Introduction to Convolutional Neural Networks (ConvNets): ← Information Distillation pipelines

Architecture:

- $\underbrace{\text{Convolutional Neural Networks} \rightarrow \text{MaxPooling2D} \rightarrow \dots}_{\text{Convolutional Base}} \rightarrow \text{Flatten} \rightarrow \underbrace{\text{Dense} \rightarrow \dots \rightarrow \text{Output}}_{\text{Densely connected classifier}}$
- Much better results on image data

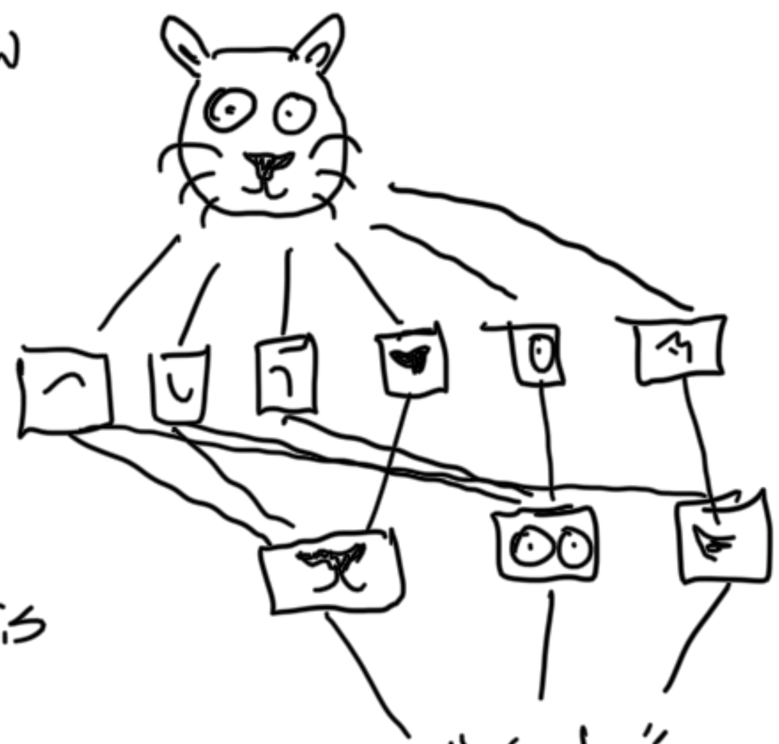
Convolution Operation:

Convolution vs Dense:

- Dense: Learn gobal patterns in input feature space
- Convolution: Learn local patterns (small 2D windows)

Properties:

- Patterns are translation invariant (patterns learned in lower right corner transfer anywhere)
    - Densely connected NNs have to learn the patterns anew
- Learn spatial hierarchies of patterns
    - Visual world is made up of hierarchical patterns



Operation:

Input Feature Map:

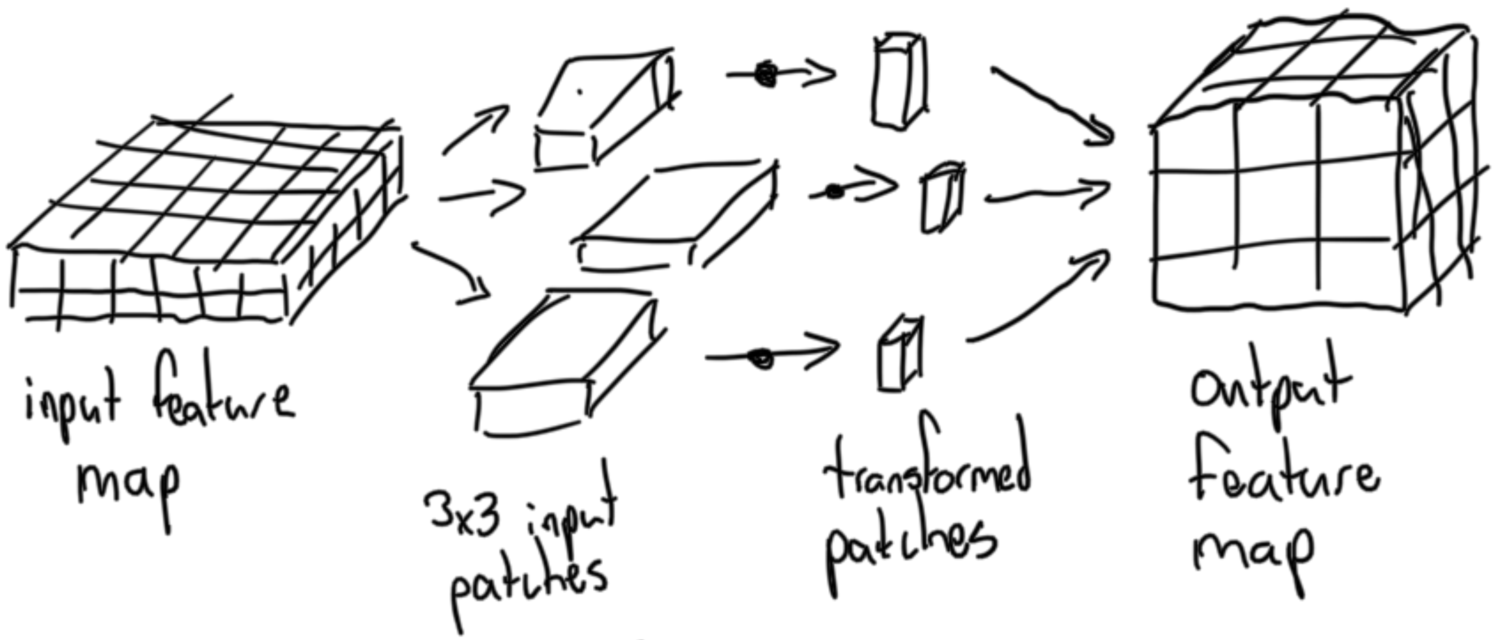
- two spatial axis (height + width) as well as a depth axis

RGB images have a depth axis of 3

Black/White have a depth axis of 1

Output Feature Map:
- two spatial axis (height+width) and now arbitrary "filter" depth
  - ↳ encode specific aspects of input data
  - High Level: Presence of face in input

Keras: modelsummary()
- Shows output shape: (height, width, filters)
  - Each filter is (height, width) large denoting a "response map"
    - ↳ indicates filter's response over input



Note: Height/Width may differ due to:
- Border Effects

- Strides

Border Effects:

- A 3x3 window can only move around a 5x5 shape 9 ways meaning the end result is a 3x3 grid (cutting off outer 2 height/width)

Solution: Use padding

- Add an appropriate # of rows/colums around the feature map so convolutional windows can be centered around every input tile

Keras: In Conv2D layers, use the "padding" argument

↳ Takes two values:

"valid" (no padding on input; use only valid window locs) ← Default

"same" (add padding so output = input)

Definition:

- Size of patches extracted from inputs (usually 3x3 or 5x5) ← Slides over image
- Depth of output feature map (number of filters computed by the convolution)
- Convolutional Kernel: Dot product conversion matrix into 1D vector of shape "output depth"
- Strides: Jump amount for each convolutional window (rarely used but may come in handy)
  ↳ Most people nowadays use MaxPooling2D
- Padding: Valid/same

Max-Pooling: Aggressively downsample feature maps (similar to strided convolutions)

- Extracts windows from input feature maps and outputs max value from each filter/channel

Conceptually similar to convolutions
- Output max value of each channel
- 2x2 window w/ stride=2

Why?
- Features from convolutional layers must be built from more and more of the original image
- Flattenning a final feature map of an unshrunken size would result in a massive number of trainable parameters leading to intense overfitting

So, Max-Pooling reduces the number of feature-map coefficients to process and induces spatial-filtering hierarchies by making succesive convolutional layers look at larger and larger windows

Variations:
- Average Pooling: Take average value of each channel over the patch (usually less effective than max because features features tend to encode the spatial presence of some pattern (concept over tiles)

Training a ConvNet from Scratch on a Small Dataset:

Process:

1) Natively train a new model without any regularization
2) Data Augmentation: mitigate overfitting
3) Feature Extraction w/ pretrained network
4) Fine-Tuning a pretrained network

Note: DL models are highly repurposable; many pre-trained models are available and can be used to bootstrap powerful vision models out of very little data

Building the ConvNet:

- Depth progressively increases
- Feature map size decreases

Binary-Classification Problem:

- Final layer Dense with 1 unit and sigmoid activation
- RMSProp optimizer
- loss: Binary crossentropy

Using Data Augmentation:

- Generate new data from existing training samples
    - Nature of Overfitting: Caused by having too few samples to learn from
      With infinite data, model would be exposed to every possible aspect of data distribution at hand
      ↳ Result: Never overfit

- Augment samples via random transformations yielding believable samples

Keras: Image Data Generator instance

- rotation_range: random rotations in range
- width_shift/height_shift: randomly translate height/width

- width_shift/height_shift: randomly translate height/width
- shear_range: shear transformations
- zoom_range: randomly zoom
- horizontal_flip: randomly flip half the images
- fill_mode: filling strategy on new pixels

- Result is highly intercorrelated so data may not entirely prevent overfitting

Dropout:

- To further prevent overfitting, add a dropout layer before the densely connected layer (after flatten)

Using a Pretrained Convnet: (common+effective)

- With enough training, a model can act as a generic model of the visual world
  - Possibly effective even if classes are completely different

Process:

- Feature Extraction: Use already learned features for new samples
  - Use $\underbrace{\text{convolutional base}}_{\text{much more generic patterns than the classifier}}$ and create new classifier
  - Densely connected networks are useless when object location matters

Keras: keras.applications contains many pretrained networks (this book uses VGG16)

VGG16:

weights: weight checkpoint to initialize model with (ie. ImageNet)

include_top: use densely connected top section

input_shape: shape of input tensors to be fed into the network

With the convolutional base, there are two ways to proceed:

1) Run convolutional base over personal dataset → record output → use as input in dense model
2) Extend convolutional base with dense layers and run everything end-to-end (GPU required)
    - Allows for data augmentation
    - Freeze: Prevent a layer from changing its weights
        Keras: trainable=False

    Result: ~95% accuracy

- Fine-Tuning
    - Unfreeze top layers of a frozen model base used for feature extraction
    - Train newly created dense layers as well as the top of the convolutional base

    Process:
    - Add densely connected top
    - Freeze convolutional base
    - Train densely connected top
    - Unfreeze top layers of convolutional base
    - Retrain cnn base and densely connected top

    Why not retrain all layers of cnn base?
    - More parameters = higher risk of overfitting
    - Early layers in the CNN are generic

    - Use a very low learning rate

    Result: ~97% accuracy

Visualizing What Convolutional Neural Networks Learn:

- ConvNets are NOT black boxes
  - Representation of visual concepts

Techniques:

- Visualize intermediate convnet outputs (intermediate activations) - Useful for understanding how successive convnet layers transform input ← involves multiple outputs (Keras: models.Model)
- Visualize convnet features: Understand patterns/concepts filters in a convnet are receptive to
- Visualize heatmaps of class activation in an image: Understand which parts of an image are identified

Intermediate Activation Visualization:

- First layer acts as a collection of edge detectors
  - activations retain nearly all information in the original image
- The higher up, activations become increasingly abstract and less interpretable
  - ↑ layer ↑ abstraction ↓ interpretable ↓ info about image contents ↑ info about class ↑ sparsity
    - ↳ not all patterns contained in input image as layer depth increases

Visualizing ConvNet Filters:

- Gradient ascent in input space to determine inputs that maximize response of specific filters
  - Build loss function that maximizes value of target filter
  - Regularize loss tensor by dividing it with its L2-norm
    - Ensures magnitude of change is always the same

Visualizing Heatmaps of Class Activation: (Class Activation Map (CAP))

- Helpful for debugging decision process of a convnet

- Helpful for debugging decision process of a convnet
- Also allows for locating specific objects in a heatmap

Implementation: Grad-CAM: Visualization Explanations from Deep Networks via Gradient-based Localization"
  - Take output feature map given an image
  - Weight every channel by the gradient of the class with respect to the channel
    - Intuition: Intensity of input image activating different channels

      +

      how important each channel is with regard to the class

      =

      how intensely input image activates class
  - Superimpose activation heatmap on original image