acm.ai

Convolutional Neural Networks

Advanced Track Workshop #3

Anonymous feedback: tinyurl.com/w21advtrack3

GitHub: github.com/uclaacmai/advanced-track-winter21

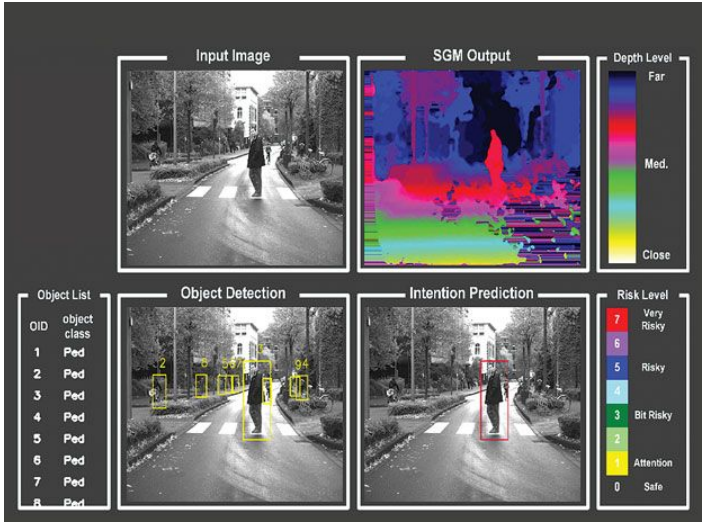
Attendance Code: skeleton

Today's Contents

- Applications of CNNs
- Review of Neural Networks and implementation in TensorFlow and PyTorch
- Theory and Intuition for CNNs
  - Brief review
  - Convolutional layer
  - Other layers

1. Applications of CNNs

Conv Nets Applications



- Used in self driving cars to make predictions on the optimal course of action by companies developing them.

Conv Nets Applications

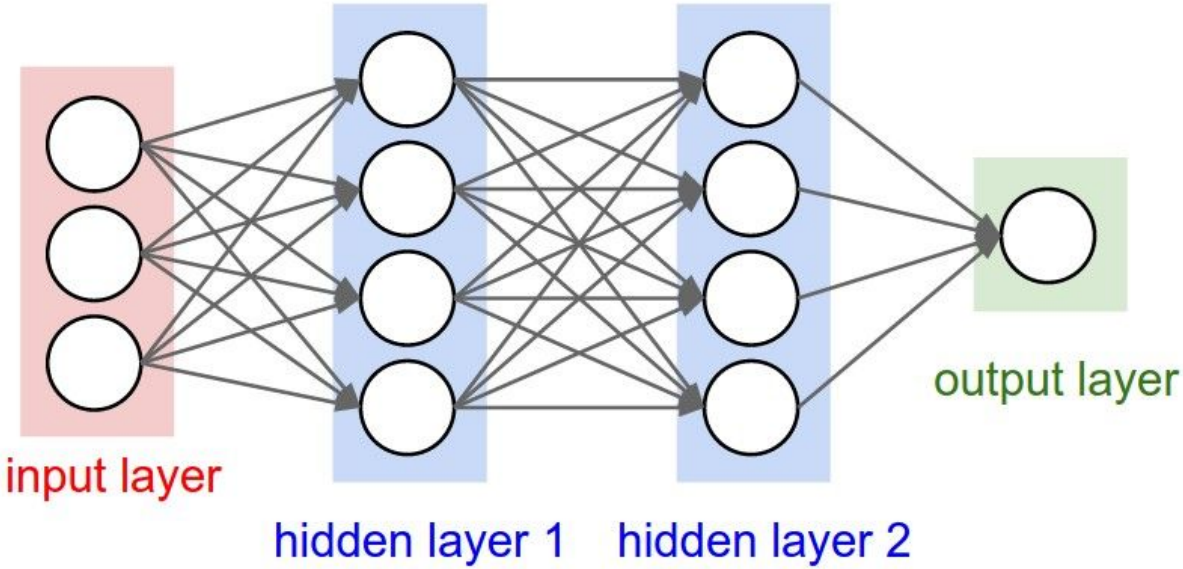
- Facebook uses Conv Nets multiple times to be able to accurately tag people in photos by picking out individual faces and classifying them.

2. Recap of Neural Networks

Recap on Neural Networks

- Let's think about working with a single image first, without worrying about batch sizes.
- Fully-connected (Dense) networks work primarily with "vectors" of input values (think 1D array containing pixel values), and "matrices" of weights (each row or column represents the weights for one neuron).
- The output of any layer is a 1D vector which is then passed on to the next layer.

Recap on Regular NNs



acm.ai

Code Snippet (Tensorflow 2.0 & Keras)

```
import tensorflow as tf

# implementation of a feed-forward/dense neural network with 2 hidden layers
mnist_classifier = tf.keras.models.Sequential()
# first hidden layer
mnist_classifier.add(tf.keras.layers.Dense(units=16, input_shape=(784,), activation='relu'))
# second hidden layer
mnist_classifier.add(tf.keras.layers.Dense(units=12, activation='relu'))
# output layer
mnist_classifier.add(tf.keras.layers.Dense(units=10, activation='softmax'))
# compile the model
mnist_classifier.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# train the model
mnist_classifier.fit(X_train, Y_train, batch_size=32, epoch=100)
# make a prediction
Y_pred = mnist_classifier.predict(X_test)
```

acm.ai

Code Snippet (Pytorch)

```
class model(nn.Module):          # defines the entire structure of the network
    def __init__(self, in_features):
        super().__init__()

        self.fc1 = nn.Linear(in_features = in_features, out_features = 11)   # first layer
        self.fc2 = nn.Linear(in_features = 11, out_features = 10)            # softmax layer
        self.relu = nn.ReLU()
        self.soft1 = nn.Softmax(dim = 1)

    def forward(self, x):       # define the forward propagation
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.soft1(x)

        return x

my_model = model(NUM_FEATURES)                                  # initialize model
opt = torch.optim.Adam(model.parameters(), lr =0.0001)          # define your optimizer
loss_fn = nn.MSELoss(reduction = 'mean')                        # define your loss function
```

= train_step(X_train, y_train)

acm.ai

Code Snippet (Pytorch) Continued

```
def make_train_step(model, opt, loss_fn):       # define one training step
    def train_step(x, y):
        model.train()                           # put model in train mode
        y_pred = model(x)                       # get your current predictions
        loss = loss_fn(y, y_pred)               # calculate loss using those predictions
        loss.backward()                         # calculate gradient of that loss
        opt.step()                              # take a optimizer step (like one step of GD)
        opt.zero_grad()                         # zero out the gradients before the next step
        return loss.item()
    return train_step


train_step = make_train_step(model, opt, loss_fn)

for i in range(5000):
    train_loss = train_step(X_train, y_train)
```

