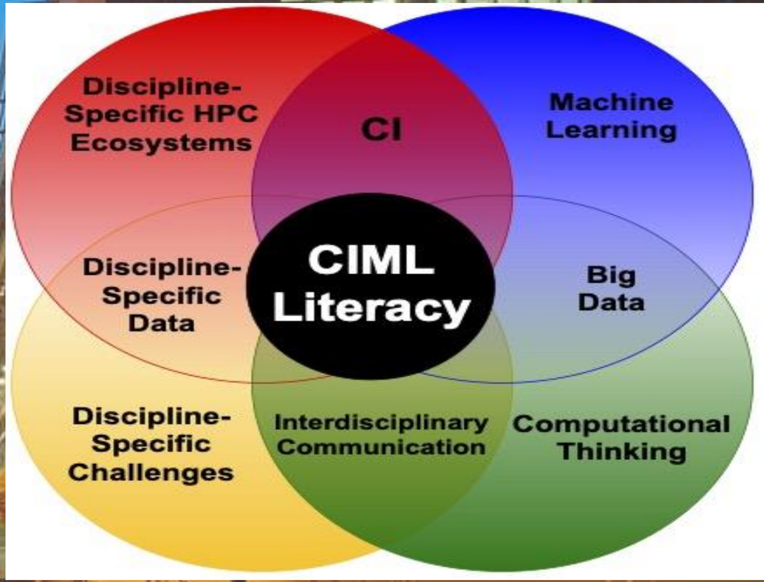
Introduction to Neural Network, Convolution Neural Networks, and Deep Learning

July, 2022



Paul Rodriguez, PhD (SDSC)

Outline

- Overview of Neural Networks (aka Multilayer Perceptron)
- Convolution Neural Networks
- Basic Keras commands for building, training CNNs - exercise
- Hyperparameters and Tuning and workflows - exercise

2nd session

4.3 Deep Learning 10:05 AM – 10:50 AM

[Here, I am going to do the basic CNN tutorial, but more of an exercise than tutorial, meaning they will have to fill in code parts more on their own. I am also going to do some tutorial/demo with the hyperparameter tuner tool, so that's a new thing, but it follows the slides I did in the webinar]

Practical things to Consider, aka, new stuff for the extended tutorial,

- **MNIST exercise/tutorial with Keras on Expanse**
- **Hyperparameter Tuning, keras tuner, tutorial**
- **Setting up configurations with yaml – demo/tutorial**
- **Sample Workflow, batch job -> yaml file -> hyperparam search ->model - demo**

to get neural network:

Consider the Logistic Function

(aka sigmoid)

$$f(x) = \frac{1}{1 + exp^{(-(b+wx))}}$$

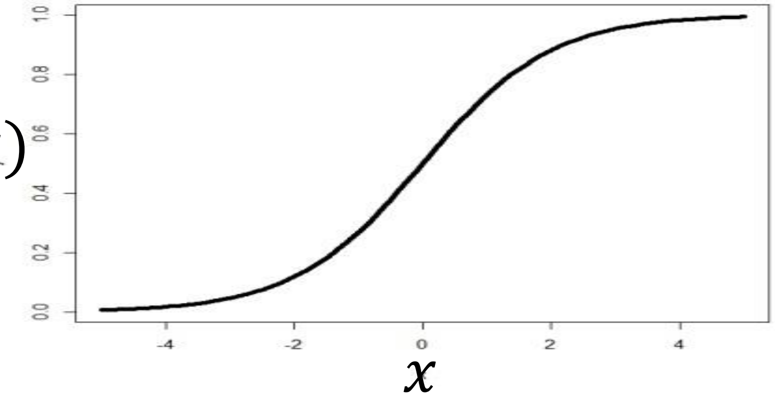
to get neural network:

Consider the Logistic Function

(aka sigmoid)

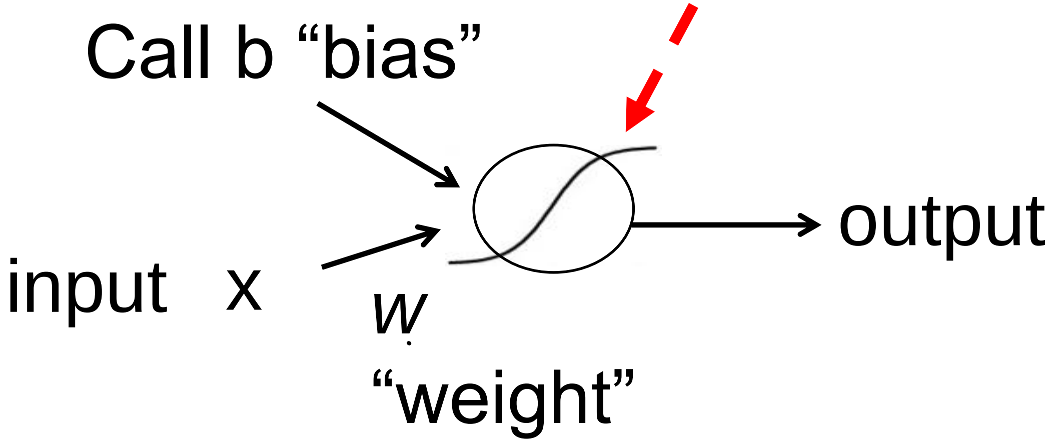
$$f(x) = \frac{1}{1 + exp^{(-(b+wx))}}$$

for parameters: $b = 0$, $w_1 = 1$



Make a graphical description of Logistic Function

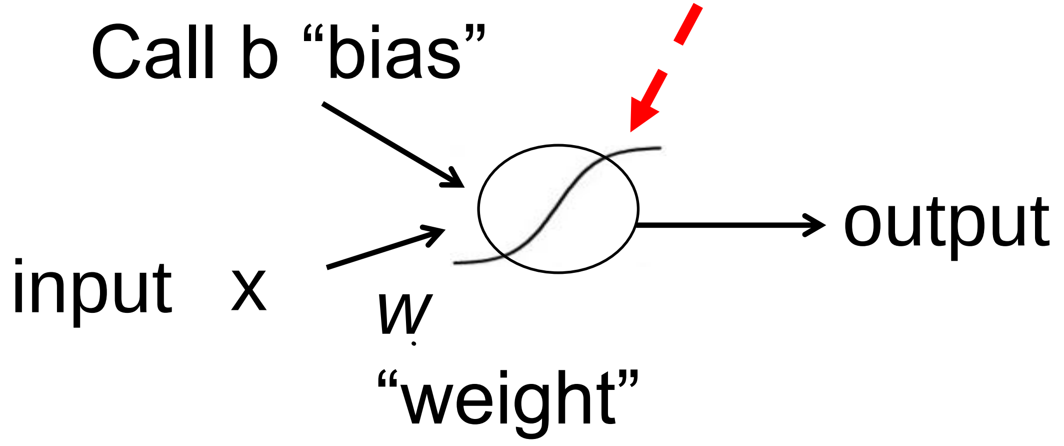
$$f(x) = \frac{1}{1 + exp^{(-(b+wx))}}$$



this node (or unit) will transform input to output with logistic activation function

Make a graphical description of Logistic Function

$$f(x) = \frac{1}{1 + exp^{(-(b+wx))}}$$

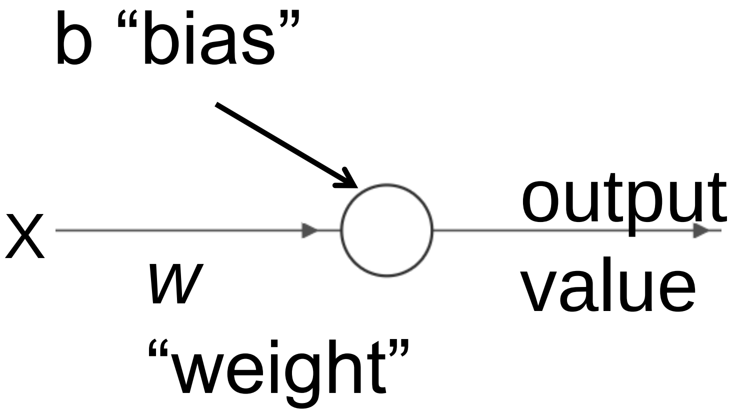


this node (or unit) will transform input to output with logistic activation function

Goal: given some data X and outputs Y find *w* & *b* so that output =1 when Y=1

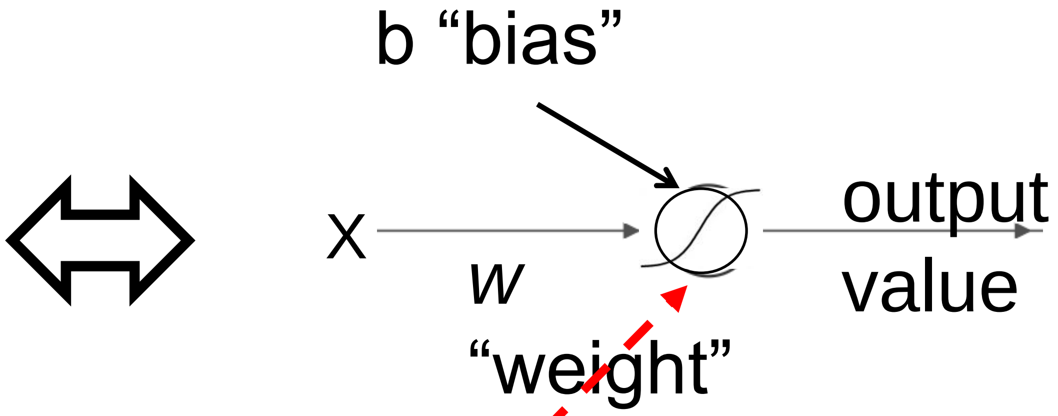
How does changing parameters affect the function?

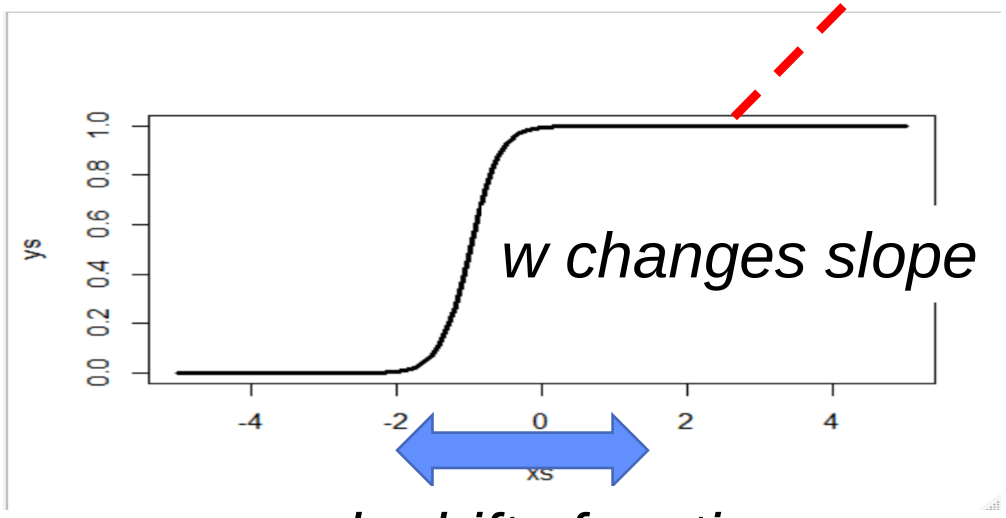
$$f(x) = \frac{1}{1 + exp^{(-(b+wx))}} \Leftrightarrow$$



How does changing parameters affect the function?

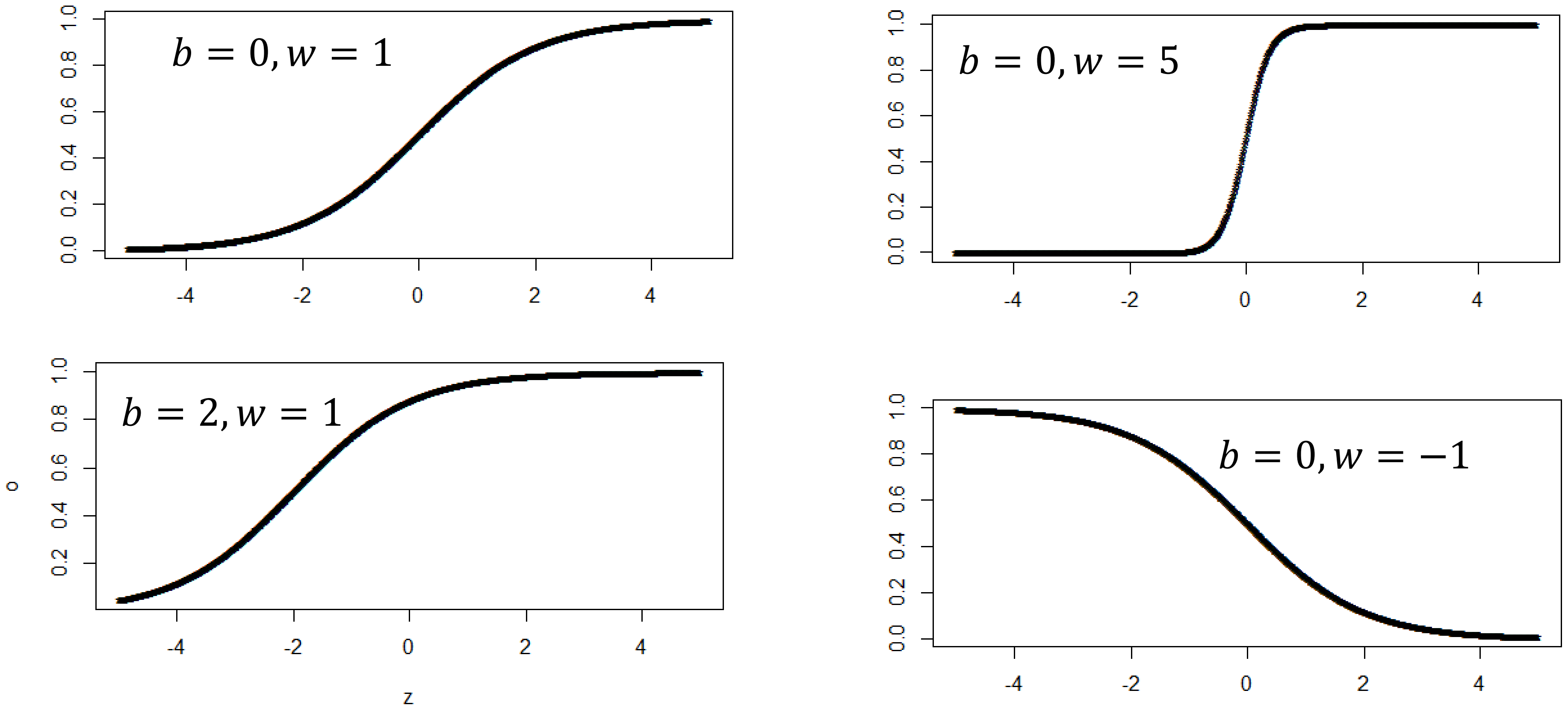
$$f(x) = \frac{1}{1 + exp^{(-(b+wx))}}$$



