THE UNIVERSITY OF MELBOURNE

Convolutional Neural Networks II

Semester 2, 2025

Kris Ehinger

Outline

- Downsampling
- Regularisation
- Training an image recognition CNN
- CNN results

Learning outcomes

- Implement max pooling and explain how/why downsampling is used in CNNs
- Implement regularisation methods and explain how/why they are used in CNNs
- Train a CNN and explain the design choices involved in the training process

Downsampling in CNNs

Convolutional neural network

"AlexNet": Krizhevsky, Sutskever, & Hinton (2012)

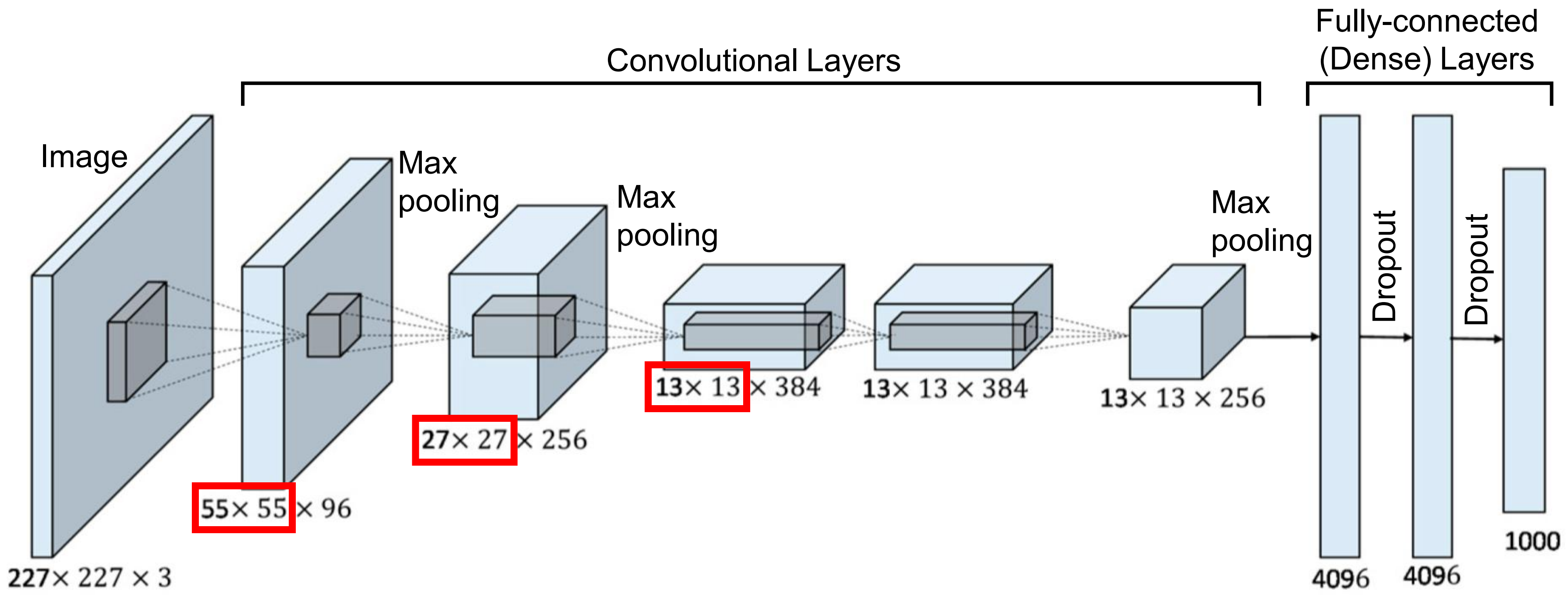


Image: Han, Zhong, Cao, & Zhang (2017)

Downsampling in CNNs

- It’s common to downsample convolution layer output
- Reduces output size and number of computations needed in later layers
- Can also improve tolerance to translation – small changes in input won’t change downsampled output

Strided convolution

- Convolutional stride = distance between successive convolution windows
- In CNNs, stride can be >1
- Assuming no padding:
  - output_size = ceil((input_size - kernel_size + 1)/stride)
- With padding:
  - output_size = ceil(input_size/stride)

Strided convolution

- Advantage
  - Efficient – higher stride means fewer convolution operations
- Disadvantage
  - Kernel window skips over parts of the image, so important image features could be missed

Convolutional neural network

"AlexNet": Krizhevsky, Sutskever, & Hinton (2012)