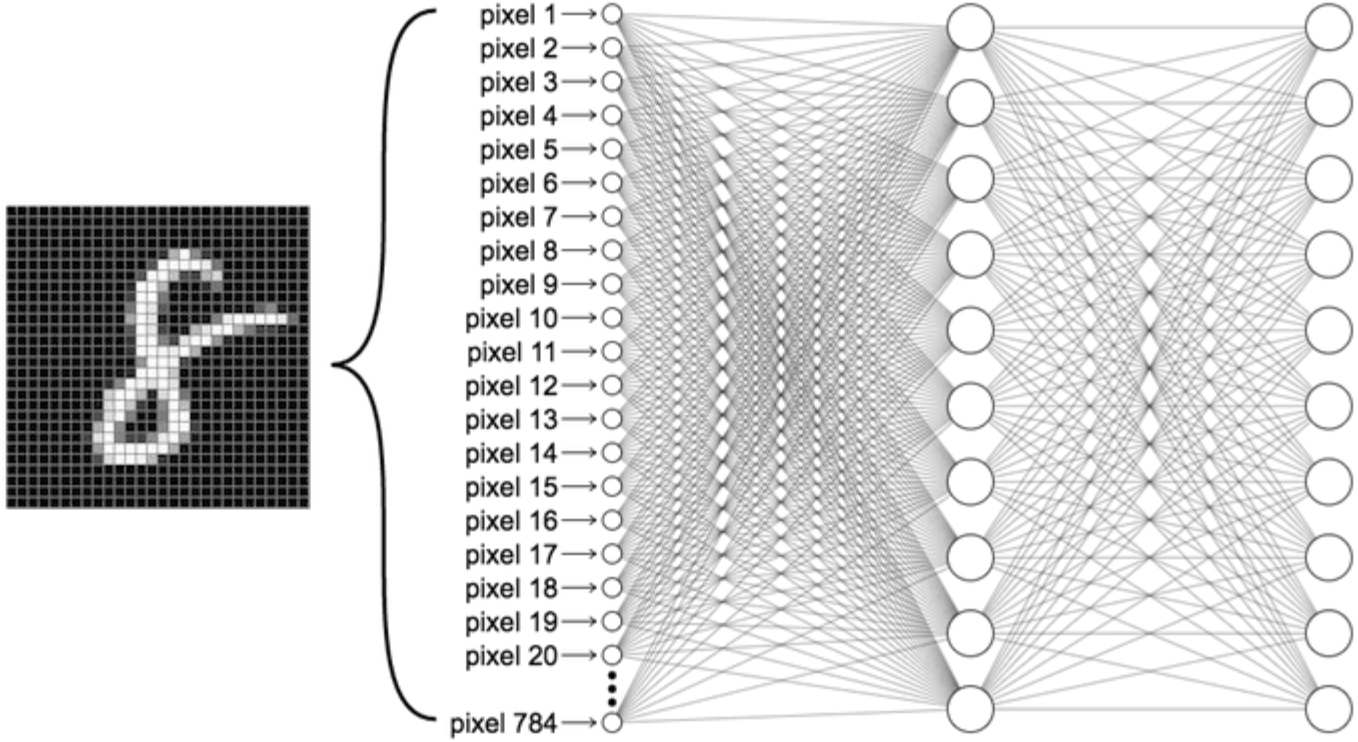
Mini-Batch Gradient Descent Review

```
def MBGD(ffnn, epochs, training_data, batch_size, lr):

    for i in range(epochs):
        shuffle(training_data)
        batches = [training_data[j:j+batch_size] for j in range(0, len(training_data), batch_size)]

        for batch in batches:
            update(batch, lr, ffnn)
```

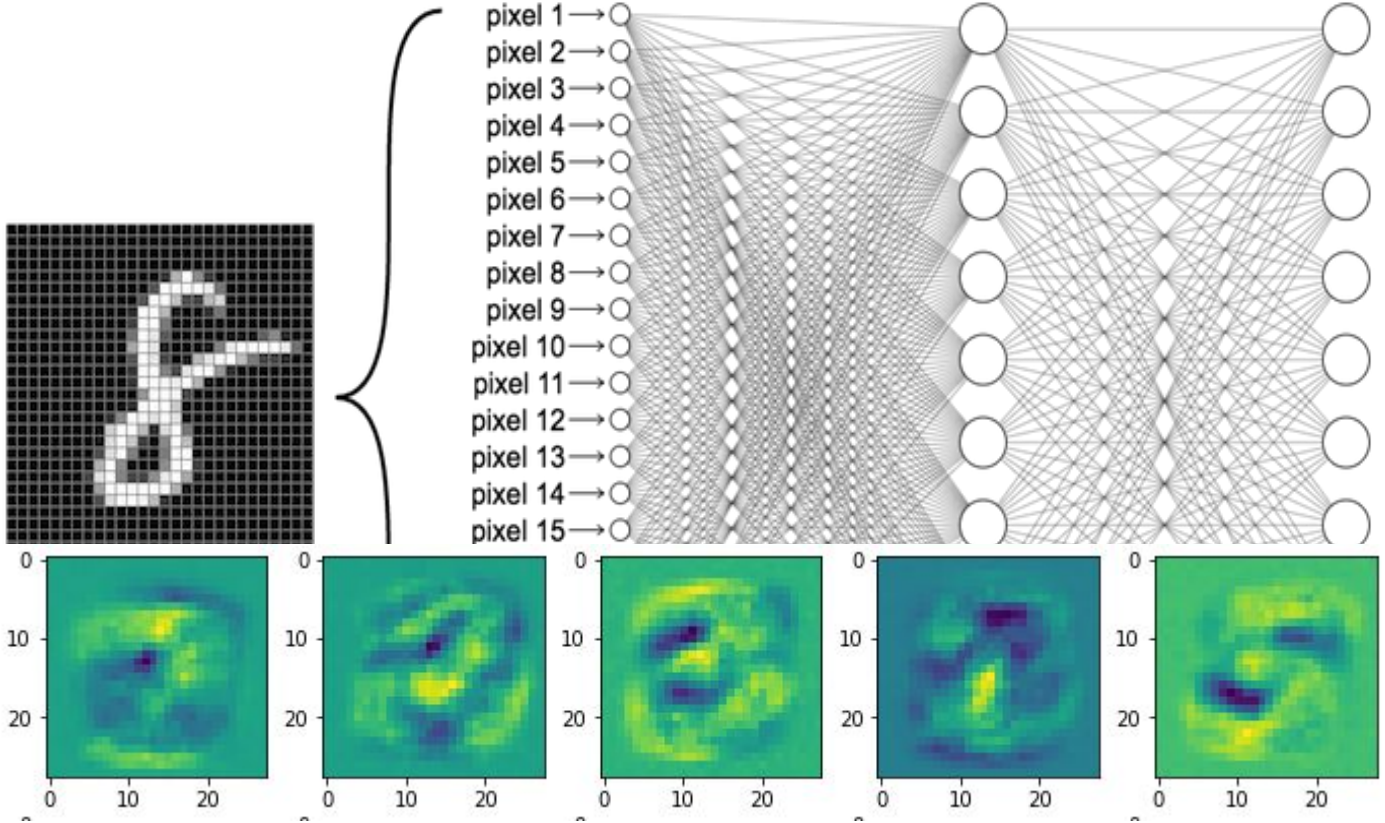
3. CNNs: Motivation

acm.ai

Our Neural Network from Last Week



Visualization of the 1st layer



4. CNNs: Context

What is a CNN?

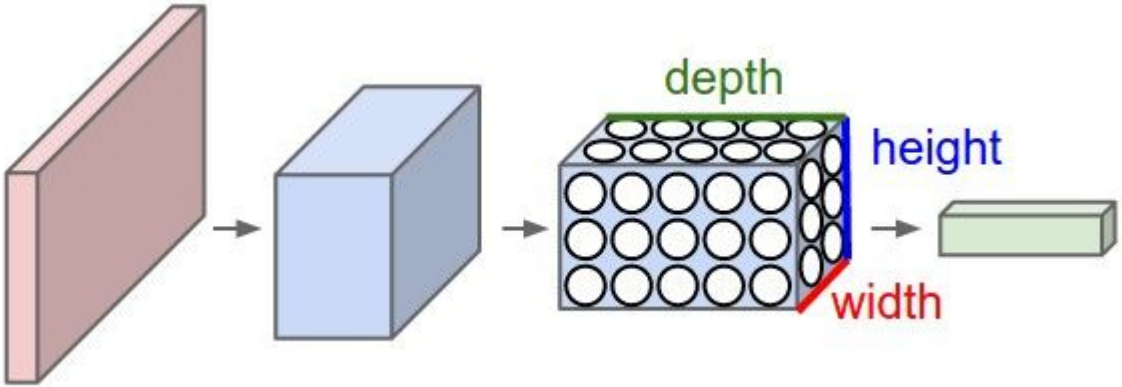
- A type of **neural network** used for computer vision
- The computer performs classification by looking for low level features, and using **convolutional layers** to build up to more complex, higher level features

Structure of CNNs: Data

1. CNNs work with ***volumes*** or **tensors**
    a. vectors are 1D, matrices are 2D, tensors are 3+D
2. An image has a width and height, but also a 3rd dimension - the **RGB** values for each pixel
3. The input is therefore 3D “matrix” of dimensions
    a. Height x Width x 3 (usually)
4. Here, the depth is 3 because each pixel has 3 “channels”, one for each of Red, Green and Blue.
5. **monochrome inputs** have only one channel and correspond to black and white images.

