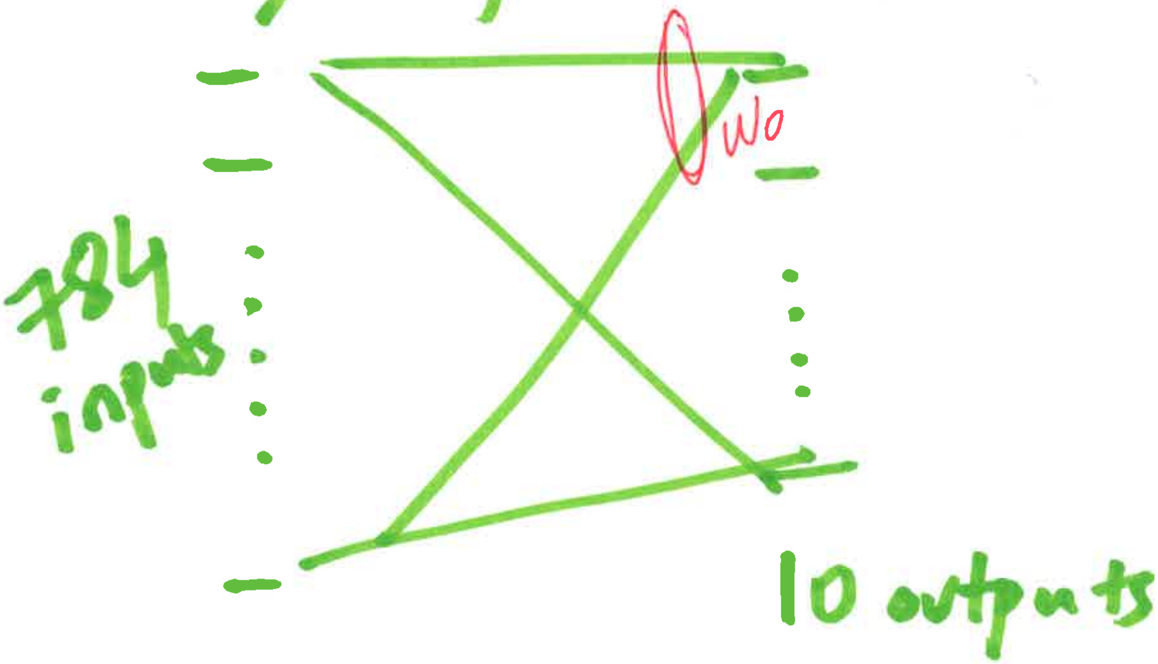
ECE/CS 559 - Neural Networks

Convolutional Neural Networks

- Remember why the single-layer FF network for classifying MNIST digits failed:



The output should be [1, 0, ..., 0] if the input is a "0" digit, [0 1 0 ... 0] if the input is a "1" digit, and so on.

Now imagine what should the corresponding weights / filters "look like"?

For example, let us call the weights corresponding to the first output neuron as W_0, as shown on Page. 1. This neuron should output a "1" for all "0" digits and a "0" for all digits $\neq 0$. Suppose that we use a "1" for a black pixel and "-1" for a white pixel. If we want to maximize the output for an input (say)

1	1	-1	1	-1

then the filter should exactly be <u>matched</u> to the input, i.e, the filter should also be:

