FUNDAMENTALS OF DEEP LEARNING FOR COMPUTER VISION

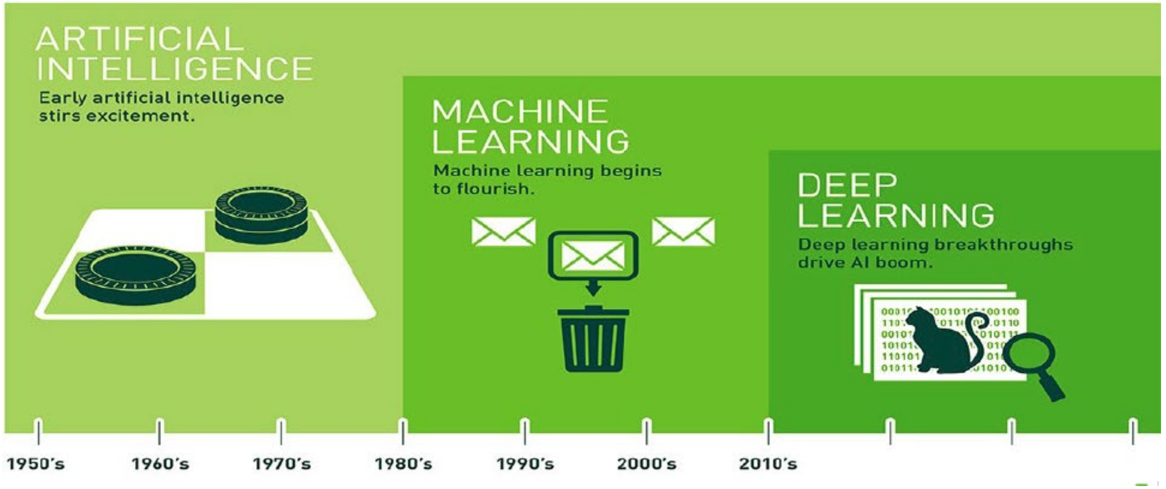
By:

Saifeddine Barkia

Mohamed Aziz Tousli

About

- **Deep Learning ∈ Machine Learning ∈ Artificial Intelligence**
  - Supervised Learning: Organized data (classification algorithm)
  - Unsupervised Learning: Non organized data (Clustering algorithm)
- History:
  - 1958 - Perceptron: First version of NN
  - 1974 – Backpropagation
  - 1995 – SVM
  - 1998 – CNN
  - 2006 – Blotzmann Machine



- ImageNet: Competition of classification
- The main limitation of computer science that deep learning removes
  → The need to write explicit instructions correct

Deep Learning

- Artificial neuron: Perceptron ⇔ Biological neuron
- Result in neuron = Activation (Input * Weight + Bias)
  →Good for linear problems
  →Multi-layer neural network → Good for non linear problems
- Learn something ⇔ Learn the parameters ⇔ Reduce the error
- How DL works?
  - Before, we used the give the characterization in input (hand-crafted features). Now, the NN itself learns automatically the necessary characterization: vertical lines, small circles…

Machine Learning		Deep Learning
Human	Neural Network	Neural Network
Feature extraction	Classification	Feature extraction + Classification

Course Architecture

FRAMEWORK

We've been working in a framework called Caffe.

Each framework requires a different way (syntax) of describing architectures and hyperparameters.

Other frameworks include TensorFlow, MXNet, etc.

NETWORK

We've been working with a network called AlexNet.

Each network can be described and trained using ANY framework.

Different networks learn differently: different training rates, methods, etc. Think different learners.

TOOL - UI

We've been working with a UI called DIGITS

The community works to make model building and deployment easier.

Other tools include Keras, Tensorboard, or APIs with common programming languages.