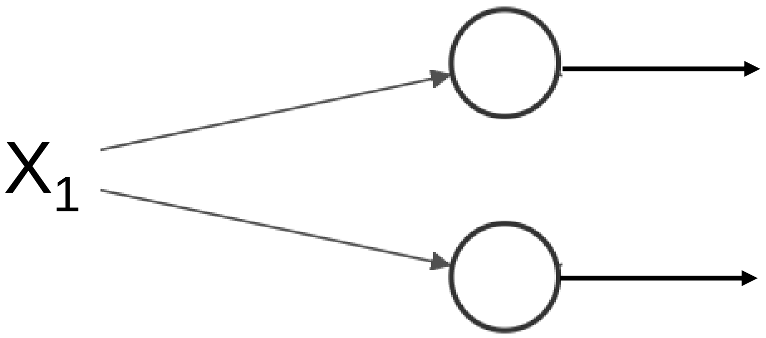


b shifts function

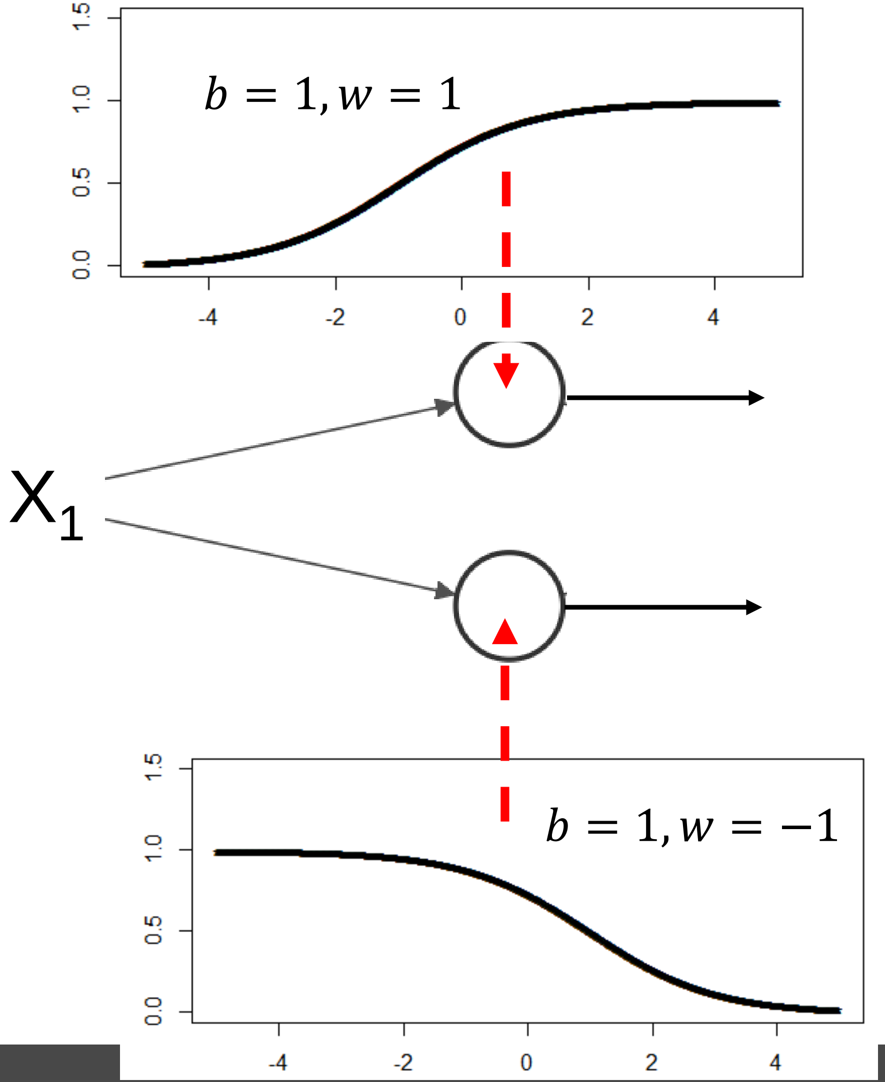
Logistic function w/various weights



Example: 1 input into 2 logistic units

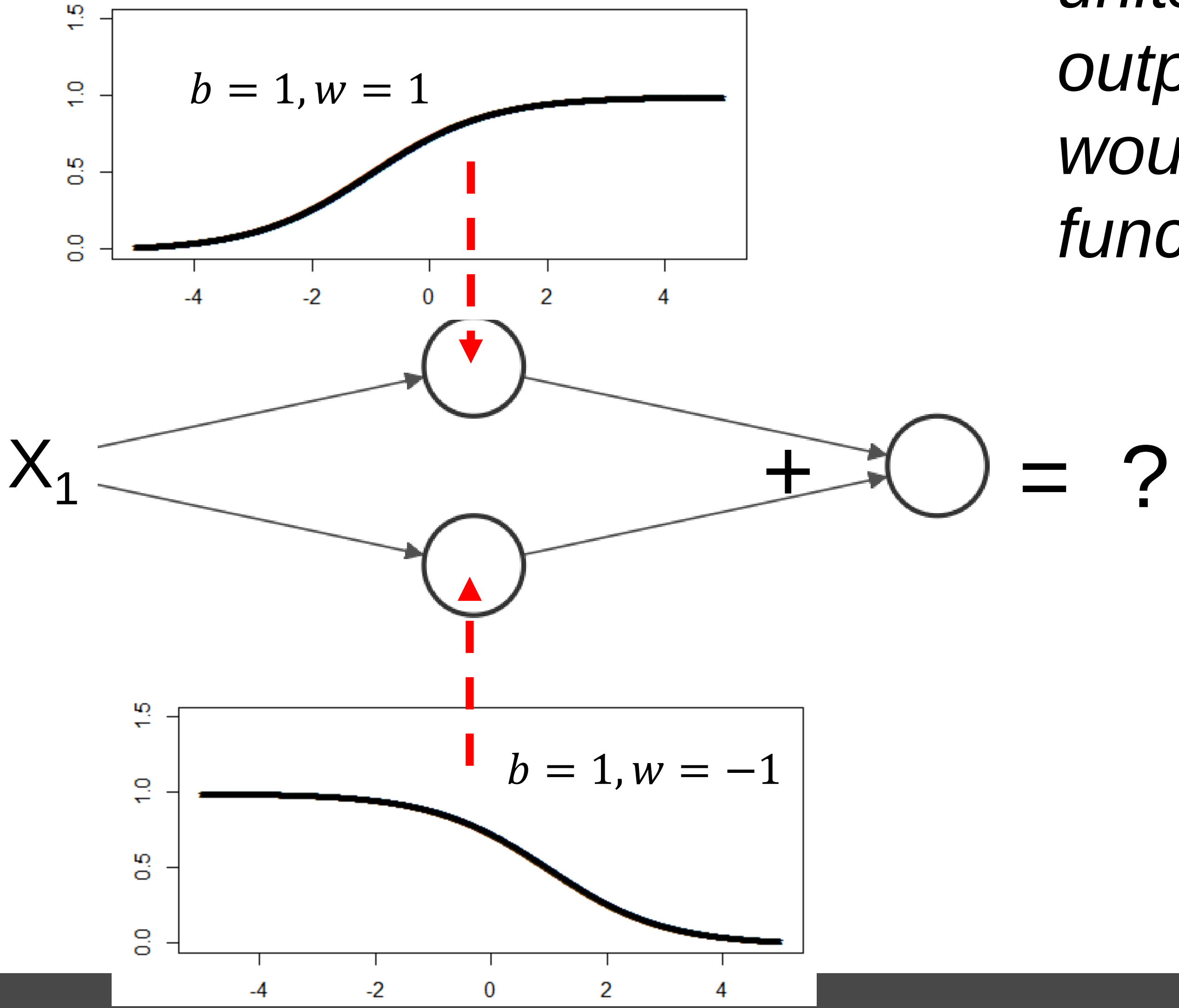


Example: 1 input into 2 logistic units with these activations



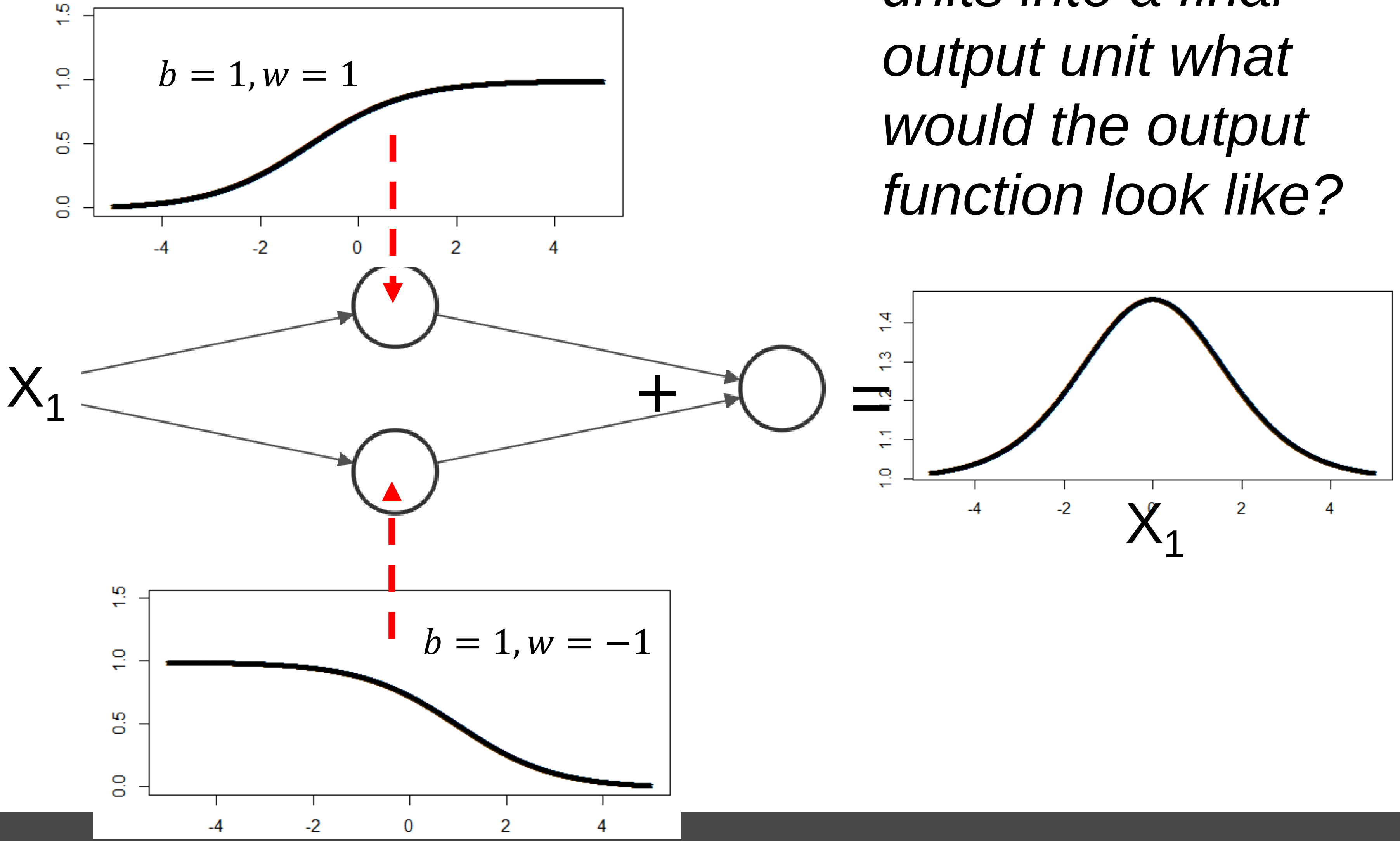
Example: 1 input into 2 logistic units with these activations

If you add these 2 units into a final output unit what would the output function look like?



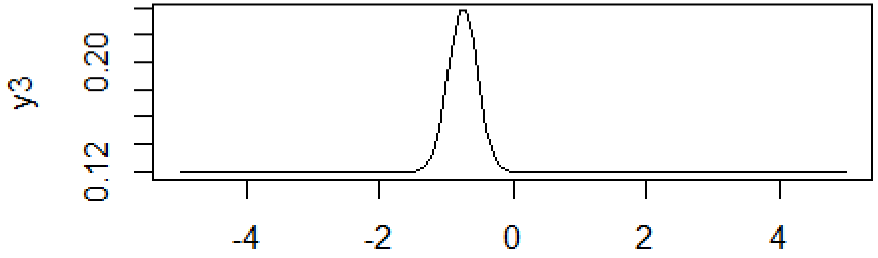
Example: 1 input into 2 logistic units with these activations

If you add these 2 units into a final output unit what would the output function look like?

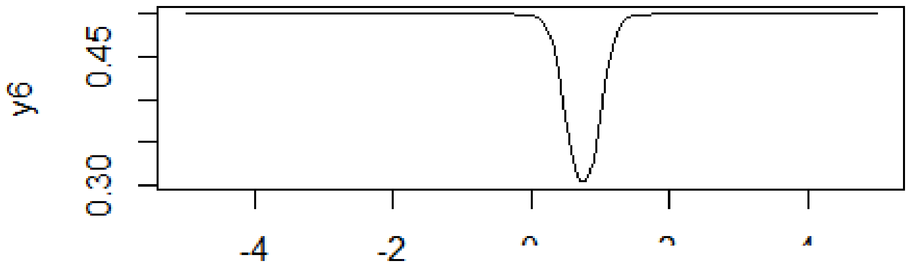


Higher level function combinations

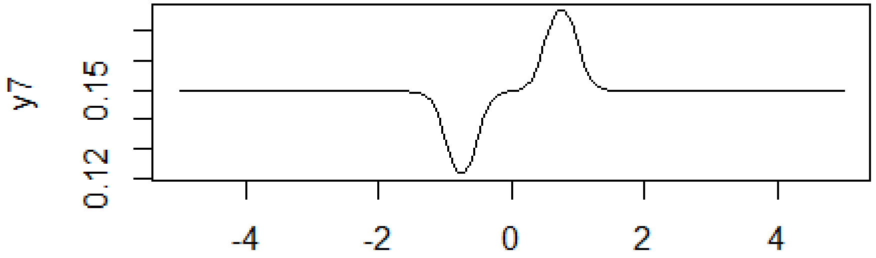
```
x=seq(-5,5,.1)
y1=1/(1+exp(10+   10*x))
y2=1/(1+exp(-5+(-10)*x))
y3=1/(1+exp(1+1*y1+1*y2))
plot(x,y3,type="l")
```




```
y4=1/(1+exp(10+ (-10)*x))
y5=1/(1+exp(-5+(10)*x))
y6=1/(1+exp(1-1*y4-1*y5))
plot(x,y6,type="l")
```

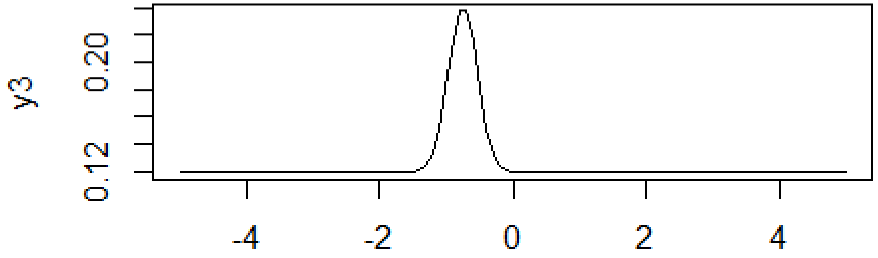



```
y7=1/(1+exp(1+2*y3+1*y6))
plot(x,y7,type="l")
```



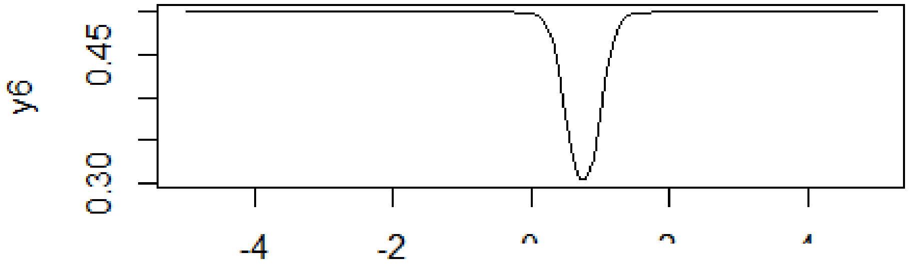
Higher level function combinations

```
x=seq(-5,5,.1)
y1=1/(1+exp(10+   10*x))
y2=1/(1+exp(-5+(-10)*x))
y3=1/(1+exp(1+1*y1+1*y2))
plot(x,y3,type="l")
```

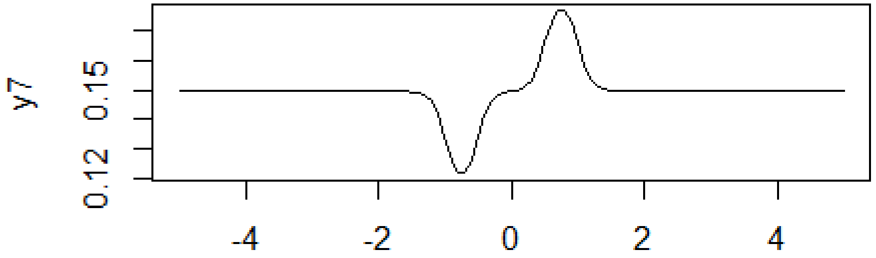


Multiple layer networks can represent any logical or real-valued functions (unbiased, but potential to overfit)

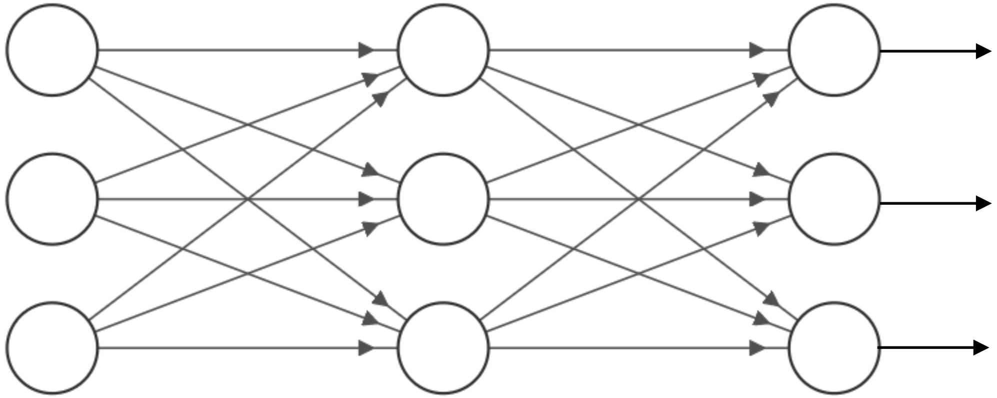
```
y4=1/(1+exp(10+ (-10)*x))
y5=1/(1+exp(-5+(10)*x))
y6=1/(1+exp(1-1*y4-1*y5))
plot(x,y6,type="l")
```




```
y7=1/(1+exp(1+2*y3+1*y6))
plot(x,y7,type="l")
```