1	1	-1	1	-1

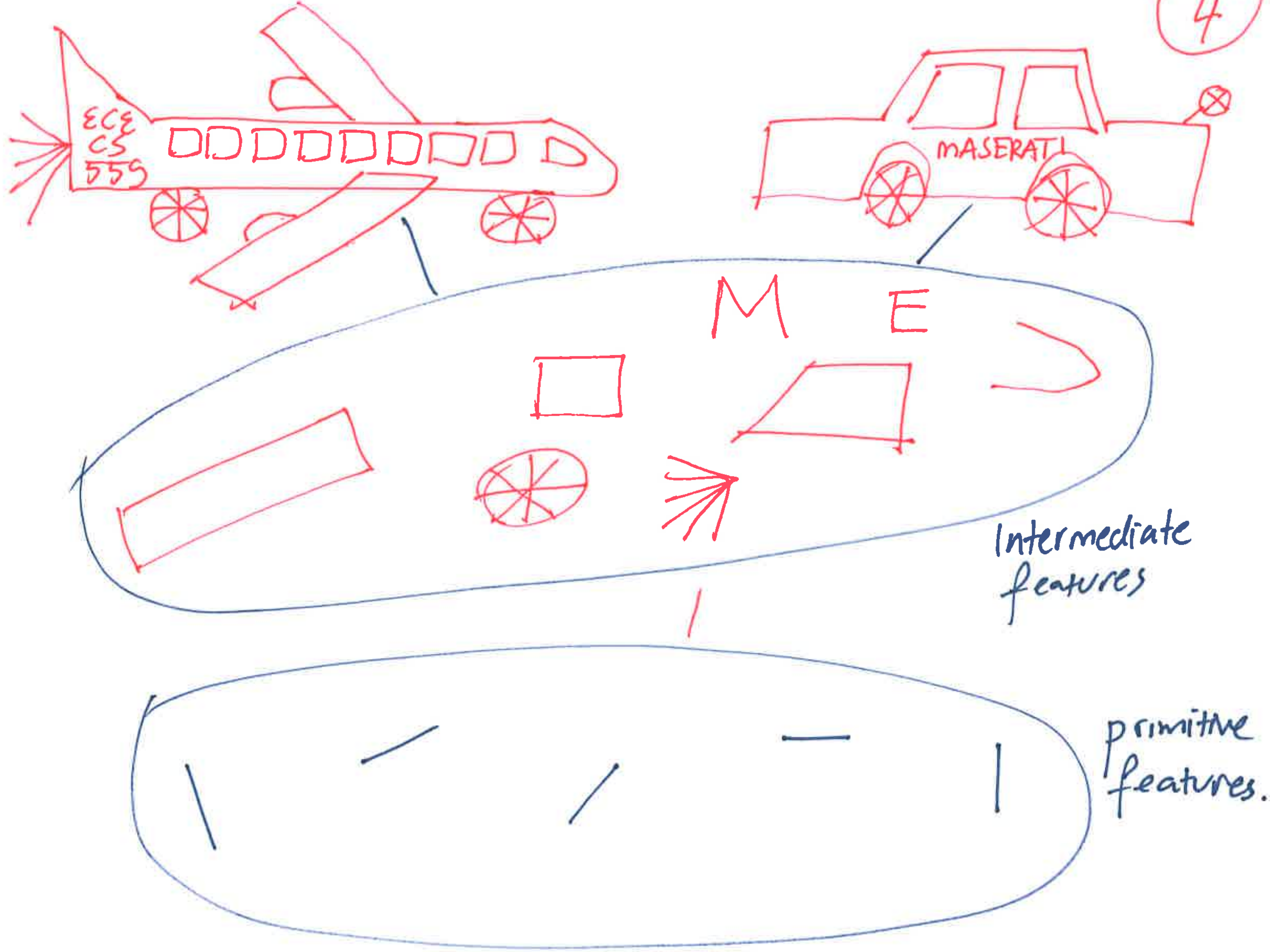
So, if we want an output of "1" for an image that looks like a zero, we ideally need a filter that "<u>looks like a zero</u>". Meanwhile this filter should look unlike a "1, 2, 3, 4, 5, 6, 7, 8, 9" so that it will not be matched to these undesired images. So, try to imagine a digit that looks like a zero, but does not look like a one, two, three, four, etc. Tough, huh?

So, why does a single layer fail? We try to very quickly decide on complex features.

Solution: Use multilayer networks.

Problems: Too many parameters may result in over-fitting, performance degradation, increased implementation complexity.
e.g. increased input image resolution
⇒ more free parameters..

