CS 615 – Deep Learning

Convolutional Neural Networks

Work adapted from that of Cameron Graybill

Objectives

- Convolutional Neural Networks

Issues with Generic Deep Networks

- There are some problems with an arbitrarily deep ANN:
  - Late layers (ones near the output) learn quickly/well and therefore early layers (near the input stage) have little error and therefor become relatively useless.
- Therefore, in practice generic deep ANNs beyond depth two aren’t successful.
- Which leads us to variations of these…..
- One that has seen success in the area of image recognition is *convolutional neural networks (CNNs)*.

Convolution

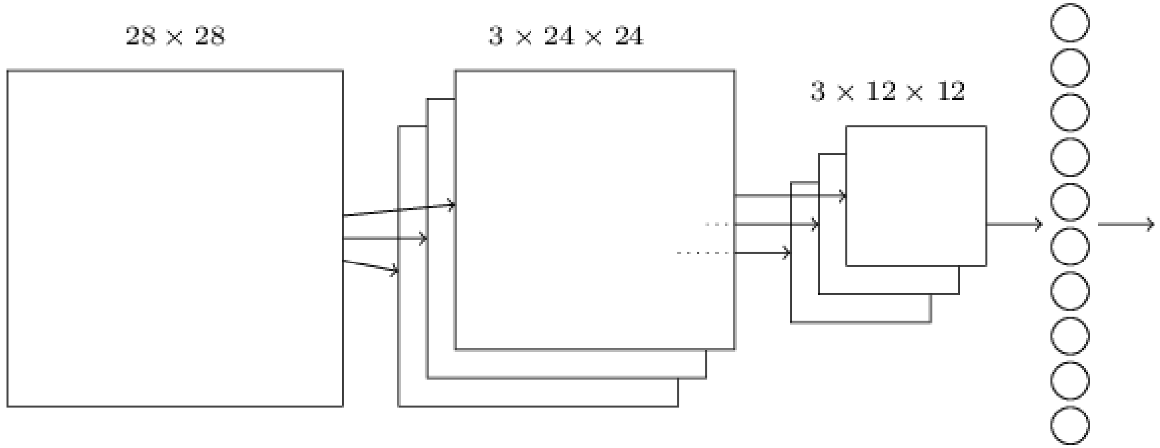
- *Convolution* is an operation used often in image processing and computer vision.
- Given a filter/kernel *K* and a sub-image, convolution gives a *filter response*.
  - Basically saying how well the sub-image matches the filter/kernel.
- If we know what we're looking for, we can *design* filters and apply them to an image to get filter responses to be used for machine learning.
- This is what is often done in traditional machine learning.
- But what if we don't know the kernels?
- Is there a way to discover/learn useful ones?

Convolution Neural Networks

- Convolutional Neural Networks (CNNs) attempt to do exactly that!
- They have become popular and successful in images and audio classification where traditionally a kernel/filter was applied to the raw data to extract features.
- This is because kernels take spatial relationships into account.
  - What is important in image and audio classification.

CNNs

- CNNs typically have
  - One or more convolution layer
  - A final fully connected shallow ANN.
- The convolution layer contains several parts
  - Feature Map Extraction
  - Pooling
- Let’s look at each of these



Convolution

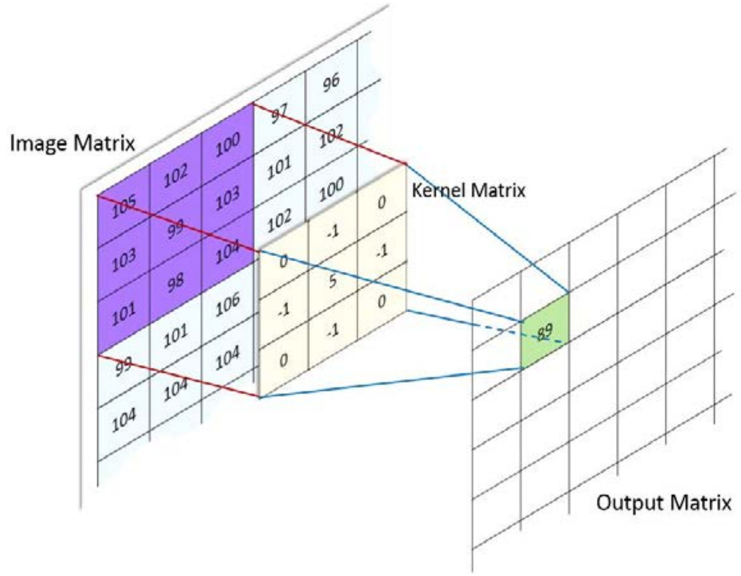
- Again, convolution is a mathematical operation that takes a filter/function, and applies it to some area to produce a single value at the center of that area.
- It's almost identical to cross-correlation
  - Flipping the filter and then doing cross-correlation is equivalent to doing convolution.
  - The pro of using convolution instead of cross-correlation is that it exhibits the commutative property.

Convolution

- We use the $*$ operator to denote convolution.
- To formalize the convolution operator, given an $M \times M$ filter, K, and an input matrix X, we can compute the convolution at location (a, b) as:

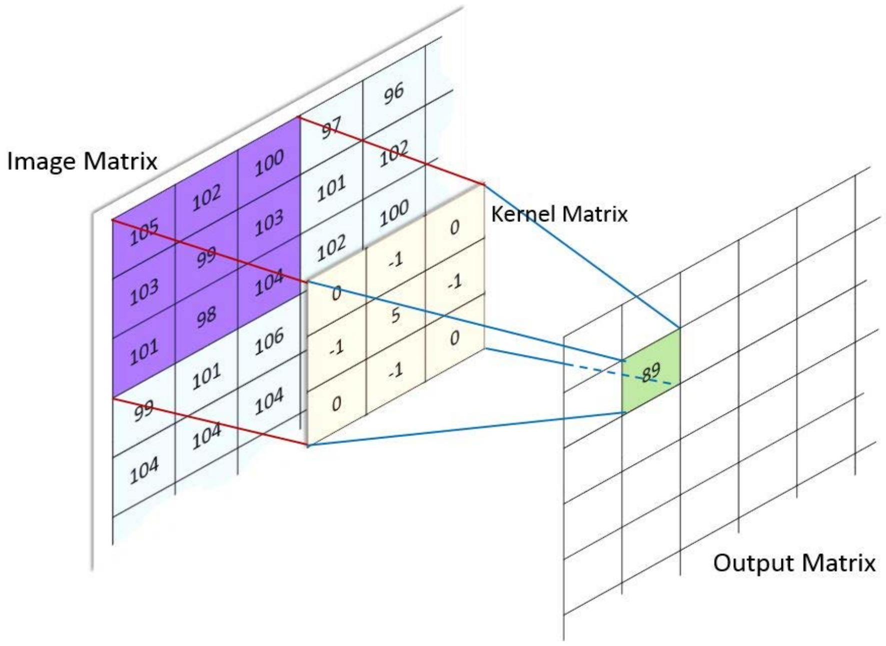
$$F_{ab} = (X_{ab} * K) = \sum_{i=-\frac{M}{2}}^{\frac{M}{2}} \sum_{j=-\frac{M}{2}}^{\frac{M}{2}} X_{(a-\mathrm{i},b-\mathrm{j})} K_{ij}$$

