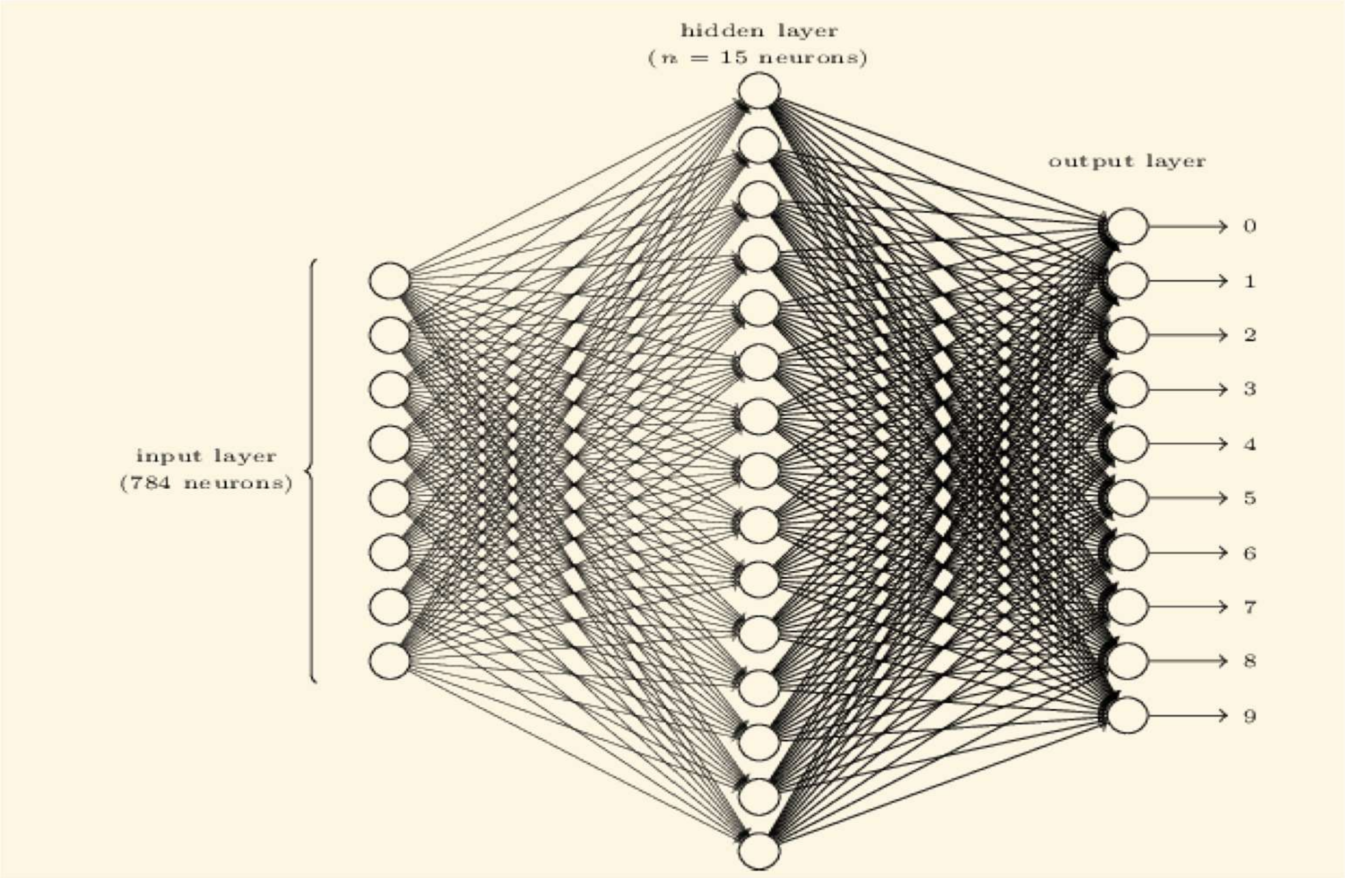
Deep Learning with Python: MINST Example

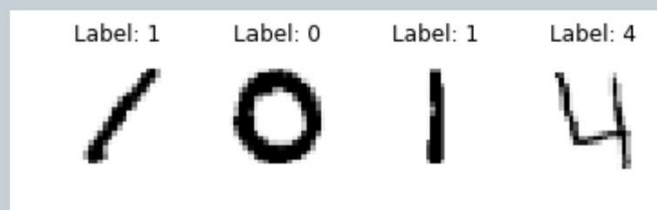
Deep Learning for MNIST Data

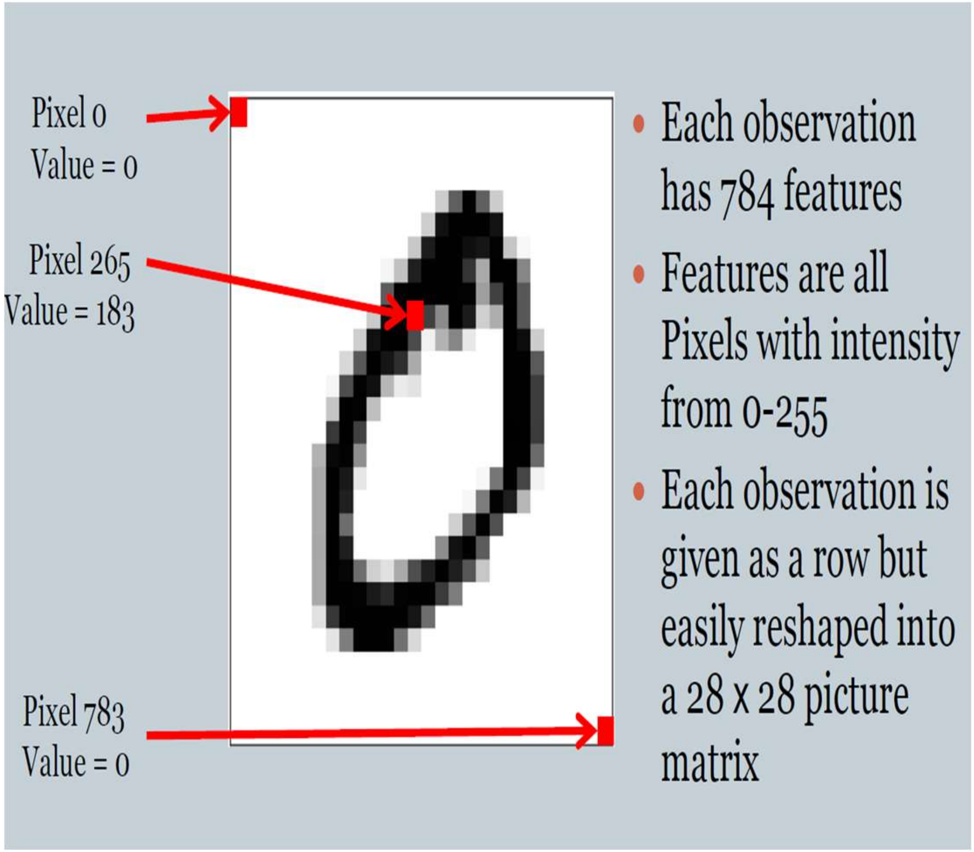


MNIST Data

- The MNIST database (Modified National Institute of Standards and Technology database) of handwritten digits consists of a training set of 60,000 examples, and a test set of 10,000 examples.
- It is a subset of a larger set available from NIST. Additionally, the black and white images from NIST were size-normalized and centered to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.

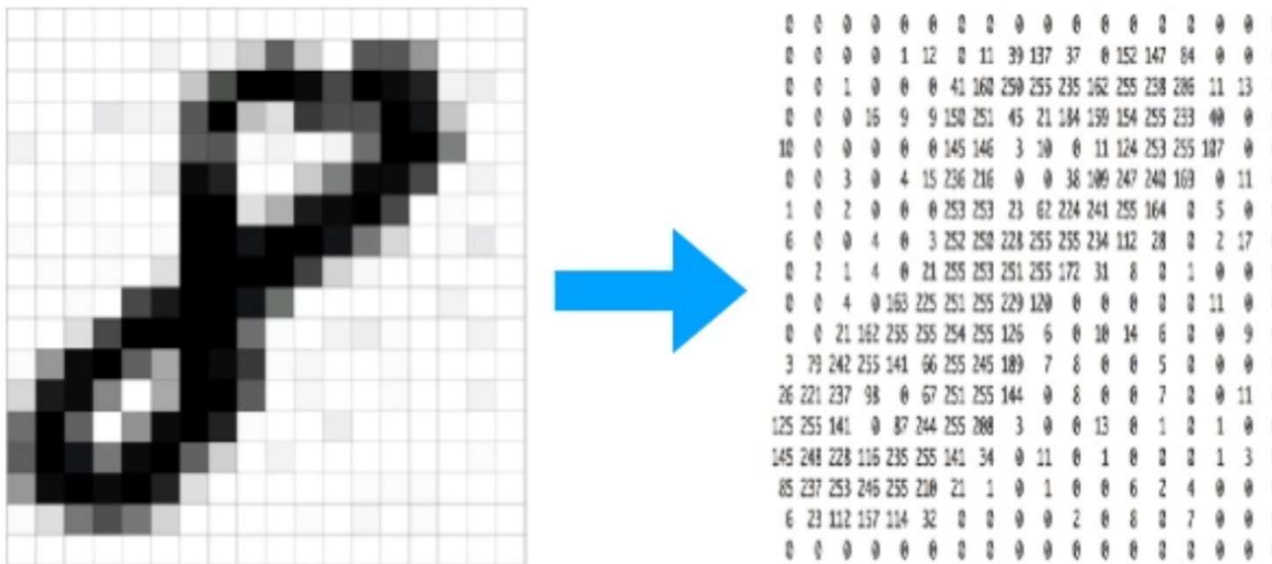
Each observation represents an image of a handwritten digit 0-9:



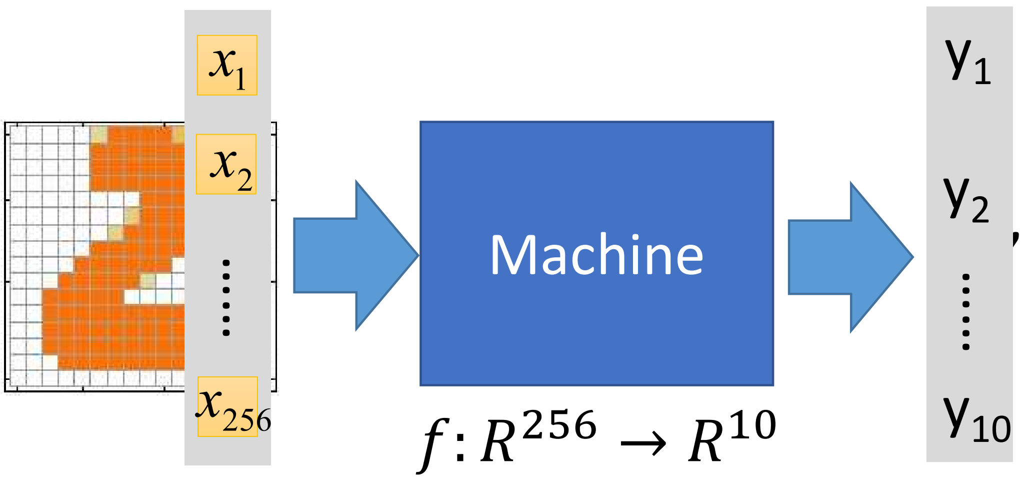


- Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total.
- Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

How does a computer read an image?



Handwriting Digit Recognition



$$f: R^{256} \rightarrow R^{10}$$

In deep learning, the function f is represented by neural network

Pixel data of an image

- For the 28×28 pixel images we've been using, this means our network has 784 (=28×28) input neurons.

- Each images is then encoded in a $m=785-1=784$ feature vector, representing the intensity of the pixel. Each element of the vector contains an integer encoding the intensity of a pixel. In other words the original image is decomposed in a 28×28 pixel grid, which is reshaped to be represented as a $28\times28=784$ column vector.

- We will train the network's weights and biases so that the network's output would - we hope! - correctly identify the input image: '0', '1', '2', ..., '8', or '9'.

