CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Deep Learning

Deep Learning

Lecture: Convolutional Neural Networks

Ted Scully

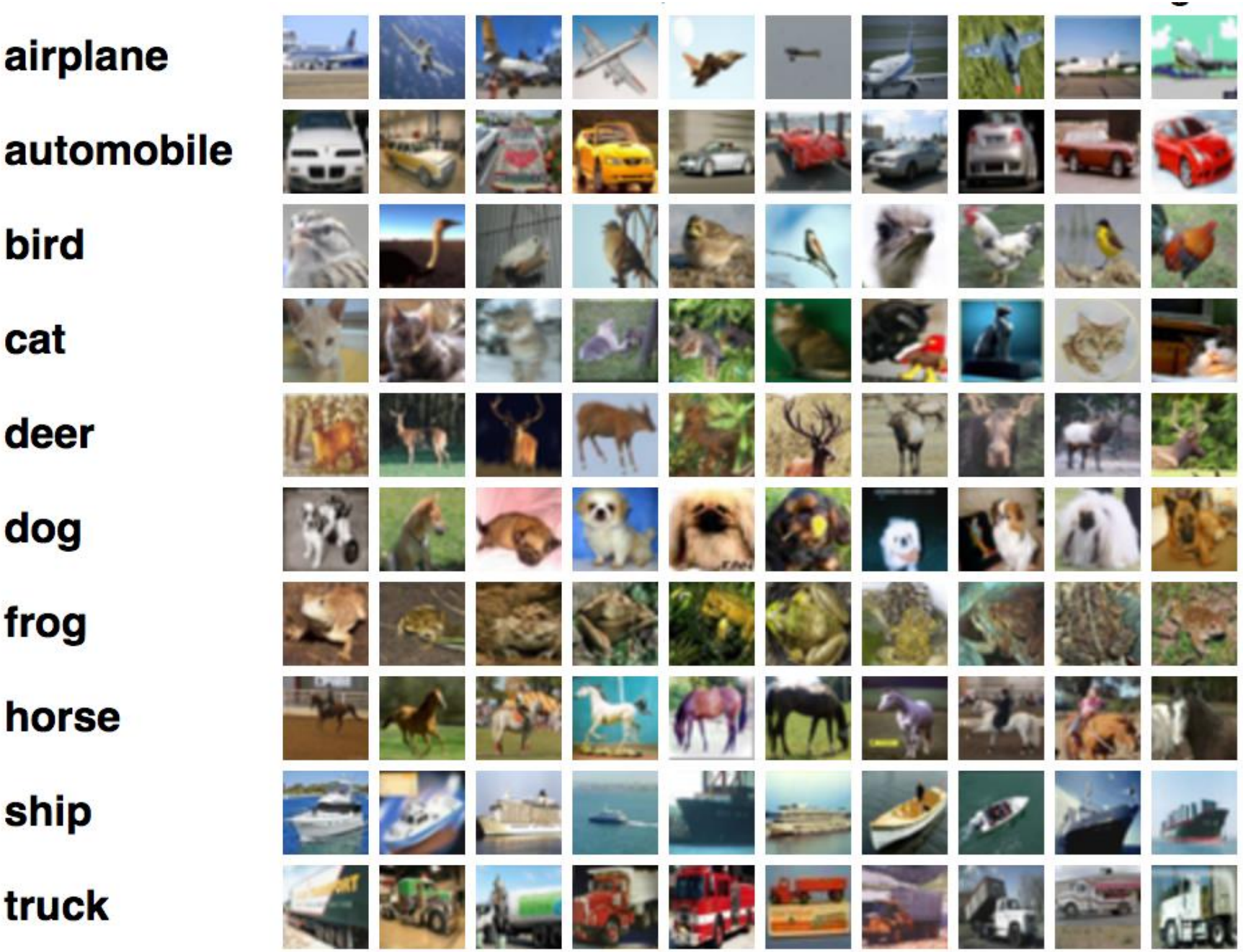
CIFAR 10 Dataset

- As the name suggests, CIFAR-10 consists of 10 classes, including airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

- For a more challenging benchmark dataset, we can use **CIFAR-10**, a collection of **60,000, 32 × 32** RGB images, thus implying that each image in the dataset is represented by 32 × 32 × 3 = 3,072 integers.

- Each class is evenly represented with **6,000 images** per class.

- There are 50,000 training images and testing 10,000.


airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

CIFAR 10 Dataset

1. CIFAR-10 is substantially harder than the MNIST dataset.

2. The challenge comes from the **dramatic variance in how objects appear**. For example, we can no longer assume that an image containing a green pixel at a given (x, y)-coordinate is a frog. This pixel could be a background of a forest that contains a deer. Or it could be the color of a green car or truck.

3. These assumptions are a stark contrast to the MNIST dataset, where the network can learn assumptions regarding the **spatial distribution of pixel intensities**. For example, the spatial distribution of foreground pixels of a 1 is substantially different than a 0 or a 5.

4. This type of variance exhibited in object appearance in CIFAR10 makes applying a series of fully-connected layers much more challenging.

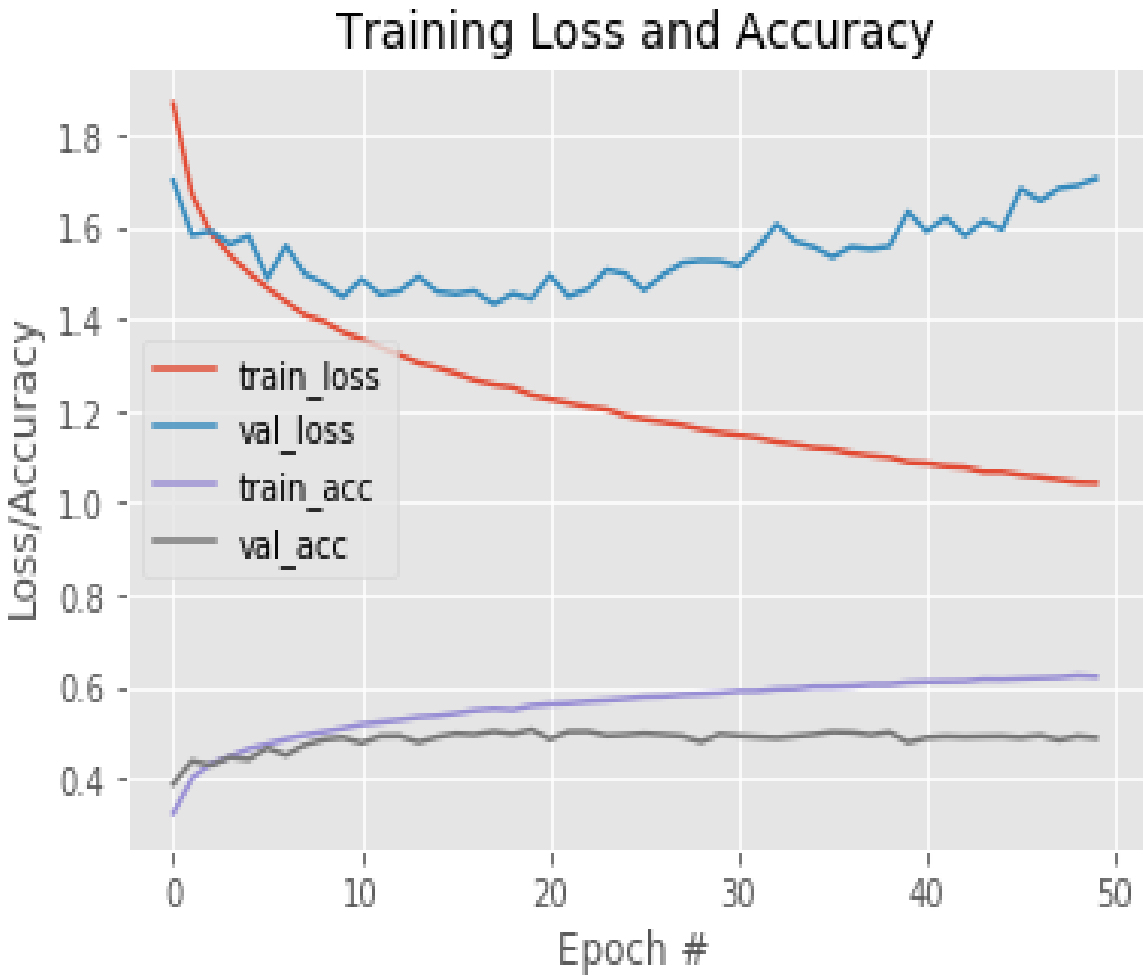
5. As you'll see in the following code, standard fully-connected layer networks are not suited for this type of image classification.

```
import tensorflow as tf
import matplotlib.pyplot as plt

num_epochs = 50
cifar = tf.keras.datasets.cifar10
(x_train, y_train),(x_test, y_test) = cifar.load_data()
x_train, x_test = x_train / 255.0, x_test / 255


model = tf.keras.models.Sequential([
    tf.layers.Flatten(input_shape=(32, 32, 3)),
    tf.keras.layers.Dense(1024, activation=tf.nn.relu),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(x_train, y_train, epochs=num_epochs,  validation_data=(x_test, y_test))

plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, num_epochs), history.history["loss"], label="train_loss")
plt.plot(np.arange(0, num_epochs), history.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, num_epochs), history.history["acc"], label="train_acc")
plt.plot(np.arange(0, num_epochs), history.history["val_acc"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend
```

Clearly the network we have trained on the previous slide is badly overfitting on the training data.

We could consider attempting to optimize our hyperparameters further, in particular, experimenting with varying learning rates, regularization rates and increasing both the depth and the number of nodes in the network. However, the impact would minimal.

The reality is that basic feedforward network with strictly fully-connected layers are not suitable for challenging image datasets. For that, we need a more advanced approach: Convolutional Neural Networks.



CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Deep Learning

Deep Learning

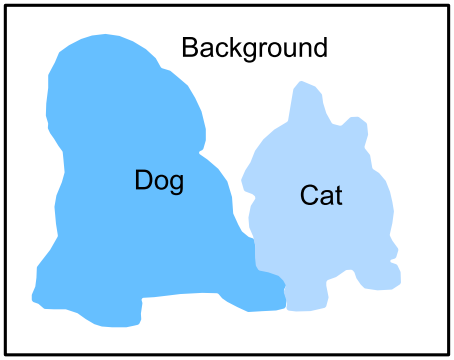
Lecture: Convolutional Networks

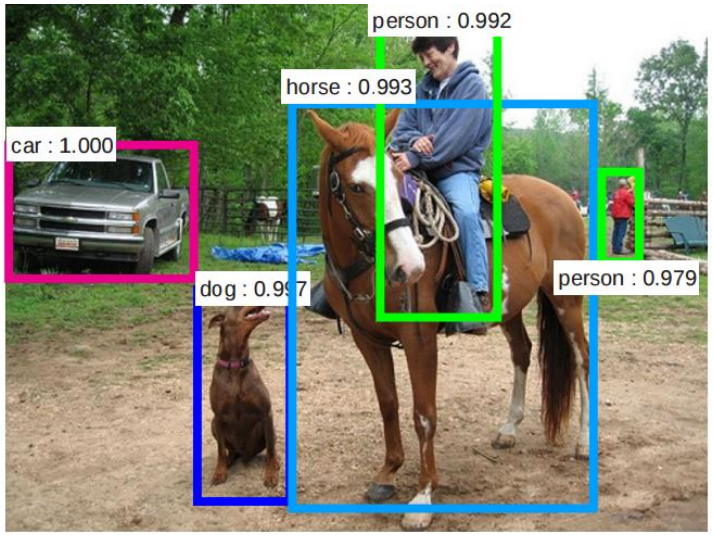
Ted Scully

Convolutional Neural Networks

- Convolutional neural networks (ConvNets) are a type of deep neural network that has been very successfully applied to image classification problems.

- Application areas include:
  - Image classification
  - Semantic segmentation
  - Object localization and detection
  - Instance segmentation





Challenges in Dealing with Images

151	121	1	93	165	204	14	214	28	235
62	67	17	234	27	1	221	37	189	141
20	168	155	113	178	228	25	130	139	221
236	136	158	230	10	5	165	17	30	155
174	148	93	70	95	106	151	10	160	214
103	126	58	16	138	136	98	202	42	233
235	103	52	37	94	104	173	86	223	113
212	15	179	139	48	232	194	46	174	37
119	81	241	172	95	170	29	210	22	194
129	19	33	253	229	5	152	233	52	44
88	200	194	185	140	200	223	190	164	102
113	16	220	215	143	104	247	29	97	203
9	210	102	246	75	9	158	104	184	129
124	52	76	148	249	107	65	216	187	181
6	251	52	208	46	65	185	38	77	240
150	194	28	206	148	197	208	28	74	93
33	183	248	153	168	205	146	100	254	218
130	53	128	212	61	226	201	110	140	183
165	246	22	102	151	213	40	138	8	93
152	251	101	230	23	162	70	238	75	24
187	105	152	83	167	98	125	180	136	121
139	197	55	209	28	124	208	208	104	40
123	19	144	223	62	253	202	108	47	242
220	144	31	16	136	123	227	62	183	163

29	142	142	75	22	109	111	28	6	5
137	168	41	206	100	70	219	127	114	191
205	154	226	14	89	86	242	67	203	15
247	47	128	123	253	229	181	251	232	28
68	75	24	99	93	63	215	222	102	180
206	246	85	103	215	3	62	64	77	216
126	80	165	149	196	75	186	60	179	193
44	253	164	253	14	216	175	30	46	254
137	23	33	203	241	21	144	63	244	188
32	214	142	121	249	109	99	232	183	71
45	36	152	27	190	137	61	1	237	247
1	14	241	70	2	30	151	67	169	205
32	80	102	32	99	169	91	166	73	214
186	219	9	203	209	240	40	249	119	122
177	252	38	203	119	0	217	139	139	157
154	145	49	251	150	185	235	23	230	156
157	168	223	60	247	118	5	180	16	206
102	208	195	246	140	138	54	191	139	79
17	233	85	169	166	24	49	40	160	97
84	242	247	144	203	3	19	24	198	88
67	67	185	98	123	106	168	105	127	153
37	113	214	252	203	80	146	211	7	16
142	241	66	86	214	133	146	253	189	200
67	215	174	111	189	54	144	56	59	163

Challenges in Dealing with Images

- While **humans can easily differentiate** between objects in the two images a computer will just see a matrix of numbers (pixel values).
- The difference between how a human perceives the content of an image versus how an image can be represented in a way a computer can understand the process is often referred to as the **semantic gap.**

151	121	1	93	165	204	14	214	28	235
62	67	17	234	27	1	221	37	189	141
20	168	155	113	178	228	25	130	139	221
236	136	158	230	10	5	165	17	30	155
174	148	93	70	95	106	151	10	160	214
103	126	58	16	138	136	98	202	42	233
235	103	52	37	94	104	173	86	223	113
212	15	179	139	48	232	194	46	174	37
119	81	241	172	95	170	29	210	22	194
129	19	33	253	229	5	152	233	52	44
88	200	194	185	140	200	223	190	164	102
113	16	220	215	143	104	247	29	97	203
9	210	102	246	75	9	158	104	184	129
124	52	76	148	249	107	65	216	187	181
6	251	52	208	46	65	185	38	77	240
150	194	28	206	148	197	208	28	74	93
33	183	248	153	168	205	146	100	254	218
130	53	128	212	61	226	201	110	140	183
165	246	22	102	151	213	40	138	8	93
152	251	101	230	23	162	70	238	75	24
187	105	152	83	167	98	125	180	136	121
139	197	55	209	28	124	208	208	104	40
123	19	144	223	62	253	202	108	47	242
220	144	31	16	136	123	227	62	183	163

29	142	142	75	22	109	111	28	6	5
137	168	41	206	100	70	219	127	114	191
205	154	226	14	89	86	242	67	203	15
247	47	128	123	253	229	181	251	232	28
68	75	24	99	93	63	215	222	102	180
206	246	85	103	215	3	62	64	77	216
126	80	165	149	196	75	186	60	179	193
44	253	164	253	14	216	175	30	46	254
137	23	33	203	241	21	144	63	244	188
32	214	142	121	249	109	99	232	103	71
45	36	152	27	190	137	61	1	237	247
1	14	241	70	2	30	151	67	169	205
32	80	102	32	99	169	91	166	73	214
186	219	9	203	209	240	40	249	119	122
177	252	38	203	119	0	217	139	139	157
154	145	49	251	150	185	235	23	230	156
157	168	223	60	247	118	5	180	16	206
102	208	195	246	140	138	54	191	139	79
17	233	85	169	166	24	49	40	160	97
84	242	247	144	203	3	19	24	198	80
67	67	185	98	123	106	168	105	127	153
37	113	214	252	203	80	146	211	7	16
142	241	66	86	214	133	146	253	189	200
67	215	174	111	189	54	144	56	59	163

Challenges in Dealing with Images

Translational Invariance. In the image on the right we have two images of a cat. In one the cat appears on the right hand side of the image and in the other on the left hand side.

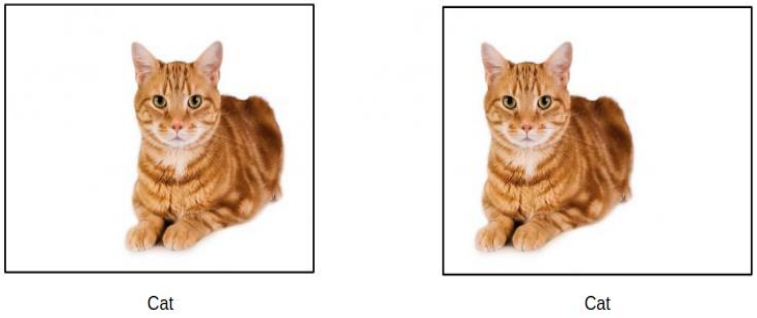


Image orientation :Traditional machine learning algorithms applied to image classification can perform poorly when basic issues such as position and rotation change from image to image.

Viewpoint variation: Where an object can be oriented/rotated in multiple dimensions with respect to how the object is photographed and captured.

Image scale variation. The objective can look quite different when it is photographed up close compared to at a distance.

Image Challenges

- Occlusions, where large parts of an object we want to classify are hidden from view in the image.

- Illumination. Objects can look quite different depending on the lighting present in the image.

- Background Clutter: Some images are very "noisy" and have a lot going of content. We are only interested in one particular object in the image; however, due to all the "noise", it's not easy to pick out

- Intra-class variation. Trying to identify a category of item is difficult where there are many variations within that category.

Image Pixels

- Image pixels in an image are typically represented in two ways:

- 1. Grayscale/single channel
  - In a grayscale image, each pixel is a scalar value between 0 and 255, where **zero corresponds to "black"** and **255 being "white"**. Values between 0 and 255 are varying shades of grey, where values closer to 0 are darker and values closer to 255 are lighter.

- 2. Colour
  - Normally represented in the RGB colour spectrum.
  - **Pixels in the RGB colour are represented by a list of three values**: one value for the Red, Green and Blue component.
  - To define a colour in the RGB model, all we need to do is define the amount of Red, Green, and Blue contained in a single pixel.
  - Each Red, Green, and Blue channel can have values defined in the range [0;255]

Image Coordinate System

The resolution of an image is commonly identified as the width by the height. For example, the image of the panda on the right is **500*375**.

Image processing libraries such as OpenCV represent RGB images as **multidimensional NumPy arrays** with the shape (height, width, depth).

Therefore, if you read in an image using OpenCV it will convert the image to a NumPy array that corresponds to a 3D NumPy array with height*width*depth.

```
import cv2

image = cv2.imread("panda.jpg")
print(image.shape)
print(type(image))
```

```
(375, 500, 3)
<class 'numpy.ndarray'>
```

Characteristics of Convolutional Neural Networks

- Convolution neural network (CNNs) are a powerful type of feed-forward neural network.
- CNNs have had a dramatic impact on the application of ML techniques to image classification.
- We will be look at the component building blocks of convolution neural network but firstly we will describe at the high level some of the core differences between them and traditional neural networks.

Traditional Image Classification and Scale

- Let's assume the image below is represented as an array with 64*32*3
- We can flatten this image into a flat NumPY array of length 6144

Assume this is the first layer of a neural network containing 1000 neurons.