- If we apply convolution to several locations in the image, what we get back is a new matrix, that is the filter response at each location, and which we call the **Feature Map**



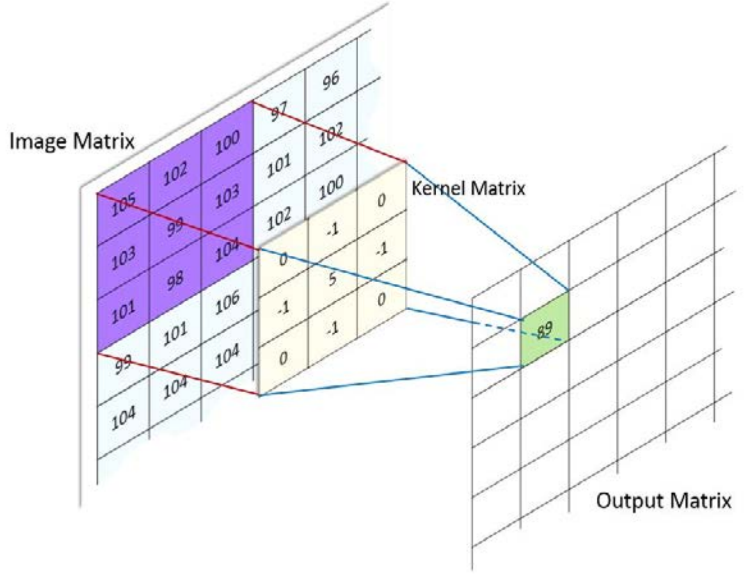
Feature Map Size

- Valid convolution can only be computed where there are $M \times M$ pixels to process.
- This results in a **smaller** image than the original one.
- If our image is $H \times W$, then our feature map will be $(H - M + 1) \times (W - M + 1)$



Convolution Example

- Let $X = \begin{bmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \\ 41 & 42 & 43 & 44 \end{bmatrix}, K = \begin{bmatrix} 0.11 & 0.12 & 0.13 \\ 0.21 & 0.22 & 0.23 \\ 0.31 & 0.32 & 0.33 \end{bmatrix}$
- $X * K = \begin{bmatrix} 37.5 & 39.48 \\ 57.3 & 59.28 \end{bmatrix}$

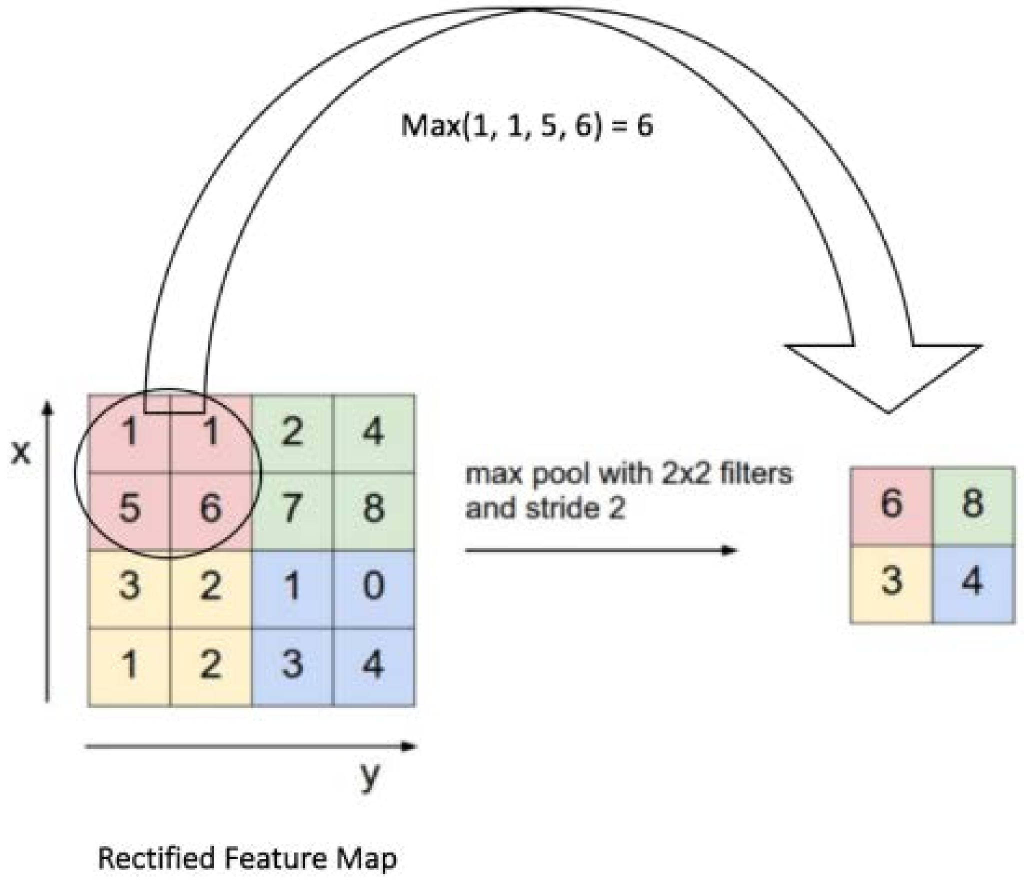


Pooling

- The next part of a CNN layer is pooling.
- Pooling is essentially down-sampling: we're taking our feature map and making a new, smaller map by "summarizing" the original one.
  - This provides some invariance to translation
- Pooling is also done by moving around a $Q \times Q$ window and extracting a value from each locations
- However, it is common not to move/slide the window by just one pixel, but instead to allow for this to be a user-defined hyperparameter, called the ***stride***.

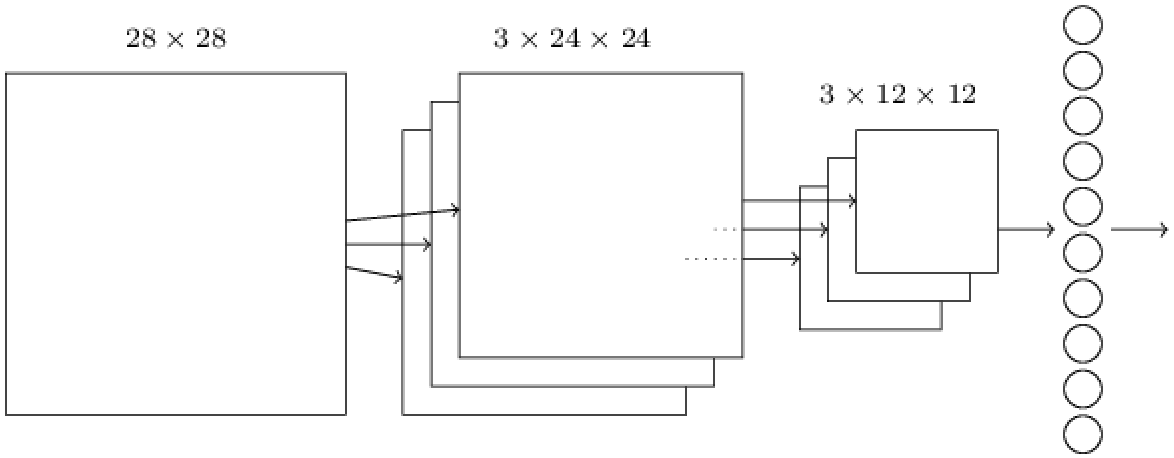
Pooling

- Common pooling techniques include:
  - Max Pooling – Select the maximum value in the square
  - L2 Pooling – Compute the square root of the sum of the values in the square.
- If our feature map is $D \times E$ and we have a stride of S, then output from the pooling process will be a $\left\lfloor \frac{D}{S} \right\rfloor \times \left\lfloor \frac{E}{S} \right\rfloor$matrix



CNNs

- Now that our Convolution Layer is made, the output of the pooling process is flattened and becomes the input of a traditional ML system.
  - Potentially adding in bias features as desired.

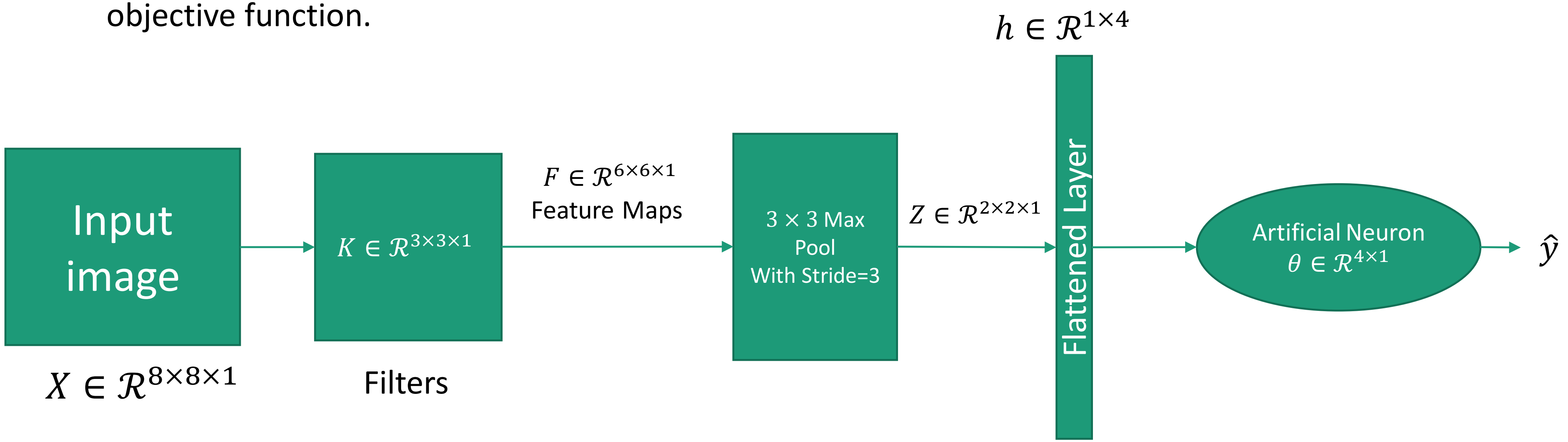


Training

- As with our ANNs, training a CNN requires *forward-backwards propagation.*
- But now our set of parameters to learn are:
  - Any parameters of our "traditional" ML part of our system.
  - The weights of each kernel in the convolution layer(s).
- And once again, given a set of parameters, the forward propagation is relatively simple:
  - Create the feature maps using the current kernels.
  - Create the pooled maps from the feature maps.
  - Feed these into our traditional ML algorithm.
- Of course the tough part is learning the weights.

Example CNN Architecture

- Let's start off with a relatively simple system/example:
  - X is input image of size 8×8
  - K is a single 3×3 kernel
  - Our pooling process using max-pooling on a 3×3 region with stride of 3
  - Our traditional ML layer is an artificial neuron with a logistic activation function and a log likelihood objective function.



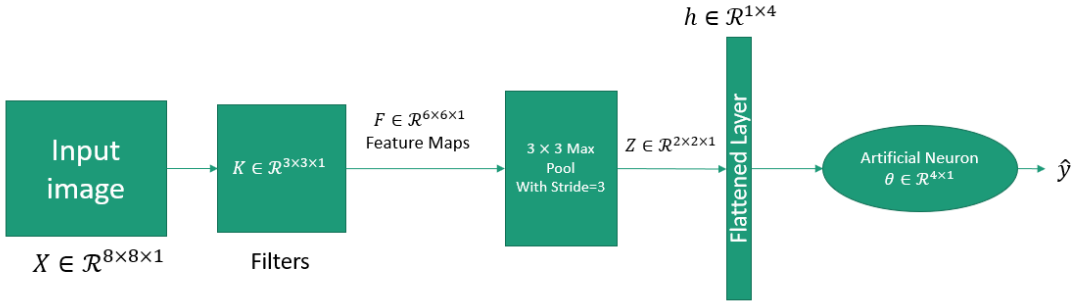
Full Example

- In an attempt to wrap our heads around all of this, let's do a numerical example.
- Let our grayscale image X , whose target class (binary classification) is $y = 0$ be:

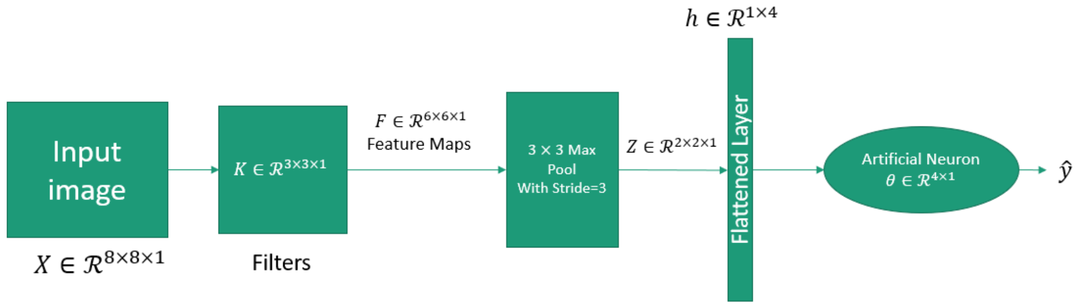
$$X = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \\ 25 & 26 & 27 & 28 & 29 & 30 & 31 & 32 \\ 33 & 34 & 35 & 36 & 37 & 38 & 39 & 40 \\ 41 & 42 & 43 & 44 & 45 & 46 & 47 & 48 \\ 49 & 50 & 51 & 52 & 53 & 54 & 55 & 56 \\ 57 & 58 & 59 & 60 & 61 & 62 & 63 & 64 \end{bmatrix}$$

