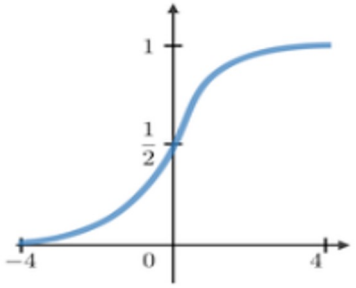
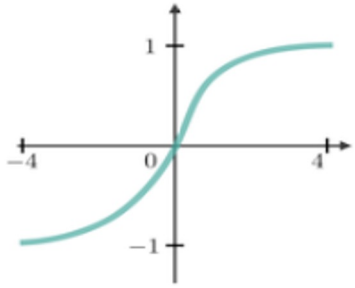
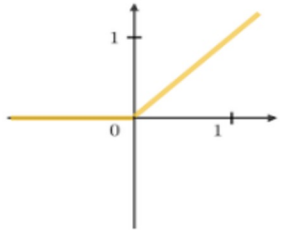
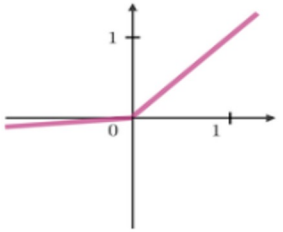
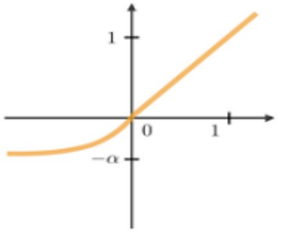
Overview of Deep Learning

Backpropagating through many layers

Activation Functions

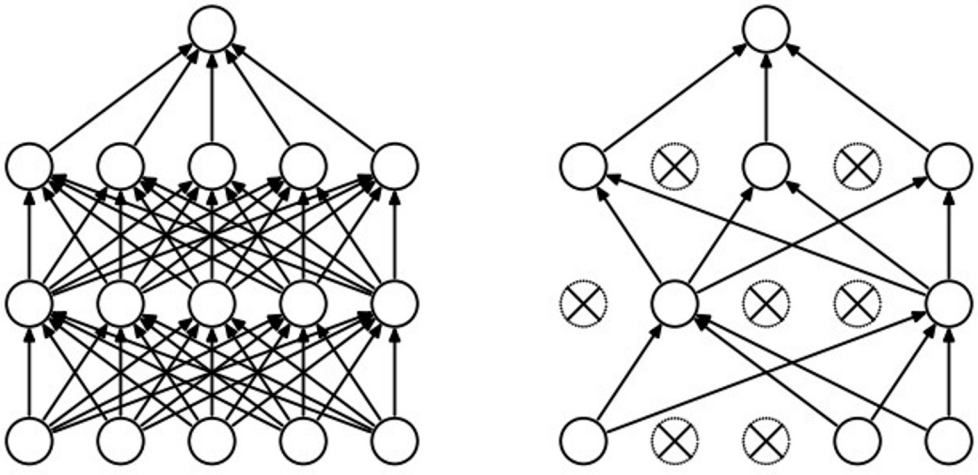
Sigmoid	Tanh
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$
	

- Vanishing gradient
- Computationally expensive

ReLU	Leaky ReLU	ELU
$g(z) = \max(0,z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$	$g(z) = \max(\alpha(e^{z}-1), z)$ with $\alpha \ll 1$
		
Non-linearity complexities biologically interpretable	Addresses dying ReLU issue for negative values	Differentiable everywhere

- ReLU = Rectified Linear Unit
- Computationally efficient
- Dying ReLU problem
  - Solved using Leaky ReLU/Parametric ReLU

Preventing over-fitting: Dropout (Lecture, Code)



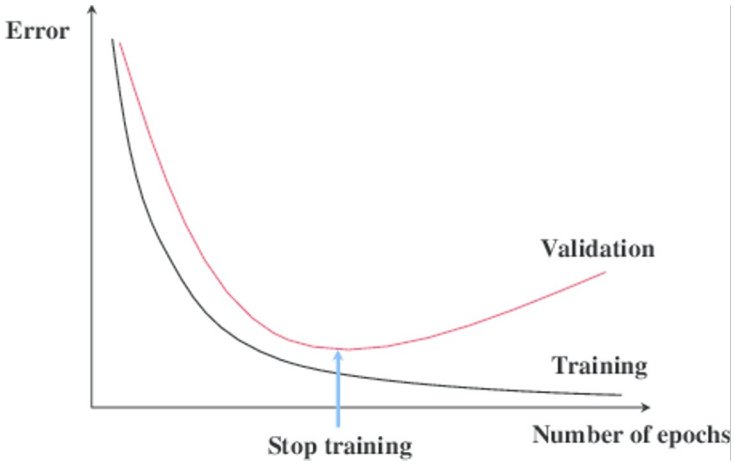
(a) Standard Neural Net (b) After applying dropout.

Dropout (input or hidden) neurons selected at random;
(output set to 0) with probability *p* in each minibatch
%dropout is a hyper-parameter

Source: https://pgaleone.eu/deep-learning/regularization/2017/01/10/anaysis-of-dropout/

Preventing over-fitting: Early Stopping

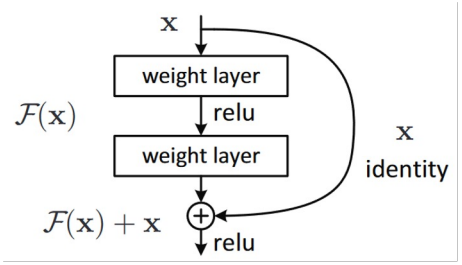
- Let us measure the performance of our model on a separate validation dataset during the training iterations.
- We may then observe that, despite constant score improvements on the training data, the model's performance on the validation dataset would only improve during the first stage of training, reach an optimum at some point and then turn to getting worse with further iterations.



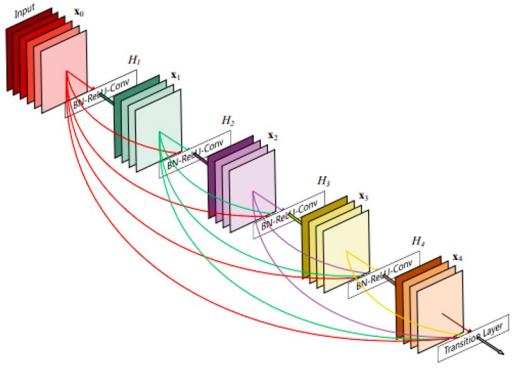
Source: http://fouryears.eu/2017/12/06/the-mystery-of-early-stopping/comment-page-1/

Skip Connections

- In long chain of multiplications (as in case of early layers), when many things less than one are multiplied together, the resulting gradient becomes very small.
- Skip some layer in the neural network and feeds the output of one layer as the input to the next layers (instead of only the next one).
- Provide an alternative path for the gradient (with backpropagation).
- Experimentally validated that this additional paths are often beneficial for model convergence.
- Resnet (top) uses skip connections via addition while Densenet (bottom) uses it via concatenation.