Convolutional Neral Networks (CNNs): Become state of the art for image & video processing.
- provides invariance to translation, scaling skewing, rotation, & other distortions relevant to image processing.
- inspired from the visual cortex of the cat and studies by Hubel & Wiesel.
- Alternatively, we first imagine extracting simple features like straight edges, corners, then move on to moderately complex features like a wheel, an arm or a leg, and then move to objects like a car, human, airplane, etc.



Primitive features form intermediate features, which in turn, form more complex features, or ultimately the objects we wish to classify.

* Extract primitive features in the first layer

Intermediate features in the second layer

⋮

and so on.

Observation. **Primitive features**

1. are **small** in size
2. can be anywhere in the image.

Due to 1, we just need a small filter to capture one primitive feature. Due to 2, we will slide the filter all over the image to capture the primitive feature, wherever it may be! This is called a convolution operation.

How do we choose the primitive filters??

We will not!!! Backpropagation algorithm will train the filters for us. In the past, people used to work with handcrafted filters (back when computers where not as powerful and deep learning methods were not as well-developed).

CNNs introduce two new kinds of neuron layers. One is the convolutional layer as outlined in the previous page. A convolutional layer is fundamentally no different than a layer of a typical feedforward neural network, but there are constraints on the weights and the "receptive fields" of the neurons. Typically, a convolutional layer consists of multiple [redacted] filters to be slided across the input to the convolutional layer. Assume a 5X5 input image, and consider a convolutional layer with one 2X2 [redacted] filter:

[redacted]

1	2	0	3	1
-1	-1	2	0	1
1	-1	-1	0	1
3	1	2	-1	1
1	4	0	1	2

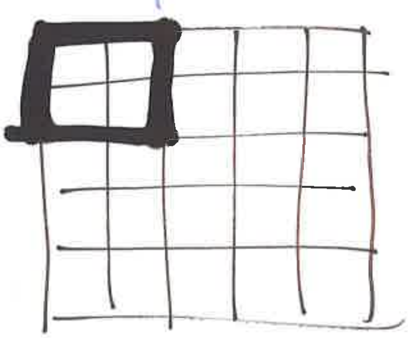
This is the input

1	-1
0	2

This is the filter.

[redacted]

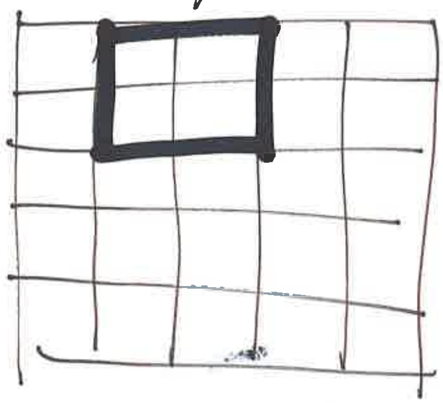
Typically, we would put the filter on the top left of the input:



... and take the inner product... we would get:

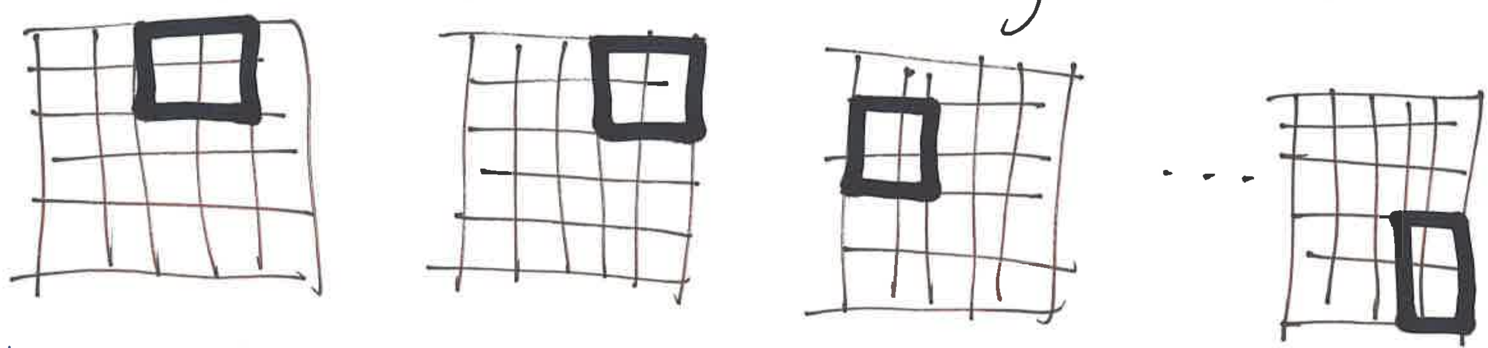
$$1\cdot 1 + (2)(-1) + (-1)\cdot 0 + (-1)(2) = -3$$

Then, we slide the filter one unit to the right and get the second output:



$$2\cdot 1 + 0\cdot(-1) + (-1)\cdot 0 + 2\cdot 2 = 4.$$

The sequence of calculations can go like this.



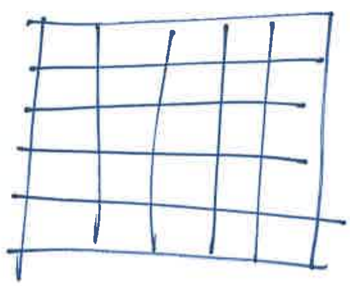
The end result is a 4×4 feature map consisting of all the 16 numbers we have calculated by sliding the filter:

-3	4	...	...
...	..		

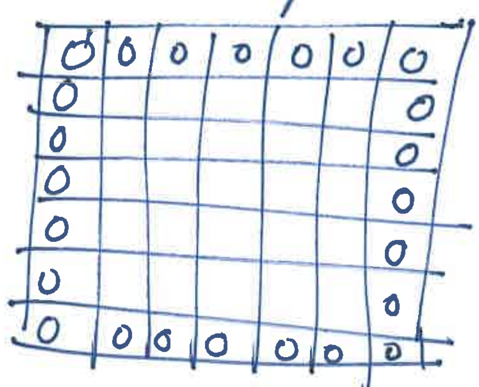
You can also imagine that we have 16 neurons that share the weights [1 -1 0 2]

Each neuron is only connected to a subset of the inputs called its local receptive field.

* We almost always have more than one filter to extract multiple features. In this case, one creates multiple feature maps in a convolutional layer. Each filter is allowed to have different weights.
* Several variations of the filtering idea exists, such as zero padding. Instead of



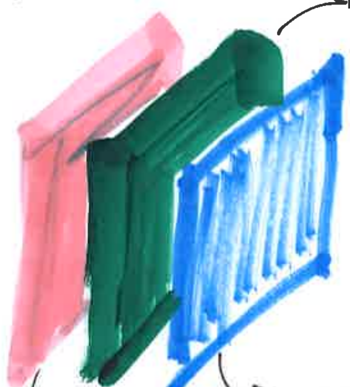
/ we can apply convolutions to the zero padded version.



* One can also define a "stride", i.e how many squares to "jump" when sliding the filter.
* The sliding manner in which local fields of neurons are calculated is analogous to signal convolution. This is also why the networks are called convolutional networks.
* Convolutions may be followed by any activation function like Relu, tanh, etc.

Handling colored inputs.