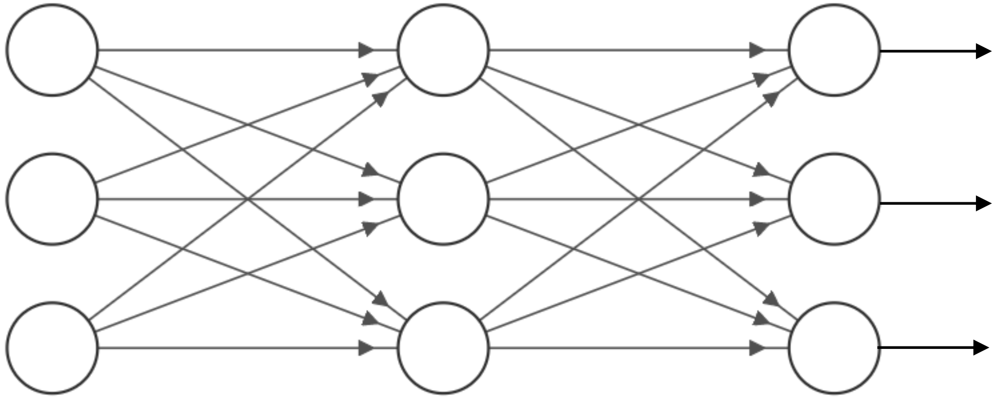
we can add layers and nodes



we can add layers and nodes

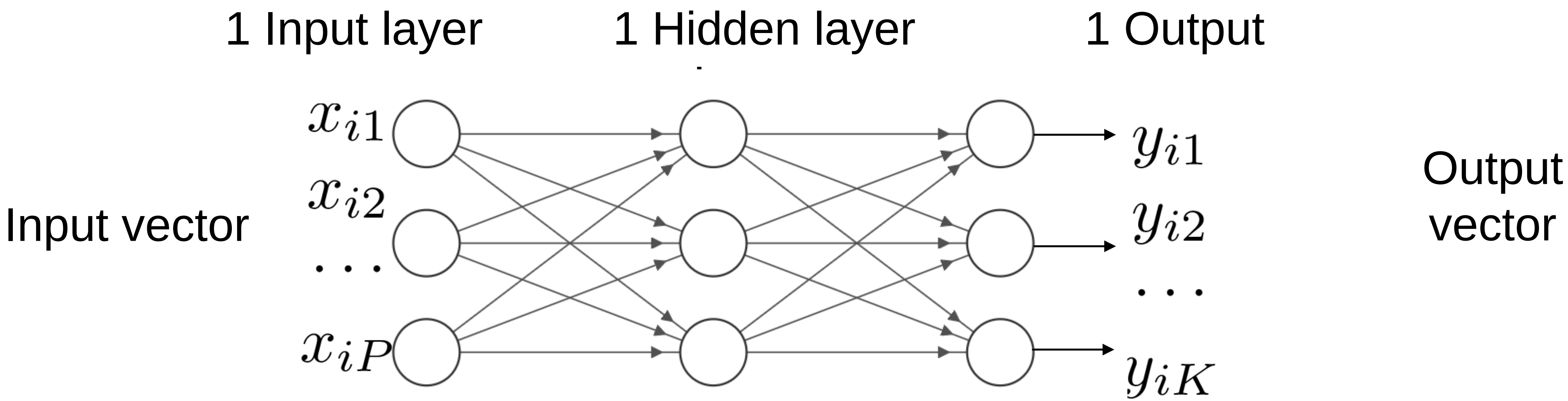
Multilayer Perceptron

1 Input layer 1 Hidden layer 1 Output



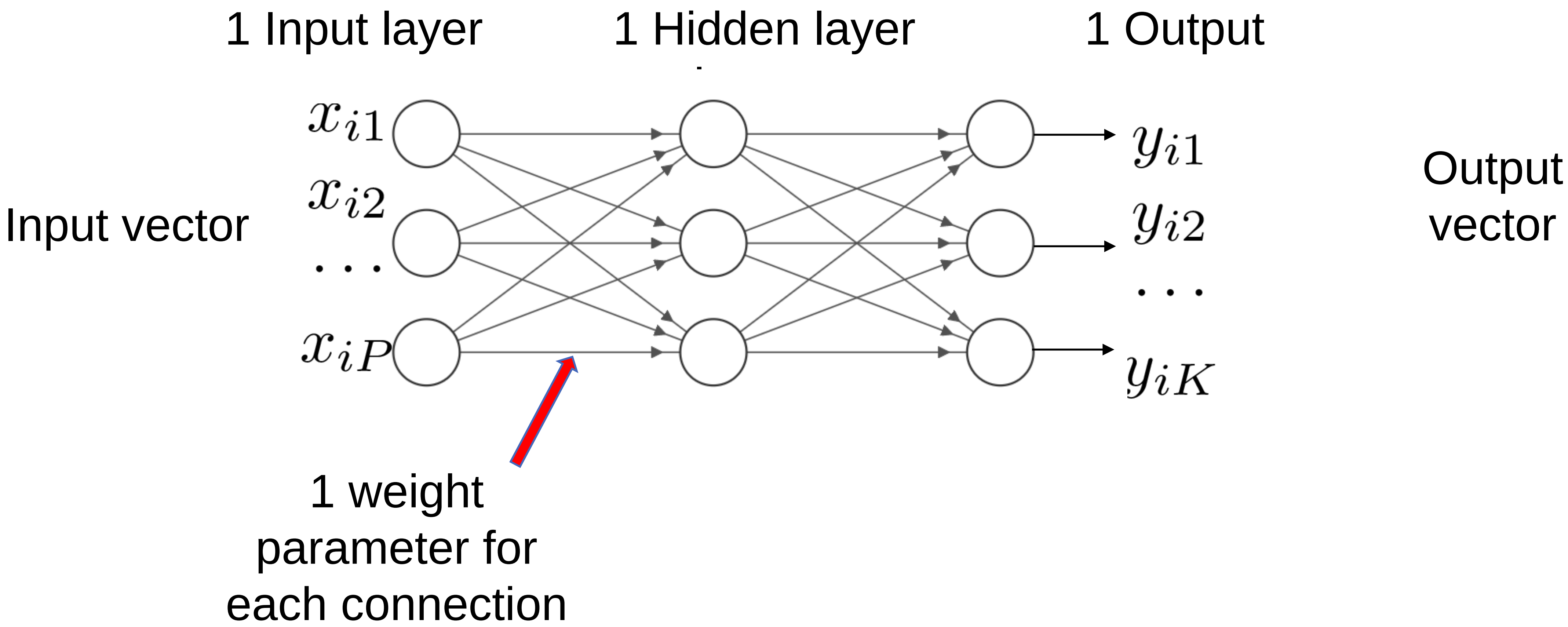
we can add layers and nodes

Multilayer Perceptron

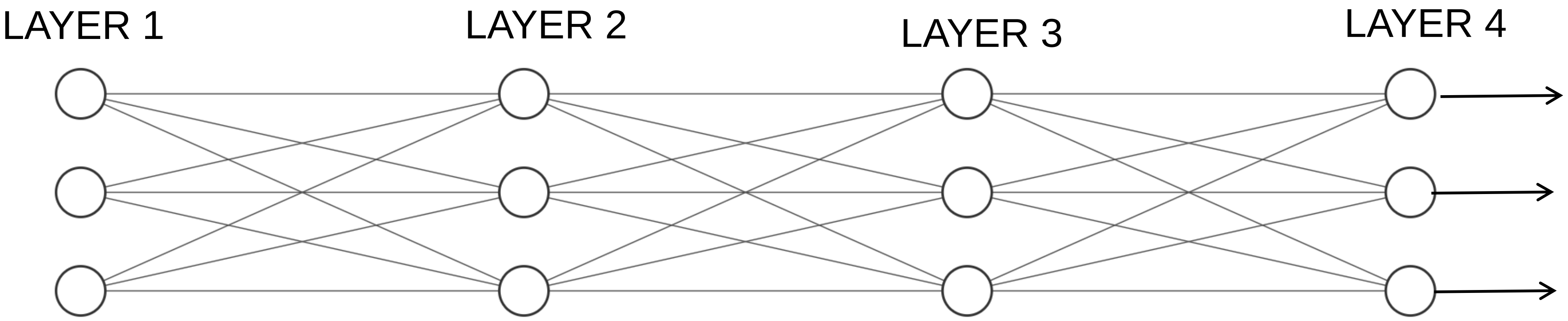


we can add layers and nodes

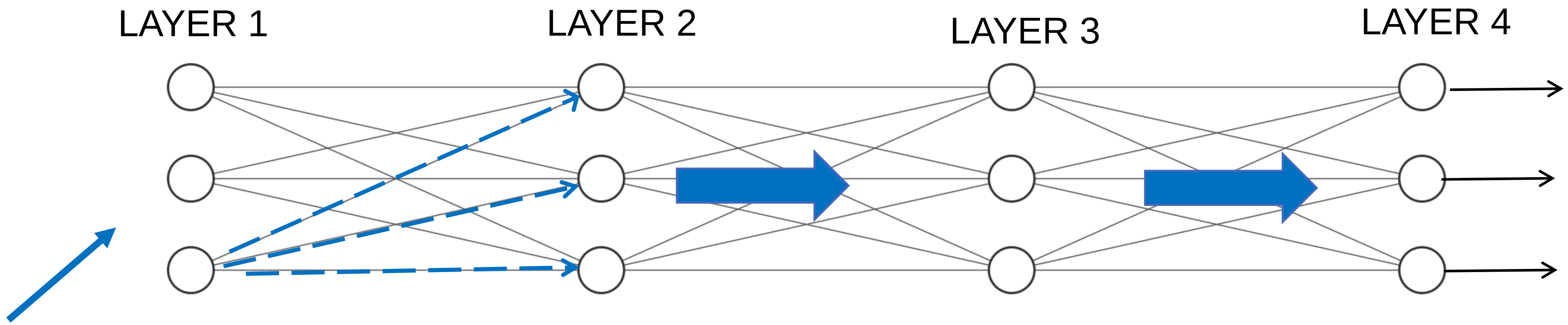
Multilayer Perceptron



First step: choose layers, connectivity, and activations



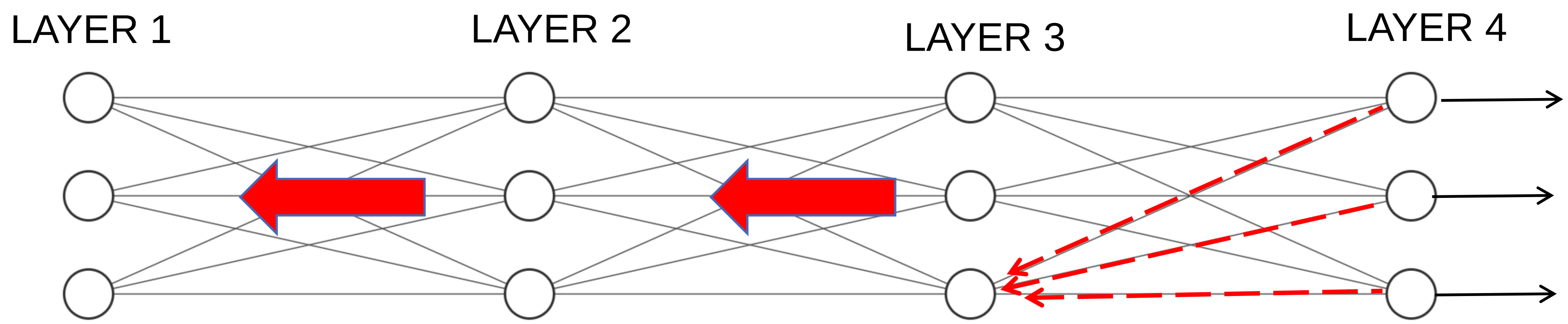
Algorithm steps:



1. **FORWARD PROPAGATE ACTIVATION**:

apply input data x_i ,
calculate all node activations

Algorithm steps:

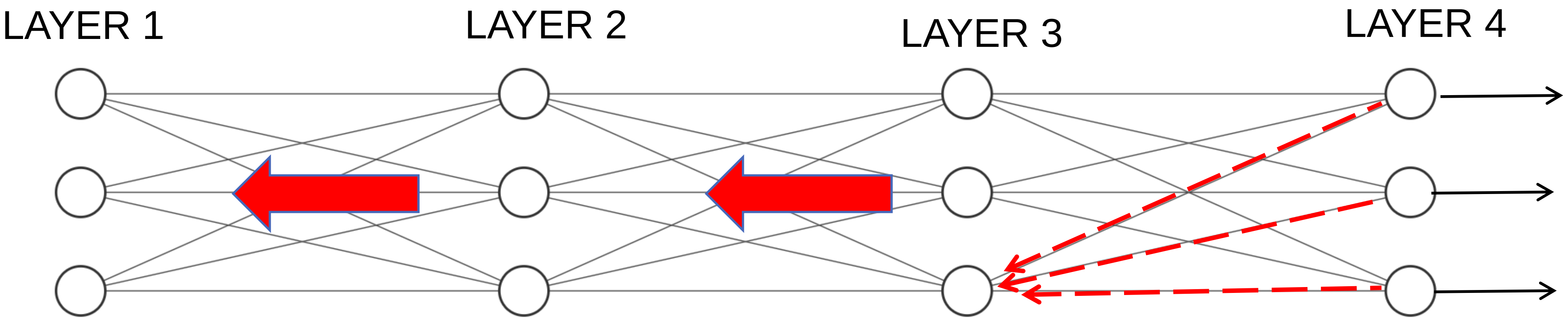


1. **FORWARD PROPAGATE ACTIVATION**:

apply input data x_i ,
calculate all node activations

2. BACKWARD PROPAGATE ERROR:
calculate Error (or Loss) derivatives, dE/dY,
pass it back to lower layer

algorithm steps:



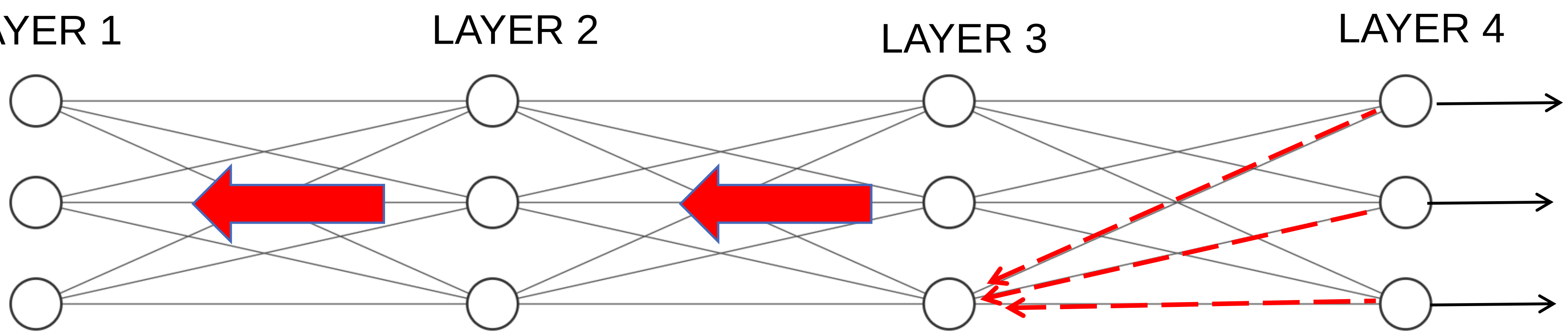
1. **FORWARD PROPAGATE ACTIVATION**:

apply input data x_i ,
calculate all node activations

2. BACKWARD PROPAGATE ERROR:
calculate Error (or Loss) derivatives, dE/dY,
pass it back to lower layer

For hidden layers use chain rule:
(dE/dY dY/dH_3 dH_3/dH_2 etc…)
needs a summation of previous layer

algorithm steps:



1. **FORWARD PROPAGATE ACTIVATION**:

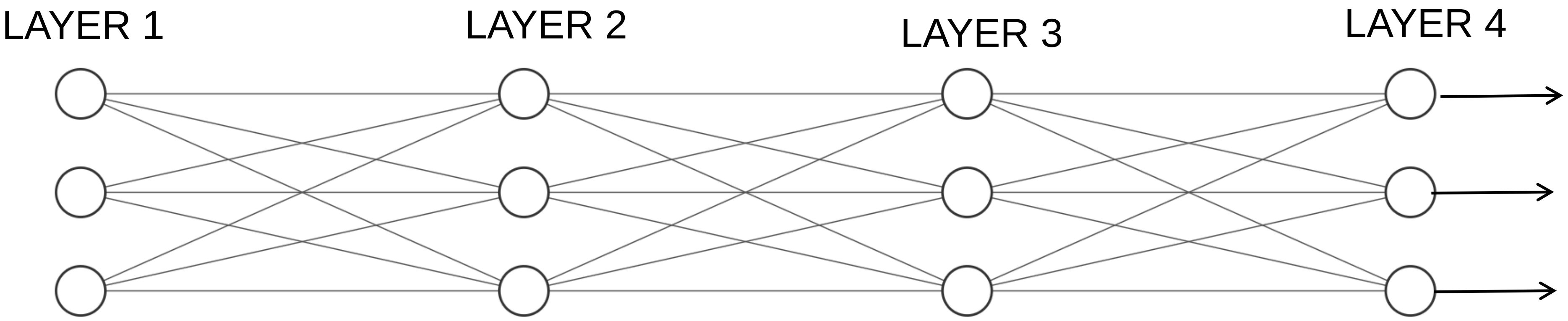
apply input data x_i ,
calculate all node activations

2. BACKWARD PROPAGATE ERROR:
calculate Error (or Loss) derivatives, dE/dY, pass it back to lower layer

For hidden layers use chain rule:
($dE/dY \quad dY/dH_3 \quad dH_3/dH_2$ etc…)
needs a summation of previous layers

Beware: error signals get diluted as you go backward - the 'vanishing gradient' problem

algorithm steps:



1. **FORWARD PROPAGATE ACTIVATION**:

apply input data x_i ,
calculate all node activations

2. BACKWARD PROPAGATE ERROR:
calculate Error (or Loss) derivatives (dE/dY)
pass it back to lower layer

3. Update weights and bias terms

$$w_{ji} = w_{ji} - \eta \frac{dE}{dw_{ji}}$$

NN Algorithm

INITIALIZE WEIGHTS (small random values)

NN Algorithm

INITIALIZE WEIGHTS

LOOP until stopping criterion:

FORWARD PROPAGATION: calculate all node activations

NN Algorithm

INITIALIZE WEIGHTS

LOOP until stopping criterion:

FORWARD PROPAGATION: calculate all node activations

BACKWARD PROPAGATION: calculate all derivatives to ***minimize Loss (dL)***

NN Algorithm

INITIALIZE WEIGHTS

LOOP until stopping criterion:

FORWARD PROPAGATION: calculate all node activations

BACKWARD PROPAGATION: calculate all derivatives to ***minimize Loss (dL)***

UPDATE WEIGHTS: $w \leftarrow w - \text{learning\_rate} * \frac{dL}{dw}$

NN Algorithm

INITIALIZE WEIGHTS

LOOP until stopping criterion:

FORWARD PROPAGATION: calculate all node activations

BACKWARD PROPAGATION: calculate all derivatives to ***minimize Loss (dL)***

UPDATE WEIGHTS: $w \leftarrow w - \text{learning\_rate} * \frac{dL}{dw}$

STOP: when validation loss reaches minimum or converges

NN Algorithm
[heuristics, options to learn faster and/or better]

INITIALIZE WEIGHTS **[use truncated distributions]**

LOOP until stopping criterion: **[work in batches of input]**

FORWARD PROPAGATION: calculate all node activations

BACKWARD PROPAGATION: calculate all derivatives to ***minimize Loss (dL)***

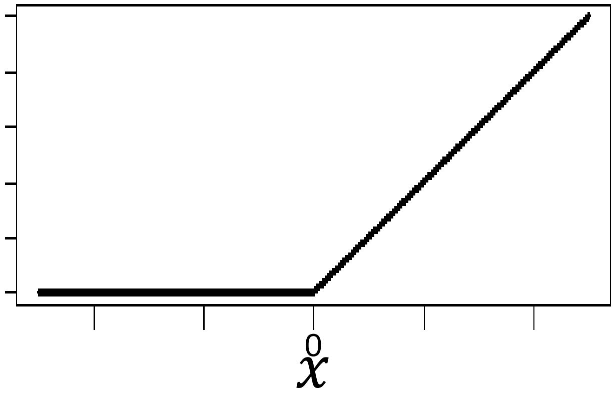
UPDATE WEIGHTS: $w \leftarrow w - learning\_rate * \frac{dL}{dw}$ **[adapt learning rate, use momentum]**

STOP: when validation loss reaches minimum or converges **[several metrics of loss are possible]**

A heuristic for deep networks

RELU activation function

RELU (rectified linear

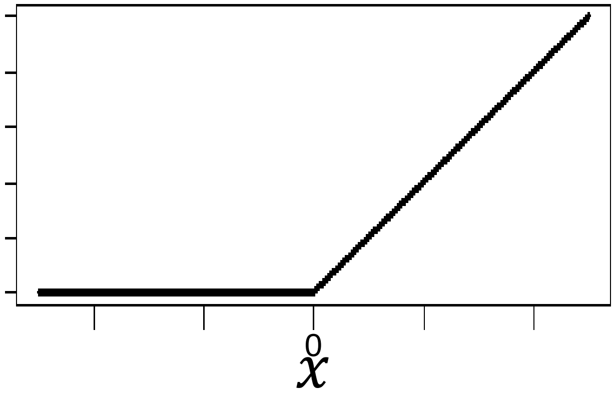


$$f(a) = \begin{cases} a & a > 0 \\ 0 & a <= 0 \end{cases}$$

where $a = XW$

A heuristic for deep networks

RELU (rectified linear



RELU activation function

It is unscaled (bad!)

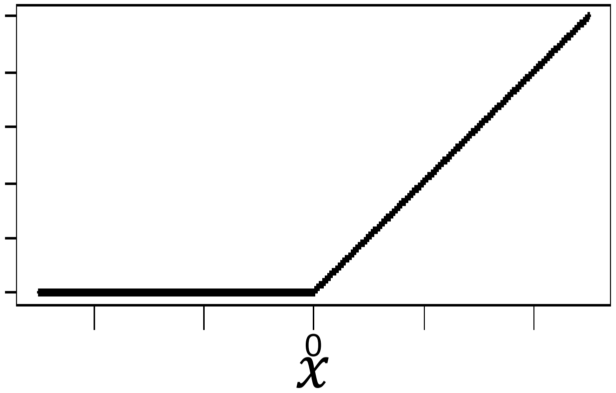
But *df/da* is constant (good!)

$$f(a) = \begin{cases} a & a > 0 \\ 0 & a <= 0 \end{cases}$$

where $a = XW$

A heuristic for deep networks

RELU (rectified linear



$$f(a) = \begin{cases} a & a > 0 \\ 0 & a <= 0 \end{cases}$$

where $a = XW$

RELU activation function

It is unscaled (bad!)

But *df/da* is constant (good!)

RELU helps mitigates vanishing gradients

Summary:

Pro:

- Neural Networks in general, are flexible, powerful learners
- Hidden layers learn a nonlinear transformation of input
- Many heuristics about what works

Summary:

Pro:

- Neural Networks in general, are flexible, powerful learners
- Hidden layers learn a nonlinear transformation of input
- Many heuristics about what works

Con:

- Hard to interpret
- Needs more data
- Lots of parameters

What is deep learning?

What is deep learning?

Deep learning refers to learning complex and varied transformations of the input

What is deep learning?

Deep learning refers to learning complex and varied transformations of the input

Deep learning refers to **discovering** useful features of the input

What is deep learning?

Deep learning refers to learning complex and varied transformations of the input

Deep learning refers to **discovering** useful features of the input

Deep learning is a neural network with many layers

Outline

- **Overview of Neural Networks (aka Multilayer Perceptron)**
- **Convolution Neural Networks**
- **Basic Keras commands for building, training CNNs - exercise**
- **Hyperparameters and Tuning and workflows - exercise**

Image features

- **MNIST - A database of handwritten printed digits**

  (National Inst. of Standards and Technology)

Image features

- **MNIST - A database of handwritten printed digits**

  (National Inst. of Standards and Technology)

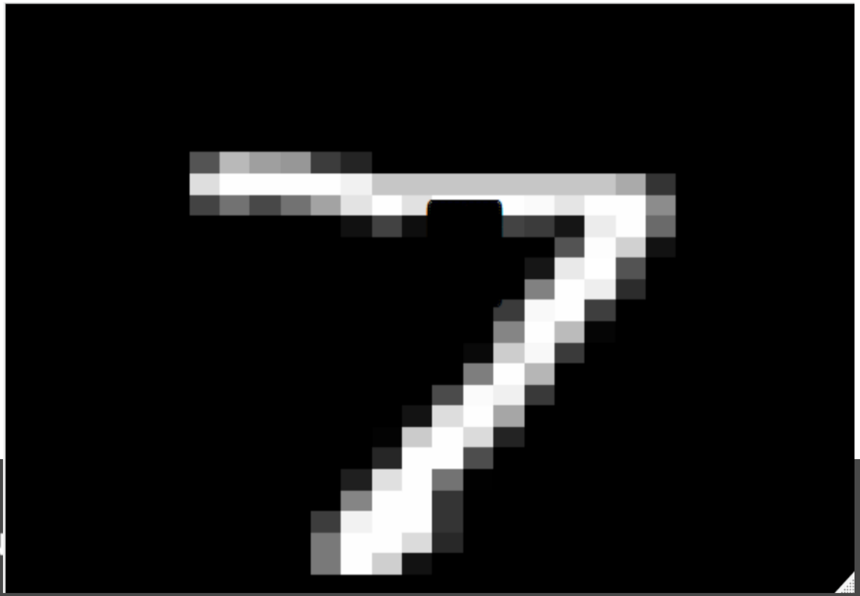
How to classify digits?

