Deep Learning. CNNs. ML Libraries

Using TensorFlow, Keras and H2O

About Me

Trayan Iliev

– CEO of IPT – Intellectual Products & Technologies

– Oracle® certified programmer 15+ Y

– End-to-end reactive fullstack apps with Go, Python, Java, ES6/7, TypeScript, Angular, React and Vue.js

– 12+ years IT trainer

– Voxxed Days, jPrime, Java2Days, jProfessionals, BGOUG, BGJUG, DEV.BG speaker

– Lecturer @ Sofia University – courses: Fullstack React, Multimedia with Angular & TypeScript, Spring 5 Reactive, Internet of Things (with SAP), Multiagent Systems and Social Robotics, Practical Robotics & Smart Things

– Robotics / smart-things/ IoT enthusiast, RoboLearn hackathons organizer

Where to Find The Code and Materials?

https://github.com/iproduct/course-ml

Machine Learning + Big Data in Real Time + Cloud Technologies

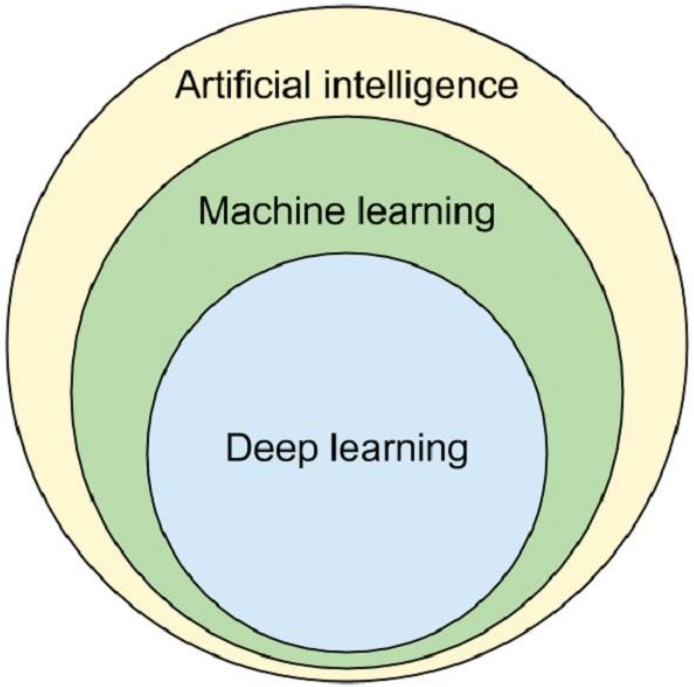
=> The Future of Intelligent Systems

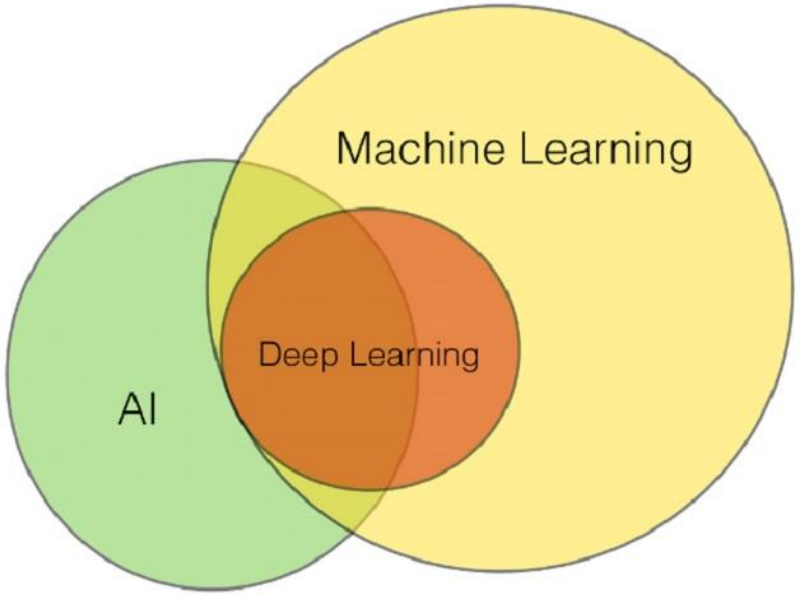
Introduction to Machine Learning

Machine Learning and Artificial Intelligence

- ML learns and predicts based on passive observations, whereas AI implies an agent interacting with the environment to learn and take actions that maximize its chance of successfully achieving its goals.

Judea Pearl, The Book of Why, 2018





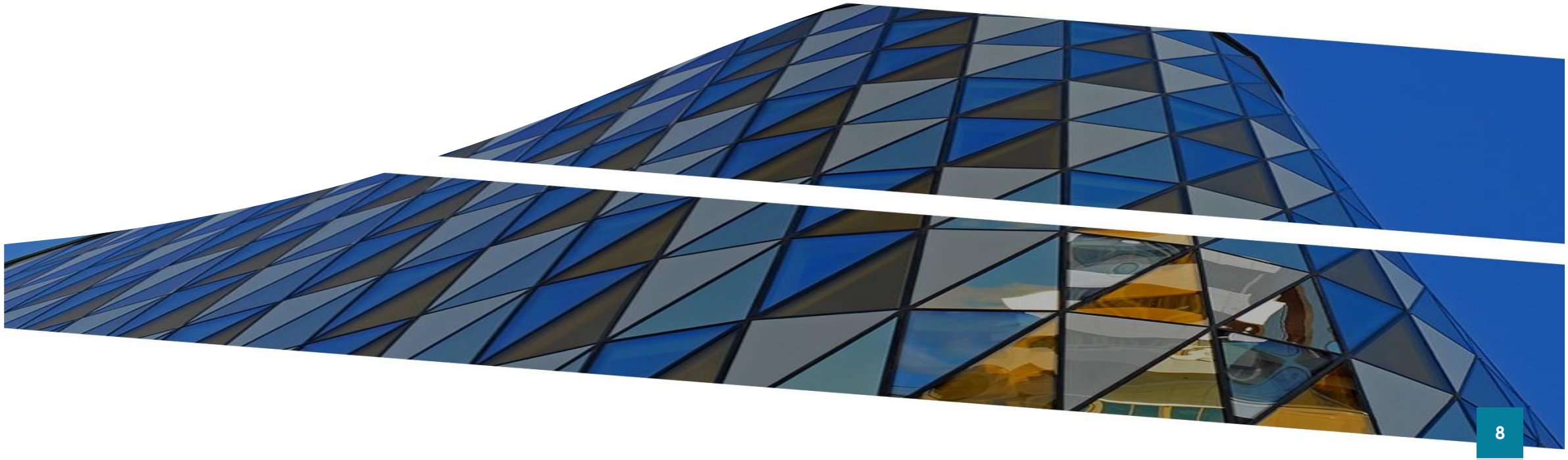
Pictures by: Yakoove - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=95589683
By Yakoove - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=95589710

MACHINE LEARNING

Supervised Learning		Unsupervised Learning		Deep Learning (semi-supervised)		
Regressors	Classifiers	Dimension Reducers	Clustering Methods	Unsupervised Pretrained Networks	Convolutional Neural Networks	Recurrent Neural Networks
Wireline log prediction Spatial interpolation Sesimic inversion (with numeric labels)	Automatic facies prediction from wireline logs Seismic inversion (with categ-orical labels)	Compressing high-dimensional data Increasing signal-to-noise ratio in data	AVO class prediction Seismic inversion (uncalibrated)	Seismic denoising Seismic multiple removal Seismic migration	Seismic structural feature/fault detection Automatic facies prediction from borehole imagery	Microseismic analysis Sedimentary process modelling Earthquake prediction

Picture by: https://wiki.seg.org/wiki/Machine_learning

Machine Learning Libraries



Top Machine Learning Libraries - I

- Apache Spark Mllib – fast, in-memory, big-data processing, which has also made it a go-to framework for ML, highly scalable, can be coupled with any Hadoop data source, supports Java, Scala, R and Python, new ML algorithms are constantly being added, and existing enhanced.

- H_20 + AutoML - open source, in-memory, distributed, fast, and scalable machine learning and predictive analytics platform allowing to learn from big data, written in Java, uses Distributed Key/Value store for accessing data, models, objects, etc. across all nodes and machines, uses distributed Map/Reduce, utilizes Java Fork/Join multi-threading, data read in parallel, distributed across the cluster, and stored in memory in a columnar format in a compressed way, data parser has built-in intelligence to guess the schema of the incoming dataset and supports data ingest from multiple sources in various formats, APIs: REST, Flow UI, H2O-R, H2O-Python, supports Deep Learning, Tree Ensembles, and Generalized Low Rank Models (GLRM).

Top Machine Learning Libraries - II

- Microsoft Azure ML Studio - Microsoft's ML framework runs in their Azure cloud, offering high-capacity data processing on a pay-as-you-go model. Its interactive, visual workspace can be used to create, test and iterate ML "experiments" using the built-in ML packages; they can then be published and shared as web services. Python or R custom coding is also supported. A free trial is available for new users.

- Amazon Machine Learning, Sagemaker - like Azure, the Amazon Machine Learning framework is cloud-based, supporting Amazon S3, Redshift and RDS. It combines ML algorithms with interactive tools and data visualizations which allow users to easily create, evaluate and deploy ML models. These models can be binary, categoric or numeric – making them useful in areas such as information filtering and predictive analytics. Includes support for labeling, data preparation, feature engineering, statistical bias detection, auto-ML, training, tuning, hosting, explainability, monitoring, and workflows. Provides integrated development environment (IDE) for ML.

Top Machine Learning Libraries - III

- Microsoft Distributed Machine Learning Toolkit (DMTK) - uses local data caching to make it more scalable and efficient on computing clusters that have limited resources on each node. Its distributed ML algorithms allow for fast training of gradient boosting, topic and word-embedding learning models:
  - LightLDA: Scalable, fast and lightweight system for large-scale topic modeling.
  - LightGBM: LightGBM is a fast, distributed, high performance gradient boosting (GBDT, GBRT, GBM or MART) framework based on decision tree algorithms, used for ranking, classification and many other machine learning tasks.
  - Distributed word embedding: a parallelization of the Word2Vec algorithm
  - Distributed Multi-sense Word Embedding (DMWE) - a parallelization of the Skip-Gram Mixture algorithm used for polysemous words.
- Google TensorFlow - open-source ML toolkit that has been applied in areas ranging from machine translation to biomedical science. Data flows are processed by a series of algorithms described by a graph. This allows for a flexible, multi-node architecture that supports multiple CPUs, GPUs, and TPUs. The recently released Tensorflow Lite adds support for neural processing on mobile phones.

Top Machine Learning Libraries - IV

- Caffe - developed by Berkeley AI Research, Caffe claims to offer one of the fastest implementations of convolutional neural networks. It was originally developed for machine vision projects but has since expanded to other applications, with its extensible C++ code designed to foster active development. Both CPU and GPU processing are supported.

- Veles - developed and released as open source by Samsung, Veles is a distributed ML platform designed for rapid deep-learning application development. Users can train various artificial neural net types – including fully connected, convolutional and recurrent. It can be integrated with any Java application, and its "snapshot" feature improves disaster recovery.

Top Machine Learning Libraries - V

- Massive Online Analysis (MOA) - developed at the University of Waikato in New Zealand, this open-source Java framework specializes in real-time mining of streaming data and large-scale ML. It offers a wide range of ML algorithms, including classification, regression, clustering, outlier detection and concept drift detection. Evaluation tools are also provided.

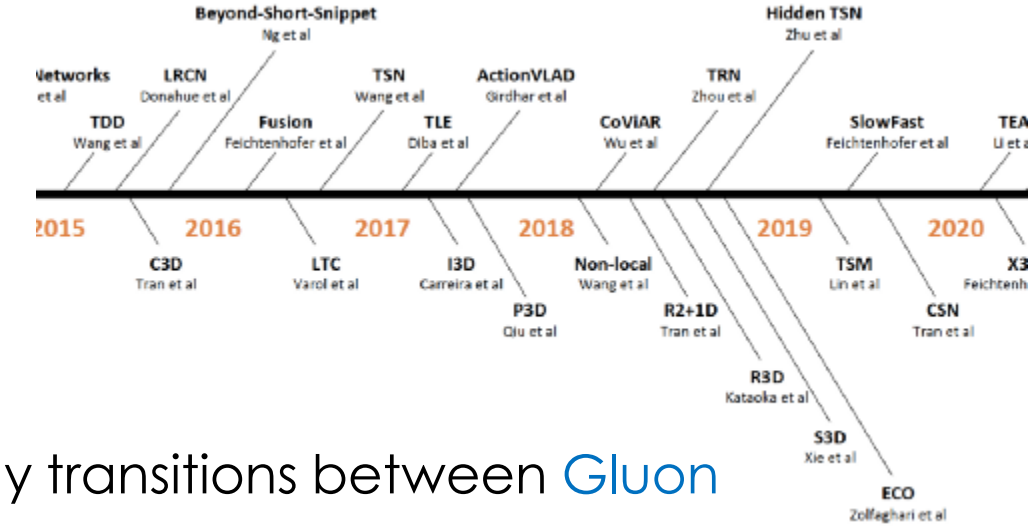
- Torch - Torch ML library makes ML easy and efficient, thanks to its use of the Lua scripting language and a GPU-first implementation. It comes with a large collection of community-driven ML packages that cover computer vision, signal processing, audio processing, networking and more. Its neural network and optimization libraries are designed to be both accessible and flexible.