Task 1: Train a Model

- In *image classification*, we train a neural network to separate images into *classes.*
- **Small data set** → Instead of "learning", the network will "*memorize*" the images
  - Overfitting*:* It is only effective on the exact images that it was exposed to during training
- DIGITS: Platform where we select a *neural network* and the *data* it uses to learn

  →It manages the training process to test the **model** *(trained neural network)*

1. New model → Images → Classification
2. Select data set and training **epochs**

   1 Epoch : 1 Trip through the data
3. Choose **AlexNet**

   AlexNet : Neural network for image classification

   PS:: AlexNet is intended to be used with 256X256 (color) images
4. Clone job and change the number of epochs (Optional)
5. Test a single image by doing choosing the Image Path

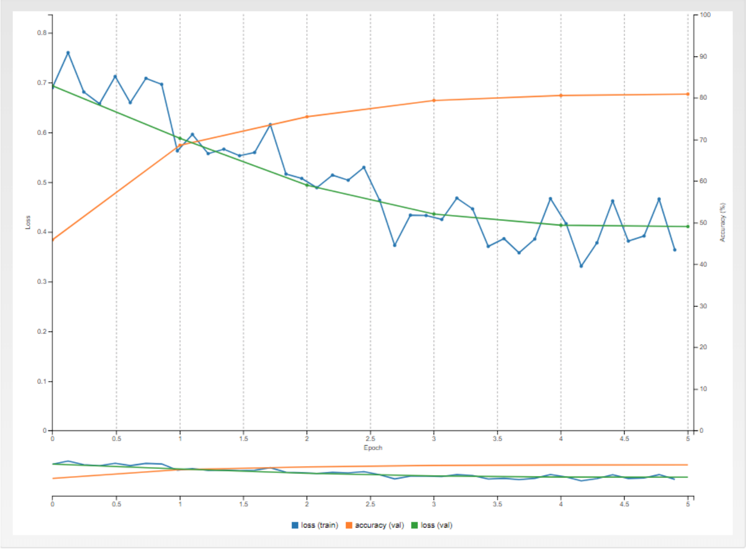
Task 2: New Data as a Goal

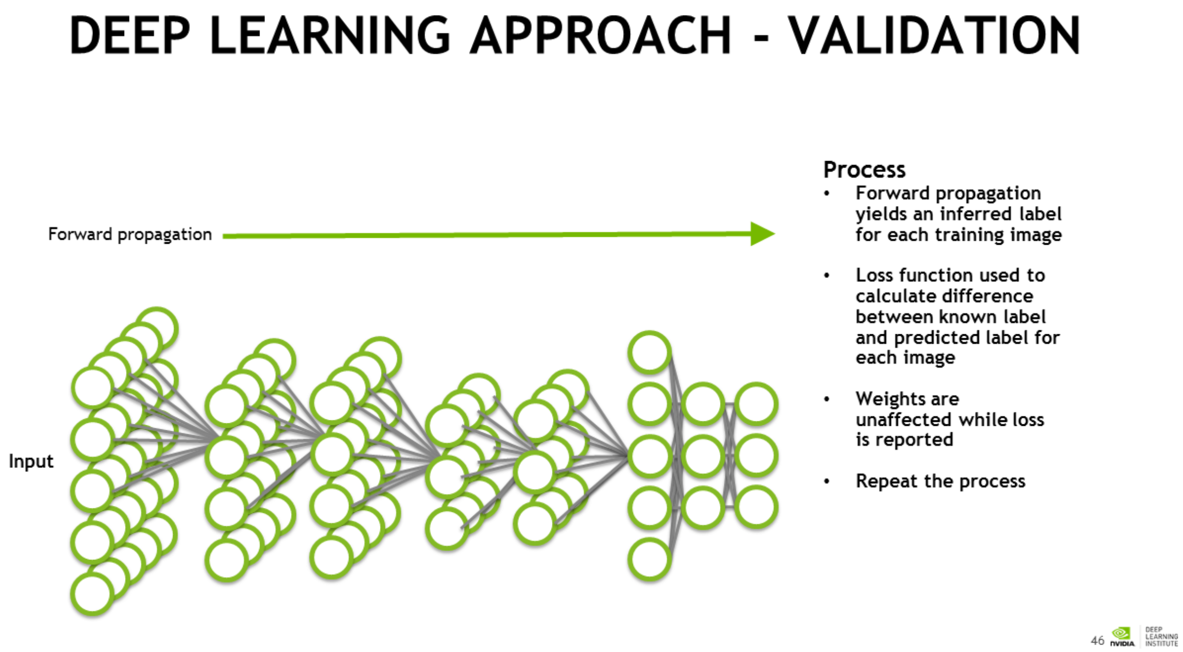
- Big ban in Machine Learning = **Deep Neural Networks** + **GPU** + **Big data**
- DIGITS: Standardize images to the same size to match what the network expects
  1. New dataset → Images → Classification
  2. Give directory of your dataset in Training Images

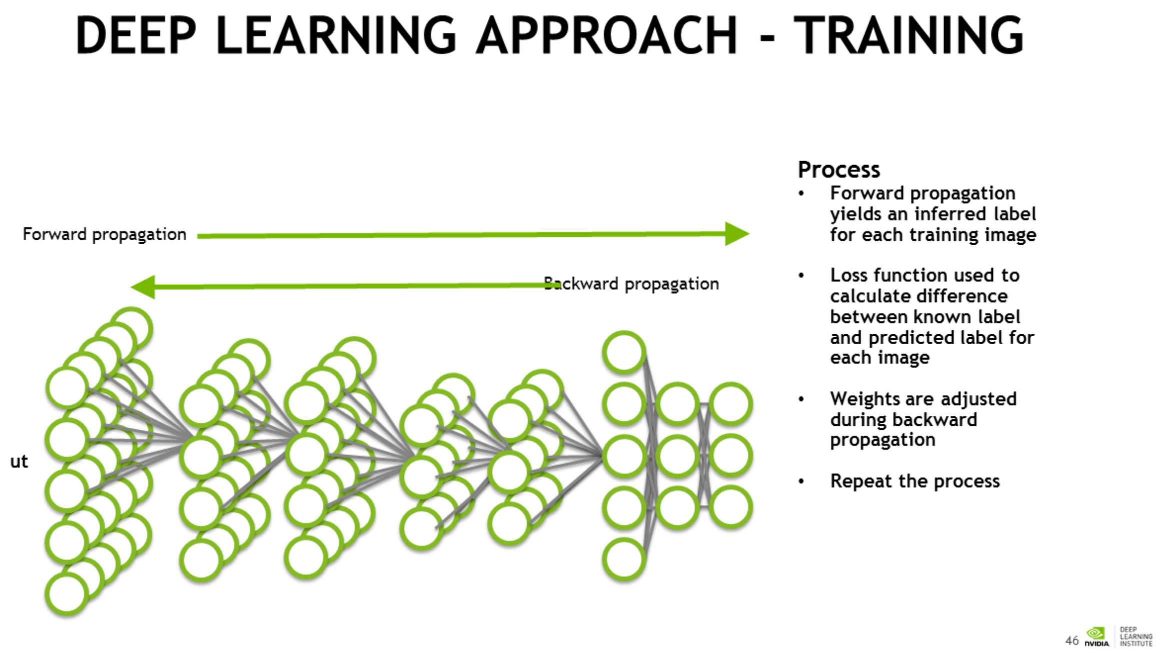
     PS: n folders → n classes
  3. Adjust the % for validation and % for testing
  4. Train the model just like in Task 1
  5. Test a single image by doing choosing the Image Path

     Inference: he process of making decisions based on what was learned

     → The model can now classify **unlabeled** images.



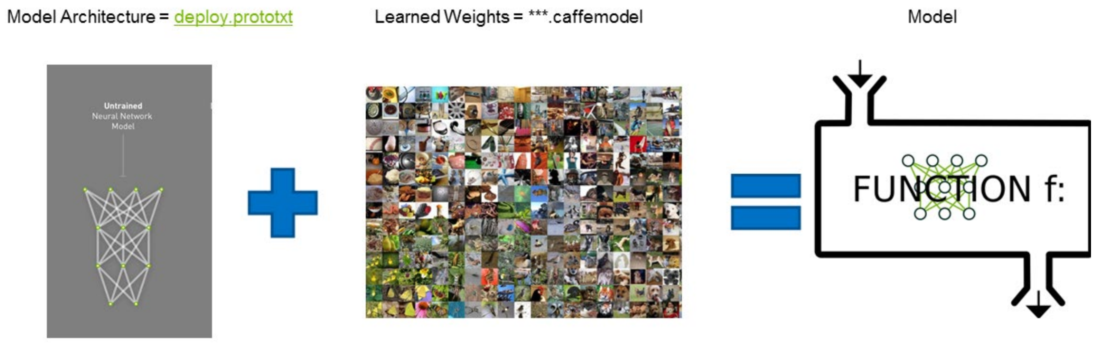




- <u>Forward propagation</u>: Matrix calculations to calculate the different weights
- <u>Backward propagation</u>: Derivative calculations to adjust the error

Task 3: Deployment

- <u>Deployment</u>: Put a trained model in **application** to solve problems
  → Deep learning + Traditional programming



- Deep learning workflow:
  1. Training
  2. Deployment

- <u>Python – Import model</u>:

```
Import caffe #Import Caffe framework
MODEL_JOB_DIR = 'jobDirectoryOfTheModel' #Open the model in DIGITS and get the directory
ARCHITECTURE = MODEL_JOB_DIR + '/' + 'deploy.prototxt' #Text file describing the network model
WEIGHTS = MODEL_JOB_DIR + '/' + 'snapshot_iter_n.caffemodel' #Binary file containing the weights at last iteration
    !ls MODEL_JOB_DIR #Get the full name of snapshot_iter_n
import caffe; caffe.set_mode_gpu(); #Import the framework and use GPU for parallel processing
net = caffe.Classifier(ARCHITECTURE, WEIGHTS, channel_swap =(2, 1, 0){RGB→BGR},
raw_scale=255{maxPixelValue}) #Initialize the Caffe model using the model trained in DIGITS
```

- Python – Import dataset & Preprocessing:

  DATA_JOB_DIR = ‘jobDirectoryOfTheDataset’ #Open the dataset in DIGITS and get the directory

  input_image = caffe.io.load_image(‘imageDirectory’) #Import an image

  import cv2; input_image=cv2.resize(input_image, (256, 256), 0,0); #Resize the image

  mean_image = caffe.io.load_image(DATA_JOB_DIR+'/mean.jpg’) #Prepare the mean image

  ready_image = input_image-mean_image #Normalize the image

- Python – Predict & Postprocessing:

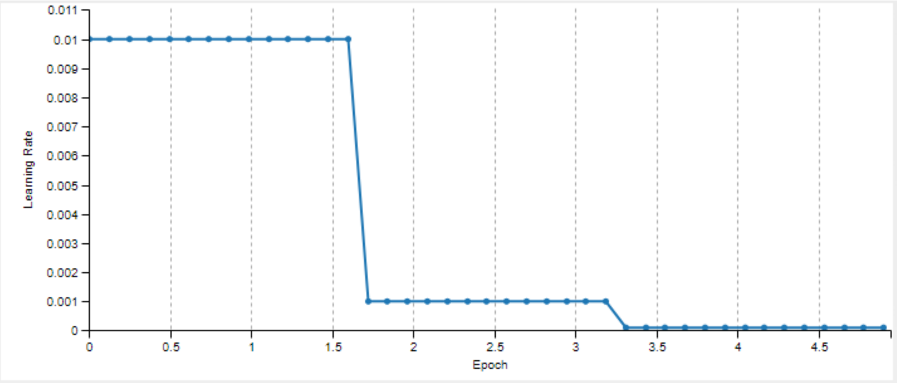
  prediction = net.predict([ready_image]) $\#Prediction = [[Class1, \dots, ClassN]]$

  prediction.argmax() #Return the index of the maximum value

  Initially, we predict randomly. Thanks to a lot of iterations, and a lot of examples, we will be able to solve the problem. We adjust the weights moreover, thus the decision.

Task 4: Improving Performance

- Study more: Run more epochs
- DIGITS:



1. Choose your model
2. Click on "Make pretrained model"

→ Make a copy of the trained model from the last starting point

3. Create a model just like in Task 1
4. Choose the number of epochs

→ Increasing the number of epochs often increases performance, but it can result in **overfitting**

5. Choose a fixed learning rate (the one attained by the pretrained model)

Learning rate: A hyper parameter for the rate at which each **weight** changes during training

Learning rate decay: Start with a big learning rate (to go fast), and when we become close to the minimum, reduce it

→ The learning rate decreases throughout the training session because the network is getting closer to its ideal solution

6. Select the pretrained model that you just created instead of the standard networks (i.e. AlexNet)

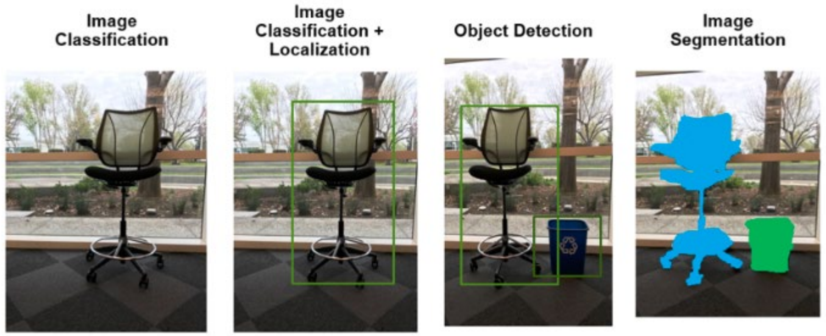
Categories of Performance

Requirement	Challenges
High Throughput	**Unable to processing high-volume, high-velocity data** ➢ Impact: Increased cost ($, time) per inference
Low Response Time	**Applications don't deliver real-time results** ➢ Impact: Negatively affects user experience (voice recognition, personalized recommendations, real-time object detection)
Power and Memory Efficiency	**Inefficient applications** ➢ Impact: Increased cost (running and cooling), makes deployment infeasible
Deployment-Grade Solution	**Research frameworks not designed for production** ➢ Impact: Framework overhead and dependencies increases time to solution and affects productivity

Influence on Performance

- **Data** - A large and diverse enough dataset to represent the environment where our model should work. Data curation is an art form in itself.
- **Hyperparameters** - Making changes to options like learning rate are like changing your training "style." Currently, finding the right hyperparameters is a manual process learned through experimentation. As you build intuition about what types of jobs respond well to what hyperparameters, your performance will increase.
- **Training time** - More epochs improve performance to a point. At some point, too much training will result in overfitting (humans are guilty of this too), so this can not be the only intervention you apply.
- **Network architecture** - We'll begin to experiment with network architecture in the next section. This is listed as the last intervention to push back against a false myth that to engage in solving problems with deep learning, people need mastery of network architecture. This field is fascinating and powerful, and improving your skills is a study in math.

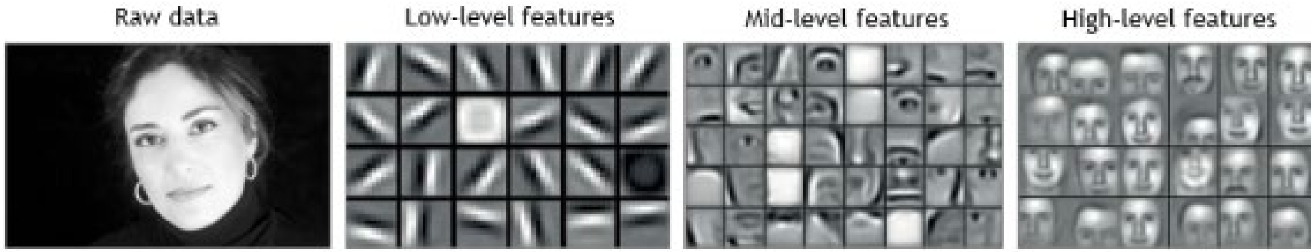
Task 5: Object Detection (1)



Workflow	Input	Output
Image Classification	Raw Pixel Values	A vector where each index corresponds with the likelihood or the image of belonging to each class
Object Detection	Raw Pixel Values	A vector with (X,Y) pairings for the top-left and bottom-right corner of each object present in the image
Image Segmentation	Raw Pixel Values	A overlay of the image for each class being segmented, where each value is the likelihood of that pixel belonging to each class
Text Generation	A unique vector for each 'token' (word, letter, etc.)	A vector representing the most likely next 'token'
Image Rendering	Raw Pixel Values of a grainy Image	Raw pixel values of a clean image

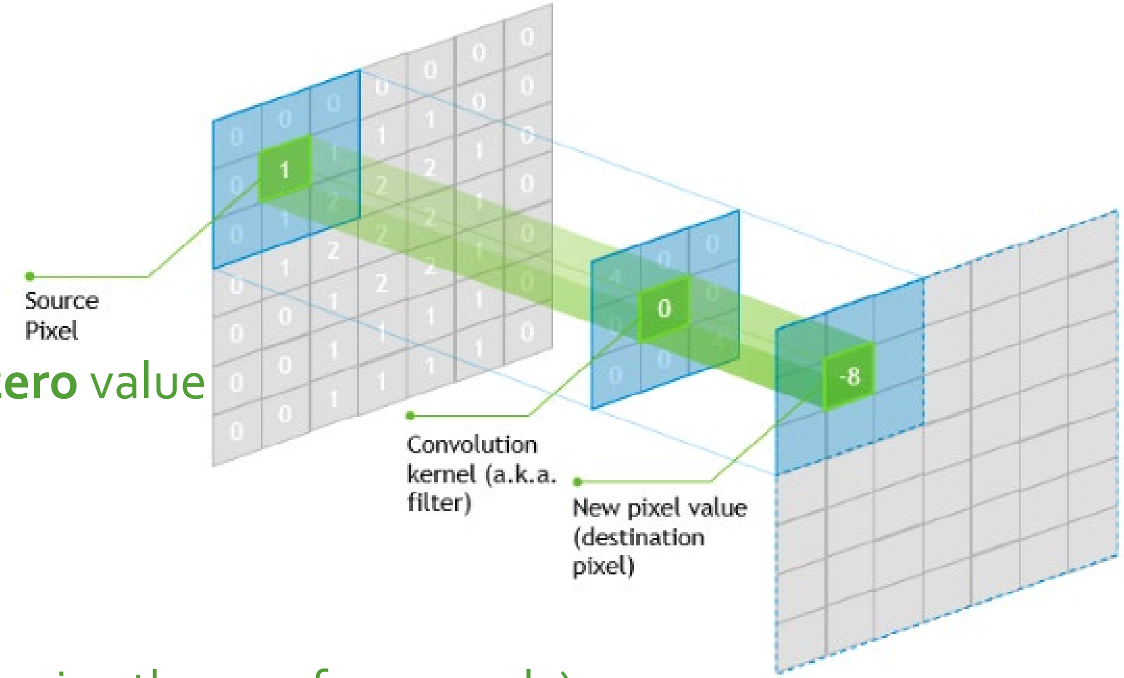
- Approach 1: **Sliding Window**
  Build a classifier and slide a window that runs it on each segment
  Slide of the window = Slide trained in the neural network
  → Slow and needs human supervision
- Approach 2: **Modifying Network Architecture**
  Change the number of layers and hidden units i.e. the structure of the network
  FC=Fully Connected=Matrix Multiplication=Size Constraint
  → Impact capability and performance
  Layers: Mathematical operations on tensors
- Approach 3: **End-to-End Solution**
  Train images with bounding box annotations → Modelzoo

CNN



- CNN = Multi-layer neural network with:
  - Local connections: Reduce the number of parameters
  - Shared weights
- In CNN, we can have a matrix as an input or output
- Number of filters to apply is an hyperparameter
- Stride: Pace
- Padding: Add exterior lines and columns
- Zero-padding: Add exterior lines and columns with **zero** value
- CNN steps:
  1. Input image
  2. Convolution: Learned)
  3. Non-linearity: Accelerate the process
  4. Spatial pooling: Compress the information (by choosing the max for example)
  5. Normalization
  6. Feature maps

Filter in CNN = Weight
Old pixel value * Filter = New pixel value



Task 5: Object Detection (2)

- <u>Approach 1</u>: Using deployment (**Python**)
  → Combine deep learning and traditional programming

```
input_image = caffe.io.load_image(imageToAnalyse) #Load the image to analyse
rows = input_image.shape[0]/256; cols = input_image.shape[1]/256; #Number of 256x256 grid squares
detections = np.zeros((rows,cols)) #Initialization of detections matrix
for i in range(0,rows):
   for j in range(0,cols):
      grid_square = input_image[i*256:(i+1)*256,j*256:(j+1)*256] #Generate the grid square
      grid_square -= mean_image #Substract the mean image
      prediction = net.predict([grid_square])  #Make the prediction
      detections[i,j] = prediction[0].argmax() #Update the detections matrix
```

- <u>Approach 2</u>: Rebuilding from an existing neural network (**DIGITS**)
    1. Select your model
    2. Select “Clone Job” to copy all of the settings to a new network
    3. Select “Customize” for AlexNet
    4. Select "Visualize" to visualize the AlexNet network
    5. Modify the network “Brain surgery”

       <u>Rule 1</u>: Data must be able to flow

       → Keep everything connected so the data can flow from input to output

       <u>Rule 2</u>: The math matters

       → Be careful from specific matrix sizes in traditional matrix multiplication

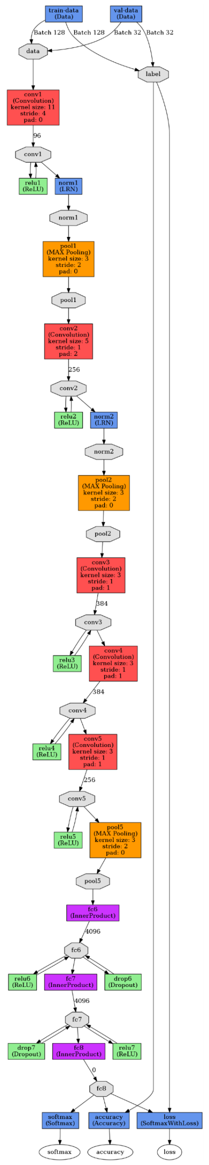
       Fully connected layer: Traditional matrix multiplication

       Fully convolutional layer: “Filter" function that moves over an input matrix

- <u>Selective search</u>: Reduce the number of windows passed on the NN
    1. CNN extracts the characterization in a vector
    2. Loss that learns to extract the coordinates (regression)
    3. Loss that classifies (softmax)

  ➔ Two different loss functions: **parallel work**

```
# AlexNet
name: "AlexNet"
layer {
  name: "train-data"
  type: "Data"
  top: "data"
  top: "label"
  transform_param {
    mirror: true
    crop_size: 227
  }
  data_param {
    batch_size: 128
  }
  include { stage: "train" }
}
layer {
  name: "val-data"
  type: "Data"
  top: "data"
  top: "label"
  transform_param {
    crop_size: 227
  }
  data_param {
    batch_size: 32
  }
  include { stage: "val" }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
```



- Approach 3: DetectNet (Fully Convolutional Network) (**DIGITS**)
  - Difference in Data:
    1. New dataset → Images → Object Detection
    2. Fill in the blanks
  - Difference in Networks:
    1. New model → Images → Object Detection
    2. Select the dataset you just loaded
    3. Select "Custom Network" and fill it
  - Difference in Compute:
    1. Fill "Pretrained Model(s)" with a pretrained model

    mAP: mean Average Precision
- R-CNN: Region CNN
- Faster R-CNN: R-CNN + Selective search

Power = Deep NN + GPU + Big Data