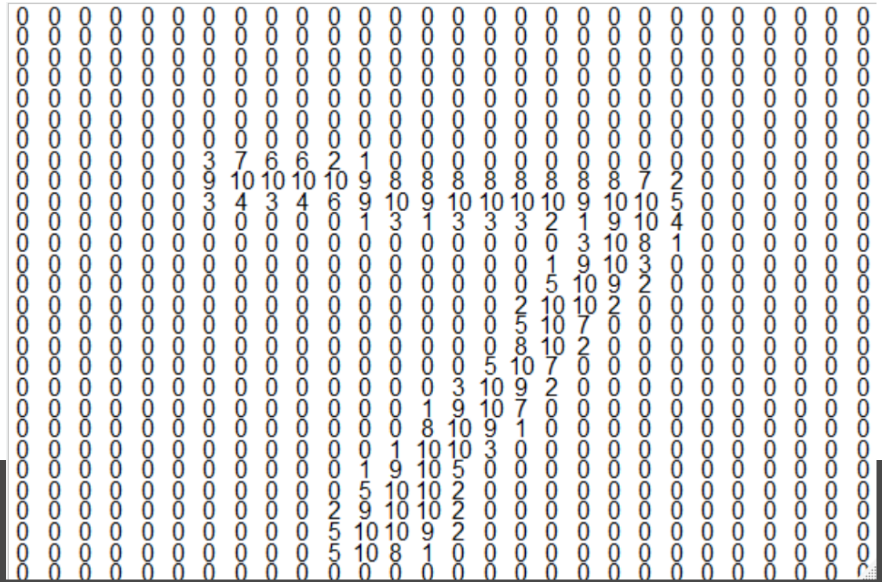
Image features

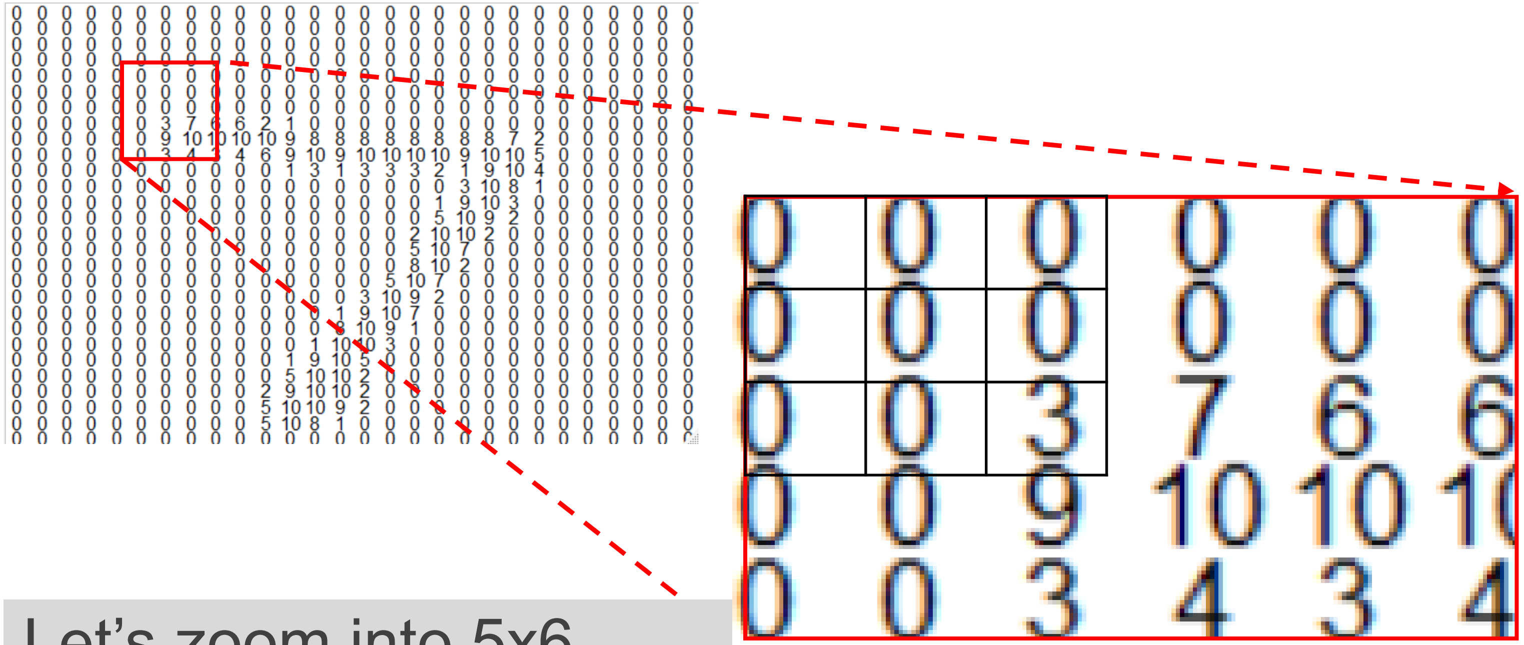
- **MNIST - A database of handwritten printed digits**

  (National Inst. of Standards and Technology)

How to classify digits?

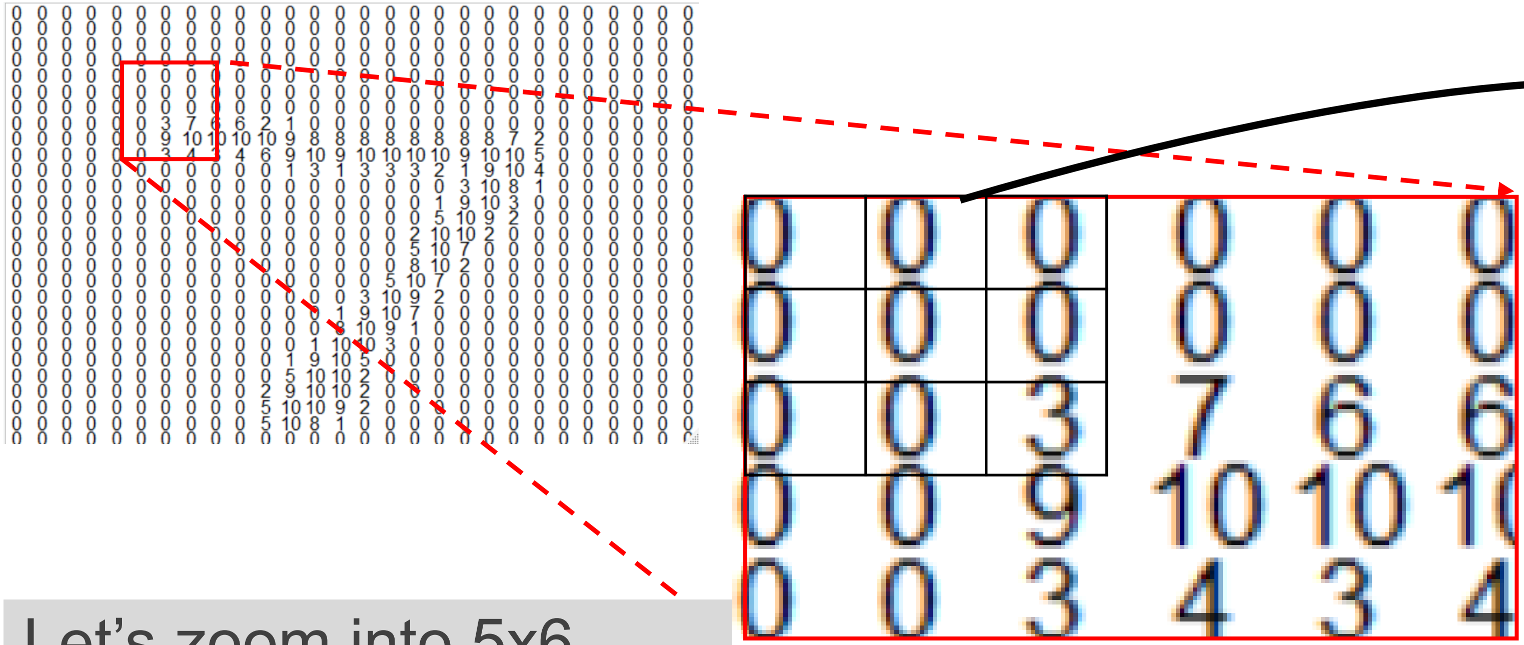






Let’s zoom into 5x6 window of pixels near the tip of ‘7’

Take a 3x3 patch of pixels and apply a ‘filter’ template – designed to find an edge



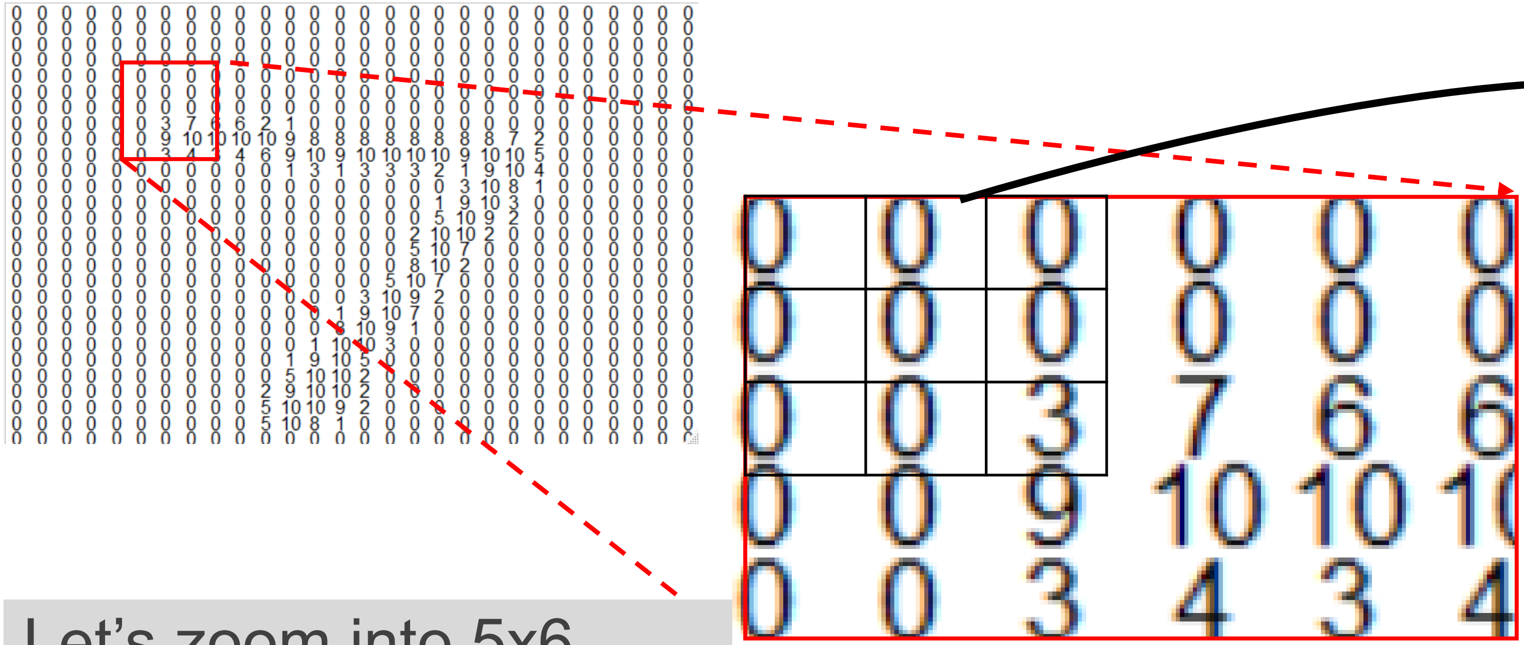
X

-1	0	+1
-1	0	+1
-1	0	+1

1. Multiply 3x3 patch of pixels with 3x3 filter

Let’s zoom into 5x6 window of pixels near the tip of ‘7’

Take a 3x3 patch of pixels and apply a ‘filter’ template – designed to find an edge



(our weight parameters)

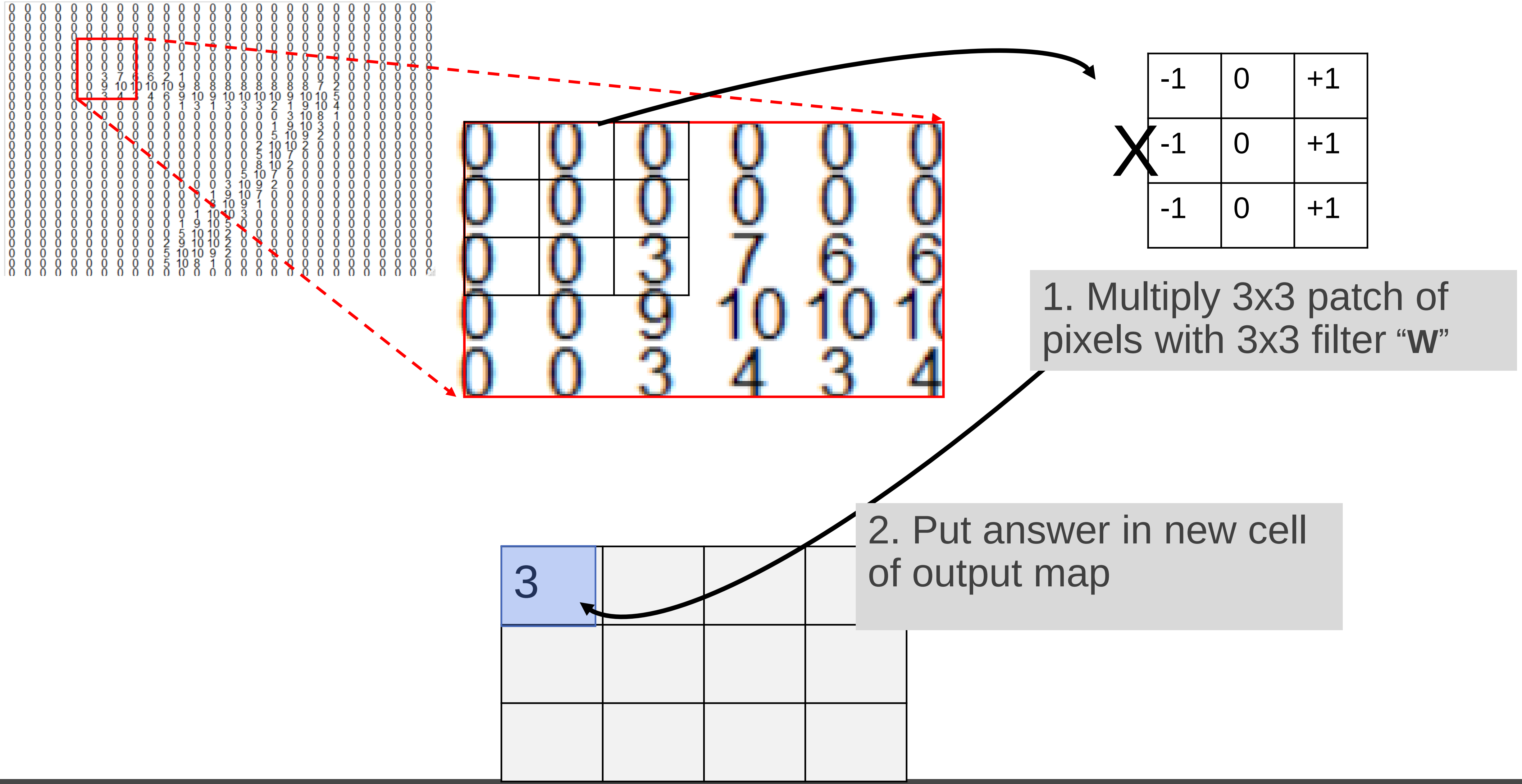
X

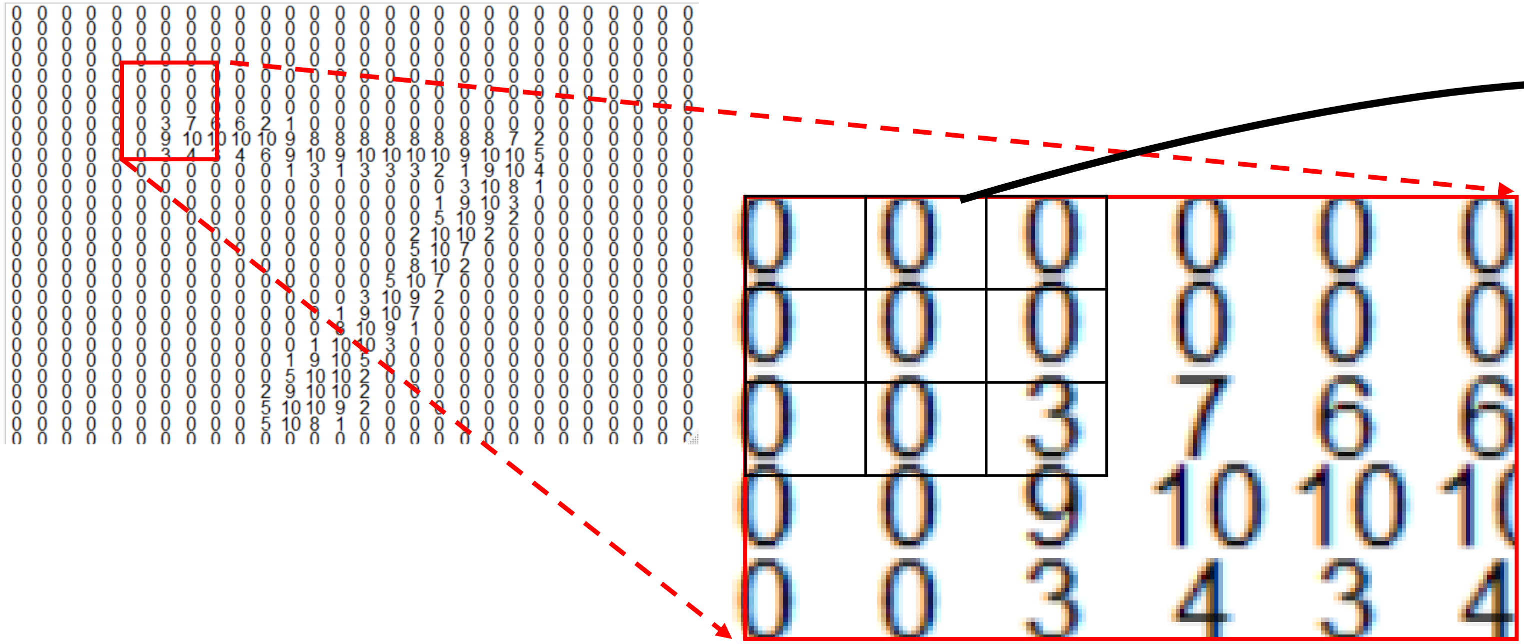
-1	0	+1
-1	0	+1
-1	0	+1

1. Multiply 3x3 patch of pixels with 3x3 filter “**W**”

Let’s zoom into 5x6 window of pixels near the tip of ‘7’

Take a 3x3 patch of pixels and apply a ‘filter’ template – designed to find an edge


0 0 0 0 0 0
0 0 0 0 0 0
0 0 3 7 6 6
0 0 9 10 10 10
0 0 3 4 3 4
X
-1 0 +1
-1 0 +1
-1 0 +1
1. Multiply 3x3 patch of pixels with 3x3 filter “**w**”
2. Put answer in new cell of output map
3



X

-1	0	+1
-1	0	+1
-1	0	+1

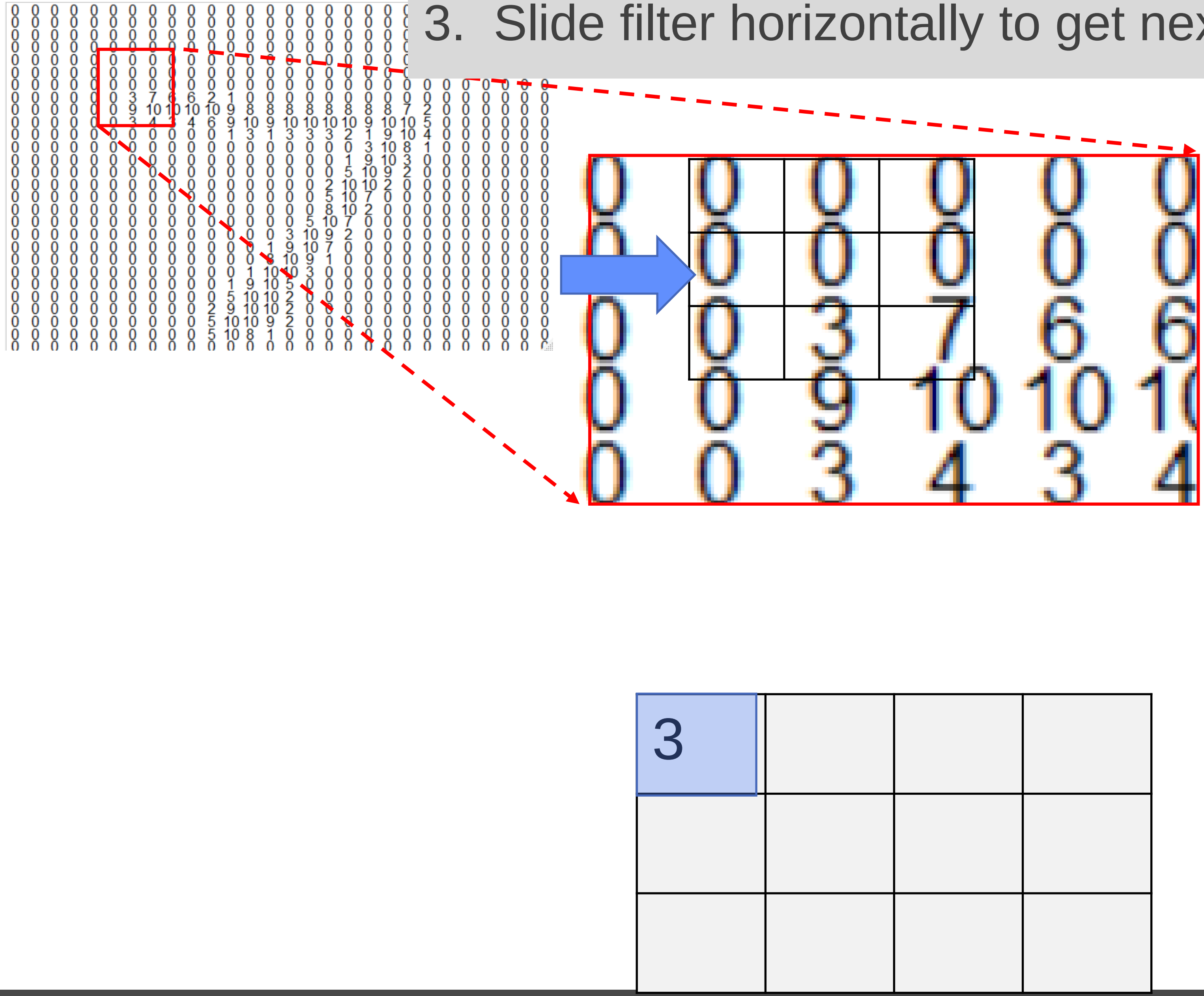
1. Multiply 3x3 patch of pixels with 3x3 filter “**W**”