In this case, the image is decomposed to R, G, and B channels. Each channel is a separate image. We can stack the channels on top of eachother to get a multichannel image that looks like as follows:



A filter to be applied to this "3D" input should now also be 3D. For example, if the image is 4x4, and the filter is 2x2; the first neuron local field would be:

3-channel input

$$\begin{bmatrix} 1 & 1 & 2 & 1 \\ 2 & 0 & -1 & 1 \\ -1 & 1 & 0 & -1 \\ 0 & 1 & 2 & 3 \end{bmatrix} \quad \begin{bmatrix} 1 & -1 & 1 & -1 \\ 0 & 1 & 0 & 2 \\ 2 & 3 & 0 & 1 \\ 1 & 4 & 3 & 1 \end{bmatrix} \quad \begin{bmatrix} 4 & 1 & 4 & 2 \\ 2 & 3 & 1 & 3 \\ 1 & 2 & 3 & 1 \\ 0 & 1 & 2 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 \\ 1 & 2 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 \\ 0 & 6 \end{bmatrix} \quad \begin{bmatrix} 3 & 2 \\ 1 & 0 \end{bmatrix}$$

3-channel-filter.

$$1\cdot 1 + 1\cdot -1 + 2\cdot 1 + 0\cdot 2$$
$$+ 1\cdot 1 + (-1)\cdot 0 + 0\cdot 0 + 1\cdot 6$$
$$+ 4\cdot 3 + 1\cdot 2 + 2\cdot 1 + 3\cdot 0$$
$$= \text{whatever}.$$

Handling multi-channel inputs

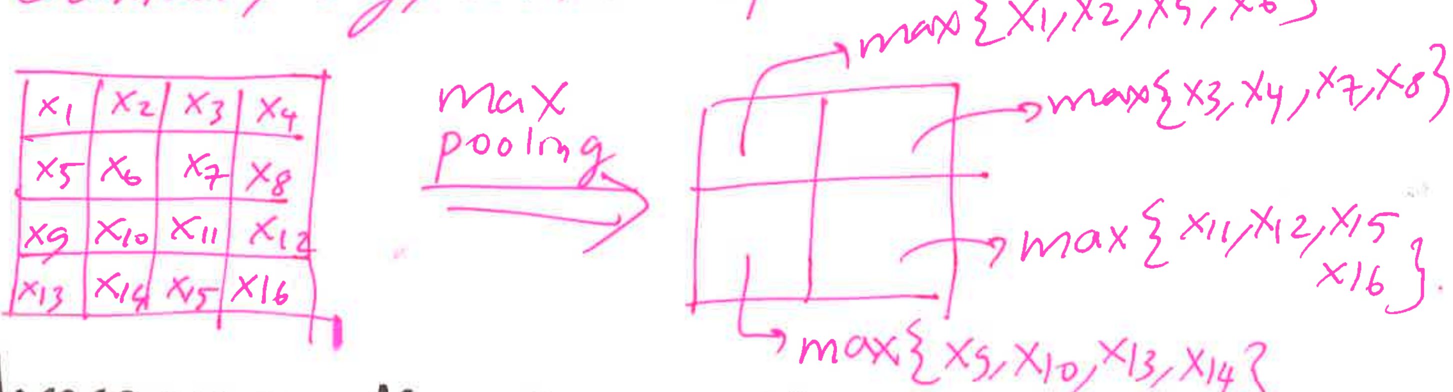
Are done in the same manner.

Suppose you applied a convolutional layer and got 10 feature maps. You can follow this by another convolutional layer. The filters of this new layer may act on **all 10** channels of the previous layer (or a subset of them).

Pooling layers.

Convolutional layers are typically followed by pooling layers to reduce the dimensions of the feature maps. A commonly used method is <u>max-pooling</u>:

Consider, eg, a stride of 2:



Average pooling is another possibility.

11

A typical CNN architecture
(From LeCun et.al. "Gradient-Based Learning Applied to Document Recognition").

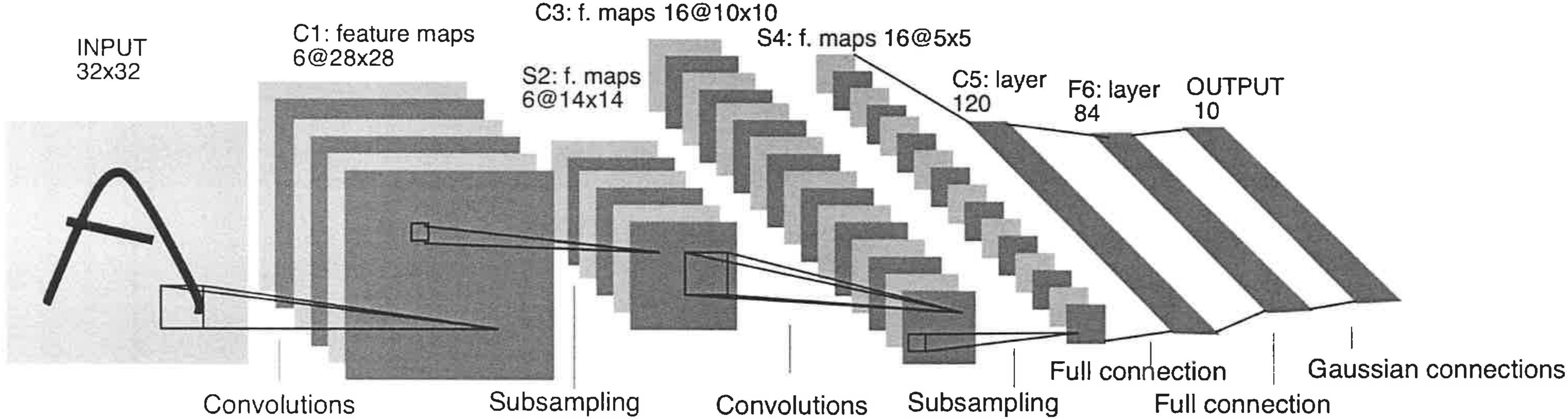


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

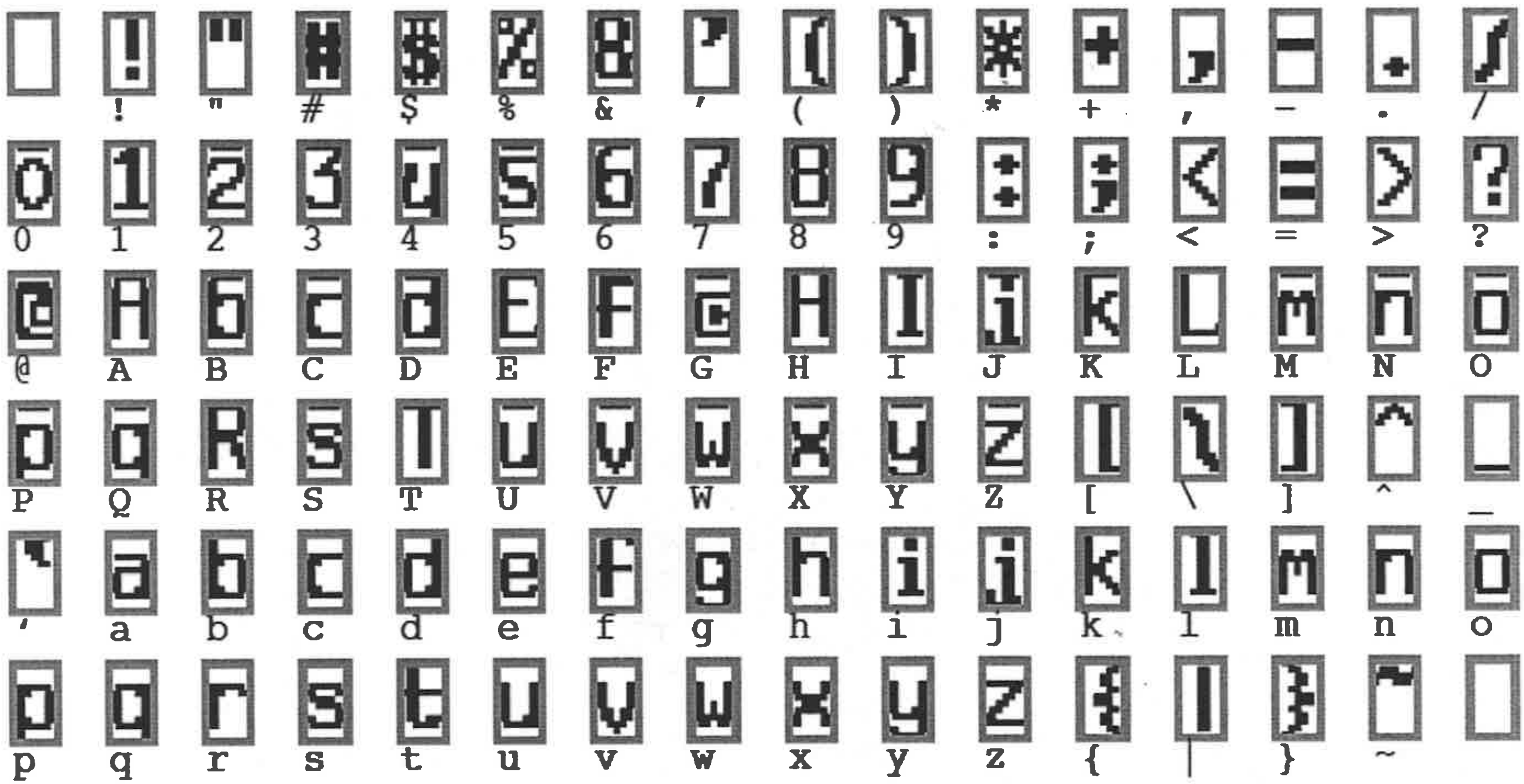
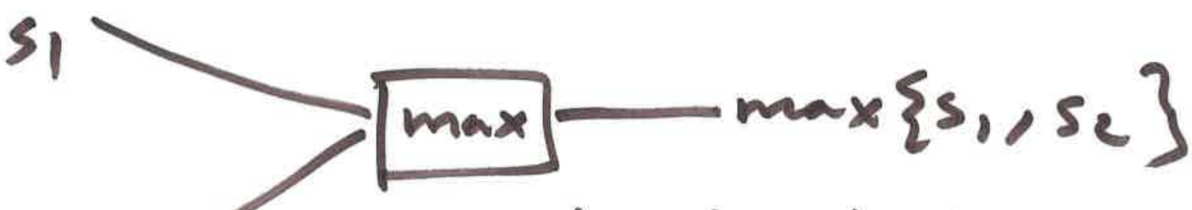


Fig. 3. Initial parameters of the output RBFs for recognizing the full ASCII set.

Training a CNN

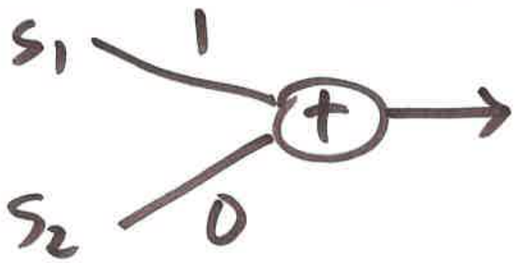
Is accomplished through the Backpropagation algorithm (similar to any multilayer NN).

- Should take care of issues like:
  * Weight sharing
    (Not a big problem; just accummulate gradients / add them up).
  * max. pooling:

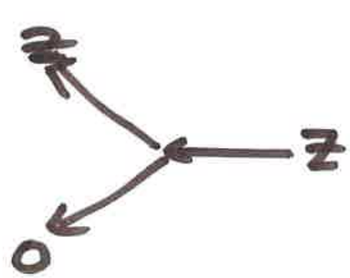


How does the backward graph look like?

Note that, if $s_1 \geqslant s_2$, then the above graph is equivalent to:



, so that the backward graph should be:



Similarly the case $s_1 \leq s_2$ is handled.