$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial H}\frac{\partial H}{\partial x} = \frac{\partial L}{\partial H}\left(\frac{\partial F}{\partial x} + 1\right) = \frac{\partial L}{\partial H}\frac{\partial F}{\partial x} + \frac{\partial L}{\partial H}$$



Source: https://theaisummer.com/skip-connections/

Adaptive Learning Factor

- RMSprop. Make update inversely proportional to the sum of past gradients, so, update more when gradient is small and less where it is large.

$$\Delta w_i^t = -\frac{\eta}{\sqrt{r_i^t}} \frac{\partial E^t}{\partial w_i}$$

where r_i is the accumulated past gradient,

$$r_i^t = \rho r_i^{t-1} + (1-\rho) \left| \frac{\partial E^t}{\partial w_i} \right|^2$$

ADAM: Adaptive Learning Factor w/ Momentum

$$s_i^t = \alpha s_i^{t-1} + (1-\alpha)\frac{\partial E^t}{\partial w_i}$$

$$r_i^t = \rho r_i^{t-1} + (1-\rho)\left|\frac{\partial E^t}{\partial w_i}\right|^2$$

$$\Delta w_i^t = -\eta \frac{\tilde{s}_i^t}{\sqrt{\tilde{r}_i^t}}$$

Initially both s and r terms are 0, so we bias-correct them (both a and r are <1, so they get smaller as t gets large):

$$\tilde{s}_i^t = \frac{s_i^t}{1-\alpha^t} \text{ and } \tilde{r}_i^t = \frac{r_i^t}{1-\rho^t}$$

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

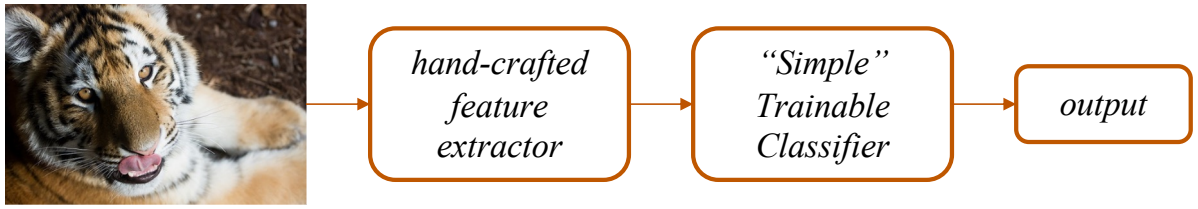
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

usually done after a fully connected/convolutional layer and before a non-linearity layer and aims at allowing higher learning rates and reducing the strong dependence on initialization.

Source: https://mlexplained.com/2018/01/10/an-intuitive-explanation-of-why-batch-normalization-really-works-normalization-in-deep-learning-part-1/

CNN Introduction

Traditional pattern recognition models use hand-crafted features and relatively simple trainable classifier.



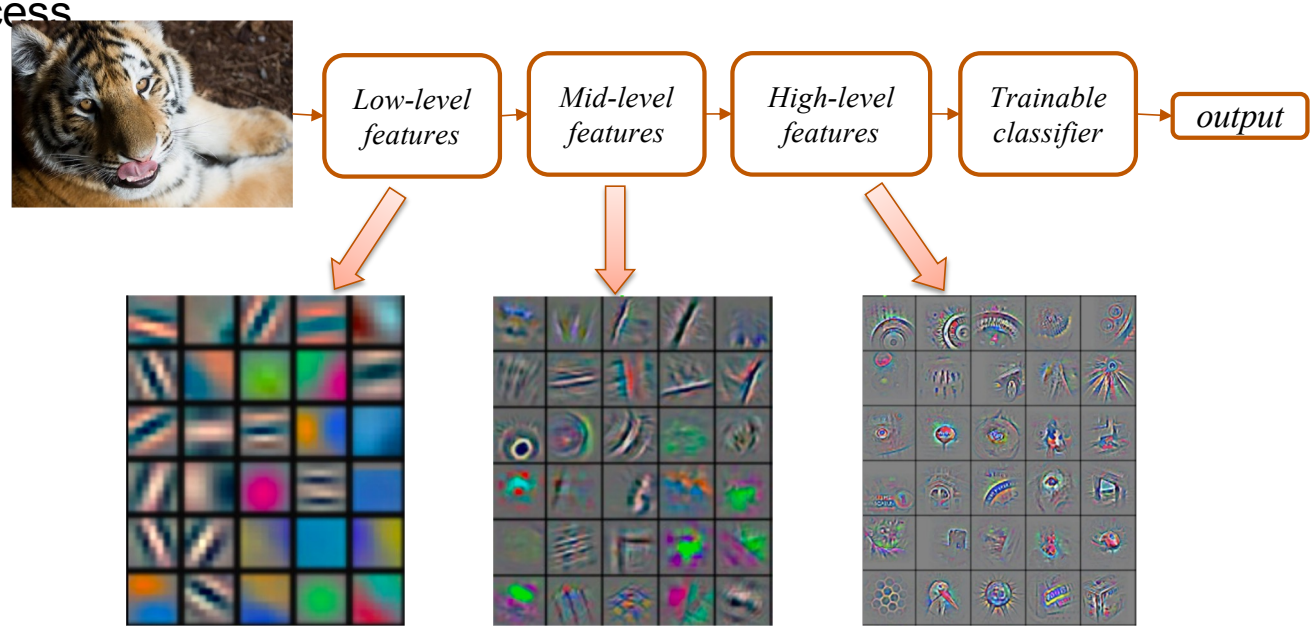
However:

Hand-crafted features are usually highly dependent on the application

Hand-crafted features need human input and perhaps several trials, so may take time