Top Machine Learning Libraries - VI

- Keras – deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible is key to doing good research. Follows the principle of progressive disclojure of complexity: it makes it easy to get started, yet it makes it possible to handle arbitrarily advanced use cases, only requiring incremental learning at each step – e.g. customizing the training loop by combining Keras functionality with the TensorFlow GradientTape:

```
optimizer = tf.keras.optimizers.Adam()
loss_fn = tf.keras.losses.kl_divergence
with tf.GradientTape() as tape:
    predictions = model(inputs)
    loss_value = loss_fn(targets, predictions)
   gradients = tape.gradient(loss_value, model.trainable_weights)
   optimizer.apply_gradients(zip(gradients, model.trainable_weights))
   …
```

Top Machine Learning Libraries



Apache MXNet:

- Hybrid Front-End – a hybrid front-end seamlessly transitions between Gluon eager imperative mode and symbolic mode to provide both flexibility and speed.
- GluonCV – a computer vision toolkit with rich model zoo – from object detection to pose estimation.
- GluonNLP – provides state-of-the-art deep learning models in NLP – for engineers and researchers to fast prototype research ideas and products.
- GluonTS – Gluon Time Series is the Gluon toolkit for probabilistic time series modeling, focusing on deep learning-based models.

Picture source: https://medium.com/apache-mxnet/which-one-is-the-best-algorithm-for-video-action-recognition-298fb5c4ad4f

Deep Learning

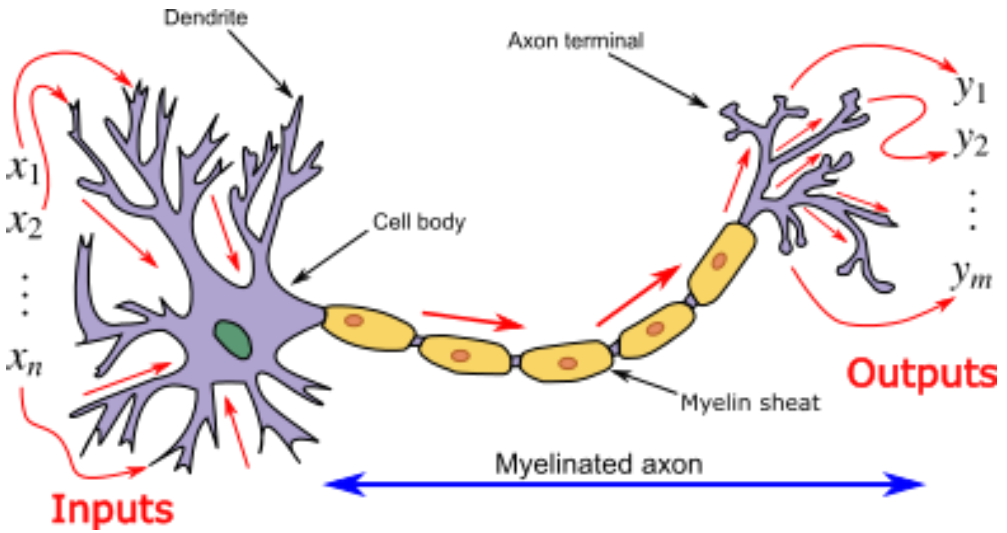
Convolutional Neural Networks

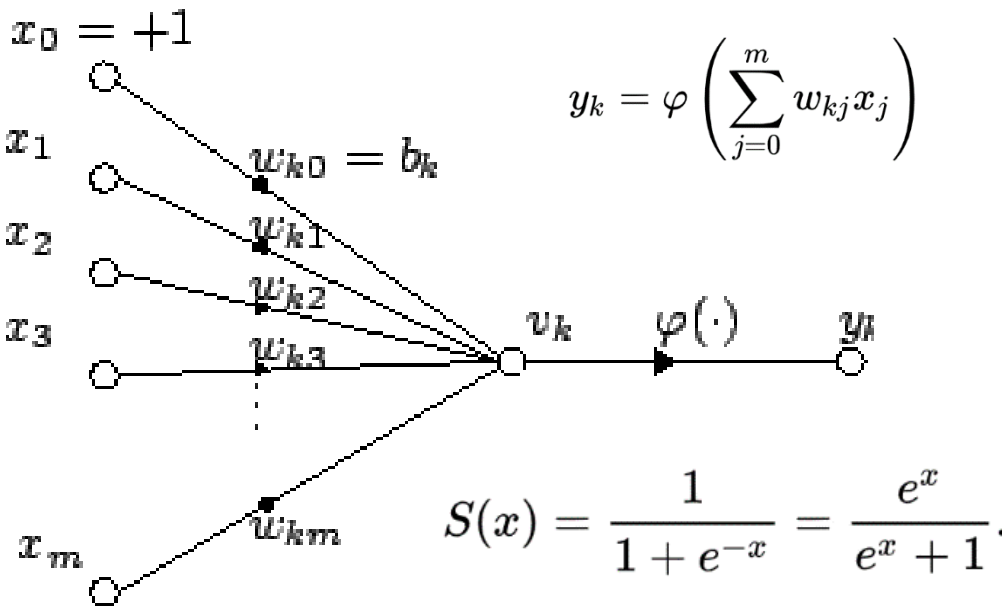
Deep Learning

- Deep learning (deep structured learning) - artificial neural networks with representation learning – supervised, semi-supervised or unsupervised.
- Deep-learning architectures such as deep neural networks, deep belief networks, recurrent neural networks (RNN) and convolutional neural networks (CNN) have been applied to fields including computer vision, machine vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and game programing, where they have produced results comparable to and in some cases surpassing human expert performance.
- Artificial neural networks (ANNs) were inspired by information processing and distributed communication nodes in biological systems. ANNs have various differences from biological brains. Specifically, neural networks tend to be static and symbolic, while the biological brain of most living organisms is dynamic (plastic) and analog.

Artificial Neural Networks (ANNs)

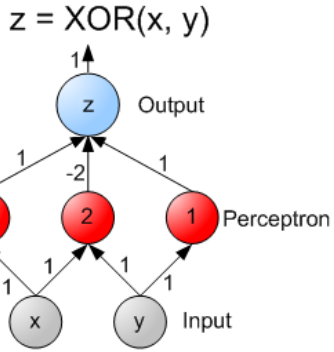
- ANN is based on a collection of connected units or nodes called artificial neurons, which model the neurons in a biological brain.
- Synapses can transmit a signal to other neurons, "signal" is a real number, output is computed by some non-linear function of the sum of its inputs -> threshold.
- The connections are called edges - typically have weight that adjusts as learning proceeds.
- Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

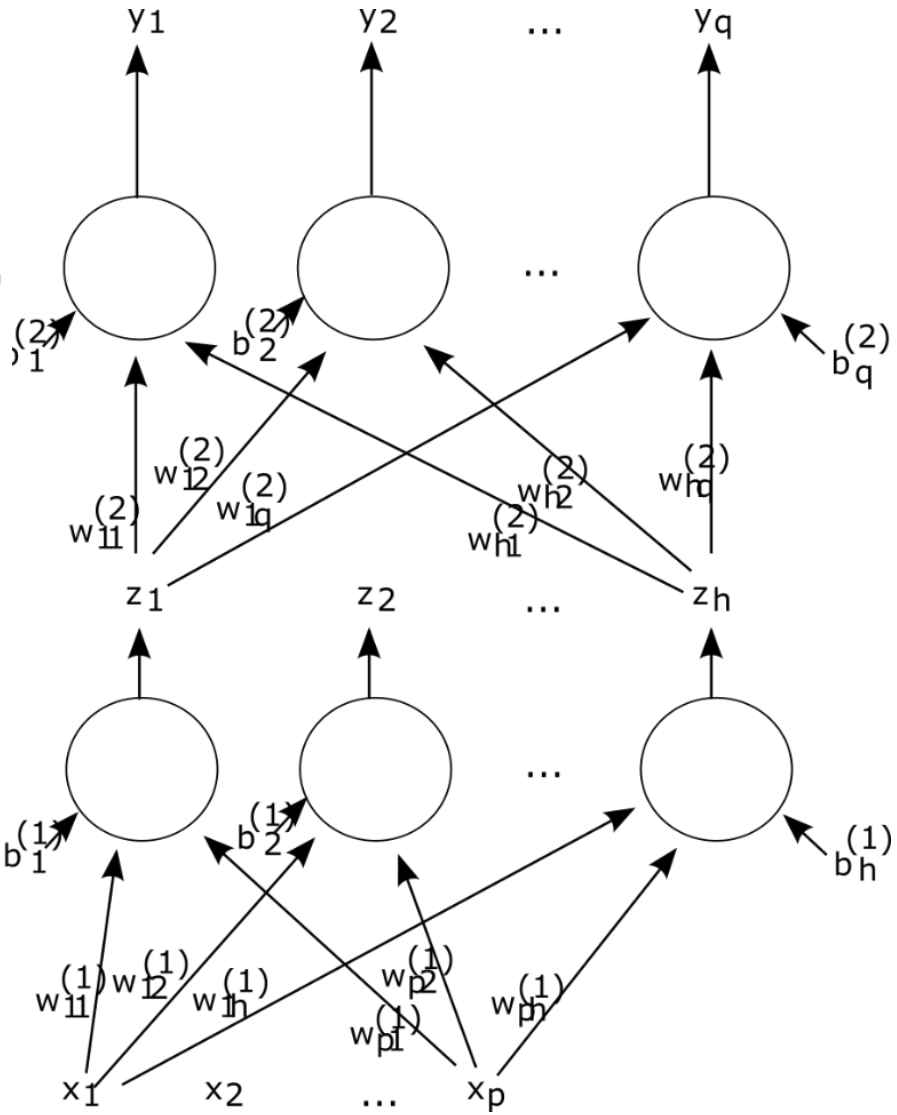


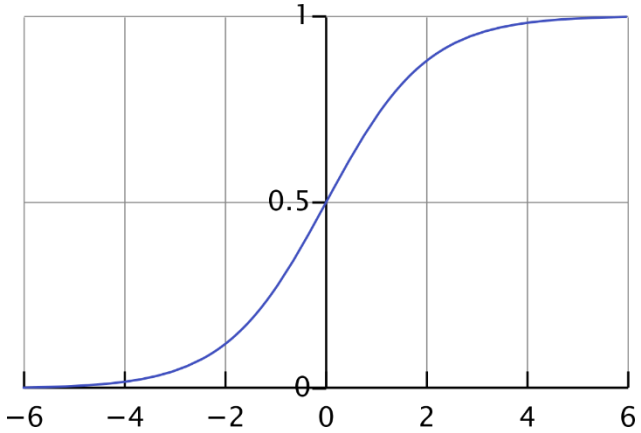


Feed Forward Networks (FFN)

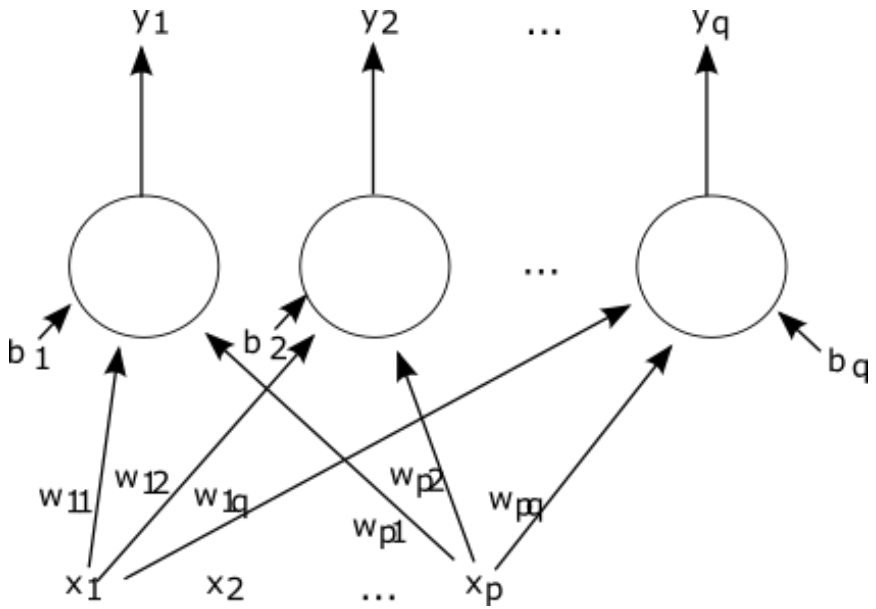
- FFN - artificial neural network wherein connections between the nodes do *not* form a cycle. As such, it is different from its descendant: recurrent neural networks.







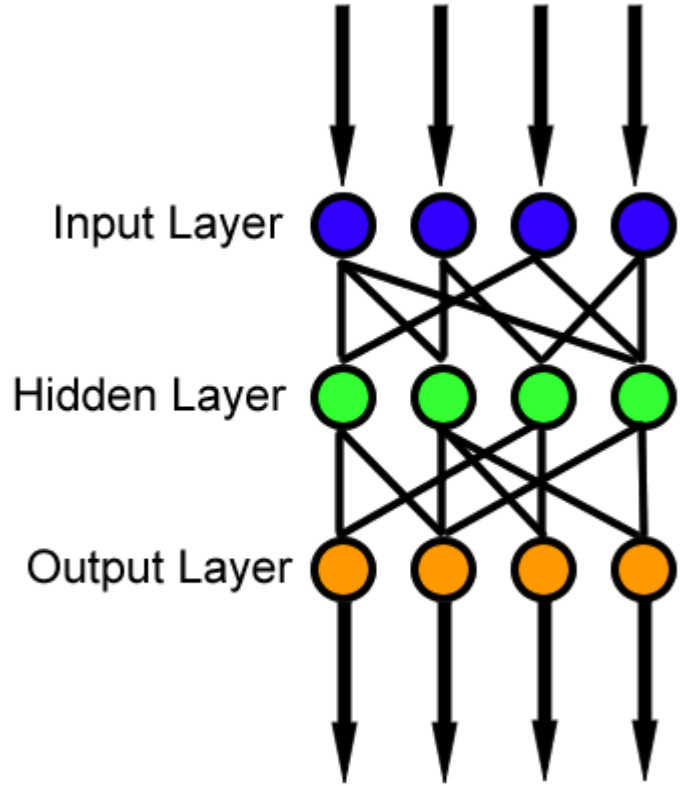
$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$

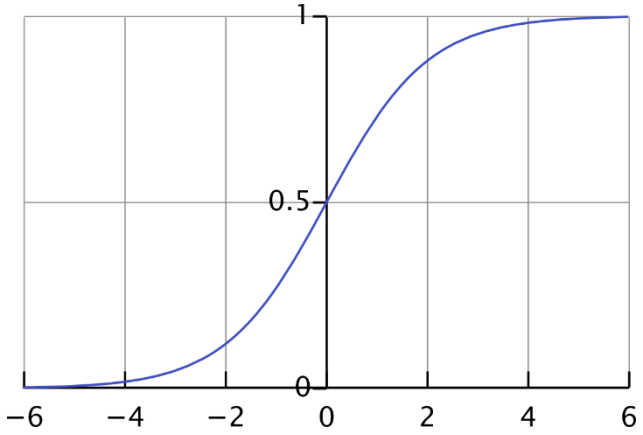


$$y_q = K*(\sum(x_i * w_{iq}) - b_q)$$

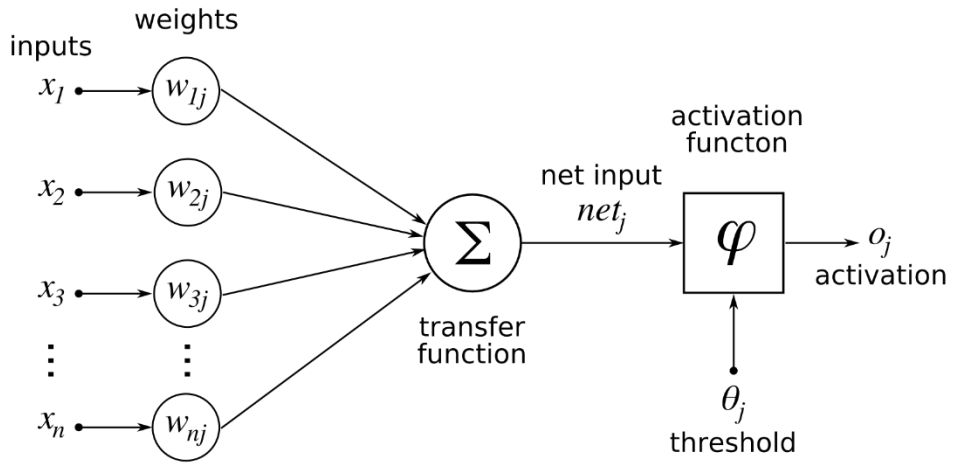
ANNs, Multilayer Perceptron (MLP)

- MLP - class of feedforward artificial neural network (ANN), three layers: input layer, hidden layer, and output layer, nonlinear activation function, supervised learning technique called backpropagation for training.





$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}.$$



Error Backprpagation

Learning occurs in the perceptron by changing connection weights after each piece of data is processed, based on the amount of error in the output compared to the expected resu
This is an example of supervised learning, and is carried out through backpropagation, a generalization of the least mean squares algorithm in the linear perceptron.

We can represent the degree of error in an output node j in the nth data point (training example) by $e_j(n) = d_j(n) - y_j(n)$, where d is the target value and y is the value produced by the perceptron. The node weights can then be adjusted based on corrections that minimize the error in the entire output, given by

$$\mathcal{E}(n) = \frac{1}{2}\sum_j e_j^2(n).$$

Using gradient descent, the change in each weight is

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} y_i(n)$$

where y_i is the output of the previous neuron and η is the *learning rate*, which is selected to ensure that the weights quickly converge to a response, without oscillations.

The derivative to be calculated depends on the induced local field v_j, which itself varies. It is easy to prove that for an output node this derivative can be simplified to

$$-\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = e_j(n)\phi'(v_j(n))$$

where ϕ' is the derivative of the activation function described above, which itself does not vary. The analysis is more difficult for the change in weights to a hidden node, but it can b shown that the relevant derivative is

$$-\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = \phi'(v_j(n))\sum_k -\frac{\partial \mathcal{E}(n)}{\partial v_k(n)} w_{kj}(n).$$

This depends on the change in weights of the kth nodes, which represent the output layer. So to change the hidden layer weights, the output layer weights change according to the derivative of the activation function, and so this algorithm represents a backpropagation of the activation function.[5]

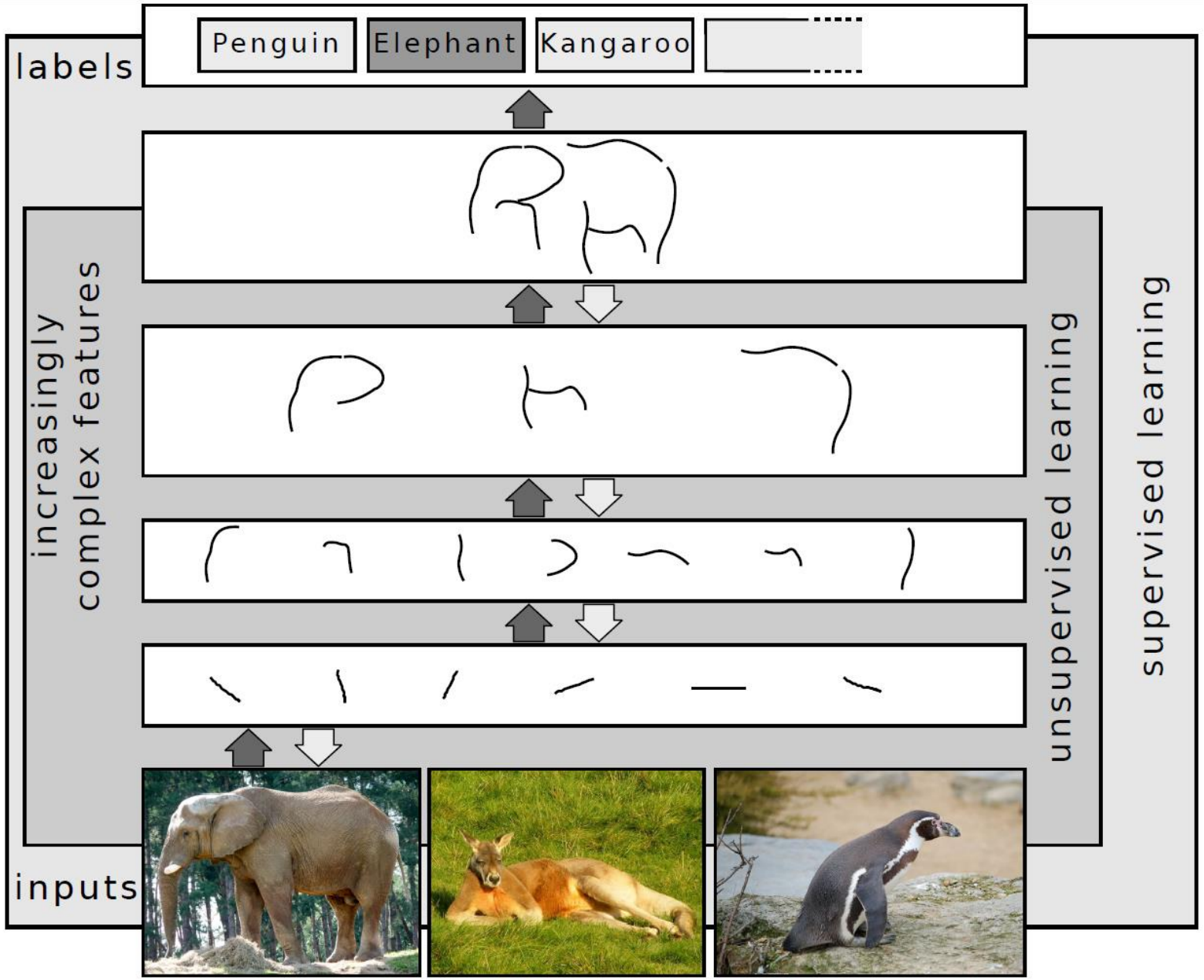
Source: https://en.wikipedia.org/wiki/Multilayer_perceptron

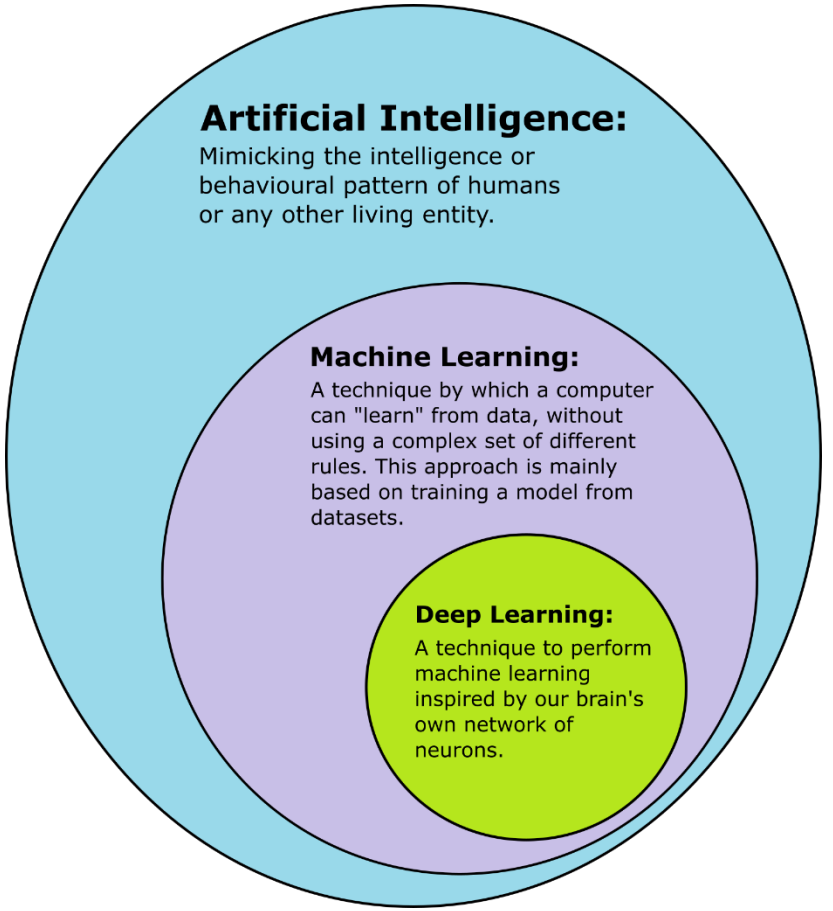
Deep Learning

- The adjective "deep" in deep learning comes from the use of multiple layers in the network. Early work showed that a linear perceptron cannot be a universal classifier, and that a network with a nonpolynomial activation function with one hidden layer of unbounded width can be so.
- Deep learning – unbounded number of layers of bounded size, which permits practical application and optimized implementation, while retaining theoretical universality under mild conditions.
- Layers can be heterogeneous and to deviate widely from biologically informed connectionist models, for the sake of efficiency, trainability and understandability, whence the "structured" part.

Convolutional Neural Networks (CNN)

- Convolutional Neural Network (CNN, or ConvNet) - a class of deep neural networks, most commonly applied to analyzing visual imagery, also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics, have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, brain-computer interfaces, and financial time series.
- CNNs are regularized versions of multilayer perceptrons – MPLs usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer -> overfitting, typical ways of regularization include adding measurement of weights to the loss function.
- CNNs take a different approach towards regularization – they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns -> lower connectedness and complexity.





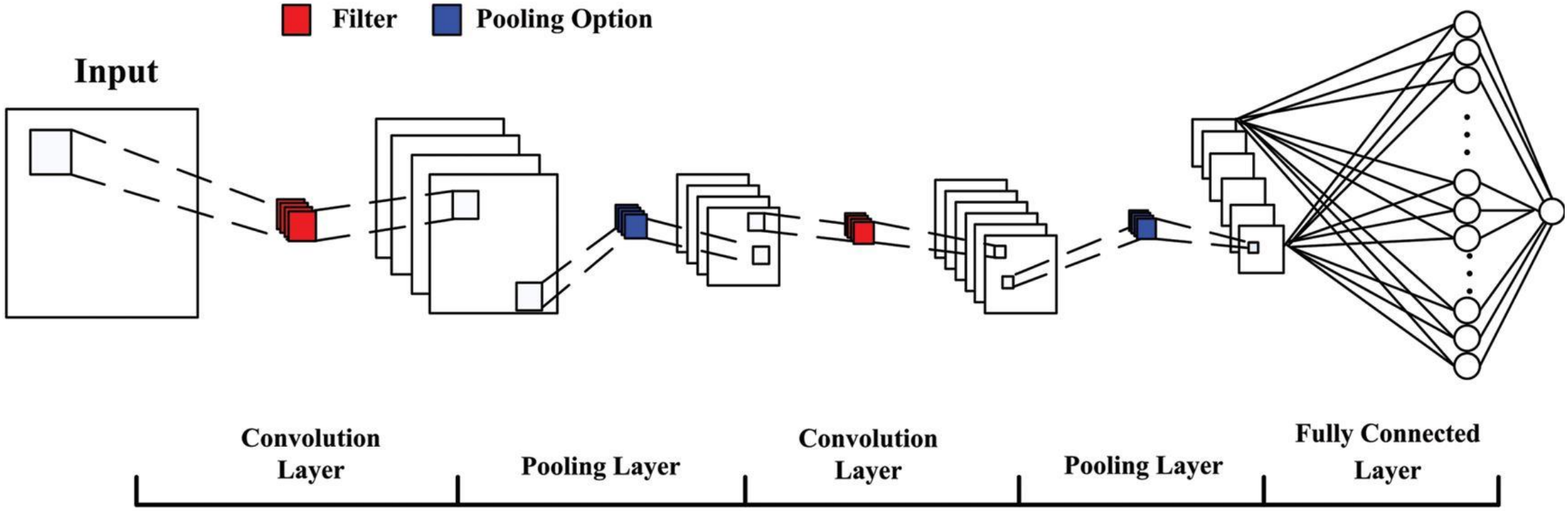
Picture by: By Sven Behnke - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=82466022
By Original file: Avimanyu786SVG version: Tukijaaliwa - File:AI-ML-DL.png, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=90131352

Convolutional Neural Networks (CNN)

- Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

- CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

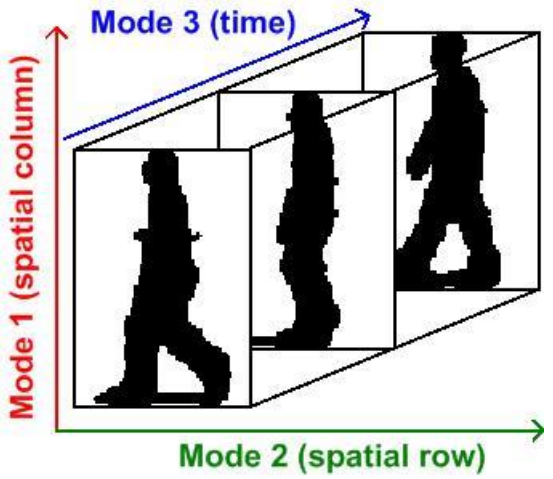
Convolutional Neural Networks (CNN)



Picture by: https://wiki.seg.org/wiki/Machine_learning

Tensors

- **Tensor** – algebraic object that describes a (multilinear) relationship between sets of algebraic objects related to a vector space. Objects that tensors may map between include vectors and scalars, and even other tensors. Tensors can take several different forms – for example: scalars and vectors (which are the simplest tensors), dual vectors, multilinear maps between vector spaces, and even some operations such as the dot product. Tensors are defined independent of any basis, although they are often referred to by their components in a basis related to a particular coordinate system.



  - Examples:
    - Vector data – 2D tensors of shape (samples, features)
    - Timeseries data or sequence data— 3D tensors of shape (samples, timesteps, features)
    - Images – 4D tensors of shape (samples, height, width, channels) or (samples, channels, height, width)
    - Video - 5D tensors of shape (samples, frames, height, width, channels) or (samples, frames, channels, height, width)

Source: https://en.wikipedia.org/wiki/Tensor
Image by WikiMSL - Make it in Microsoft Visio myself, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=23571011

Tensor Operations

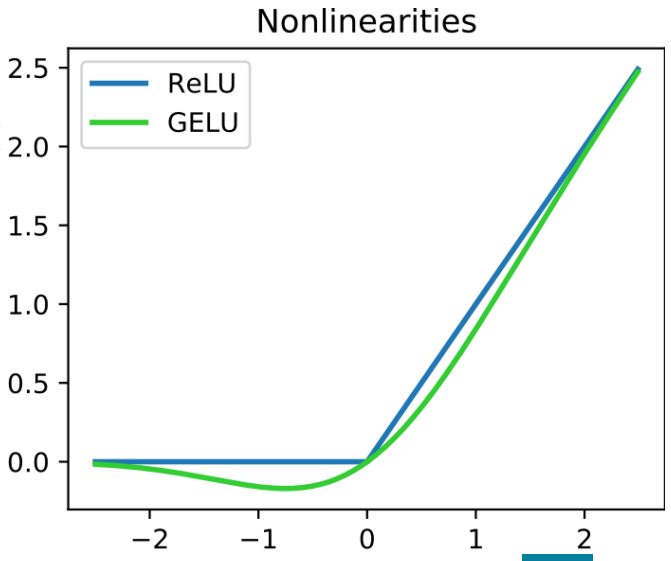
All transformations learned by deep neural networks can be reduced to a number of tensor operations applied to tensors of numeric data. For instance, it's possible to add tensors, multiply tensors, etc. For example:

keras.layers.Dense(512, activation='relu')

can be interpreted as a function, which takes as input a 2D tensor and returns another 2D tensor – as a function of the input tensor. If **W** is a 2D tensor and **b** is a vector (both are attributes of the layer):

output = relu(dot(W, input) + b) – there are 3 tensor operations here:

1) dot product (**dot**) between the input tensor and **W** tensor;
2) an addition (**+**) between the resulting 2D tensor and vector **b**;
3) a **ReLU** operation: **relu(x)** = **max(x, 0)**



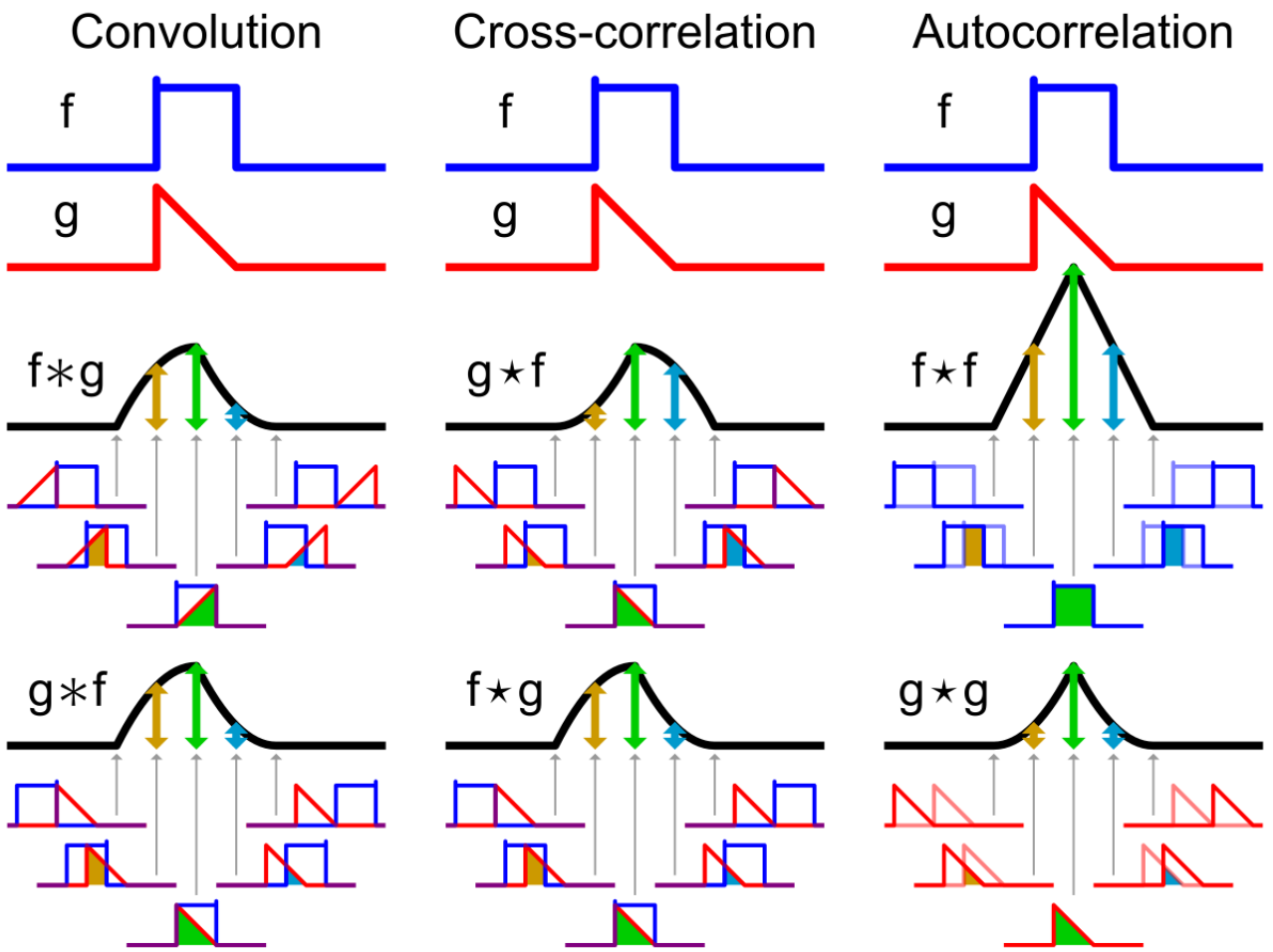
Sources: https://en.wikipedia.org/wiki/Tensor, https://en.wikipedia.org/wiki/Activation_function
Image by Ringdongdang - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=95947821

Convolution

$$f(t) * g(t) := \underbrace{\int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau}_{(f*g)(t)}$$

$$f(t) * g(t) := \underbrace{\int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau}_{(f*g)(t)}$$

- Convolution – a mathematical operation on two functions (**f** and **g**) that produces a third function **f*g** that expresses how the shape of one is modified by the other.
- The term convolution refers to both the result function and to the process of computing it. It is defined as the integral of the product of the two functions after one is reversed and shifted.
- The integral is evaluated for all values of shift, producing the convolution function.



Source: https://en.wikipedia.org/wiki/Convolution
Image by Cmglee - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=20206883

Multidimensional Discrete Convolution

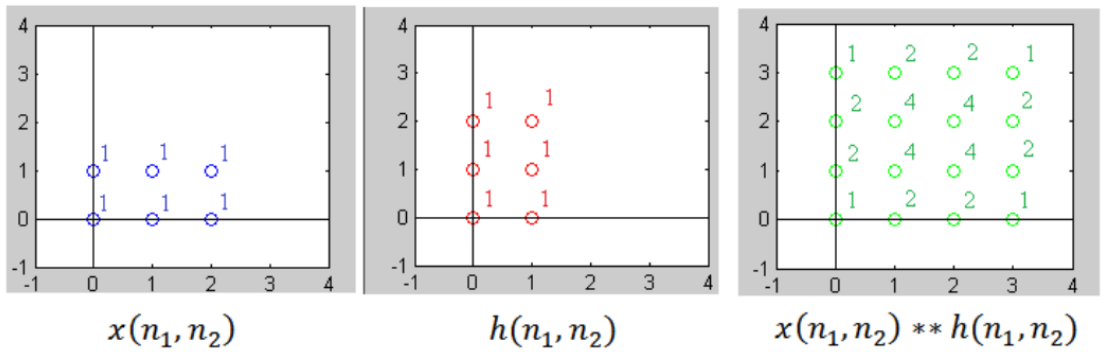
Similar to the one-dimensional case, an asterisk is used to represent the convolution operation. The number of dimensions in the given operation is reflected in the number of asterisks. For example, an *M*-dimensional convolution would be written with *M* asterisks. The following represents a *M*-dimensional convolution of discrete signals:

$$y(n_1, n_2, \ldots, n_M) = x(n_1, n_2, \ldots, n_M) *^{M}* h(n_1, n_2, \ldots, n_M)$$

For discrete-valued signals, this convolution can be directly computed via the following:

$$\sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \cdots \sum_{k_M=-\infty}^{\infty} h(k_1, k_2, \ldots, k_M) x(n_1 - k_1, n_2 - k_2, \ldots, n_M - k_M)$$

The resulting output region of support of a discrete multidimensional convolution will be determined based on the size and regions of support of the two input signals.



Visualization of Convolution between two Simple Two-Dimensional Signals

Explained by example:
https://www.allaboutcircuits.com/technical-articles/two-dimensional-convolution-in-image-processing/

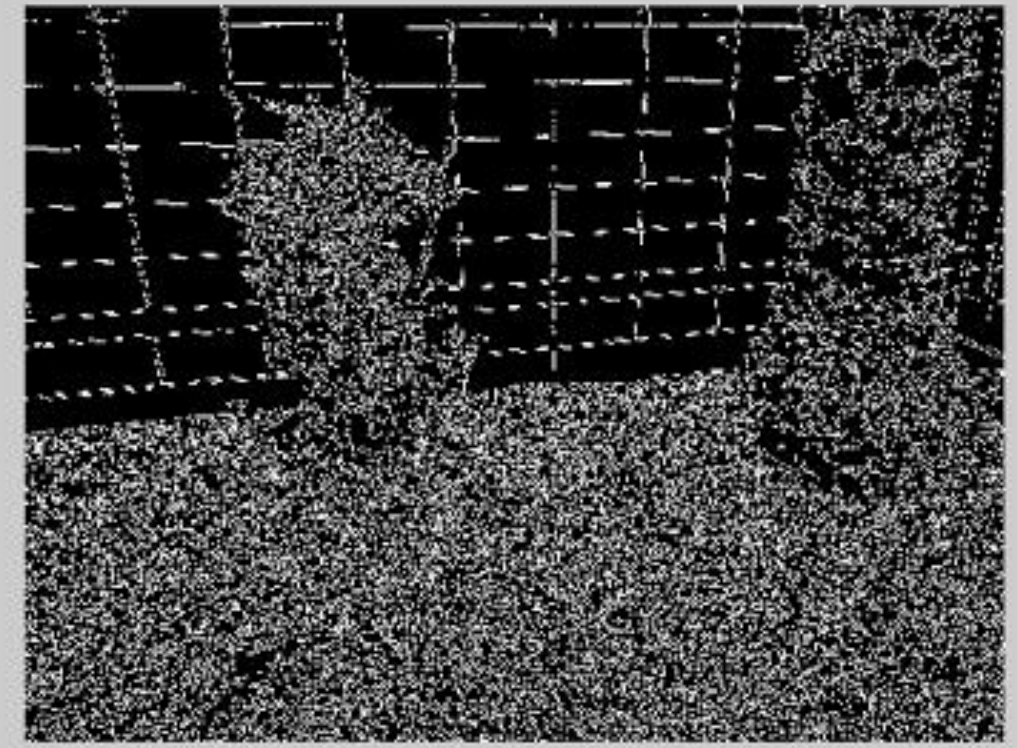
Listed are several properties of the two-dimensional convolution operator. Note that these can also be extended for signals of N-dimensions.

Source: https://en.wikipedia.org/wiki/Multidimensional_discrete_convolution
Image by Shaddowffax - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=44909356

2D Convolution Example – Edge Detection

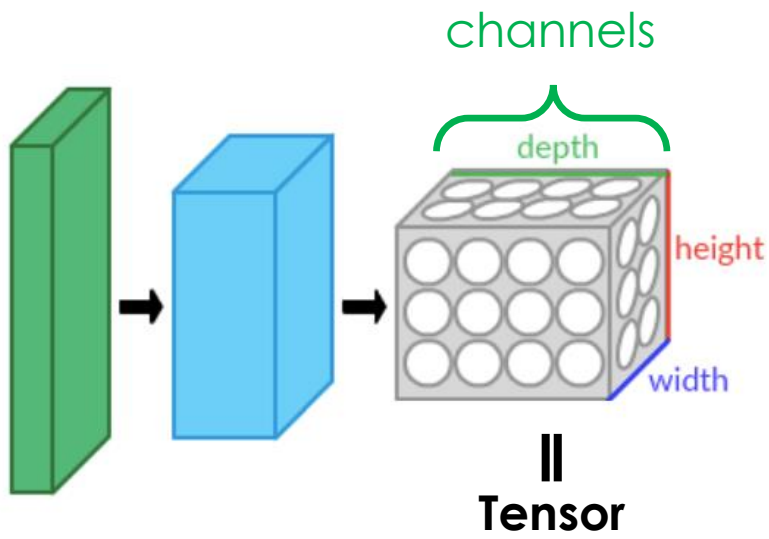
0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

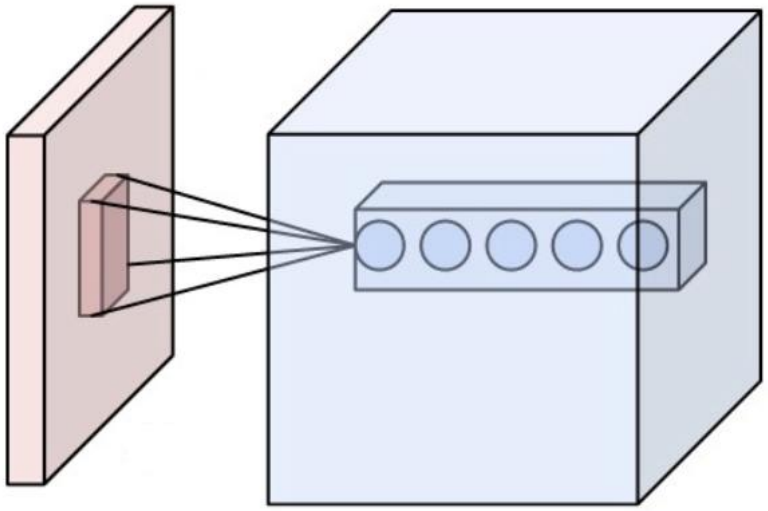


CNNs Specific Features - I

- **3D volumes of neurons** - the layers of a CNN have neurons arranged in 3 dimensions: width, height and depth where each neuron inside a convolutional layer is connected to only a small region of the layer before it, called a receptive field. Distinct types of layers, both locally and completely connected, are stacked to form a CNN architecture.



- **Local connectivity** - CNNs exploit spatial locality by enforcing a local connectivity pattern between neurons of adjacent layers. The architecture thus ensures that the learned "filters" produce the strongest response to a spatially local input pattern. Stacking many such layers leads to non-linear filters that become increasingly global (i.e. responsive to a larger region of pixel space) so that the network first creates representations of small parts of the input, then from them assembles representations of larger areas.



Source: https://en.wikipedia.org/wiki/Convolutional_neural_network, By Aphex34 - CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45661858
By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

CNNs Specific Features - II

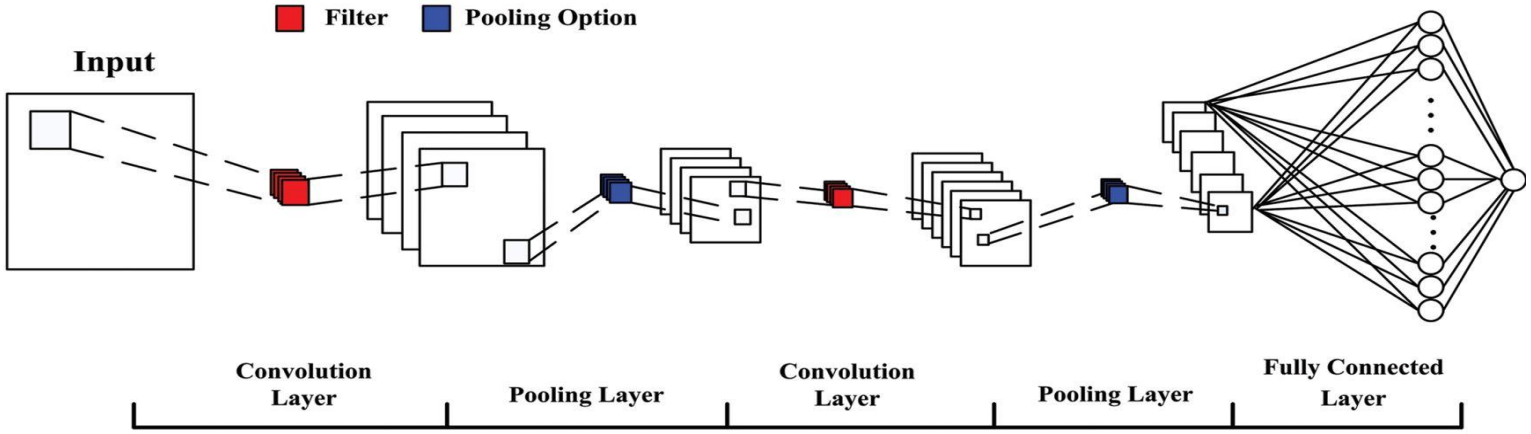
- **Shared weights** – in CNNs, each filter is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a feature map. This means that all the neurons in a given convolutional layer respond to the same feature within their specific response field. Replicating units allows for the feature map to be equivariant under changes in the locations of input features in the visual field -> translational equivariance.

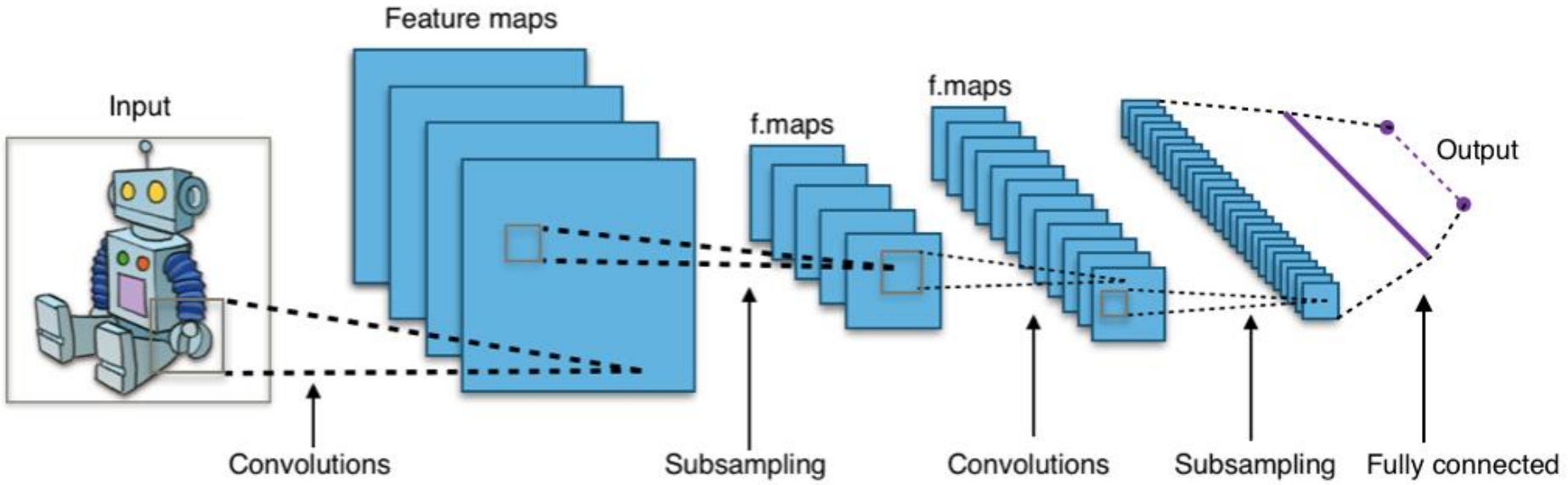
- **Pooling** – in a CNN's pooling layers, feature maps are divided into rectangular sub-regions, and the features in each rectangle are independently down-sampled to a single value, commonly by taking their average or maximum value. In addition to reducing the sizes of feature maps, the pooling operation grants a degree of translational invariance to the features contained therein, allowing the CNN to be more robust to variations in their positions.

- Together, these properties allow CNNs to achieve better generalization on vision problems. Weight sharing dramatically reduces the number of free parameters learned, thus lowering the memory requirements for running the network and allowing the training of larger, more powerful networks.

Source: https://en.wikipedia.org/wiki/Convolutional_neural_network

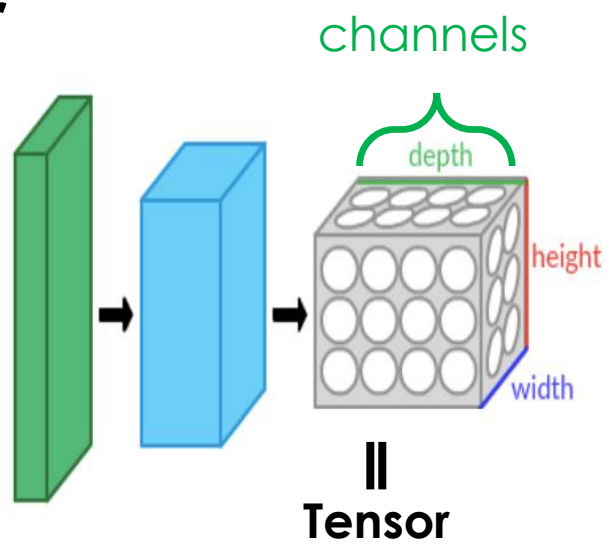
Convolutional Neural Networks (CNN)



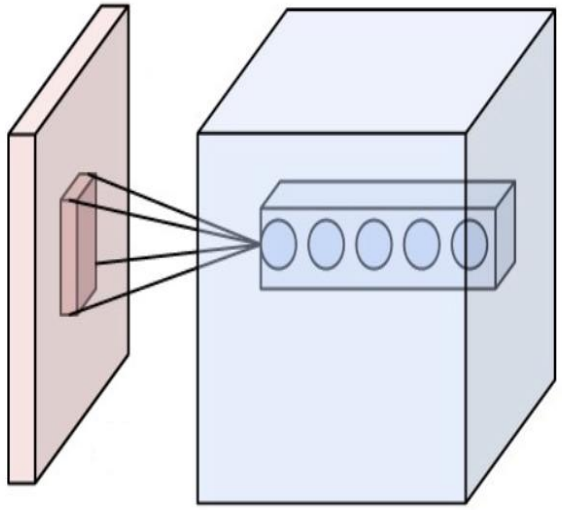


Picture by: https://wiki.seg.org/wiki/Machine_learning
By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45679374

CNN – Types of Layers – Convolutional layer



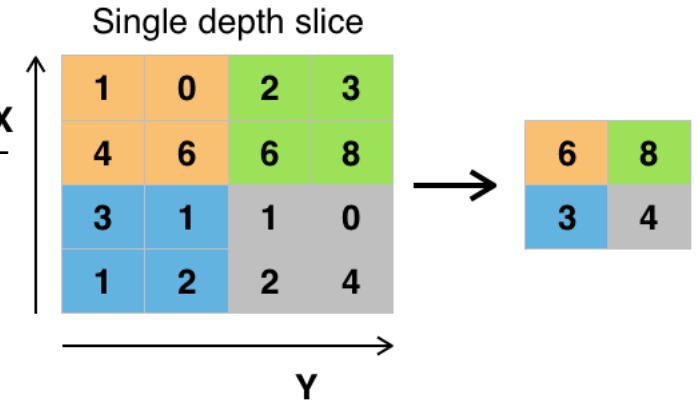
- Local connectivity - the extent of this connectivity is a hyperparameter called the receptive field of the neuron - connections are local in space (along width and height), but always extend along the entire depth of the input volume.
- Spatial arrangement – depth, stride and zero-padding – depth of the output volume controls the number of neurons in a layer that connect to the same region of the input volume. These neurons learn to activate for different features in the input (e.g. various oriented edges, or blobs of color); 2) stride – 1 meaning we move the filters one pixel at a time; 3) padding input with zeros on the border of the input volume.
- Parameter sharing - used in convolutional layers to control the number of free parameters. It relies on the assumption that if a patch feature is useful to compute at some spatial position, then it should also be useful to compute at other positions., it is common to refer to the sets of weights as a filter (or a kernel), which is convolved with the input. -> translation invariance of CNN.

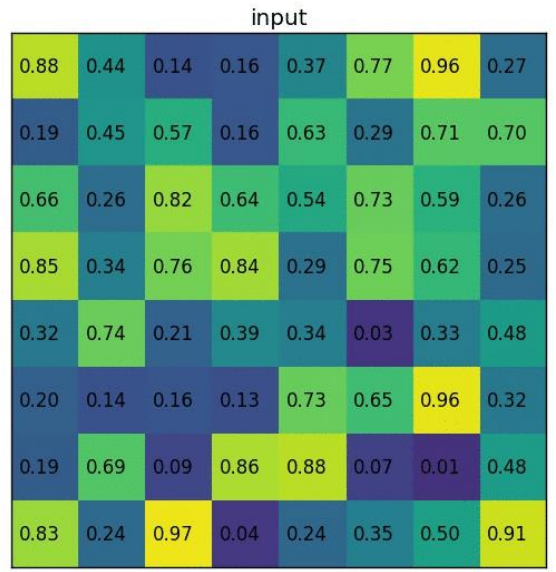


Source: https://en.wikipedia.org/wiki/Convolutional_neural_network, By Aphex34 - CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45661858
By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

CNN – Types of Layers - Pooling layer

- Pooling = non-linear down-sampling. There are several non-linear functions to implement pooling among which max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum - location of a feature is less important than its rough location relative to other features.
- The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters, memory footprint and amount of computation in the network, and hence to also control overfitting. It is common to periodically insert a pooling layer between successive convolutional layers (each one typically followed by a ReLU layer) in a CNN architecture.
- The most common form is a pooling layer with filters of size 2×2 applied with a stride of 2 downsamples at every depth slice in the input by 2 along both width and height, discarding 75% of the activations.
- ℓ2-norm pooling, Region of Interest (RoI) pooling - output size is fixed and input rectangle is a parameter – most often pooling pooling to size 2x2. In this example region proposal (an input parameter) has size 7x5.





Source: By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45673581
By Tomasz Grel (deepsense.io) - https://deepsense.io/region-of-interest-pooling-explained/, CC BY 4.0

CNN Example with Keras and TensorFlow (MNIST Dataset)

```
from keras import layers
from keras import models
from keras.datasets import mnist
from keras.utils import to_categorical


model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.summary()
```

CNN Example with Keras and TensorFlow - Summary

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
```

CNN Example with Keras – Training & Testing (with MNIST)

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test Accuracy: ${test_acc}')
print(f'Test Loss: ${test_loss}')
print('Demo finished')
```

CNN Example with Keras – Training & Testing (CPU only)

```
938/938 [==============================] - 18s 14ms/step - loss: 0.3878 - accuracy: 0.8740
Epoch 2/5
938/938 [==============================] - 13s 14ms/step - loss: 0.0502 - accuracy: 0.9852
Epoch 3/5
938/938 [==============================] - 13s 14ms/step - loss: 0.0326 - accuracy: 0.9897
Epoch 4/5
938/938 [==============================] - 13s 14ms/step - loss: 0.0248 - accuracy: 0.9928
Epoch 5/5
938/938 [==============================] - 13s 14ms/step - loss: 0.0175 - accuracy: 0.9945
313/313 [==============================] - 1s 2ms/step - loss: 0.0286 - accuracy: 0.9911
Test Accuracy: 0.991100013256073
Test Loss: 0.028569826856255 53
Demo finished
```

CNN Example with Keras – Training & Testing (using GPU)

```
938/938 [==============================] - 7s 4ms/step - loss: 0.3917 - accuracy: 0.8749
Epoch 2/5
938/938 [==============================] - 4s 4ms/step - loss: 0.0490 - accuracy: 0.9851
Epoch 3/5
938/938 [==============================] - 3s 3ms/step - loss: 0.0334 - accuracy: 0.9897
Epoch 4/5
938/938 [==============================] - 3s 3ms/step - loss: 0.0242 - accuracy: 0.9928
Epoch 5/5
938/938 [==============================] - 3s 3ms/step - loss: 0.0183 - accuracy: 0.9946
313/313 [==============================] - 1s 2ms/step - loss: 0.0213 - accuracy: 0.9930
Test Accuracy: 0.9930000305175781
Test Loss: 0.021334622055292 13
Demo finished
```

Search nodes. Regexes supported.

Fit to Screen

Download PNG

Run (4) 20210109-050204\train

Tag (3) Default

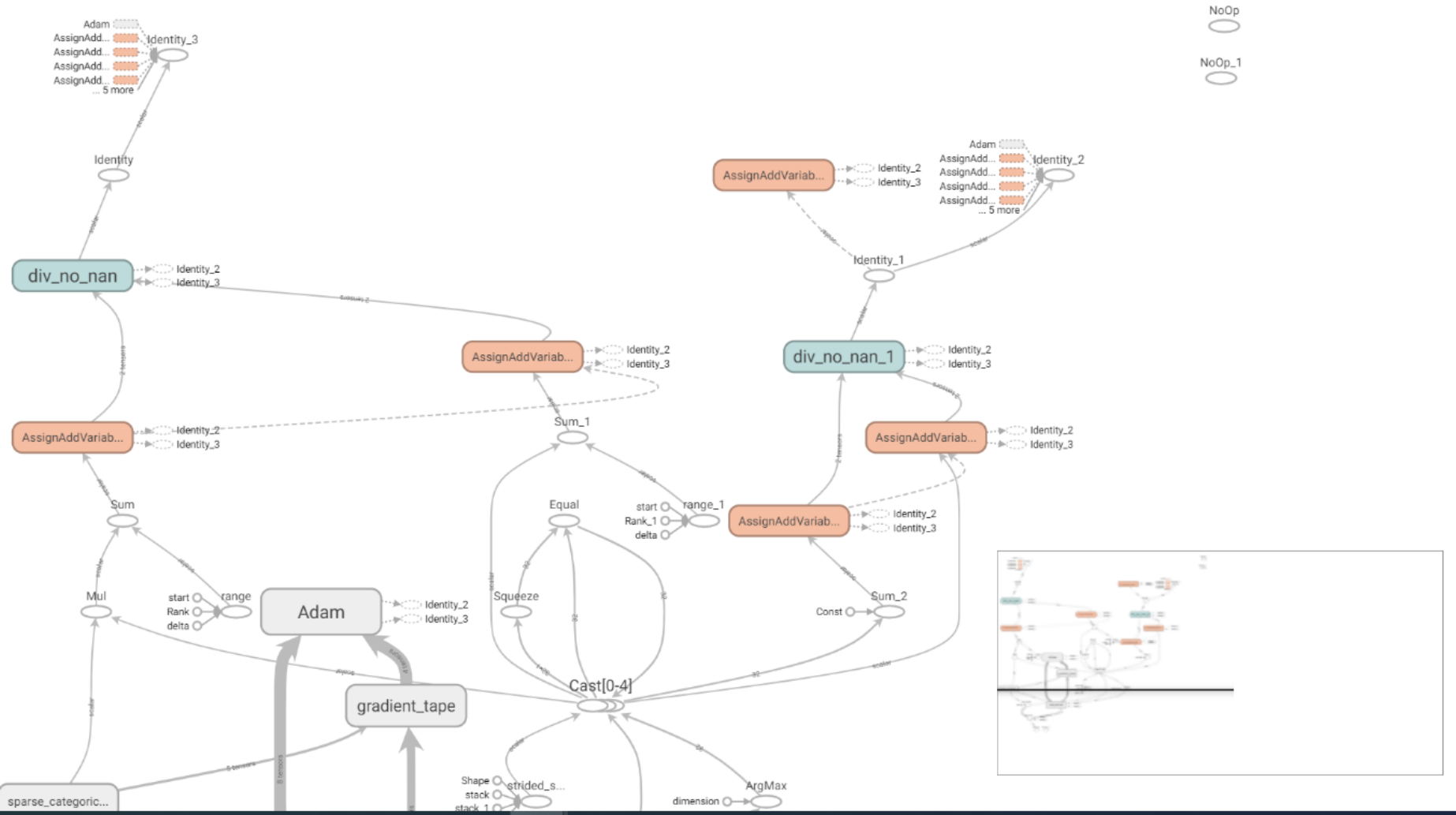
Upload Choose File

- Graph
- Conceptual Graph
- Profile

Close legend.

Graph (* = expandable)

- Namespace* ?
- OpNode ?
- Unconnected series* ?
- Connected series* ?
- Constant ?
- Summary ?
- Dataflow edge ?
- Control dependency edge ?
- Reference edge ?



Histogram mode

OVERLAY OFFSET

Offset time axis

STEP RELATIVE WALL

Runs

Write a regex to filter runs

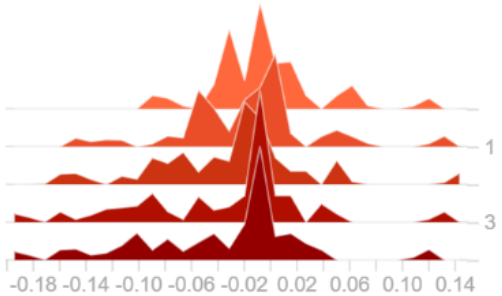
- [ ] 20210109-050204\train
- [ ] 20210109-050204\validation
- [x] 20210109-051457\train
- [x] 20210109-053307\train
- [x] 20210109-053933\train

TOGGLE ALL RUNS

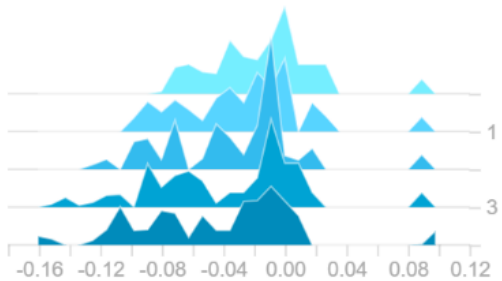
./logs

conv2d

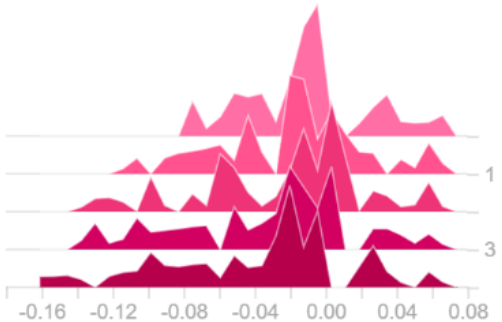
conv2d/bias_0 20210109-051457\train



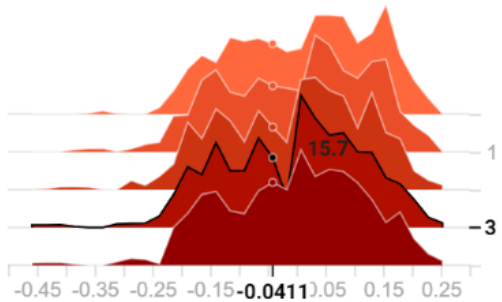
conv2d/bias_0 20210109-053307\train



conv2d/bias_0 20210109-053933\train



conv2d/kernel_0 20210109-051457\train



conv2d/kernel_0 20210109-053307\train

conv2d/kernel_0 20210109-053933\train

CAPTURE PROFILE

Runs (4)

20210109-053933\train\2021_01_...

Tools (8)

overview_page

Hosts

OFFICE27

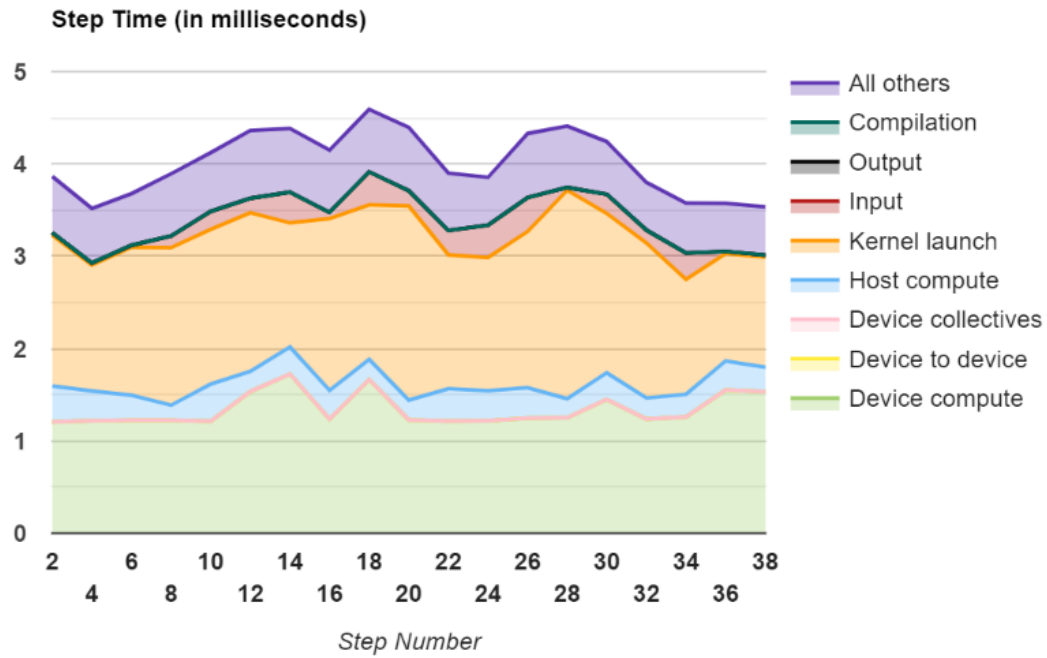
Performance Summary

Average Step Time — 4.0 ms
lower is better
(σ = 0.3 ms)

- Idle: ms
- Input: ms
- Compute: ms
- Idle: ms
- Input: ms
- Compute: ms

- All Others Time — 0.6 ms
  (σ = 0.1 ms)
- Compilation Time — 0.0 ms
  (σ = 0.0 ms)
- Output Time — 0.0 ms
  (σ = 0.0 ms)
- Input Time — 0.2 ms
  (σ = 0.1 ms)
- Kernel Launch Time — 1.6 ms
  (σ = 0.3 ms)

Step-time Graph



Recommendation for Next Step

- Your program is POTENTIALLY input-bound because 15.4% of the total step time sampled is spent on 'All Others' time (which could be due to I/O or Python execution or both).

CAPTURE PROFILE

Runs (4)

20210109-053933\train\2021_01_...

Tools (8)

tensorflow_stats

Hosts

OFFICE27

TensorFlow Stats

(1) In the charts and table below, "IDLE" represents the portion of the total execution time on device (or host) that is idle.
(2) In the pie charts, the "Other" sector represents the sum of sectors that are too small to be shown individually.

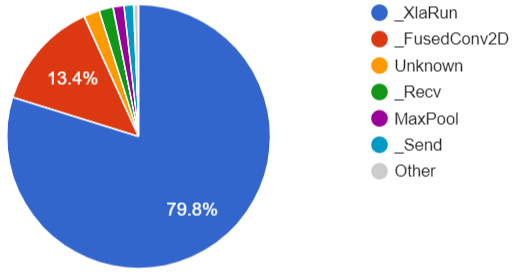
Export as CSV

Include IDLE time in statistics

No

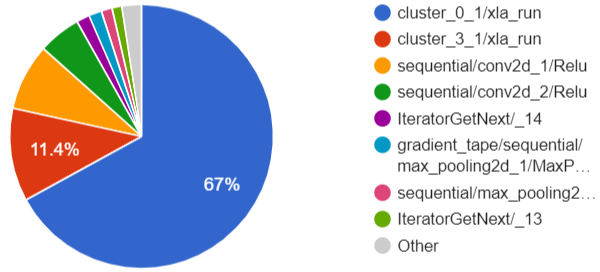
ON DEVICE: TOTAL SELF-TIME (GROUPED BY TYPE)

(in microseconds) of a TensorFlow operation type



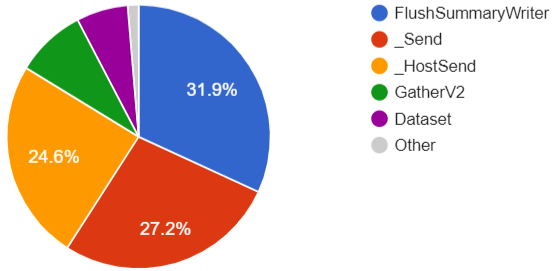
ON DEVICE: TOTAL SELF-TIME

(in microseconds) of a TensorFlow operation



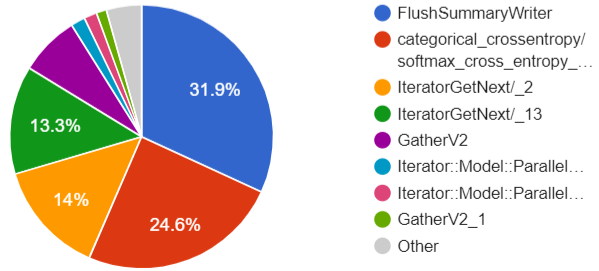
ON HOST: TOTAL SELF-TIME (GROUPED BY TYPE)

(in microseconds) of a TensorFlow operation type



ON HOST: TOTAL SELF-TIME

(in microseconds) of a TensorFlow operation



CAPTURE PROFILE

Runs (4)

20210109-053933\train\2021_01_...

Tools (8)

kernel_stats

Hosts

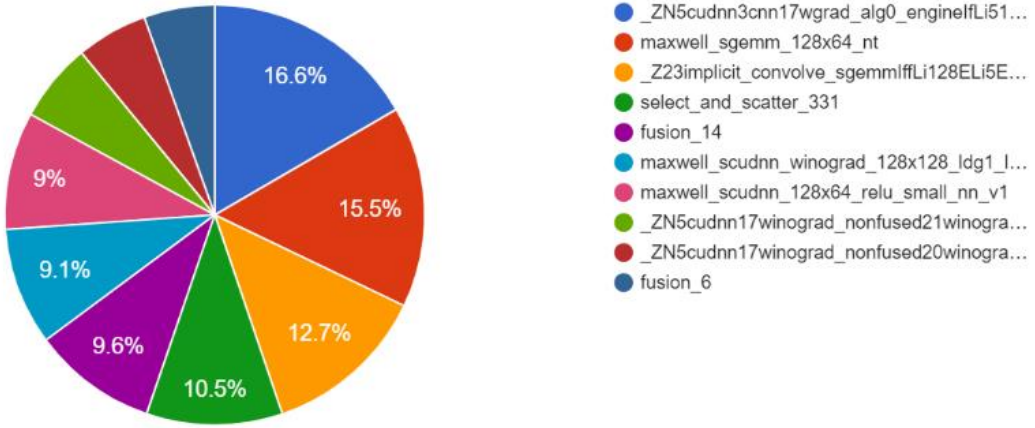
OFFICE27

GPU Kernel Stats

Export as CSV

Top 10 Kernels with highest Total Duration

Show top 10 Kernels



GPU Kernels

Kernel Name Op Name

CAPTURE PROFILE

Runs (4)

20210109-053933\train\2021_01_...

Tools (8)

overview_page

Hosts

OFFICE27

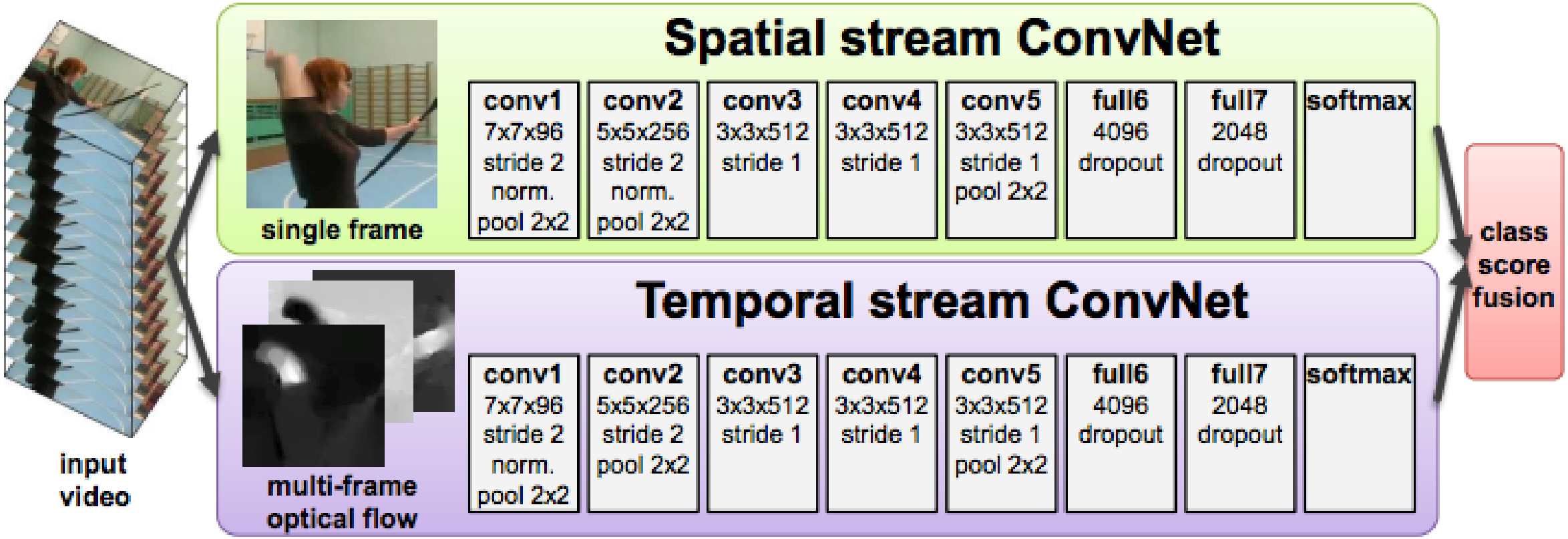
Device type: Nvidia GPU (Pascal)

Number of device cores: 1

Top 10 TensorFlow operations on GPU

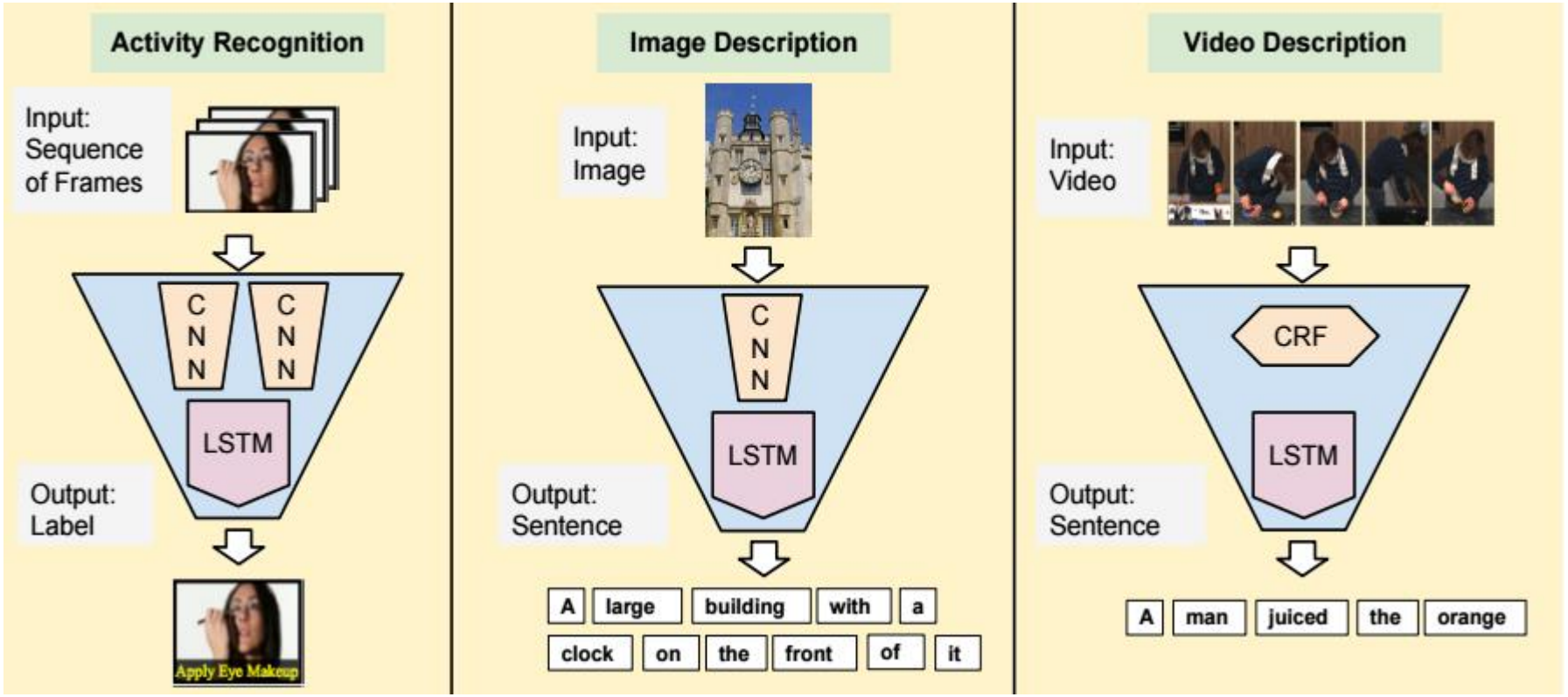
Time (%)	Cumulative time (%)	Category	Operation	TensorCore eligibility	Op is using TensorCore
67%	67%	_XlaRun	cluster_0_1/xla_run	X	X
11.4%	78.4%	_XlaRun	cluster_3_1/xla_run	X	X
8.2%	86.6%	_FusedConv2D	sequential/conv2d_1/Relu	X	X
5.2%	91.8%	_FusedConv2D	sequential/conv2d_2/Relu	X	X
1.6%	93.5%	_Recv	IteratorGetNext/_14	X	X
1.6%	95%	Unknown	gradient_tape/sequential/max_pooling2d_1/MaxPool/MaxPoolGrad-0-TransposeNHWCToNCHW-LayoutOptimizer:Transpose	X	X
1.3%	96.4%	MaxPool	sequential/max_pooling2d_1/MaxPool	X	X
1.2%	97.6%	_Send	IteratorGetNext/_13	X	X
0.9%	98.5%	_XlaRun	cluster_1_1/xla_run	X	X
0.5%	99%	_XlaRun	cluster_2_1/xla_run	X	X

Recurrent Neural Networks (RNN)



Picture by: https://wiki.tum.de/display/lfdv/Video+Analysis

RNN for Video Analysis



Picture by: https://wiki.tum.de/display/lfdv/Video+Analysis

Deep Leaning with H2O - I

- purely supervised training protocol for regression and classification tasks
- fast and memory-efficient Java implementations based on columnar compression and fine- grain Map/Reduce
- multi-threaded and distributed parallel computation to be run on either a single node or a multi-node cluster
- fully automatic per-weight adaptive learning rate for fast convergence
- optional specification of learning rate, annealing and momentum options
- regularization options include L1, L2, dropout, Hogwild! and model averaging to prevent model overfitting
- grid search for hyperparameter optimization and model selection

Deep Leaning with H2O - II

- model checkpointing for reduced run times and model tuning
- automatic data pre- and post-processing (one-hot encoding and standardization) for categorical and numerical data
- automatic imputation of missing values
- automatic tuning of communication vs computation for best performance
- model export in plain java code for deployment in production environments
- export of weights and biases as H2O frames
- additional expert parameters for model tuning
- deep autoencoders for unsupervised feature learning and anomaly detection capabilities

H2O- DeepLear | pandas.DataFra | python - Creatir | Python String sp | Using Flow - H2 | Intellij Idea 201 | Reading a CSV | OrdinalModelPr | Deep Learning | The Definitive P

localhost:54321/flow/index.html#

convert 1/2

H2O FLOW Flow Cell Data Model Score Admin Help

DeepLearning_FaceRecognition

262ms

Model

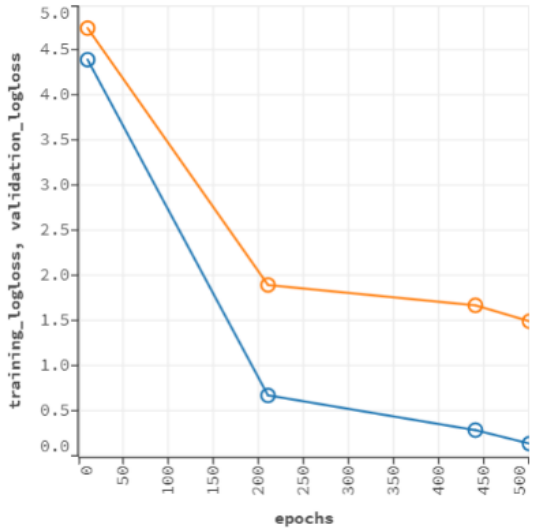
Model ID: deeplearning-699a1500-b9e4-44af-8e89-772f3b6628e3

Algorithm: Deep Learning

Actions: Refresh | Predict... | Download POJO | Download Model Deployment Package (MOJO) | Export | Inspect | Delete | Download Gen Model

▸ MODEL PARAMETERS

▾ SCORING HISTORY - LOGLOSS



▾ VARIABLE IMPORTANCES

OUTLINE FLOWS CLIPS HELP

Help

PACK

examples

- GBM_Example.flow
- DeepLearning_MNIST.flow
- GLM_Example.flow
- DRF_Example.flow
- K-Means_Example.flow
- Million_Songs.flow
- KDDCup2009_Churn.flow
- QuickStartVideos.flow
- Airlines_Delay.flow
- GBM_Airlines_Classification.flow
- GBM_GridSearch.flow
- RandomData_Benchmark_Small.flow
- GBM_TuningGuide.flow
- XGBoost_Example.flow

Ready

Connections: 0

deeplearning_699....java

Show all

Type here to search

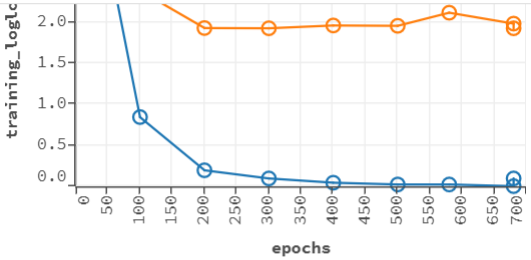
ENG 16:14 21.12.2020 г.

H2O- DeepLear | pandas.DataFra | python - Creatir | Python String sp | Using Flow - H2 | Intellij Idea 201 | Reading a CSV | OrdinalModelPr | Deep Learning | The Definitive P

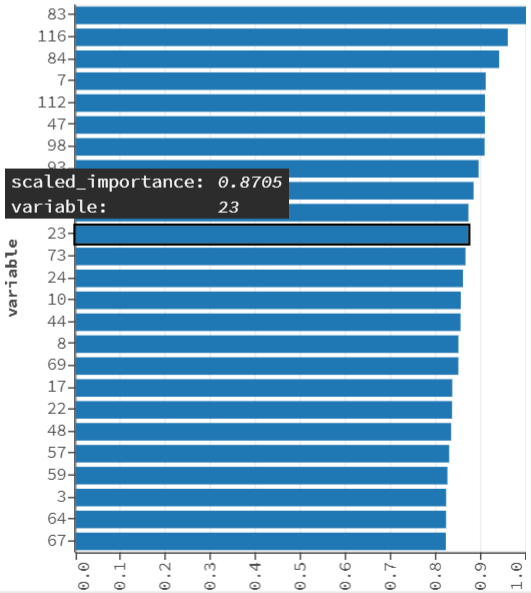
localhost:54321/flow/index.html#

H2O FLOW Flow Cell Data Model Score Admin Help

DeepLearning_FaceRecognition



VARIABLE IMPORTANCES



OUTLINE FLOWS CLIPS HELP

Help

PACK

examples

- GBM_Example.flow
- DeepLearning_MNIST.flow
- GLM_Example.flow
- DRF_Example.flow
- K-Means_Example.flow
- Million_Songs.flow
- KDDCup2009_Churn.flow
- QuickStartVideos.flow
- Airlines_Delay.flow
- GBM_Airlines_Classification.flow
- GBM_GridSearch.flow
- RandomData_Benchmark_Small.flow
- GBM_TuningGuide.flow
- XGBoost_Example.flow

Ready

Connections: 0

deeplearning_699....java | deeplearning_699....java | deeplearning_699....java | deeplearning_699....java

Show all

Type here to search

ENG 17:26 21.12.2020 г.

H2O- DeepLear | pandas.DataFra | python - Creatir | Python String sp | Using Flow - H2 | Intellij Idea 201 | Reading a CSV | OrdinalModelPr | Deep Learning | The Definitive P

localhost:54321/flow/index.html#

H2O FLOW Flow Cell Data Model Score Admin Help

DeepLearning_FaceRecognition

scaled_importance

TRAINING METRICS - CONFUSION MATRIX ROW LABELS: ACTUAL CLASS; COLUMN LABELS: PREDICTED CLASS

1	0	87	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0.0440	4 / 91	1.0
2	0	0	39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 / 39	1.0
3	0	0	0	82	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 / 82	0.85
4	0	0	0	4	97	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0.0490	5 / 102	0.99
5	0	0	0	0	0	56	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 / 56	0.98
6	0	0	0	0	0	0	92	0	0	0	0	0	0	0	0	0	6	3	0	0	0.0891	9 / 101	1.0
7	0	0	0	0	1	0	0	34	0	0	0	0	0	0	0	0	0	0	0	0	0.0286	1 / 35	1.0
8	1	0	0	0	0	0	0	0	75	0	0	3	0	0	0	0	0	0	0	0	0.0506	4 / 79	0.99
9	1	0	0	0	0	0	0	0	0	100	0	1	0	0	0	0	0	0	0	0	0.0196	2 / 102	0.91
10	0	0	0	0	0	0	0	0	0	7	72	1	0	0	0	0	0	0	0	0	0.1000	8 / 80	1.0
11	0	0	0	0	0	0	0	0	0	0	0	101	0	0	0	0	0	0	0	0	0	0 / 101	0.89
12	1	0	0	2	0	0	0	0	0	0	0	5	87	0	0	0	0	0	0	0	0.0842	8 / 95	1.0
13	1	0	0	6	0	0	0	0	1	1	0	2	0	68	0	0	0	0	0	0	0.1392	11 / 79	1.0
14	1	0	0	0	0	0	0	0	0	0	0	0	0	0	36	0	0	0	0	0	0.0270	1 / 37	1.0
15	1	0	0	0	0	1	0	0	0	2	0	0	0	0	0	39	1	1	0	0	0.1333	6 / 45	1.0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	103	0	0	0	0	0 / 103	0.92
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	79	0	0	0.0125	1 / 80	0.84
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	7	88	0	0.0833	8 / 96	1.0
19	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	84	0.0233	2 / 86	1.0
Total	100	87	39	96	98	57	92	34	76	110	72	113	87	68	36	39	112	94	88	84	0.0442	70 / 1Â 582	
Recall	1.0	0.96	1.0	1.0	0.95	1.0	0.91	0.97	0.95	0.98	0.90	1.0	0.92	0.86	0.97	0.87	1.0	0.99	0.92	0.98			

OUTLINE FLOWS CLIPS HELP

Help

PACK

examples

- GBM_Example.flow
- DeepLearning_MNIST.flow
- GLM_Example.flow
- DRF_Example.flow
- K-Means_Example.flow
- Million_Songs.flow
- KDDCup2009_Churn.flow
- QuickStartVideos.flow
- Airlines_Delay.flow
- GBM_Airlines_Classification.flow
- GBM_GridSearch.flow
- RandomData_Benchmark_Small.flow
- GBM_TuningGuide.flow
- XGBoost_Example.flow

Ready

Connections: 0

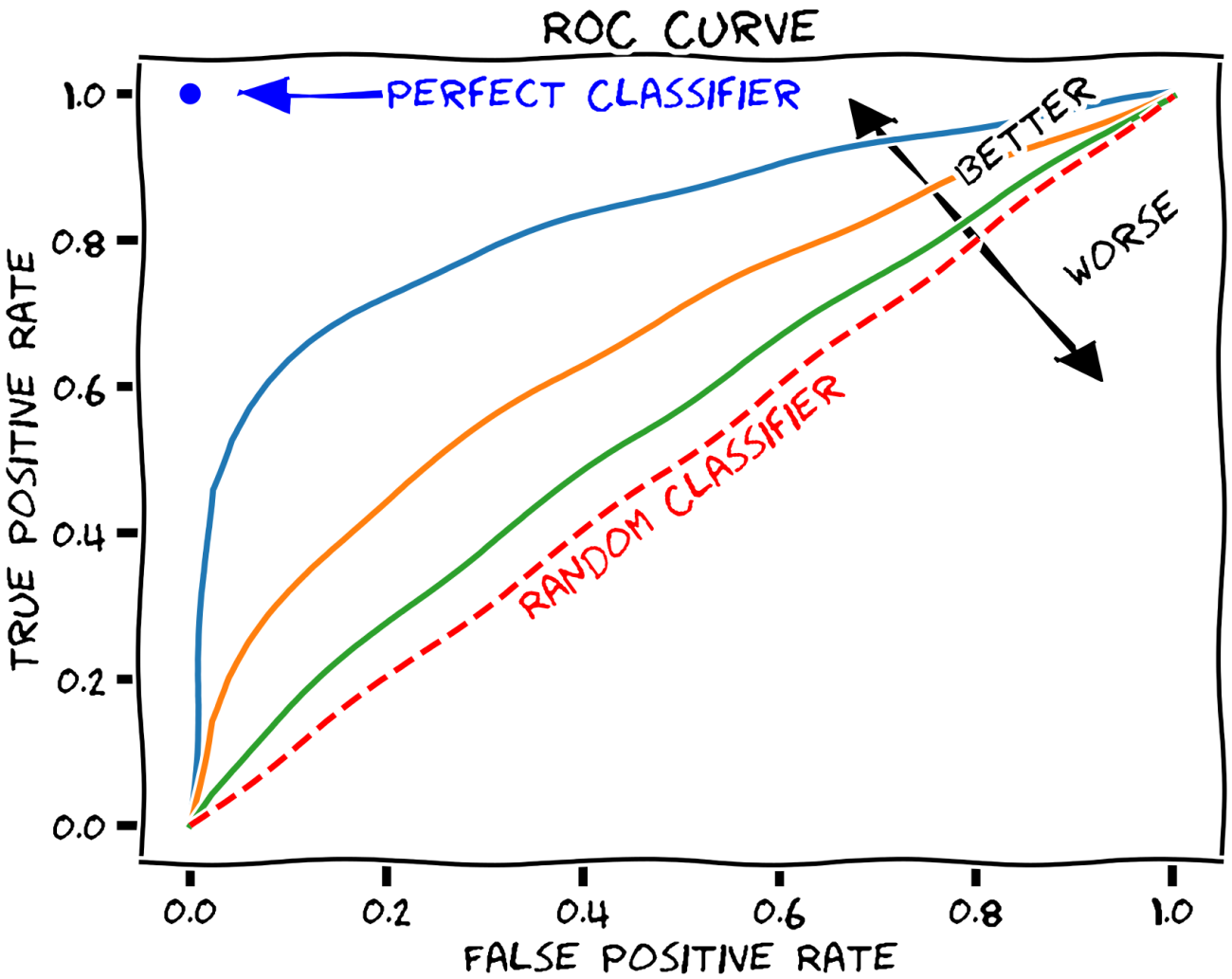
deeplearning_699....java

Show all

Type here to search

ENG 16:17 21.12.2020 г.

Receiver Operating Characteristic (ROC)



Receiver operating characteristic (ROC) curve - graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

Receiver Operating Characteristic (ROC)

- condition positive (P) / negative (N) - the number of real positive / negative cases in the data
- true positive (TP) - eqv. with hit, true negative (TN) - eqv. with correct rejection
- false positive (FP) - eqv. with false alarm, Type I error, false negative (FN) - eqv. with miss, Type II error

		True condition				
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$	F_1 score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$		

Thank's for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD