



USING MACHINE LEARNING TECHNIQUES FOR DATA QUALITY MONITORING AT CMS EXPERIMENT

GUILLERMO A. FIDALGO RODRÍGUEZ

ML HACKATHON

The tools to achieve the mission to find new physics

LHC is one of the largest and truly global scientific projects ever, is a turning point in modern physics, big bang machine

- Large is an understatement
- Hadrons referred to here are protons
- Collide is what it does, as we will see



CMS



LHC ring
27 km circumference



LHCb



ALICE



ATLAS

The LHC Tunnel

16 miles of magnets and connections are kept colder than outer space, using over 100 tons of liquid helium

Protons race around a 27 km circuit at 99.999999% the speed of light

Protons crashing head on into each other 40,000,000 times a second

Violent collisions corresponding to temperatures a billion times higher than the core of the sun will be produced.

That is roughly 160,000,000,000,000,000 C

The beam pipe is evacuated to the same vacuum as interplanetary space

The pressure is about 1/10th that of the surface of the moon



CMS DETECTOR

Total weight : 14,000 tonnes
Overall diameter : 15.0 m
Overall length : 28.7 m
Magnetic field : 3.8 T

STEEL RETURN YOKE
12,500 tonnes

SILICON TRACKERS
Pixel (100x150 μm) $\sim 16\text{m}^2 \sim 66\text{M}$ channels
Microstrips (80x180 μm) $\sim 200\text{m}^2 \sim 9.6\text{M}$ channels

SUPERCONDUCTING SOLENOID
Niobium titanium coil carrying $\sim 18,000\text{A}$

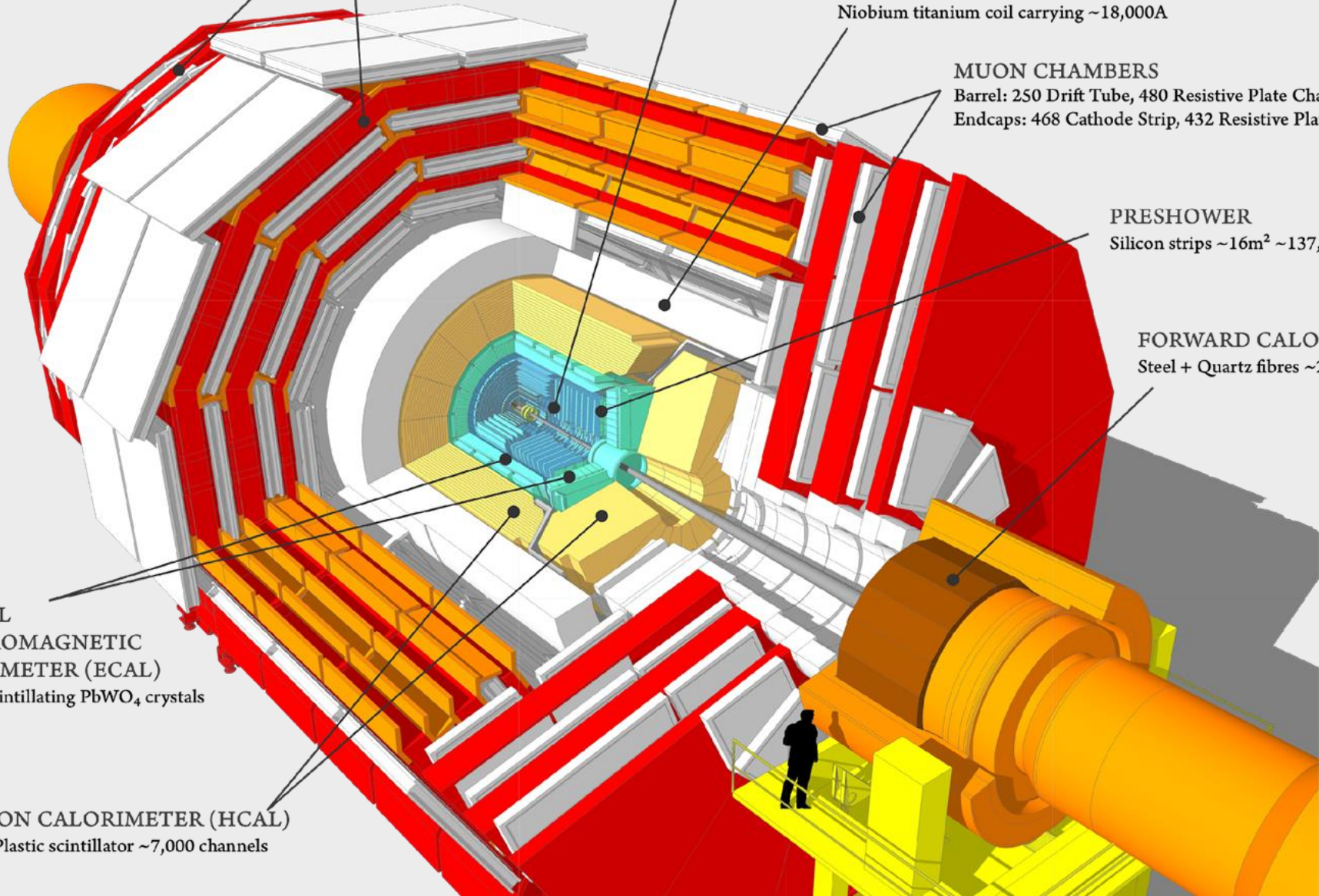
MUON CHAMBERS
Barrel: 250 Drift Tube, 480 Resistive Plate Chambers
Endcaps: 468 Cathode Strip, 432 Resistive Plate Chambers