Deep Learning

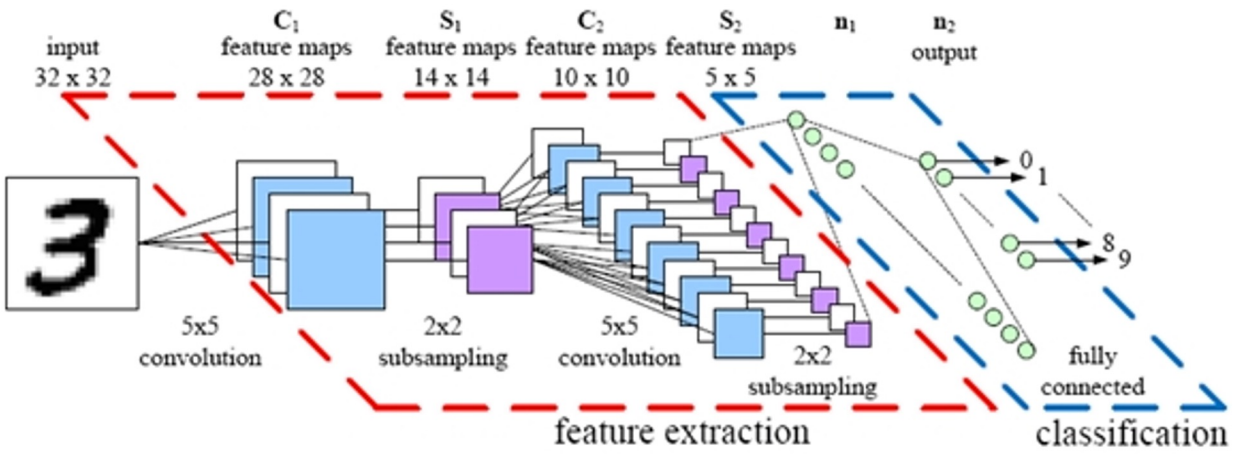
Deep learning (a.k.a. representation learning) seeks to learn rich hierarchical representations (i.e. features) automatically through multiple stage of feature learning process



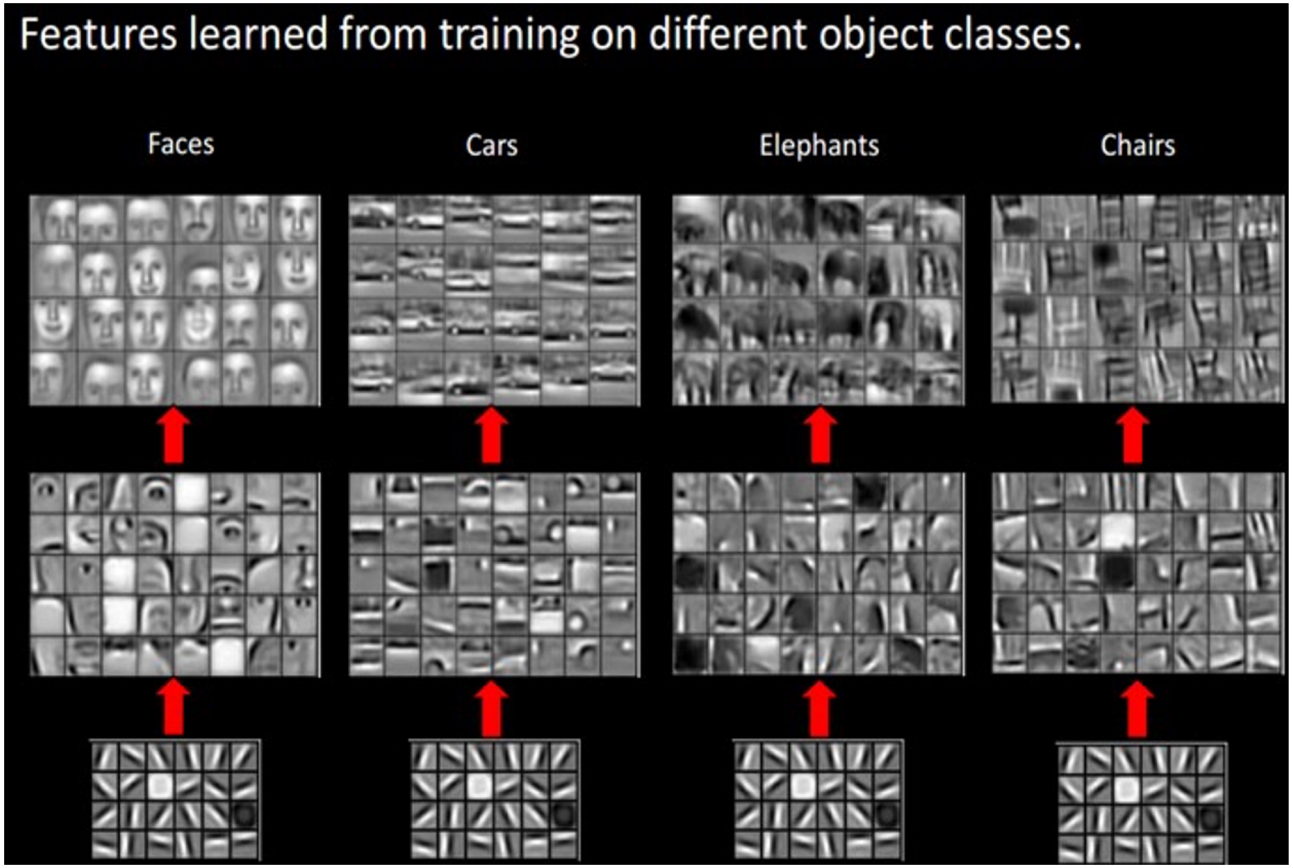
Feature visualization of convolutional net trained on ImageNet (Zeiler and Fergus, 2013)

Convolutional Neural Nets (CNNs)

- How do they work? (excellent blog by Brandon Rohrer)
- How we teach computers to understand pictures | Fei Fei Li

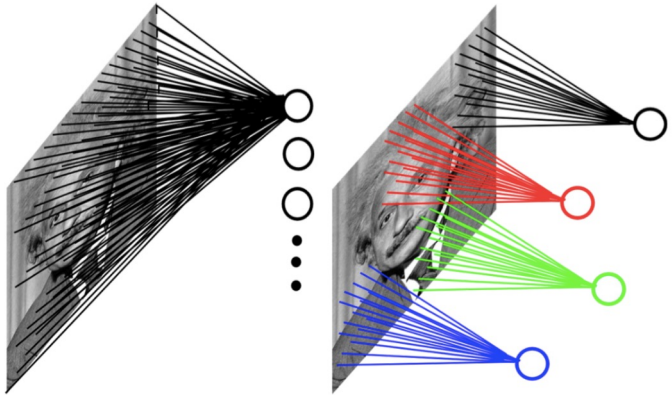


From: http://parse.ele.tue.nl/cluster/2/CNNArchitecture.jpg



CNNs

- The hidden units in a convolution layer are only connected to a local receptive field.
- Their weights may be shared.
- The number of parameters needed by CNNs is much smaller.