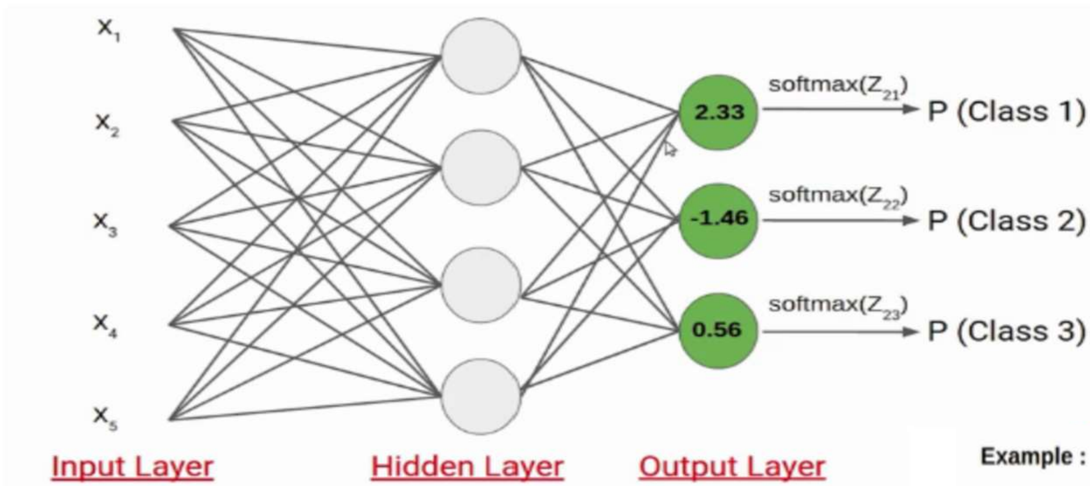
Training data

- The training data set has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.
- Each pixel column in the training set has a name like pixelx, where x is an integer between 0 and 783, inclusive. To locate this pixel on the image, suppose that we have decomposed x as x = i * 28 + j, where i and j are integers between 0 and 27, inclusive. Then pixelx is located on row i and column j of a 28 x 28 matrix, (indexing by zero).
- For example, pixel31 indicates the pixel that is in the fourth column from the left, and the second row from the top, as in the ascii-diagram below.

What's Softmax Function?

$$f_i(x) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Example: Softmax layer as the output layer



Example :

$$2.33 \rightarrow P(\text{Class 1}) = \frac{\exp(2.33)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.83827314$$

$$-1.46 \rightarrow P(\text{Class 2}) = \frac{\exp(-1.46)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.01894129$$

$$0.56 \rightarrow P(\text{Class 3}) = \frac{\exp(0.56)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.14278557$$

- Softmax function is used to convert the numerical output to values in the range [0, 1]
- The output of softmax function can be seen as probability distribution given the output sums up to 1
- Softmax function is used in multiclass classification methods such as neural networks, multinomial logistic regression, multiclass LDA, Naive Bayes classifier.
- Softmax function is used to output action probabilities in case of reinforcement learning
- Softmax function is used as an activation function in the last / final layer of neural network algorithm.

Cross Entropy Loss

This is the most common setting for classification problems. Cross-entropy loss increases as the predicted probability diverges from the actual label.

Mathematical formulation :-

$$CrossEntropyLoss = -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

Cross entropy loss

For multiclass classification, we calculate a separate loss for each class label per observation and sum the result.

Accuracy

- For example, if y_true is [1, 2, 3, 4] and y_pred is [0, 2, 3, 4] then the accuracy is 3/4 or .75. If the weights were specified as [1, 1, 0, 0] then the accuracy would be 1/2 or .5.

- one epoch = one forward pass and one backward pass of all the training examples
- batch size = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.
- number of iterations = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).
- Example: if you have 1000 training examples, and your batch size is 500, then it will take 2 iterations to complete 1 epoch.

verbose: Integer. 0, 1, or 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per epoch.

By setting verbose 0, 1 or 2 you just say how do you want to 'see' the training progress for each epoch.

`verbose=0` will show you nothing (silent)

`verbose=1` will show you an animated progress bar like this:

```
[==============================]
```

`verbose=2` will just mention the number of epoch like this:

```
Epoch 1/10
```

Avoiding Overfitting

- With thousands of weights to tune, overfitting is a problem

Early stopping (when performance starts dropping)

- Regularization terms added to cost function during training
- Dropout –ignore say 50% of all neurons randomly at each training step
- Works surprisingly well! Forces your model to spread out its learning