PRESHOWER
Silicon strips $\sim 16\text{m}^2 \sim 137,000$ channels

FORWARD CALORIMETER
Steel + Quartz fibres $\sim 2,000$ Channels

CRYSTAL ELECTROMAGNETIC CALORIMETER (ECAL)
 $\sim 76,000$ scintillating PbWO_4 crystals

HADRON CALORIMETER (HCAL)
Brass + Plastic scintillator $\sim 7,000$ channels



THE COMPACT MUON SOLENOID (CMS) EXPERIMENT

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

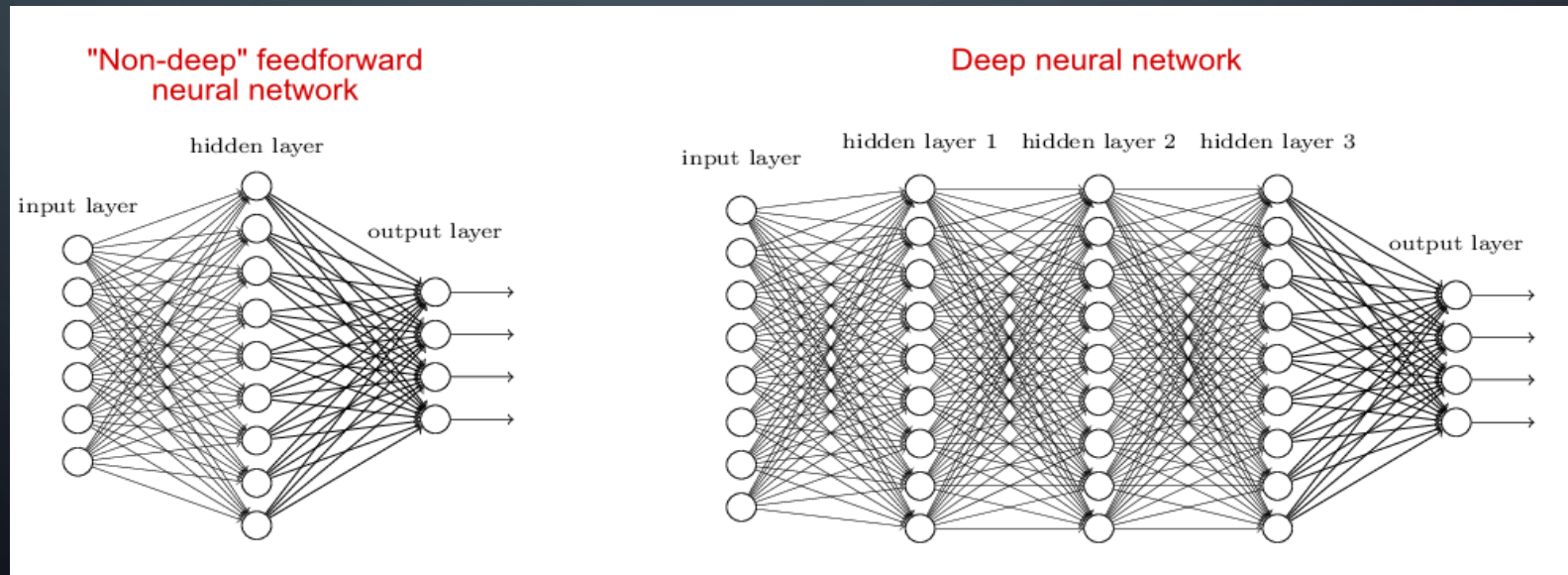
THE CHALLENGE

- You have to make sure that the detector behaves well in order to perform sensible data analysis.
- Reduce man power.
 - Shifters monitor constantly the quality of the data flow.
 - Discriminate between good and bad data to have high purity
 - Build something that helps the people to minimize the time needed to spot problems and save time examining hundreds of histograms
 - Build intelligence that analyzes the data and raises alarms in case of problems. Have quick feedback.



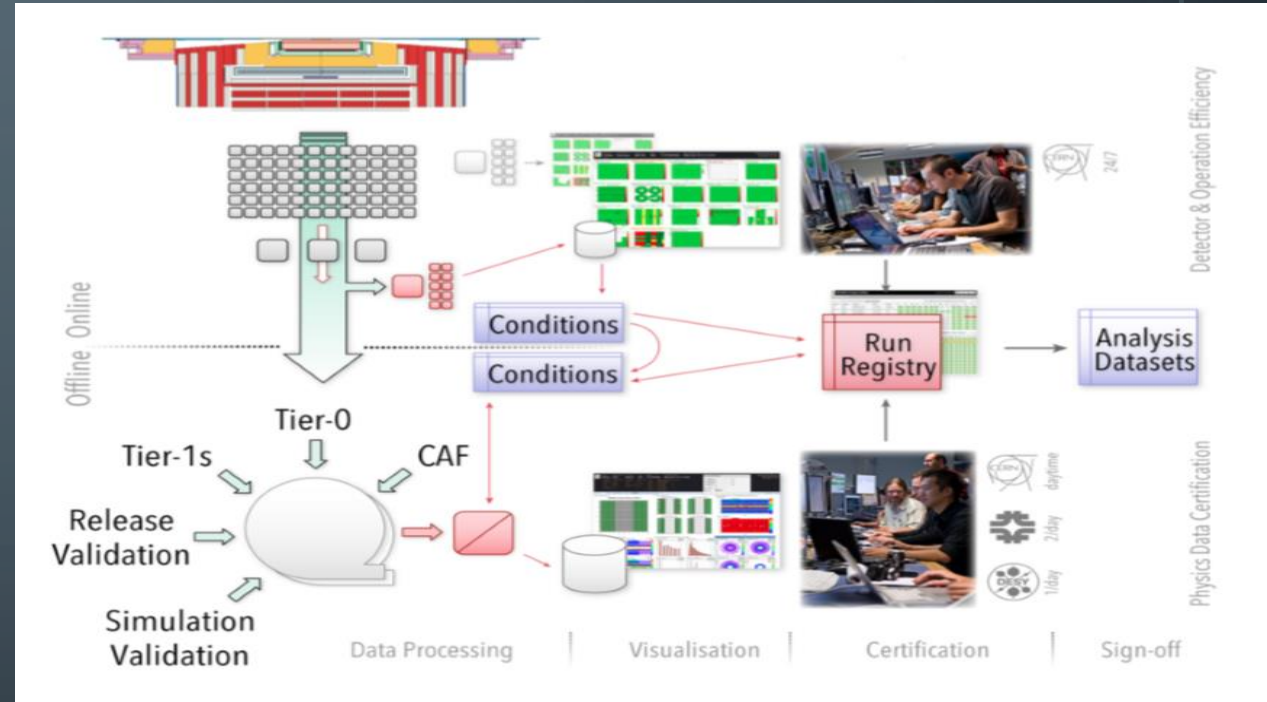
THE CHALLENGE (2)