How many learnable weights in the first layer alone (assuming we ignore the biases)?

Traditional Image Classification and Scale

- Let's assume the image below is represented as an array with 64*32*3
- We can flatten this image into a flat NumPY array of length 6144

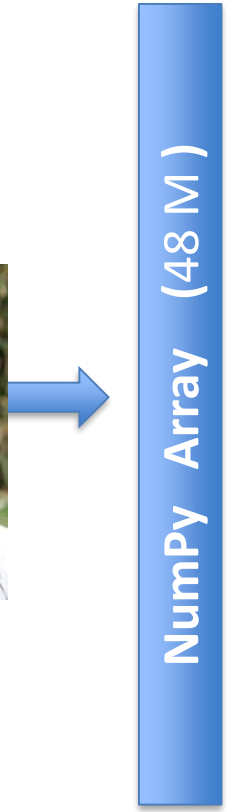
Assume this is the first layer of a neural network containing 1000 neurons.

We can represent the weights for the first layer only as a matrix **1000*6144**.

This means in the first layer alone we will have **6,144,000** (over six million weights in the first layer).

Traditional Image Classification and Scale

- Now let's scale the problem to something a little more realistic. A 4 megapixel image would contain **4000* 4000 * 3** resolution.
- We can flatten this image into a flat NumPY array of length 48 million (**48,000,000**).



Assume this is the first layer of a neural network containing **1000 neurons**.

We can represent the weights for the first layer only as a matrix 1000*48M. The weights we have to learn for the first layer now jumps to **48,000,000,000 (48 billion!!).**

Computational requirements for building a network like this is not feasible.

However, CNNs provide with a way of dealing with high res image effectively

Composition of a Convolutional Neural Network

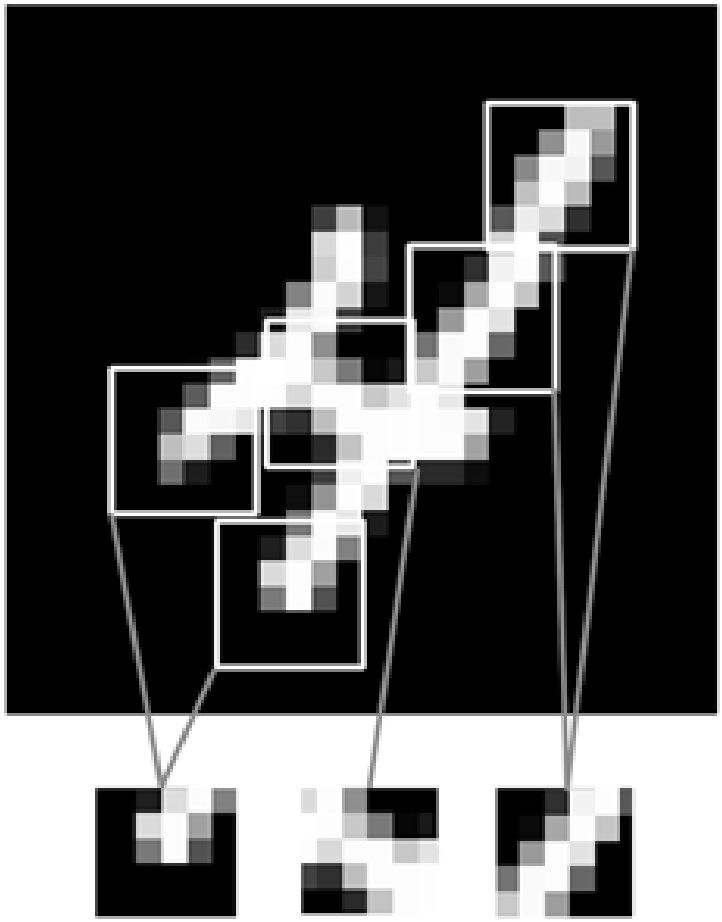
- Dense layers learn **global patterns** in their input feature space

Characteristics of Convolutional Networks

The fundamental difference between a densely connected layer and a convolution layer is that Dense layers **learn global patterns** in their input feature space (for example, for a MNIST digit, patterns involving all pixels),

In contrast convolution layers **learn local patterns** in the case of images, patterns found in small 2D windows of the inputs.

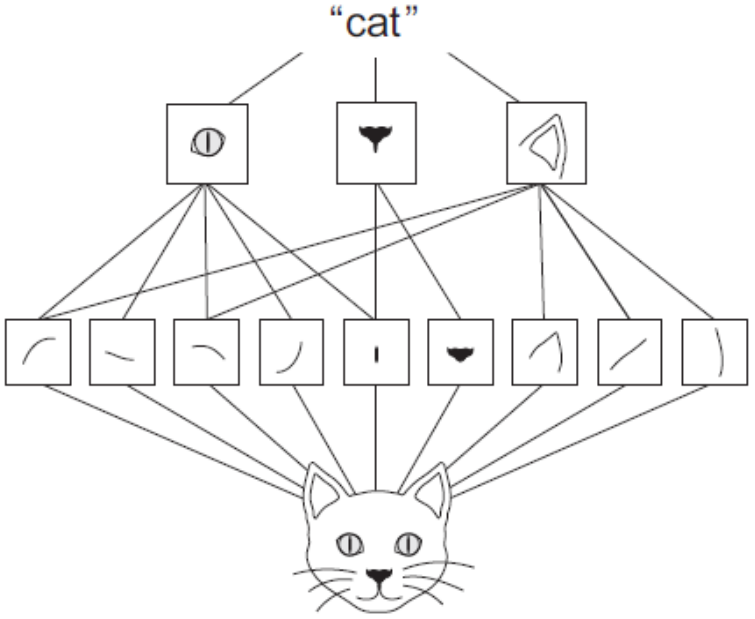
Another advantage of convolution networks is that they conserve spatial structure.



Characteristics of Convolutional Networks

The patterns they learn are **translational invariant**. After learning to identify a certain pattern a CNN can recognize it anywhere within that image.

A first convolution layer will learn small local patterns such as edges, a second convolution layer will learn larger patterns made of the features of the first layers, and so on. This allows convolution nets to efficiently learn increasingly complex and abstract visual concepts.



Characteristics of Convolutional Networks

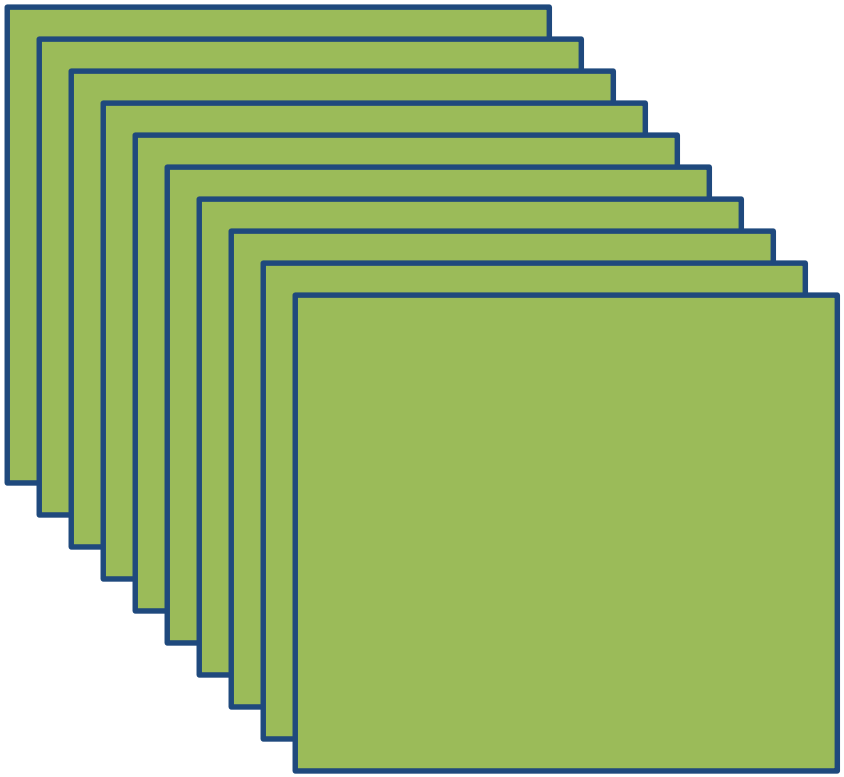
Convolutions operate over 3D tensors, called **feature maps**, with a height and width and depth axis.

The convolution operation analyzes a small portion or **window** of the input feature map and applies a **transformation** (using a **filter**) to this window.

The **window slides** slowly along the original feature map and we continue to apply the filter. This produce an output feature map (more on this later).

The **number of layers** in the outputted feature map correspond to the **number of filters** applied to the original feature map.

Each filter map encodes specific characteristics of the input data.



Components of a CNN

- The understand the architecture of most convolution networks you need to understand the following components.
  - **Convolutional Layer**
  - Padding
  - Stride
  - Pooling Layer
  - 1*1 Convolutional Layer
  - Fully Connected Layers

Components of a CNN – Convolutional Layer

- A convolution involves repeatedly applying a **filter** (kernel) to an original image.

- More technically, in the context of deep learning, an (image) convolution is an **element-wise multiplication of two matrices followed by a summation**.
- One matrix is a (local) portion of the **original image**.
- The other matrix is a **filter** (also often referred to as a kernel).

- For example, a basic convolution might involve applying an **edge detection** filter or a **blurring** filter to an existing image.