This is “image patch”$*$W

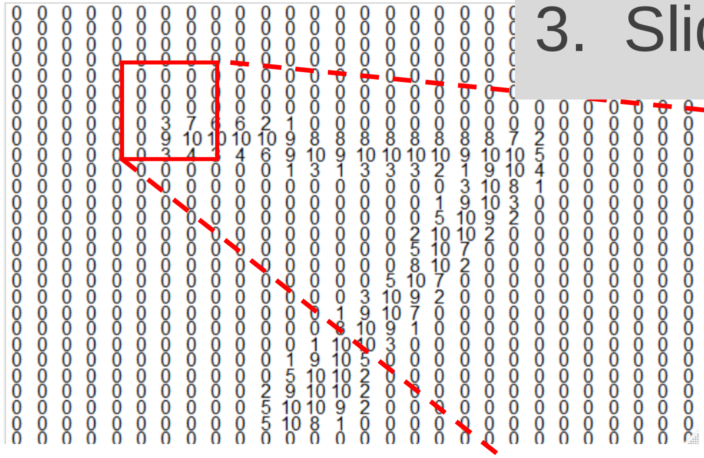
2. Put answer in new cell of output map

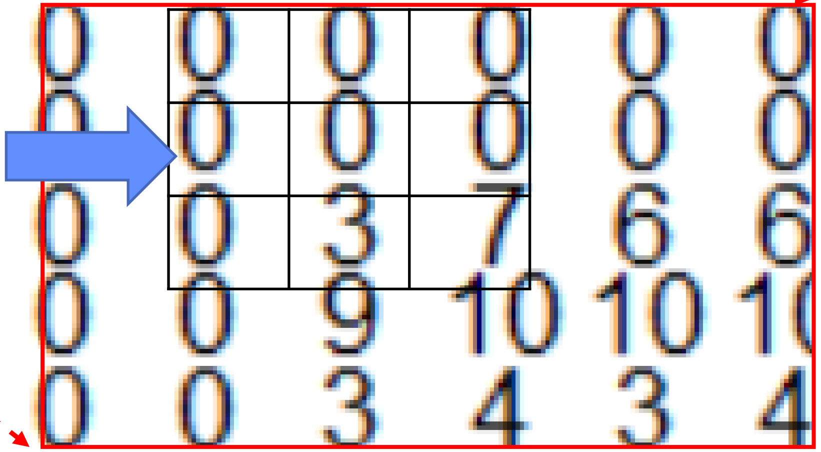
3			

3. Slide filter horizontally to get next output value



3. Slide filter horizontally to get next output value





X

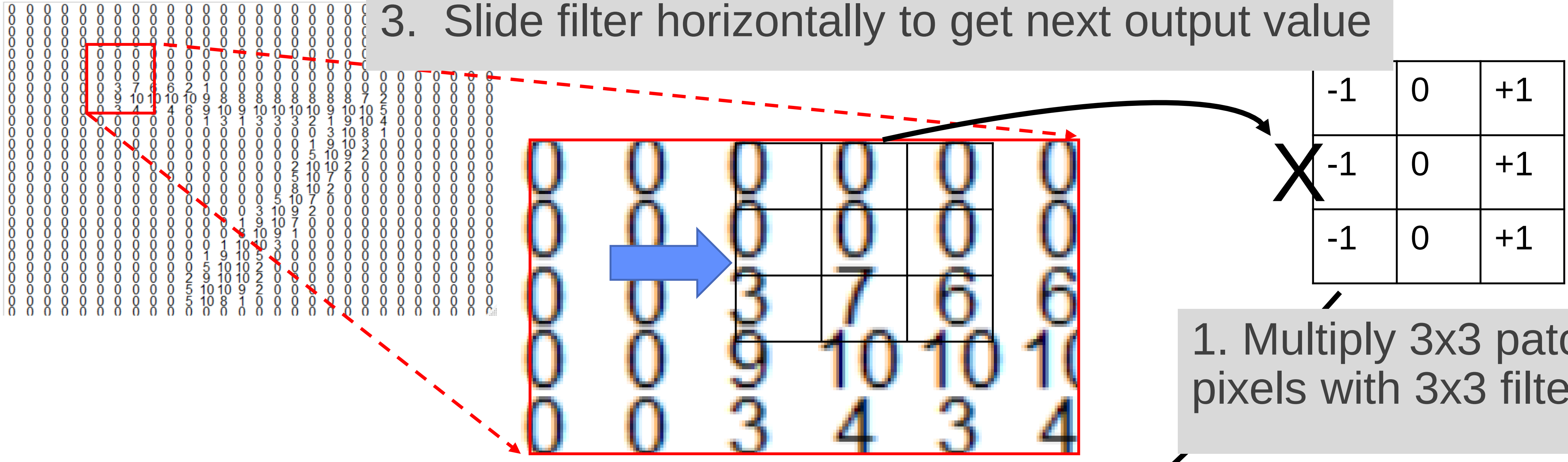
-1	0	+1
-1	0	+1
-1	0	+1

1. Multiply 3x3 patch of pixels with 3x3 filter “**w**”

2. Put answer in new cell of output map

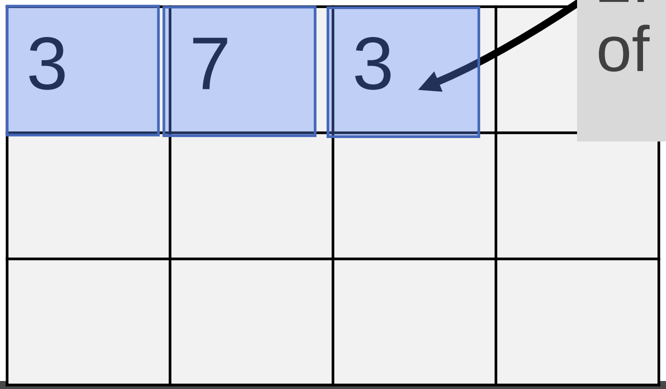
3	7		

3. Slide filter horizontally to get next output value



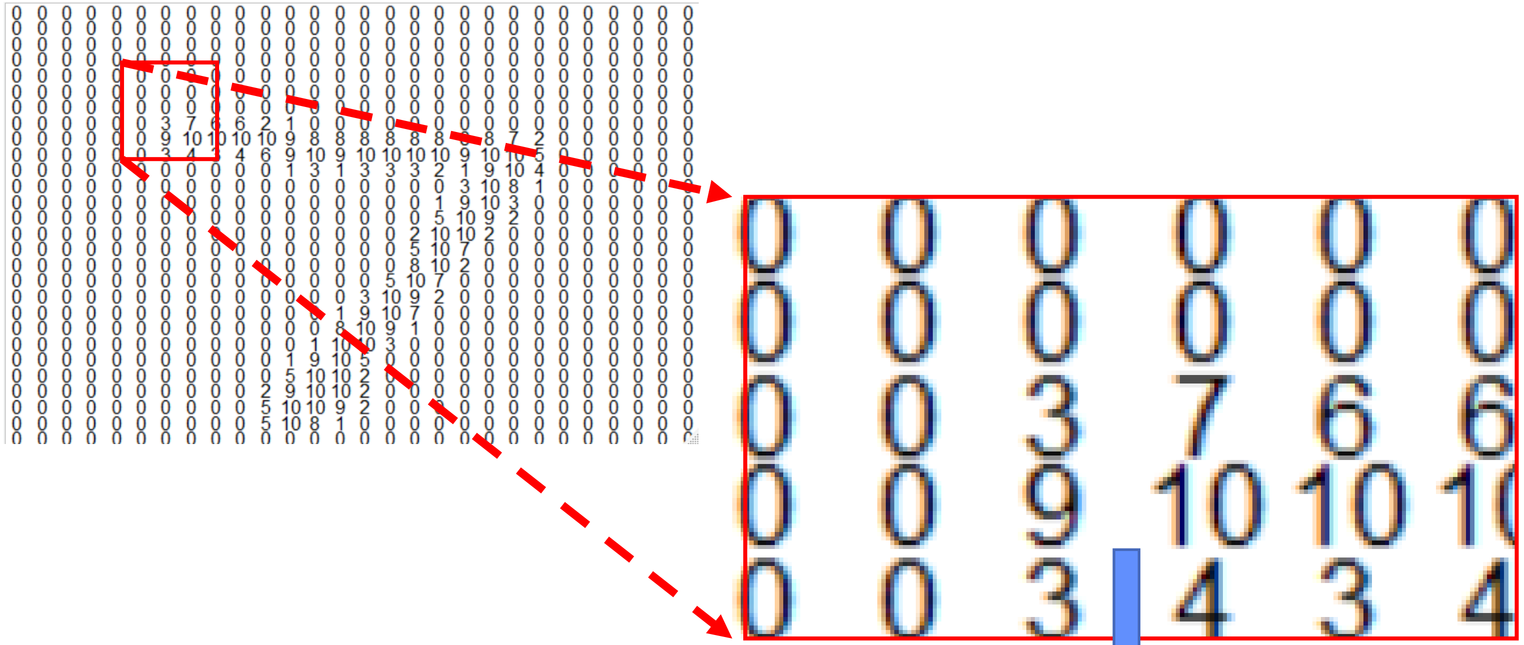
1. Multiply 3x3 patch of pixels with 3x3 filter "**W**"

2. Put answer in new cell of output map

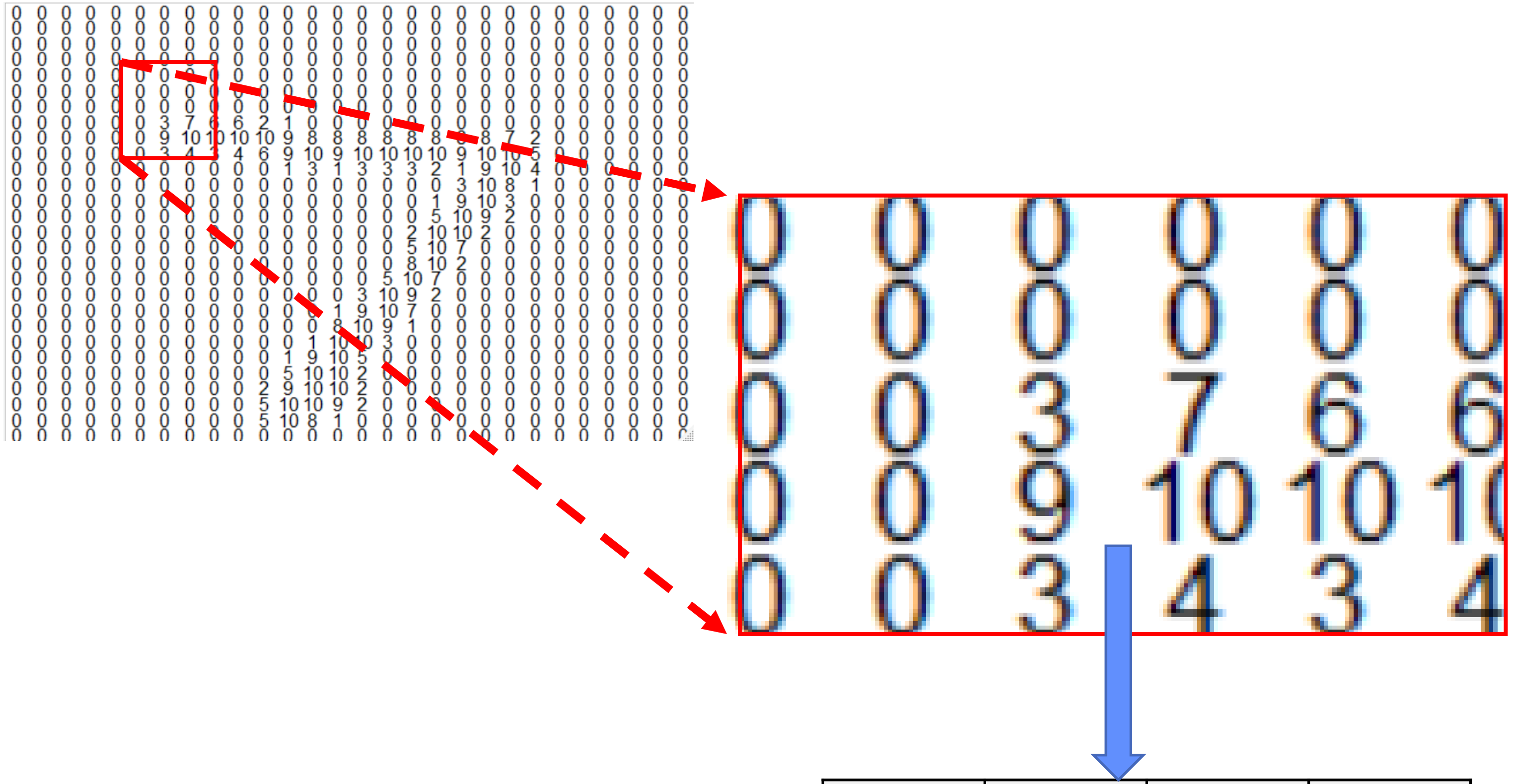


NOTE: sliding a filter is known as a "convolution" operation

After vertical and horizontal sliding the 5x6 patch is now a 3x5 ***feature map.***



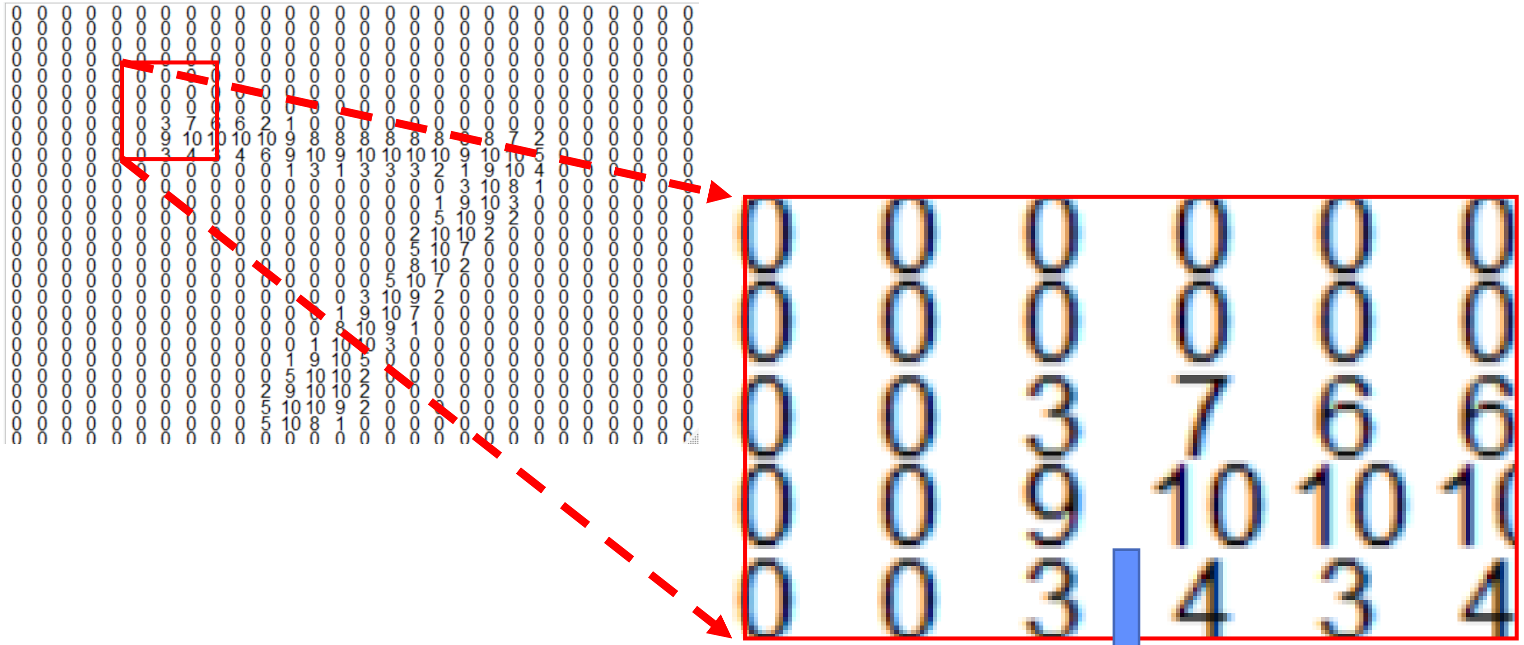
3	7	3	-1
12	17	4	-1
15	21	4	-1



3	7	3	-1
12	17	4	-1
15	21	4	-1

After vertical and horizontal sliding the 5x6 patch is now a 3x5 ***feature map.***

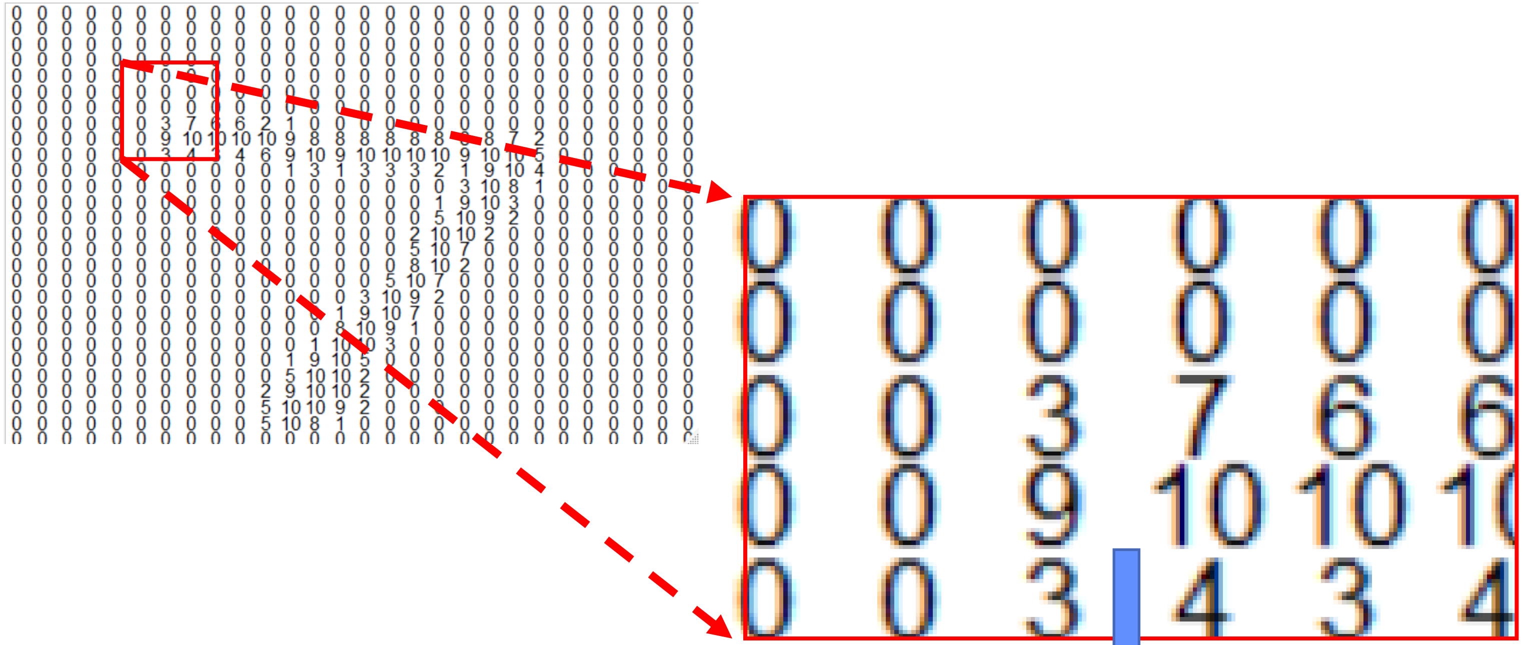
What do the highest values in the feature map represent?



