INFO251 – Applied Machine Learning

Lab 10

Suraj R. Nair

Topics

- Neural Networks (Practicalities)
  - Deep learning boilerplate
  - Hyperparameters
  - Transfer learning / fine tuning

  - Code

Boilerplate / Recipe

- Load data
- Specify model
- Parameters you need to specify:
  - Number of epochs
  - Batch size
- Dataloaders
- Loss function
- Optimizer
- Train / evaluate

Batch Sizes

- Determines training speed
  - Usually, the ideal batch size is the largest batch size that hardware supports
- Validation performance:
  - Ideally independent of batch size (especially if other hyperparameters are tuned)
- Do batch sizes need to multiples of 2?
  - Andrew Ng: 64, 128, 256, and 512
  - Nvidia: multiples of 8
  - Good overview article (ideally: focus on training speed, accuracy and memory consumption)

Learning Rates

- “A good learning rate is like Goldilocks: not too hot, not too cold, but just right.” — Geoffrey Hinton
- Speed vs overfitting
  - High LR → overfitting
  - Low LR → slow learning

Picking a good learning rate

- Start with a small learning rate and increase it gradually until you find the best value.
- Learning rate scheduler:
  - Linear / cosine decay
  - Tuning is important!
- Experiment!

Optimizers

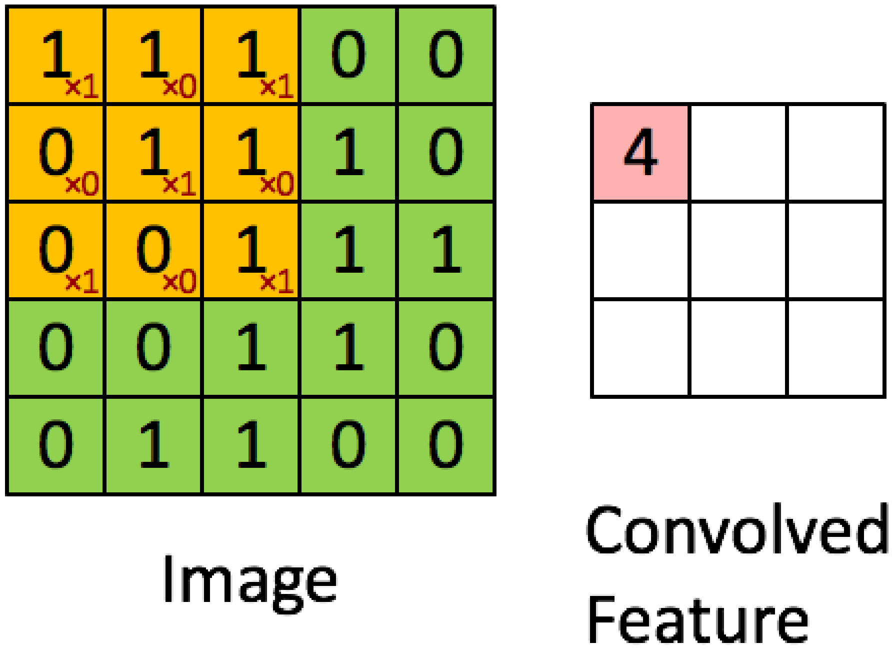
- No one size fits all solution
- Often hard to compare across different optimizers
- In general, pick popular / common optimizers as a starting point
  - SGD with momentum (Google recommends the Nesterov variant)
  - Adam
  - Nadam

CNN Model Architecture

- Common: Convolutions + ReLU activation + Pooling
- Small kernel sizes are better (3 x3, 5 x5)
- A stack of convolutional layers with small kernel sizes tends to work better than one conv. layer with a large kernel
- For convenience:
  - Use conv. layers with stride = 1
  - Leave spatial downsampling to pool layers
  - Use padding!

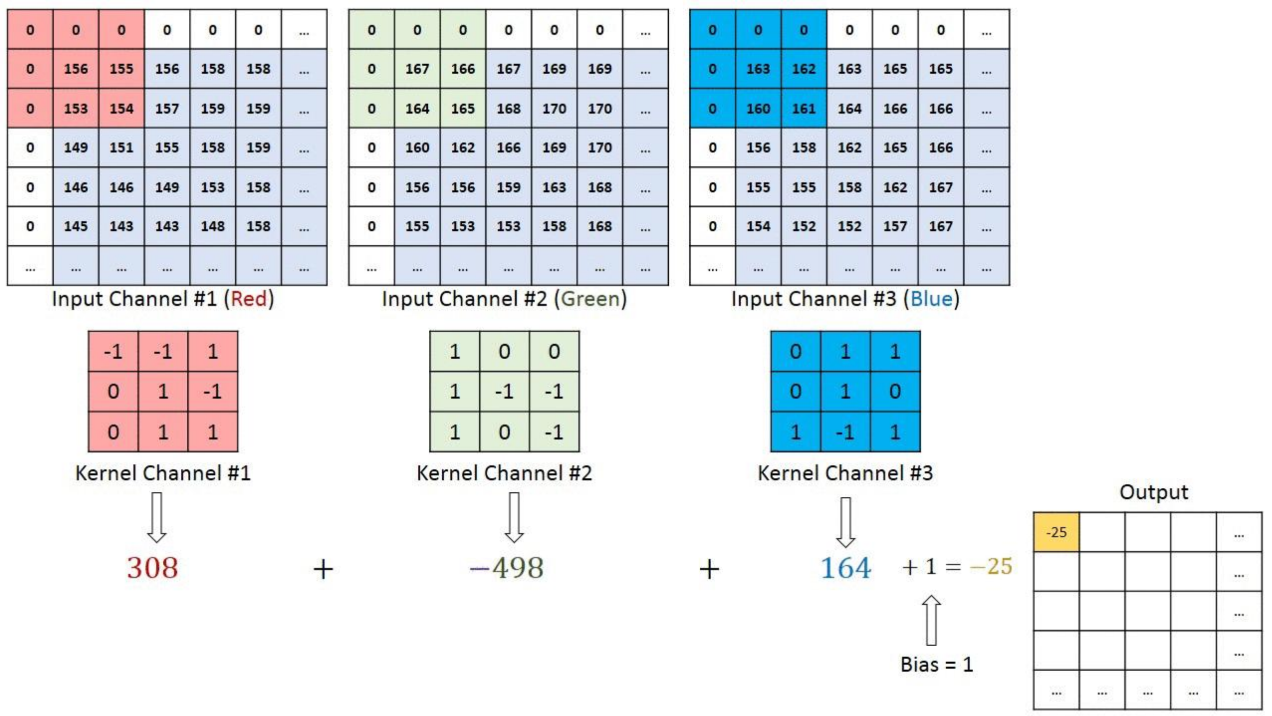
Convolutional Layers

- Goal: Capture the spatial dependencies in parts of an image
- Multiply a **kernel matrix** (“**filter**”) *k* by subsets of the input image
    - Hyperparameter: Size of *k* (often 3x3, 5x5, or 7x7)
    - Learn: The weights of *k*
- **Stride**: How to shift the kernel matrix
    - Hyperparameter: Stride value (integer)



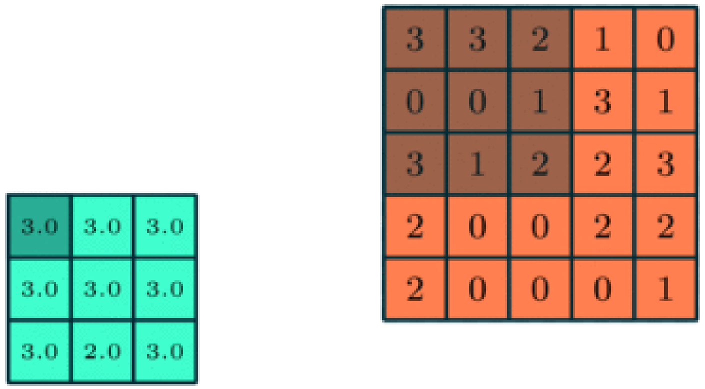
Convolutional Layers

- **Channels:** Number of ”layers” in the input image
  - Grayscale: 1 channel
  - RGB: 3 channels
  - RGBA: 4 channels
- Same filter size and stride length, but each channel has different weights
- Outputs of channels are summed up



Pooling Layers

- Goal: Reduce size of convolved layer to decrease compute cost
- Again, operates kernel matrix *k* over the convolved matrix
  - Hyperparameter: Size of *k* (usually 2x2)
  - Hyperparameter: Stride width (usually 2)
- **Max pooling**: Return maximum value in area covered by kernel
- **Average pooling**: Return average value in area covered by kernel



Convolutional Neural Network Structure

- Convolutional layer
- Pooling layer
- Convolutional layer
- Pooling layer

Repeat convolution followed by pooling any number of times. Option to add dropout after pooling.

- Flatten
- Fully connected layer(s) (activation: sigmoid/tanh/relu)

Add any number of fully connected layers. Option to add dropout.

- Output layer (activation: determined by problem type)

Dimensions

- In general, output size from a convolutional layer: $[(N - F + 2P)/S] + 1$
  - N: Input image dimensions (e.g. **28** X 28)
  - F: Filter / Kernel dimension(e.g. **3** X 3)
  - P: Padding (default 0)
  - S: Stride (default 1)
- Number of parameters associated with a convolutional layer: $((f * f * d)+1)* k)$
  - d is the number of filters/ channels in the previous layer
  - k is the number of filters in the current layer

Example: VGG-16 (Simonyan & Zisserman, 2015)