Convolution - Applying a Filter to Image

5	1	3	5	7	9
3	4	4	0	7	8
6	1	8	6	3	4
1	3	5	8	4	6
1	7	5	3	1	2
7	2	2	6	4	6

Original Image

-1	0	1
-1	0	1
-1	0	1

Filter (Kernel)

Result of Convolution

5 **-1**	1 **0**	3 **1**	5	7	9
3 **-1**	4 **0**	4 **1**	0	7	8
6 **-1**	1 **0**	8 **1**	6	3	4
1	3	5	8	4	6
1	7	5	3	1	2
7	2	2	6	4	6

Original Image

-1	0	1
-1	0	1
-1	0	1

(5*-1) + (1*0) + (3*1) +
(3*-1) + (4*0) + (4*1) +
(6*-1) + (1*0) + (8*1)

Result of Convolution (often referred to as a feature map)

1			

5	1 **-1**	3 **0**	5 **1**	7	9
3	4 **-1**	4 **0**	0 **1**	7	8
6	1 **-1**	8 **0**	6 **1**	3	4
1	3	5	8	4	6
1	7	5	3	1	2
7	2	2	6	4	6

Original Image

Notice that we move (slide) the filter over one pixel width to the right

Result of Convolution

1	5		

-1	0	1
-1	0	1
-1	0	1

(1*-1) + (3*0) + (5*1) +
(4*-1) + (4*0) + (0*1) +
(1*-1) + (8*0) + (6*1)

5	1	3 **-1**	5 **0**	7 **1**	9
3	4	4 **-1**	0 **0**	7 **1**	8
6	1	8 **-1**	6 **0**	3 **1**	4
1	3	5	8	4	6
1	7	5	3	1	2
7	2	2	6	4	6

Original Image

-1	0	1
-1	0	1
-1	0	1

(3*-1) + (5*0) + (7*1) +
(4*-1) + (0*0) + (7*1) +
(8*-1) + (6*0) + (3*1)

Result of Convolution

1	5	2	

5	1	3	5 **-1**	7 **0**	9 **1**
3	4	4	0 **-1**	7 **0**	8 **1**
6	1	8	6 **-1**	3 **0**	4 **1**
1	3	5	8	4	6
1	7	5	3	1	2
7	2	2	6	4	6

Original Image

-1	0	1
-1	0	1
-1	0	1

(5*-1) + (7*0) + (9*1) +
(0*-1) + (7*0) + (8*1) +
(6*-1) + (3*0) + (4*1)

Result of Convolution

1	5	2	10

5	1	3	5	7	9
3 **-1**	4 **0**	4 **1**	0	7	8
6 **-1**	1 **0**	8 **1**	6	3	4
1 **-1**	3 **0**	5 **1**	8	4	6
1	7	5	3	1	2
7	2	2	6	4	6

Original Image

-1	0	1
-1	0	1
-1	0	1

(3*-1) + (4*0) + (4*1) +
(6*-1) + (1*0) + (8*1) +
(1*-1) + (3*0) + (5*1)

Result of Convolution

1	5	2	10
7			

5	1	3	5	7	9
3	4	4	0	7	8
6	1	8	6	3	4
1	3	5	8 **-1**	4 **0**	6 **1**
1	7	5	3 **-1**	1 **0**	2 **1**
7	2	2	6 **-1**	4 **0**	6 **1**

Original Image

Result of Convolution

1	5	2	10
7	.	.	.
.	.	.	.
.	.	.	.

Notice here we have a **6*6** matrix initially. When we convolve it with a **3*3** filter we get a **4*4** matrix.

We can calculate the row dimension of the resulting matrix as follows:

n – k + 1 where:

- **n** is the number of *rows* in the original matrix
- **k** is the number of rows in the filter.

We can apply the same equation to calculate the number of columns (where n is original number of columns and k is number of filter columns).

- We have a broad range of possible filters that we can apply to an image. How do we determine **which filters we should apply to an image in order to generate some features?**
- One option is that we could **use a range of pre-made standard filters** (based on computer vision research).
- Alternatively we could view the **filter contents as learnable parameters** that we can train our neural network to estimate using back propagation (sound familiar?)
- So the neural network can learn to populate it's filter which can be used to extract relevant features (horizontal, vertical edges etc) from the image.

5	1	3	5	7	9
3	4	4	0	7	8
6	1	8	6	3	4
1	3	5	8	4	6
1	7	5	3	1	2
7	2	2	6	4	6

Original Image

?	?	?
?	?	?
?	?	?

Filter (Kernel)

1	5	2	10
7	.	.	.
.	.	.	.
.	.	.	.

5	1	3	5	7	9
3	4	4	0	7	8
6	1	8	6	3	4
1	3	5	8	4	6
1	7	5	3	1	2
7	2	2	6	4	6

Original Image

?	?	?
?	?	?
?	?	?

Filter (Kernel)

1	5	2	10
7	.	.	.
.	.	.	.
.	.	.	.

5	1	3	5	7	9
3	4	4	0	7	8
6	1	8	6	3	4
1	3	5	8	4	6
1	7	5	3	1	2
7	2	2	6	4	6

Original Image

?	?	?
?	?	?
?	?	?

Filter (Kernel)

1	5	2	10
7	.	.	.
.	.	.	.
.	.	.	.

+ bias

5	1	3	5	7	9
3	4	4	0	7	8
6	1	8	6	3	4
1	3	5	8	4	6
1	7	5	3	1	2
7	2	2	6	4	6

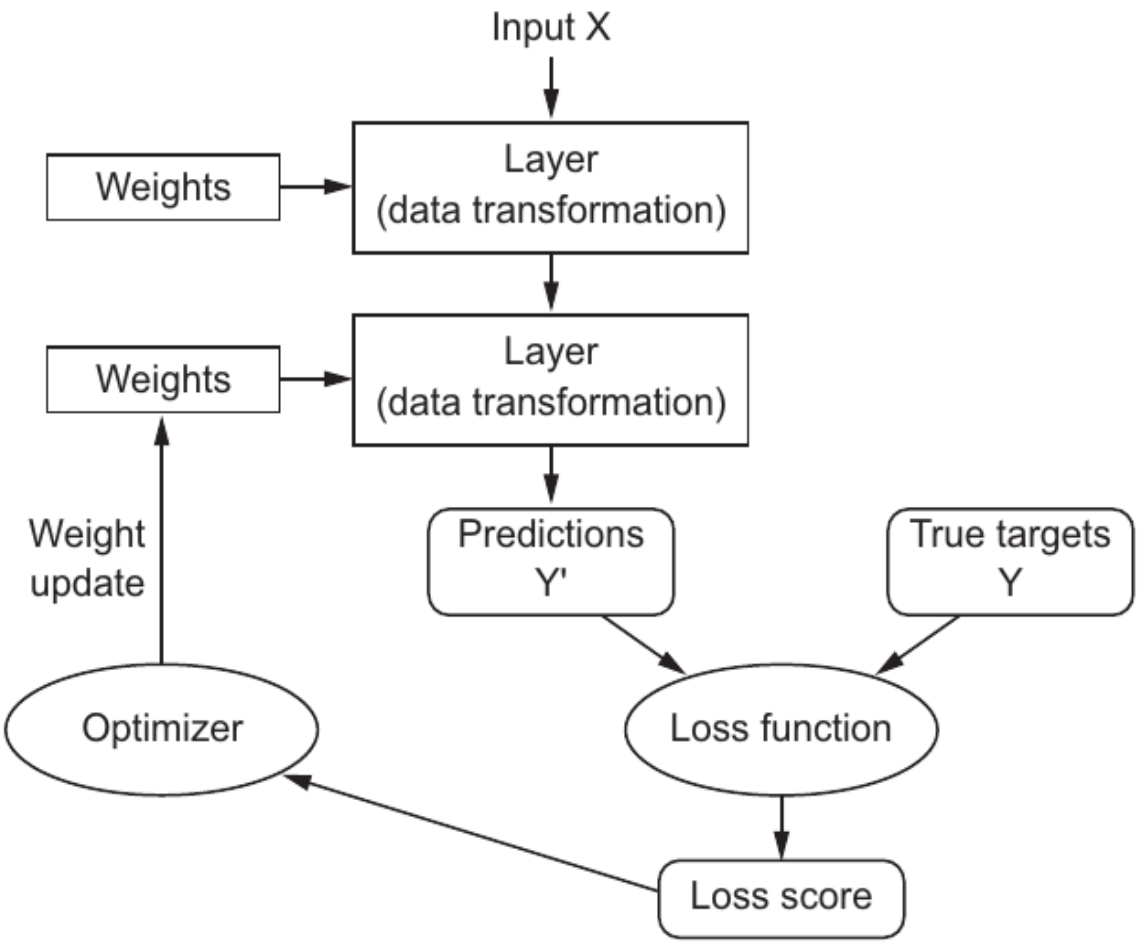
Original Image

?	?	?
?	?	?
?	?	?

Filter (Kernel)

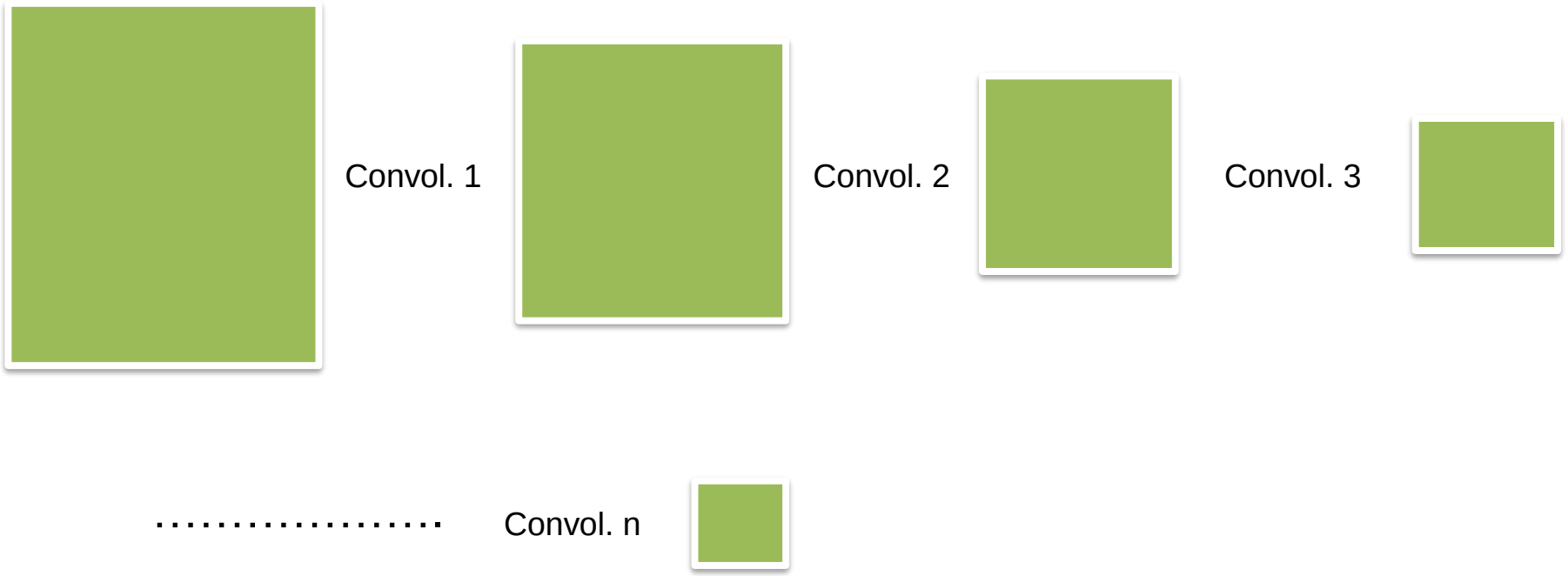
RELU(| 1 | 5 | 2 | 10 |
|---|---|---|---|
| 7 | . | . | . |
| . | . | . | . |
| . | . | . | . |

+ bias)


Input X
Weights
Layer
(data transformation)
Weights
Layer
(data transformation)
Weight
update
Predictions
Y'
True targets
Y
Optimizer
Loss function
Loss score

In a convolution network we can have many convolutional layers.

At lower levels of a convolutional neural network the features tend to low level features. However, at deeper layers the feature tend to be higher level and more specific to the particular problem domain.



Components of a CNN

- A convolution neural network typically consists of the following components
  - Convolutional Layer
  - **Padding**
  - Stride
  - Pooling Layer
  - 1*1 Convolutional Layer
  - Fully Connected Layers

Shrinkage of Feature Map

- When we applied our **3*3 filter** to the original image we went from a matrix with **6*6 to 4*4 (n – k + 1)**.

- If we were to apply another convolution to the resulting matrix (with the same filter size) what would be the size of the resulting feature map?

5	1	3	5	7	9
3	4	4	0	7	8
6	1	8	6	3	4
1	3	5	8	4	6
1	7	5	3	1	2
7	2	2	6	4	6

Original Image

?	?	?
?	?	?
?	?	?

Filter (Kernel)

1	5	2	10
7			

Result of Convolution

Shrinkage of Feature Map

- When we applied our **3*3 filter** to the original image we went from a matrix with **6*6 to 4*4 (n – k + 1)**.
- If we were to apply another convolution to the resulting matrix (with the same filter size) what would happen?
- The resulting matrix would be a 2*2 matrix.
- The problem we encounter is that the resulting matrices rapidly become smaller and smaller.

5	1	3	5	7	9
3	4	4	0	7	8
6	1	8	6	3	4
1	3	5	8	4	6
1	7	5	3	1	2
7	2	2	6	4	6

Original Image

1	5	2	10
7			

Convolution 1

Convolution 2

Exposure of Pixels to Convolutional Operations

- Another issues is that some pixels in our image are involved **more regularly** in the convolutions than others.

5	1	3	5	7	9
3	4	4	0	7	8
6	1	8	6	3	4
1	3	5	8	4	6
1	7	5	3	1	2
7	2	2	6	4	6

Original Image

?	?	?
?	?	?
?	?	?

Filter (Kernel)

1	5	2	10
7			

Result of Convolution

Exposure of Pixels to Convolutional Operations

- Another issues is that some pixels in our image are involved **more regularly** in the convolutions than others.
- For example, the pixel on the top right or top left (highlighted in blue) are only ever involved in a single convolution. Other pixels such as the one highlighted in green could be involved many times (9 times in this example).
- The pixels are the corners and edges of the images are used much less frequently as such there is a **risk of losing potentially useful features** from this portion of the image.

5	1	3	5	7	9
3	4	4	0	7	8
6	1	8	6	3	4
1	3	5	8	4	6
1	7	5	3	1	2
7	2	2	6	4	6

Original Image

?	?	?
?	?	?
?	?	?

Filter (Kernel)

1	5	2	10
7			

Result of Convolution

Components of CNNs - Padding

- A simple way to address the two problems outlined in the previous slide is to pad the image with zero value pixels around the edge before performing a convolution. In the example below we have a padding of 1.
- This means the actual **image pixels will get more exposure to the convolutions**. Also we can mitigate against the impact of convolutions making our feature space smaller and smaller.

0	0	0	0	0	0	0	0
0	5	1	3	5	7	9	0
0	3	4	4	0	7	8	0
0	6	1	8	6	3	4	0
0	1	3	5	8	4	6	0
0	1	7	5	3	1	2	0
0	7	2	2	6	4	6	0
0	0	0	0	0	0	0	0

Original Image

Components of CNNs - Padding

- **n** = number of rows in the image matrix
- **k** = number of rows in the filter.
- **P** = layers of padding

Resulting matix will have size:

(6+2)-3+1 (**n+2p-k+1**) = 6.

0	0	0	0	0	0	0	0
0	5	1	3	5	7	9	0
0	3	4	4	0	7	8	0
0	6	1	8	6	3	4	0
0	1	3	5	8	4	6	0
0	1	7	5	3	1	2	0
0	7	2	2	6	4	6	0
0	0	0	0	0	0	0	0

Original Image

?	?	?
?	?	?
?	?	?

Filter (Kernel)

Result of Convolution

Components of CNNs - Padding

- Typically if padding is used in a CNN it is done in such a way to **conserve the original height and width** of the incoming matrix.

- There are two types of padding that is used, which are referred to as either “valid convolutions” or “same convolutions”.

- **Valid convolutions** simply means that no padding is applied.
- Therefore, if you have a n*n image and a filter of size k*k then your resulting image would be n-k+1

- **Same convolutions** means sufficient padding is applied to ensure that the height and width of the original matrix is unaltered in the resulting matrix.
- So in order to perform a same convolution we set p = (k-1)/2 .

Components of a CNN

- A convolution neural network typically consists of the following components
  - Convolutional Layer
  - Padding
  - **Stride**
  - Pooling Layer
  - 1*1 Convolutional Layer
  - Fully Connected Layers

Stride

- By default when performing our convolution we slide the filter in steps of **one pixel at a time**.
- The stride is a integer parameter that is used to specify the **step size in pixels** when performing convolutions (in the previous example our stride s = 1).
- Notice below we are moving the filter in two pixels steps (s = 2)

0 ?	0 ?	0 ?	0	0	0	0
0 ?	5 ?	1 ?	3	5	7	0
0 ?	3 ?	4 ?	4	0	7	0
0	6	1	8	6	3	0
0	1	3	5	8	4	0
0	1	7	5	3	1	0
0	0	0	0	0	0	0

?	?	?
?	?	?
?	?	?

Stride

- By default when performing our convolution we slide the filter in steps of one pixel at a time.
- The stride is a integer parameter that is used to specify the **step size in pixels** when performing convolutions (in the previous example our stride s = 1).
- Notice below we are moving the filter in two pixels steps (s = 2)

0	0	0 ?	0 ?	0 ?	0	0
0	5	1 ?	3 ?	5 ?	7	0
0	3	4 ?	4 ?	0 ?	7	0
0	6	1	8	6	3	0
0	1	3	5	8	4	0
0	1	7	5	3	1	0
0	0	0	0	0	0	0

Stride

- By default when performing our convolution we slide the filter in steps of one pixel at a time.
- The stride is a integer parameter that is used to specify the **step size in pixels** when performing convolutions (in the previous example our stride s = 1).
- Notice below we are moving the filter in two pixels steps (s = 2)

0	0	0	0	0 ?	0 ?	0 ?
0	5	1	3	5 ?	7 ?	0 ?
0	3	4	4	0 ?	7 ?	0 ?
0	6	1	8	6	3	0
0	1	3	5	8	4	0
0	1	7	5	3	1	0
0	0	0	0	0	0	0

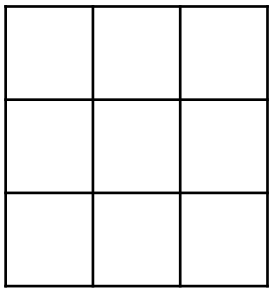
Stride

- By default when performing our convolution we slide the filter in steps of one pixel at a time.
- The stride is a integer parameter that is used to specify the **step size in pixels** when performing convolutions (in the previous example our stride s = 1).
- Notice below we are moving the filter in two pixels steps (s = 2)

0	0	0	0	0	0	0
0	5	1	3	5	7	0
0 ?	3 ?	4 ?	4	0	7	0
0 ?	6 ?	1 ?	8	6	3	0
0 ?	1 ?	3 ?	5	8	4	0
0	1	7	5	3	1	0
0	0	0	0	0	0	0

- With a stride size of 2 and a filter size of 3 the matrix below will produce a matrix of dimension 3*3.
- Notice that if we tried to set our stride to 3, we will encounter issues with spacing. The filter will not fit into the available image evenly.
- The formula for calculating output size m of a matrix is $$m = \frac{n + 2p - k}{s} + 1$$
  - **n** is the rows or cols dimension of original matrix
  - **p** is amount of padding
  - **k** is filter size
  - **s** is stride size

0	0	0	0	0	0	0
0	5	1	3	5	7	0
0 ?	3 ?	4 ?	4	0	7	0
0 ?	6 ?	1 ?	8	6	3	0
0 ?	1 ?	3 ?	5	8	4	0
0	1	7	5	3	1	0
0	0	0	0	0	0	0



Result of Convolution

- With a stride size of 2 and a filter size of 3 the matrix below will produce a matrix of dimension 3*3.
- Notice that if we tried to set our stride to 3, we will encounter issues with spacing. The filter will not fit into the available image evenly.
- The formula for calculating output size m of a matrix is $$m = \frac{n + 2p - k}{s} + 1$$
  - **n** is the rows or cols dimension of original matrix
  - **p** is amount of padding
  - **k** is filter size
  - **s** is stride size

0	0	0	0	0	0	0
0	5	1	3	5	7	0
0 ?	3 ?	4 ?	4	0	7	0
0 ?	6 ?	1 ?	8	6	3	0
0 ?	1 ?	3 ?	5	8	4	0
0	1	7	5	3	1	0
0	0	0	0	0	0	0

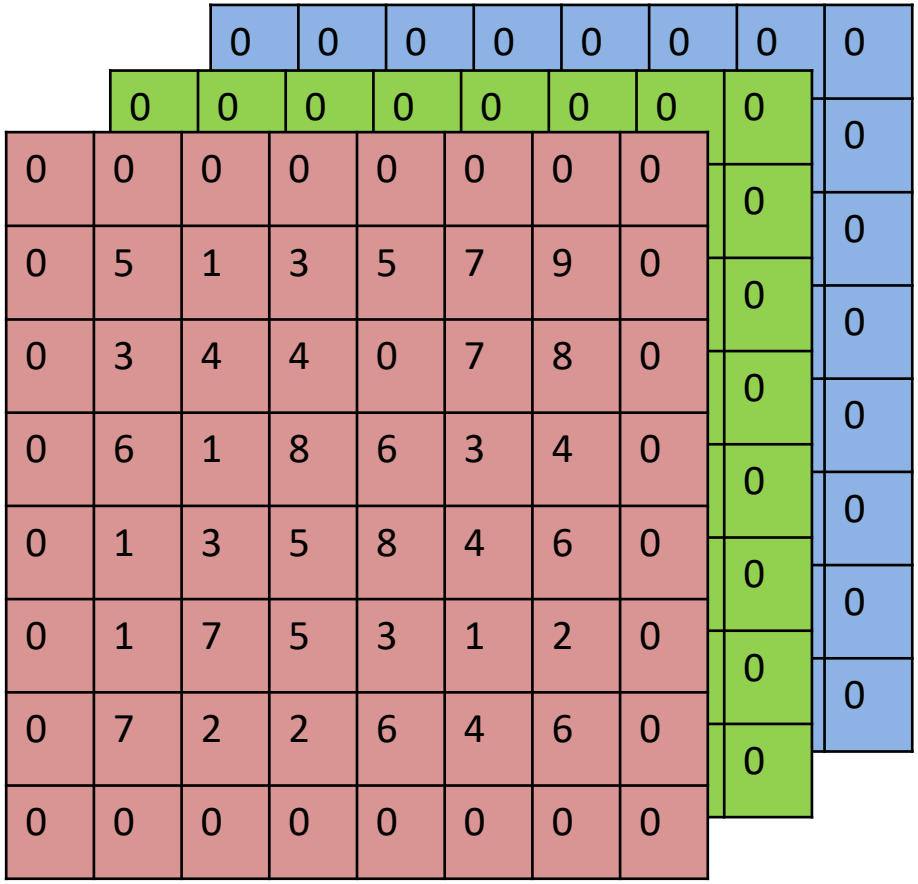
A smaller stride size will lead to **larger overlapping** and **larger output size**.

Conversely, a **larger stride value** will result in **less overlapping** and **smaller output size**.

Thus, we can see how the stride value in a convolution layer can be used to **reduce the spatial dimensions** of the input simply by changing the stride of the kernel.

Dealing with 3D Feature Maps

- In the previous examples we only dealt with images represented as 2D arrays. However, as we have seen with colour images there are three colour channels.
- For example, the image below can be represented as a 8*8*3 3D matrix.
- So the question is **how do we perform a convolution for a 3D image?**



Original Image (8 (height) * 8 (width)* 3 (channels))

Dealing with 3D Images

- So the question is now do we perform a convolution for a 3D image?

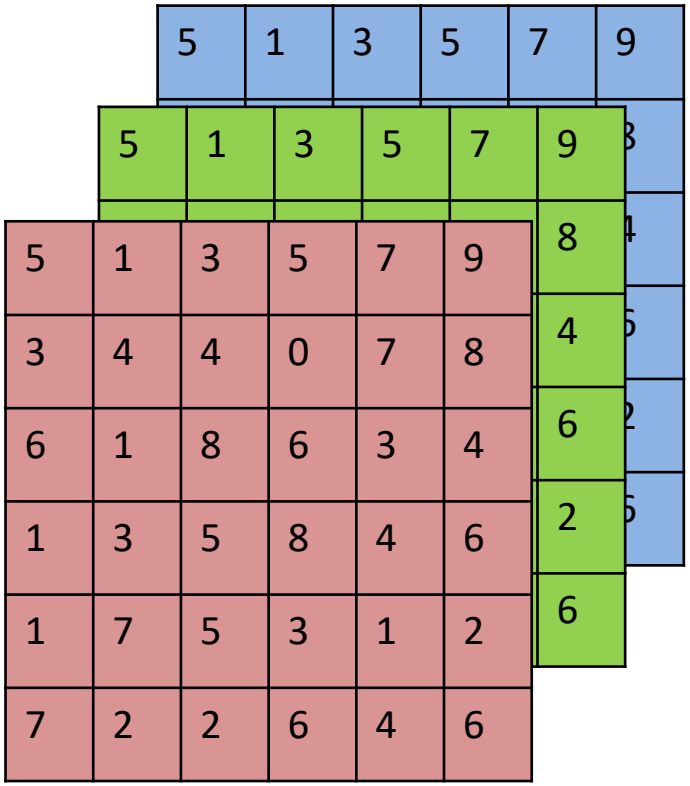
- **We use a filter with the same number of channels (the same depth)**.

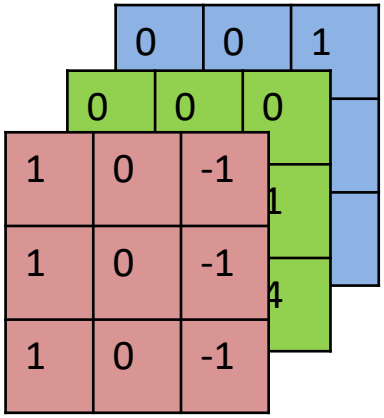
- In other words we must have a **3D filter**. The first channel in the filter will be applied to the first channel in the image. The second channel in the filter will be applied to the second channel in the image and so on.
  - Notice this means we can have a filter that might detect edges on the red channel only or on the blue channel only or across all channels.

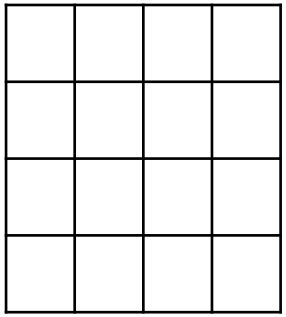
- The **convolution process** is the similar to that previously described.
- We convolve the **red filter channel** with the **red image channel** and calculate the result.
- We convolve the **green filter channel** with the **green image channel** and calculate the result.
- Finally we convolve the **blue filter channel** with the **blue image channel** and calculate the result.
- We then **add the result** of all three convolutions and this value is placed in the result matrix.

Dealing with a 3D Input Matrix

- We have a filter with the same number of channels. In other words we must have a 3D filter. The first channel in the filter will be applied to the first channel in the image. The second channel in the filter will be applied to the second channel in the image and so on.

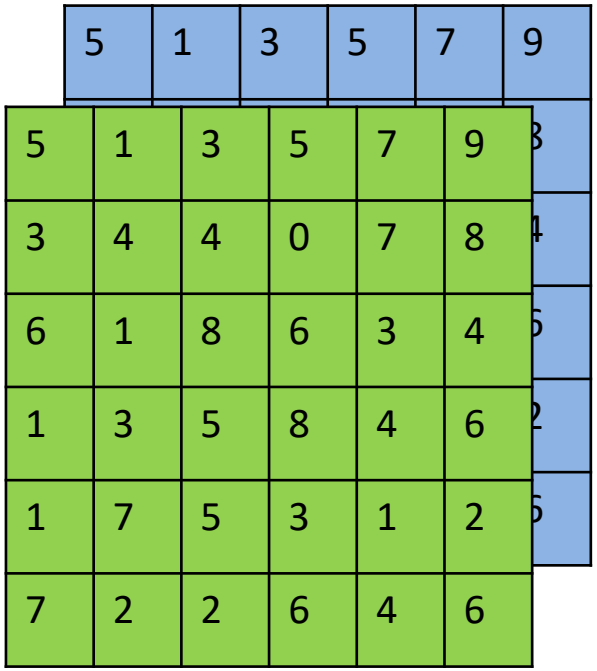


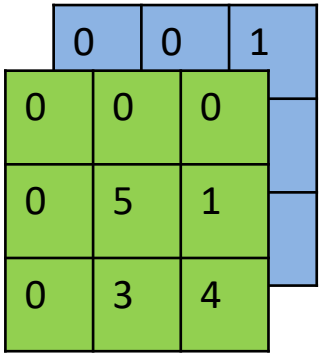


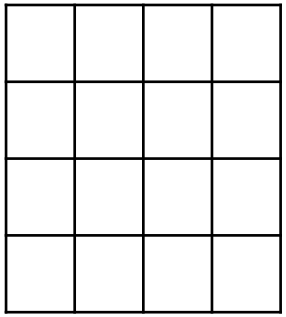


Dealing with a 3D Input Matrix

- We have a filter with the same number of channels. In other words we must have a 3D filter. The first channel in the filter will be applied to the first channel in the image. The second channel in the filter will be applied to the second channel in the image and so on.







Dealing with a 3D Input Matrix

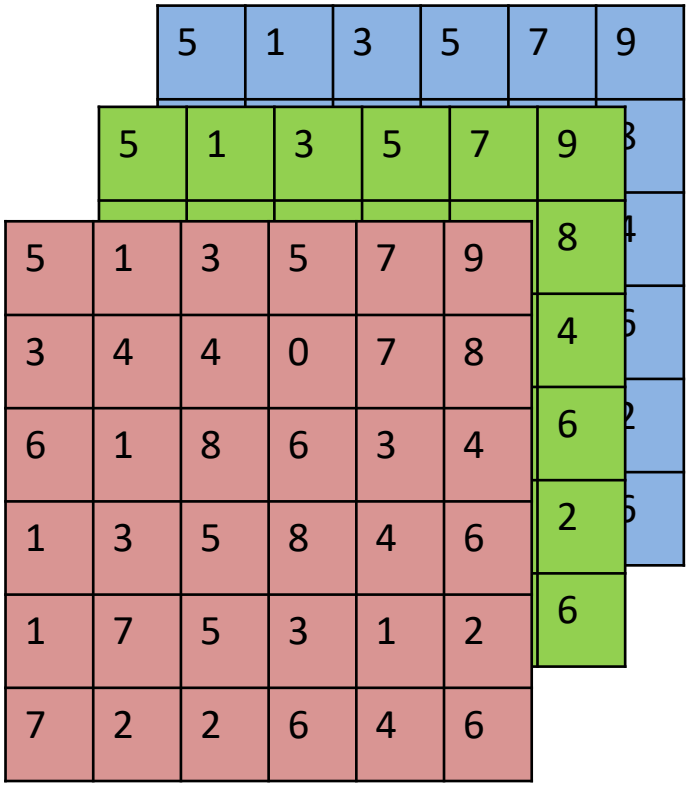
- We have a filter with the same number of channels. In other words we must have a 3D filter. The first channel in the filter will be applied to the first channel in the image. The second channel in the filter will be applied to the second channel in the image and so on.

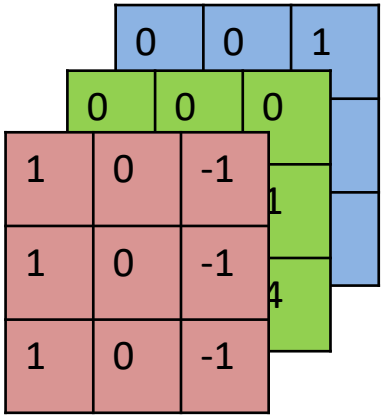
5	1	3	5	7	9
3	4	4	0	7	8
6	1	8	6	3	4
1	3	5	8	4	6
1	7	5	3	1	2
7	2	2	6	4	6

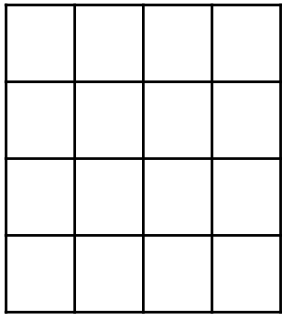
0	0	1
5	5	1
1	3	4

Dealing with a 3D Input Matrix

- We have a filter with the same number of channels. In other words we must have a 3D filter. The first channel in the filter will be applied to the first channel in the image. The second channel in the filter will be applied to the second channel in the image and so on.







Notice that despite the fact that the original feature map/image had a depth of 3 and we convolved it with a filter of depth 3, this produced a resulting feature map that was just **2D (no depth).**

Dealing with Volume – Multiple Filters

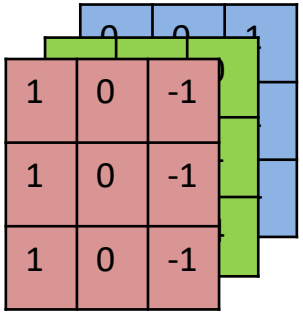
- At this stage we now understand how a **three channel filter** can be applied to a three channel image (note it is important that the number of layers in your filter match the number of layers in the original matrix).

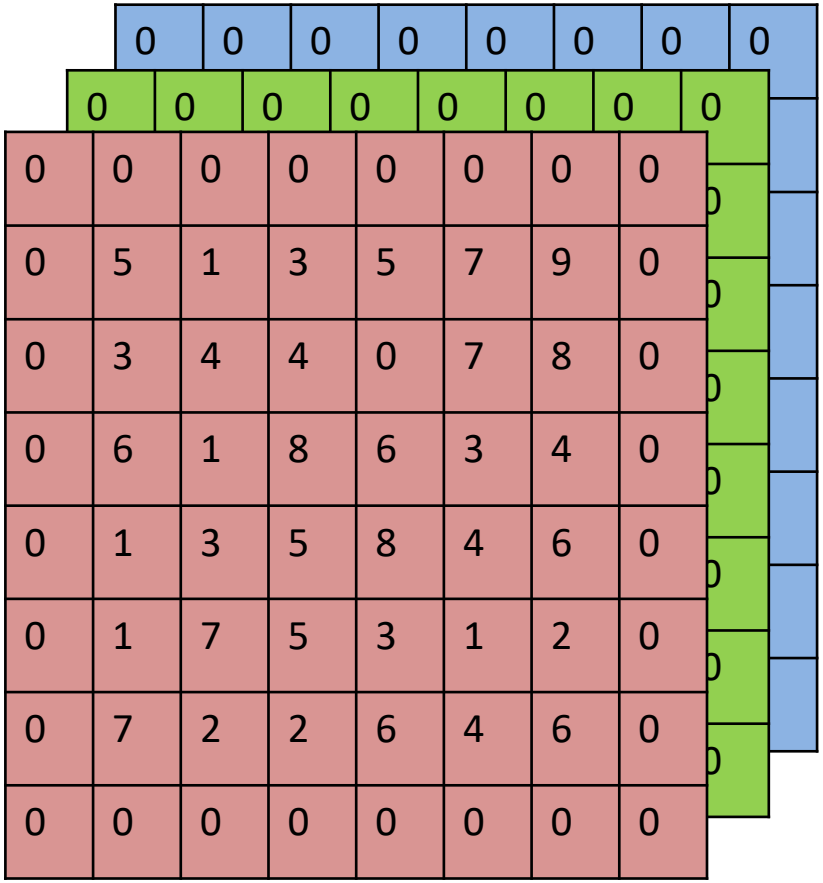
- Our objective is that our network will learn the values for each layer of the filter. Let's assume it manages to **detect edges at a 45 degree angle**.

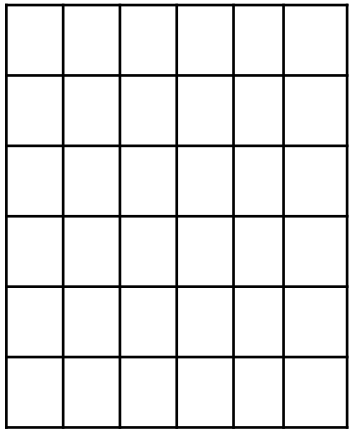
- This **just a single low level feature** we are generating from the data and it is highly likely that this filter by itself will not be sufficient.

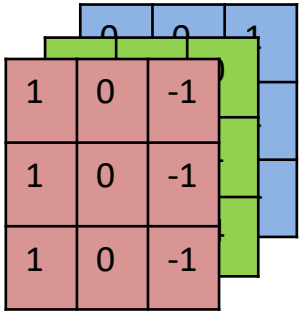
- We may also want filters to **detect horizontal edge**, **vertical edges** etc.

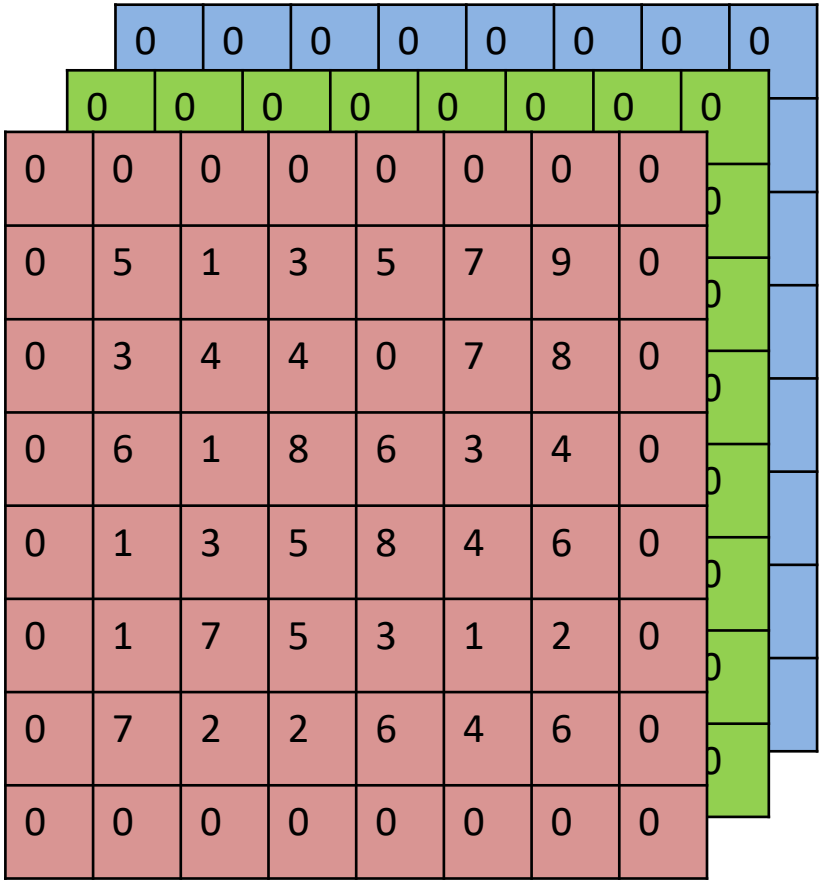
- Therefore, CNNs attempt to learn many filters (as opposed to just a single 3 channel filter)

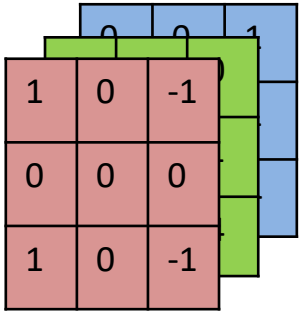

1 0 -1
1 0 -1
1 0 -1

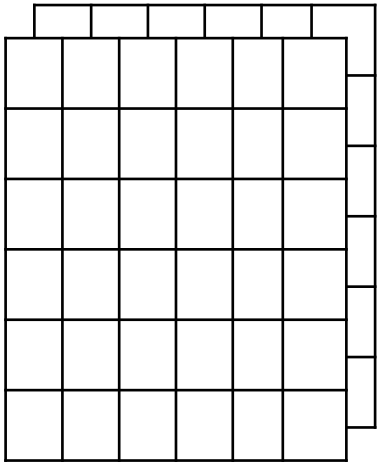

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 5 1 3 5 7 9 0
0 3 4 4 0 7 8 0
0 6 1 8 6 3 4 0
0 1 3 5 8 4 6 0
0 1 7 5 3 1 2 0
0 7 2 2 6 4 6 0
0 0 0 0 0 0 0 0