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
```

One hot encoding (or dummy variables)

id	country
0	russia
1	germany
2	australia
3	korea
4	germany

id	country=russia	country=germany	country=australia	country=korea
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	0	1	0	0

Feature detectors

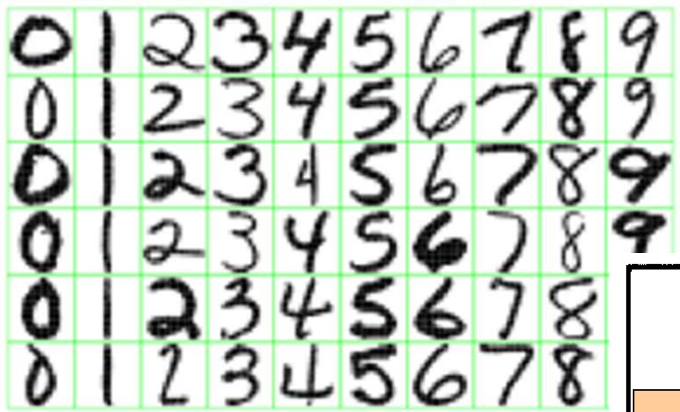
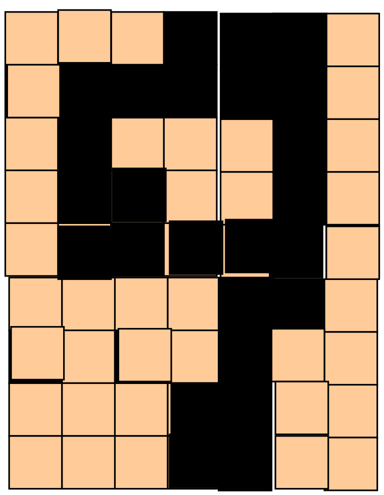
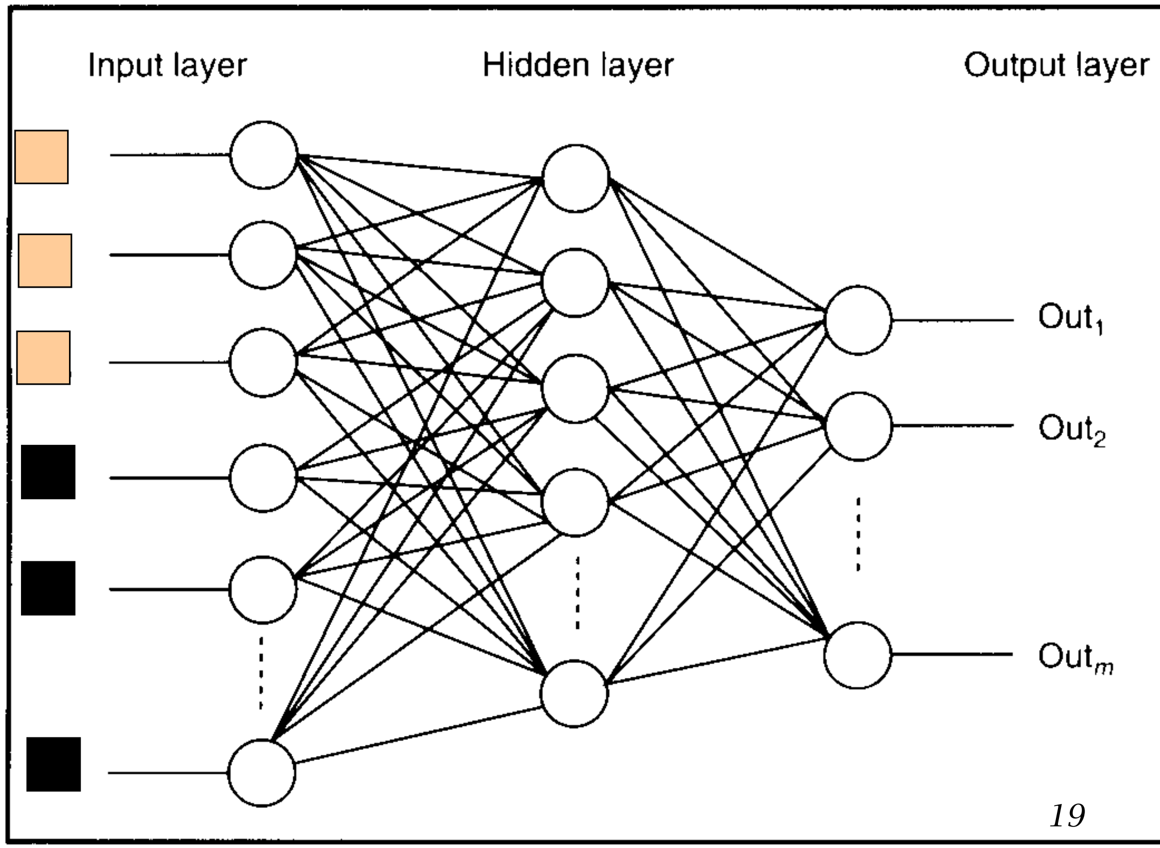


