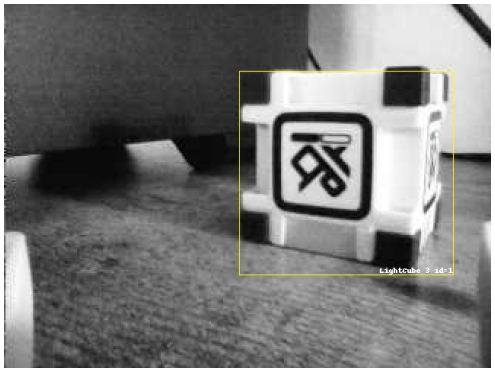
# When Cozmo sees full cube with no partial cube




https://www.youtube.com/watch?v=4QBaHlsAA-4

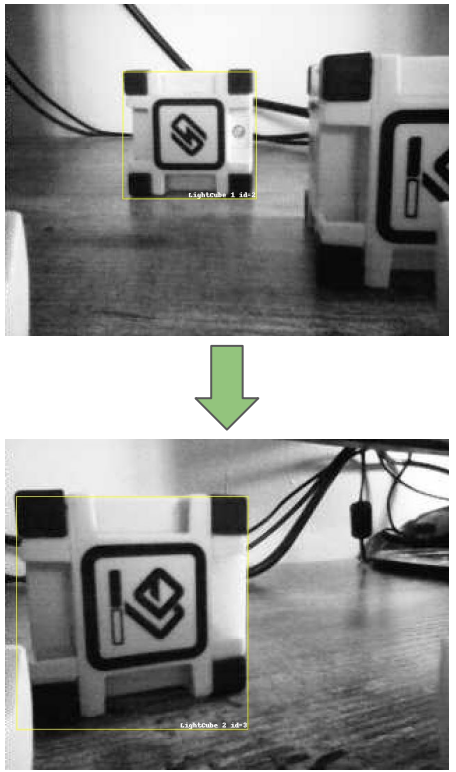
# When Cozmo sees full cube with left partial cube





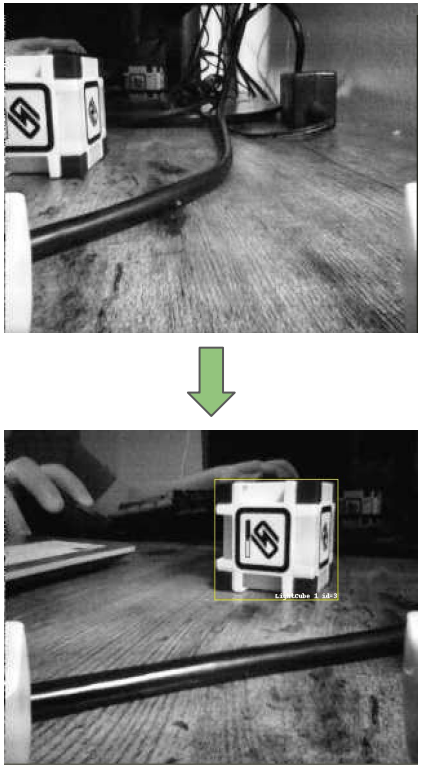
https://www.youtube.com/watch?v=lZ9sL7kmJww

# When Cozmo sees full cube with right partial cube



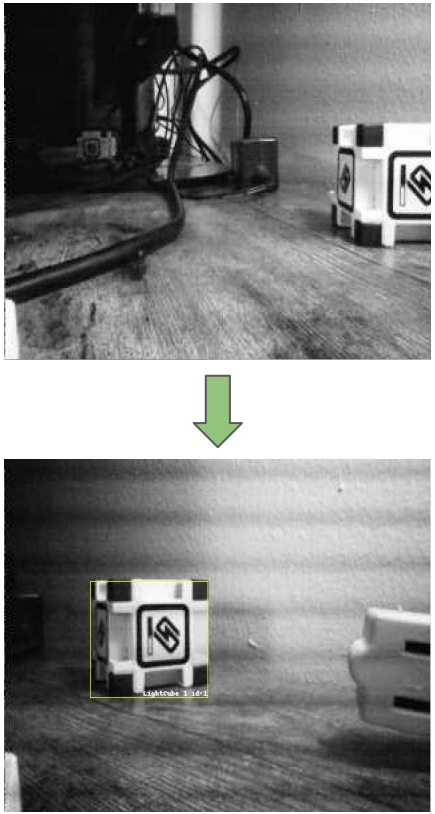
https://www.youtube.com/watch?v=thlll4S_nTo

# When Cozmo sees partial cube left



https://www.youtube.com/watch?v=QhsCd-kJe_E

# When Cozmo sees partial cube right



https://www.youtube.com/watch?v=MF4Bv42DQnc

# Thank You

More details are at

https://jakeshin45.github.io/Projects/CogRob.html