Example: 200x200 image

a) fully connected: 40,000 hidden units => 1.6 billion parameters

b) CNN: 5x5 kernel, 100 feature maps => 2,500 parameters

Convolution Lecture, Code

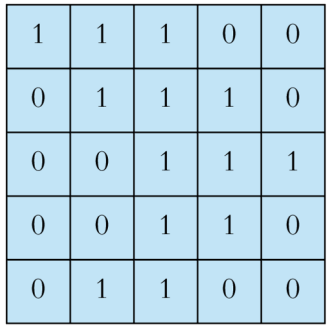
- Useful with features are ordered
  - by time (1-d)
  - By space e.g. images (2-d)
  - Logical space (>2-d)

- Trying to find a localized match with a given template/filter/kernel
- Operation:
  - Center at a given "location"
  - Point-wise multiply and add = response at that location
  - Shift to next location (determined by "stride length").
  - Equation (Details):

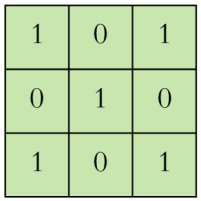
$$f(x) * g(x) = \int_{-\infty}^{\infty} f(\tau) g(x - \tau)\, d\tau$$

  - Practical: Zero-Padding

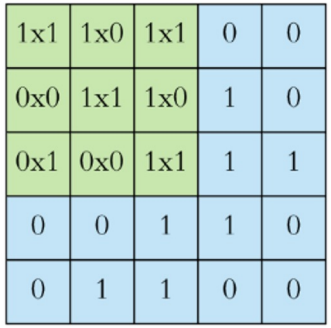
How Convolutions Work (Details)



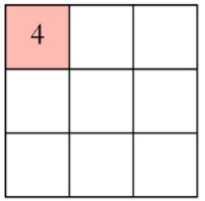
Input



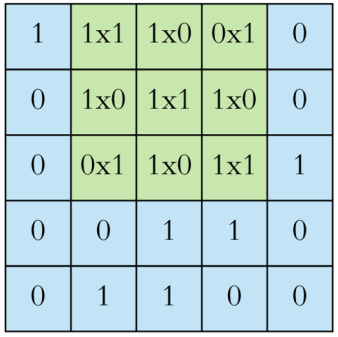
Filter / Kernel



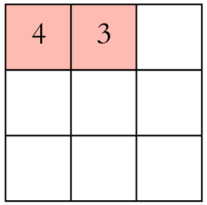
Input x Filter



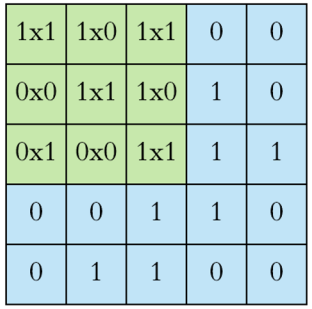
Feature Map

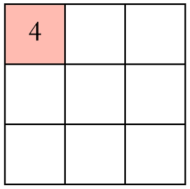


Input x Filter



Feature Map



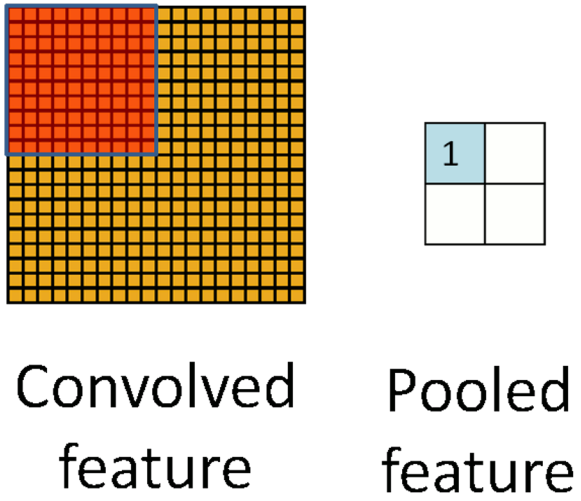


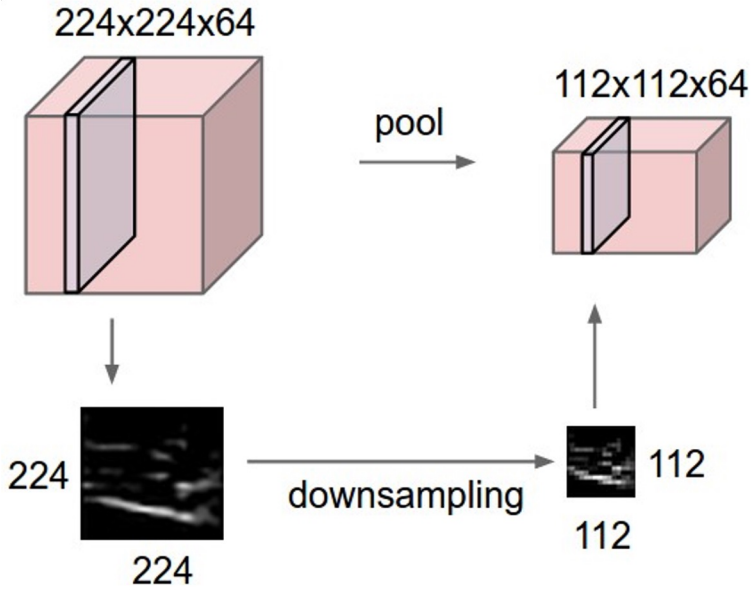
Source: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

Pooling Lecture, Code

Helps in reducing the number of features by aggregating them using a single operation since learning a classifier with large number of features may be unwieldly and can also be prone to over-fitting.

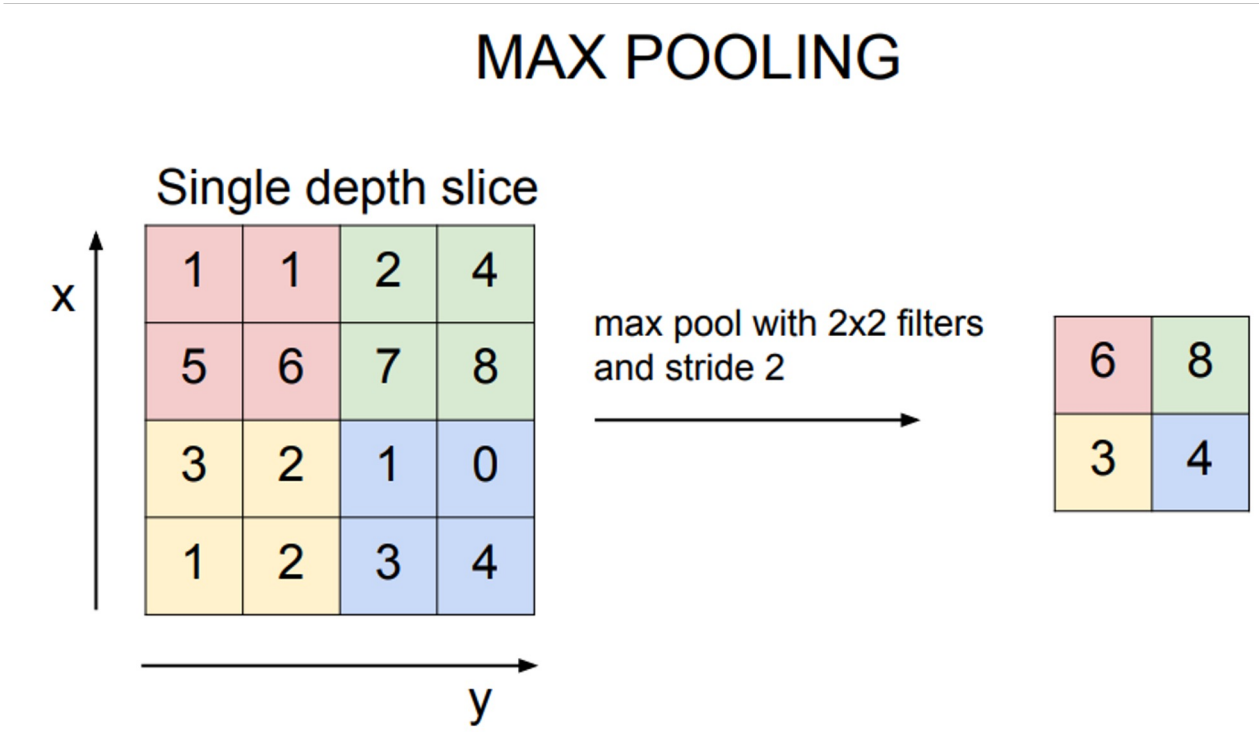
E.g.: Max pooling, mean pooling



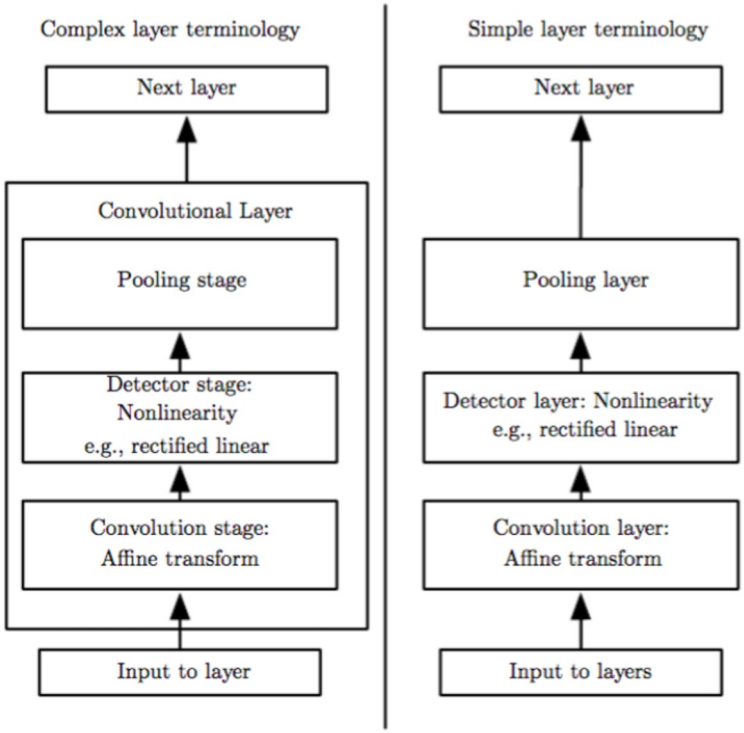


Source: http://deeplearning.stanford.edu/tutorial/supervised/Pooling/ and https://gist.github.com/fabsta/cb0f216982a4ed01ea60a060955d95c5

Pooling (and sub-sampling)

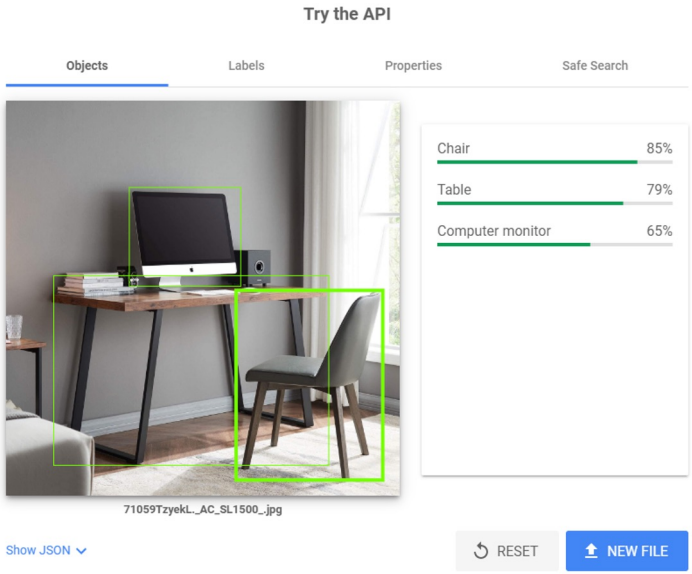


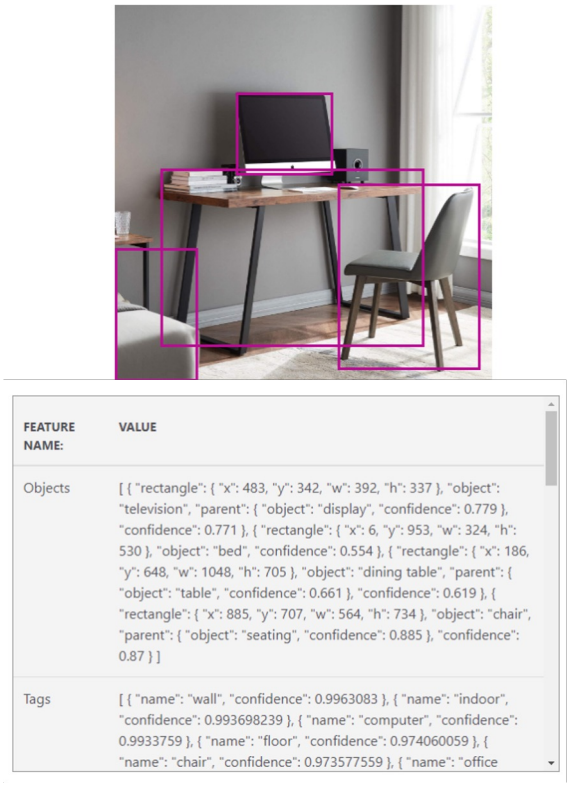
Three Stages of a Convolutional Layer



1. Convolution operation
2. Nonlinearity: a nonlinear transform such as rectified linear or tanh
3. Pooling: output a summary statistics of local input, such as max pooling and average pooling

Deep CNN for OBJECT DETECTION





Try out with your own images using
Left: GCP, Right: Azure

Data Augmentation

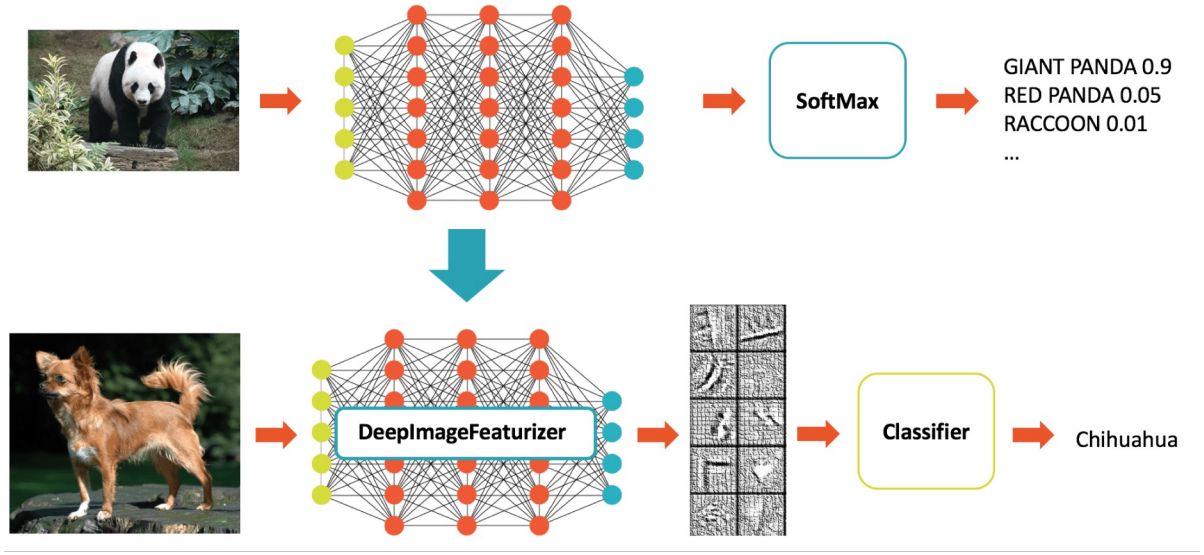
- to get more data from the existing ones;
  - Also for better representation learning

Original	Flip	Rotation	Random crop
			
- Image without any modification	- Flipped with respect to an axis for which the meaning of the image is preserved	- Rotation with a slight angle - Simulates incorrect horizon calibration	- Random focus on one part of the image - Several random crops can be done in a row

Color shift	Noise addition	Information loss	Contrast change
			
- Nuances of RGB is slightly changed - Captures noise that can occur with light exposure	- Addition of noise - More tolerance to quality variation of inputs	- Parts of image ignored - Mimics potential loss of parts of image	- Luminosity changes - Controls difference in exposition due to time of day

Transfer Learning with CNNs

- From https://databricks.com/blog/2017/06/06/databricks-vision-simplify-large-scale-deep-learning.html
- Examples: Cancer detection: https://www.nature.com/nature/journal/v542/n7639/full/nature21056.html
- Metamind (now powering Salesforce CRM)



Issues to think about

- Architecture design
  - number and type of layers (conv, pool)
  - kernel sizes for (conv, pool)
  - activation function (relu, sigmoid, softmax, softplus)
  - number of feature maps (conv) or neurons (fully connected)
  - loss function choice (cross entropy, regression)
  - Advanced architecture choices (BatchNorm, Local Response Normalization (AlexNet paper), Residual (ResNet), Skip)
- Proper initialization
  - random, xavier (scaling)
  - unsupervised pre-training
- Learning
  - SGD, SGD+Momentum, AdaGrad, Adam, RMSProp
  - weight decay, learning rate, momentum
  - Dropout, gradient clipping
- Hardware
  - mapping onto GPUs
  - heterogeneous distributed (synchronous, asynchronous)

Hyperparameter Optimization

- Also called tuning.
- Differences between hyperparameters and model parameters:
  - Model parameters are learnt during training, e.g. Weights and Biases.
  - Model hyperparameters cannot be learnt during training of the model, e.g. Learning rate, batch size, but effect the estimated model parameters
- How to choose hyperparameters automatically based on a given metric (e.g. Accuracy)?

Source: http://www.cs.cornell.edu/courses/cs6787/2017fa/Lecture6.pdf

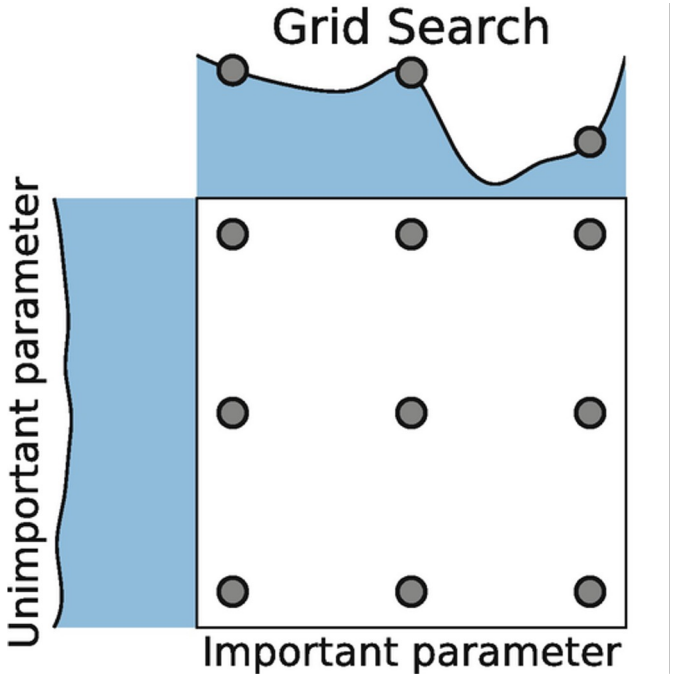
Simple Strategies

- Use well-known hyperparameters based on folklore.
  - Can be wrong in a lot of cases.
  - Can also be a good way to start a research project.

