USING MACHINE LEARNING TECHNIQUES FOR DATA QUALITY MONITORING AT CMS EXPERIMENT

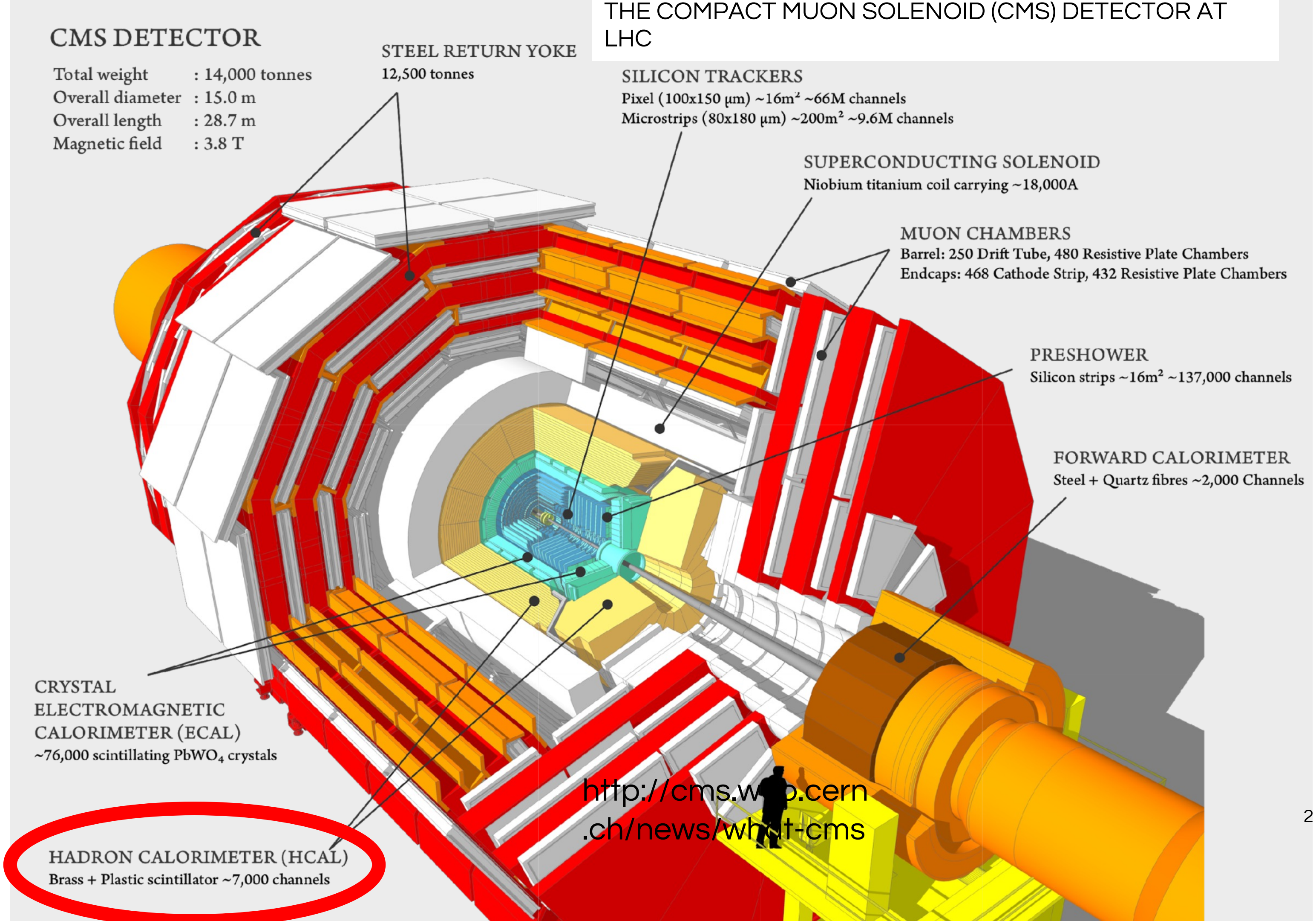
GUILLERMO A. FIDALGO RODRÍGUEZ

PHYSICS DEPARTMENT

UNIVERSITY OF PUERTO RICO MAYAGÜEZ

PRISM/JTM 2019

THE COMPACT MUON SOLENOID (CMS) DETECTOR AT LHC



http://cms.web.cern.ch/news/what-cms

OBJECTIVES

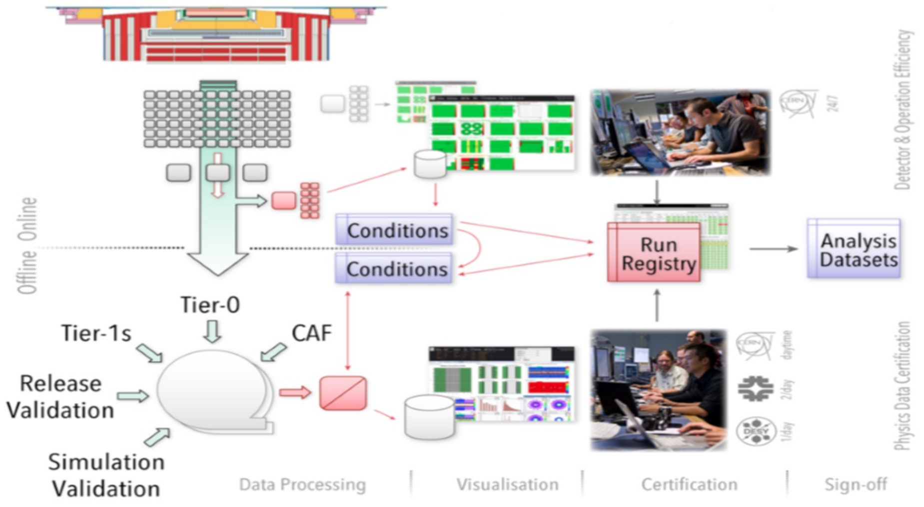
- Apply recent progress in Machine Learning techniques regarding automation of DQM scrutiny for HCAL
  - To focus on the Online DQM.
  - To compare the performance of different ML algorithms.
  - To compare fully supervised vs semi-supervised approach.
- Impact the current workflow, make it more efficient and can guarantee that the data is useful for physics analysis.

CHALLENGE

- Make sure detector behaves well to perform sensible data analysis.
- Reduce man power to discriminate good and bad data, spot problems, save time examining hundreds of histograms.
  - By building intelligence to analyze data, raise alarms, quick feedback.

- Implementing the best architecture for neural networks
  - Underfitting - Too simple and not able to learn
  - Overfitting - Too complex and learns very specific and/or unnecessary features
- There is no rule of thumb
  - Many, many, many…… possible combinations.

WHAT IS DATA QUALITY MONITORING (DQM)?



- Two kinds of workflows:
- Online DQM
  - Provides feedback of live data taking.
  - Alarms if something goes wrong.
- Offline DQM
  - After data taking
  - Responsible for bookkeeping and certifying the final data with fine time granularity.

TOOLS AND DATA PROCESSING

- Working env: python Jupyter notebook
- Keras (with Tensorflow as backend) and Scikit-learn
  - Creation of a model
  - Train and test its performance
- The input data consists of occupancy maps
  - one map for each luminosity section
  - Used 2017 good data and generate bad data artificially

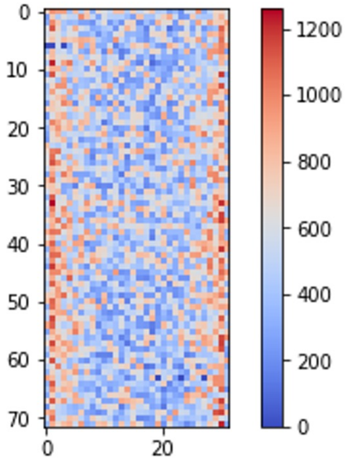
TensorFlow

Keras

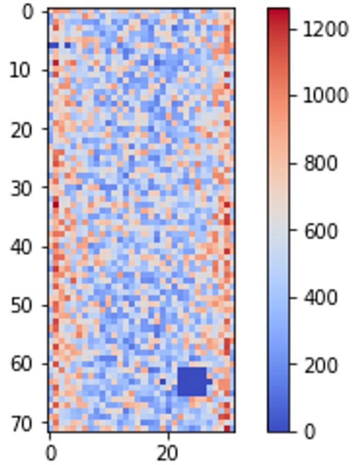
scikit learn

IMAGES AND READOUT CHANNELS USED AS INPUTS FOR THE ML ALGORITHM

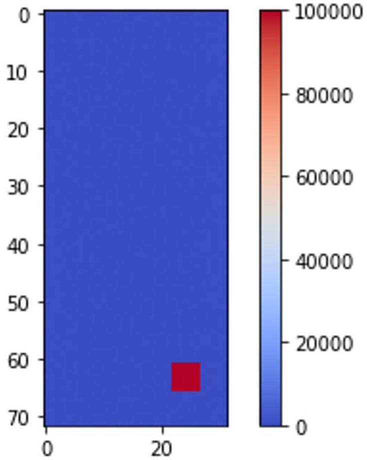
- Supervised and Semi-Supervised Learning
- 1x1 problematic region with random location (On SL model)
- 5x5 (readout channels) problematic region with random location (on SSL model)



Good

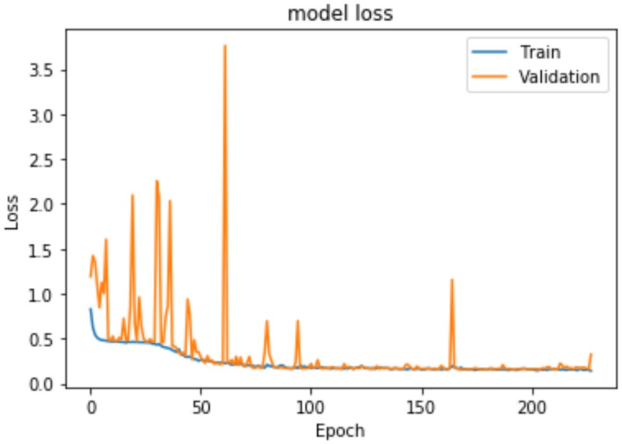


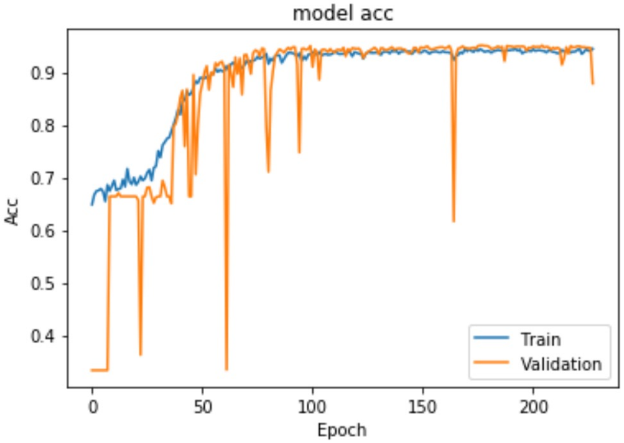
Dead



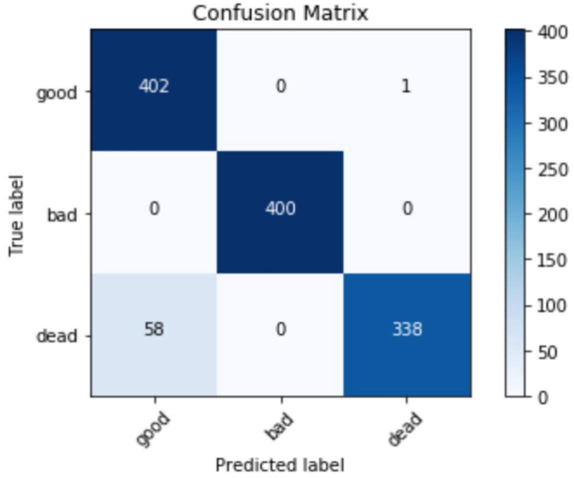
Hot

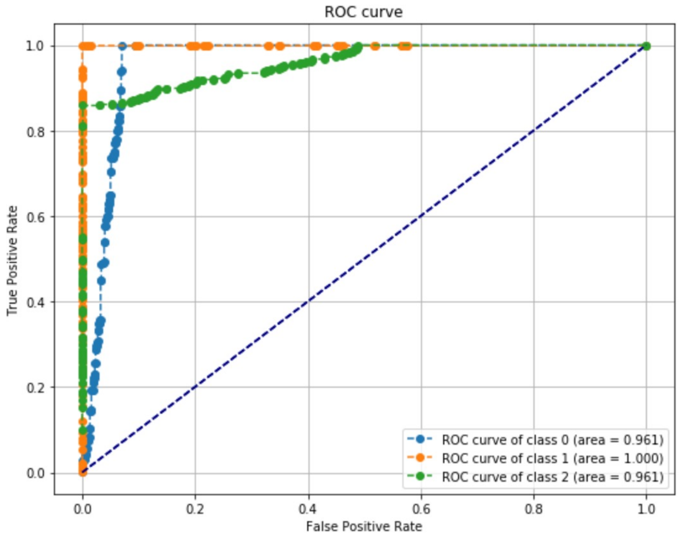
SUPERVISED LEARNING



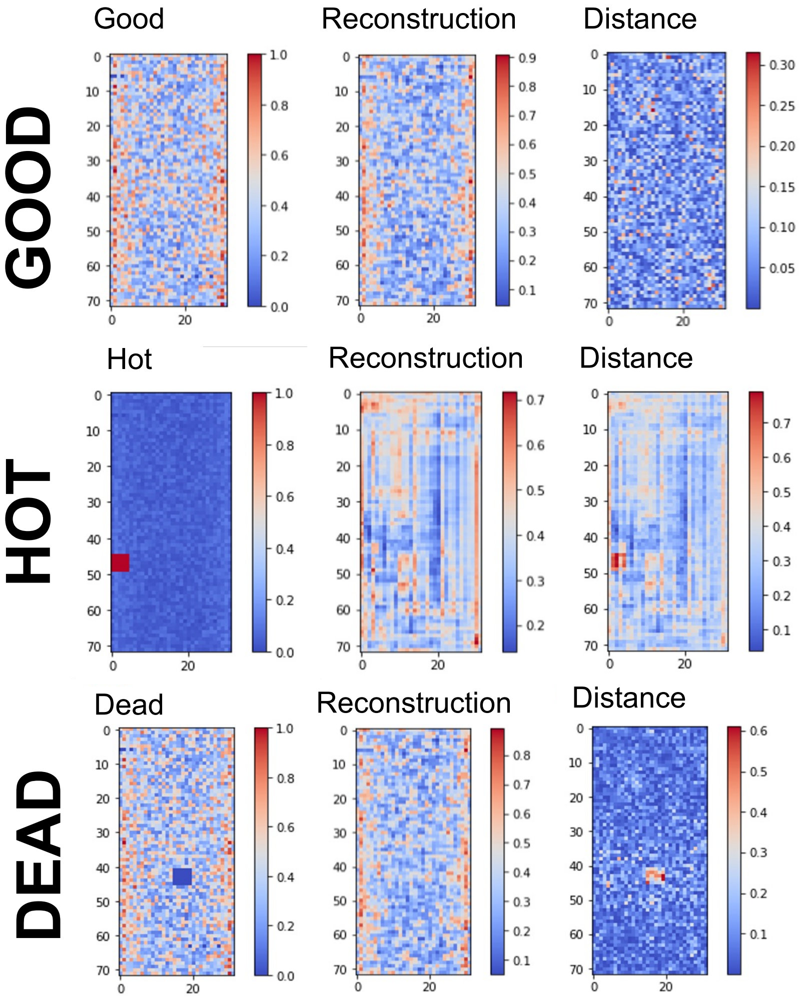


accuracy score: 0.9507923269 39



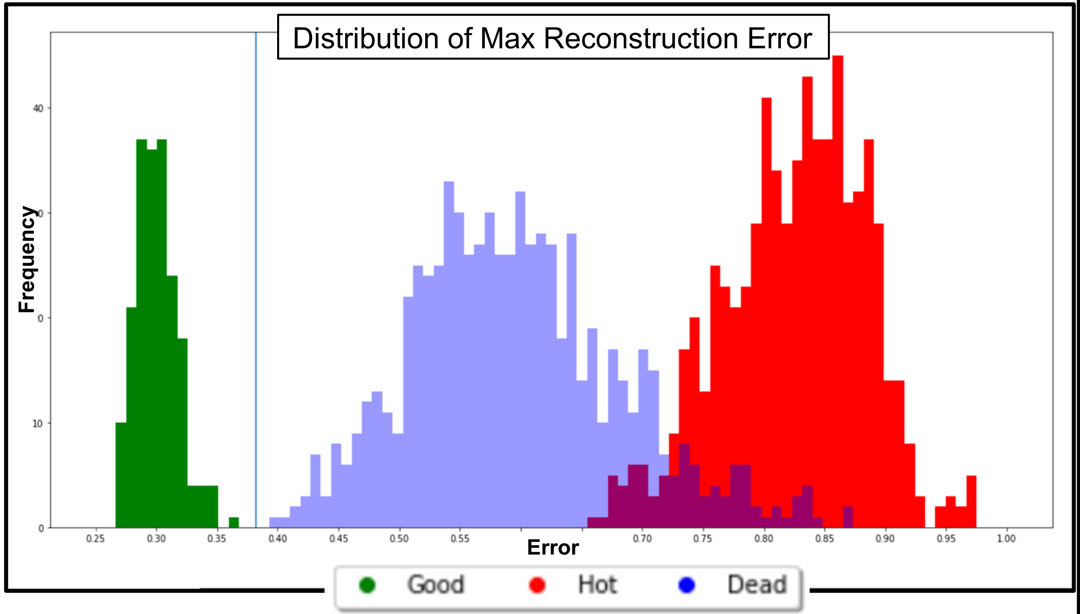


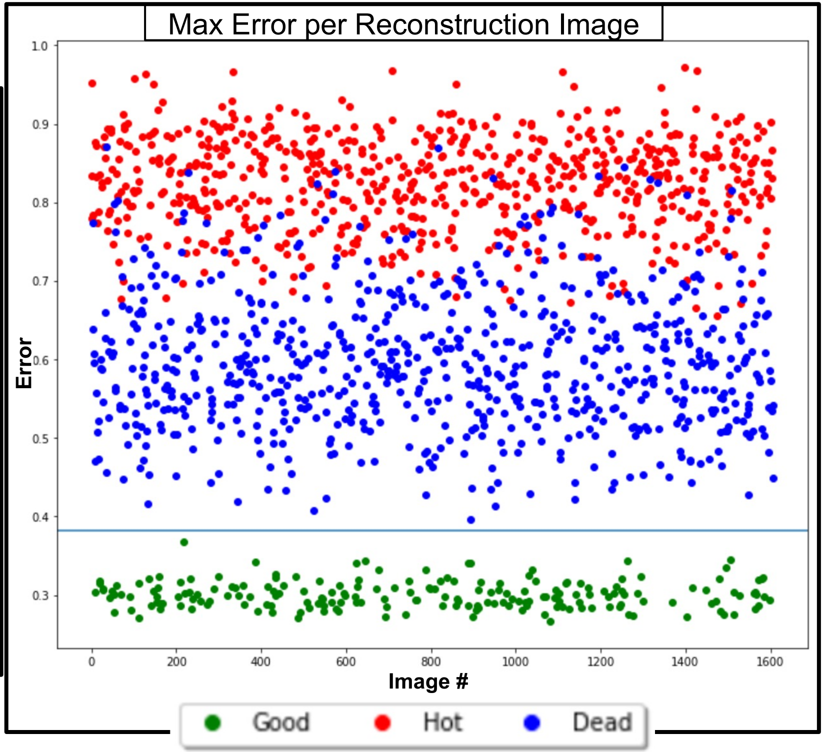
SEMI SUPERVISED LEARNING



1) Trained only on good images
2) Expected to see better reconstruction for good images and a much different reconstruction for bad images.
3) Use this as discriminating factor.
4) Bad images have 5x5 bad regions
   a) Hot
   b) Dead
5) Images have been normalized

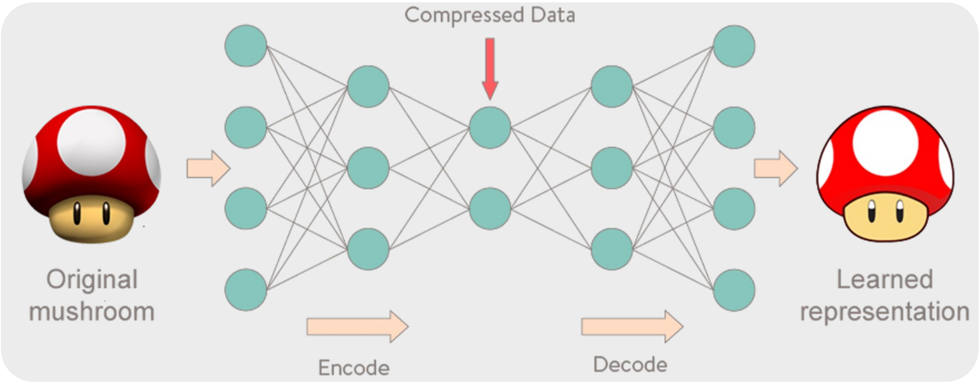
ERROR DISTRIBUTION PER IMAGE CLASS

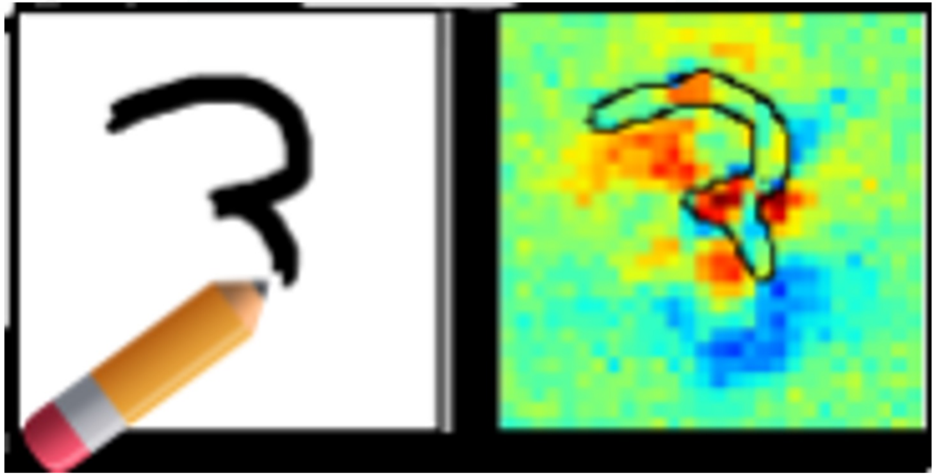




WHAT'S NEXT?

- Can it predict changes with temporal information?
- Can we make it work with something more realistic?
    - 1x1 bad region (channel)
    - Can it identify what values should be expected after each lumi-section?
    - Move from artificial bad data to real cases of bad data (in progress)





Acknowledgments

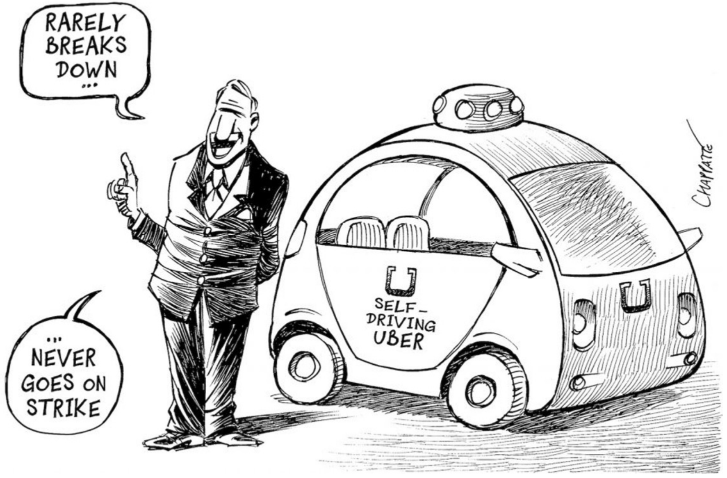
- The US State Dept.
- The University of Michigan
- CERN/CMS
- Texas Tech
- University of Puerto Rico Mayagüez

Thank You!

BACKUP

HOW TO AUTOMATE THE DATA QUALITY CHECKS? USE MACHINE LEARNING!

- It’s everywhere now!
  - A.I. Learning
  - Self-driving cars
  - How do Google/Facebook know what you want?
  - Face/Handwriting Recognition
- In our case everything is reduced to a classification problem
  - Anomaly Detection



Machine Learning libraries

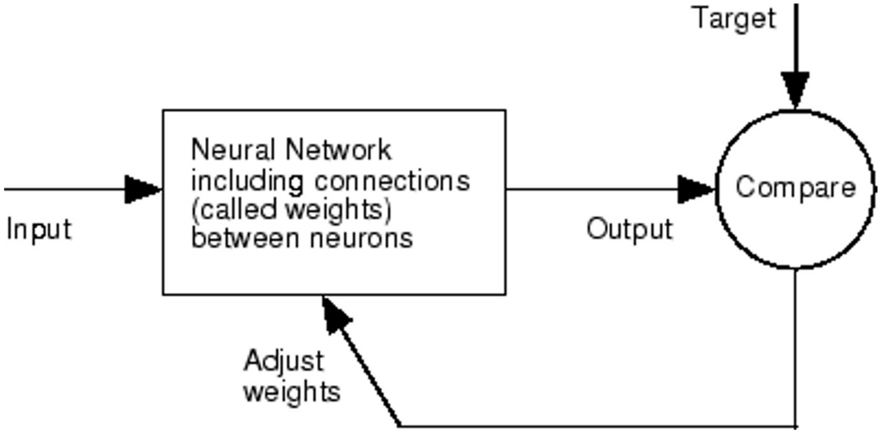
SCIKIT-LEARN

- Pre-defined models
  - Logistic Regression
  - MLP
- Not much control over the model's architecture
- Very useful for testing performance

KERAS

- Make your own models
  - A bit sophisticated
  - Only for making NN
- Neural Networks
  - Deep Convolutional
    - Best with image recognition

How to train a model



Gradient Descent

The "Learning" in Machine Learning.

Update the values of X (punish) it when it is wrong.

$$\boxed{X = X - \eta\nabla(X)}$$

X: weights or biases

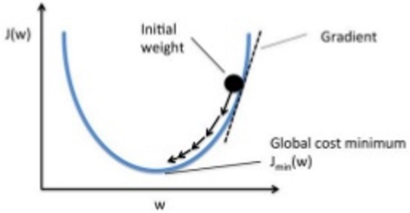
η: Learning Rate (typically 0.01 to 0.001)

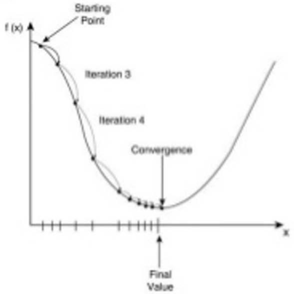
η :The rate at which our network learns. This can change over time with methods such as Adam, Adagrad etc. (hyperparameter)

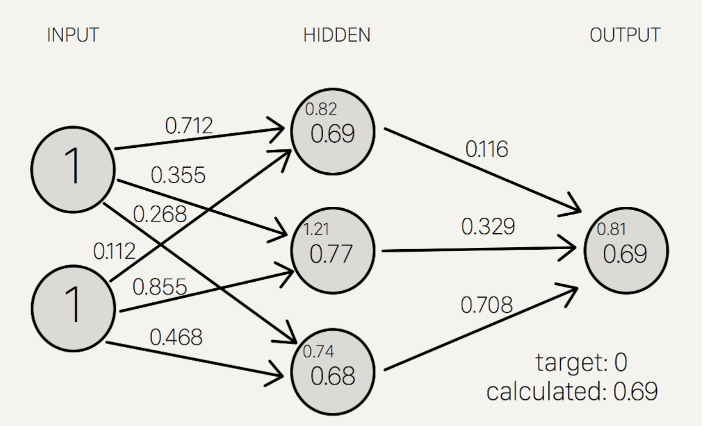
∇(x): Gradient of X

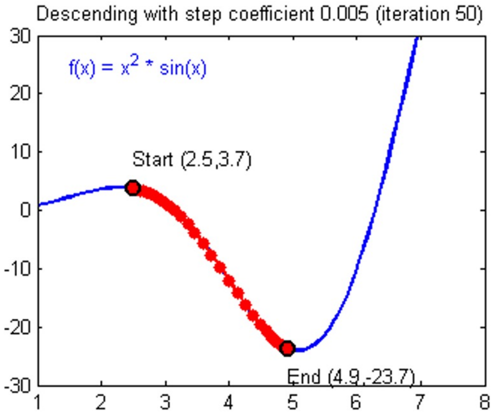
We seek to update the weights and biases by a value indicating how "off" they were from their target.

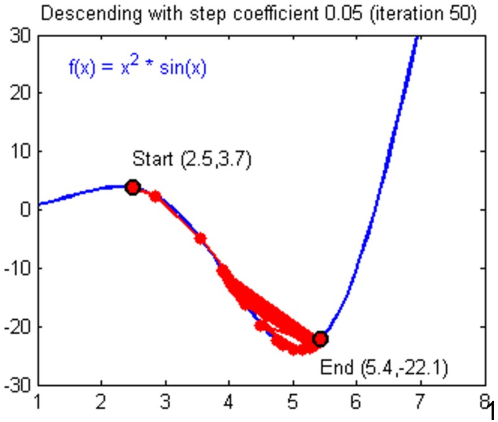
Gradients naturally have increasing slope, so we put a negative in front of it to go downwards



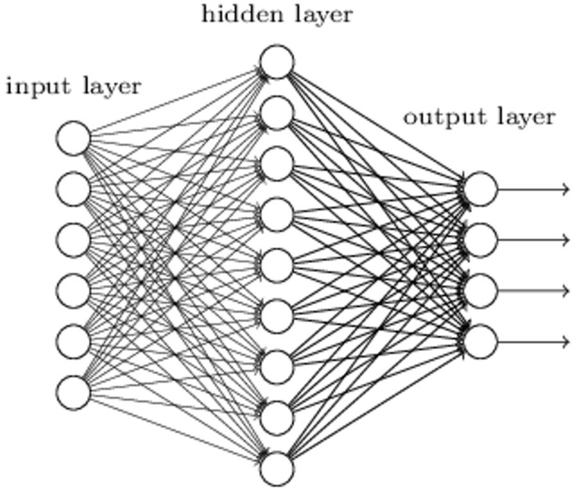




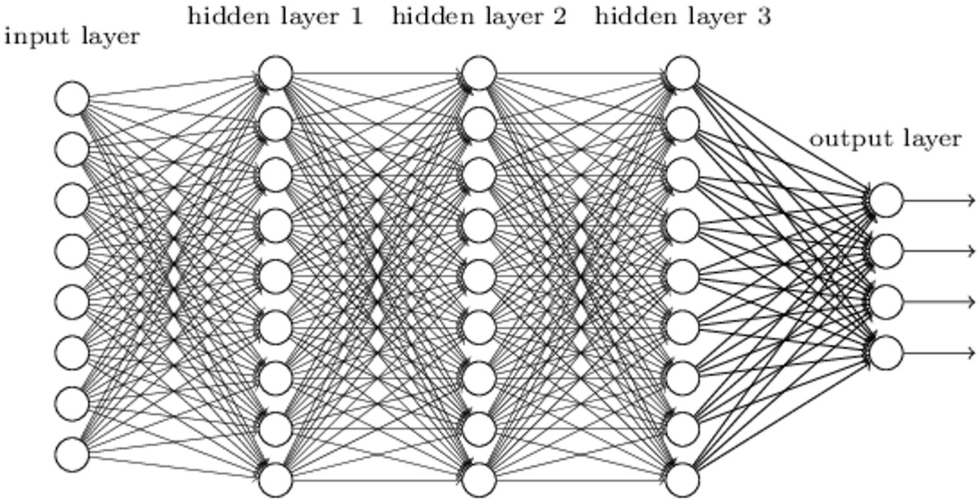




"Non-deep" feedforward neural network



Deep neural network



HOW A DEEP NEURAL NETWORK SEES

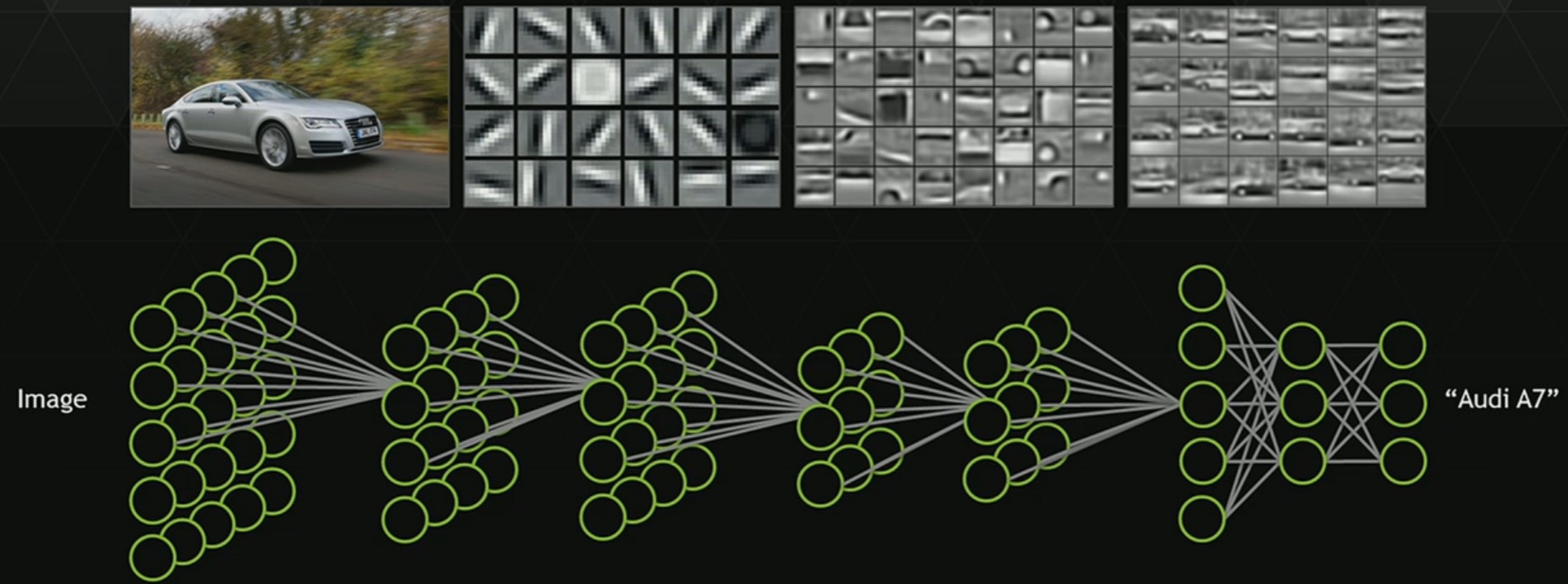
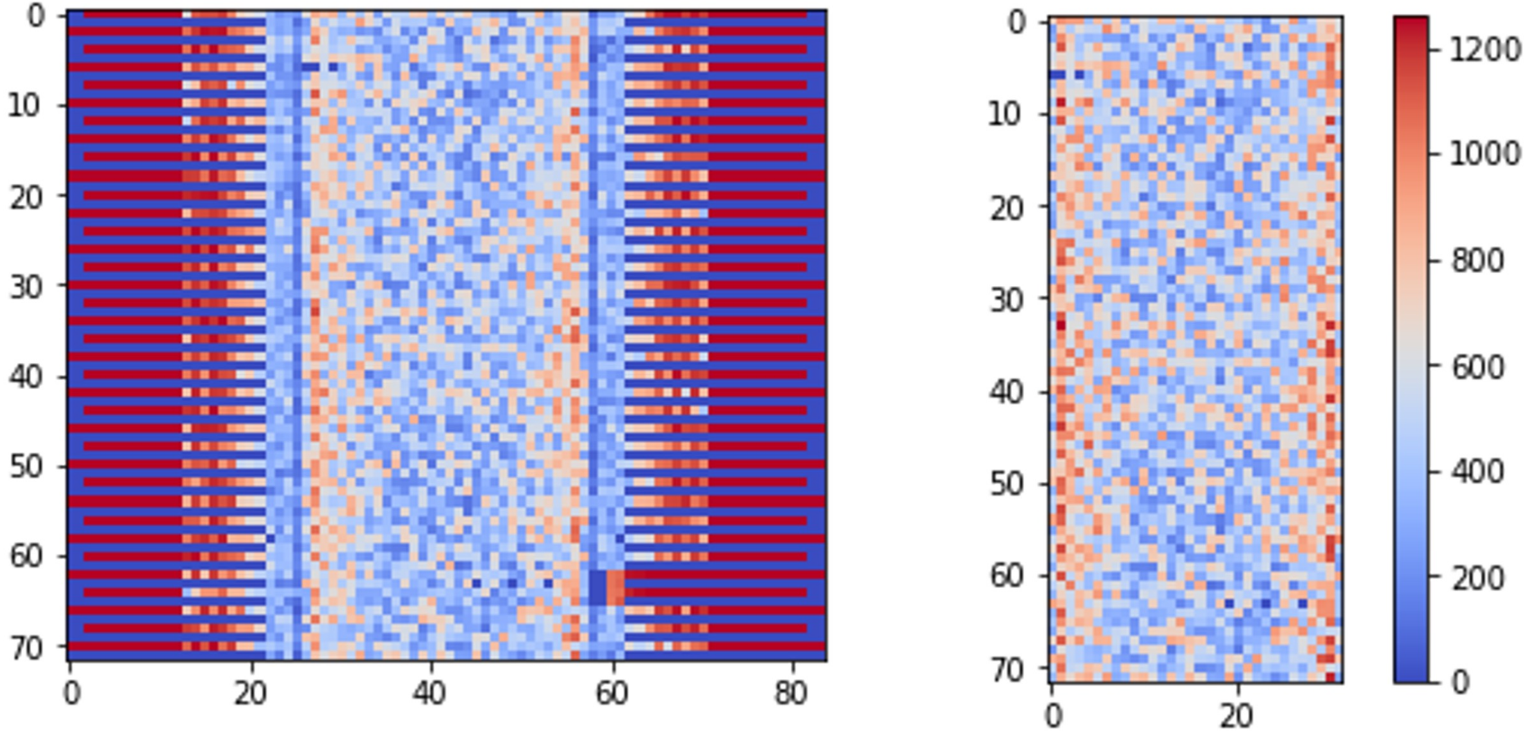


Image source: “Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks” ICML 2009 & Comm. ACM 2011
Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Ng.

SAMPLE IMAGES TO STUDY

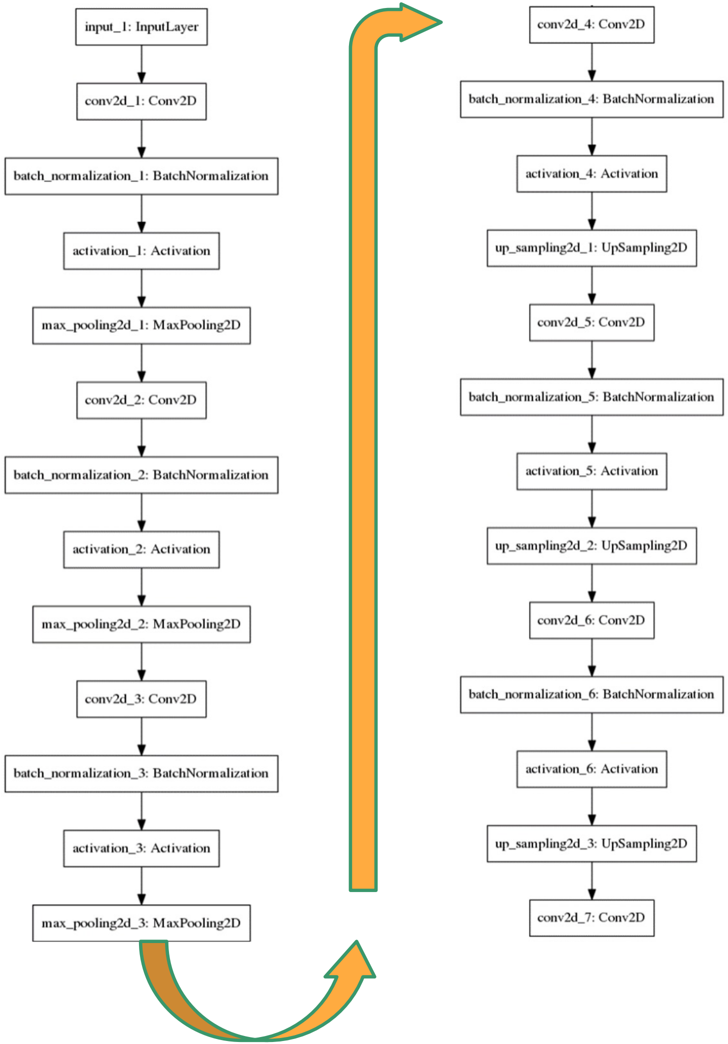


NEW ARCH.

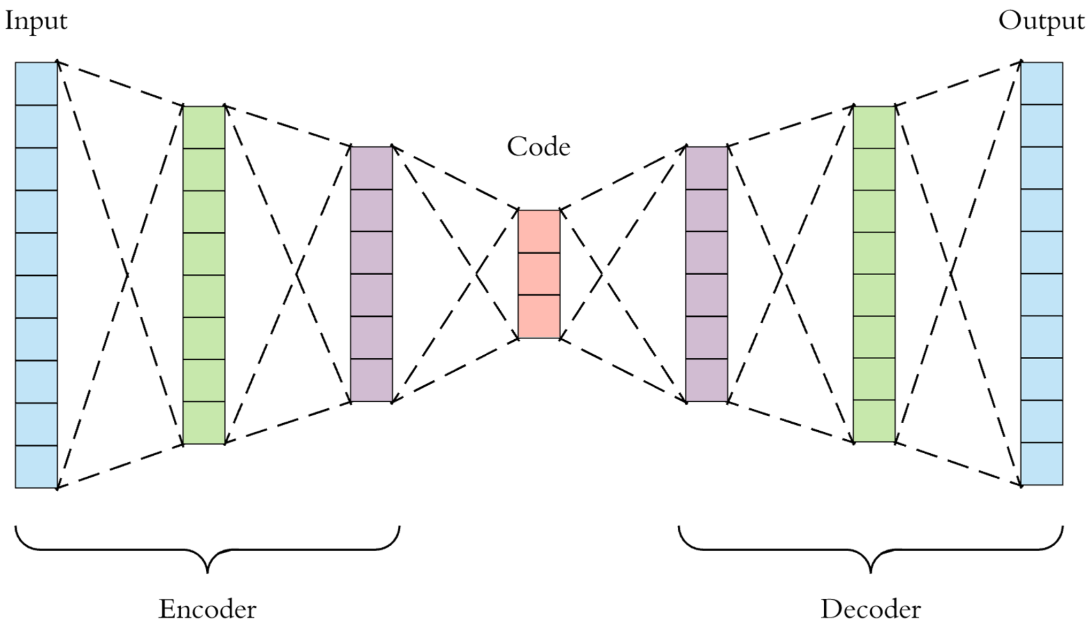
```
model = Sequential()

model.add(Conv2D(10, kernel_size=(2, 2), strides=(1, 1),input_shape=input_shape))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(8, kernel_size=(3, 3),strides=(1, 1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(8,kernel_size=(1,1)))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dropout(0.25))
model.add(Flatten())

model.add(Dense(8))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dense(3, activation='softmax'))


model.compile(loss='categorical_crossentropy',
              optimizer='adam',#Adam(lr=1e-3),
              metrics=['accuracy'])
```

ARCHITECTURE

```
input_img = Input(shape=(input_shape))  # adapt this if using `channels_first

x = Conv2D(86, (3, 3), padding='same')(input_img)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(64, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)

# at this point the representation is (4, 4, 8) i.e. 128-dimensional

x = Conv2D(32, (3, 3), padding='same')(encoded)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(64, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(86, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adadelta', loss='mse')
```



Auto-Encoder ARCHITECTURES



- The bottleneck structures work using dimensionality reduction.
  - We are interested in seeing the features that are learned at the bottleneck stage of the AE after a successful reconstruction.
- We can use the reconstruction loss as a discriminant

REMARKS

- Slight improvement in the performance overall
- This is still a toy model with very specific examples
- Has not been tested with actual data
- Shows potential but there is room for improvement

- With this project I’ve noticed
  - There are many parameters to consider (architecture, nodes, optimizers)
  - There is no rule that let’s you know where to start or how to develop the correct model
  - There is a lot of trial and error.
  - You have to spend more time building the model than tuning the parameters.
- There have been many other versions of the architectures shown.
  - All show similar patterns for results

USED MODELS

For the models in the supervised approach :

- Loss is categorical cross entropy

For the more complex models

- Optimizer is Adam or other adaptive optimizers with similar results