3	7	3	-1
12	17	4	-1
15	21	4	-1

Optional next step:

Use another filter, and take maximum over elements - “max pooling”



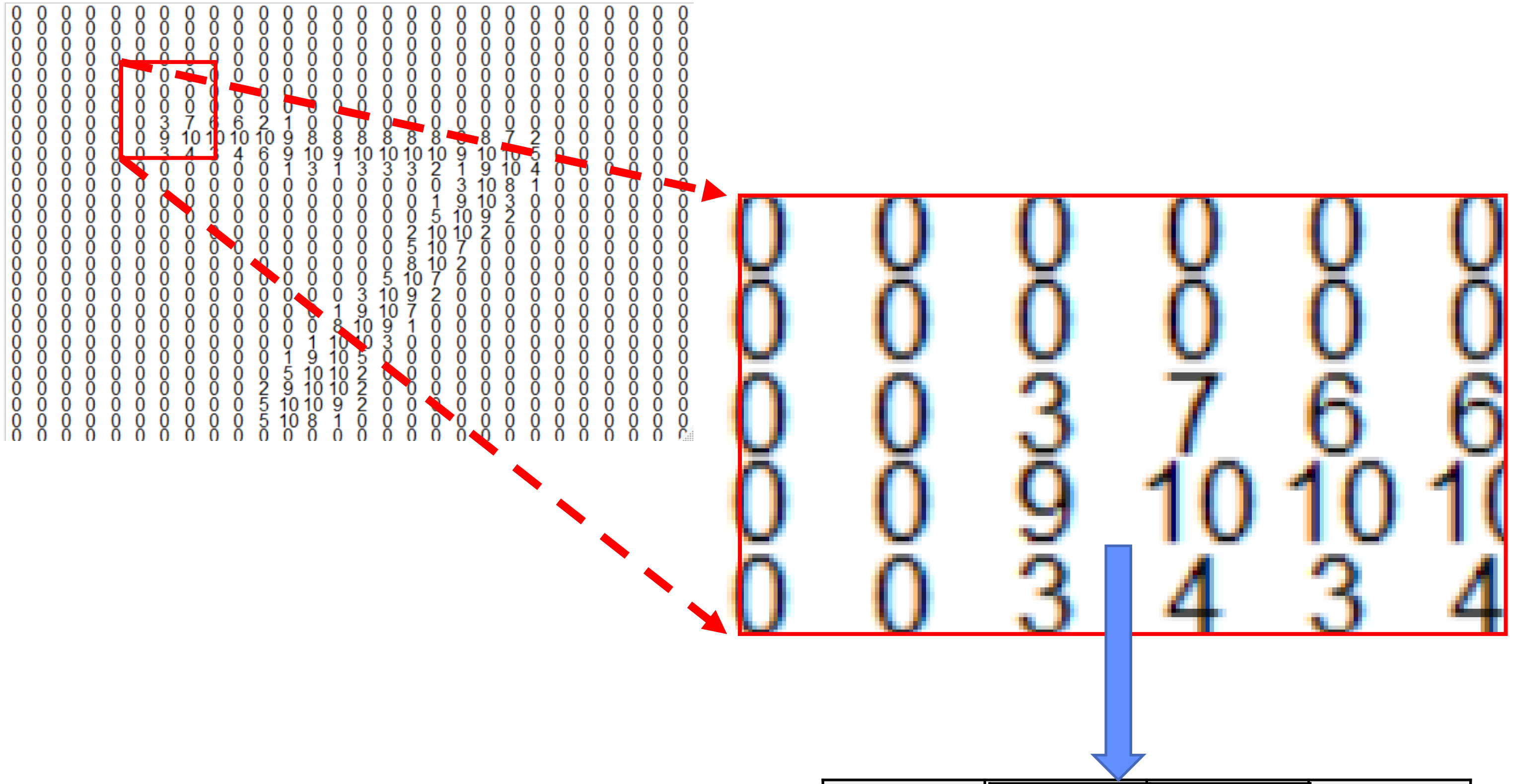
3	7	3	-1
12	17	4	-1
15	21	4	-1

Optional next step:

Use another filter, and take maximum over elements - “max pooling”

2x2 filter has max=17

17		



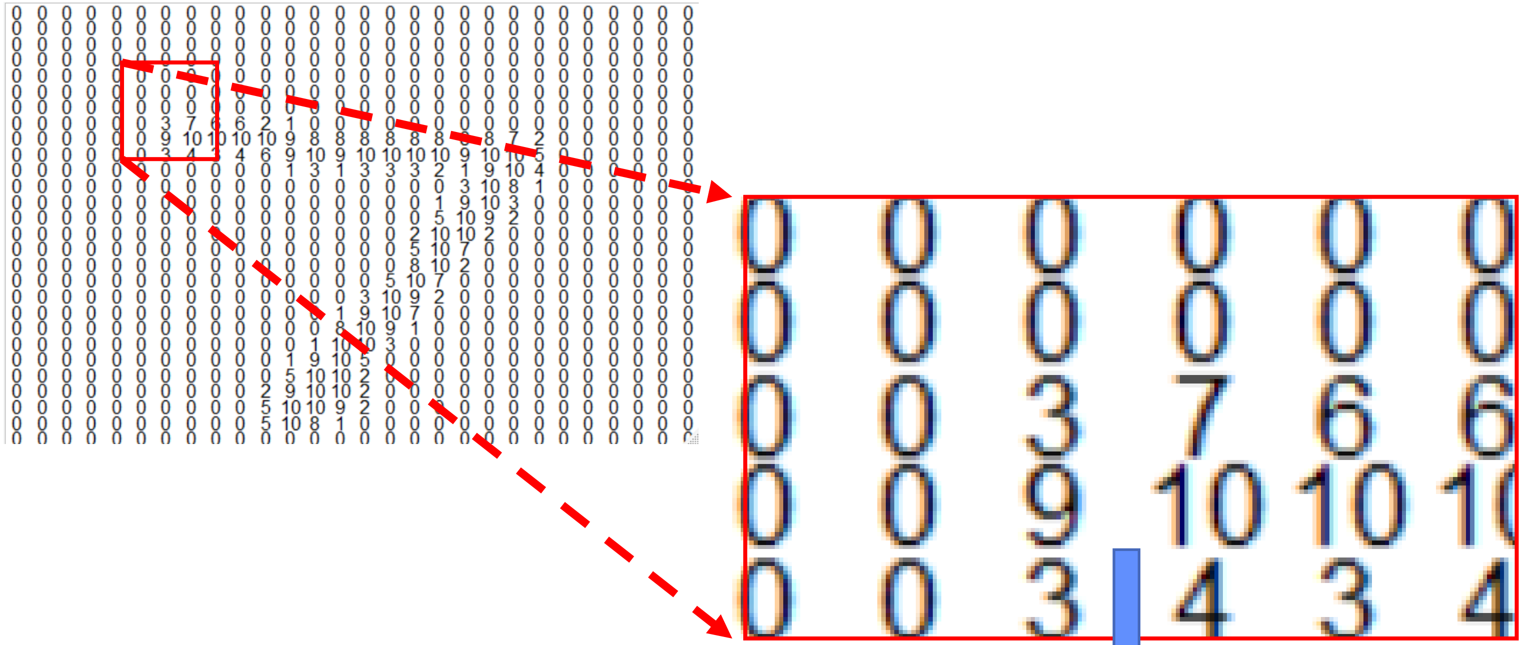
3	7	3	-1
12	17	4	-1
15	21	4	-1

Optional next step:

Use another filter, and take maximum over elements - “max pooling”

Slide filter …

17	17	4
21	21	4



After convolution and pooling 5x6 patch is ***transformed*** *into a 2x3 feature map of 'edge gradients'*

3	7	3	-1
12	17	4	-1
15	21	4	-1

Slide filter …

17	17	4
21	21	4

Feature engineering

In Computer Vision there are many kinds of edge detectors and many ways to scale them

-1	0	+1
-1	0	+1
-1	0	+1

But building features is hard, so if you have enough data …

Convolution Neural Network (CNN)

In CNNs the filter values are weight parameters that are learned (**feature discovery**)

W_{11}	W_{12}	W_{13}
W_{21}	W_{22}	W_{23}
W_{31}	W_{32}	W_{33}

Convolution Neural Network (CNN)

In CNNs the filter values are weight parameters that are learned (**feature discovery**)

W_{11}	W_{12}	W_{13}
W_{21}	W_{22}	W_{23}
W_{31}	W_{32}	W_{33}

A convolution layer is a set of feature maps, where each map is derived from convolution of 1 filter with input

Convolution Neural Network (CNN)

More hyperparameters:

Size of filter (smaller is more general)

Convolution Neural Network (CNN)

More hyperparameters:

Size of filter (smaller is more general)

Number of pixels to slide over (1 or 2 is usually fine)

Convolution Neural Network (CNN)

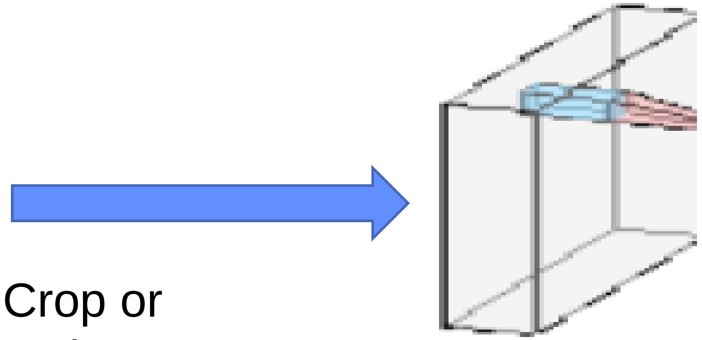
More hyperparameters:

- Size of filter (smaller is more general)
- Number of pixels to slide over (1 or 2 is usually fine)
- Number of filters (depends on the problem!)
- Max pooling or not (usually some pooling layers)

Convolution with image

- **Make 1 layer, using HxWx3 image (3 for Red,Green,Blue channels)**

Crop or resize image as needed

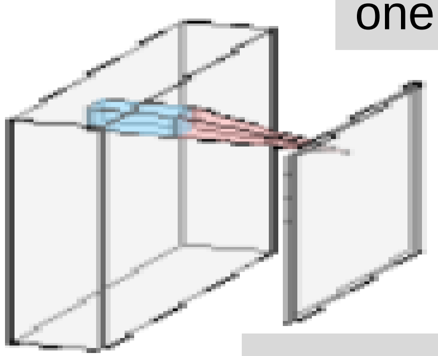


224x224x3

Convolution with image

- **Make 1 layer, using HxWx3 image (3 for RGB channels)**

Crop or resize image as needed



one 5x5x3 filter

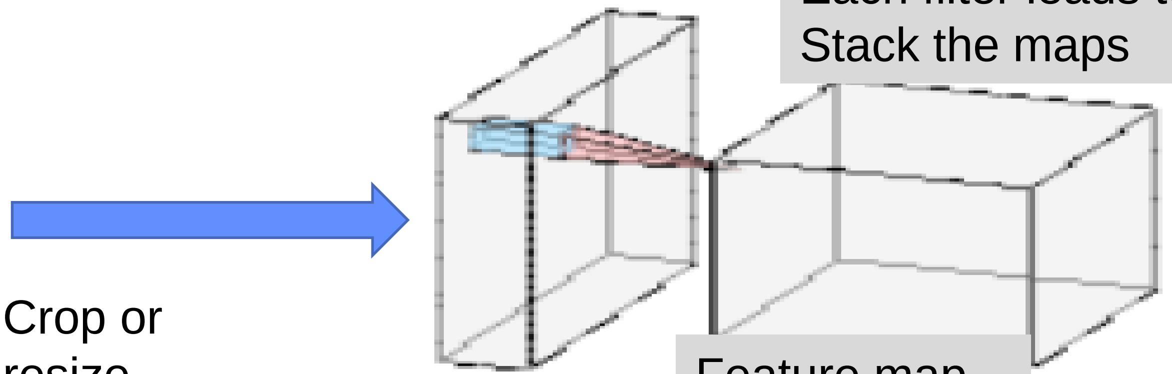
224x224x3

Feature map
Size depends on filter size and slide parameter

Convolution with image

- **Make 1 layer, using HxWx3 image (3 for RGB channels)**

Crop or resize image as needed



224x224x 3

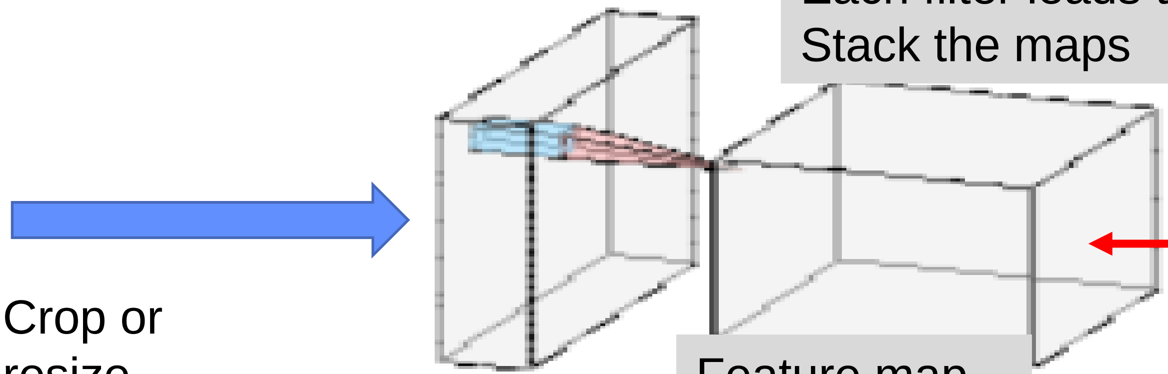
Repeat for many filters, all 5x5x3.
Each filter leads to 1 feature map
Stack the maps

Feature map
Size depends on filter size and slide parameter

Convolution with image

- **Make 1 layer, using HxWx3 image (3 for RGB channels)**

Crop or resize image as needed



224x224x3

Repeat for many filters, all 5x5x3.
Each filter leads to 1 feature map
Stack the maps

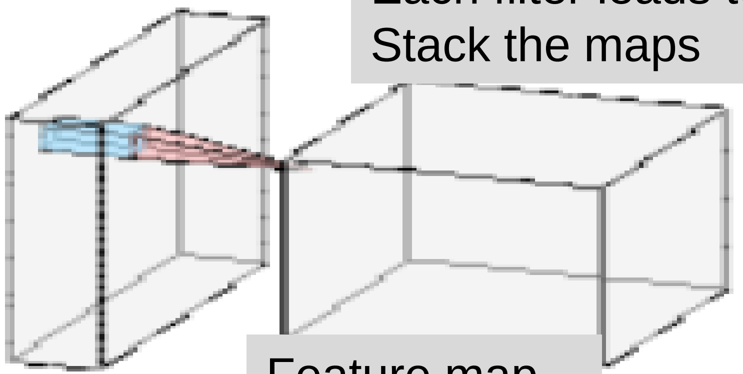
Feature map
Size depends on filter size and slide parameter

This is a "convolution layer"
The thickness is the number of different feature maps, sometimes called 'channels' or 'number of filters

Convolution with image

- **Make 1 layer, using HxWx3 image (3 for RGB channels)**

Crop or resize image as needed



224x224x
3

Repeat for many filters, all 5x5x3.
Each filter leads to 1 feature map
Stack the maps

Feature map
Size depends on filter size and slide parameter

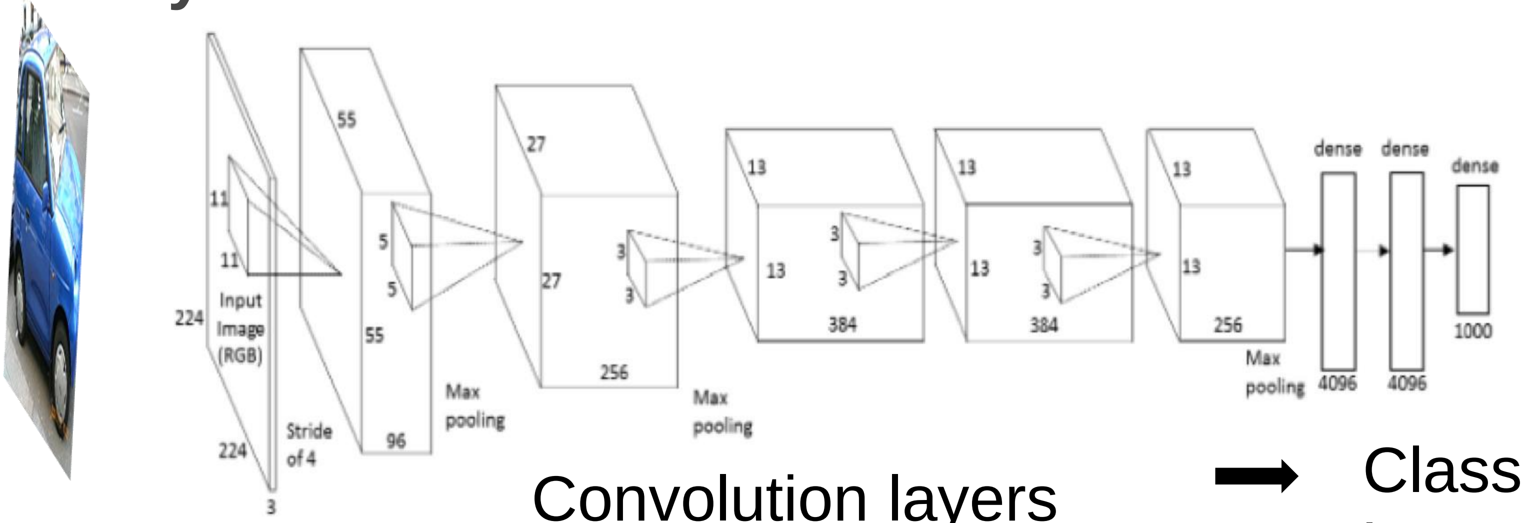
Add a layer of max-pool to down sample each feature map

OR

Add a next convolution layer

Large Scale Versions

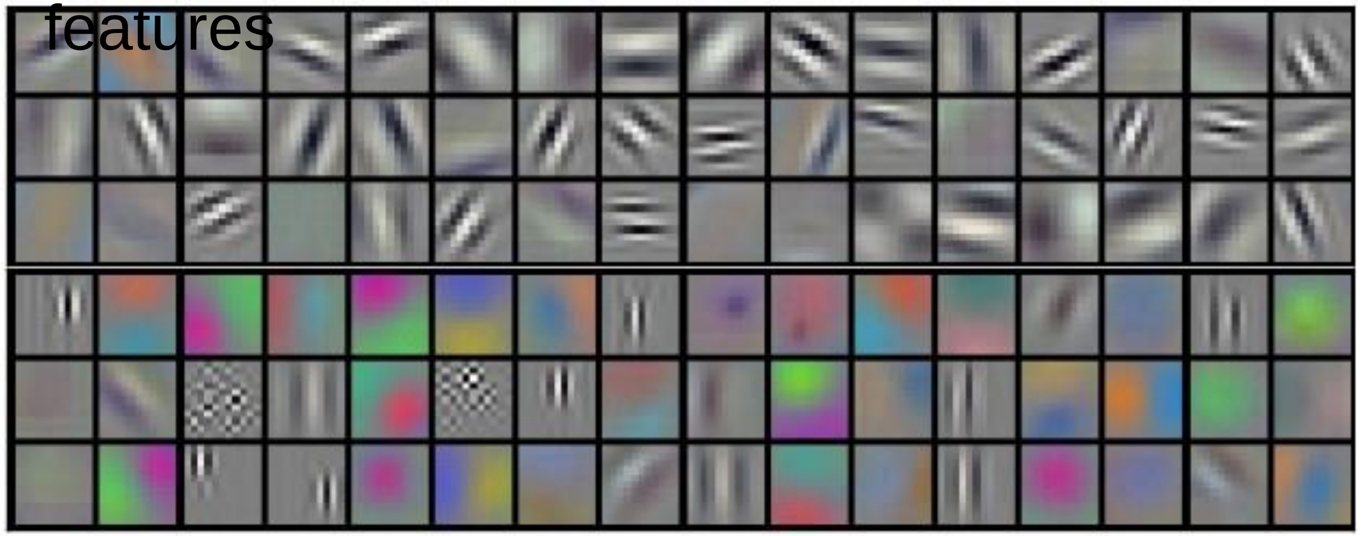
- Large (deep) Convolution Networks are turning out to be feasible with GPUs (some are 100+ layers)
- Need large amounts of data and many heuristics to avoid overfitting and increase efficiency