- Tuning by hand.
  - Change the parameters manually until you get a good result.
  - The most common type of hyperparameter optimization.
  - Lots of effort required.

Grid Search (brute force)

- Define a grid of parameters.
- Run the system using every possible combination of parameters.
- Choose the setting with the best results.
- Can be parallelized.
- As the number of parameters increase, the cost increases exponentially.
- Need some way to choose the grid initially, which is like choosing a hyperparameter.
- Cannot exploit the system's insights.



Source: http://www.cs.cornell.edu/courses/cs6787/2017fa/Lecture6.pdf

Best Ball (A variant)

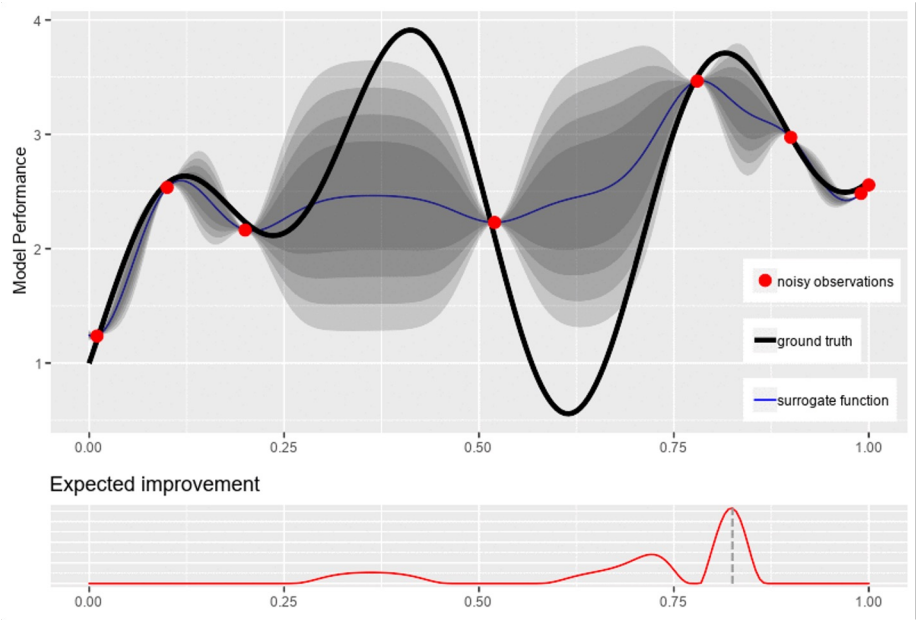
- Works with epochs.
- At each epoch, do a small grid search around the current hyperparameter settings.
- Now evaluate the objective and choose the best ball (the settings that give the best objective for that epoch).
- Repeat until a good enough solution is obtained.

Source: http://www.cs.cornell.edu/courses/cs6787/2017fa/Lecture6.pdf

Bayesian Optimization (e.g. using Gaussian Processes Details)

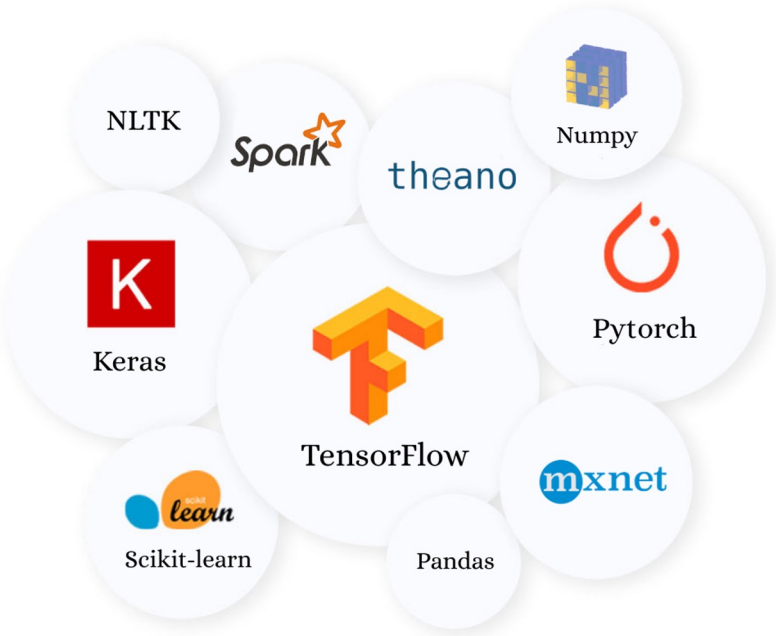
SEQUENTIAL BAYESIAN ESTIMATION

- Learns approx. probability distribution of the loss function given hyperparameter values.
- Use a simple surrogate function that is quick to compute
- Use surrogate + occasional calls to actual model to update probabilities and zero in on a good hyperparameter region.



Source: http://www.cs.cornell.edu/courses/cs6787/2017fa/Lecture6.pdf

Deep Learning Ecosystem

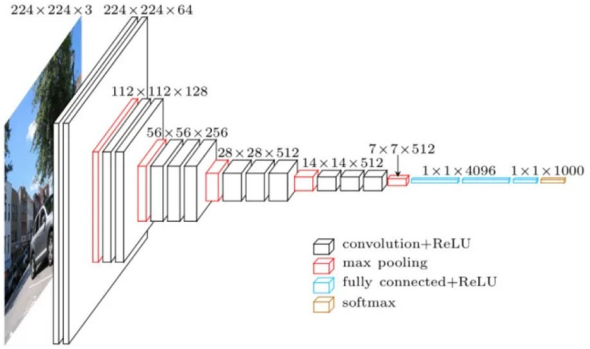


Keras: High Level Library for Tensorflow (and others)

