Mathematics of Image and Data Analysis
Math 5467

Convolutional Neural Networks and Classification

Instructor: Jeff Calder
Email: jcalder@umn.edu

http://www-users.math.umn.edu/~jwcalder/5467

Last time

- Intro to Neural Networks

Today

- Classification with neural networks.
- Convolutional neural networks.

Neural networks

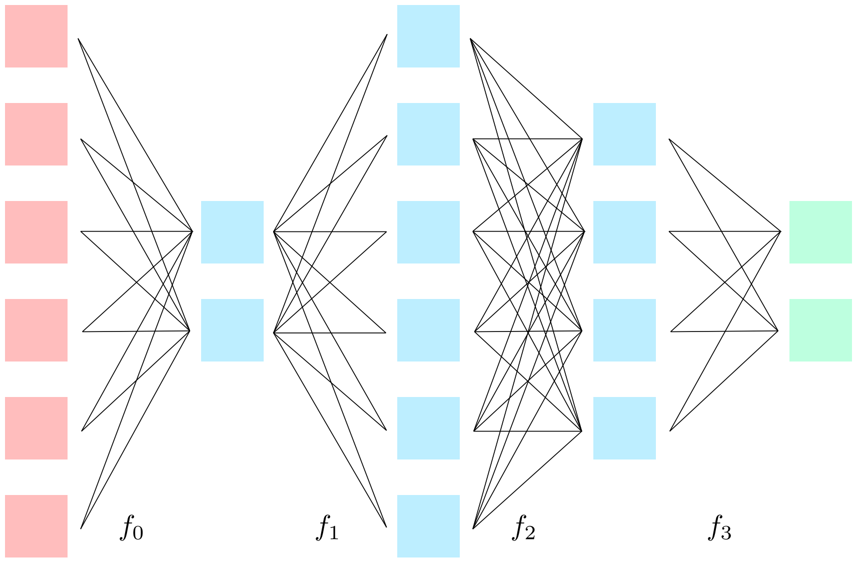


Figure 1: An example of a fully connected neural network with three hidden layers. The blue nodes are the hidden layers, the red is the input, and the green is the output. The hidden layers have width $n_1 = 2$, $n_2 = 6$, and $n_3 = 4$ and the number of input variables is $n_0 = 6$.

Neural network

In more compact notation, we can write a fully connected neural network with L layers recursively as

$$f_k = \sigma_k(W_k f_{k-1} + b_k), \quad k = 1, \dots, L, \tag{1}$$

where

- $f_0 \in \mathbb{R}^{n_0}$ is the input to the network,
- $f_k \in \mathbb{R}^{n_k}$ for $k = 1, \dots, L-1$ are the values of the network at the hidden layers,
- f_L is the output of the neural network,
- n_k is the number of hidden nodes in the k^{th} layer,
- The weights $W_k \in \mathbb{R}^{n_k \times n_{k-1}}$ and biases $b_k \in \mathbb{R}^{n_k}$ are the learnable parameters in the neural network.

Loss

The output of the neural network $f_L \in \mathbb{R}^{n_L}$ is typically fed into a loss function

$$\mathcal{L} : \mathbb{R}^{n_L} \to \mathbb{R},$$

which measures the performance of the network for the given learning task.

Typically the loss has the form

$$\mathcal{L}(W_1, b_1, \ldots, W_L, b_L) = \sum_{i=1}^{m} \ell(f_L(x_i), y_i), \tag{2}$$

where (x_i, y_i) for $i = 1, \ldots, m$ are the training data. Here, we write $f_L(x)$ to denote the value of the output of the network f_L given the input is $f_0 = x$.

Neural networks are trained by minimizing the loss function $\mathcal{L}$ with gradient descent.

Back Propagation

For notational simplicity, we will write

$$z_k = W_k f_{k-1} + b_k, \tag{3}$$

so that $f_k = \sigma_k(z_k)$. Let $\frac{\partial \mathcal{L}}{\partial z_k} \in \mathbb{R}^{n_k}$ denote the gradient of $\mathcal{L}$ with respect to z_k. We also let D_k be the diagonal $n_k \times n_k$ matrix with diagonal entries given by the vector $\sigma'_k(z_k)$. That is

$$D_k = \operatorname{diag}(\sigma'_k(z_k)).$$

Theorem 1 (Back propagation)**.** *For $k = 2, \ldots, L$ we have*

$$\frac{\partial \mathcal{L}}{\partial z_{k-1}} = D_{k-1} W_k^T \frac{\partial \mathcal{L}}{\partial z_k}, \tag{4}$$

$$\frac{\partial \mathcal{L}}{\partial W_k} = \frac{\partial \mathcal{L}}{\partial z_k} f_{k-1}^T, \quad \textit{and} \quad \frac{\partial \mathcal{L}}{\partial b_k} = \frac{\partial \mathcal{L}}{\partial z_k}. \tag{5}$$

ResNet

The traditional neural network architecture

$$f_k = \sigma_k(W_k f_{k-1} + b_k), \quad k = 1, \dots, L,$$

often yields worse performance for deeper networks compared to shallower ones.

The *Residual Neural Network (ResNet)* architecture is a recent development in deep learning that solves this problem by changing the architecture to

$$f_k = f_{k-1} + W_{k,1}\sigma_k(W_{k,2} f_{k-1} + b_k), \quad k = 1, \dots, L. \tag{6}$$

The idea is to have each layer learn the *residual* $f_k - f_{k-1}$, which allows the network to easily skip layers, by setting $f_k = f_{k-1}$. Thus, a deeper network with ResNet architecture should not perform worse than a shallower network.

Classification with neural networks

Recall that our label vectors are given as one hot vectors $e_1, \ldots, e_k$ in $\mathbb{R}^k$ (i.e., the standard basis vectors), where e_i represents the i^{th} class.

For a k-class classification problem, the output of the neural network $f_L(x)$ has k components, so $f_L(x) \in \mathbb{R}^k$, and the classification of x is taken to be the largest component of $f_L(x)$.

Let $z_1, \ldots, z_m \in \mathbb{R}^k$ denote the output vectors of the neural network aplied to m training points $x_1, x_2, \ldots, x_m$, so

$$z_i = f_L(x_i).$$

These outputs are fed through the *soft-max* function to convert them into probability vectors $p_1, \ldots, p_m \in \mathbb{R}^k$ given by

$$p_i(j) := \frac{e^{z_i(j)}}{\sum_{q=1}^k e^{z_i(q)}}.$$

log(

Loss

The loss used for classification is normally the negative log likelihood loss. Letting $y_1, \dots, y_m \in \mathbb{R}^k$ denote the one hot vectors representing the classes of the training data, the negative log likelihood loss is

$$\mathcal{L}(f_L) = -\sum_{i=1}^{m} y_i^T \log(p_i). \tag{7}$$

We claim that

$$\mathcal{L}(f_L) = -\sum_{i=1}^{m} \underline{f_L(x_i)^T} y_i + \sum_{i=1}^{m} \log\left(\sum_{j=1}^{k} e^{\overbrace{f_L(x_i)^T e_j}^{f_L(x_i)(j)}}\right).$$

$$p_i(j) := \frac{e^{z_i(j)}}{\sum_{q=1}^{k} e^{z_i(q)}}.$$

$$\log(p_i(j)) = \log\left(e^{z_i(j)}\right) - \log\left(\sum_{q=1}^{k} e^{z_i(q)}\right)$$

$$\rightarrow \boxed{= z_i(j) - \log\left(\sum_{q=1}^{k} e^{z_i(q)}\right).}$$

$$y_i = e_\ell \qquad = f_L(x_i)(j) - \underbrace{\log\left(\sum_{q=1}^{k} e^{f_L(x_i)(q)}\right)}_{\text{constant in } j}$$

$$-y_i^T \log(p_i) = -y_i^T f_L(x_i) + \log\left(\sum_{j=1}^{k} e^{f_L(x_i)^T e_j}\right)$$

Sum over $i: 1 \to m$.

Example: $$L = -\sum_{i=1}^{m} \frac{f_\ell(x_i)^T y_i}{\|f_\ell(x_i)\|}$$

works equally well.

Note in code: log_softmax

Toy data

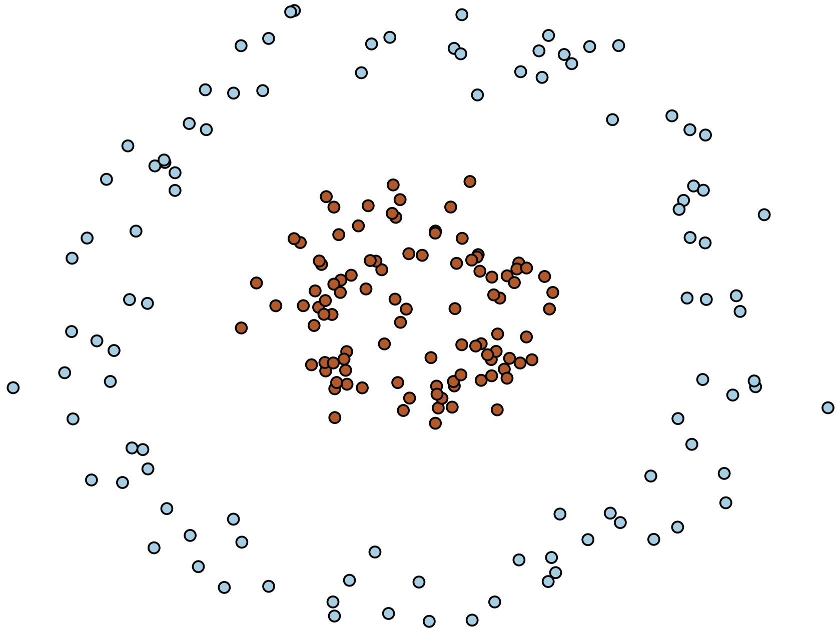
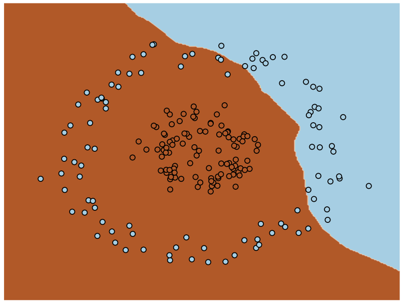
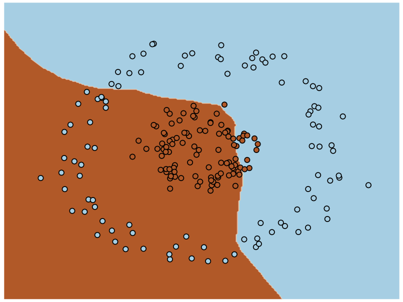
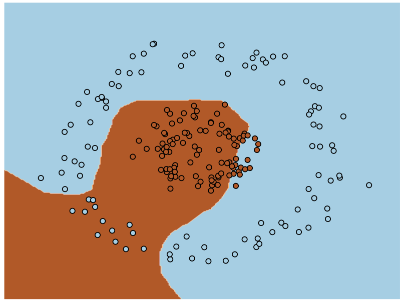


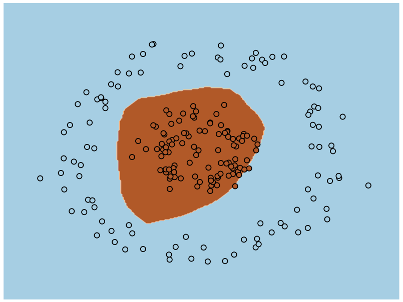
Figure 2: Synthetic data on rings consisting of two classes.

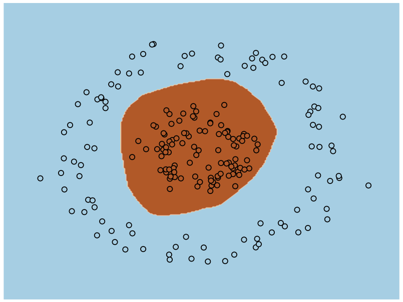
Training

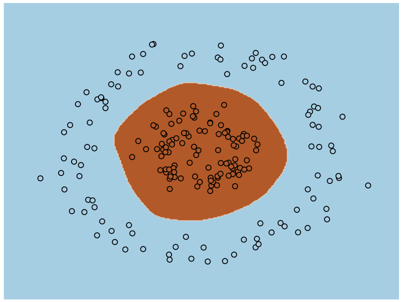

(a) Iteration 50


(b) Iteration 150


(c) Iteration 250


(d) Iteration 350


(e) Iteration 500


(f) Iteration 3000

MNIST

We consider classification of MNIST digits with 60000 training images and 10000 testing images.

- 2-Layer network with 10 hidden nodes: 93% testing accuracy
- 2-Layer network with 32 hidden nodes: 97% testing accuracy
- Even with good testing accuracy, there is overfitting:
    - 1-pixel shift of testing data: 87% accuracy
    - 2-pixel shift of testing data: 62% accuracy

MNIST

We consider classification of MNIST digits with 60000 training images and 10000 testing images.

