# DATA 442: Neural Networks & Deep Learning

Dan Runfola – danr@wm.edu

icss.wm.edu/data442/

# Some Reminders

- Piazza
  - I - and the TAs - are checking Piazza regularly.
- Lab 1
  - Launches at midnight tonight! See Piazza for the deadline.
  - We'll be covering the content for lab 1 over the next couple of lectures.

# Image Classification

We must pre-specify a set of labels that we want to choose between. For example:

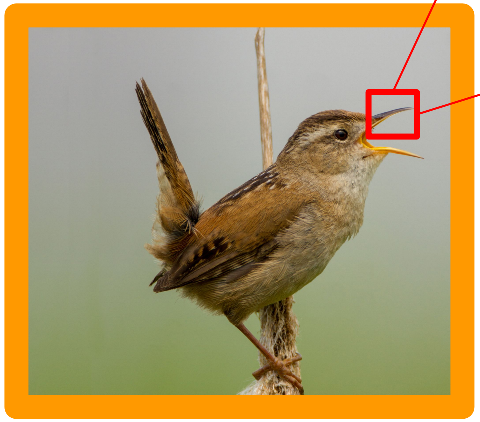
Car

Building

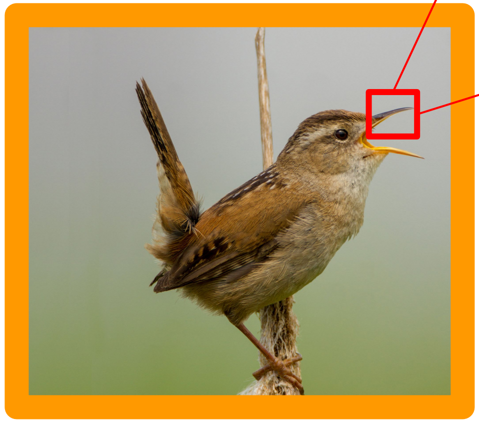
Person

**Bird**

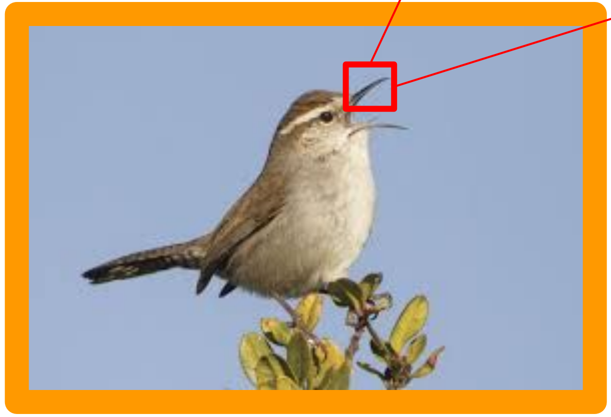
| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 34 | 40 | 34 | 3 | 8 | 30 | 50 | 26 | 16 | 28 | 41 | 6 | 23 | 2 | 24 | 0 |
| 14 | 37 | 19 | 25 | 6 | 9 | 14 | 17 | 4 | 46 | 20 | 7 | 38 | 6 | 29 | 28 |
| 4 | 39 | 0 | 29 | 15 | 6 | 50 | 2 | 21 | 10 | 8 | 45 | 150 | 145 | 106 | 46 |
| 42 | 10 | 15 | 19 | 24 | 18 | 111 | 123 | 118 | 104 | 119 | 122 | 117 | 140 | 138 | 28 |
| 21 | 35 | 19 | 30 | 14 | 143 | 146 | 147 | 142 | 103 | 109 | 127 | 108 | 148 | 20 | 23 |
| 30 | 105 | 147 | 102 | 126 | 118 | 108 | 101 | 140 | 131 | 124 | 136 | 47 | 27 | 26 | 38 |
| 135 | 133 | 137 | 108 | 140 | 144 | 135 | 120 | 118 | 137 | 125 | 43 | 8 | 31 | 45 | 10 |
| 106 | 142 | 108 | 138 | 137 | 111 | 38 | 36 | 32 | 1 | 19 | 44 | 34 | 4 | 38 | 49 |
| 122 | 142 | 127 | 131 | 143 | 8 | 47 | 4 | 0 | 31 | 39 | 18 | 46 | 1 | 50 | 25 |
| 149 | 137 | 122 | 36 | 50 | 19 | 24 | 45 | 16 | 30 | 2 | 47 | 2 | 35 | 29 | 50 |
| 147 | 115 | 3 | 29 | 10 | 2 | 13 | 1 | 48 | 3 | 45 | 28 | 39 | 14 | 14 | 20 |



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 34 | 40 | 34 | 3 | 8 | 30 | 50 | 26 | 16 | 28 | 41 | 6 | 23 | 2 | 24 | 0 |
| 14 | 37 | 19 | 25 | 6 | 9 | 14 | 17 | 4 | 46 | 20 | 7 | 38 | 6 | 29 | 28 |
| 4 | 39 | 0 | 29 | 15 | 6 | 50 | 2 | 21 | 10 | 8 | 45 | 150 | 145 | 106 | 46 |
| 42 | 10 | 15 | 19 | 24 | 18 | 111 | 123 | 118 | 104 | 119 | 122 | 117 | 140 | 138 | 28 |
| 21 | 35 | 19 | 30 | 14 | 143 | 146 | 147 | 142 | 103 | 109 | 127 | 108 | 148 | 20 | 23 |
| 30 | 105 | 147 | 102 | 126 | 118 | 108 | 101 | 140 | 131 | 124 | 136 | 47 | 27 | 26 | 38 |
| 135 | 133 | 137 | 108 | 140 | 144 | 135 | 120 | 118 | 137 | 125 | 43 | 8 | 31 | 45 | 10 |
| 106 | 142 | 108 | 138 | 137 | 111 | 38 | 36 | 32 | 1 | 19 | 44 | 34 | 4 | 38 | 49 |
| 122 | 142 | 127 | 131 | 143 | 8 | 47 | 4 | 0 | 31 | 39 | 18 | 46 | 1 | 50 | 25 |
| 149 | 137 | 122 | 36 | 50 | 19 | 24 | 45 | 16 | 30 | 2 | 47 | 2 | 35 | 29 | 50 |
| 147 | 115 | 3 | 29 | 10 | 2 | 13 | 1 | 48 | 3 | 45 | 28 | 39 | 14 | 14 | 20 |



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 34 | 40 | 34 | 3 | 8 | 30 | 50 | 26 | 16 | 28 | 41 | 6 | 23 | 2 | 24 | 0 |
| 14 | 37 | 19 | 25 | 6 | 9 | 14 | 17 | 4 | 46 | 20 | 7 | 38 | 6 | 29 | 28 |
| 28 | 6 | 31 | 39 | 0 | 23 | 36 | 34 | 21 | 10 | 8 | 45 | 6 | 29 | 45 | 46 |
| 71 | 6 | 9 | 44 | 41 | 23 | 36 | 61 | 4 | 104 | 119 | 122 | 117 | 140 | 138 | 28 |
| 8 | 60 | 45 | 11 | 12 | 165 | 122 | 115 | 142 | 103 | 109 | 127 | 108 | 148 | 100 | 23 |
| 2 | 94 | 156 | 88 | 174 | 160 | 62 | 59 | 140 | 131 | 124 | 136 | 127 | 150 | 145 | 38 |
| 183 | 94 | 160 | 101 | 108 | 163 | 135 | 119 | 118 | 137 | 125 | 43 | 8 | 31 | 45 | 10 |
| 128 | 179 | 74 | 122 | 89 | 140 | 59 | 22 | 32 | 1 | 19 | 44 | 34 | 4 | 38 | 49 |
| 165 | 100 | 106 | 172 | 110 | 41 | 58 | 11 | 0 | 31 | 39 | 18 | 46 | 1 | 50 | 25 |
| 154 | 175 | 158 | 76 | 53 | 58 | 61 | 69 | 16 | 30 | 2 | 47 | 2 | 35 | 29 | 50 |
| 153 | 138 | 43 | 12 | 19 | 40 | 62 | 26 | 48 | 3 | 45 | 28 | 39 | 14 | 14 | 20 |



# Viewpoint

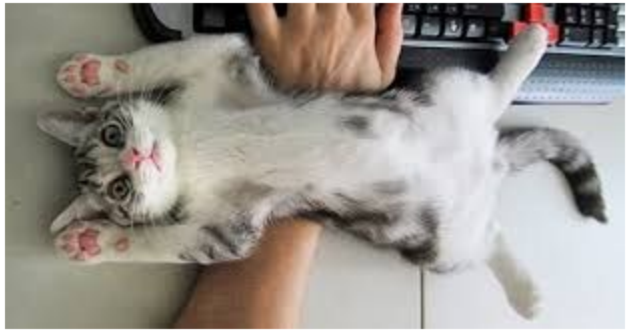
# Lighting

# Background

# Background

# Deformation

# Deformation









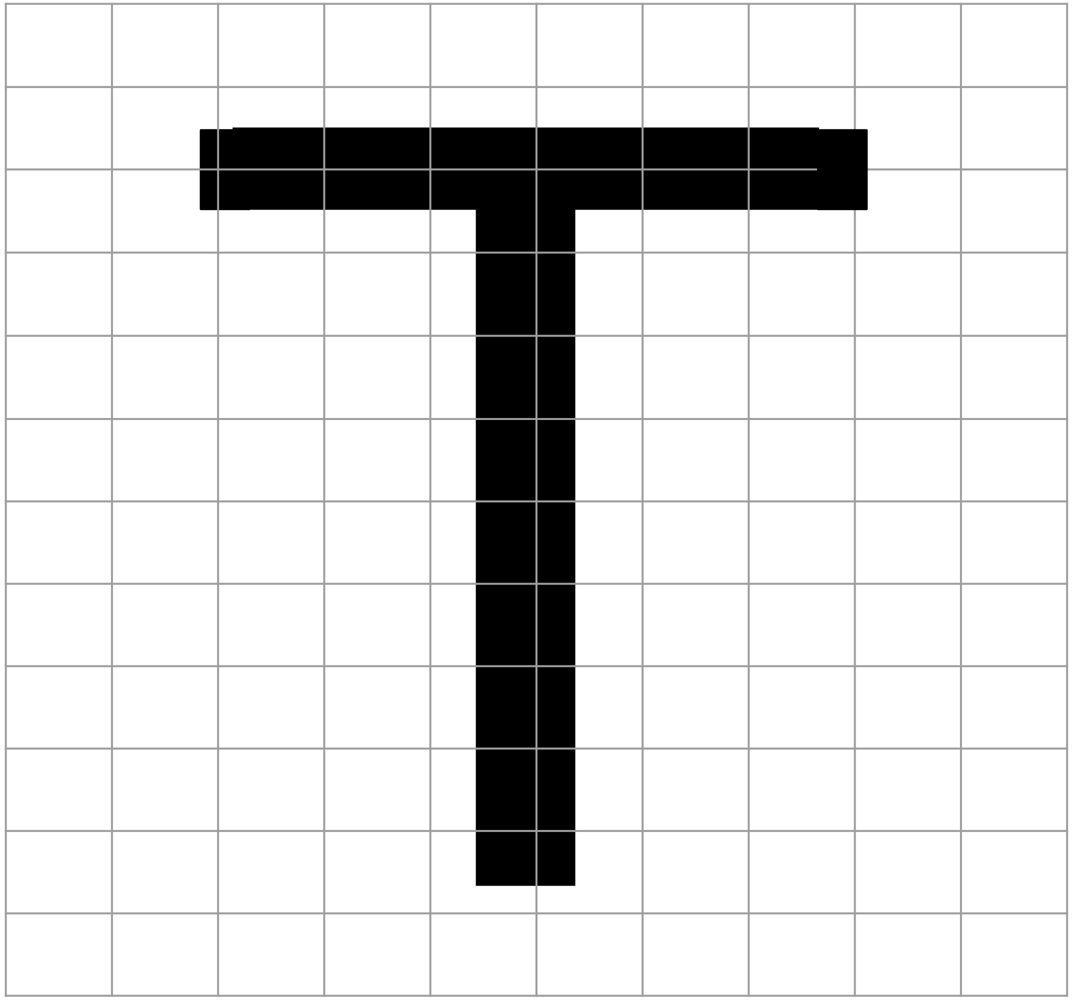
# Occlusion






```
letterT = [0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
           0,  1,  1,  1,  1,  1,  1,  1,  1,  0,
           0,  1,  1,  1,  1,  1,  1,  1,  1,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  0,  0,  0,  0,  0,  0]
```

**12 Row * 10 Column
Resolution Data**

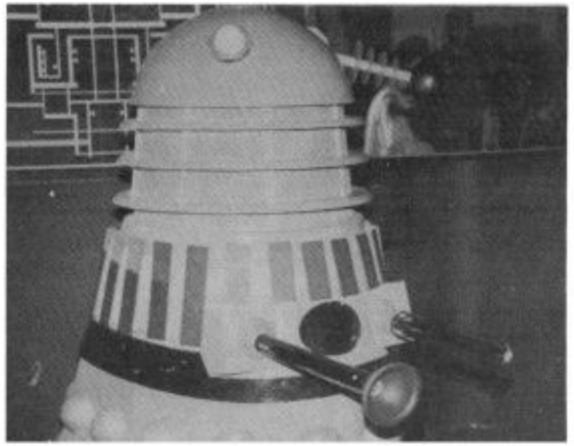


```python
def imageClassifier(letter):

    # Pick a label based on the data input
    predictedLabel = "Some Letter"

    return(predictedLabel)

print(imageClassifier(letterT))
```

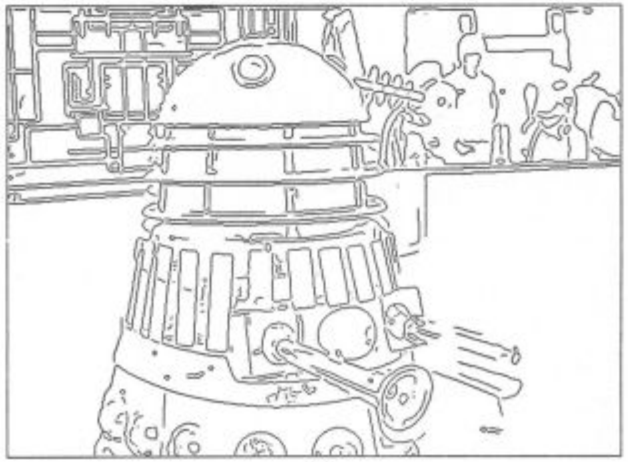
32 “1” Values

```
letterT = [0  0  0  0  0  0  0  0  0  0
           0  1  1  1  1  1  1  1  1  0
           0  1  1  1  1  1  1  1  1  0
           0  0  0  0  1  1  0  0  0  0
           0  0  0  0  1  1  0  0  0  0
           0  0  0  0  1  1  0  0  0  0
           0  0  0  0  1  1  0  0  0  0
           0  0  0  0  1  1  0  0  0  0
           0  0  0  0  1  1  0  0  0  0
           0  0  0  0  1  1  0  0  0  0
           0  0  0  0  1  1  0  0  0  0
           0  0  0  0  0  0  0  0  0  0 ]
```

```
def imageClassifier(letter):
    if(sum(letter) == 32):
        predictedLabel = "T"

    # Pick a label based on the data input

    return(predictedLabel)


print(imageClassifier(letterT))
```



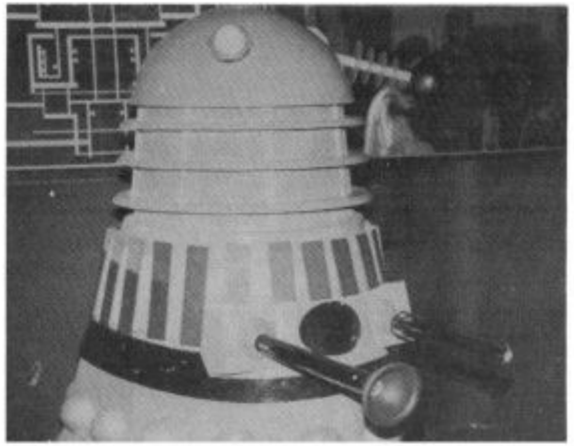
(a)



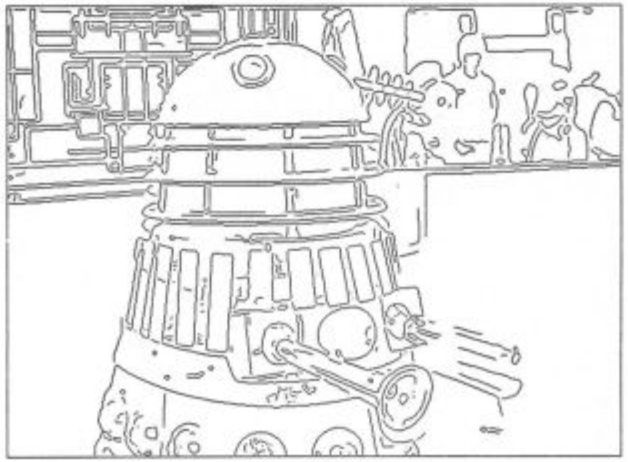
(b)

```python
def imageClassifier(letter):
    #Detect number of line intersections
    #(For the case of "T", this would be 1 intersection)
    intersections = 1

    if(intersections == 1):
        predictedLabel = "T"
    # Pick a label based on the data input

    return(predictedLabel)

print(imageClassifier(letterT))
```

J. Canny, "A Computational Approach to Edge Detection," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.



(a)



(b)

J. Canny, "A Computational Approach to Edge Detection," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.

# Machine Learning & AI


1) Curate / Label a Huge Dataset of Images

```
def train(observedImages, humanLabels):
    #Teach the model all of the exceptions and rules
    return imageClassifier
```

2) Train a Classifier

```
def predict(imageClassifier, myNewImage):
    #Use the classifier
    return predictedLabel
```

3) Test How Well It Does on Data It's Never Seen

# Nearest Neighbor & Imagery

```python
def train(observedImages, humanLabels):
    #Teach the model all of the exceptions and rules
    return imageClassifier
```

1) Saves All Observations into Memory

```python
def predict(imageClassifier, myNewImage):
    #Use the classifier
    return predictedLabel
```

2) Compares and Contrasts Input to all Observations to Select Most Similar

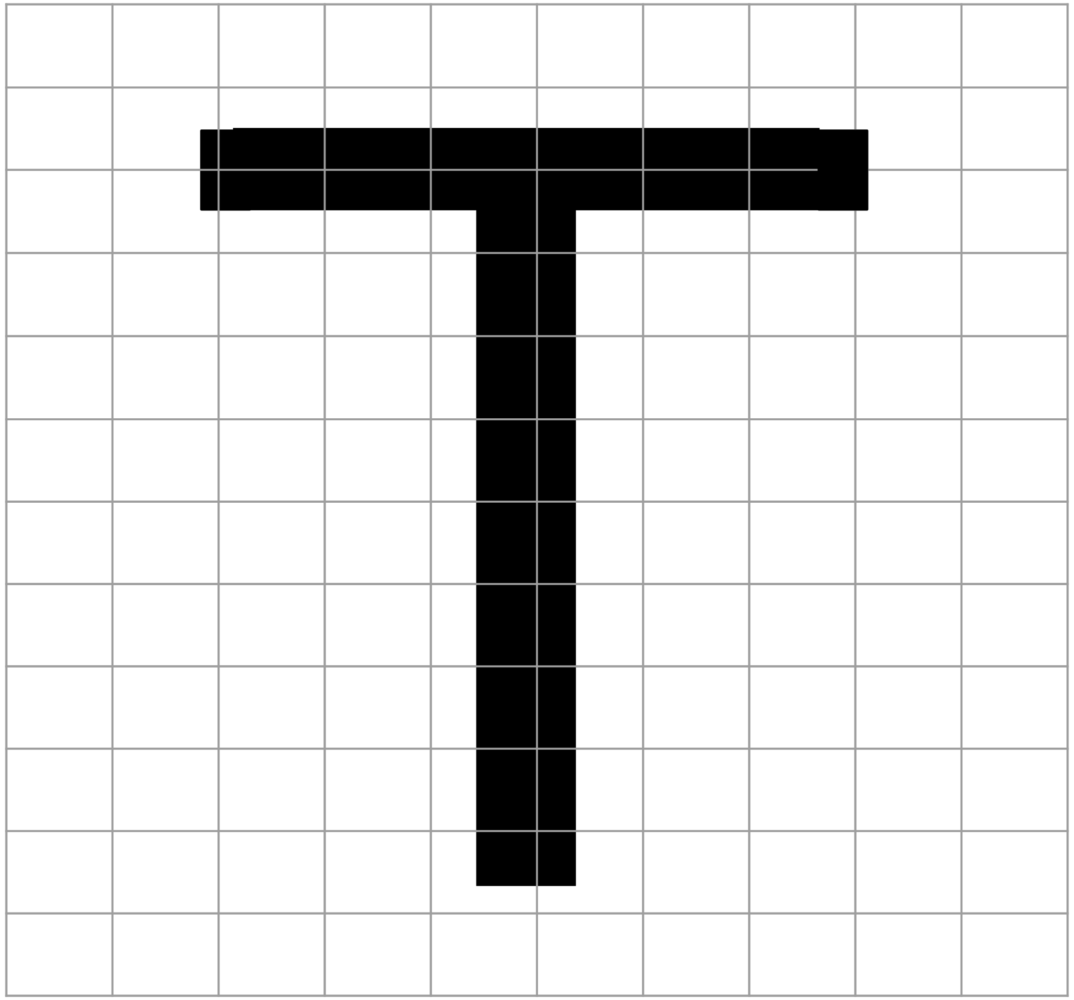
# Example Training Data

```
letterT = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
           0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

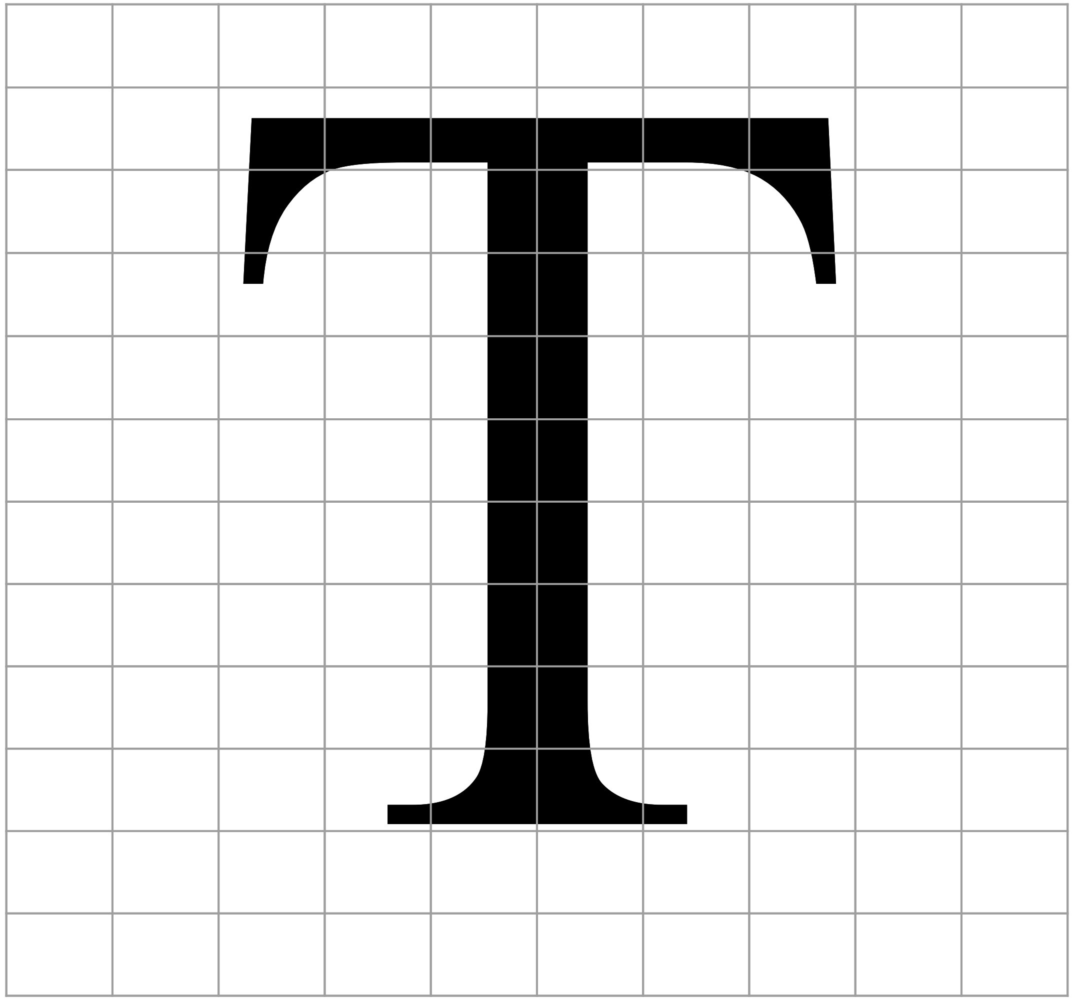
```
letterl = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
letterI = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
           0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
           0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
           0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

T from Training Data



T to Recognize



## T from Training Data

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | | | | | | |

## T to Recognize

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | | 1 | 1 | 1 | 1 | 1 | 1 | | |
| | | 1 | 1 | 1 | 1 | 1 | 1 | | |
| | | 1 | | 1 | 1 | | 1 | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | 1 | 1 | 1 | 1 | | | |
| | | | | | | | | | |
| | | | | | | | | | |

# Nearest Neighbor - L1 Distance

$$L_1(I_1, I_2) = \sum_p |I_{1,p} - I_{2,p}|$$

$L_1$ = L1 Distance

$I_i$ = Image $i$

$p$ = Index for a given pixel in image

$I_{i,p}$ = Value for a given pixel $p$ in image $I$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | | | | | | |

**T from Training Data**

$-$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | | 1 | 1 | 1 | 1 | 1 | 1 | | |
| | | 1 | 1 | 1 | 1 | 1 | 1 | | |
| | | 1 | | 1 | 1 | | 1 | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | | 1 | 1 | | | | |
| | | | 1 | 1 | 1 | 1 | | | |
| | | | | | | | | | |
| | | | | | | | | | |

**T we want to Recognize**

$=$

**Sum of Absolute Difference: 10**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 1 | | | | | | | 1 | |
| | 1 | | | | | | | 1 | |
| | | 1 | | | | | 1 | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | 1 | | | 1 | | | |
| | | | | 1 | 1 | | | | |
| | | | | | | | | | |

$$L_1(I_1, I_2) = \sum_p |I_{1,p} - I_{2,p}|$$

$$L_1 = \text{L1 Distance}$$

$$I_i = \text{Image } i$$

$$p = \text{Index for a given pixel in image}$$

$$I_{i,p} = \text{Value for a given pixel } p \text{ in image } I$$

```python
import numpy as np
def L1Norm(imageA, imageB):
    print("Total number of black pixels in Image A: " +
            str(np.sum(imageA)))

    print("Total number of black pixels in Image B: " +
            str(np.sum(imageB)))

    pixelWiseDiff = str(np.sum(np.abs(np.asarray(imageA) - np.asarray(imageB))))
    print("Absolute pixelwise difference: " +
            pixelWiseDiff)

    return(pixelWiseDiff)

L1Norm(letterT, testedT)
```

```
Total number of black pixels in Image A: 32
Total number of black pixels in Image B: 30
Absolute pixelwise difference: 10
```

## Training Data

```
letterT = [0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
           0,  1,  1,  1,  1,  1,  1,  1,  1,  0,
           0,  1,  1,  1,  1,  1,  1,  1,  1,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  0,  0,  0,  0,  0,  0]
```

```
letterl = [0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  0,  0,  0,  0,  0,  0]
```

```
letterI = [0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
           0,  1,  1,  1,  1,  1,  1,  1,  1,  0,
           0,  1,  1,  1,  1,  1,  1,  1,  1,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  1,  1,  1,  1,  1,  1,  1,  1,  0,
           0,  1,  1,  1,  1,  1,  1,  1,  1,  0,
           0,  0,  0,  0,  0,  0,  0,  0,  0,  0]
```

```
testedT = [0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
           0,  0,  1,  1,  1,  1,  1,  1,  0,  0,
           0,  0,  1,  1,  1,  1,  1,  1,  0,  0,
           0,  0,  1,  0,  1,  1,  0,  1,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  1,  1,  1,  1,  0,  0,  0,
           0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
           0,  0,  0,  0,  0,  0,  0,  0,  0,  0]
```

```python
training = {}
training["T"] = letterT
training["l"] = letterl
training["I"] = letterI

predictLetter = testedT
estimates = {}
for l in training:
    distance = L1Norm(training[l], predictLetter)
    estimates[l] = distance

print(estimates)
```

```
{'T': '10', 'l': '14', 'I': '18'}
```

# Interlude - Numpy

```
letterT = [0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
           0,  1,  1,  1,  1,  1,  1,  1,  1,  0,
           0,  1,  1,  1,  1,  1,  1,  1,  1,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  0,  0,  0,  0,  0,  0]
```

```
letterT = np.asarray(letterT)
print(letterT)
```

```
[0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 0
 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0]
```

# A toy nearest neighbor classifier

```python
class NearestNeighborSinglePrediction:
    def __init__(self):
        pass

    def train(self, X, y):
        #For nearest neighbor, we just copy the data for later use.
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        l1Distances = np.sum(np.abs(self.Xtr - X[0]), axis=1)
        minimumDistance = np.argmin(l1Distances)
        Ypred = self.ytr[minimumDistance]

        return Ypred
```

# A toy nearest neighbor classifier

```
class NearestNeighborSinglePrediction:
    def __init__(self):
        pass

    def train(self, X, y):
        #For nearest neighbor, we just copy the data for later use.
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        l1Distances = np.sum(np.abs(self.Xtr - X[0]), axis=1)
        minimumDistance = np.argmin(l1Distances)
        Ypred = self.ytr[minimumDistance]

        return Ypred
```

# A toy nearest neighbor classifier

```
class NearestNeighborSinglePrediction:
    def __init__(self):
        pass

    def train(self, X, y):
        #For nearest neighbor, we just copy the data for later use.
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        l1Distances = np.sum(np.abs(self.Xtr - X[0]), axis=1)
        minimumDistance = np.argmin(l1Distances)
        Ypred = self.ytr[minimumDistance]

        return Ypred
```

```
trainingX = [letterT, letterI, letterl]
trainingy = np.array(["T", "I", "l"])

nn = NearestNeighborSinglePrediction()
nn.train(X=trainingX, y=trainingy)
estimates = nn.predict(X=[testLetter])

print(estimates)
```

T

# A toy nearest neighbor classifier

```
class NearestNeighborSinglePrediction:
    def __init__(self):
        pass

    def train(self, X, y):
        #For nearest neighbor, we just copy the data for later use.
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        l1Distances = np.sum(np.abs(self.Xtr - X[0]), axis=1)
        minimumDistance = np.argmin(l1Distances)
        Ypred = self.ytr[minimumDistance]

        return Ypred
```

```
for l in training:
    distance = L1Norm(training[l], predictLetter)
    estimates[l] = distance
```

# A toy nearest neighbor classifier

*l* =

```
testedL = [0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  1,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  1,  1,  0,  0,
           0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
           0,  0,  0,  0,  0,  0,  0,  0,  0,  0]
```

T =

```
testedT = [0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
           0,  0,  1,  1,  1,  1,  1,  1,  0,  0,
           0,  0,  1,  1,  1,  1,  1,  1,  0,  0,
           0,  0,  1,  0,  1,  1,  0,  1,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  0,  1,  1,  0,  0,  0,  0,
           0,  0,  0,  1,  1,  1,  1,  0,  0,  0,
           0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
           0,  0,  0,  0,  0,  0,  0,  0,  0,  0]
```

# A toy nearest neighbor classifier

```
class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        #For nearest neighbor, we just copy the data for later use.
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        #We'll be doing our test for every input X this time,
        #just in case we want to test multiple
        #cases (As we'll be doing later!)

        #Create an empty list to hold our results
        #Note the dtype tells Numpy if the output (y) estimates
        #should be a float, integer, or string based on the training y.
        Ypred = np.zeros(len(X), dtype=np.dtype(self.ytr.dtype))

        for i in range(0, len(X)):
            l1Distances = np.sum(np.abs(self.Xtr - X[i]), axis=1)
            minimumDistance = np.argmin(l1Distances)
            Ypred[i] = self.ytr[minimumDistance]

        return Ypred
```

```
nn = NearestNeighbor()
nn.train(X=trainingX, y=trainingy)
estimates = nn.predict(X=[testLetter, testLetter2])

print(estimates)
```

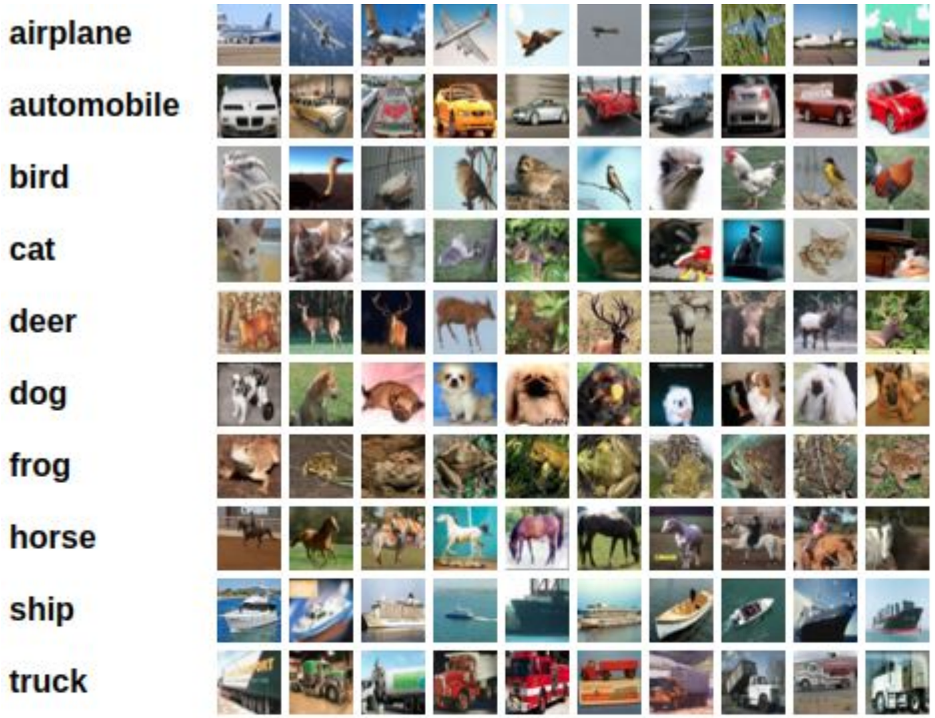
[‘T’, ‘1’]

# A toy nearest neighbor classifier

```python
class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        #For nearest neighbor, we just copy the data for later use.
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        #We'll be doing our test for every input X this time,
        #just in case we want to test multiple
        #cases (As we'll be doing later!)

        #Create an empty list to hold our results
        #Note the dtype tells Numpy if the output (y) estimates
        #should be a float, integer, or string based on the training y.
        Ypred = np.zeros(len(X), dtype=np.dtype(self.ytr.dtype))

        for i in range(0, len(X)):
            l1Distances = np.sum(np.abs(self.Xtr - X[i]), axis=1)
            minimumDistance = np.argmin(l1Distances)
            Ypred[i] = self.ytr[minimumDistance]

        return Ypred
```

Slow!  We have to compare every single case in our training data to every input.

# Example: Nearest Neighbor & CIFAR 10




**60,000 Images**
**50,000 Training**
**10,000 Testing**

**32 x 32 Pixels**

**10 Classes (shown to left)**



https://www.cs.toronto.edu/~kriz/cifar.html
Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

# K Nearest Neighbors

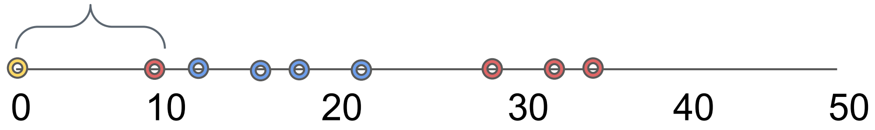
In this example, we have multiple examples of letters we're using for training.  Every red dot is an "A", and every blue dot is a "T".  The yellow dot is representative of a hand-written T, that we're trying to identify the letter of.

Red = Examples of Letter A
Blue = Examples of Letter T


**Pixelwise Difference Between Test Image and Observed Image**

# K Nearest Neighbors

Most similar letter with a L1 Distance of ~10.

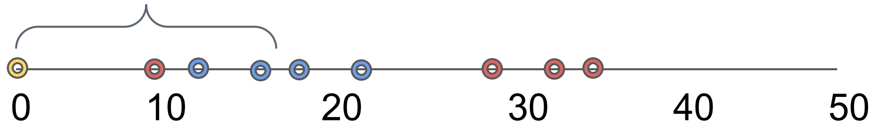



Red = Examples of Letter A
Blue = Examples of Letter T

**Pixelwise Difference Between Test Image and Observed Image**

# K=3 Nearest Neighbors

Red = Examples of Letter A
Blue = Examples of Letter T



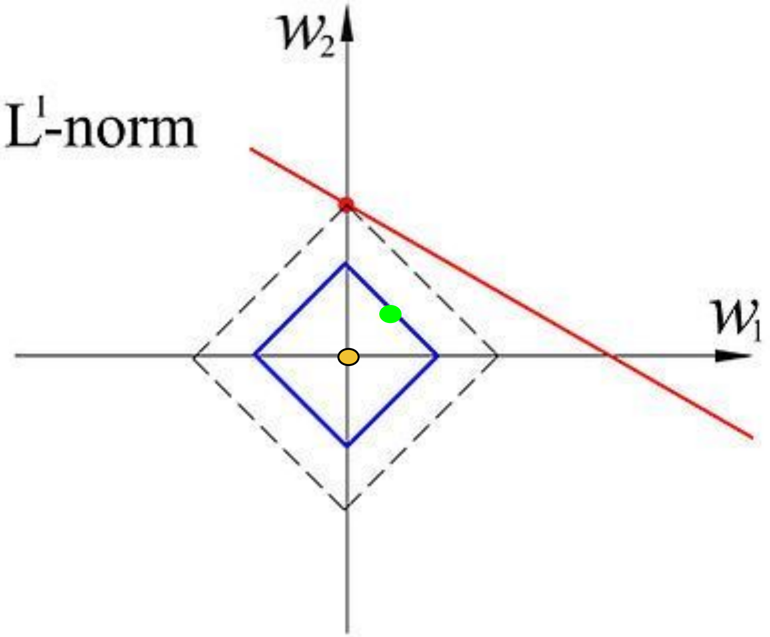

**Pixelwise Difference Between Test Image and Observed Image**

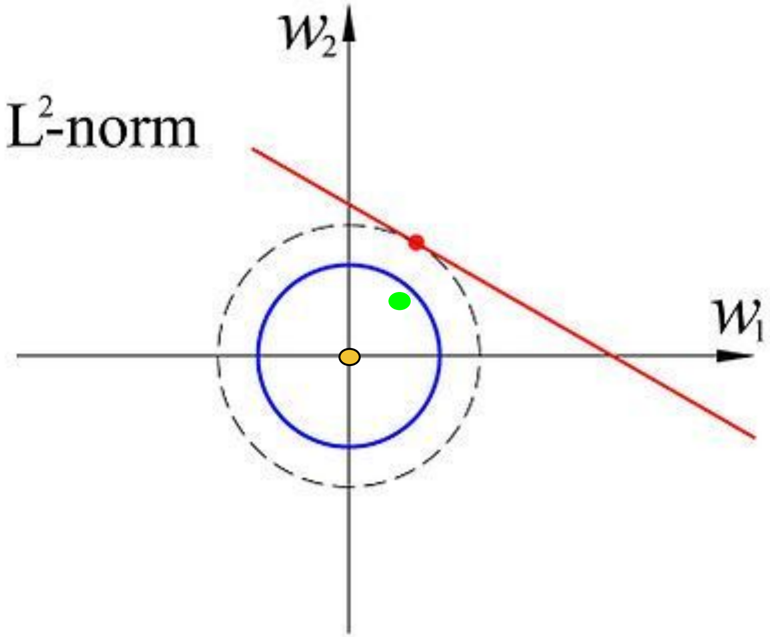
| | N=1 | N=2 | N=3 | N=4 | N=5 | N=6 | N=7 | N=8 | N=9 | N=10 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Airplane** | 0 | 0 | 0 | 0 | 20% | 16% | 14% | 13% | 22% | 20% |
| **Car** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 13% | 11% | 10% |
| **Bird** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Cat** | 100% | 50% | 66% | 50% | 40% | 50% | 43% | 37% | 33% | 30% |
| **Deer** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Dog** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Frog** | 0 | 50% | 33% | 50% | 40% | 34% | 43% | 37% | 33% | 30% |
| **Horse** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Ship** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10% |
| **Truck** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# KNN - Distance Metric pt 2.

$$L_1(I_1, I_2) = \sum_p |I_{1,p} - I_{2,p}|$$

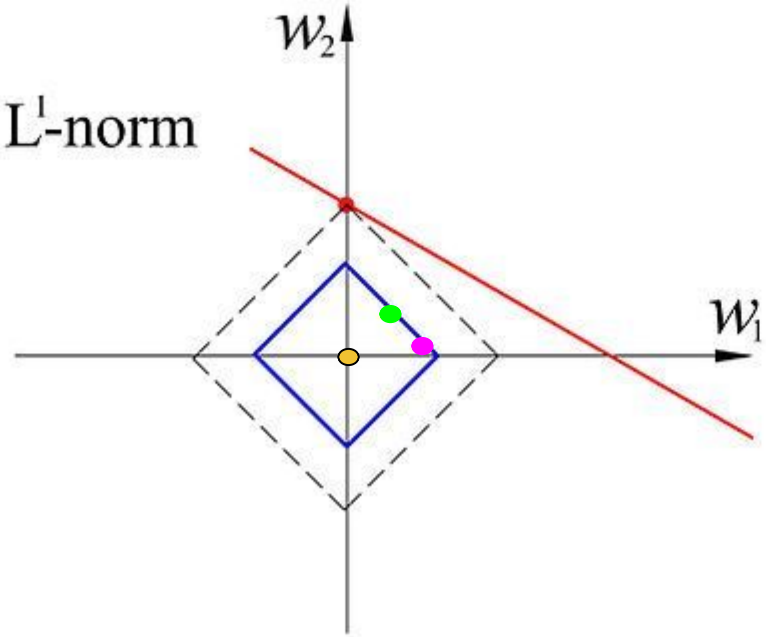
$$L2(I_1, I_2) = \sqrt{\sum_p (I_{1,p} - I_{2,p})^2}$$

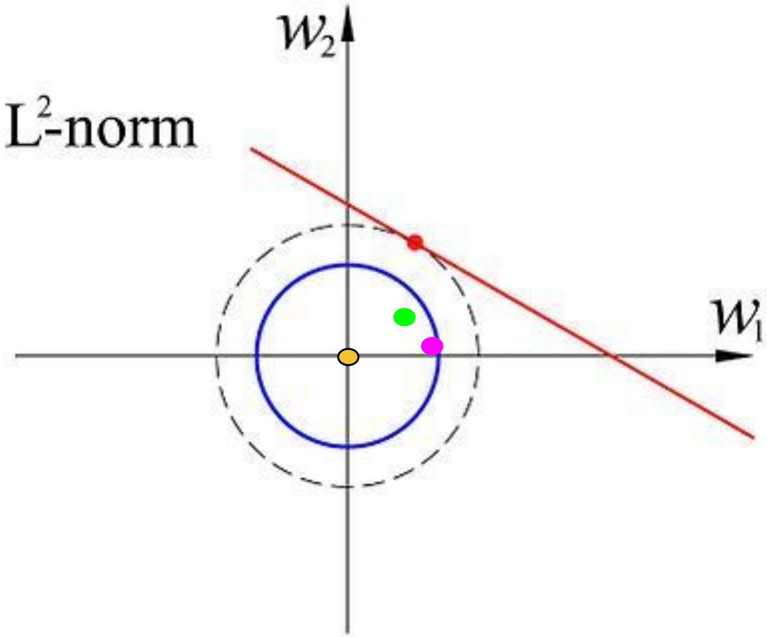


# KNN - Distance Metric pt 2.

$$L_1(I_1, I_2) = \sum_p |I_{1,p} - I_{2,p}| \qquad L2(I_1, I_2) = \sqrt{\sum_p (I_{1,p} - I_{2,p})^2}$$



# Hyperparameters

How do we choose the right K?

How do we choose between L1 and L2?

Both of these are **hyperparameters** - settings we choose about the algorithm that are not learned from the data.

# Choosing Hyperparameters

The Data

Model 1: **K** = 1 | **Distance** = L1

Model 2: **K** = 2 | **Distance** = L1

Model 3: **K** = 3 | **Distance** = L1

Model 4: **K** = 4 | **Distance** = L1

Model 5: **K** = 5 | **Distance** = L1

Model 6: **K** = 1 | **Distance** = L2

Model 7: **K** = 2 | **Distance** = L2

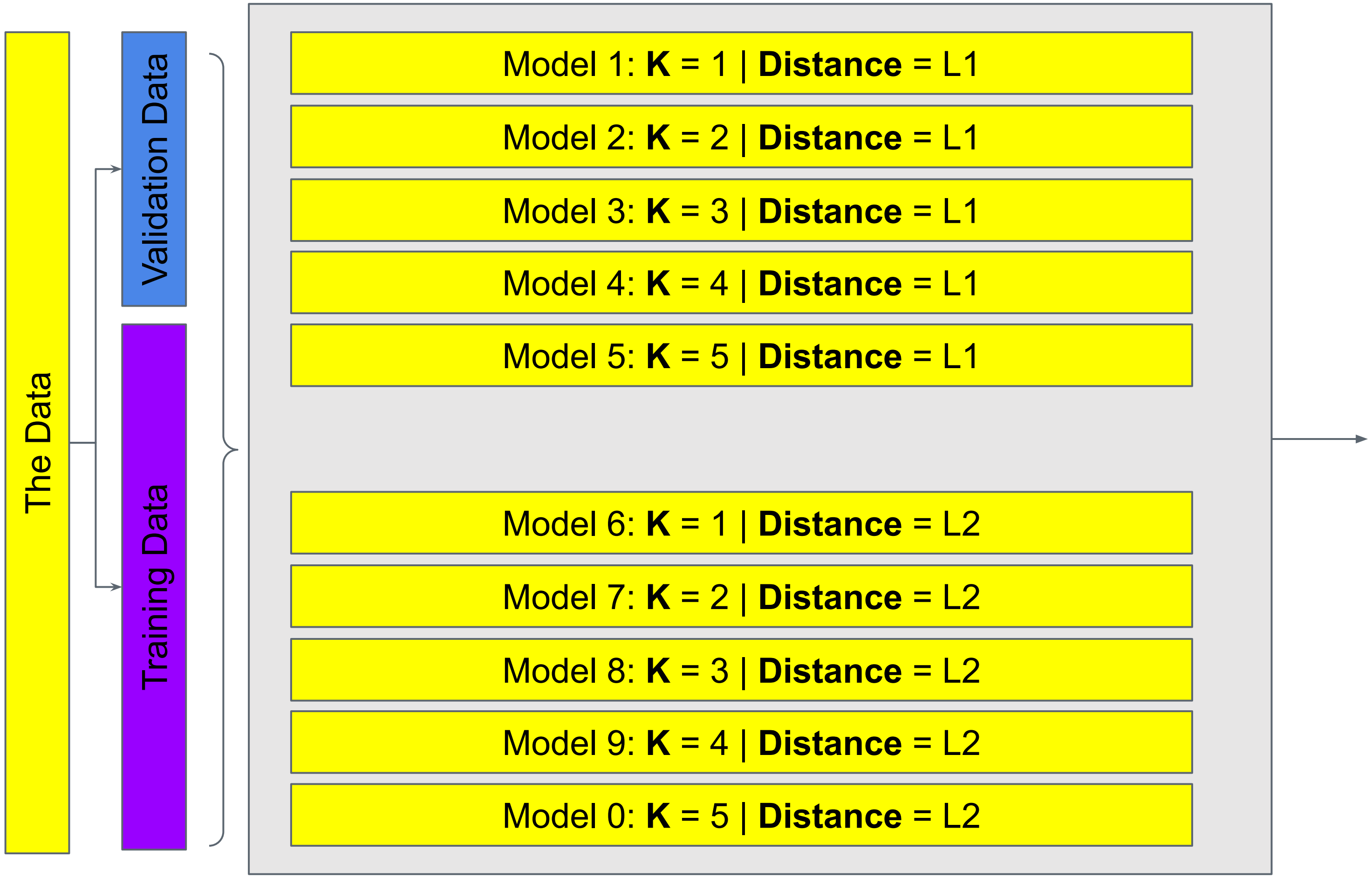
Model 8: **K** = 3 | **Distance** = L2

Model 9: **K** = 4 | **Distance** = L2

Model 0: **K** = 5 | **Distance** = L2

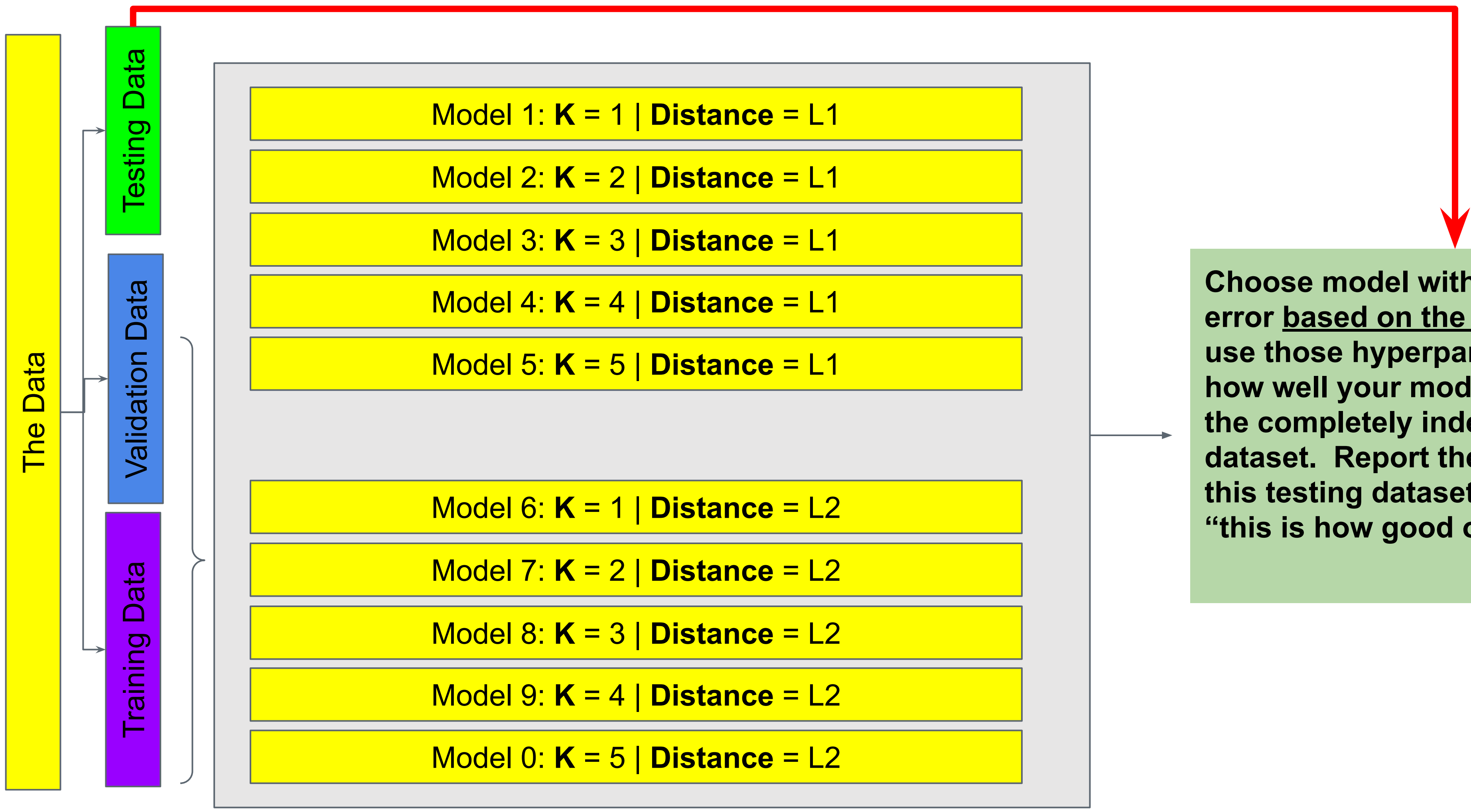
**Choose model with lowest overall error, use those hyperparameters. All data is used for fitting and accuracy calculation.**

# Choosing Hyperparameters



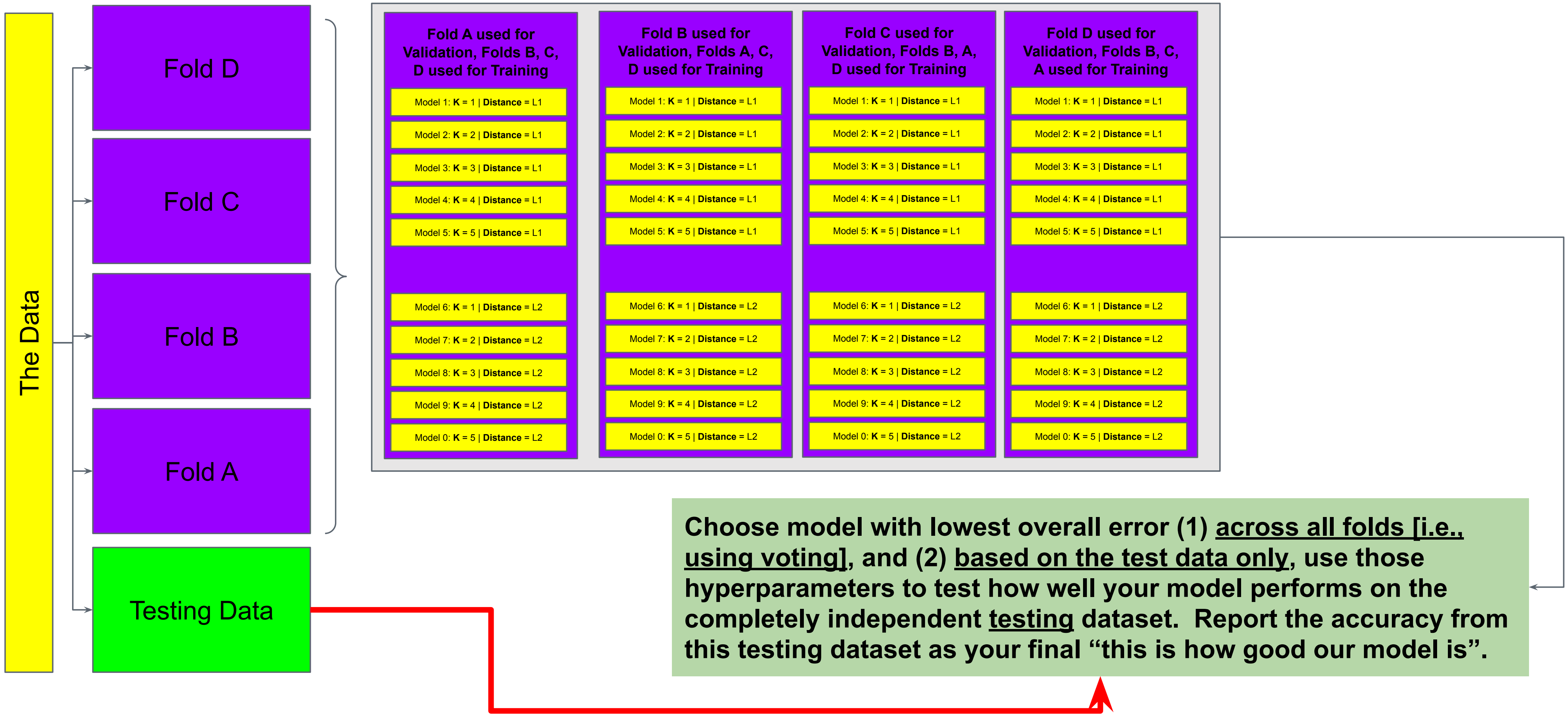

**Choose model with lowest overall error based on the test data, use those hyperparameters.**

# Choosing Hyperparameters




**Choose model with lowest overall error based on the test data only, use those hyperparameters to test how well your model performs on the completely independent testing dataset. Report the accuracy from this testing dataset as your final "this is how good our model is".**

# Cross Validation




**Choose model with lowest overall error (1) <u>across all folds [i.e., using voting]</u>, and (2) <u>based on the test data only</u>, use those hyperparameters to test how well your model performs on the completely independent <u>testing</u> dataset. Report the accuracy from this testing dataset as your final "this is how good our model is".**

# KNN

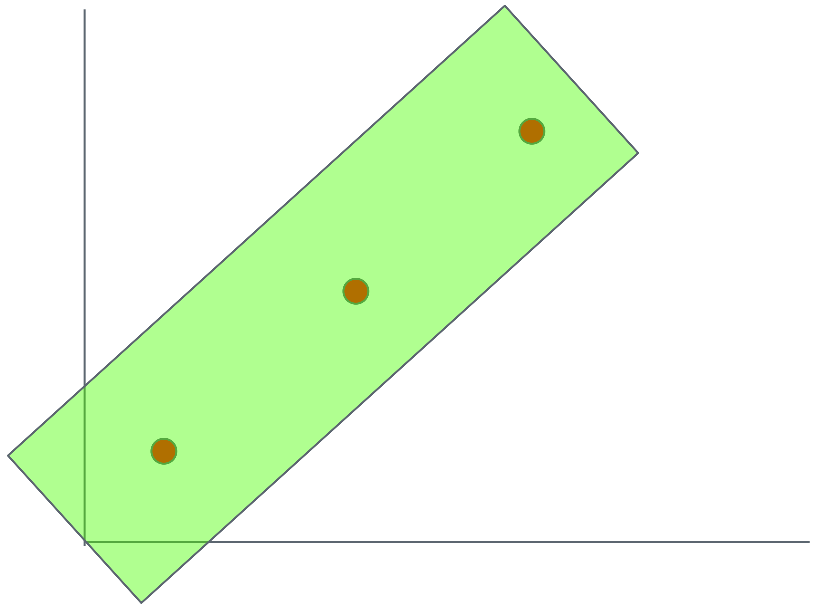
- Great as an example for some basic machine learning terminology.
- Not great for actual use.
  - Operational use very slow (training is fast, prediction is slow).
  - Simple distance metrics can’t capture perceptual differences that matter.
  - “Curse of Dimensionality”

# Dimensionality of Images

## "Normal Data"

**Dimensions**: 2

| **Observation** | **Height** | **Weight** |
|---|---|---|
| A | 3ft | 10lb |
| B | 4ft | 20lb |
| C | 5ft | 30lb |

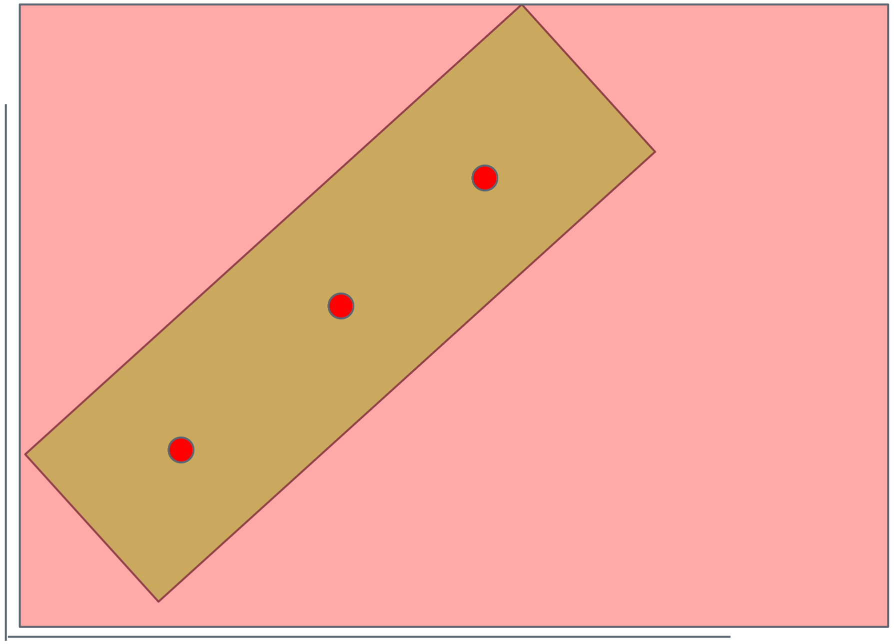


# Dimensionality of Images

<u>"Normal Data"</u>

**Dimensions**: 2

| **Observation** | **Height** | **Weight** |
|---|---|---|
| A | 3ft | 10lb |
| B | 4ft | 20lb |
| C | 5ft | 30lb |

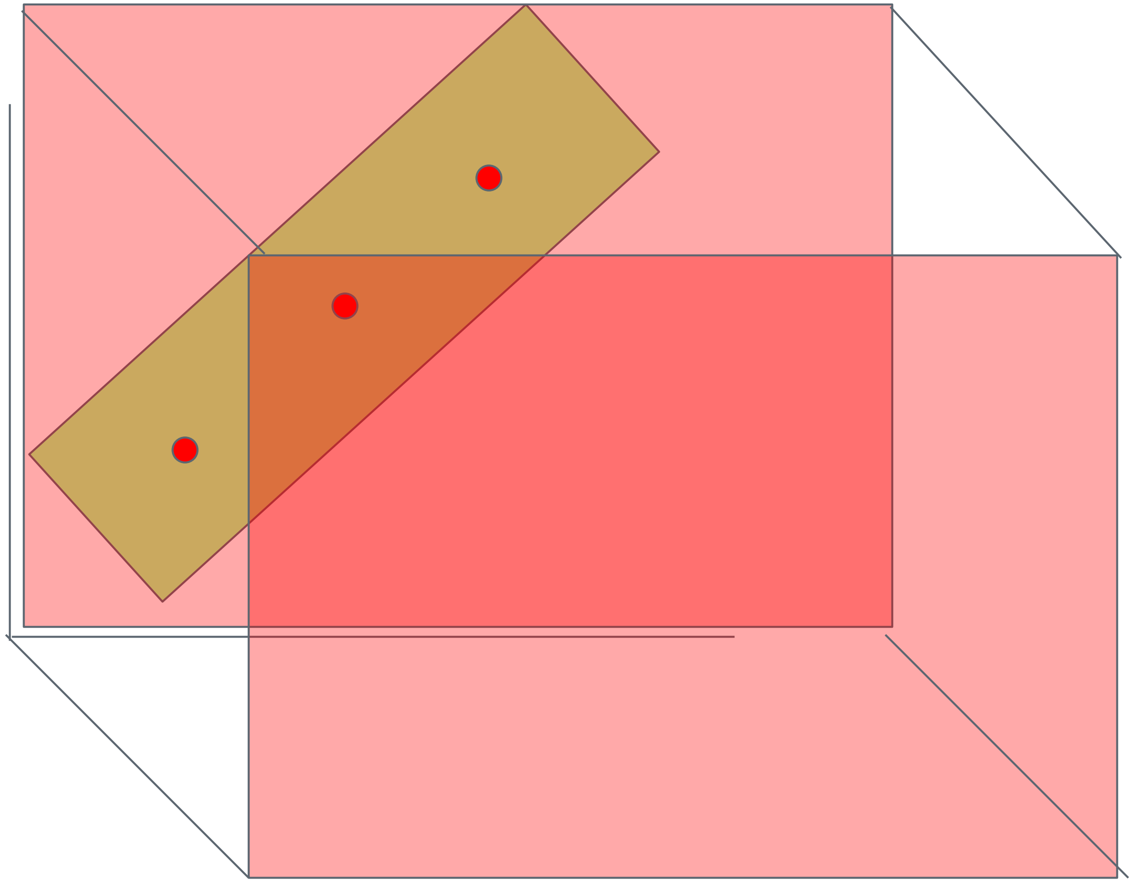


# Dimensionality of Images

## "Normal Data"

**Dimensions**: 2

| Observation | Height | Weight | Age |
| --- | --- | --- | --- |
| A | 3ft | 10lb | 5 |
| B | 4ft | 20lb | 5 |
| C | 5ft | 30lb | 5 |



# Dimensionality of Images

## Image Data

**Dimensions**: Thousands

**Observations**: 3

| Observation | Pixel 1 | Pixel 2 | ... | Pixel 12000 | Pixel 12001 |
|---|---|---|---|---|---|
| A | 10 | 10 | | 25 | 85 |
| B | 20 | 20 | | 35 | 75 |
| C | 30 | 30 | | 65 | 95 |

# Recap

- We are exploring the topic of image classification, in which we are using a large training set of images that have human-created labels, and we are using this to predict the correct labels for a test set of data.

- KNN is an example of how you can do this, though not a good one.  It predicts based on the nearest training example.

- In the case of KNN, the distance metric (L1 vs. L2) and K are the hyperparameters you must choose.

- A validation set and test set allow you to choose appropriate hyperparameters.

- For small datasets, cross-fold validation can improve the robustness of your results.

# Reminders

- Remember to check in on Piazza with any questions!
- Piazza will also have information on the first lab.
- Group study is encouraged, but your submissions should be your own!