Convolution layers ➡ Classification layers and output

What Learned Convolutions Look Like

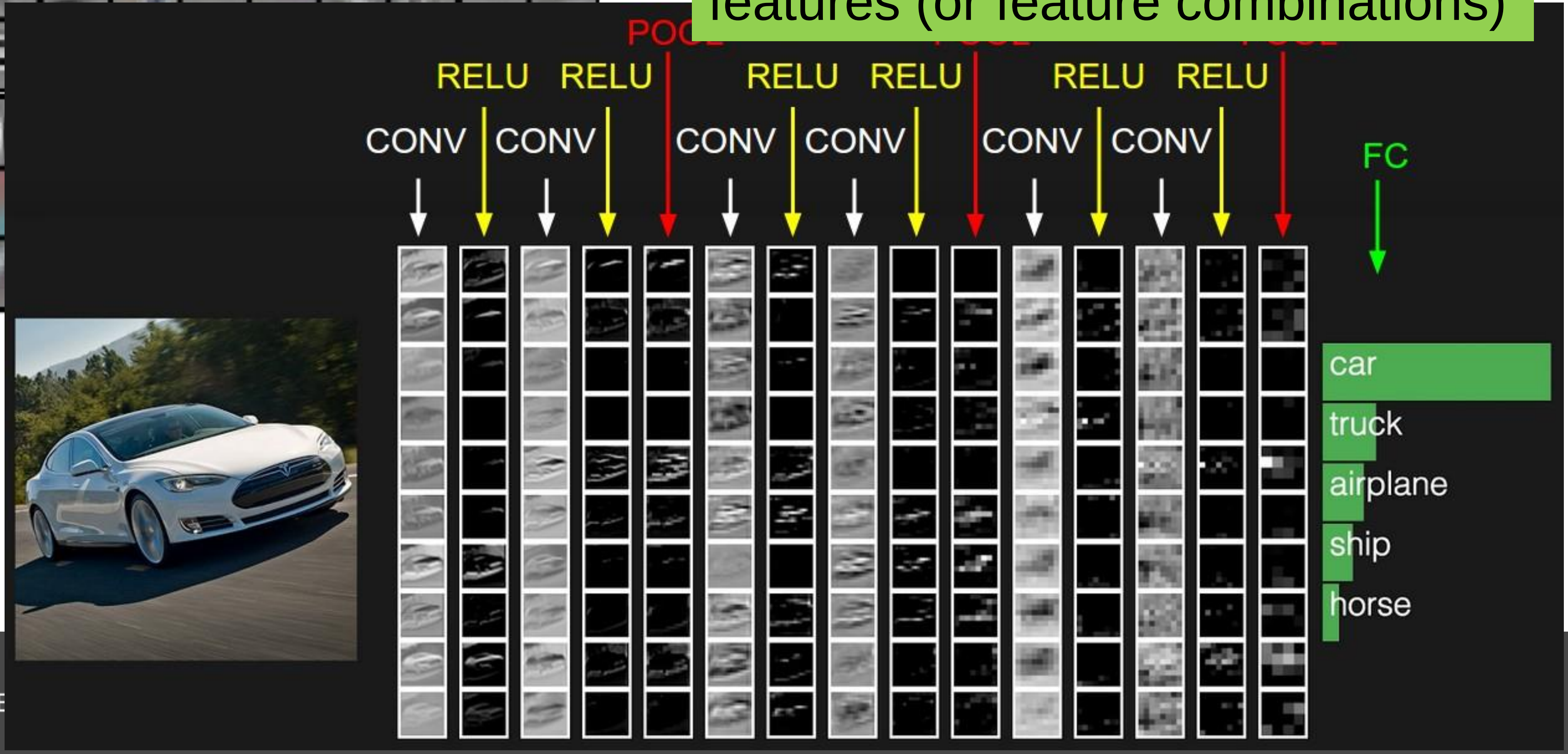
First convolution layer filters are simple features



What Learned Convolutions

First convolution layer filters are simple features

Higher layers are more abstract features (or feature combinations)



Convolution Neural Network Summary

CNNs works because convolution layers have a special architecture and function – it is biased to do certain kind of transformations

Low layers have less filters that represent simple local features for all classes

Higher layers have more filters that cover large regions that represent object class features

Outline

- Overview of Neural Networks (aka Multilayer Perceptron)
- Convolution Neural Networks
- Basic Keras commands for building, training CNNs - exercise
- Hyperparameters and Tuning and workflows - exercise

Exercise CNN for Digit Classification

- **The 'hello world' of CNNs**
- **Uses MNIST dataset and Keras**

intro_mnist_mlp2 Last Checkpoint: a few seconds ago (autosaved)

View Insert Cell Kernel Widgets Help

Run Code

```python
import warnings
warnings.filterwarnings("ignore")
import tensorflow as tf
tf.get_logger().setLevel('ERROR')

#load and prepare data
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train, x_test                      = x_train / 255.0, x_test / 255.0

#specify the neural network model and optimization
my_model   =  tf.keras.models.Sequential([
                      tf.keras.layers.Flatten(input_shape=(28, 28)),
                      tf.keras.layers.Dense(10)   ])
loss       =  tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
optimizer  =  tf.keras.optimizers.SGD(learning_rate=0.01)
my_model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])

#fit the model
fit_history= my_model.fit(x_train, y_train, epochs=5, batch_size=128)

#evaluate the fit
my_model.evaluate(x_test, y_test)
```

A basic model workflow in 4 steps

load/prepare data

define a model

fit a model

test the model

Zooming in on keras.models statements

Use a sequence of layers

Flatten image into vector

```
#specify the neural ne                              ation
my_model   =  tf.keras.models.Sequential([
                        tf.keras.layers.Flatten(input_shape=(28, 28)),
                        tf.keras.layers.Dense(10)  ])
```

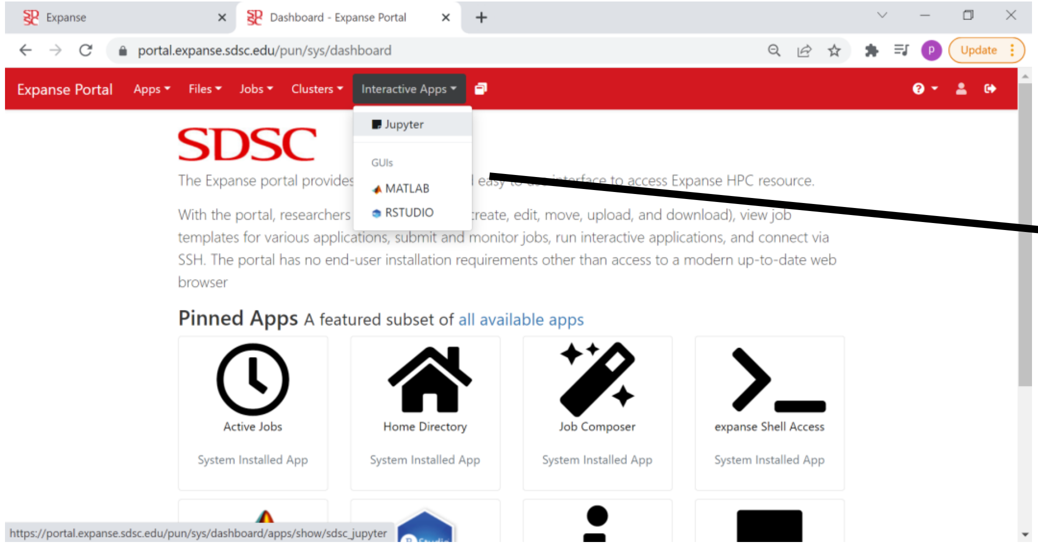
Create hidden layer that is fully connected to input

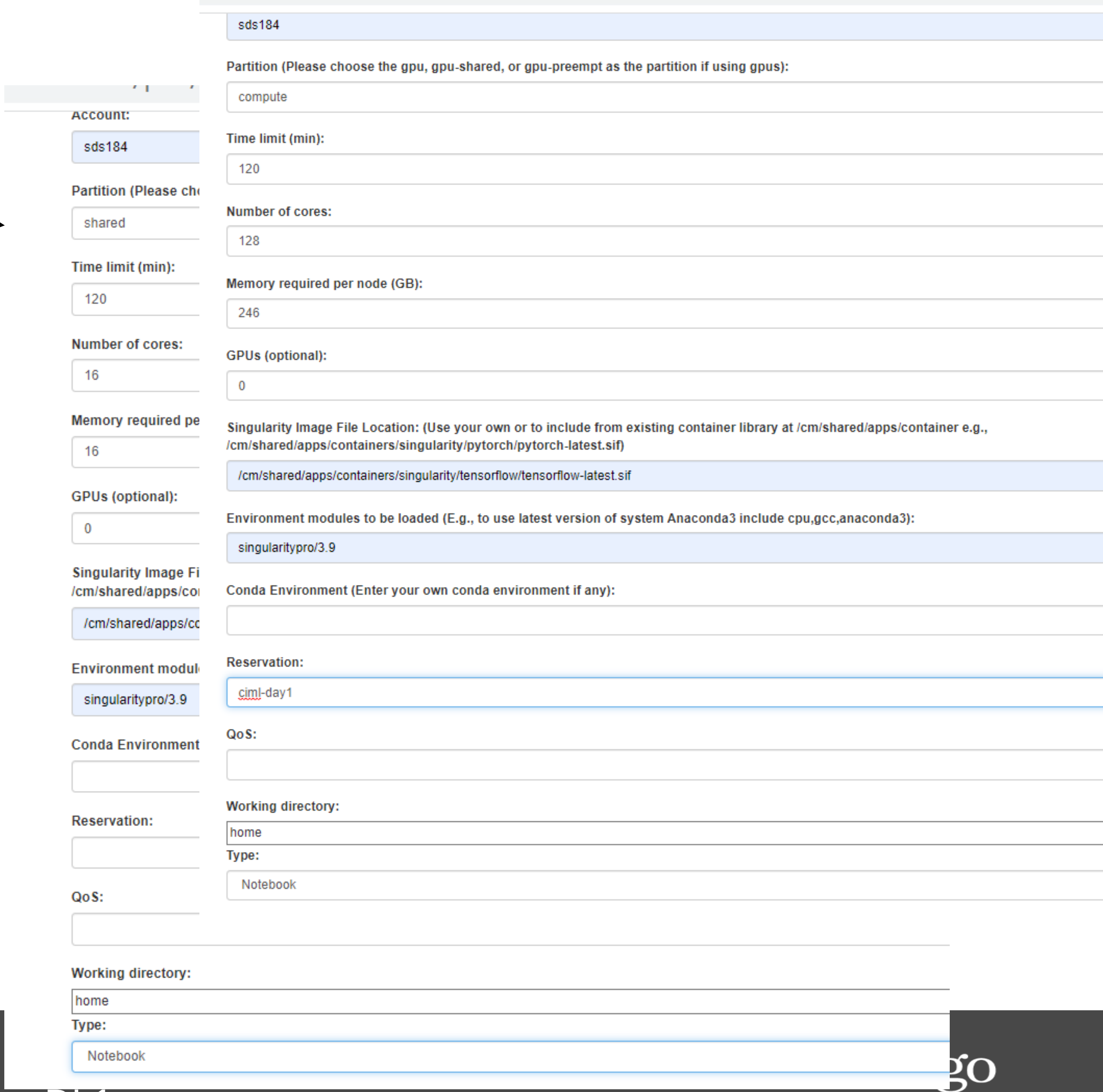
Zooming in on keras convolution layers statements

Use 16 filters, each of size 3x3

```
#  argument to Sequential
my_model.add(tf.keras.layers.Convolution2D(filters=16,
                                           kernel_size=(3, 3),
                                           strides=1,
                                           data_format="channels_last",
                                           activation='relu',
                                           input_shape=(28,28,1)))
```

Input shape does not include number of images

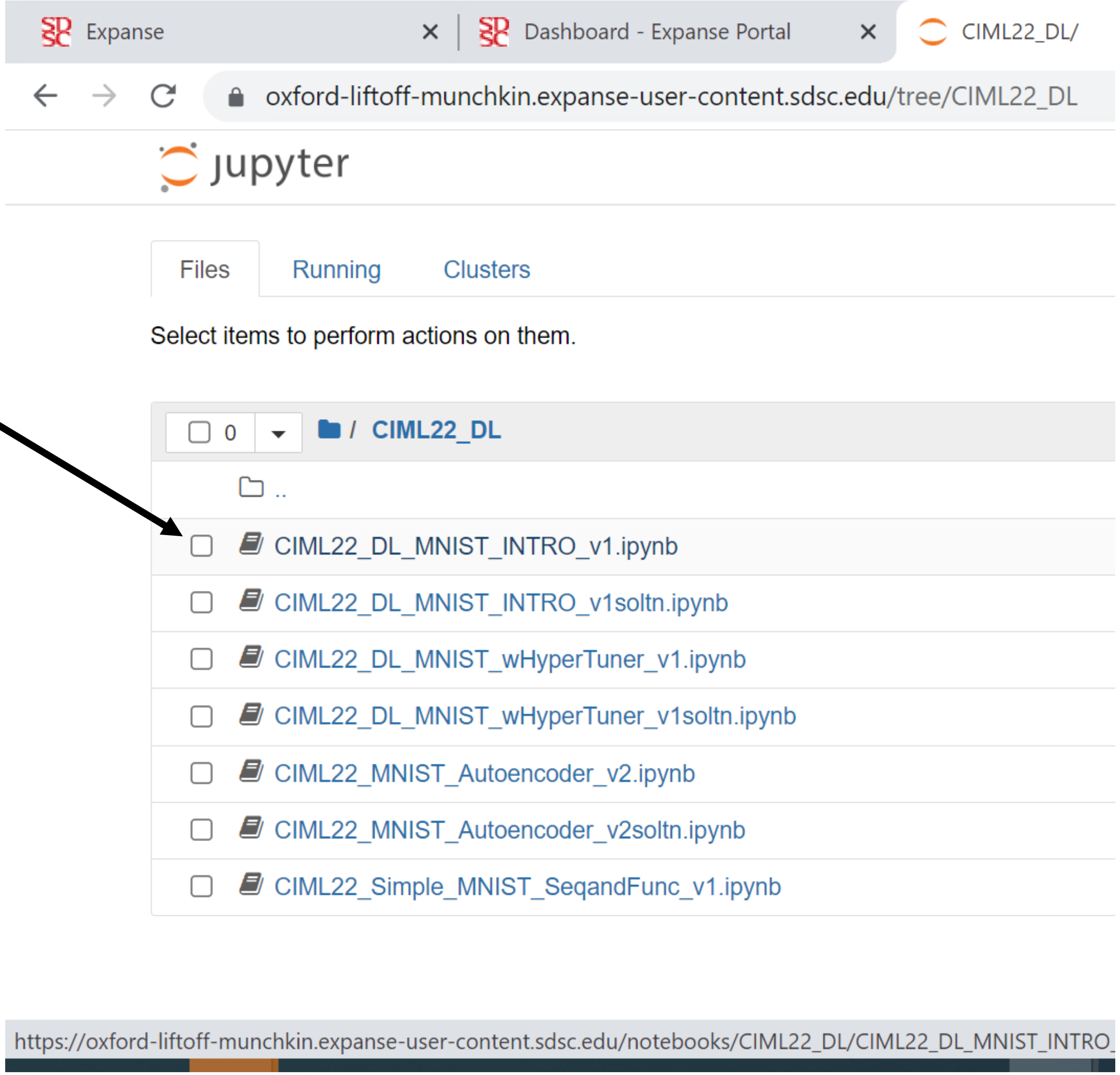




Expanse user portal: launch a jupyter notebook session (with compute or shared is OK)

In jupyter notebook session open the MNIST_Intro notebook

Follow instructions in the notebook



Outline

- Overview of Neural Networks (aka Multilayer Perceptron)
- Convolution Neural Networks
- Basic Keras commands for building, training CNNs - exercise
- Hyperparameters and Tuning and workflows - exercise

Things to think about for running a project

- Choosing Hyperparameters – a bit of exploration and exploitation
- Need to figure out efficient Job workflow
- On HPC, CPU work fine for many cases, you will want to use GPUs for ‘large’ models and/or large datasets.
- Model saves and/or checkpoints are available in tensorflow; tensorboard available but needs to be secure (ask for details)

Choosing Hyperparameters

- Generally:
  - architecture (layers, units, activation, filters, …)
  - algorithm (learning rate, optimizer, epochs, …)
  - efficient learning (batch size, normalization, initialization, …)

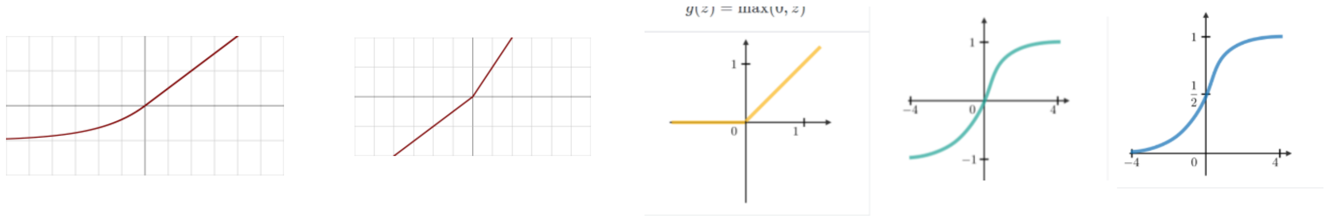
- Some options are determined by task:
  - loss function, CNN vs MLP, …

- Use what works, from related work or the latest recommendations,

Some general recommendations (with caveats)

- **From Hands-On Machine Learning, Keras, and TensorFlow, 2nd Ed, 2019, Géron,**

pg 338 Activations: *Scaled-ELU > ELU > leaky ReLU > ReLU > tanh > logistic”*



pg 359 Optimizer tradeoffs speed vs quality: * is bad * is good**

Table 11-2. Optimizer comparison

Class	Convergence speed	Convergence quality
SGD	*	***
SGD(momentum=...)	**	***
SGD(momentum=..., nesterov=True)	**	***
Adagrad	***	* (stops too early)
RMSprop	***	** or ***
Adam	***	** or ***
Nadam	***	** or ***
AdaMax	***	** or ***

Hyperparameters Search

- Can take a long time, hard to find global optimal
- Start with small data, short runs to get sense of range of good parameter values
- Easy but possibly time-consuming method:

  grid search over uniformly spaced values
- Do "exploration" then "exploitation", search wide then search deep

  Keras Tuner functions can help with the wide search

Hyperparameter Search Tool

- Keras Hypertuner class implements several search strategies:

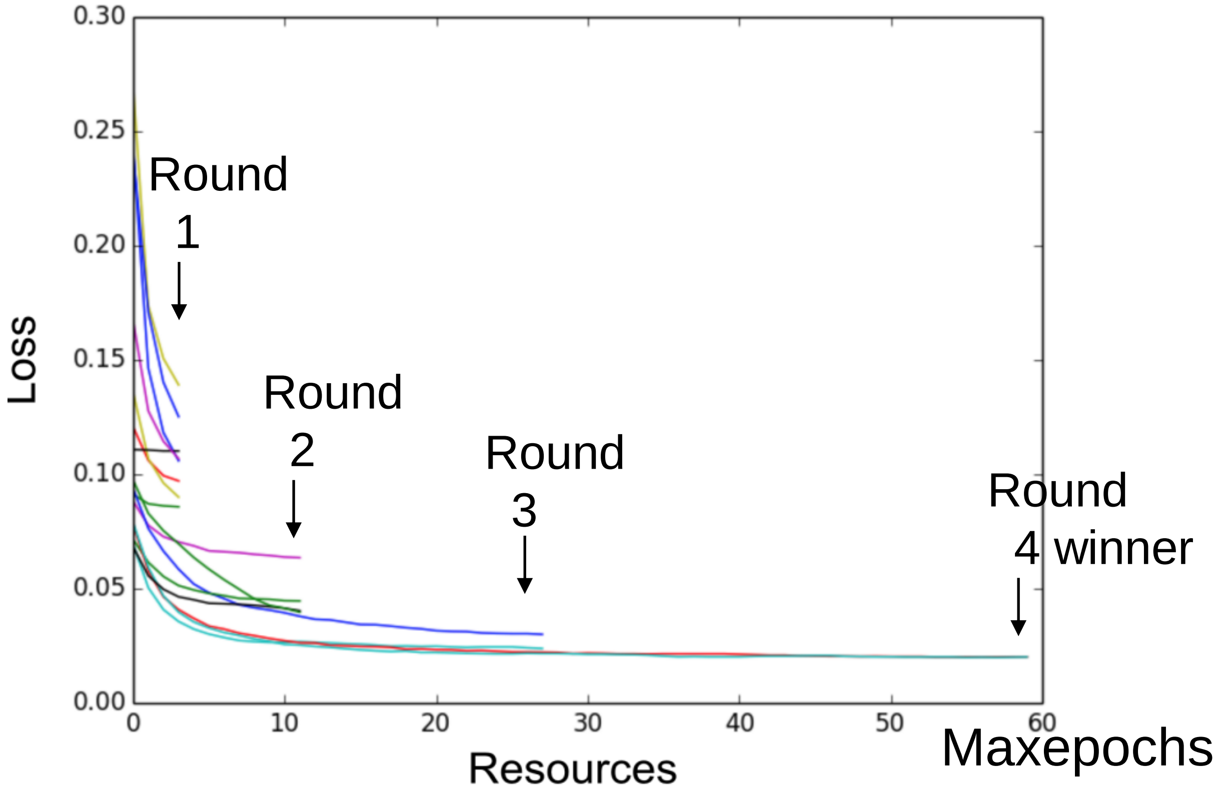
Hyperband is like a tournament competition of hyperparameter configurations, with incremental training, to weed out worse ones

RandomSearch will search randomly through the space of configurations and try to find better regions

Bayesian optimization is like function approximation to pick out next configuration

Hyperband Bracket

Each round runs several network configurations for small number of epochs
Several rounds with increasing epochs make up a bracket
Several brackets are run to end up with several possible overall winners.



Note, you could run a small grid search around hyperband winners to confirm performance

Keras Tuner code snippet

Set up function to make the model

```
def build_model_hp(hp):
    hp_numfilters     = hp.Int('hpnumfilters',min_value=8,max_value=32,step=4)
    #your variable name          ^^^ the parameter name in the hp object
```

Set up hyperparameter choices

```
def build_model(numfilters,activation_choice):   #<<------add code: if yo
                                                 #        list and change code t

    mymodel = keras.models.Sequential()
    mymodel.add(keras.layers.Convolution2D(numfilters,
                                           (3, 3),
                                           strides=1
```

Define 'tuner' object; tuner search uses the model fit function

```
tuner = kt.Hyperband(build_model_hp,
                     objective  = 'val_accuracy',
                     max_epochs = num_max_epochs,
                     factor     = 3,
                     hyperband_iterations=10,
                     directory   =dirname,
                     overwrite   =True,    #overwrite directo
                     project_name='hyperbandtest',
                     executions_per_trial=5,  #to try severa
                     seed         =777)
```

Workflow and Organizing Jobs

Job Level: What makes sense to include in each job?

Model Level: run & test model for each parameter configuration

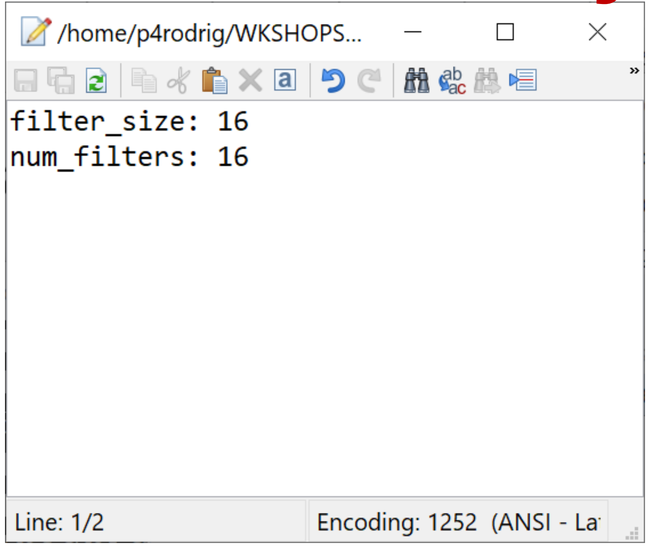
Data Level: loop through cross validation datasets (if applicable)

- **Consider how long each a model runs for 1 configuration of hyperparameters for 1 dataset**
- **Organize jobs into reasonable chunks of work**
- **For large models consider model-checkpoints**

Organizing Configurations – one way

Code snippet: using 'YAML' file to set up hyperparameter configuration

Create text file with "Parameter: Value" pairs



Read file as python dictionary

```python
import yaml

with open("./modelrun_args.yaml", "r") as f:
    my_yaml=yaml.safe_load(f)  #this returns a python dictionary

filter_size=my_yaml.get("filter_size")
num_filters=my_yaml.get("num_filters")
print('arguments, filter size:',filter_size,' num filters',num_filters)
```

Example slurm job script and execution for Expanse

```
#!/usr/bin/env bash
#SBATCH --job-name =mnist0522
#SBATCH --account=sds164
#SBATCH --partition=compute
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=128
#SBATCH --time=00:10:00
#SBATCH --output=myjoboutput.o%j.%N.out

module purge
module load singularitypro
module list

echo "filter_size: 3 " >  modelrun_args.yaml
echo "num_filters: 16 " >> modelrun_args.yaml

singularity exec --bind /expanse,/scratch  --nv \
  /cm/shared/apps/containers/singularity/tensorflow/tensorflow-latest.s
  python3 Intro_mnist_cnn2_forbatch.py > mymnist_stdoutput.txt
```

You could also modify or set up parameters; and save yaml files for each run

```
02 EXP-05192022]$
02 EXP-05192022]$ sbatch run-job-tf-compute.sbatch
 job 12502781
02 EXP-05192022]$ squeue --me
BID PARTITION     NAME     USER ST        TIME  NODES NODELIST(REASON)
781   compute =mnist05 p4rodrig PD        0:00      1 (Priority)
813   compute galyleo- p4rodrig  R       26:03      1 exp-13-06
02 EXP-05192022]$
```

Python Notebook vs Scripts

- **On HPC you may want to run batch jobs on a script not a notebook.**

1 Papermill is one tool (see PRose talk)

2 Or, you can use "*jupyter nbconvert --to script your-python.ipynb***" in the batch job.**

Also, turnoff plot display, save plots in files, and use a configuration file to pass in parameters

note on using GPU

- **GPU node has multiple GPU devices**
- **By default tensorflow will run on 0th gpu device if GPU is available, otherwise it will use all CPU cores**

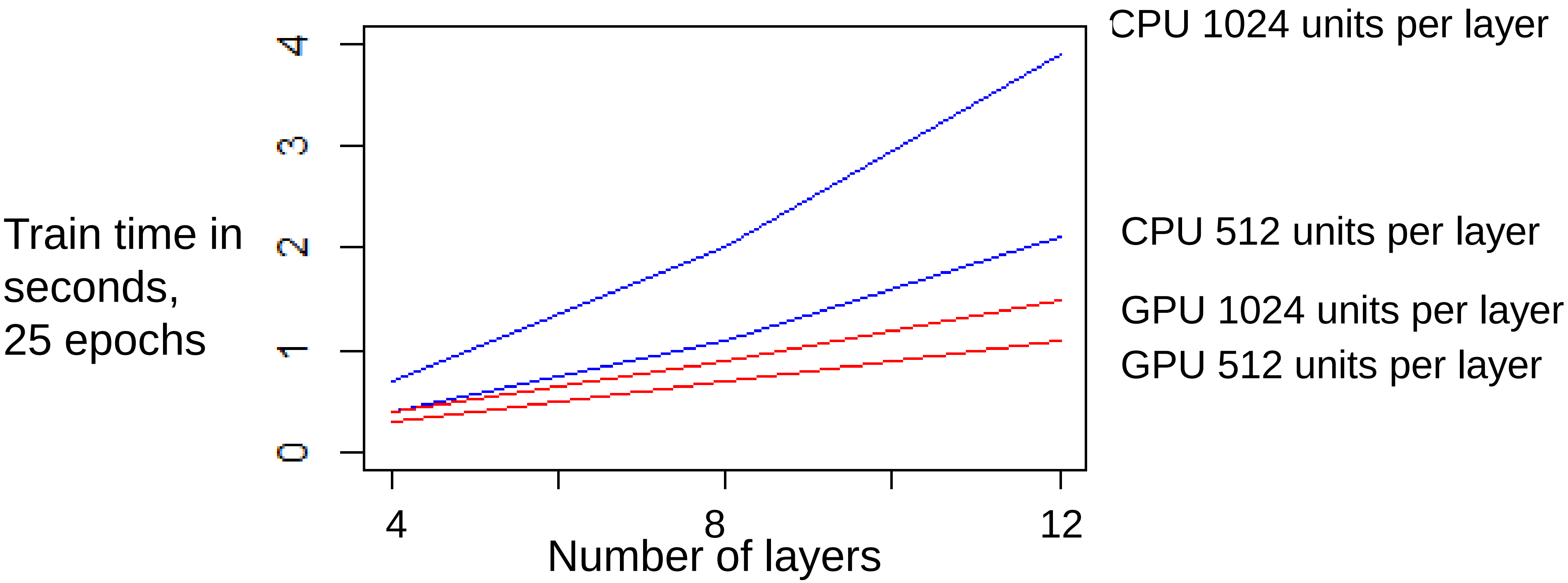
Code snippet to check for GPU devices

/home/p4rodrig/WKSHOPS/EXP-05192022/Intro_mnist_cnn2_forbatch.py - p4rodrig@login.expanse.sdsc.edu - Editor - W

Encoding Color

```
physical_devices = tf.config.list_physical_devices('GPU')
n_gpus           = len(physical_devices)
print("Info,config,Num GPUs:", n_gpus)
if tf.test.gpu_device_name():
  print('Info, test,Default GPU Device:{}'.format(tf.test.gpu_device_name()))
```

GPU shared (V100) vs CPU (128 cores) For MLP with Dense Layers, 80000x200 data matrix



GPUs faster, but you might have to wait more in job queue; also some memory limits compared to CPU

Where to go from here

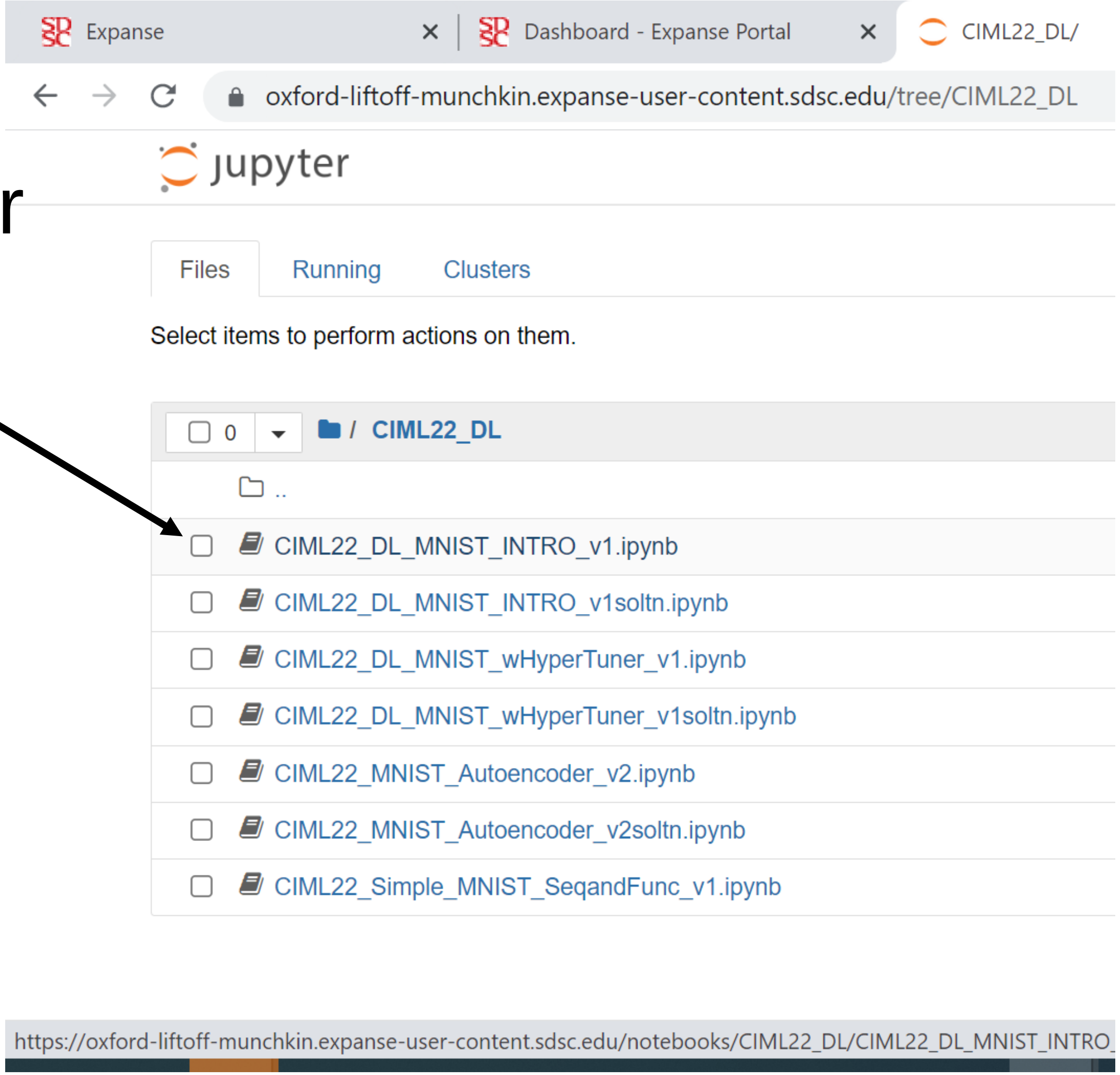
- Find relevant examples to your domain or task
- Tensorflow has many examples with tutorials in their documentation
- Tensorflow hub and model examples have code and pretrained models

https://tfhub.dev/google/imagenet/inception_v1/classification/4

https://keras.io/examples/

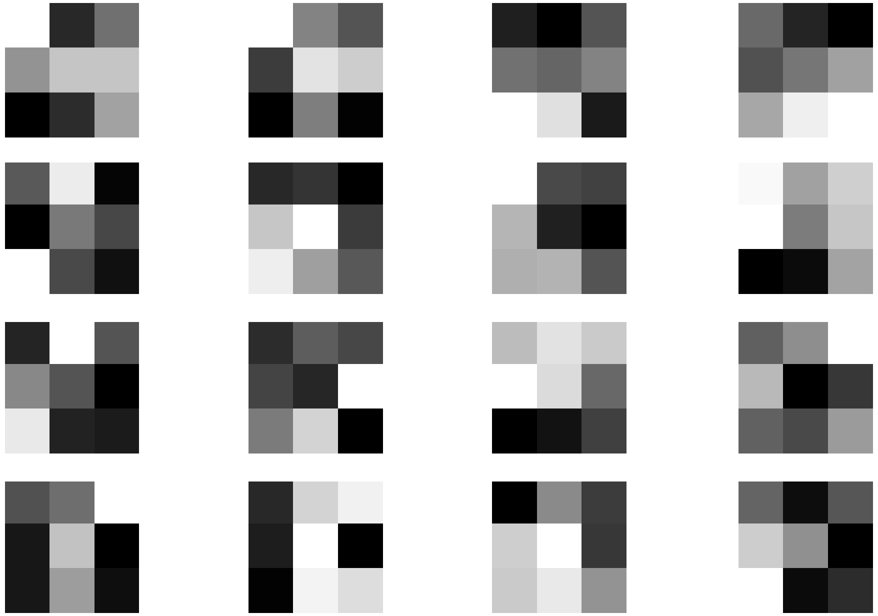
In jupyter notebook session open the MNIST_wHyperTuner notebook

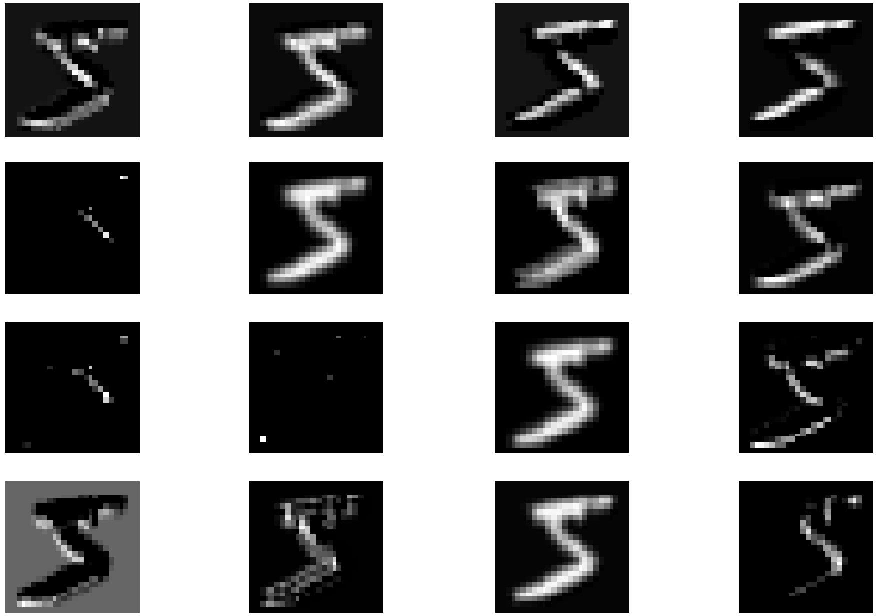
Follow instructions in the notebook



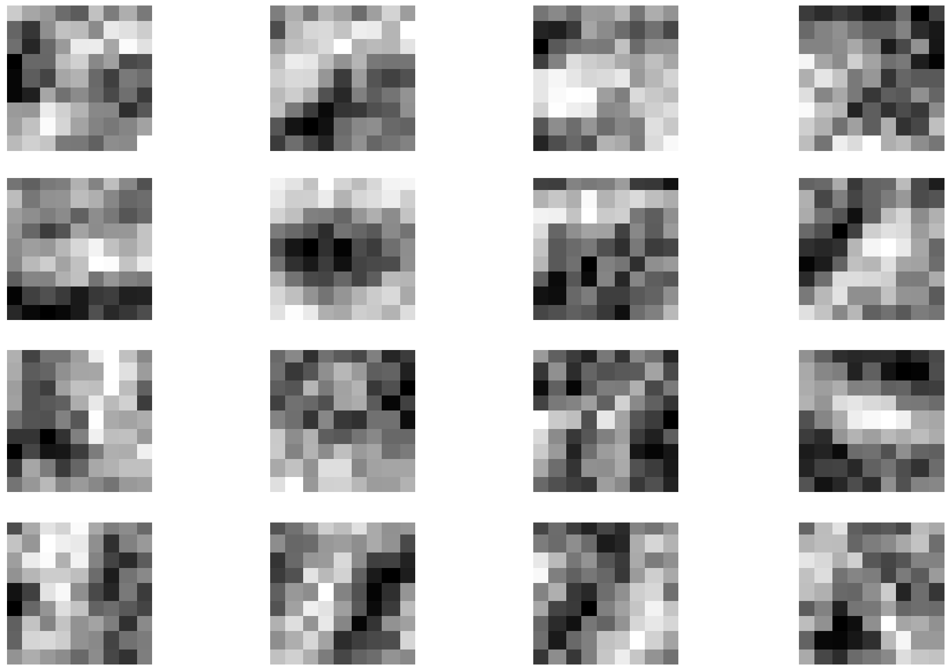
- **End**

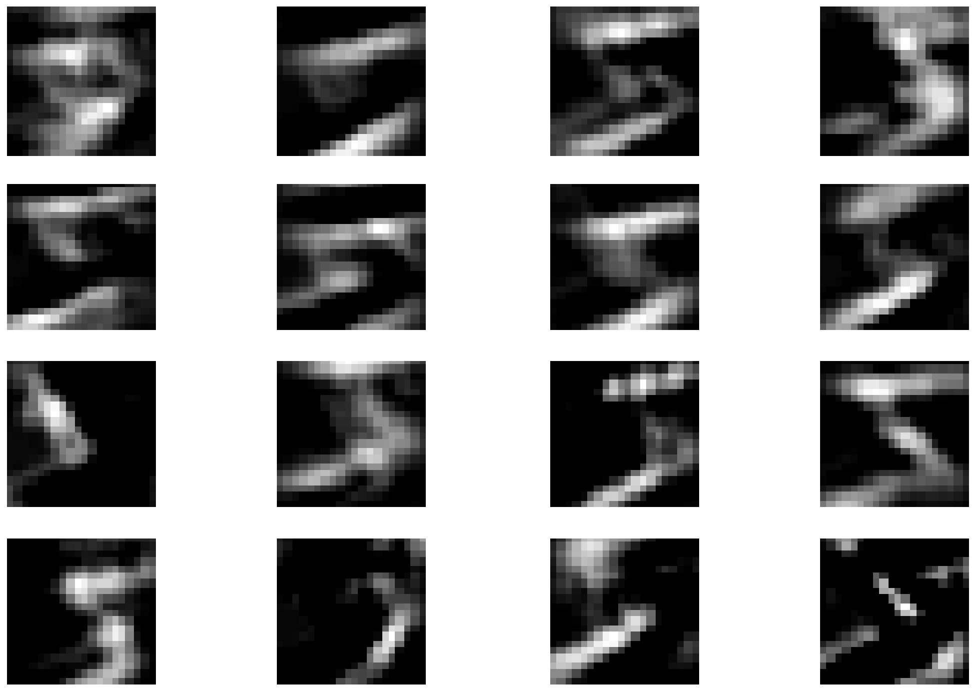
Exercise notes: 3x3 first convolution layer filter and activation





9x9 first convolution layer filter and activation





View Insert Cell Kernel Widgets Help

Run Code

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 26, 26, 16)        160
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 24, 24, 16)        2320
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 16)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_6 (Dense)              (None, 32)                73760
_________________________________________________________________
dense_7 (Dense)              (None, 10)                330
=================================================================
Total params: 76,570
Trainable params: 76,570
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
```

Filter_wts X num_filters + filter_bias:
(3x3x1) *16 + 16*1 = 9*16+16=160