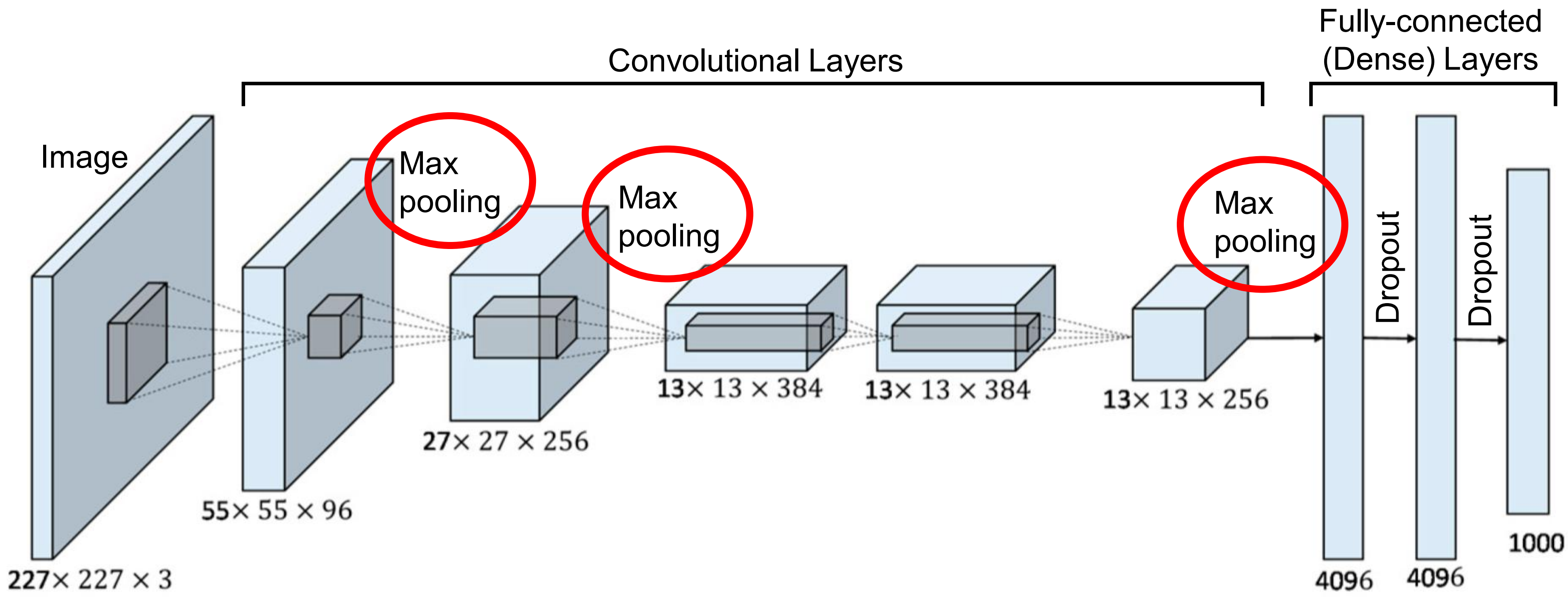
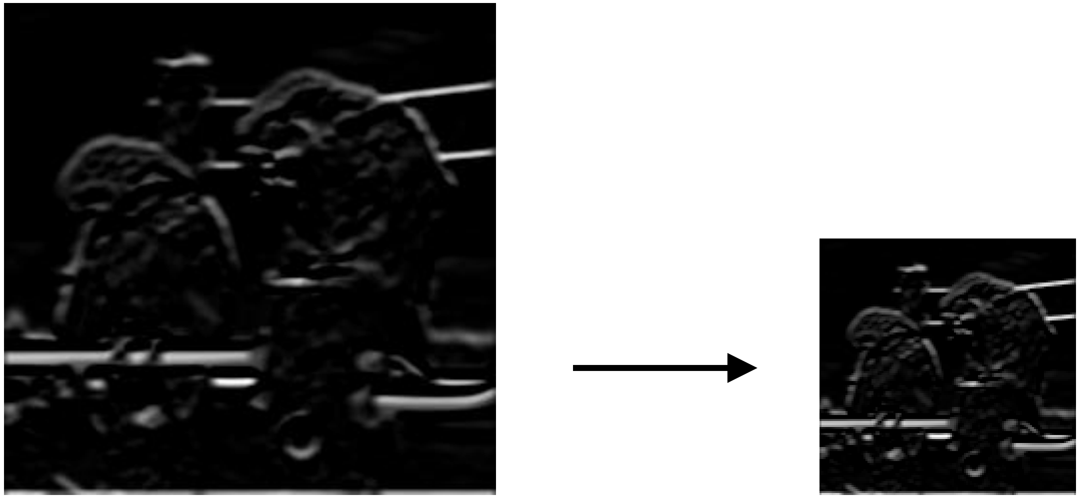


Image: Han, Zhong, Cao, & Zhang (2017)

Max pooling

- After convolution, each activation map is separately downsampled
- Max pool stride determines the amount of downsampling (output_size = input_size/stride)



Max pooling

- Within a given window in the activation map, take the highest value and discard the rest

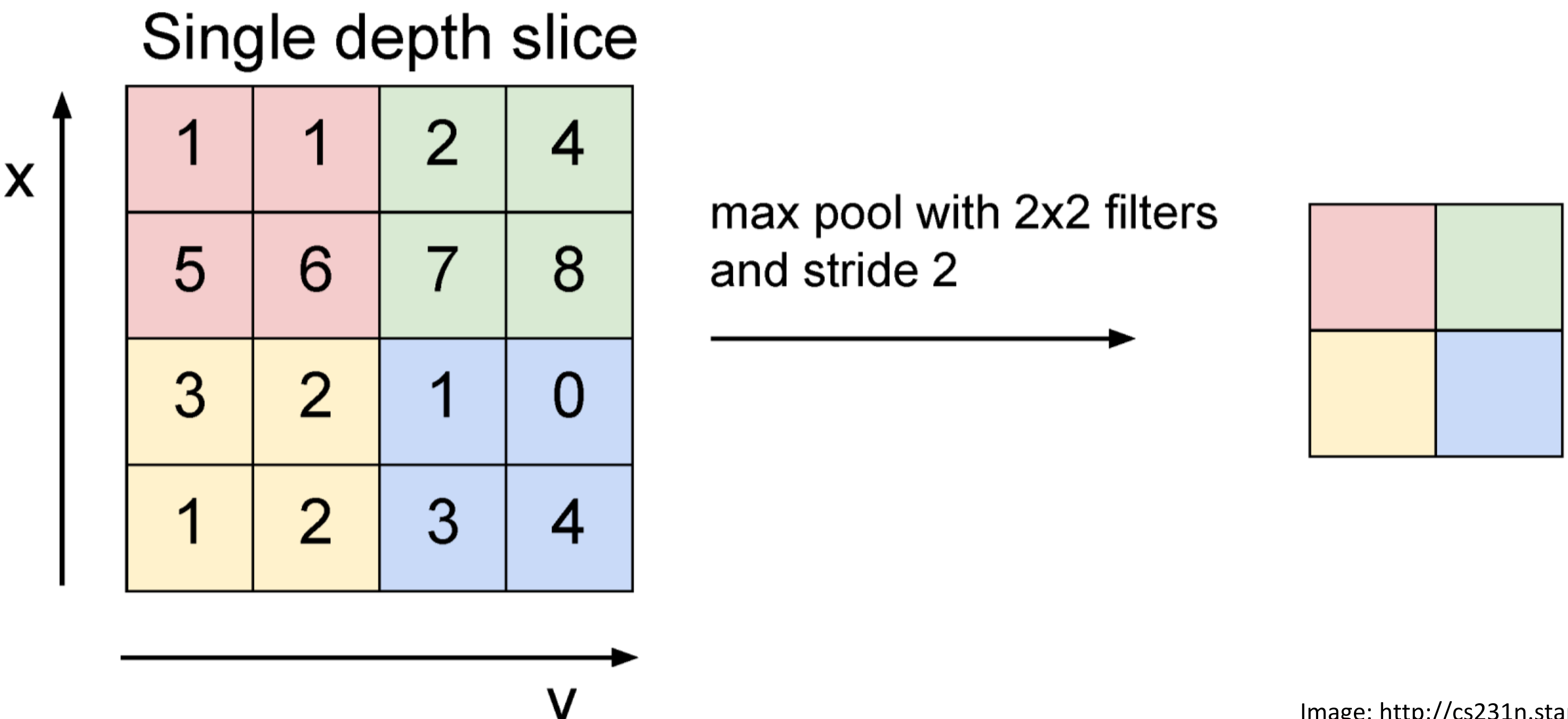


Image: http://cs231n.stanford.edu/

Average pooling

- Within a given window in the activation map, average the values

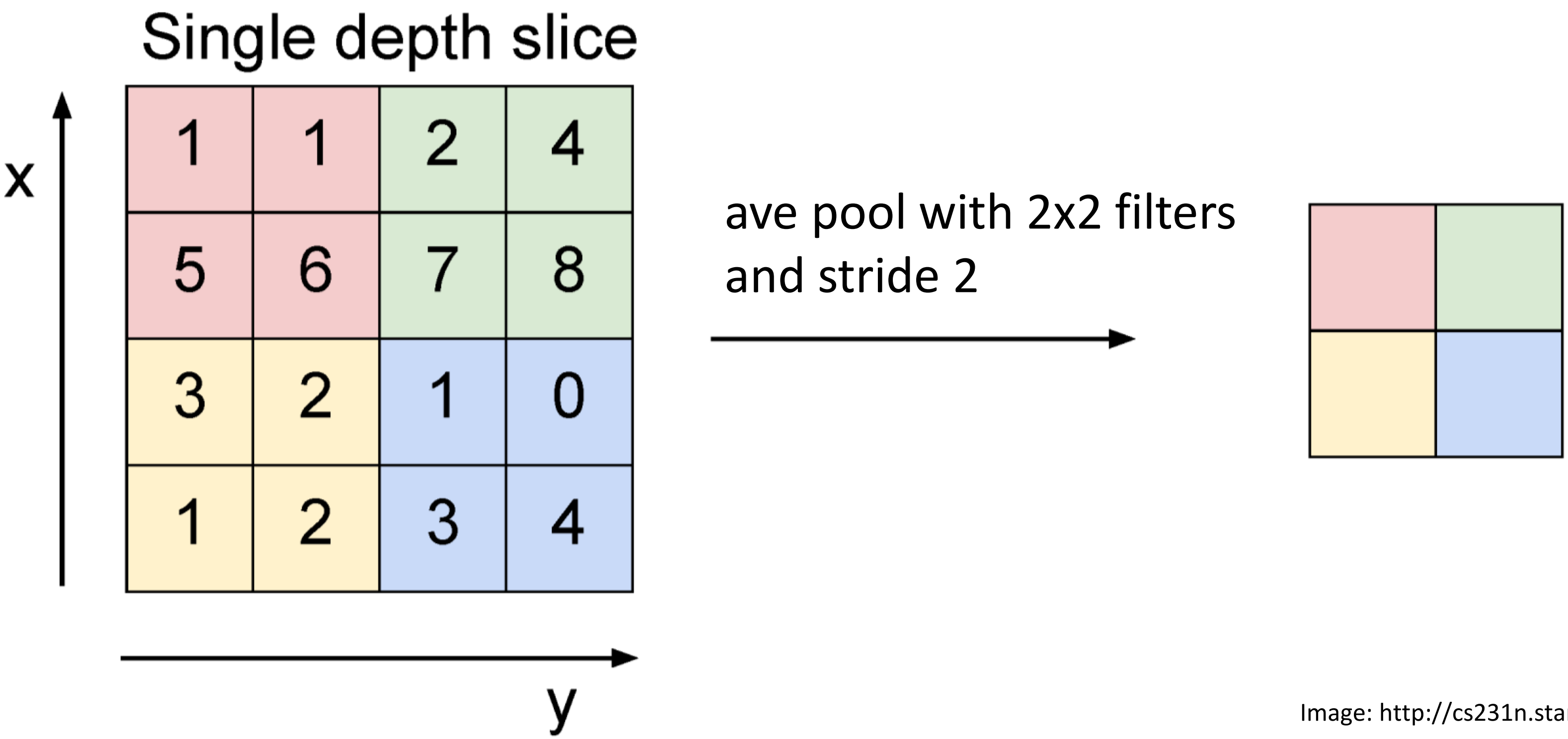


Image: http://cs231n.stanford.edu/

Max/average pooling

- Advantage
  - Max pooling is most likely to preserve the most important features, compared to strided convolution or average pooling
- Disadvantages
  - Average pooling “blurs” over features; important features may be lost
  - Pooling is slower than strided convolution

Summary

- Downsampling is common in CNNs to make computation more efficient in later layers and increase translation invariance
- Methods include strided convolution, max pooling, and average pooling

Regularisation in CNNs

Regularisation

- Due to the very high number of parameters, CNNs are prone to overfitting, even on large datasets
- Regularisation is usually needed to reduce overfitting
- Common options:
  - L1 or L2 regularisation
  - Dropout
  - Early stopping

L1, L2 regularisation

- Adds an additional term to the loss function that encourages smaller values for the network parameters
- L1 regularisation adds the term: $\sum_i |\theta_i|$
  - Penalise the sum of the absolute value of all parameters
  - Encourages sparse representation – many parameters should be 0
- L2 regularisation adds the term: $\sum_i \theta_i^2$
  - Penalise the sum of the squares of all parameters
  - Encourages small (but not 0) parameters

L1, L2 regularisation

- Free parameters when adding regularisation:
  - How much weight to give regularisation term vs. other terms in the loss function
  - Which layers to include in regularisation – all layers or just later layers?
  - Which parameters to include – sometimes only weights are included, not biases
- Adding regularisation tends to slow down training
- Too much regularisation can result in underfitting

Convolutional neural network

"AlexNet": Krizhevsky, Sutskever, & Hinton (2012)

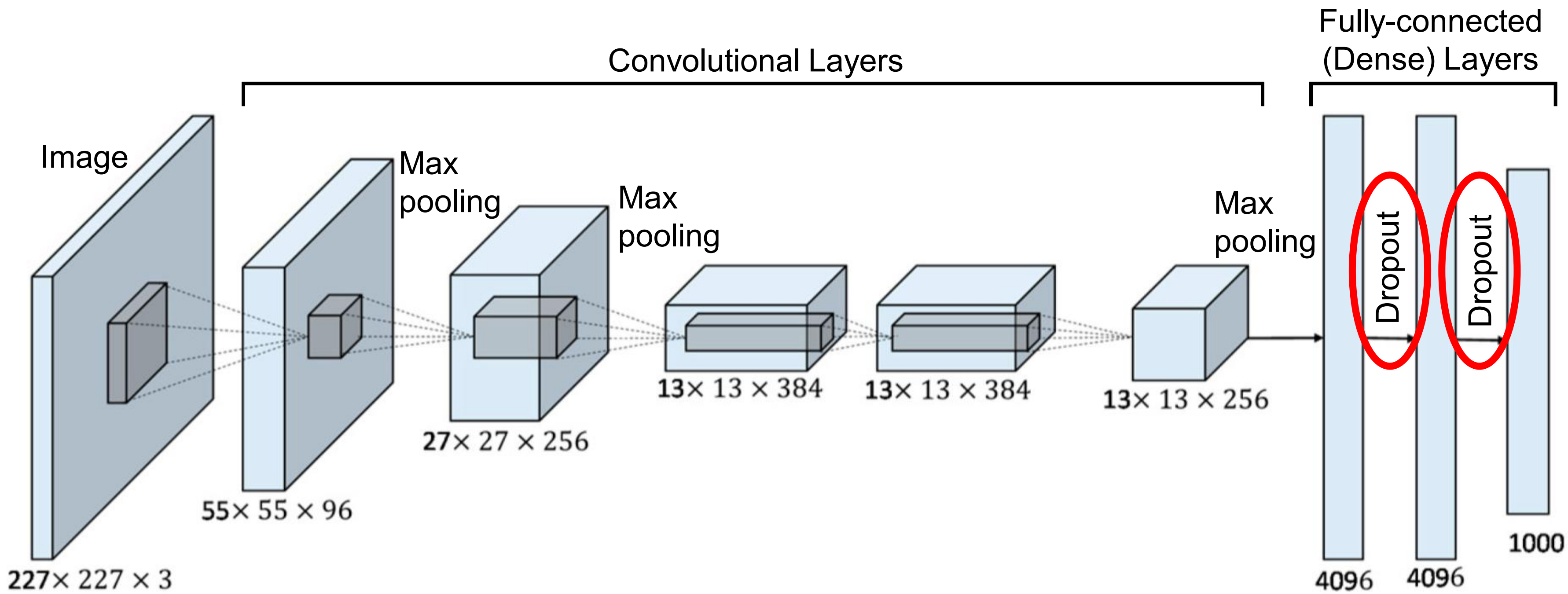


Image: Han, Zhong, Cao, & Zhang (2017)

Dropout

- Randomly discard some neurons (set output = 0)
- Forces neurons to find useful features independently of each other
- Effectively, trains multiple architectures in parallel

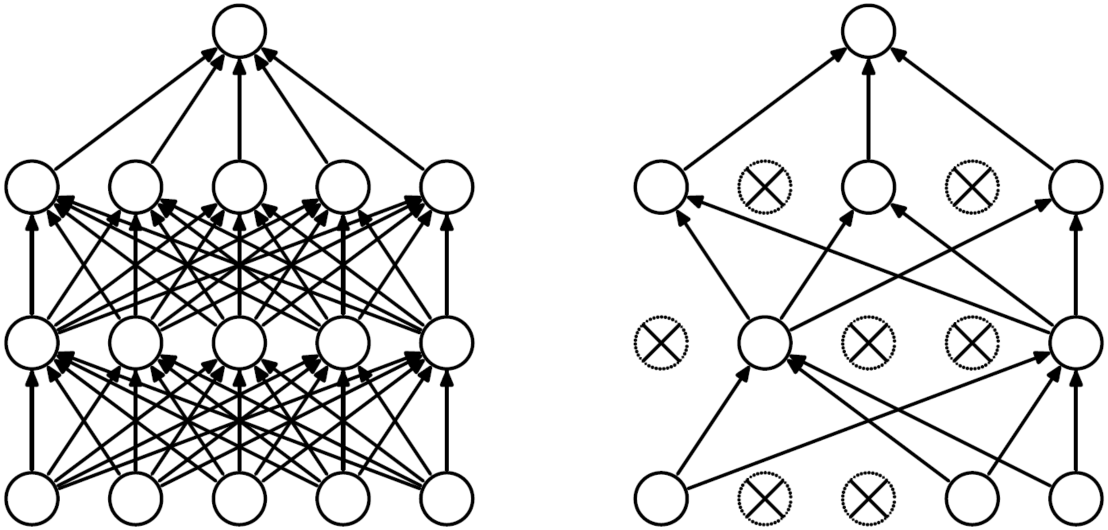


Image: Srivastava, Hinton, Krizhevsky, Sutskever, Salakhutdinov (2014)

Dropout

- What percentage of neurons to drop is a free parameter (e.g., drop 50% or drop 20%)
- Can be applied to all layers, or just later layers
  - Different dropout percentages can be applied to different layers – typically later layers would have more dropout
- Adding dropout tends to slow down training
- Dropout is *only* used in training – when evaluating the network on new data (validation/test), all neurons are active

Early stopping

- Stop training the network when it shows signs of overfitting
- Monitor performance on a **validation** set
  - Subset of data not seen in training and not included in test set
  - During training, periodically check model's performance on the validation set – a decrease suggests overfitting
- Encourages smaller values for network parameters by keeping them close to their initial values (which are typically near 0)

Summary

- Regularisation is usually necessary to prevent overfitting
- Common options: L1 or L2 regularisation, dropout, and early stopping
- Frequently unclear which method (or combination) will work best for a given optimisation problem, so it's common to experiment and combine them

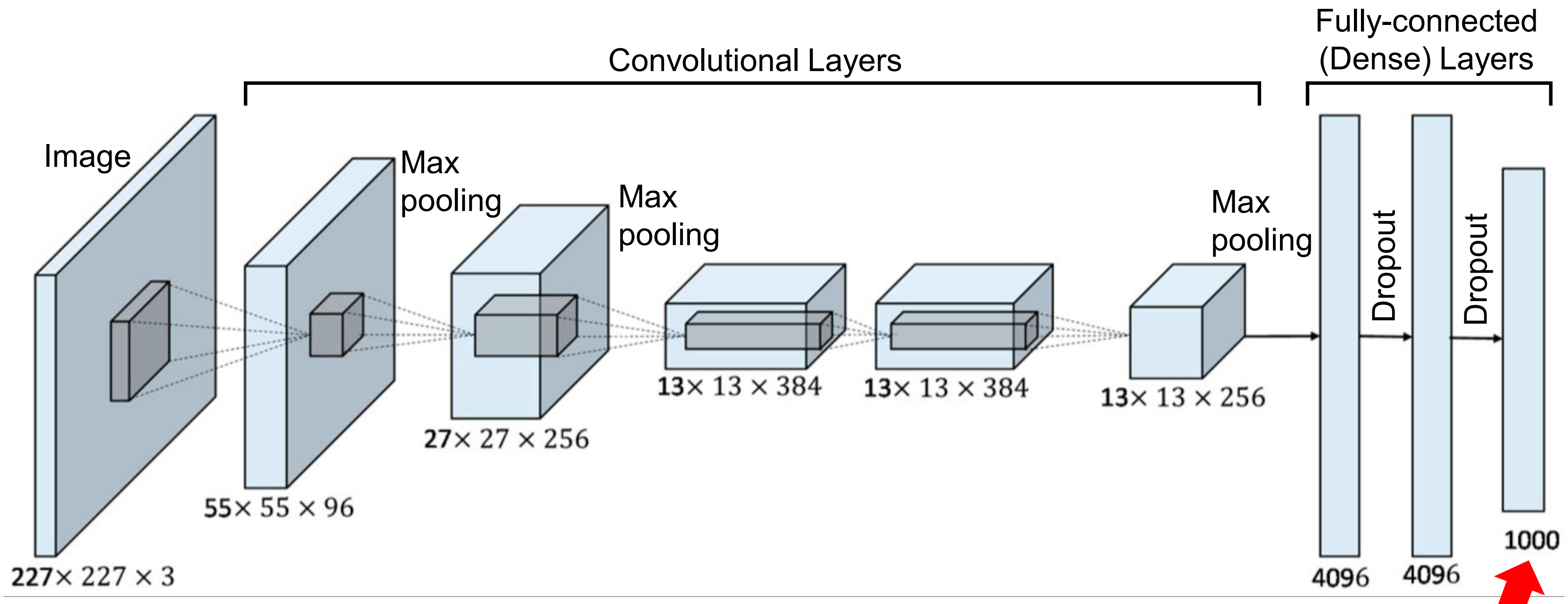
Training an Image Recognition CNN

CNN overview

- Typical architecture for image recognition:
  - Some number of convolutional layers, with downsampling
  - One or more fully-connected layers
  - Softmax output with cross-entropy loss
- Basic idea:
  - Do **feature embedding** in convolutional layers (transform images from pixels to useful high-level features)
  - Fully-connected layers are effectively a linear classifier (or MLP) to predict class from high-level features

Convolutional neural network

"AlexNet": Krizhevsky, Sutskever, & Hinton (2012)



Output: softmax

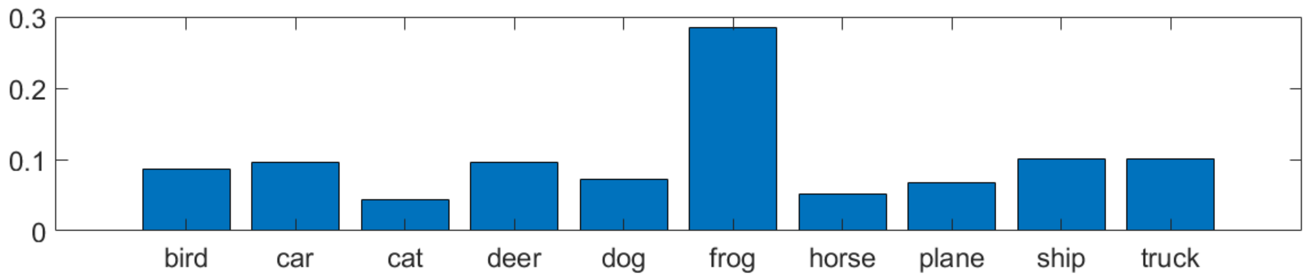
Image: Han, Zhong, Cao, & Zhang (2017)

Loss function: Softmax

- Apply softmax function to last layer's output:

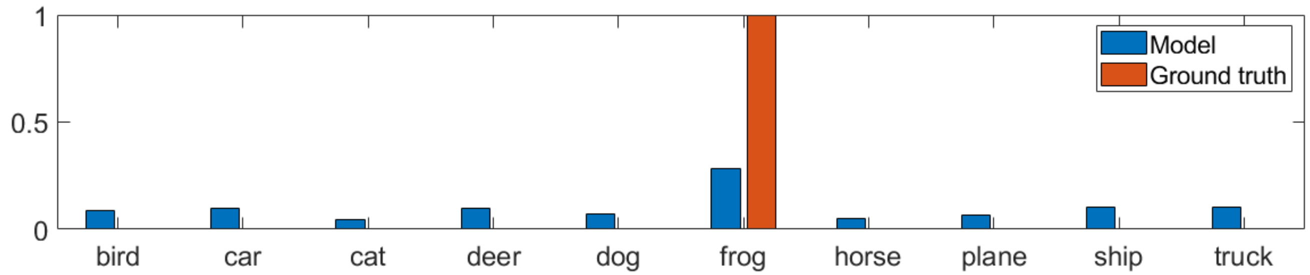
$$\sigma(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{N} e^{y_j}}$$

- Produces a vector that has the properties of a probability distribution:
  - All values in range 0-1
  - Values sum to 1



Loss function: Cross-entropy loss

- Measure of the difference between the model and ground truth probability distributions



- Cross-entropy loss between predicted class and ground-truth class:

Model probability from softmax

$$E = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

N classes

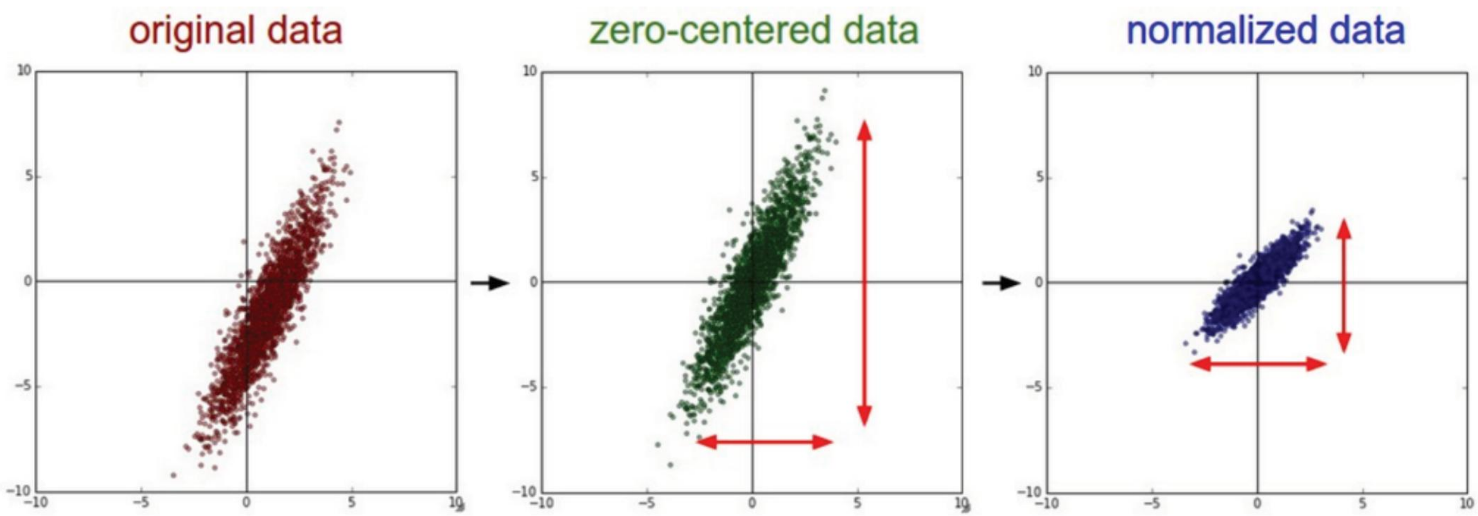
Ground truth probability (1 or 0)

Training process

- Split data into train/validation/test sets
- Split training data into batches
- For N = 1 - ?
  - Preprocess a batch of image data
  - Classify batch, compute loss
  - Update model parameters with backprop
- Periodically check trained model's performance on the validation set (for early stopping)

Data preprocessing

- Image whitening – scale each image to 0-255 range, then normalise so each pixel has mean=0 and (optionally) std=1

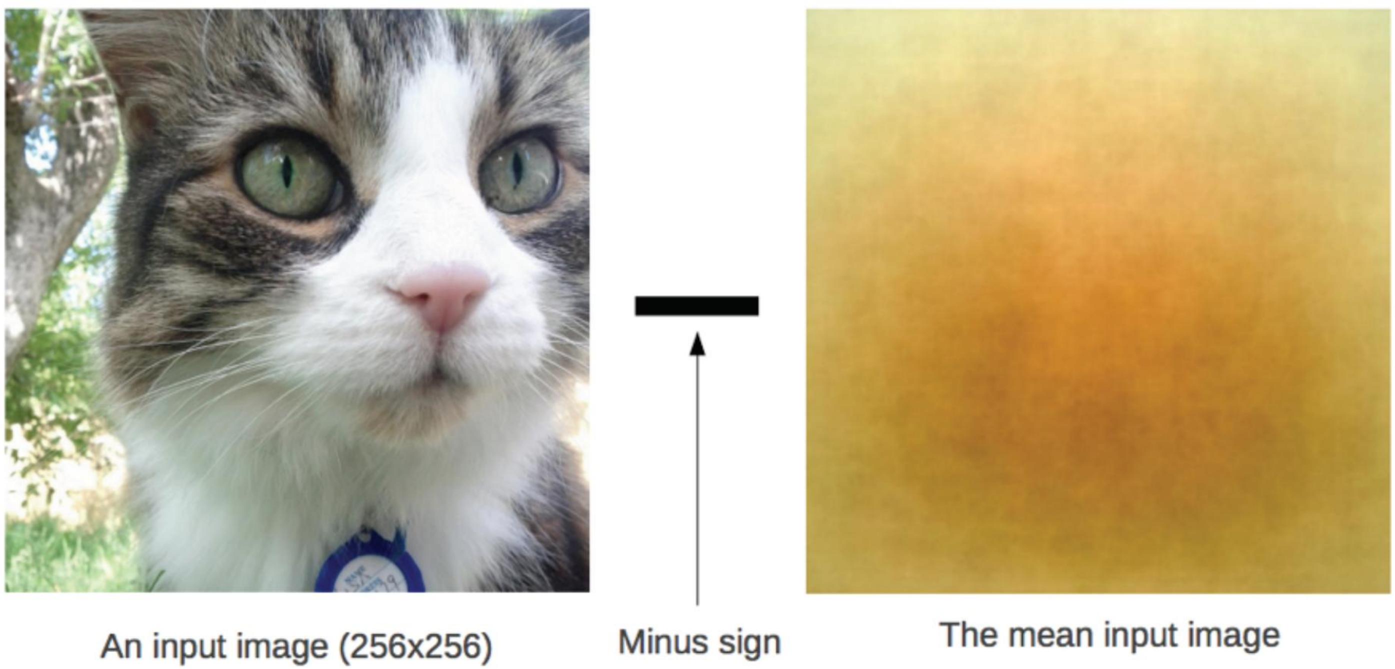



```
X -= np.mean(axis=0, keepdims=True)
```

```
X /= np.std(axis=0, keepdims=True)
```

Figure: Andrej Karpathy

Data preprocessing



A per-channel mean also works (one value per R,G,B).

Figure: Alex Krizhevsky

Data augmentation

- Manipulate training data to generate more samples
- Without data augmentation, even smaller networks (e.g., AlexNet) overfit to ImageNet

Data augmentation

- Common options:
  - Random crops (e.g., 224 x 224 from 256 x 256 images)
  - Horizontal reflection
  - Small colour/contrast adjustments (to simulate different camera settings or times of day)
- Less common:
  - Random rotation (e.g., +/- 15 degrees) ← slow
  - Random scale ← slow
  - Random occluders

Data augmentation

- Why not include other variations?
  - Vertical reflection
  - Large colour changes

Training process

- Initialise network weights and bias
  - Typically, weights initialised to small values from a Gaussian distribution around zero
  - Bias initialised to zero or small positive values
- Set training parameters
  - Batch size
  - Optimiser
  - Learning rate + decay
- Monitor training and validation loss
  - Stop training when validation loss no longer decreases

Batch size

- Batch size (or mini-batch size) = portion of the training data used to compute gradient for parameter update
- It's not computationally feasible to use the whole dataset to compute each update
- Dataset is randomly split into N batches of size b
- N updates = 1 epoch (every image has been seen once)

Batch size

- Smaller batch size
  - More updates (but these are faster to compute)
  - Noisier updates (high variance in gradient)
- Larger batch size
  - Fewer updates (but each update takes longer to compute)
  - More stable updates
- In practice, batch size tends to be limited by memory constraints

Optimiser

- Stochastic Gradient Descent (SGD)
- Root Mean Square Propagation (Rmsprop)
- Adaptive moment estimation (Adam)
  - Keep a moving average of the squared gradient/gradient to divide the learning rate
  - Different from SGD that maintains a single learning rate for different gradients with different magnitudes

Learning rate + decay

- Learning rate = how much to change network parameters on each update
  - Too high rate – unstable training
  - Too low rate – very slow learning

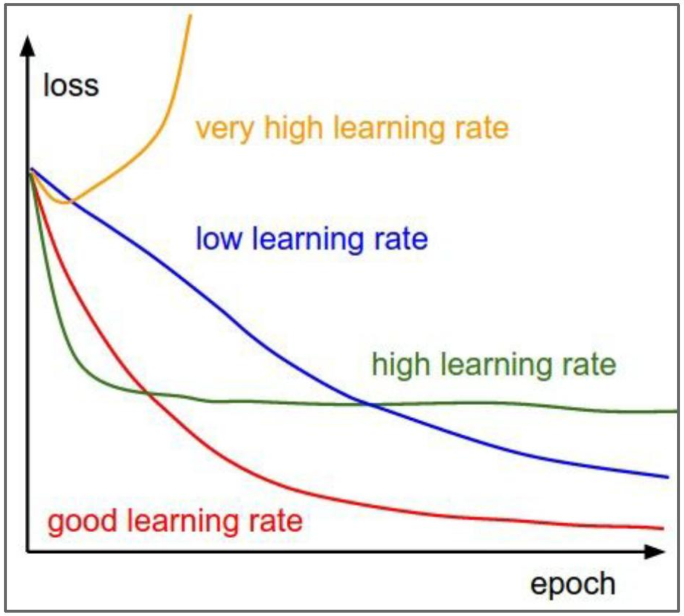


Image: http://cs231n.stanford.edu/

How long to train?

- Generally, train until model's performance on a validation set stops improving

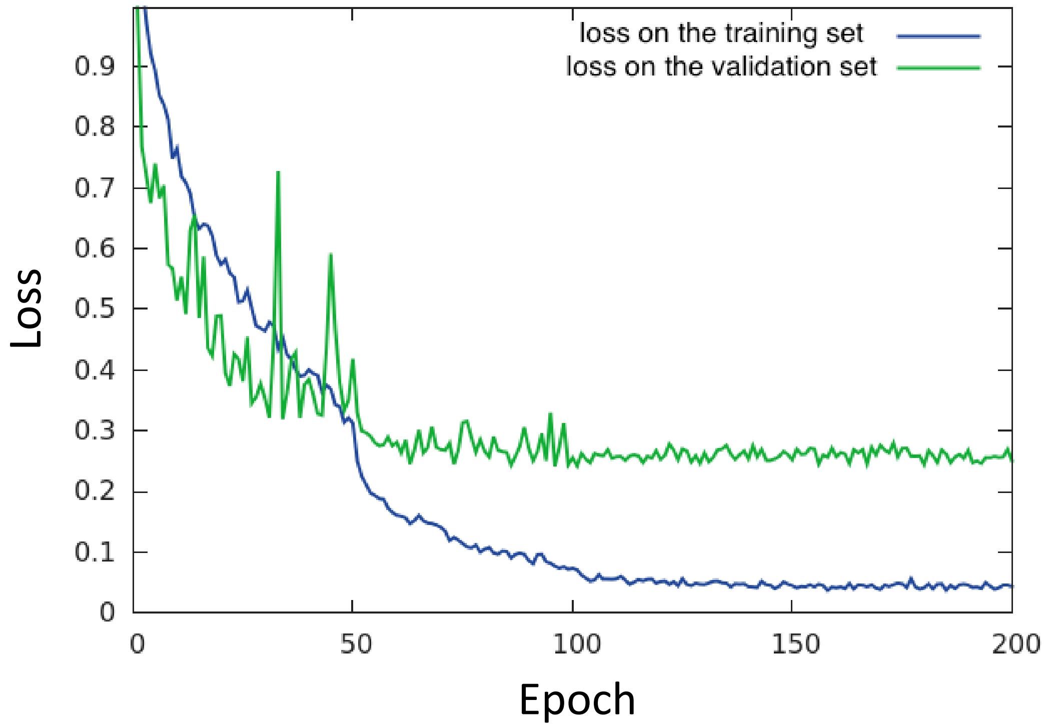


Image: Korbar, et al. (2017)

Summary

- Training CNNs can be difficult – parameter space is extremely large
- Data augmentation is usually required to avoid overfitting
- Hyperparameters (batch size, optimizer, learning rate) can affect how well the network learns