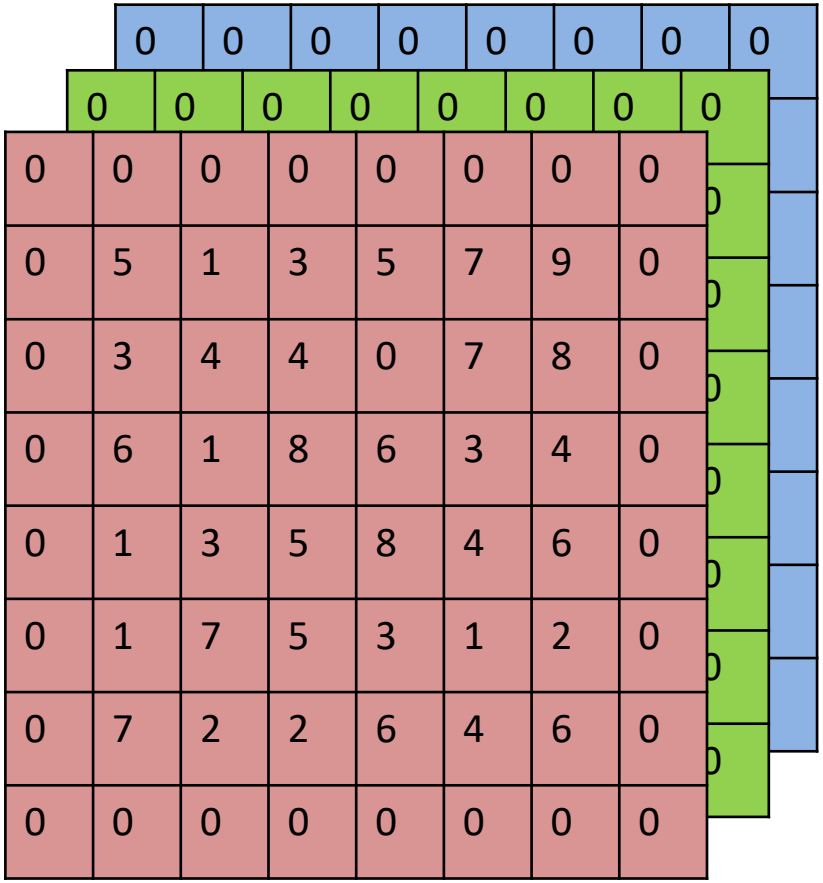


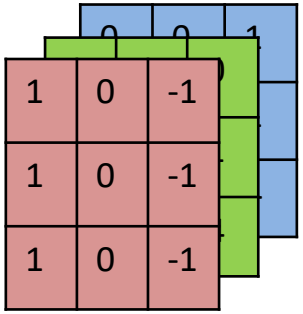

1 0 -1
1 0 -1
1 0 -1

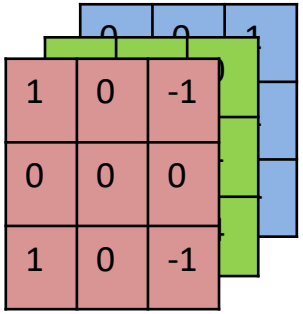

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 5 1 3 5 7 9 0
0 3 4 4 0 7 8 0
0 6 1 8 6 3 4 0
0 1 3 5 8 4 6 0
0 1 7 5 3 1 2 0
0 7 2 2 6 4 6 0
0 0 0 0 0 0 0 0

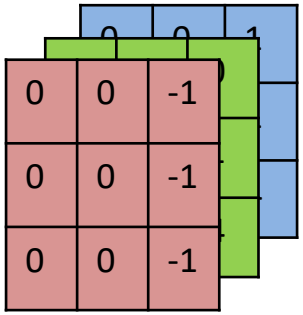

1 0 -1
0 0 0
1 0 -1

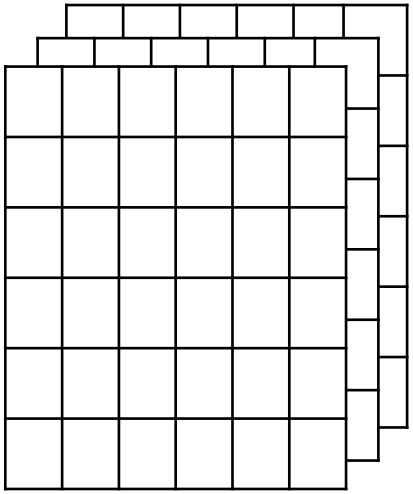


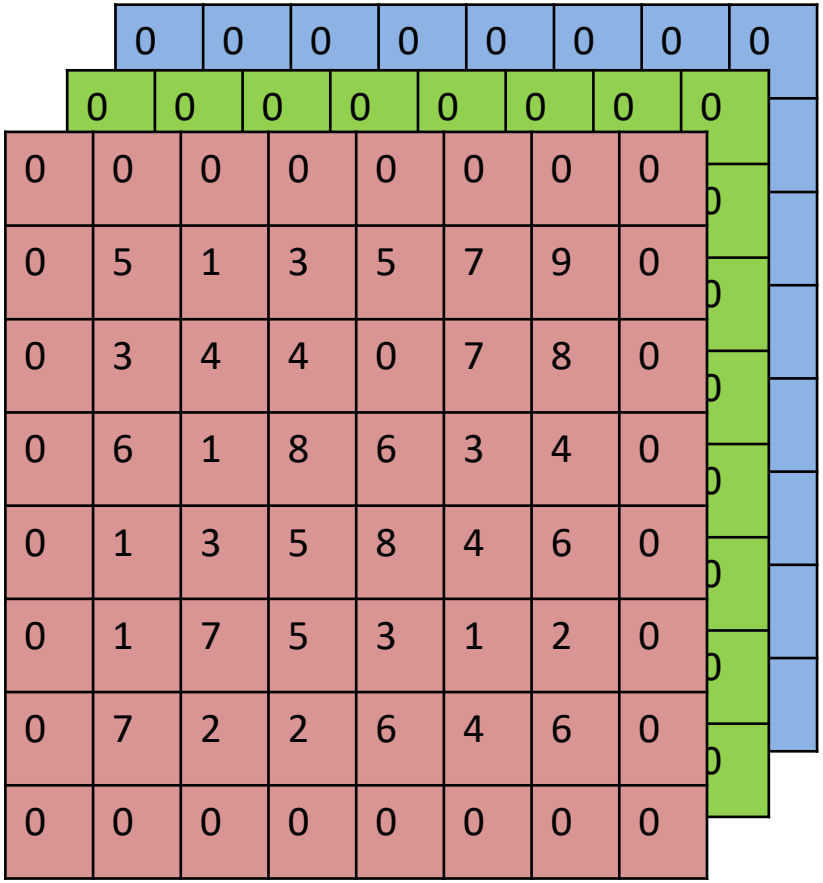

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 5 1 3 5 7 9 0
0 3 4 4 0 7 8 0
0 6 1 8 6 3 4 0
0 1 3 5 8 4 6 0
0 1 7 5 3 1 2 0
0 7 2 2 6 4 6 0
0 0 0 0 0 0 0 0

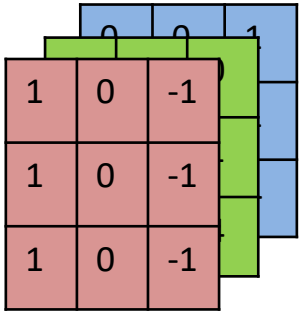

1 0 -1
1 0 -1
1 0 -1

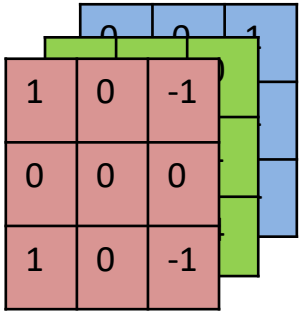

1 0 -1
0 0 0
1 0 -1

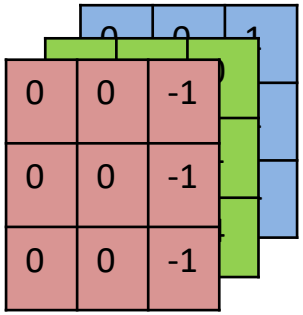

0 0 -1
0 0 -1
0 0 -1

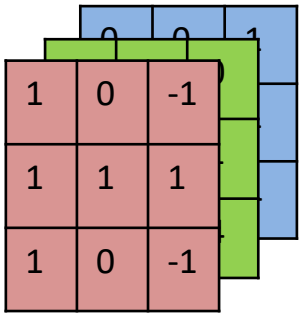


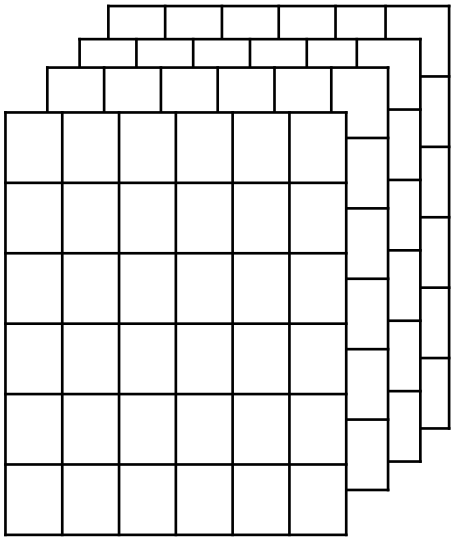

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 5 1 3 5 7 9 0
0 3 4 4 0 7 8 0
0 6 1 8 6 3 4 0
0 1 3 5 8 4 6 0
0 1 7 5 3 1 2 0
0 7 2 2 6 4 6 0
0 0 0 0 0 0 0 0


1 0 -1
1 0 -1
1 0 -1


1 0 -1
0 0 0
1 0 -1


0 0 -1
0 0 -1
0 0 -1


1 0 -1
1 1 1
1 0 -1



Feature Maps – Dealing with Volume

- As we see in the previous slide the convolution operation extracts **localized regions from its input** and applies the same transformation to all of these regions, producing an output which is called a **feature map**.

- This output feature map is a 3D tensor: it has a width, height and depth. The **depth** or number of channels correspond to the **number of filters** we applied.

- **The layers of the filter map represent presence of specific features in the input data**.