Structure of CNNs: Visual

- Like regular NNs, CNNs are made of **multiple layers**
- Each layer takes some input volume and produces an output volume

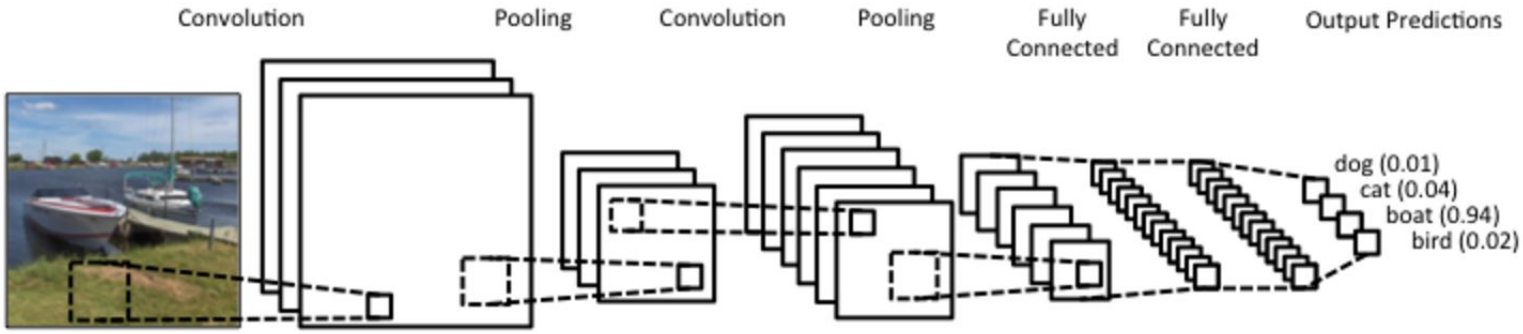


Structure of CNNs: Layers

- Input layer: holds the original image
- Convolutional layer: creates a feature map
- ReLU: an activation function
- Pooling: downsamples
- Final layer: fully connected layer
    - Outputs an N-dimensional vector (N being the number of classes)

Architecture: General Layer Patterns

Input → (Conv → ReLU) → MaxPool → FC Layers → Softmax



( ) - repeated layer

Formalizing different kinds of layers

- Though each layer takes a volume and gives a volume, the actual computation it performs depends on what kind of layer it is
- Common examples: **Convolutional**, **Pooling**, and **Fully Connected**

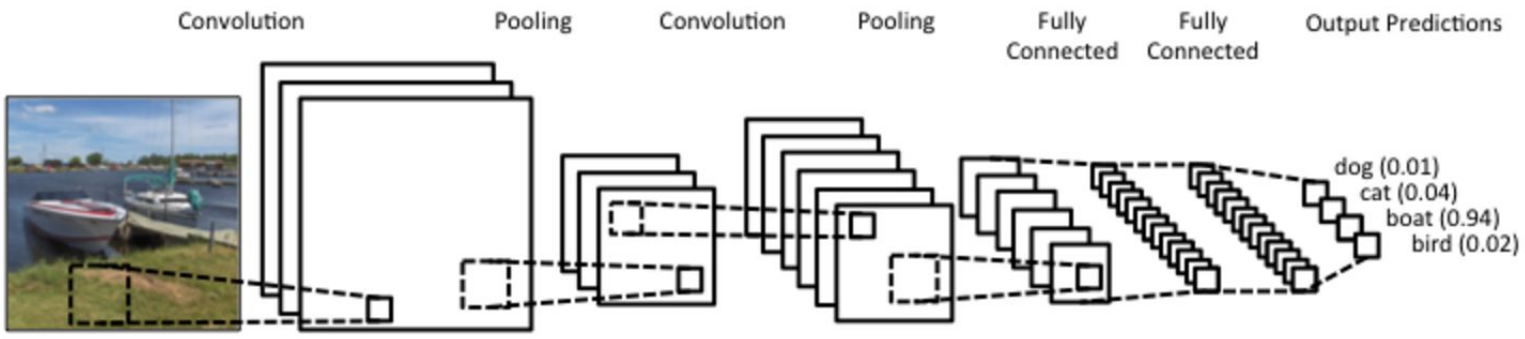
“

Questions?

5. Convolutional Layer

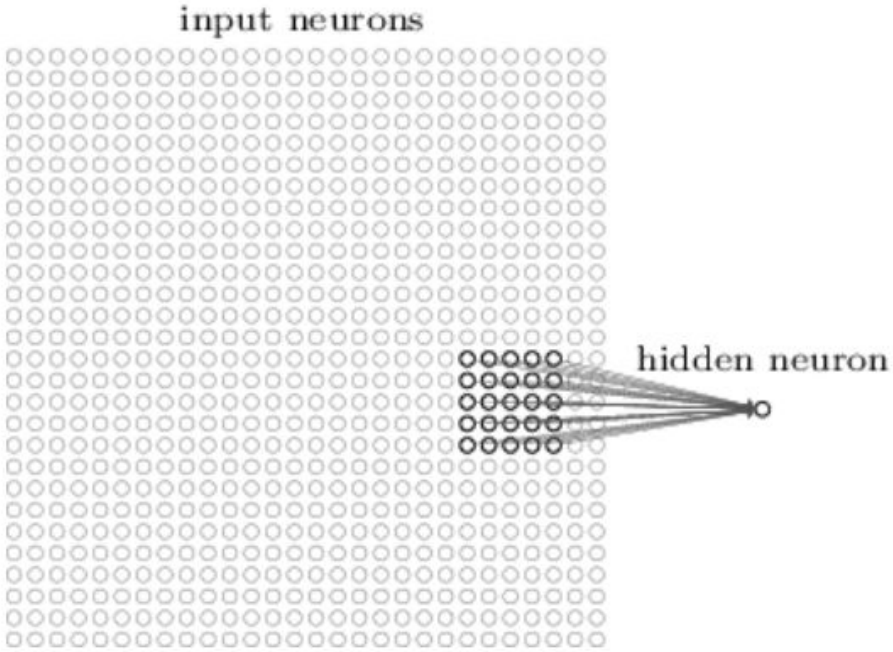
Architecture: General Layer Patterns

Input → [**Conv** → ReLU] → MaxPool → FC Layers → Softmax

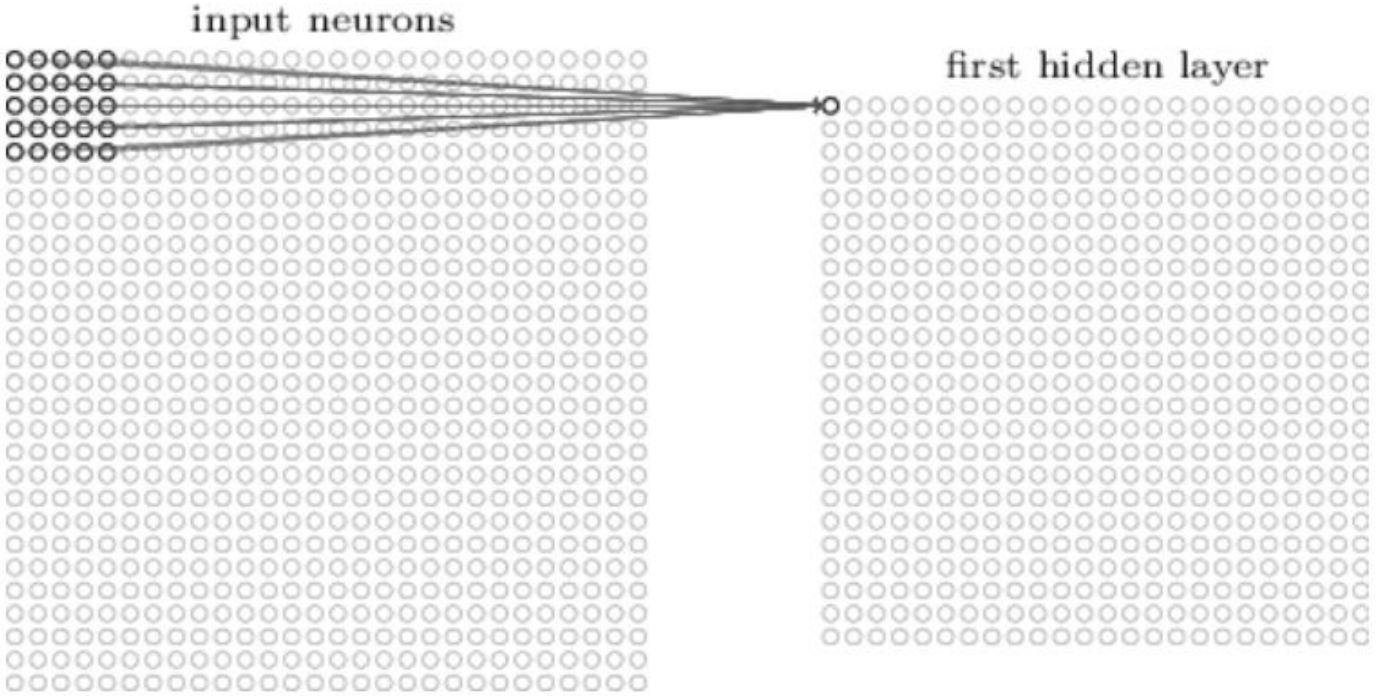


[ ] - repeated layer

Local Receptive Fields



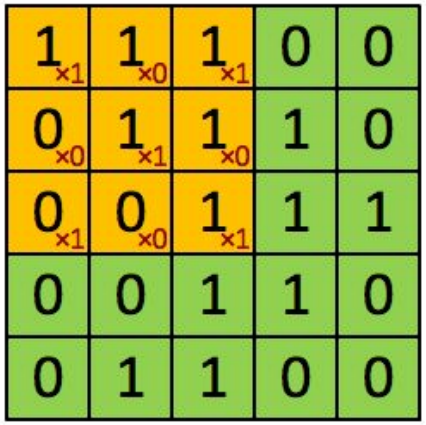
Convolutional Layer



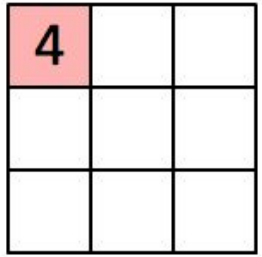
Filters - What do they do?

- A filter is a volume of weights that moves across (convolves) the input volume, performing element-wise multiplications (dot products) at every place it “stops”
- A filter moves with a **stride**. A stride of *n* means the filter moves by *n* positions before stopping to carry out a dot product.
- Every dot product results in a scalar. These numerous scalars make up the output volume

Filters – Visualization (ignoring depth)

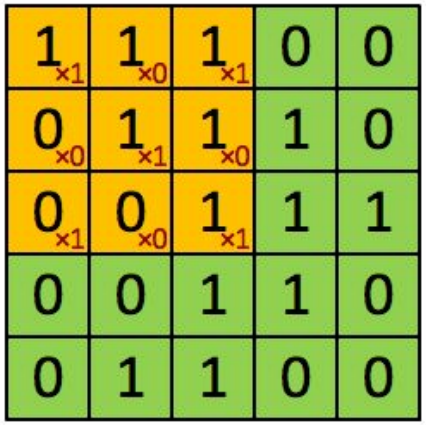


Image

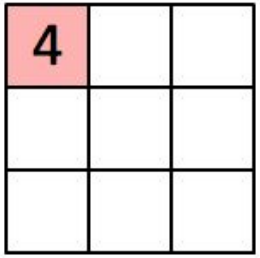


Convolved Feature

What is the stride of this filter in this example?

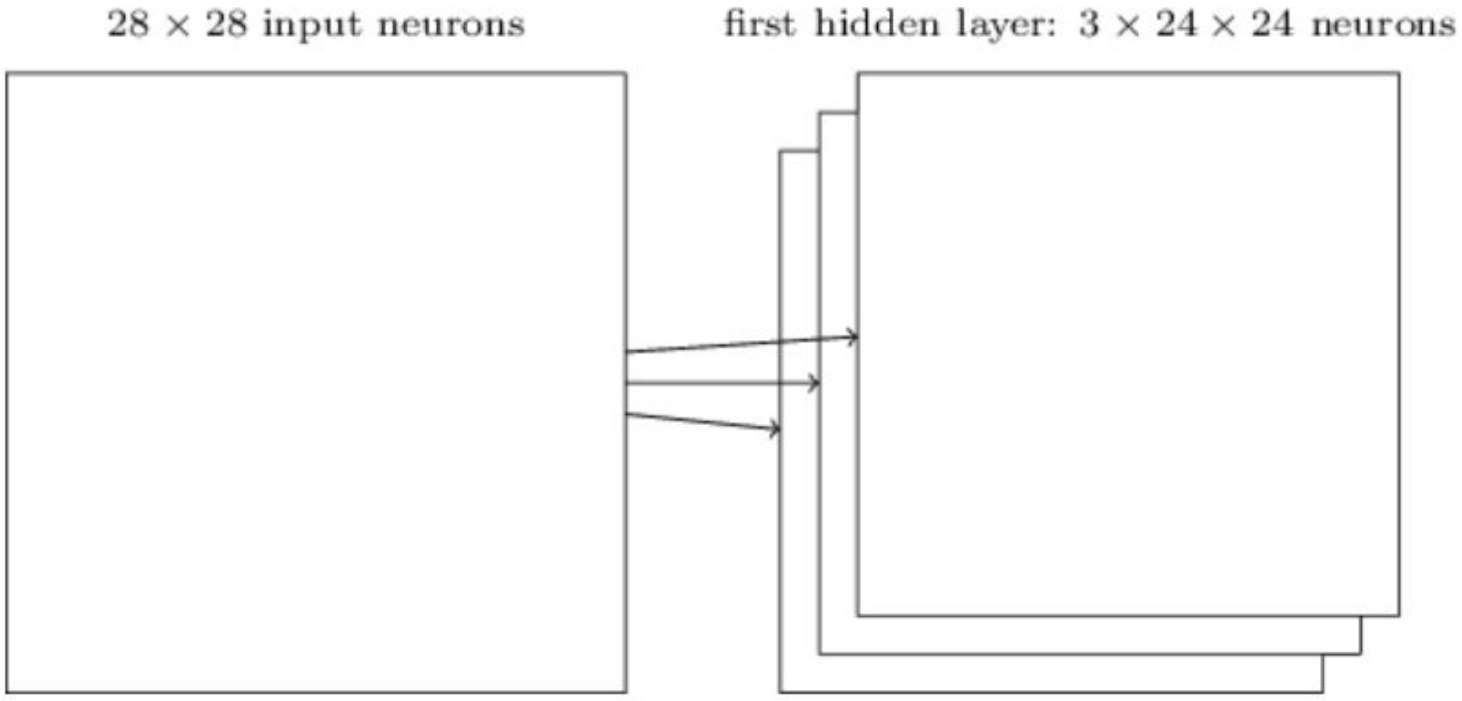


Image



Convolved Feature

Convolutional Layer (Feature Maps)



Convolutional Layer – Feature Maps

- A filter is a 3D volume too. It is the volume of ***weights*** that acts on an input volume
- This “window” has the same depth as the input volume, but its height and width can vary (the area of a filter is usually much less than the spatial area of the input volume)
- So if an input volume has the dimensions 28x28x10, the filter could have dimensions 5x5x10 or 7x7x10 and so on

acm.ai

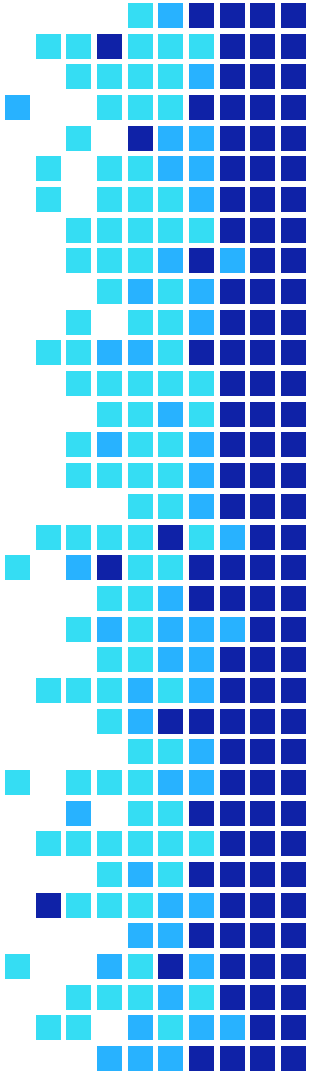
Discussion Time!

1. What sort of tasks is a convolutional neural network best suited for?

2. Consider a color image, what is the dimension of this input to a model?

3. What is a filter?

4. What is a stride?



Feature Maps - What do they do?

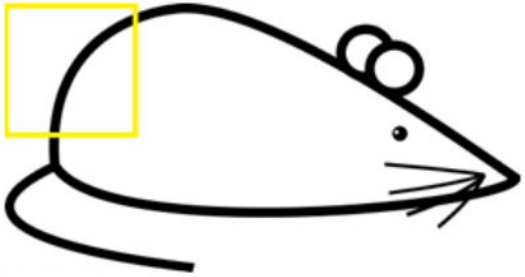
- A filter “scans” the input volume, resulting in a **2D** output.
- A single filter acts on a volume and outputs an “area” - Why?
- How then do we get an output **volume**?
- We use multiple filters for a single input volume.

Convolutional Layers

- A Convolutional layer consists of multiple filters of the same shape acting on an input volume to produce an output volume.
- Each filter is a set of weights that “looks for features” in the input volume, producing a higher value (greater activation) if that feature is found in a particular region.
- A bias vector of size K (number of filters) is also used

Visualization

Original image



Visualization of the filter on the image

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Why filters help

- Filters reduce total number of weights - each “slice” of output volume corresponds to single filter - single set of weights
- Filters also allow detection of relationships between pixels
- Filters look for features - using the same filter across image makes sense - a feature could be present at any region of the image

Hyperparameters to keep track of

- Number of filters (K) (output volume depth)
- Stride of filters (S) (how many pixels to skip while moving)
- Zero-padding (P) (we'll see this now)

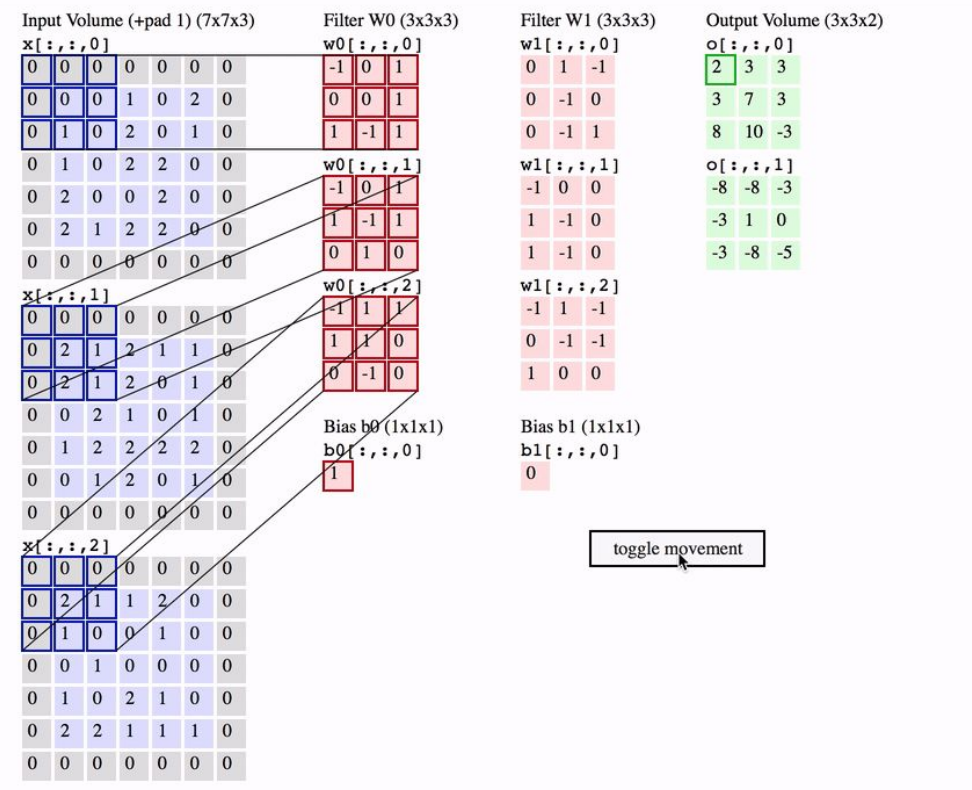
Zero Padding

- May want the output volume to have particular spatial arrangement
- Zero padding can help.
- Add zeros around the edges of the input volume (“pad”)
- Output volume size is determined by:

$$floor(\frac{W - F + 2P}{S} + 1)$$

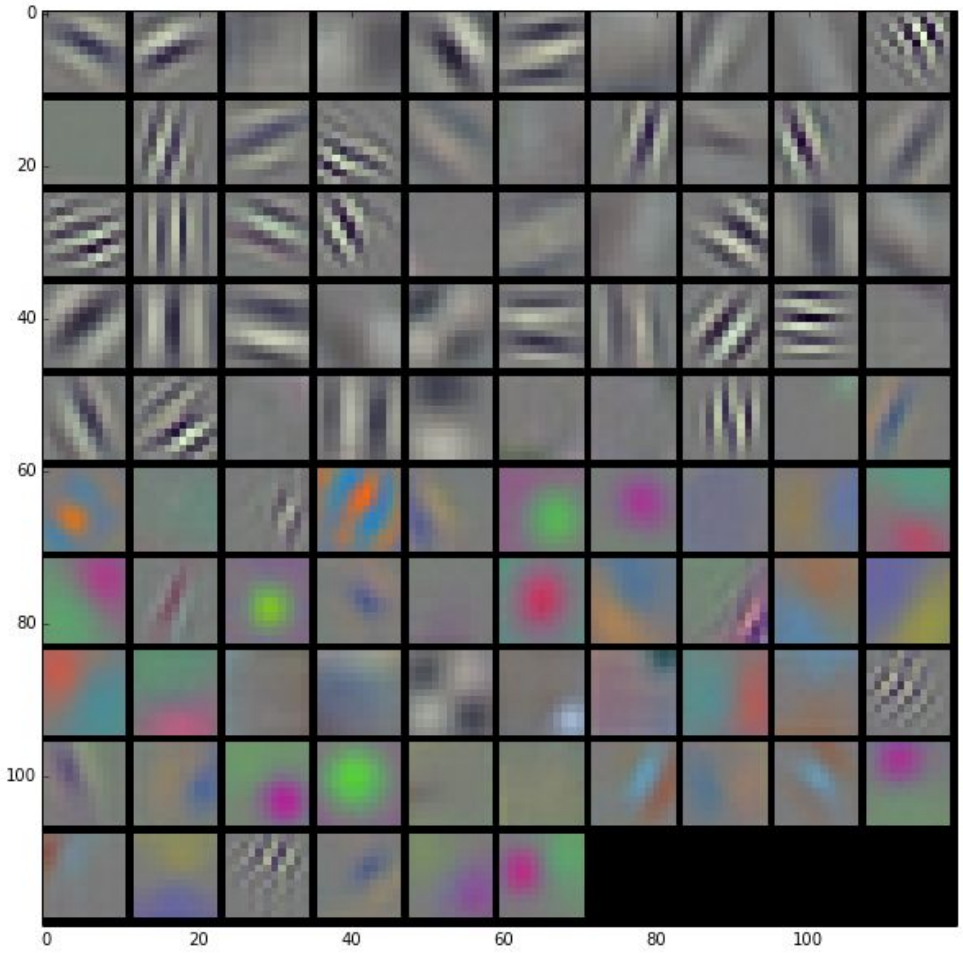
- *W* - Input volume size, *F* - Filter size, *P* - Zero Padding
- *S* - Stride

Conv Layer - Visualization (multi filter)



acm.ai

Filters - First layer (ImageNet)



acm.ai

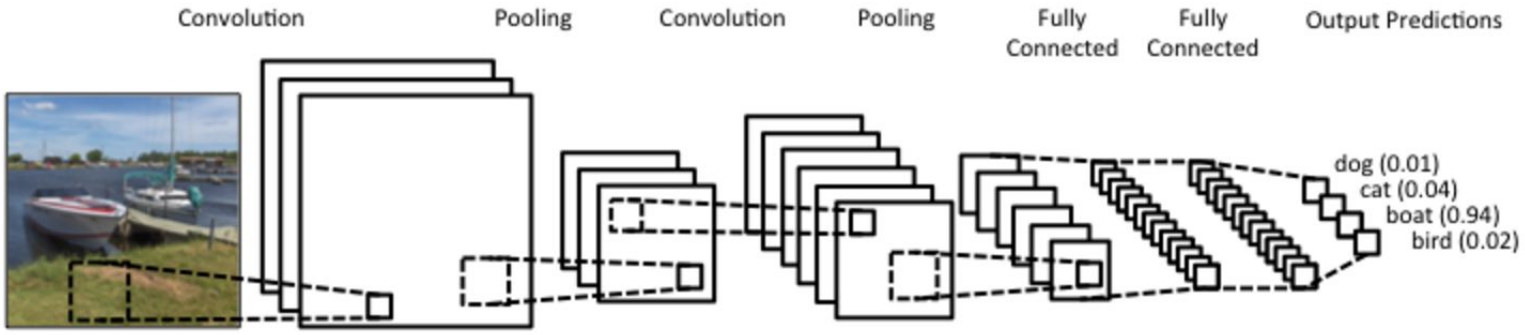
“

Questions?

6. Other Layers

Architecture: General Layer Patterns

Input → [Conv → **ReLU** → MaxPool] → FC Layers → Softmax

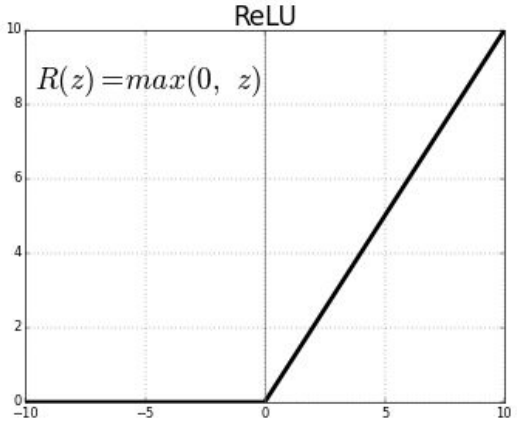


[ ] - repeated layer

Non-linear Layer (ReLU)

- What exactly is ReLu and what about its shape is important?

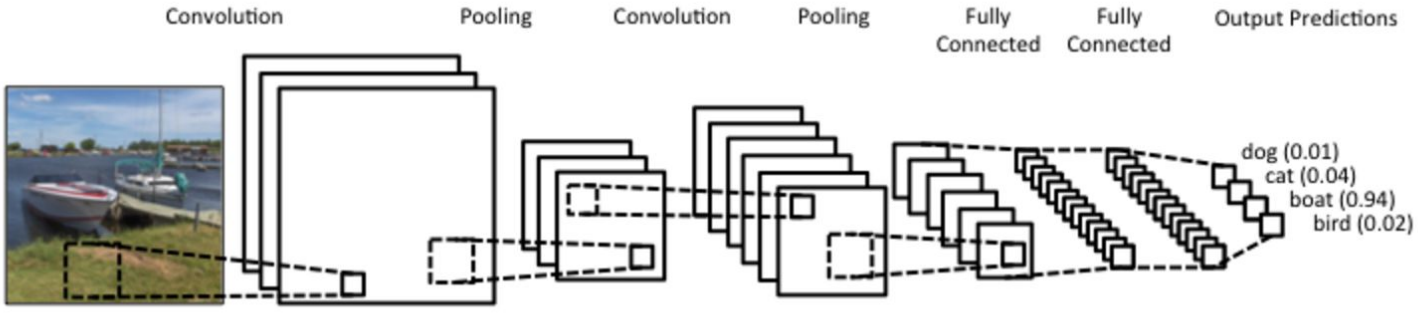
Non-linear Layer (ReLU)



- We apply ReLU to the output of convolutional layers
- Why ReLU?
  - derivative of sigmoid has a small range (0, 0.25]
    - leads to vanishing gradient
  - ReLU gradient values are larger
  - ReLU can “kill” neurons (negative output), creating sparseness

Architecture: General Layer Patterns

Input → [Conv → ReLU → **MaxPool**] → FC Layers → Softmax



[ ] - repeated layer

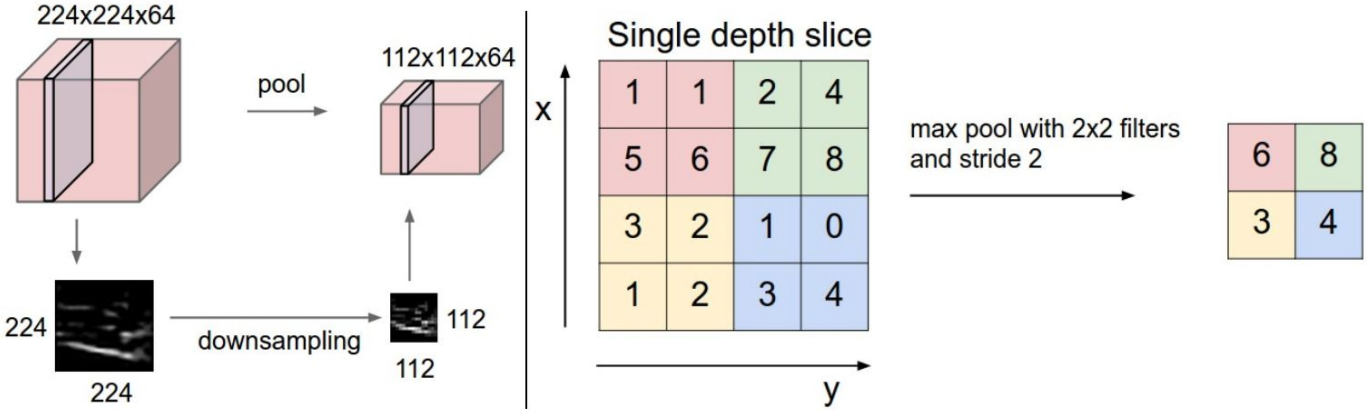
Pooling Layers

Objectives:

1. reduce spatial size of features (height and width)
2. reduce amount of parameters and computation
3. control overfitting

Max Pooling

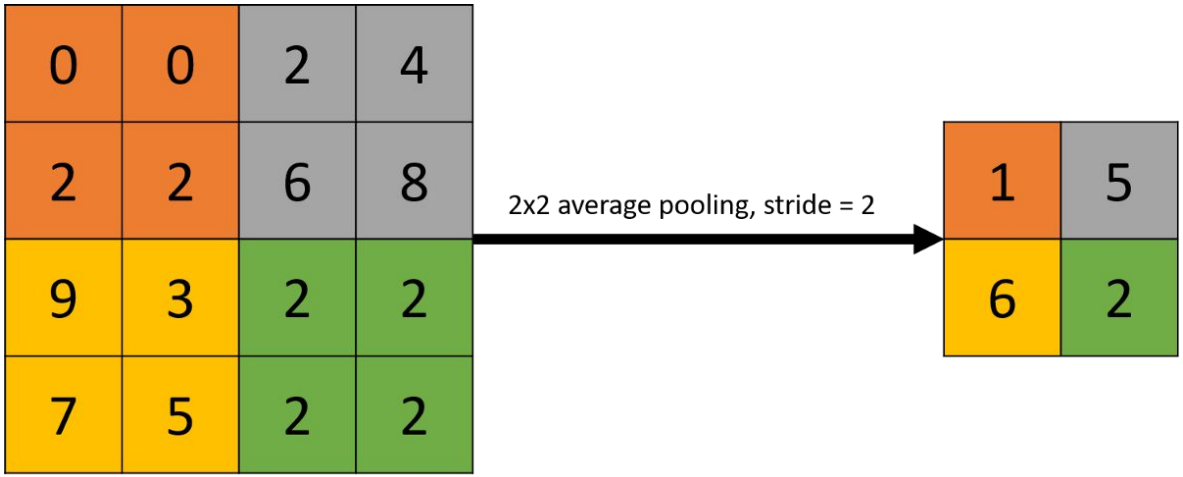
Take the maximum value from each f x f section.



Filter size = 2, stride = 2, downsample by 75%

Average Pooling

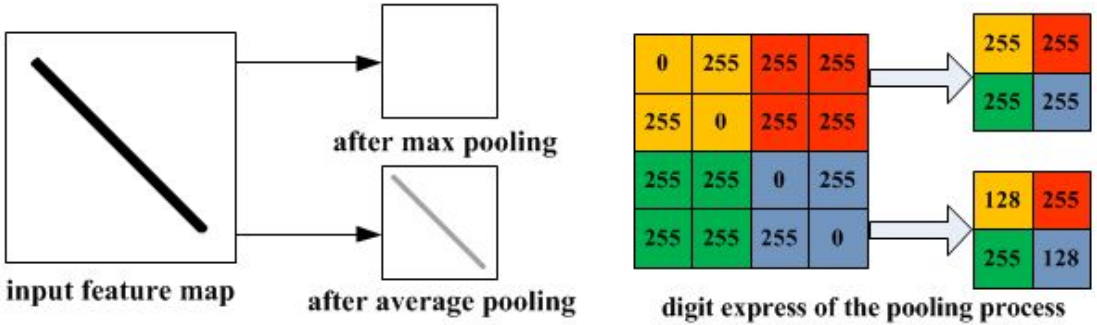
Take the average value from each f x f section.



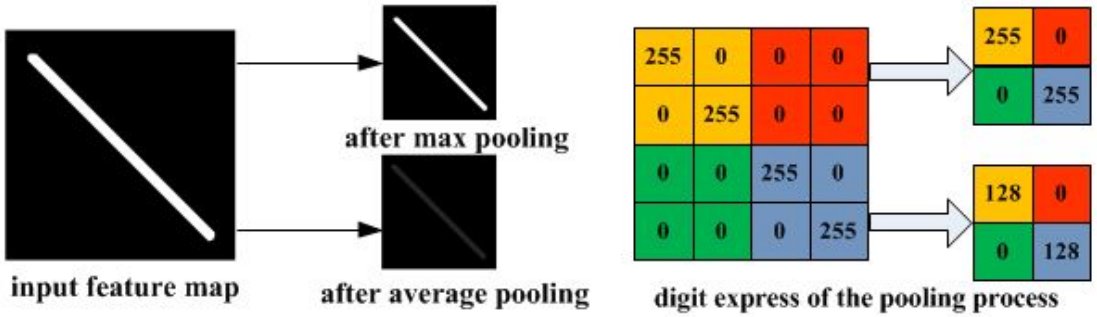
Pooling Layer Common Practices

- Possible hyperparameters for filter (F) and stride (S):
  - F = 2, S = 2
  - F = 3, S = 2 (Overlapping pooling)
- What about padding?
  - uncommon to use padding in the pooling layer
- Of the pooling layers, max pooling generally performs best in practice

Pooling Comparison



(a) Illustration of max pooling drawback



(b) Illustration of average pooling drawback

Pooling Layer Questions

1. What is the purpose of a pooling layer?
2. Why might it be okay that we lose information during pooling?

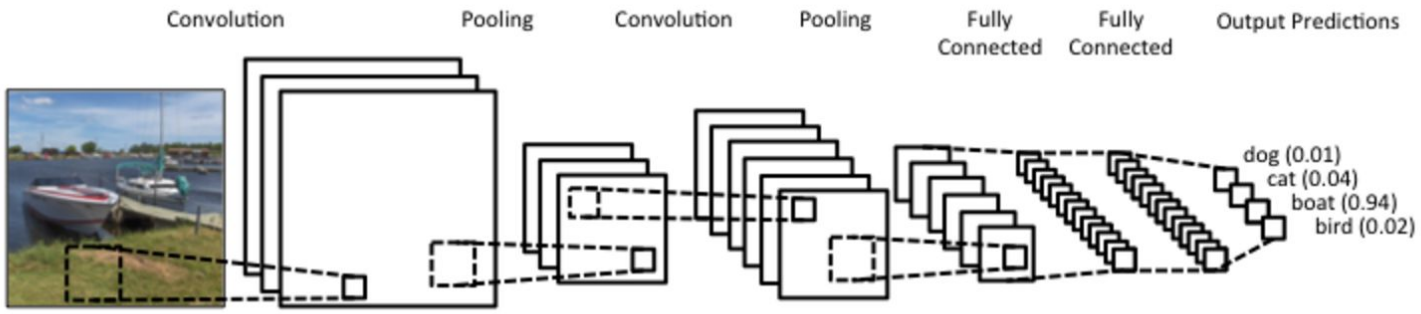
More Information:

https://stats.stackexchange.com/questions/387482/pooling-vs-stride-for-downsampling

https://stackoverflow.com/questions/44666390/max-pool-layer-vs-convolution-with-stride-performance

Architecture: General Layer Patterns

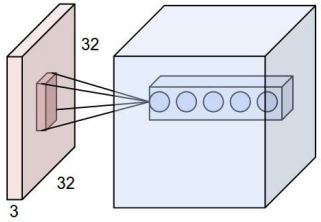
Input → [Conv → ReLU] → MaxPool → **FC Layers** → **Softmax**



[ ] - repeated layer

Making Predictions

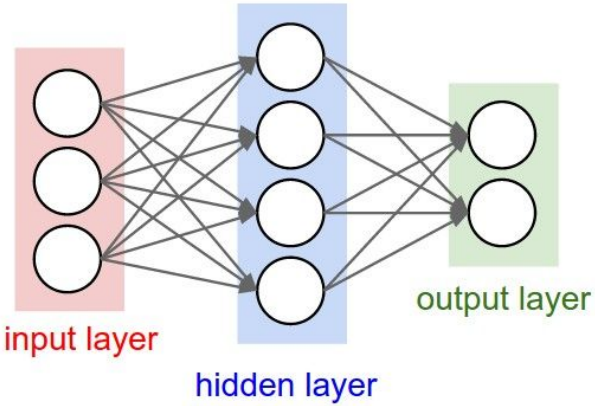
- Need a way to compute predictions for each class
- For vanilla neural networks we used softmax
  - Input is vector where each element corresponds to a class
  - Outputs vector of probabilities for each class
- Convolutional layers don't output vectors
  - Outputs are 3D (h x w x c)
  - We want to convert that output to a vector



Fully Connected Layer

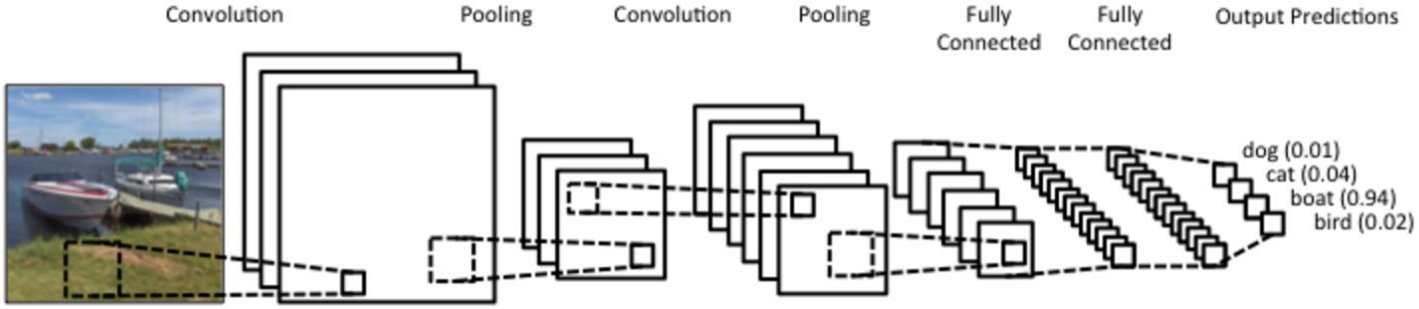
Fully connected (FC): neurons have full connections to all activations in the previous neural networks

- Just like layers in a vanilla neural network
- We “unroll” (flatten) the output of the max pooling layer before the FC layers
- Forward propagate the flattened output through the FC layers like in a vanilla neural network



Architecture: General Layer Patterns

Input → [Conv → ReLU] → MaxPool → FC Layers → Softmax



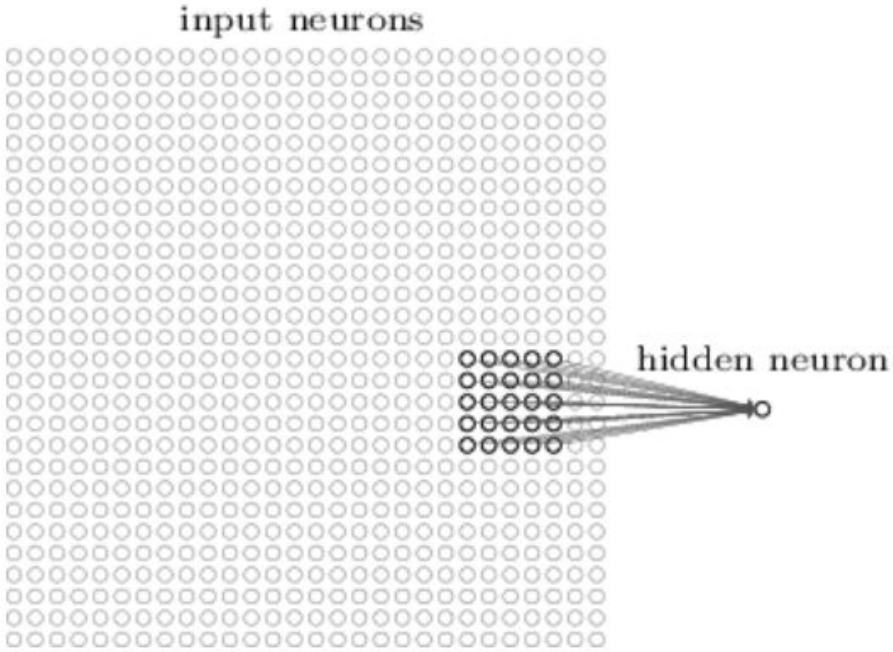
[ ] - repeated layer

Advantages of CNN compared to FFNN

- Sparse Connectivity:
  - A value in the output volume is influenced by a subset of the input volume.
- Parameter Sharing:
  - The same filter is convolved with the input volume to produce multiple output values
- Equivariant Representation:
  - Translation in the input volume is equivalent to translation in the output volume

More information at: https://cedar.buffalo.edu/~srihari/CSE676/9.2%20CNN-Motivation.pdf

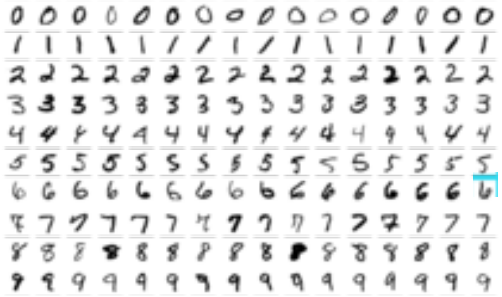
Local Receptive Fields



acm.ai

MNIST

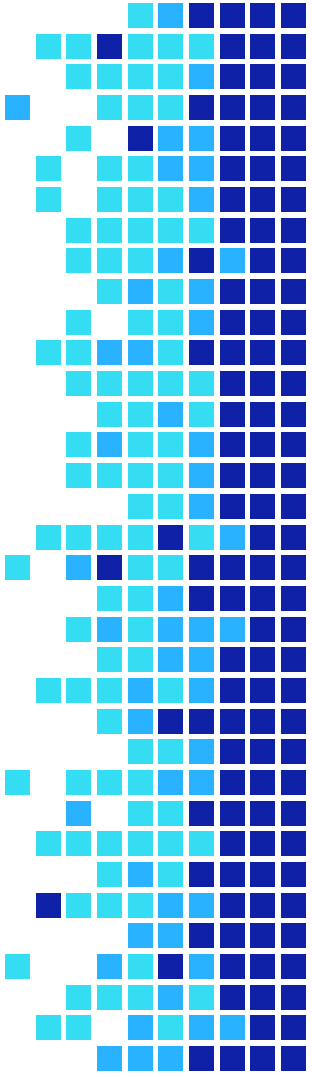
- Large database of handwritten digits
- Commonly used to train/test image processing methods like CNNs
- Full database contains 60,000 training images, 10,000 test images
- Researchers have been able to accomplish error rates as low as 0.23% with CNNs on this dataset.



acm.ai

Discussion Time!

1. Why are filters used for CNNs?
2. What is the intuition behind zero padding?
3. Why ReLU?
4. Why does max pooling generally perform better than average pooling in practice?



Discussion Answers

- https://tinyurl.com/advworksheet3

“

Questions?

Other Resources

- Beginner's Guide to Understanding CNNs
- Stanford CS231n: Convolutional Neural Networks for Visual Recognition
- 2017 CS231n Lectures on YouTube
- Stanford UFLDL CNN Tutorial
- CNN Video Tutorial (Brandon Rohrer)
- CNN Chapter from NN and Deep Learning Book

acm.ai

Thank you all for coming!

Anonymous Feedback: forms.gle/Eg1Ub8s7cMUuSuDc9

Facebook Group: www.facebook.com/groups/uclaacmai/