Keras Models

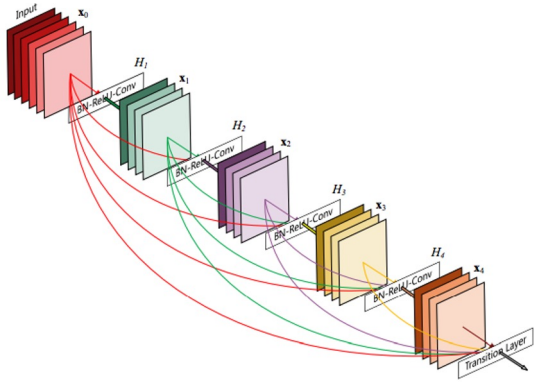
Sequential Model

- Linear stack of layers
- Each layer feeds into the next layer
- e.g. simple classification network



Functional Model

- Layers connect to more than just the previous and next layers.
- Can create complex neural networks
- e.g. residual network



Sequential Model

Do the required imports

```
import tensorflow as tf
from tensorflow.keras import Sequential,layers
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.utils import plot_model
```

Defining a simple model

```
model = Sequential()
model.add(Input(shape=(10,5)))
model.add(Dense(100, activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 10, 100)           600
_________________________________________________________________
dense_13 (Dense)             (None, 10, 20)            2020
_________________________________________________________________
dense_14 (Dense)             (None, 10, 1)             21
=================================================================
Total params: 2,641
Trainable params: 2,641
Non-trainable params: 0
_________________________________________________________________
```

How params are computed?

No. of params = (i * h) + b
i=input, h=hidden, b=bias

Input = 10 * 5 = 50

Layer 1 = (5 * **100**) + 100 = 600

Layer 2 = (100 * **20**) + 20 = 2020

Layer 3 = (20 x **1**) + 1 = 21

The model has 10x5=50 inputs, 2 hidden layers with 100 and 20 neurons, and an output layer with 1 output.

Sequential Model

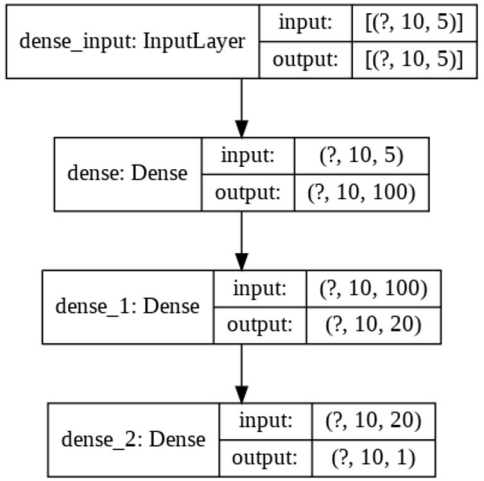
Model Compile

- Transform a sequence of layers into series of matrix transforms to be executed.
- Pre-compute step for the network
- Required after defining a model
- Specify the parameters to be used
  - Optimizer
  - Loss function
  - Learning rate

```
model.compile(optimizer=Adam(learning_rate=0.001),
              loss=BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Model plot

```
plot_model(model, show_shapes=True, show_layer_names=True, to_file='model.png')
from IPython.display import Image
Image(retina=True, filename='model.png')
```



Functional Model

Do the required imports

```
] import tensorflow as tf
  from tensorflow.keras import Sequential,layers
  from tensorflow.keras.layers import Input, Dense
  from tensorflow.keras.optimizers import Adam
  from tensorflow.keras.losses import CategoricalCrossentropy
  from tensorflow.keras.models import Model
  from tensorflow.keras.utils import plot_model
```

Defining a simple model

```
input1 = Input(shape=(16,))
x1 = Dense(8, activation='relu')(input1)
input2 = Input(shape=(32,))
x2 = Dense(8, activation='relu')(input2)
added = concatenate([x1, x2])
out = Dense(4)(added)
model = Model(inputs=[input1, input2], outputs=out)

model.summary()
```

```
Model: "model_3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_18 (InputLayer)           [(None, 16)]         0
__________________________________________________________________________________________________
input_19 (InputLayer)           [(None, 32)]         0
__________________________________________________________________________________________________
dense_50 (Dense)                (None, 8)            136         input_18[0][0]
__________________________________________________________________________________________________
dense_51 (Dense)                (None, 8)            264         input_19[0][0]
__________________________________________________________________________________________________
concatenate_3 (Concatenate)     (None, 16)           0           dense_50[0][0]
                                                                 dense_51[0][0]
__________________________________________________________________________________________________
dense_52 (Dense)                (None, 4)            68          concatenate_3[0][0]
==================================================================================================
Total params: 468
Trainable params: 468
Non-trainable params: 0
__________________________________________________________________________________________________
```

How params are computed?

No. of params = (i * h) + b
i=input, h=hidden, b=bias

input1 = 16, input2 = 32

x1 = (16 * 8) + 8 = 136

x2 = (32 * 8) + 8 = 264

concatenate = 8 + 8 = 16

out = (16 * 4) + 4 = 68

Functional Model

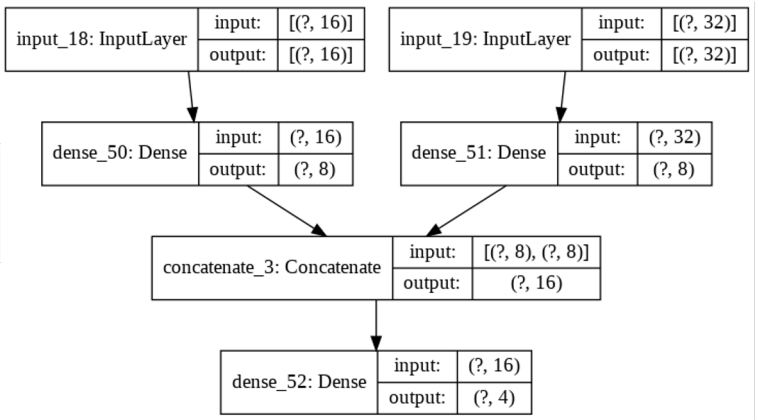
Model Compile

- Transform a sequence of layers into series of matrix transforms to be executed.
- Pre-compute step for the network
- Required after defining a model
- Specify the parameters to be used
  - Optimizer
  - Loss function
  - Learning rate

```
model.compile(optimizer=Adam(learning_rate=0.001),
              loss=CategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Model plot

```
plot_model(model, show_shapes=True, show_layer_names=True, to_file='model.png')
from IPython.display import Image
Image(retina=True, filename='model.png')
```



Code

Run on Google Colab

MNIST demo

Resources

- https://www.deeplearningbook.org/
- https://d2l.ai/
- https://github.com/fastai/fastbook
- Visual guide to Neural Nets: https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi