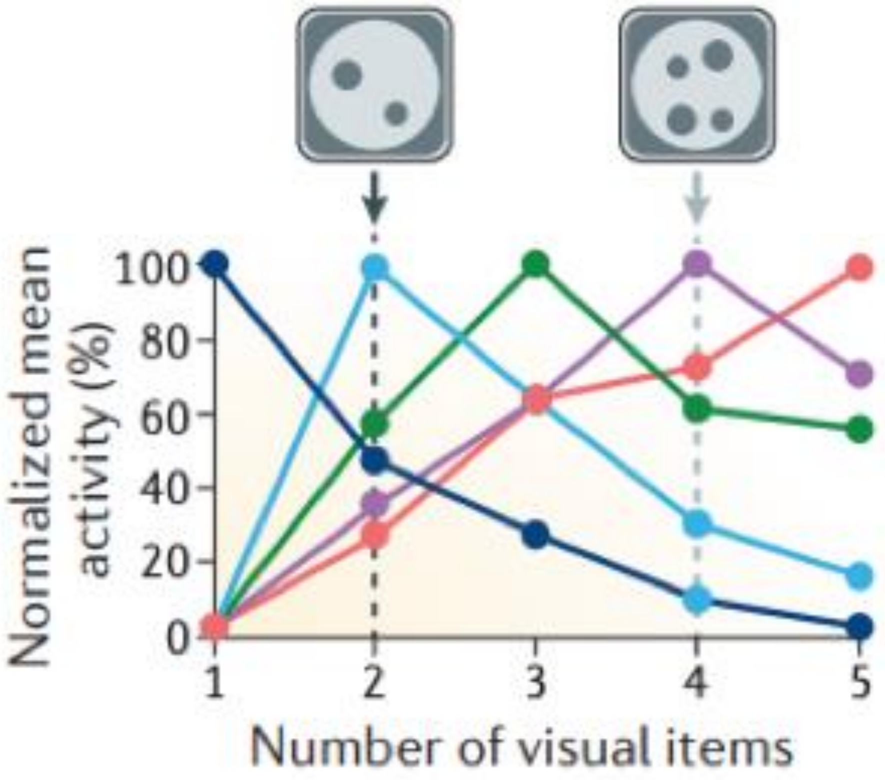
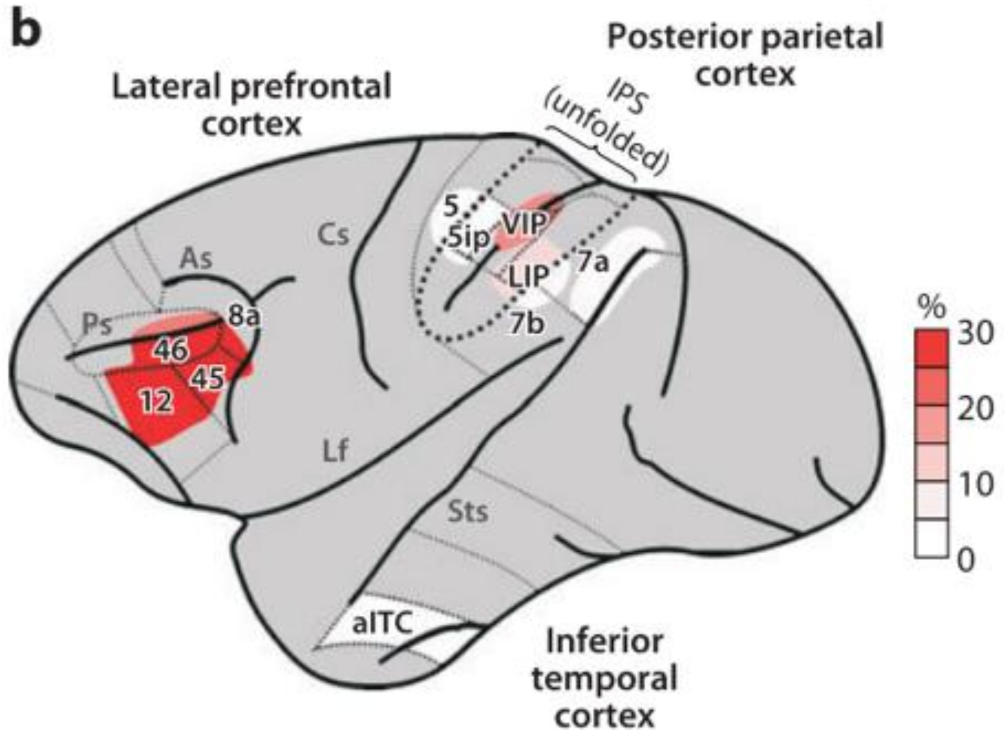
Group 2

Investigating Numerosity in Heirarchical Convolutional Neural Networks

Background



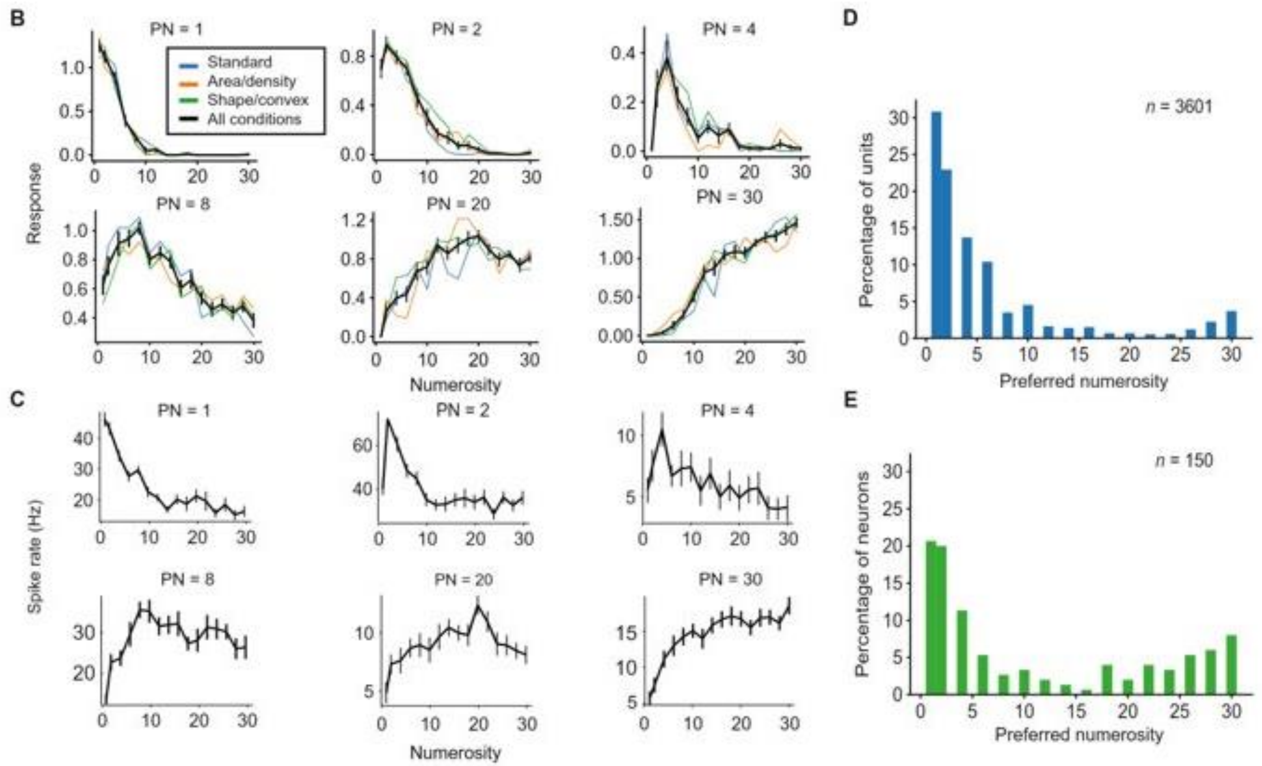


Nieder and Dehaene, 2009.
Annu. Rev. Neurosc

Number detectors spontaneously emerge in a deep neural network designed for visual object recognition

- Nasr et al 2019

Main Results



- 9.6% of units in final layer were numerosity-selective
- These units displayed clear tuning curves like real neurons
- More network units preferred smaller numbers

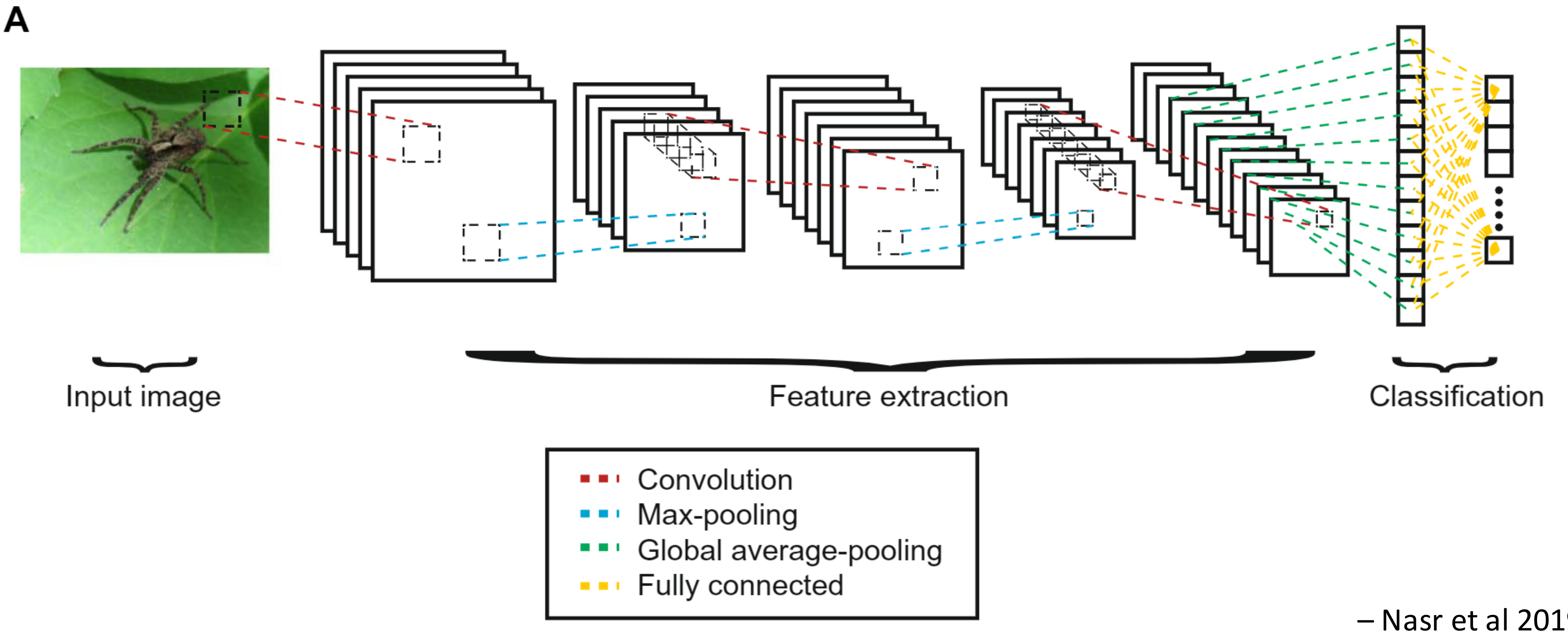
Hypothesis

- Some units will develop a sensitivity to numerosity
- Those units will display tuning curves – prefer a specific number

Process

- Design and train network for image categorization
- Generate and test network on numerosity images
- Analyse responses to see if units have a number preference (ANOVA)
- Determine tuning curves

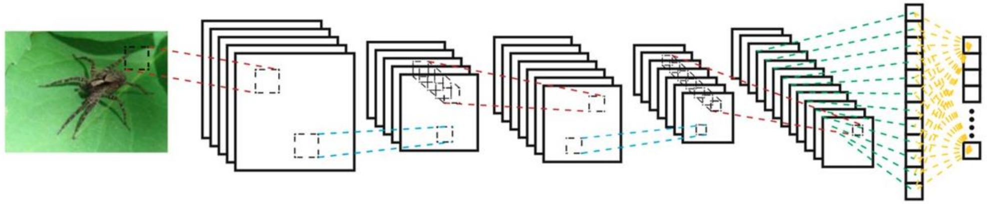
Network Structure



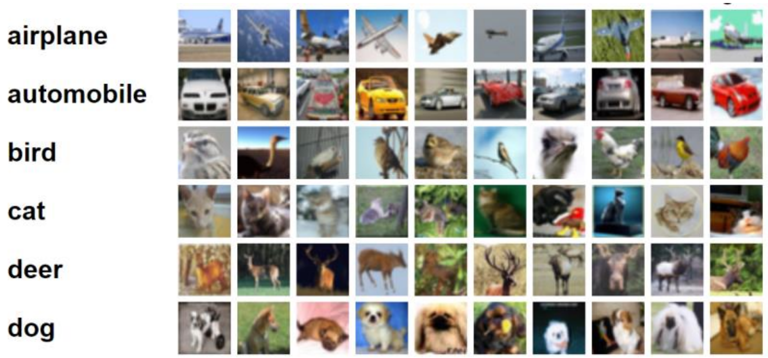
– Nasr et al 2019

Network Creation

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        #define our 3 convolution operations
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=5, kernel_size=3)
        self.conv2 = nn.Conv2d(in_channels=5, out_channels=7, kernel_size=3)
        self.conv3 = nn.Conv2d(in_channels=7, out_channels=12, kernel_size=3)

        #define our max pooling operations
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        #global average pooling
        self.gap = nn.AvgPool2d(kernel_size=2, padding=1)

        #readout layer (10 labels)
        self.fc = nn.Linear(in_features=3*3*12, out_features=10)

    def forward(self, x):
        #convolution 1
        x = F.relu(self.conv1(x))
        # max pooling 1
        x = self.pool(x)

        #convolution 2
        x = F.relu(self.conv2(x))
        #max pooling 2
        x = self.pool(x)

        #convolution 3
        x = F.relu(self.conv3(x))

        #global average pooling
        x = self.gap(x).flatten()
        x = x.view(-1, 3 * 3 * 12) #"fill in the blank" row size, col size 3*3*12

        #readout layer
        x = self.fc(x)
        return x
```



Training on CIFAR-10

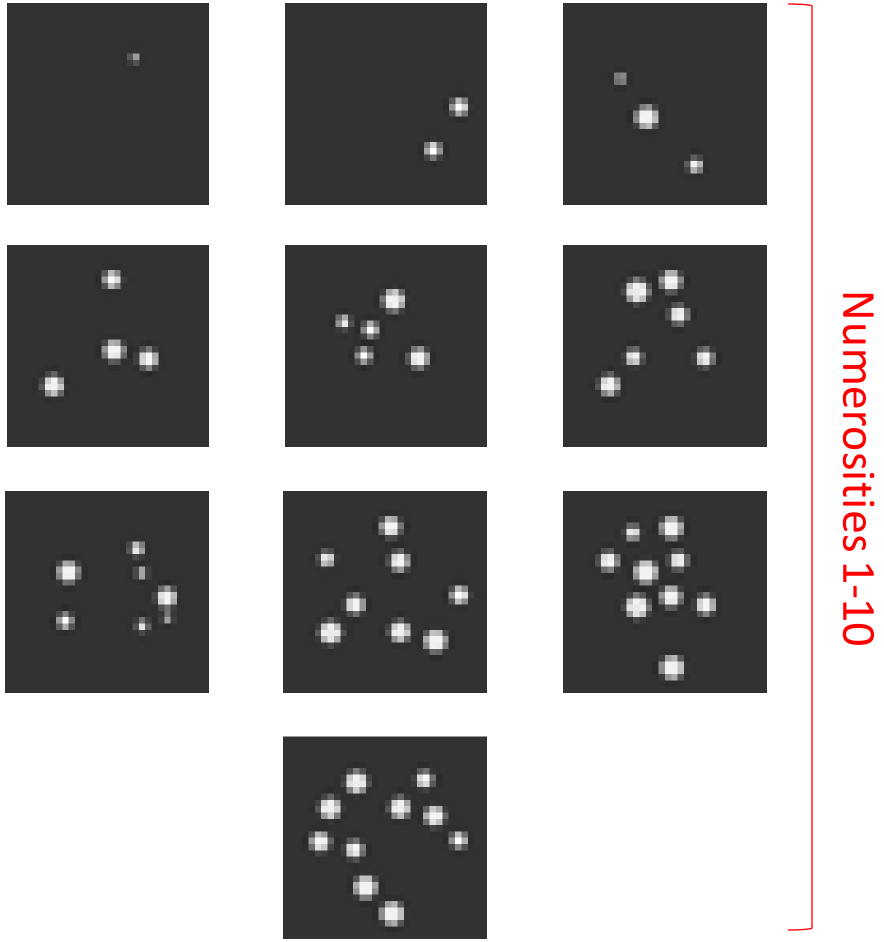


- Train on 50,000 32x32 images (10 output classes)
- Save weights to cifar_net.pth
- Test accuracy on CIFAR-10 test set ~ 47% (not saved or used in our testing)

```python
#train
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data[0].to(device), data[1].to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0
```

"Training a Classifier — PyTorch Tutorials"

Testing on Numerosity Images




```python
def test():
    #test images on modified HCNN architecture
    pred_activations = net2(images)

    print("Testing on " , len(pred_activations), " samples..")

    return pred_activations
```

- Modify architecture by removing readout layer, test on 800 images (~ 80 per numerosity)

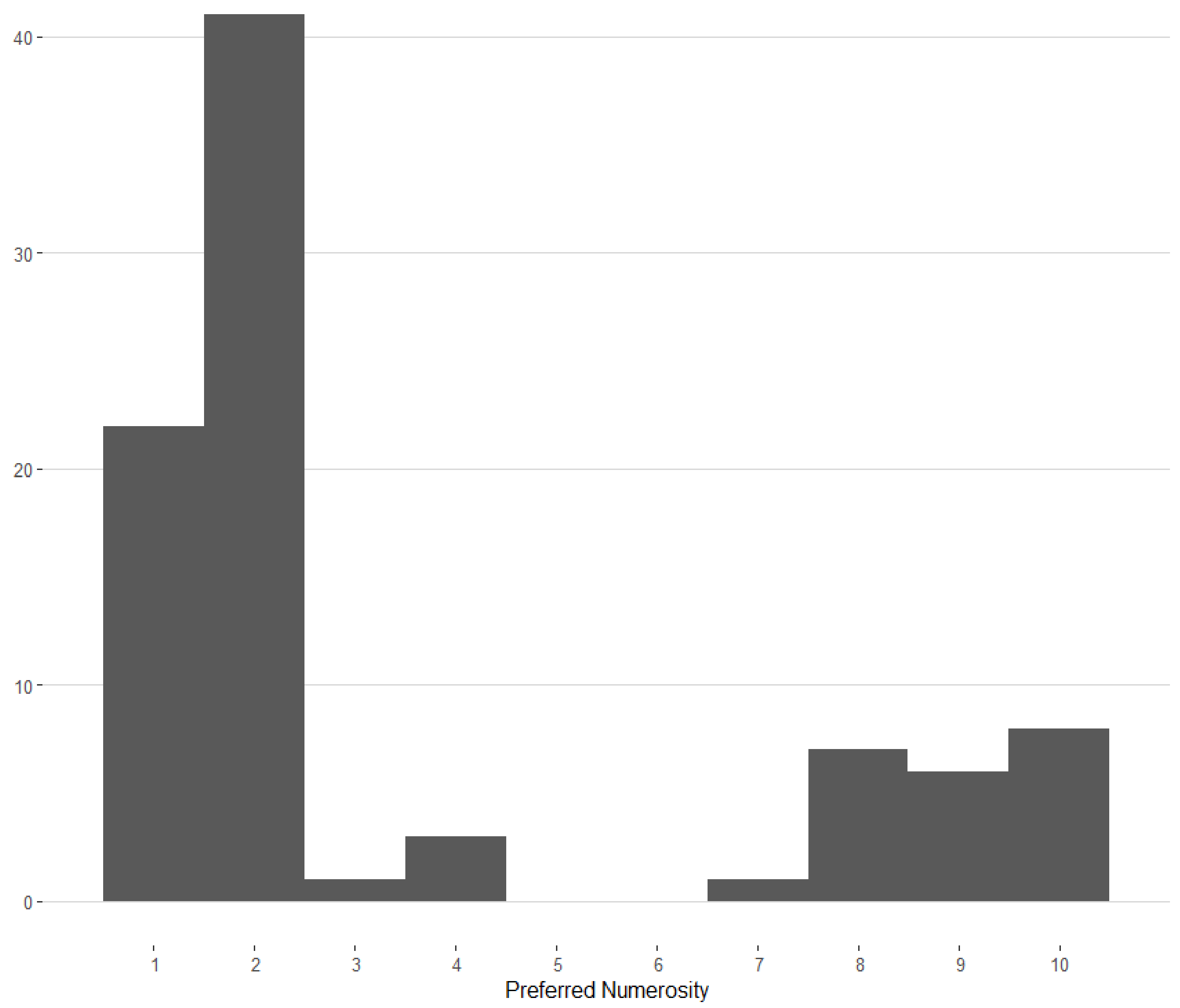
```
[0.44742343 0.87400204 0.44792068 0.8134078  1.5436721  0.8962422
 0.43015313 0.8266827  0.45581347 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.08943099 0.16499045 0.03358285
 0.11300338 0.26664707 0.03293232 0.03955748 0.17989935 0.02609798
 0.71318907 1.3896146  0.6663604  1.5090163  2.4444728  1.3116536
 0.6059805  1.2454991  0.6437001  0.23953682 0.48878807 0.22219077
 0.3166254  0.7000599  0.41260156 0.09097621 0.27457237 0.20924196
 0.         0.0359775  0.0767187  0.05252134 0.07046689 0.15003325
 0.15639073 0.34489945 0.09020855 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.11341855 0.09604472 0.01178682
 0.         0.01273203 0.00603012 0.         0.         0.
 0.         0.         0.         0.         0.07231098 0.
 0.16231327 0.27018127 0.13447088 0.19195852 0.387986   0.2759186
 0.12385792 0.21380201 0.1366865  0.         0.         0.
 0.05109971 0.02357246 0.         0.         0.         0.        ]

Process finished with exit code 0
```

Methods

- One-way ANOVA conducted for each unit
- Tukey HSD tested for pairwise differences
- Representational Similarity Analysis
- Tuning curves were *not* fit
  - Bimodal tuning curves
  - Numerosity is not a continuous variable
  - Not necessary in ascertaining tuning

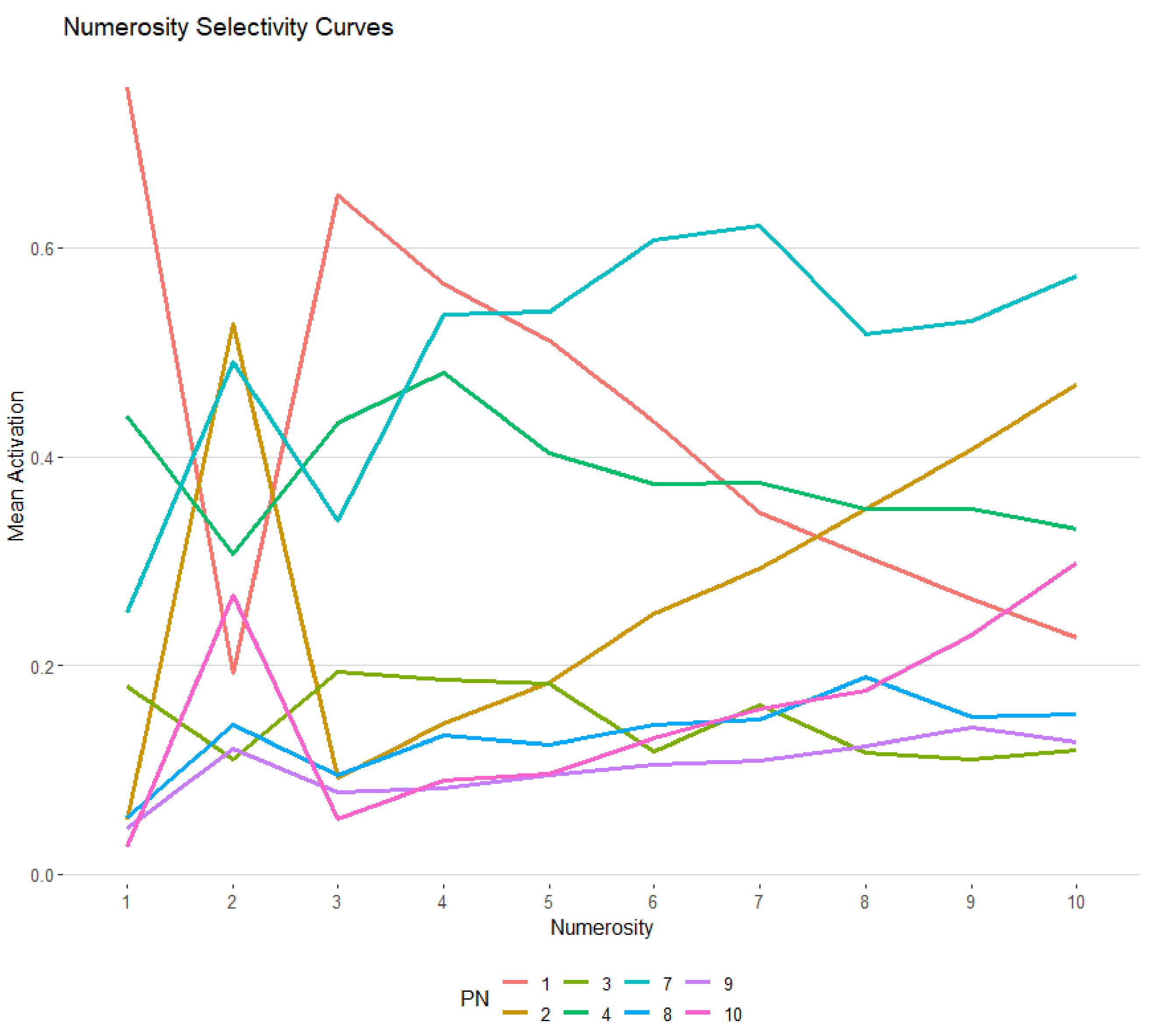
Results

- 82% (89/108) of units were sensitive to numerosity
  - Omnibus test at .01 level
  - Assumptions were met
- Sensitivity tended to emerge for lower numbers
  - No units were sensitive to 5 or 6
  - Over 2/3 are sensitive to 1 or 2
  - Bimodal curve



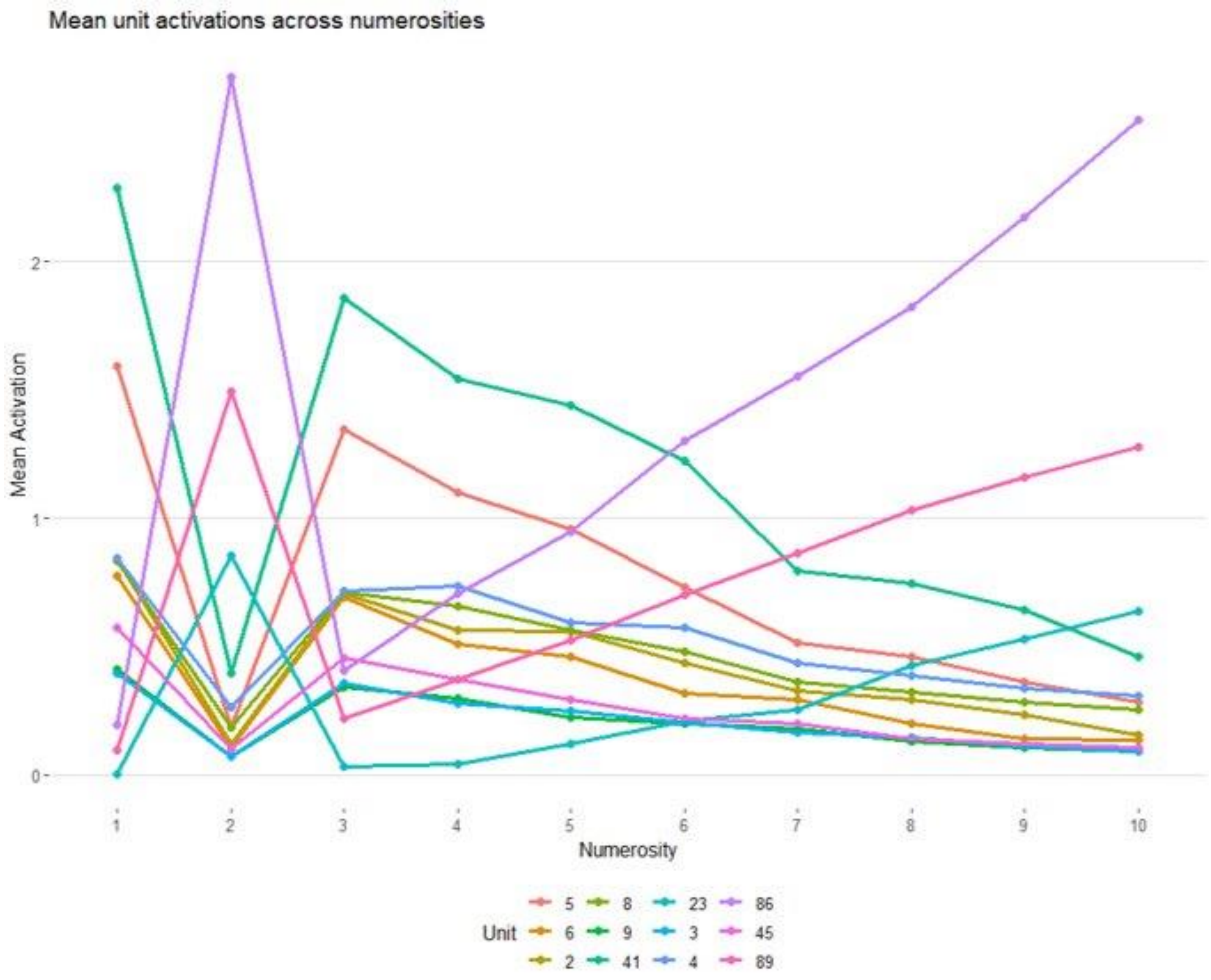
Results

- 82% (89/108) of units were sensitive to numerosity
  - Omnibus test at .01 level
  - Assumptions were met
- Sensitivity tended to emerge for lower numbers
  - No units were sensitive to 5 or 6
  - Over 2/3 are sensitive to 1 or 2
  - Bimodal curve
- Notice similarities between 2 & 10 and 1 & 3

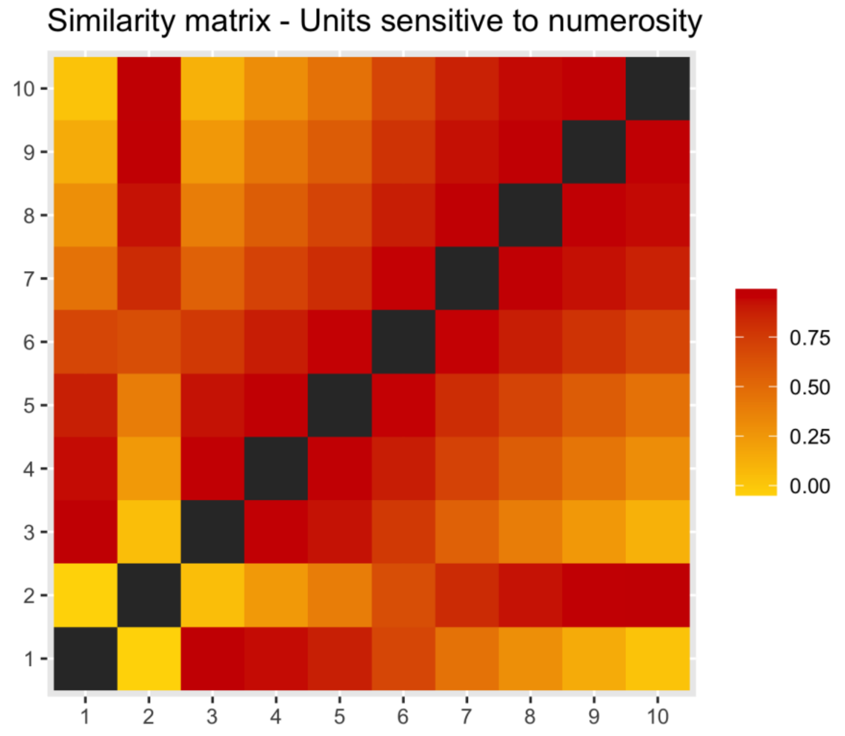


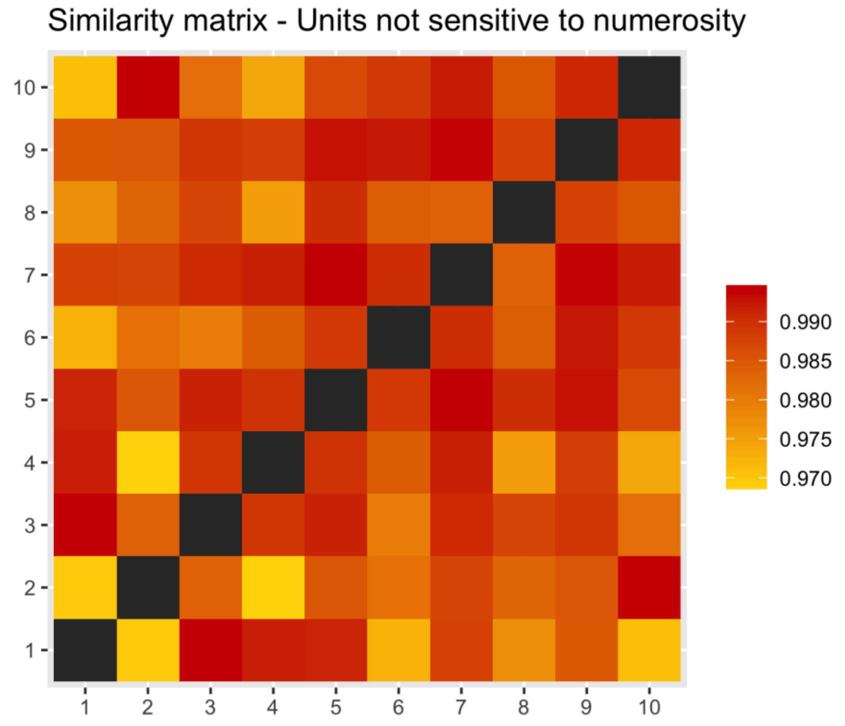
Results

- 82% (89/108) of units were sensitive to numerosity
  - Omnibus test at .01 level
  - Assumptions were met
- Sensitivity tended to emerge for lower numbers
  - No units were sensitive to 5 or 6
  - Over 2/3 are sensitive to 1 or 2
  - Bimodal curve
- Notice similarities between 2 & 10 and 1 & 3
- "Most sensitive" units preferred smaller numerosities



Representational Similarity Analysis





Conclusion & Limitations

- Sensitivity to numerosity emerged
  - Network was trained on fewer images
  - Dot test images were lower resolution
- We had fewer units in final layer
  - May explain how many units were sensitive (82% vs. 10%)
  - May explain similarities between neurons preferring 1-3 and 2-10
- Sensitivity did not emerge for *each* numerosity
- Network was not exposed to varying numerosities in training

References

[1] K. Nasr, P. Viswanathan, A. Nieder, Number detectors spontaneously emerge in a deepneural network designed for visual object recognition.Sci. Adv.5, eaav7903 (2019).SCIENCE ADVANCES|RESEARCH ARTICLENasret al.,Sci. Adv.2019;5:eaav7903 8 May 201910 of 10 on March 22, 2021http://advances.sciencemag.org/Downloaded from

[2] Training a Classifier — PyTorch Tutorials 1.8.1+cu102 Documentation. https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html. Accessed 5 Apr. 2021.