Full Example

- In addition our kernel will be initialized as $K = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$
- And the output layer weights are initialized as $\theta = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$



Full Example

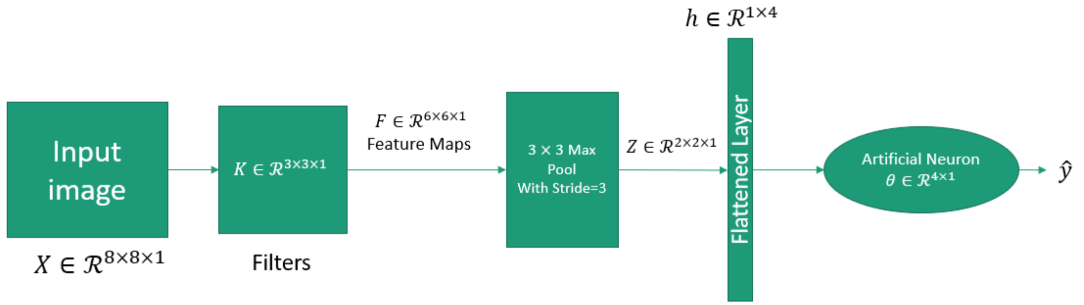


$$X = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \\ 25 & 26 & 27 & 28 & 29 & 30 & 31 & 32 \\ 33 & 34 & 35 & 36 & 37 & 38 & 39 & 40 \\ 41 & 42 & 43 & 44 & 45 & 46 & 47 & 48 \\ 49 & 50 & 51 & 52 & 53 & 54 & 55 & 56 \\ 57 & 58 & 59 & 60 & 61 & 62 & 63 & 64 \end{bmatrix}, y = 0, K = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \theta = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

- First let's forward propagate!
- Here's the convolution

$$F = X * K = \begin{bmatrix} 300 & 345 & 390 & 435 & 480 & 525 \\ 660 & 705 & 750 & 795 & 840 & 885 \\ 1020 & 1065 & 1110 & 1155 & 1200 & 1245 \\ 1380 & 1425 & 1470 & 1515 & 1560 & 1605 \\ 1740 & 1785 & 1830 & 1875 & 1920 & 1965 \\ 2100 & 2145 & 2190 & 2235 & 2280 & 2325 \end{bmatrix}$$

Full Example

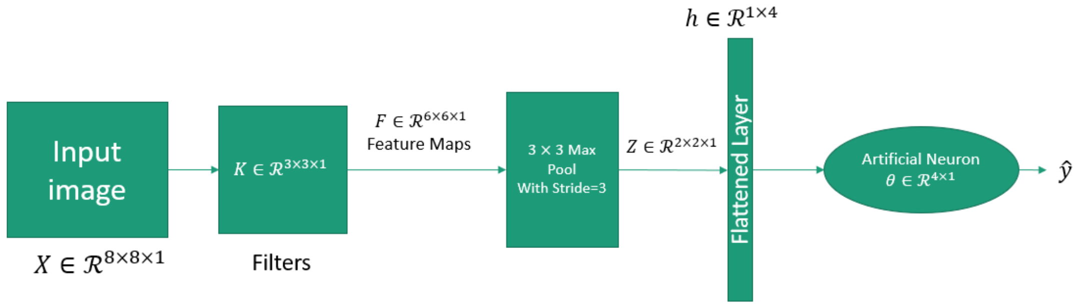


$$\theta = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, F = \begin{bmatrix} 300 & 345 & 390 & 435 & 480 & 525 \\ 660 & 705 & 750 & 795 & 840 & 885 \\ 1020 & 1065 & 1110 & 1155 & 1200 & 1245 \\ 1380 & 1425 & 1470 & 1515 & 1560 & 1605 \\ 1740 & 1785 & 1830 & 1875 & 1920 & 1965 \\ 2100 & 2145 & 2190 & 2235 & 2280 & 2325 \end{bmatrix}$$

- And now let’s 3x3 max-pool with stride of 3.
- This results in:

$$Z = \begin{bmatrix} 1110 & 1245 \\ 2190 & 2325 \end{bmatrix}$$

Full Example



$$\theta = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, F = \begin{bmatrix} 300 & 345 & 390 & 435 & 480 & 525 \\ 660 & 705 & 750 & 795 & 840 & 885 \\ 1020 & 1065 & 1110 & 1155 & 1200 & 1245 \\ 1380 & 1425 & 1470 & 1515 & 1560 & 1605 \\ 1740 & 1785 & 1830 & 1875 & 1920 & 1965 \\ 2100 & 2145 & 2190 & 2235 & 2280 & 2325 \end{bmatrix}, Z = \begin{bmatrix} 1110 & 1245 \\ 2190 & 2325 \end{bmatrix}$$

- Then we'll flatted this to be a single row vector. Since Matlab uses column major indexing, this becomes:

$$h = [1110 \quad 2190 \quad 1245 \quad 2325]$$

- To which we apply θ

$$net = h \cdot \theta = 18525$$

- Then apply activation function $g(z)$ (for which we'll use the logistic activation function):

$$\hat{y} = \frac{1}{1 + e^{-18525}} \approx 1$$

Gradient of Artificial Neuron Layer

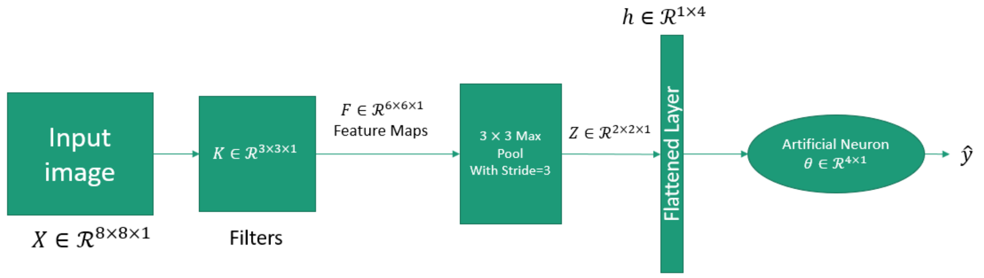
- Ok, now it's time to update our parameters based on the forward propagation.
- As a reminder, at the end of our CNN we have an artificial neuron with a logistic activation function and log likelihood objective function for binary classification.
- Given this configuration, from our previous work, the gradient with respect to our last set of parameters is:

$$\frac{\partial J}{\partial \theta_s} = \frac{\partial J}{\partial g(h|\theta)} \cdot \frac{\partial g(h|\theta)}{\partial \theta_s} = \left(\frac{y-\hat{y}}{\hat{y}(1-\hat{y})}\right) \cdot \big(h_s \hat{y}(1-\hat{y})\big) = (y-\hat{y})h_s$$

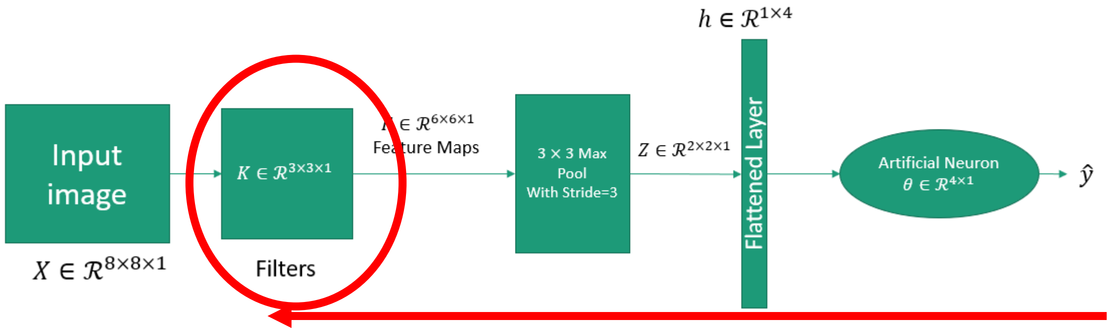
$$\frac{\partial J}{\partial \theta} = h^T(y-\hat{y})$$

- For our example this is:

$$\frac{\partial J}{\partial \theta} = \begin{bmatrix} -1110 \\ -2190 \\ -1245 \\ -2325 \end{bmatrix}$$



Gradient of Kernel



- The next set of parameters we need to update are those of the convolution kernel, K
- So the question is, what is:

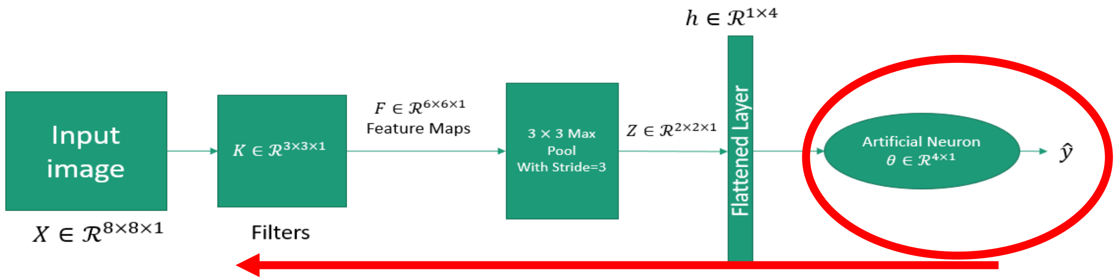
$$\frac{\partial J}{\partial K_{ij}}?$$

- Well let's try to chain rule this guy!

$$\frac{\partial J}{\partial K_{ij}} = \frac{\partial J}{\partial g(h|\theta)} \cdot \frac{\partial g(h|\theta)}{\partial h} \cdot \frac{\partial h}{\partial Z} \cdot \frac{\partial Z}{\partial F} \cdot \frac{\partial F}{\partial K_{ij}}$$

- Ugh ☹!

Gradient of Kernel



$$\frac{\partial J}{\partial K_{ij}} = \frac{\partial J}{\partial g(h|\theta)} \cdot \frac{\partial g(h|\theta)}{\partial h} \cdot \frac{\partial h}{\partial Z} \cdot \frac{\partial Z}{\partial F} \cdot \frac{\partial F}{\partial K_{ij}}$$

- We already have $\frac{\partial J}{\partial g(h|\theta)}$ from our previous layer:

$$\frac{\partial J}{\partial g(h|\theta)} = \left(\frac{y-\hat{y}}{\hat{y}(1-\hat{y})}\right)$$

- What is $\frac{\partial g(h|\theta)}{\partial h}$?

$$\frac{\partial g(h|\theta)}{\partial h} = \theta^T \hat{y}(1-\hat{y})$$

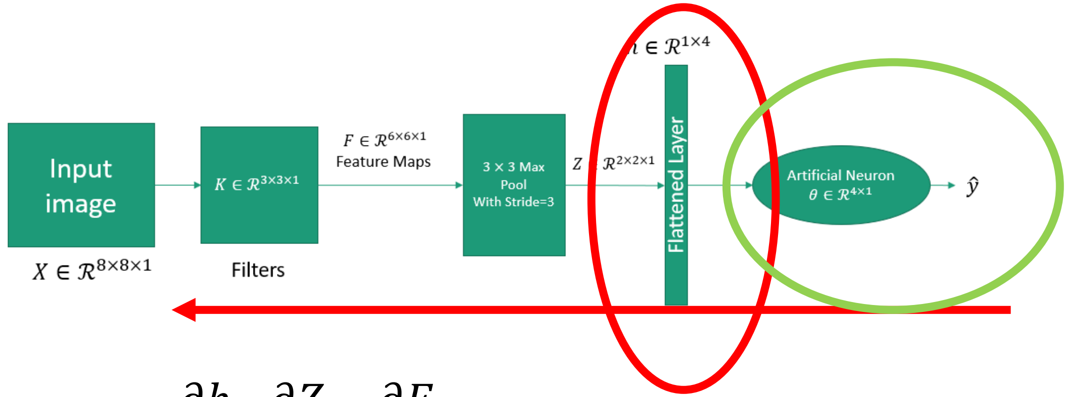
- So:

$$\frac{\partial J}{\partial K_{ij}} = (y-\hat{y})\theta^T \cdot \frac{\partial h}{\partial Z} \cdot \frac{\partial Z}{\partial F} \cdot \frac{\partial F}{\partial K_{ij}}$$

- For the example:

$$\frac{\partial J}{\partial K_{ij}} = (-1)\begin{bmatrix}1\\2\\3\\4\end{bmatrix}^T \cdot \frac{\partial h}{\partial Z} \cdot \frac{\partial Z}{\partial F} \cdot \frac{\partial F}{\partial K_{ij}}$$

Gradient of Kernel



$$\frac{\partial J}{\partial K_{ij}} = (y-\hat{y})\theta^T \cdot \frac{\partial h}{\partial Z} \cdot \frac{\partial Z}{\partial F} \cdot \frac{\partial F}{\partial K_{ij}}$$

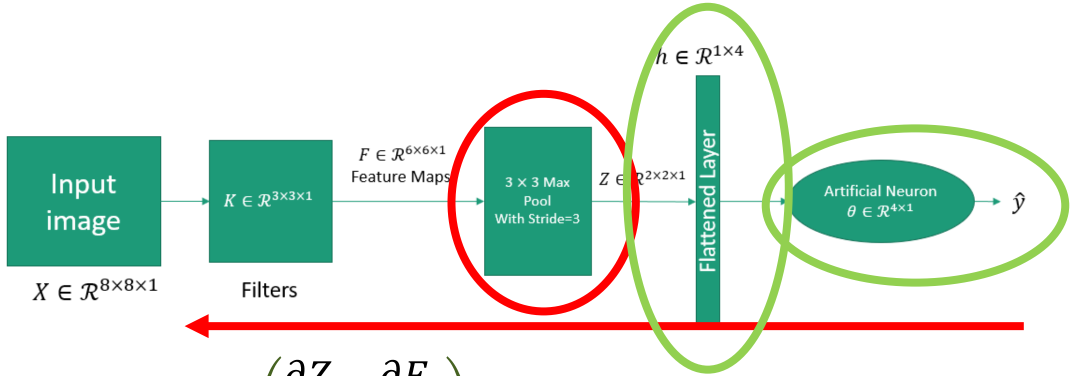
- h is just a flattened version of Z, so $\frac{\partial h}{\partial Z}$ is just a reshaping function

$$\frac{\partial J}{\partial K_{ij}} = (y-\hat{y})\theta^T \cdot reshape\left(\frac{\partial Z}{\partial F} \cdot \frac{\partial F}{\partial K_{ij}}\right)$$

- For our example:

$$\frac{\partial J}{\partial K_{ij}} = (-1) \cdot \begin{bmatrix}1\\2\\3\\4\end{bmatrix}^T \cdot reshape\left(\frac{\partial Z}{\partial F} \cdot \frac{\partial F}{\partial K_{ij}}\right)$$

Gradient of Kernel



$$\frac{\partial J}{\partial K_{ij}} = (y-\hat{y})\theta^T \cdot reshape\left(\frac{\partial Z}{\partial F}\cdot\frac{\partial F}{\partial K_{ij}}\right)$$

- Since Z is created by max-pooling, we can think of it as selecting elements from $\frac{\partial F}{\partial K_{ij}}$.
- Let's call this gradient *select*

$$\frac{\partial J}{\partial K_{ij}} = (y-\hat{y})\cdot\theta^T \cdot reshape\left(select\left(\frac{\partial F}{\partial K_{ij}}\right)\right)$$

- Where were the max locations selected from?
- That leaves us with $\frac{\partial F}{\partial K_{ij}}$

$$F = \begin{bmatrix} 300 & 345 & 390 & 435 & 480 & 525 \\ 660 & 705 & 750 & 795 & 840 & 885 \\ 1020 & 1065 & 1110 & 1155 & 1200 & 1245 \\ 1380 & 1425 & 1470 & 1515 & 1560 & 1605 \\ 1740 & 1785 & 1830 & 1875 & 1920 & 1965 \\ 2100 & 2145 & 2190 & 2235 & 2280 & 2325 \end{bmatrix}$$

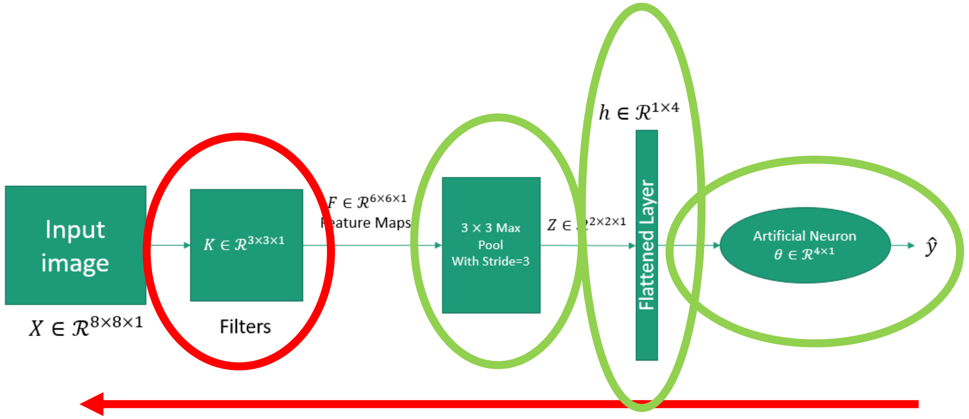
Gradient of Convolution Function

- Recall the convolution function

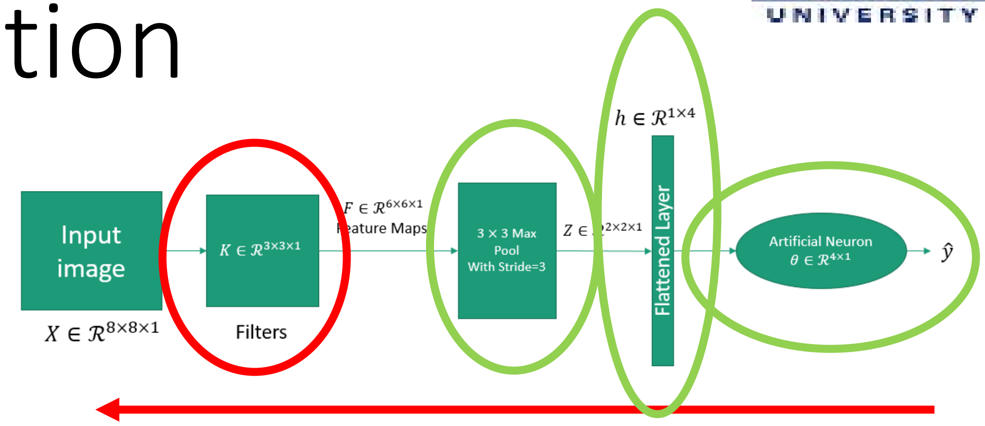
$$F_{ab} = \sum_{i=-\frac{M}{2}}^{\frac{M}{2}} \sum_{j=-\frac{M}{2}}^{\frac{M}{2}} X_{a-\mathrm{i},b-\mathrm{j}} K_{ij}$$

- So let's start with, what is $\frac{\partial F_{ab}}{\partial K_{ij}}$?

$$\frac{\partial F_{ab}}{\partial K_{ij}} = X_{a-\mathrm{i},b-\mathrm{j}}$$



Gradient of Convolution Function



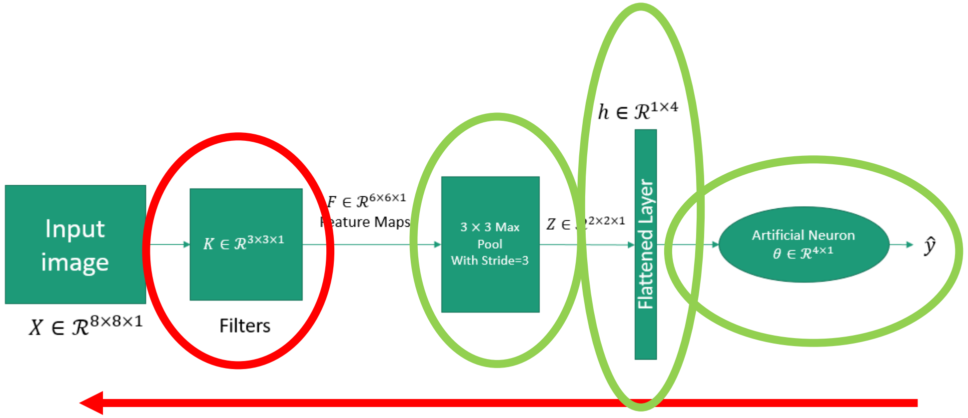
$$\frac{\partial F_{ab}}{\partial K_{ij}} = X_{a-\mathrm{i},b-\mathrm{j}}$$

- So then, what is $\frac{\partial F}{\partial K_{ij}}$?
- We need to think about what elements of F will K_{ij} affect?
- This is actually (somewhat) simple!

$$\frac{\partial F}{\partial K_{ij}} = X_{H-i+1:-1:M-i+1,\ W-j+1:-1:M-j+1}$$

- Note: $H - i + 1: -1: M - i + 1$ means from $H - i + 1$ ***down to*** $M - i + 1$
  - This achieves the flipping done by convolution.

Full Example



$$\frac{\partial J}{\partial K_{ij}} = (-1)\cdot\begin{bmatrix}1\\2\\3\\4\end{bmatrix}^T \cdot reshape\left(select\left(X_{H-i+1:-1:M-i+1,\ W-j+1:-1:M-j+1}\right)\right)$$

$$\frac{\partial J}{\partial K_{12}}$$

$$K = \begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}$$

- Let's find $\frac{\partial J}{\partial K_{11}}$ for our example:

$$\frac{\partial J}{\partial K_{11}} = (-1)\cdot\begin{bmatrix}1\\2\\3\\4\end{bmatrix}^T \cdot reshape\left(select\left(\begin{bmatrix}[illegible]&63&62&61&60&59\\ [illegible]&55&54&53&52&51\\ [illegible]&47&46&45&44&43\\ [illegible]&39&38&37&36&35\\ [illegible]&31&30&29&28&27\\ [illegible]&23&22&21&20&19\end{bmatrix}\right)\right) = [-1]\begin{bmatrix}1\\2\\3\\4\end{bmatrix}^T reshape\left(\begin{bmatrix}46&43\\22&19\end{bmatrix}\right)$$

$$[3\ 2\ 1]$$

$$X = \begin{bmatrix}1&2&3&4&5&6&7&8\\9&10&11&12&13&14&15&16\\17&18&19&20&21&22&23&24\\25&26&27&28&29&30&31&32\\33&34&35&36&37&38&39&40\\41&42&43&44&45&46&47&48\\49&50&51&52&53&54&55&56\\57&58&59&60&61&62&63&64\end{bmatrix}$$

$$\frac{\partial J}{\partial K_{11}} = (-1)\begin{bmatrix}1\\2\\3\\4\end{bmatrix}^T [46\ 22\ 43\ 19] = -295$$

$$[45, 21, 42, 18]$$

$$F = \begin{bmatrix}300&345&390&435&480&525\\660&705&750&795&840&885\\1020&1065&1110&1155&1200&1245\\1380&1425&1470&1515&1560&1605\\1740&1785&1830&1875&1920&1965\\2100&2145&2190&2235&2280&2325\end{bmatrix}$$

Full Example

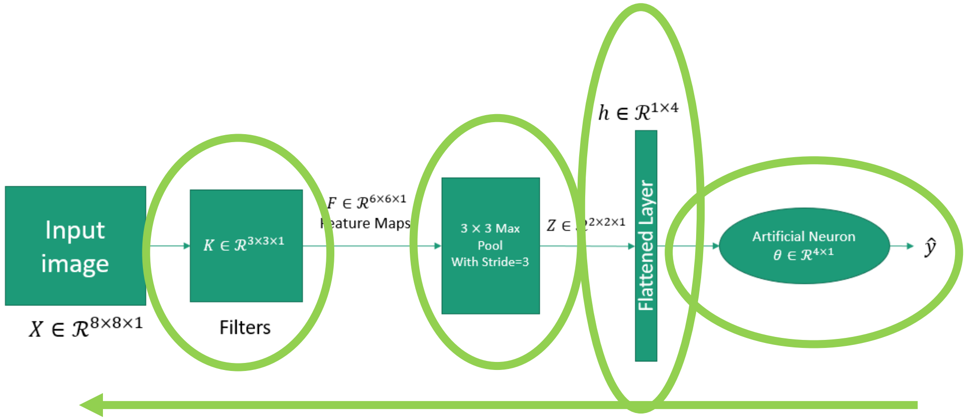
$$X = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \\ 25 & 26 & 27 & 28 & 29 & 30 & 31 & 32 \\ 33 & 34 & 35 & 36 & 37 & 38 & 39 & 40 \\ 41 & 42 & 43 & 44 & 45 & 46 & 47 & 48 \\ 49 & 50 & 51 & 52 & 53 & 54 & 55 & 56 \\ 57 & 58 & 59 & 60 & 61 & 62 & 63 & 64 \end{bmatrix}, K = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

- Doing this for all elements in K

$$\frac{\partial J}{\partial K} = \begin{bmatrix} -295 & -285 & -275 \\ -215 & -205 & -195 \\ -135 & -125 & -115 \end{bmatrix}$$

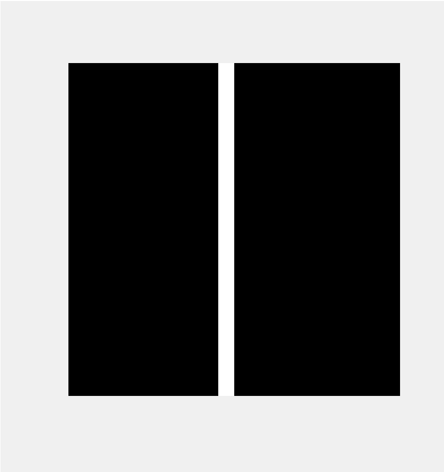
- And since our objective function is log likelihood, we want to maximize it and thus:

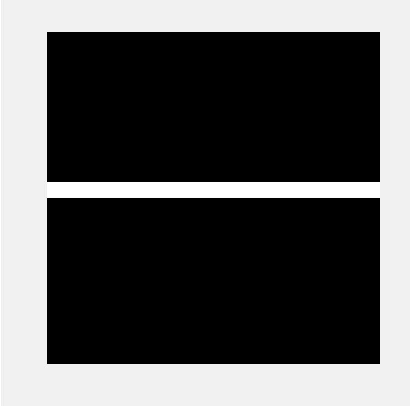
$$K = K + \eta \frac{\partial J}{\partial K}$$



Another Example

- Below are two artificially created 20×20 images.
- The left one we'll call "class 1" and the right one "class 0"



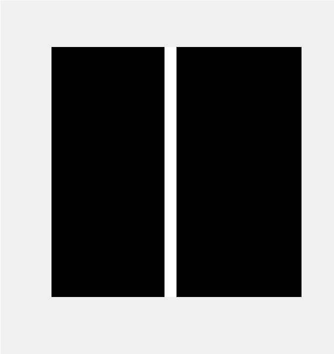


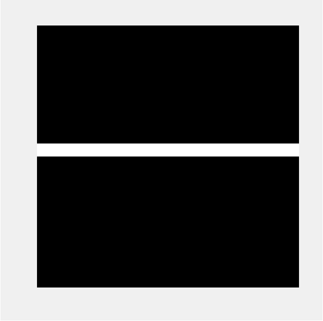
Another Example

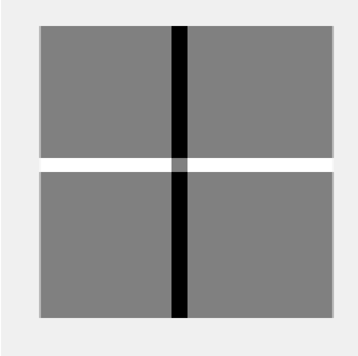
- Our CNN will be configured to have a convolutional layer with a single 20×20 kernel filter and a 1×1 max-pool layer with a stride of 1.
  - Note that the kernel size is the same as the image size!
  - Although this is rarely done, I did this for illustrative purposes.
- Therefore, our filter map will be 1×1 and the output of max-pooling will be 1×1.
- We will then flatten the output of the max-pooling layer to form a 1×1 vector to be fed into an artificial neuron with a logistic activation function and log likelihood objective function.
  - Therefore $\theta \in \mathcal{R}^{1\times 1}$

Another Example

- After 1,000 iterations, with $\eta = 0.1$ and L^2 regularization of 0.1, I thought it would be interesting to see what filter it learned!
- Thoughts?







Convolving with Multiple Kernels

- Most CNNs apply more than one kernel to an image.
- This shouldn't change much other than the fact that the output of convolution stage will be a 3D matrix (fyi, a multi-dimensional matrix is called a *tensor).*
- If our input to convolution is a $W \times H$ image and we have P $M \times M$ kernels, we will then obtain multiple feature maps as a tensor.
- The output value for feature map p, at location (a, b), we now have:

$$F_{(a,b,p)} = \sum_{i=-\frac{M}{2}}^{\frac{M}{2}} \sum_{j=-\frac{M}{2}}^{\frac{M}{2}} X_{(a-i,b-j)} K_{(i,j,p)}$$

- And of course the output of the pooling stage will also be a tensor.
- So we need to keep this in mind when we flatten the output to be the input to our traditional network.

Convolving with Color Images

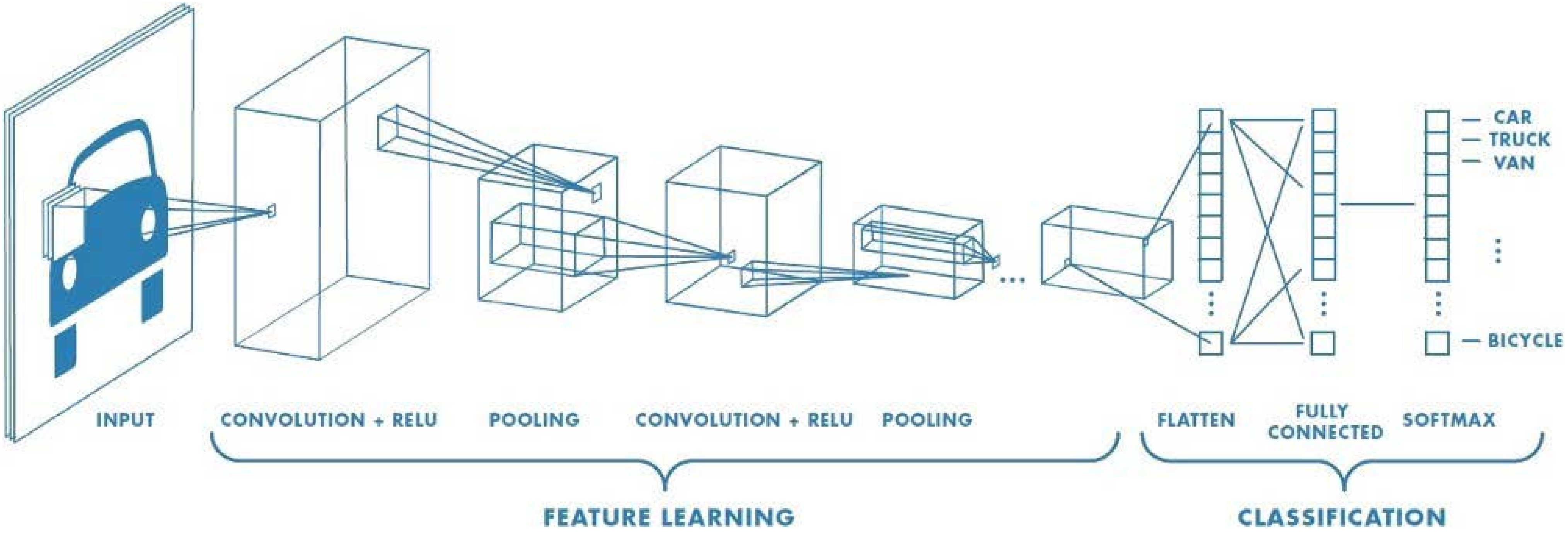
- What if we have an RGB color image?
- Then our image is $W \times H \times 3$
- And therefore our filters/kernels will be $M \times M \times 3$
- And our convolution function will be:

$$F_{a,b} = \sum_{l=1}^{3} \sum_{i=-\frac{M}{2}}^{\frac{M}{2}} \sum_{j=-\frac{M}{2}}^{\frac{M}{2}} X_{(a-i,b-j,l)} K_{(i,j,l)}$$

- This will also be important if/when we stack several convolution layers together.
  - Since the output of one layer *could* be a tensor.

Bringing it all together

- Used in series with fully connected layers
- Convolutions find the important patterns in the input
- Fully connected layers abstract the existence of the patterns into a label



Vectorizing/Batching

- Due to CNNs being not fully connected, sharing weights, and the pooling layer, vectorizing some of the partial gradients is a bit tricky, so we'll leave this to figure reading.
- But here's one resource:
  - http://lxu.me/mypapers/vcnn_aaai15.pdf
- Same thing goes for batch processing.
- Of course we can just compute the gradients for each sample then take the average of those gradients over a batch.

Sources

- Articles
  - http://www.robots.ox.ac.uk/~vgg/practicals/cnn/
  - https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199
- Videos
  - https://www.youtube.com/watch?v=YRhxdVk_sIs
  - https://youtu.be/FTr3n7uBIuE
- Lectures
  - https://www.youtube.com/watch?v=BvrWiL2fd0M
  - http://neuralnetworksanddeeplearning.com/chap6.html