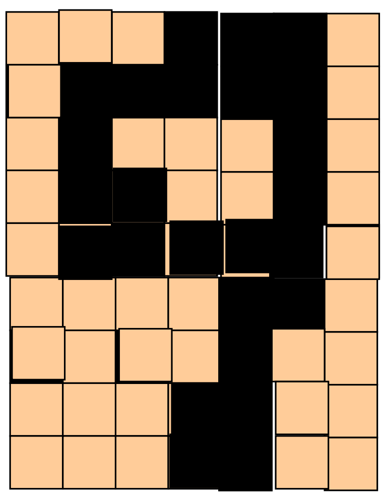
Figure 1.2: *Examples of handwritten digits from postal envelopes.*

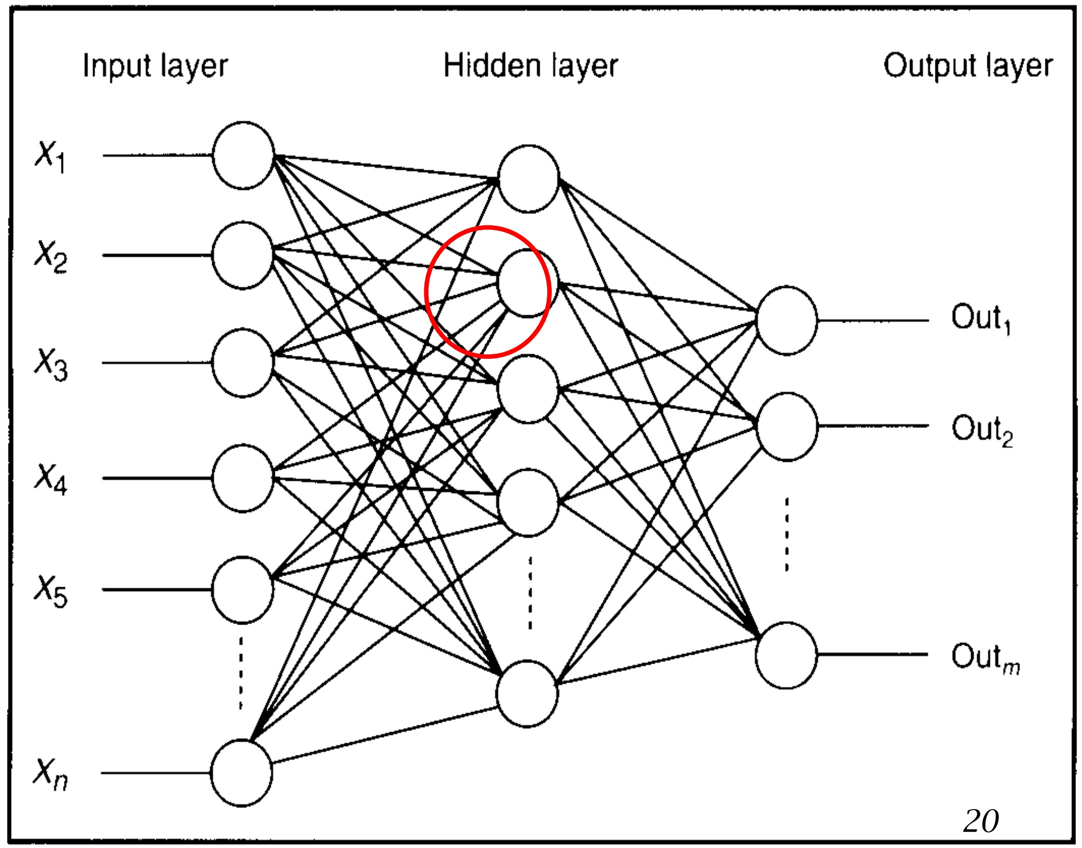




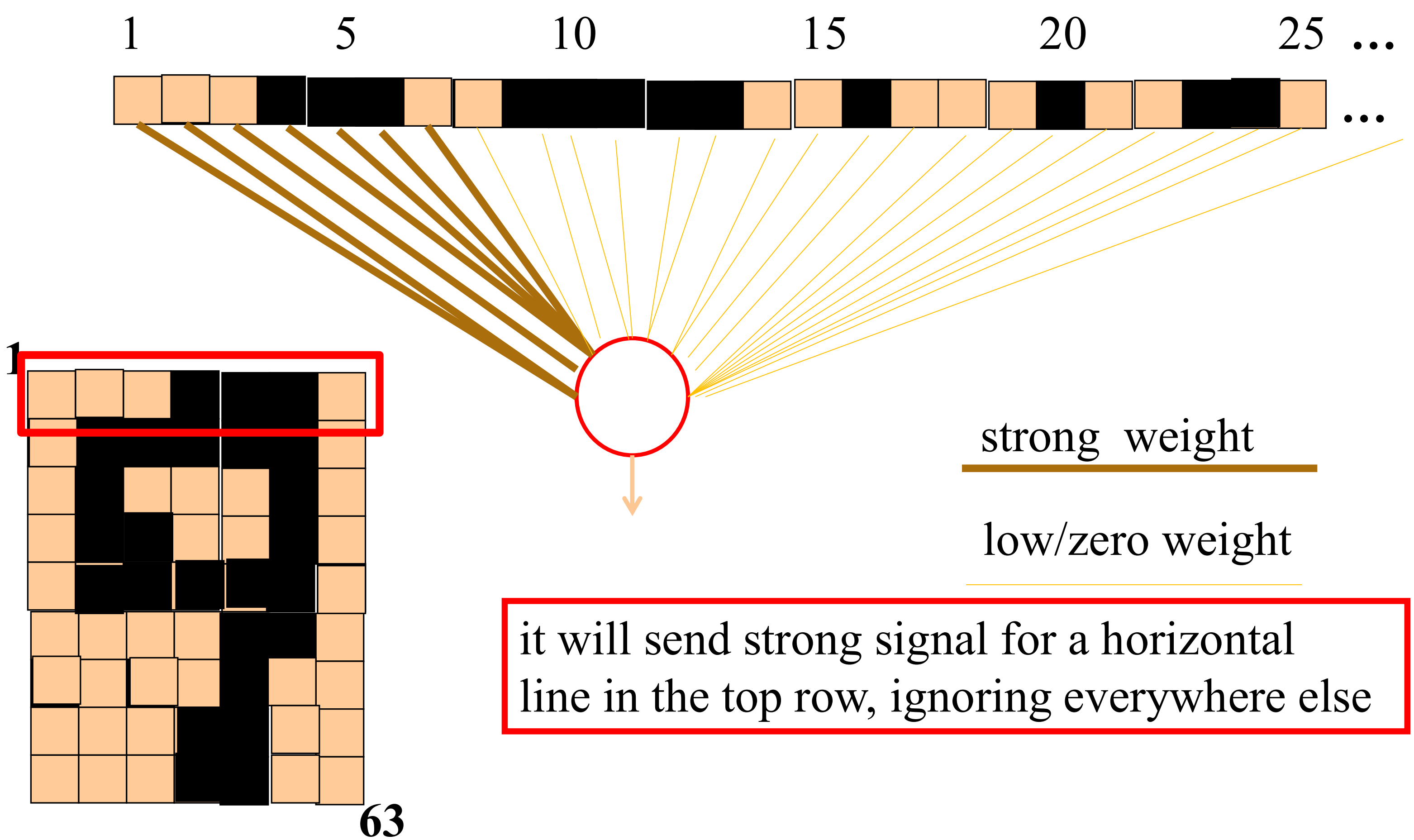
what is this unit doing?

Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*



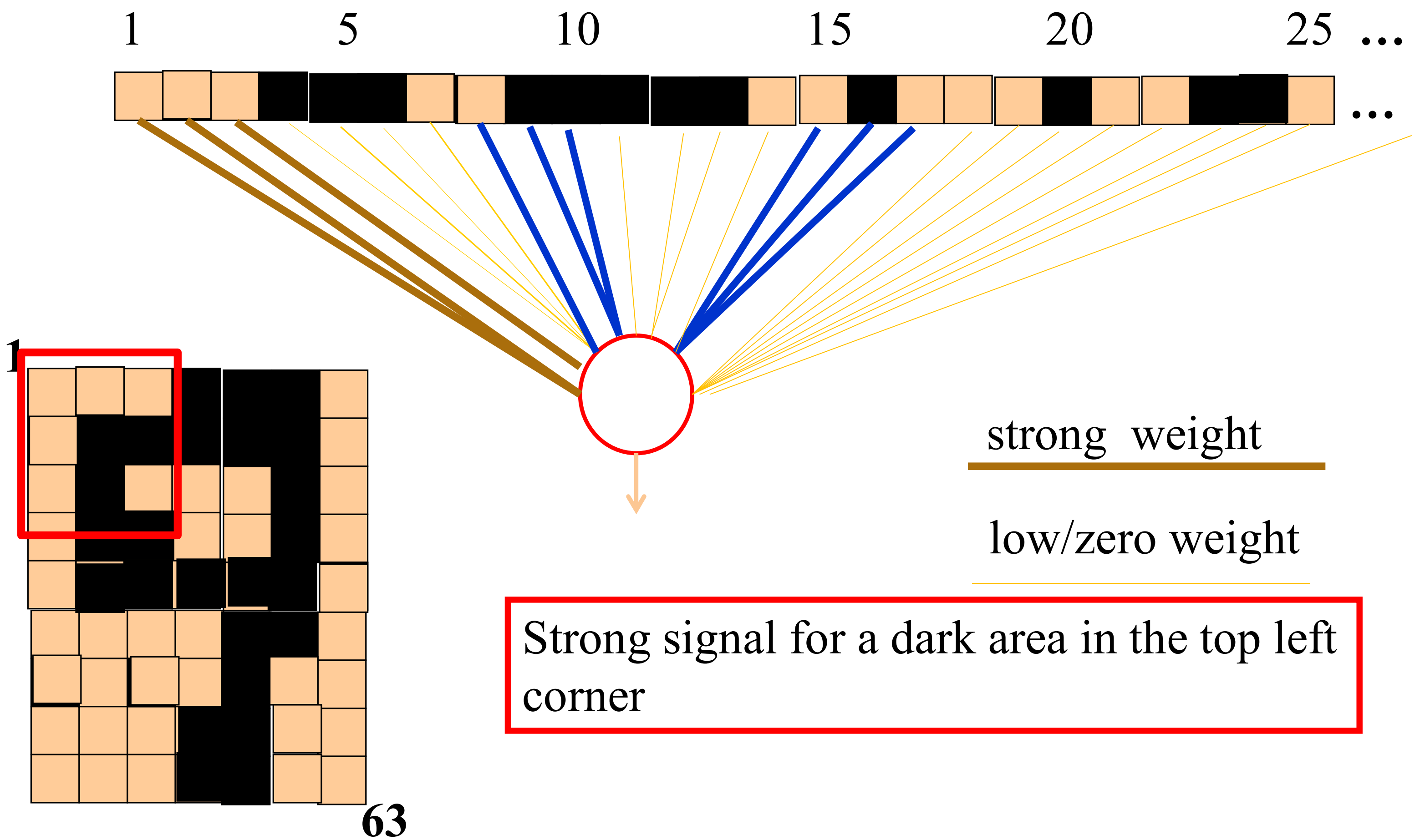


What does this unit detect?



it will send strong signal for a horizontal line in the top row, ignoring everywhere else

What does this unit detect?



Strong signal for a dark area in the top left corner

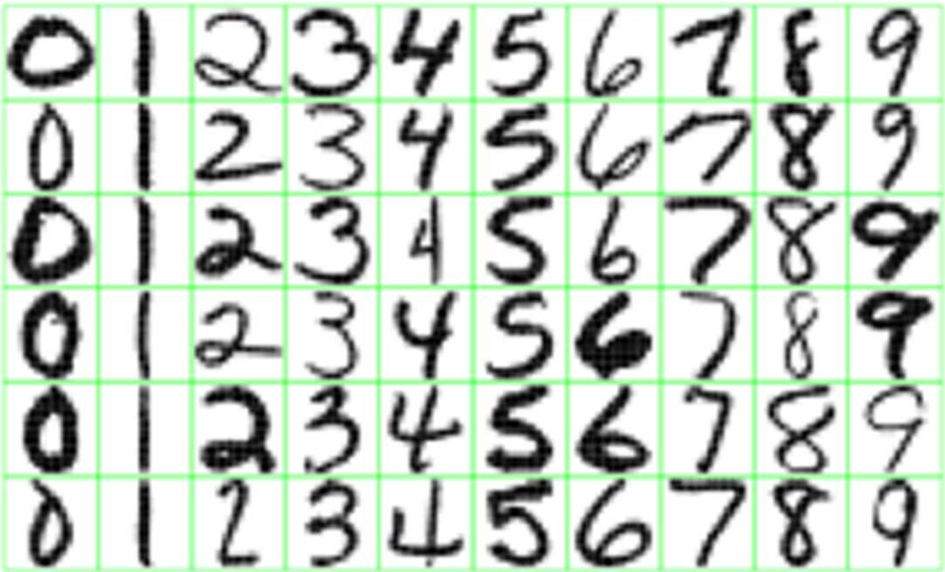
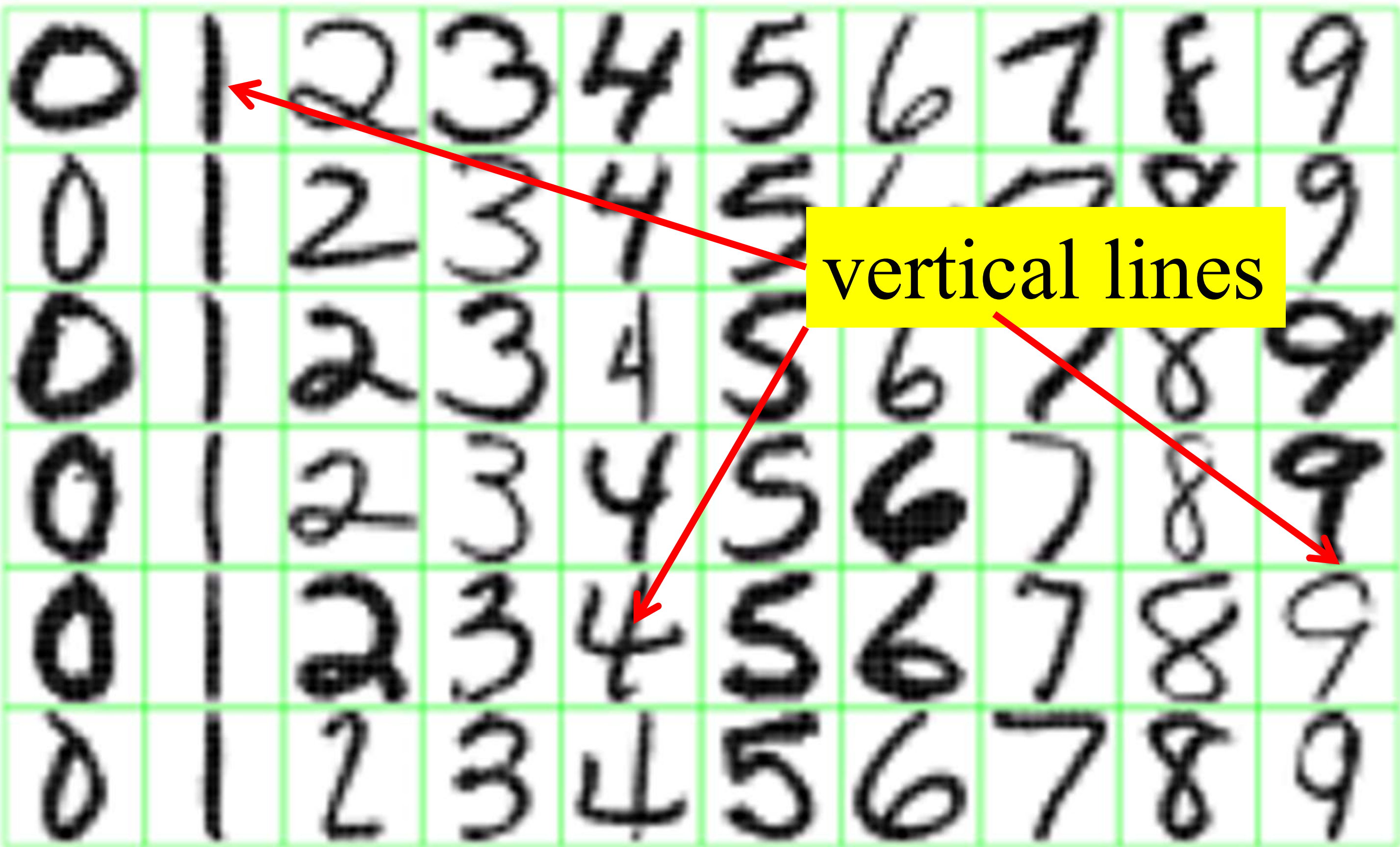


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

What features might you expect a good NN to learn, when trained with data like this?

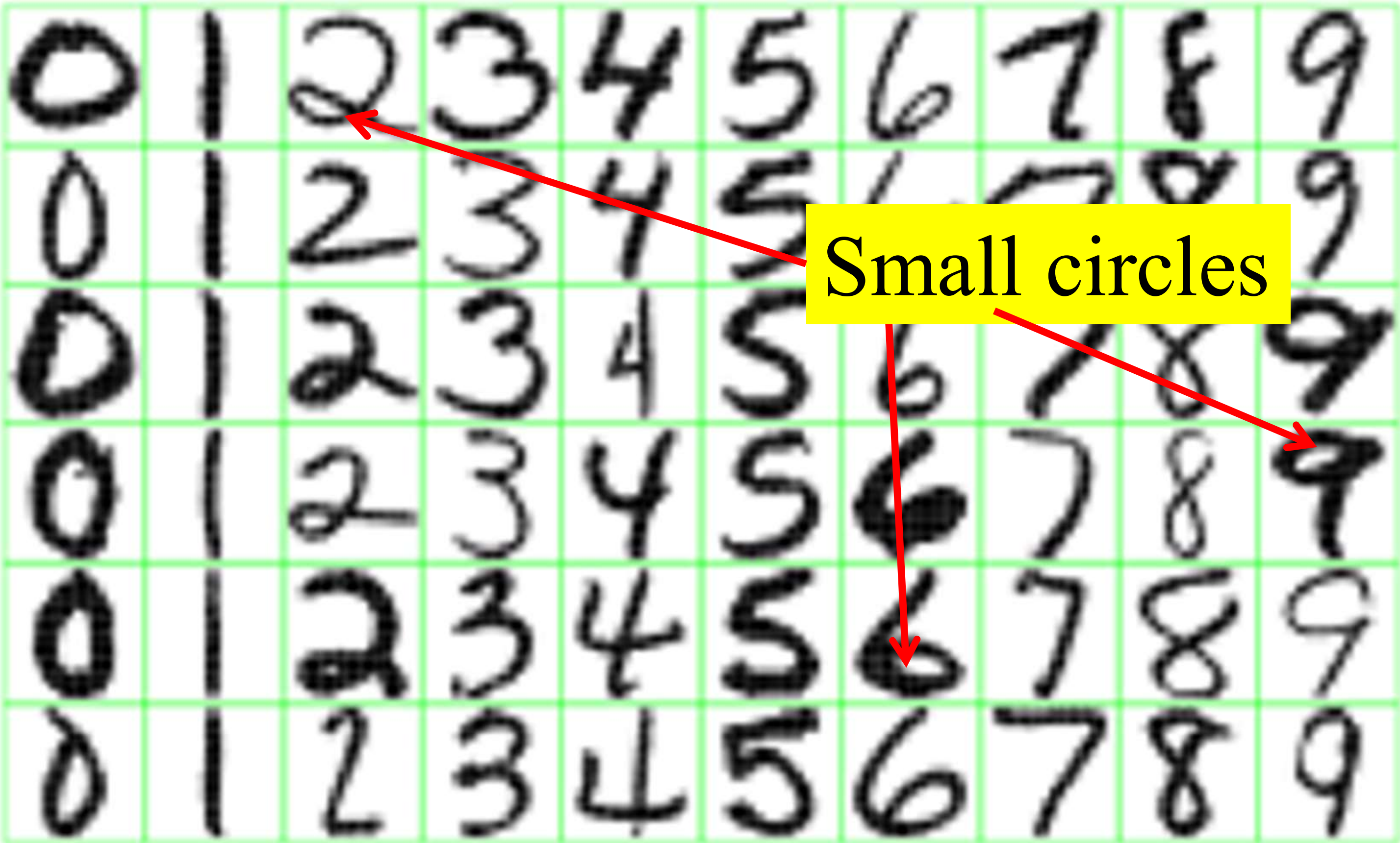


1

Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

1



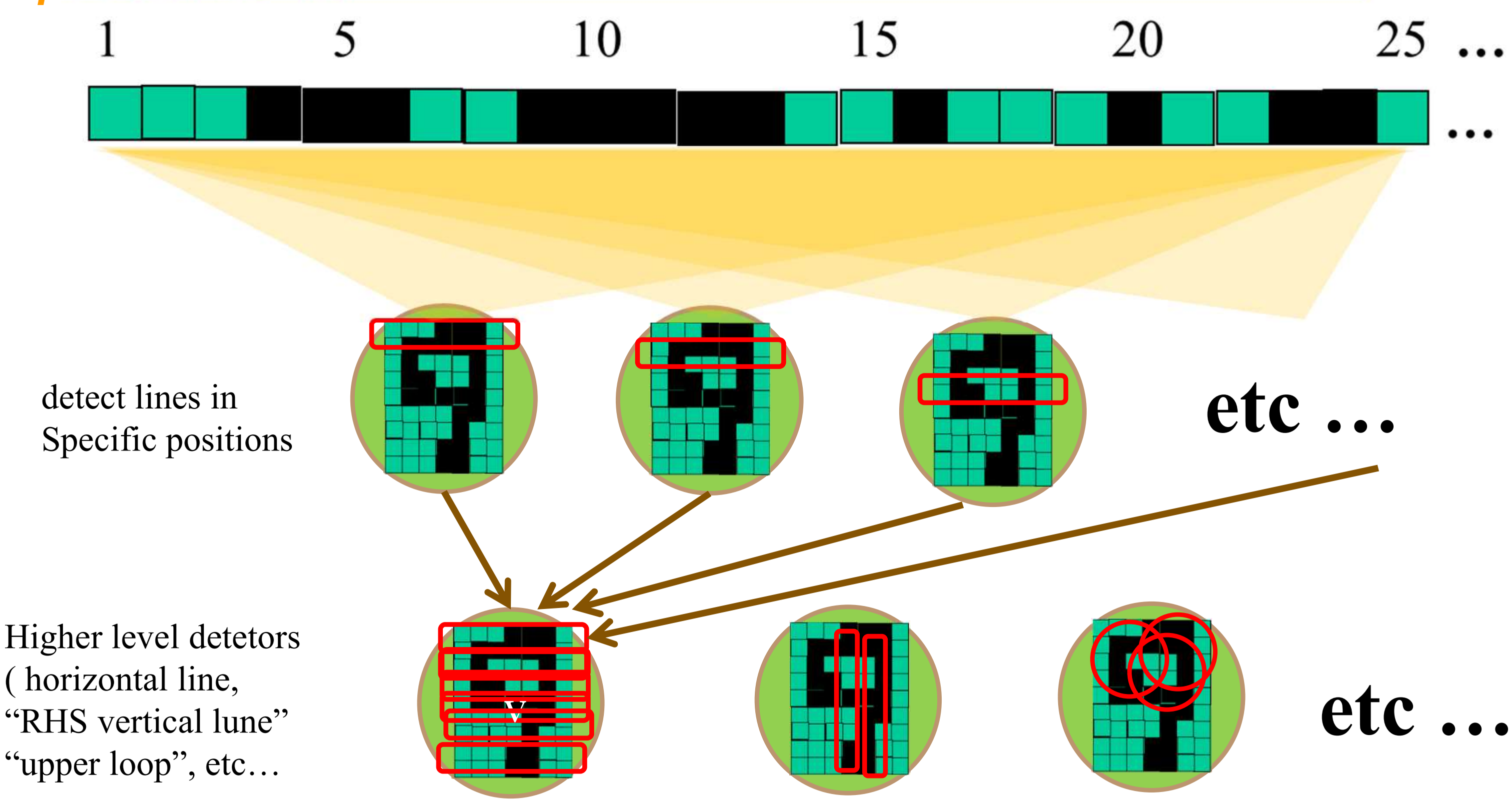
1

Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

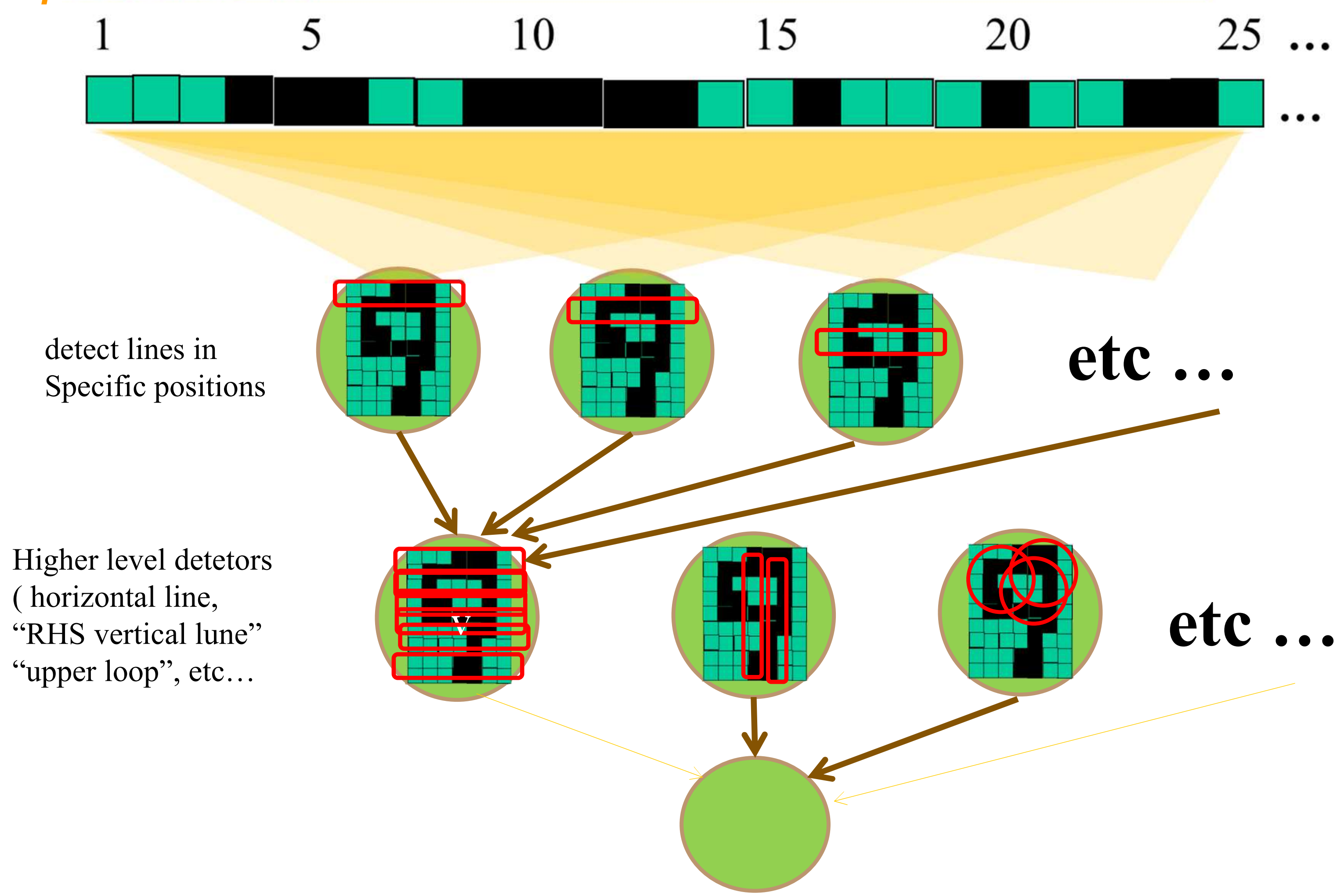
1

But what about position invariance ???
our example unit detectors were tied to
specific parts of the image

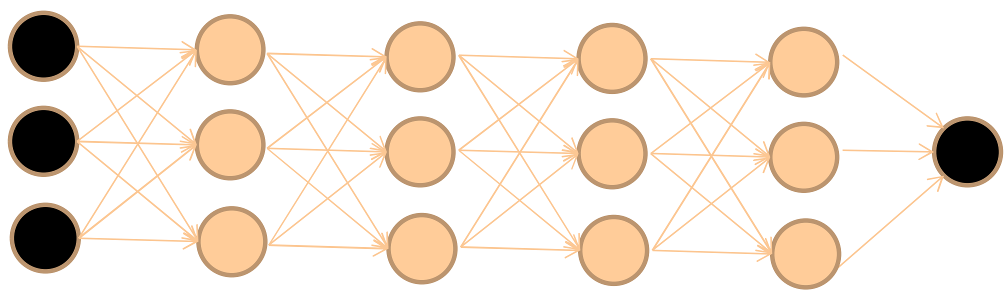
successive layers can learn higher-level features …



successive layers can learn higher-level features …

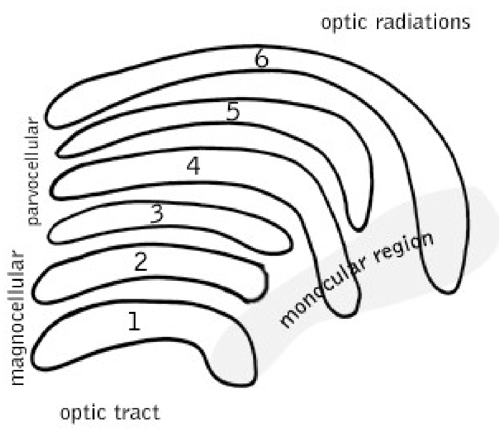


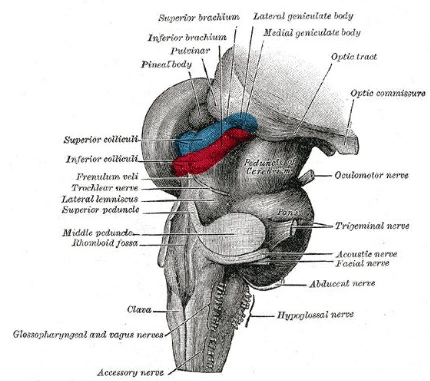
So: multiple layers make sense



So: multiple layers make sense

Your brain works that way





So: multiple layers make sense

Many-layer neural network architectures should be capable of learning the true underlying features and 'feature logic', and therefore generalise very well …