- 2-Layer network with 10 hidden nodes: 93% testing accuracy
- 2-Layer network with 32 hidden nodes: 97% testing accuracy
- Even with good testing accuracy, there is overfitting:
  - 1-pixel shift of testing data: 87% accuracy
  - 2-pixel shift of testing data: 62% accuracy

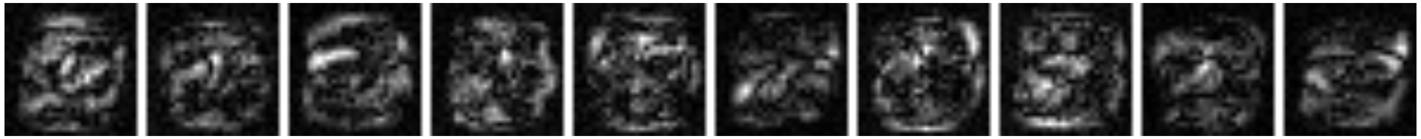
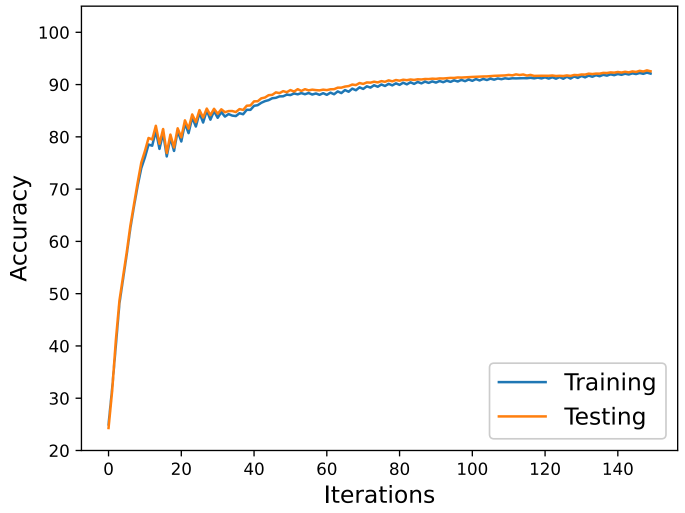
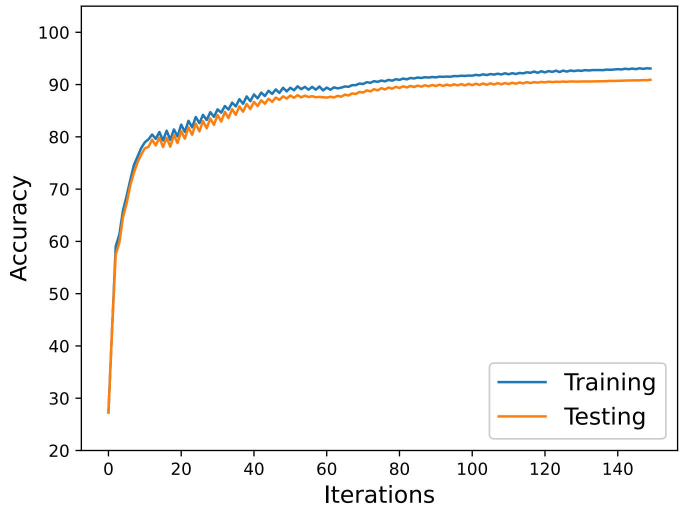


Figure 3: The 10 hidden nodes for MNIST classification.

Overfitting with less tranining data

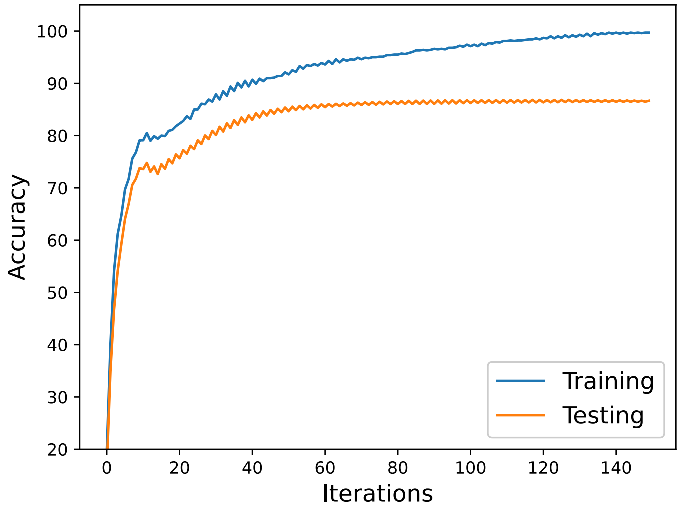


(a) 60000 Training images

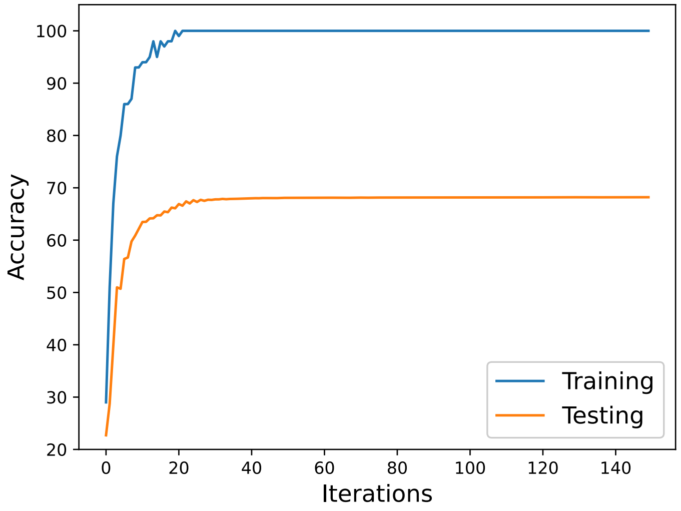


(b) 10000 Training images

Overfitting with less tranining data



(c) 1000 Training images



(d) 100 Training images

Classification with Torch (.ipynb)

Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) are one of the most powerful tools for image processing. They are special cases of fully connected neural networks that replace the linear function in a neuron with the convolution

$$(W * I)(i,j) = \sum_{p,q=-N}^{N} W(N+1+p, N+1+q)I(i+p, j+q).$$

Here I is a 2D image, W is a $(2N+1) \times (2N+1)$ matrix representing the filter, which is trainable. N is the width of the filter, which is normally small compared to the size of the image I.
Key points:

- **Translation Invariance**: The convolution applies the same filter W to *all* locations in the image, finding features no matter where the are within the image.
- **Locality:** The small size of filters restricts the features to be localized in the image.

Image convolution

Convolution is one of the most basic and important operations in image processing. It produces a new image that is *filtered* using a convolutional kernel. The filter below is called a *Sobel* filter that looks for vertical edges.

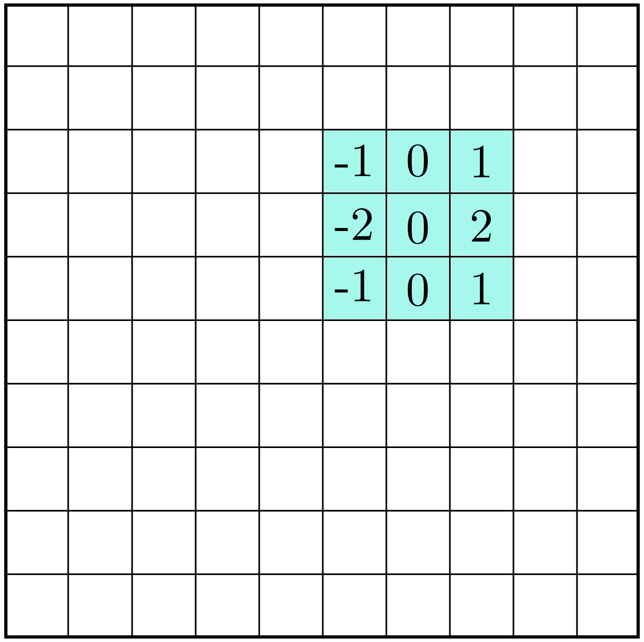


Image convolution

Convolution is one of the most basic and important operations in image processing. It produces a new image that is *filtered* using a convolutional kernel. The filter below is called a *Sobel* filter that looks for vertical edges.

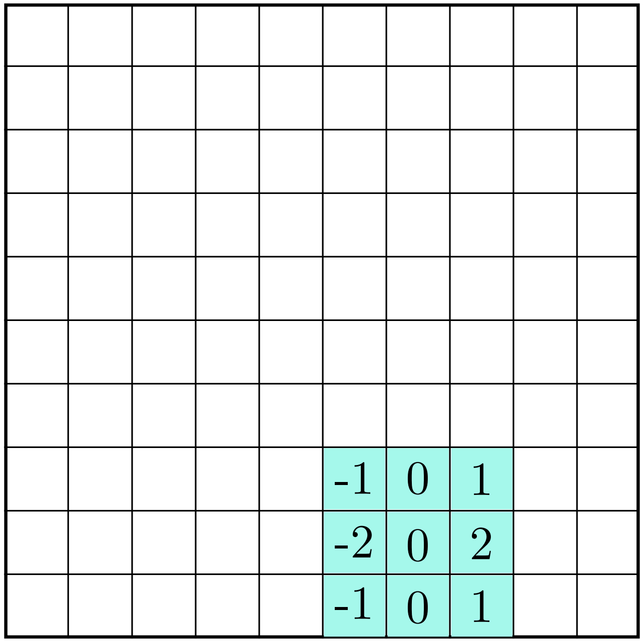
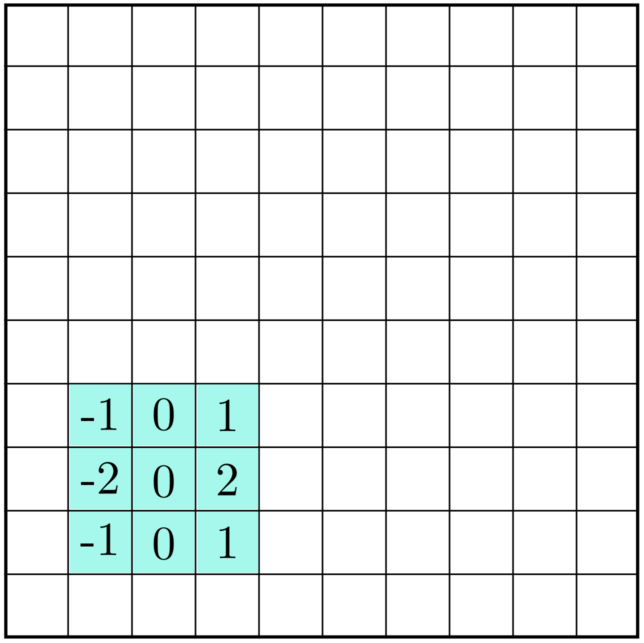


Image convolution

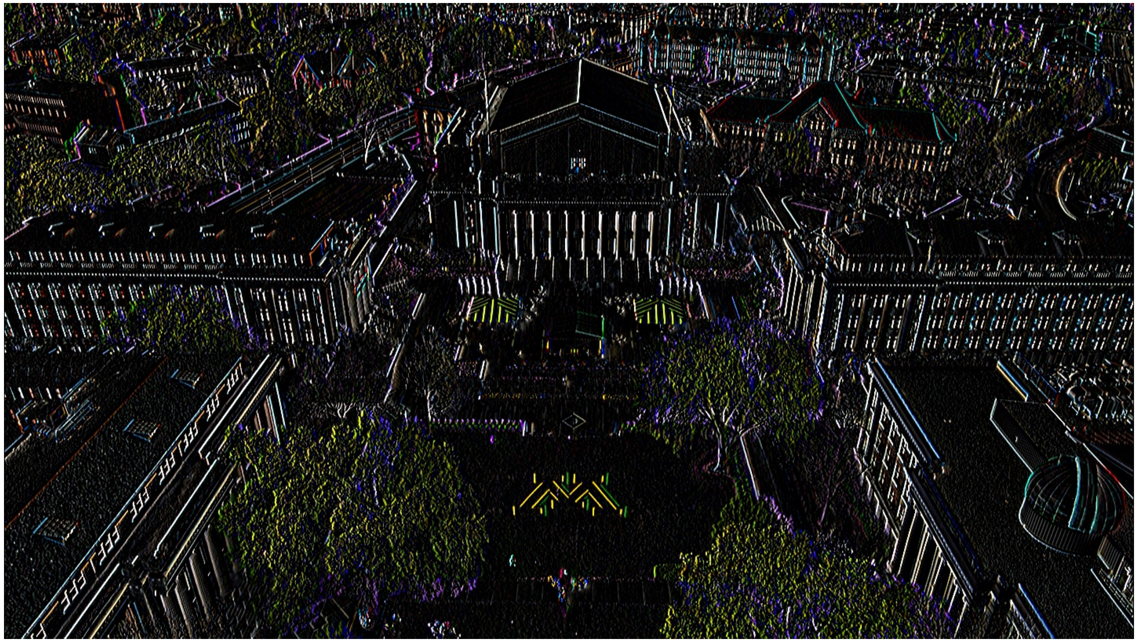
Convolution is one of the most basic and important operations in image processing. It produces a new image that is *filtered* using a convolutional kernel. The filter below is called a *Sobel* filter that looks for vertical edges.



Test image

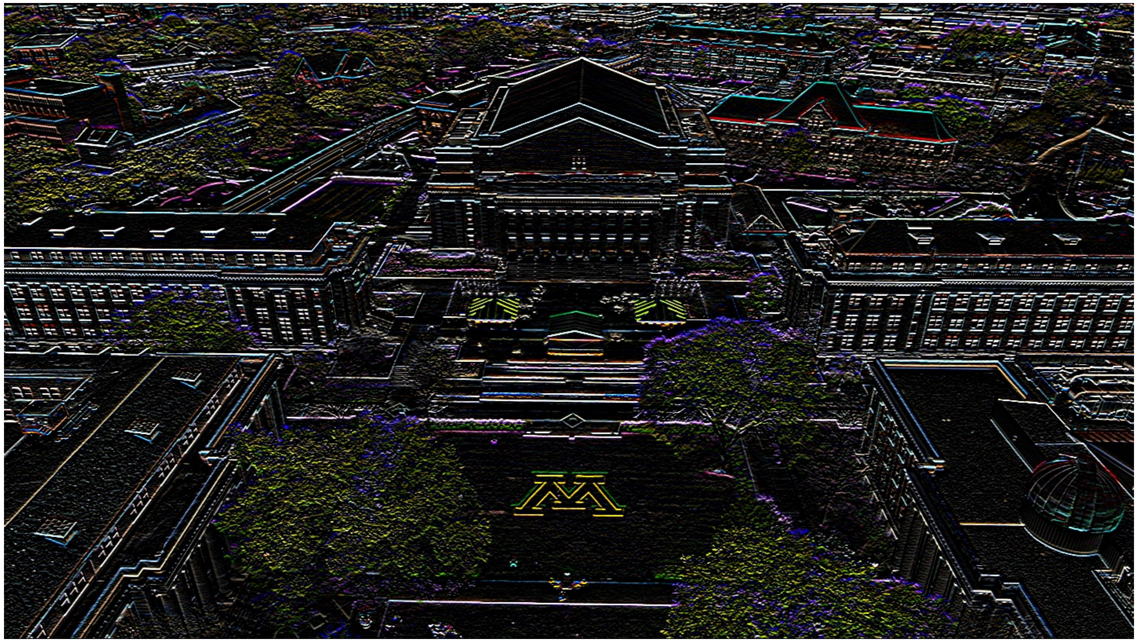
Sobel vertical

$$\text{Convolutional Kernel} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$



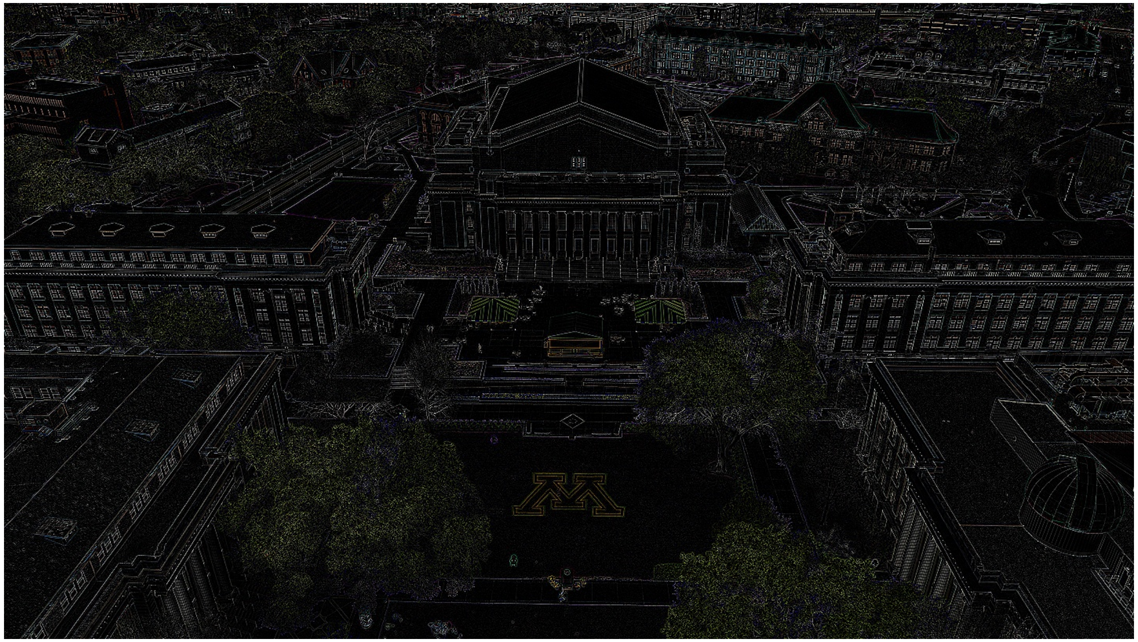
Sobel horizontal

$$\text{Convolutional Kernel} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$



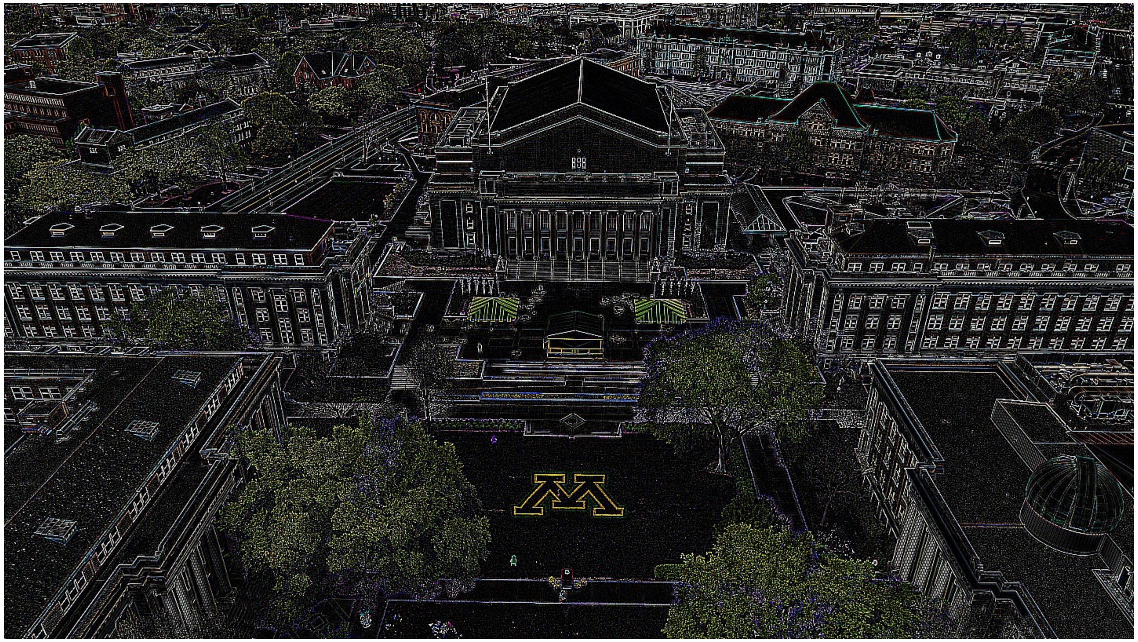
Laplacian

$$\text{Convolutional Kernel} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



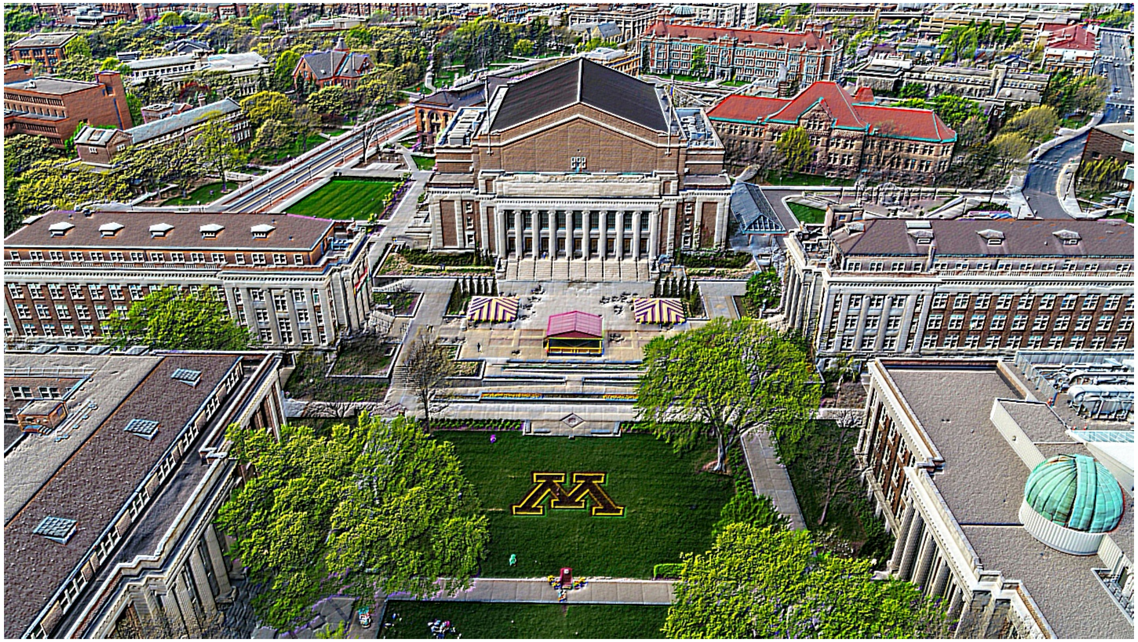
Outline

$$\text{Convolutional Kernel} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



Emboss

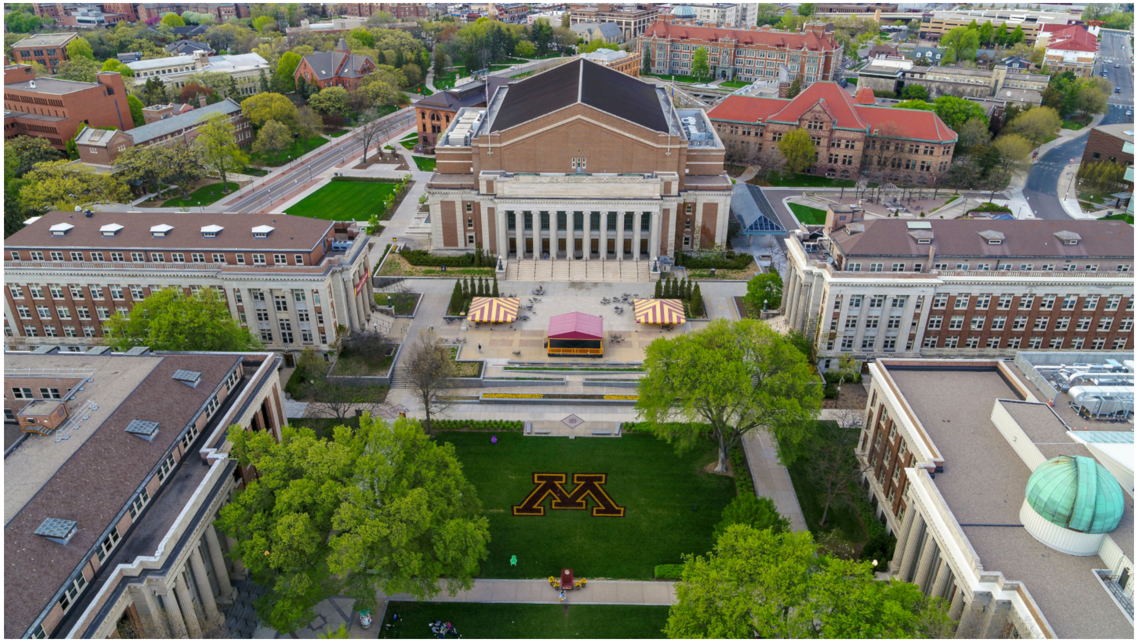
$$\text{Convolutional Kernel} = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$



Sharpen

$$\text{Convolutional Kernel} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Original image



Typical CNN architecture

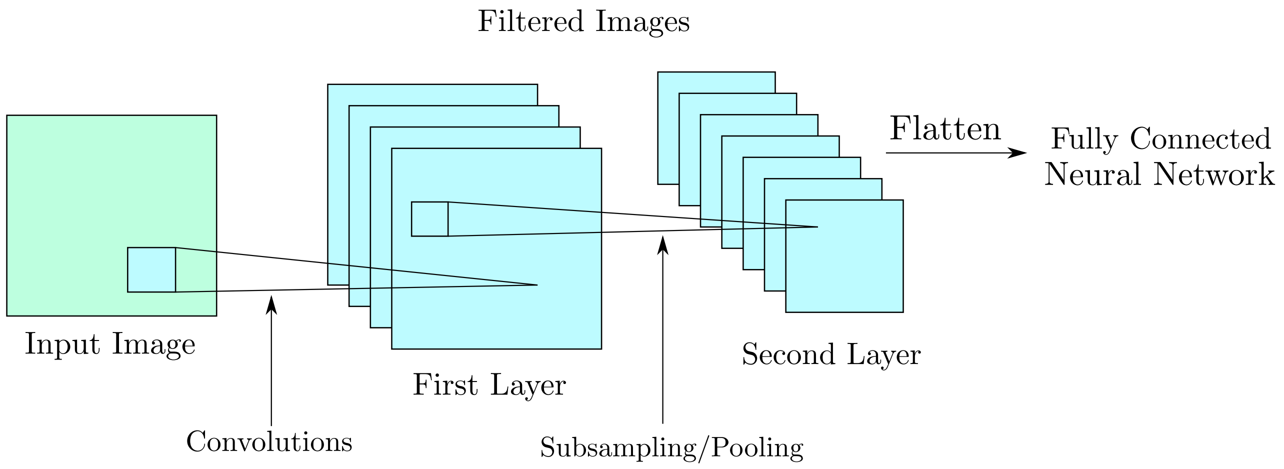
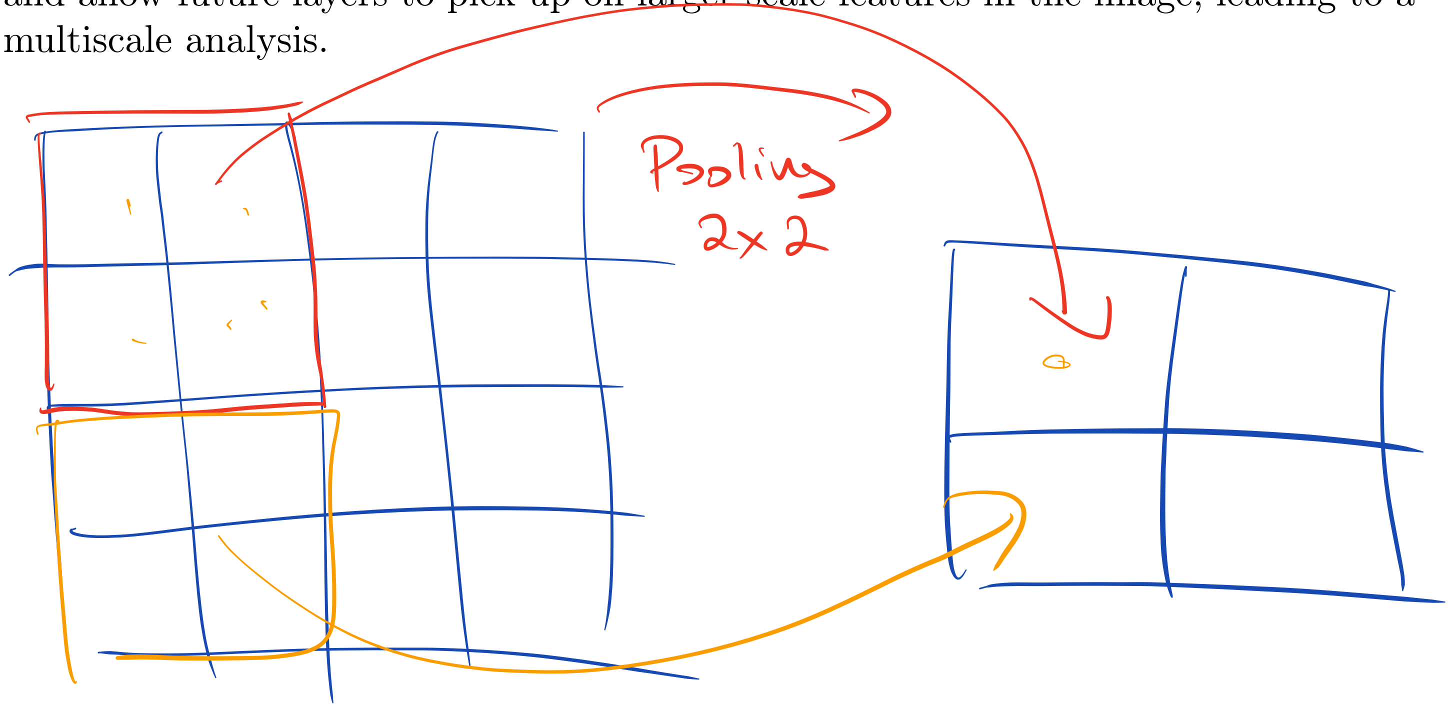


Figure 4: An example of a typical Convolutional Neural Network (CNN) architecture.

Pooling

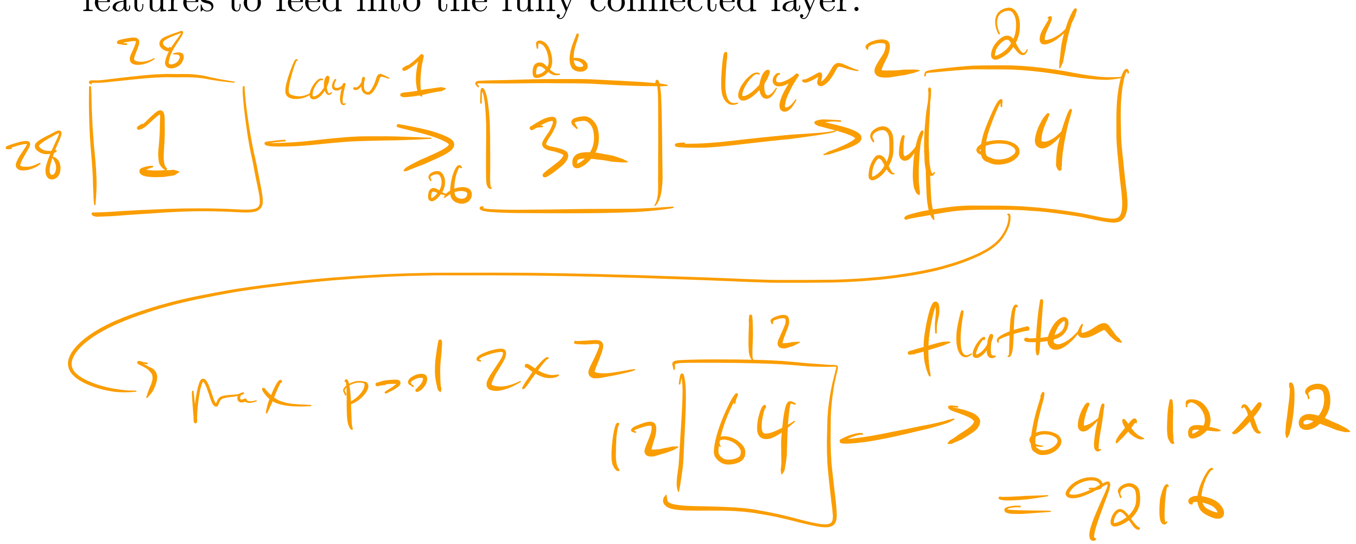
Pooling is a form of subsampling that introduces translation invariance into CNNs, and allow future layers to pick up on larger scale features in the image, leading to a multiscale analysis.



Example on MNIST

We ran an experiment testing a simple convolutional neural network for classification of MNIST digits.

- 4 layer neural network 3x3 conv. filters
  - First 2 layers are convolutional with 32 and 64 channels
  - Final 2 layers are fully connected with 128 hidden nodes. max pooling 2x2
- Output of the convolutional layers is flattened into a length 9216 array of features to feed into the fully connected layer.



MNIST Accuracy

- Accuracy is 99.04% after 14 epochs.
- 98.07% after single pixel shift
- 92.06% after two-pixel shift
- 75% after three-pixel shift

The translation invariance is better than with fully connected networks due to the pooling and translation invariance of convolutional. To get better translation invariance we can introduce more pooling into the network.

Convolutional filters

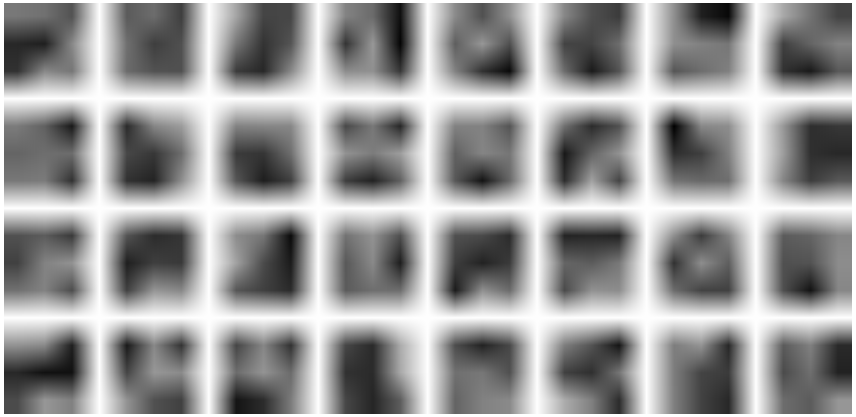


Figure 5: The 32 3×3 filters from the first layer of the convolutional neural network for classifying MNIST digits.

First layer channels

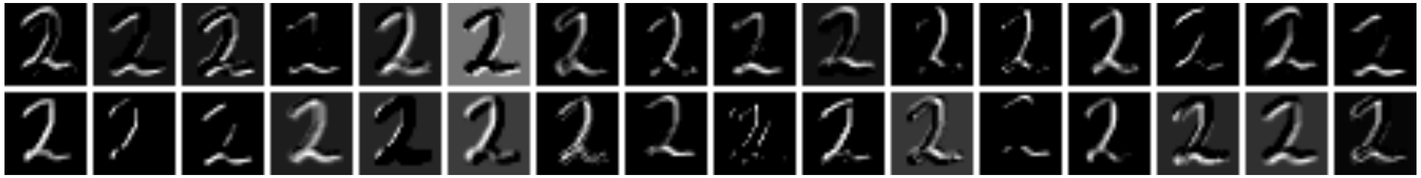


Figure 6: The 32 channels of output from the first convolutional layer acting on an image of a 2.

Second layer channels

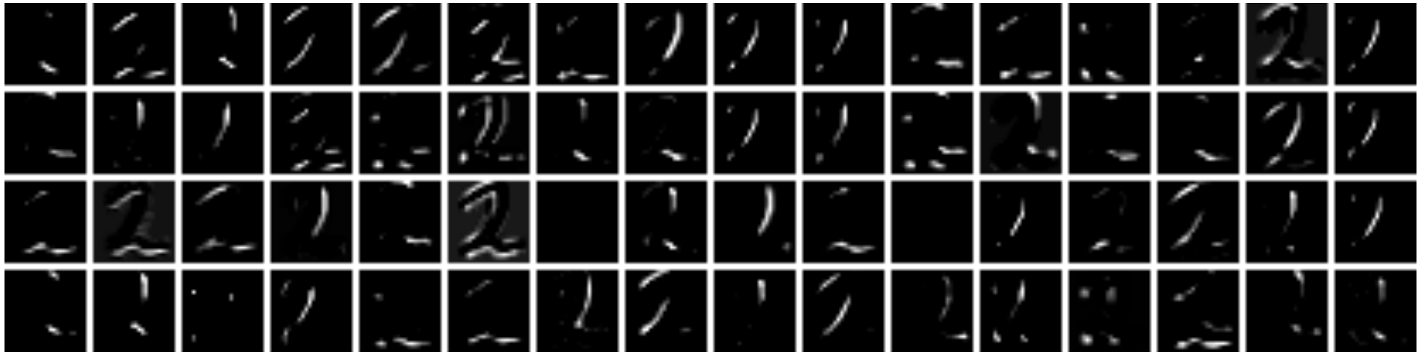
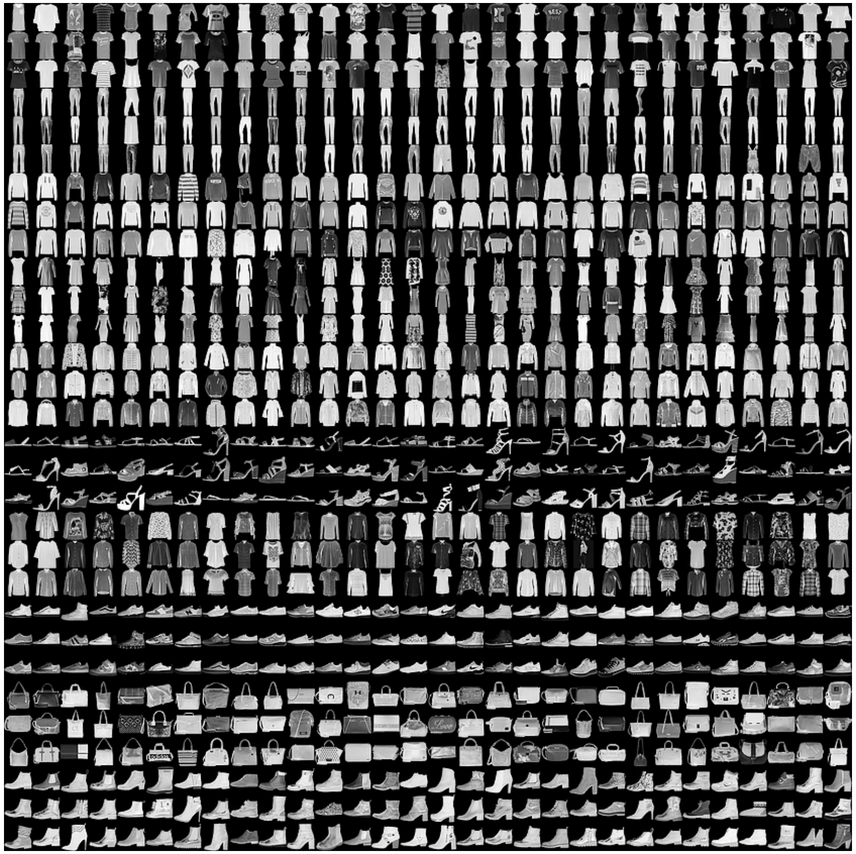


Figure 7: The 64 channels of output from the second convolutional layer acting on an image of a 2. Notice the channels appear to be detecting edges in various directions.

FashionMNIST



FashionMNIST accuracy

- Accuracy is 92.55% after 14 epochs.
- 89.73% after single pixel shift
- 75.94% after two-pixel shift
- 47.38% after three-pixel shift

We could choose a larger network with more channels and hidden layers to achieve better results (around 99%) for FashionMNIST.

Convolutional Neural Networks (.ipynb)