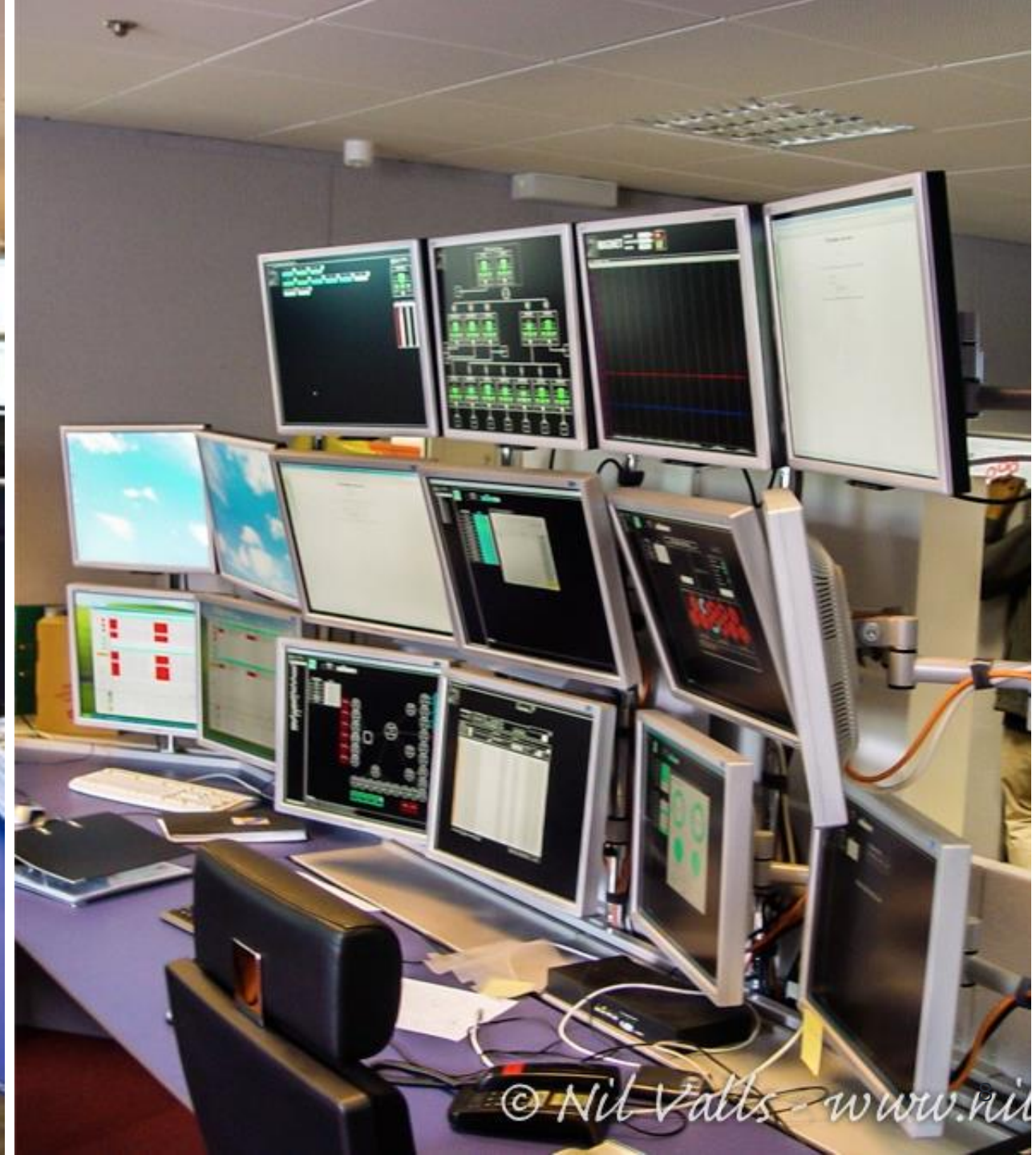
- Deciding the best architecture of the network is key
 - Too little and it may not be able to learn (underfitting)
 - Too big and it may learn to only identify very specific and/or unnecessary features (overfitting)
- There is no rule of thumb
 - Many, many, many..... possible combinations.



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.





© Nil Valls - www.nilvalls.com

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

- It's everywhere now!
 - A.I. Learning
 - Self-driving cars
 - How does Google/Facebook know what you want?
 - Face/Handwriting Recognition
- In our case everything reduces to a Classification problem
 - Anomaly Detection

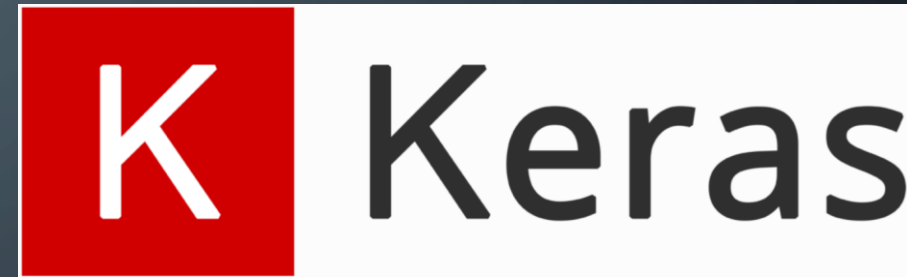


OBJECTIVES

- The project aims at applying recent progress in Machine Learning techniques to the automation of the DQM scrutiny for HCAL
 - Focus on the Online DQM.
 - Compare the performance of different ML algorithms.
 - Fully supervised vs semi-supervised approach.

TOOLS AND DATA PREPARATION

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



Machine Learning libraries

SCIKIT-LEARN

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

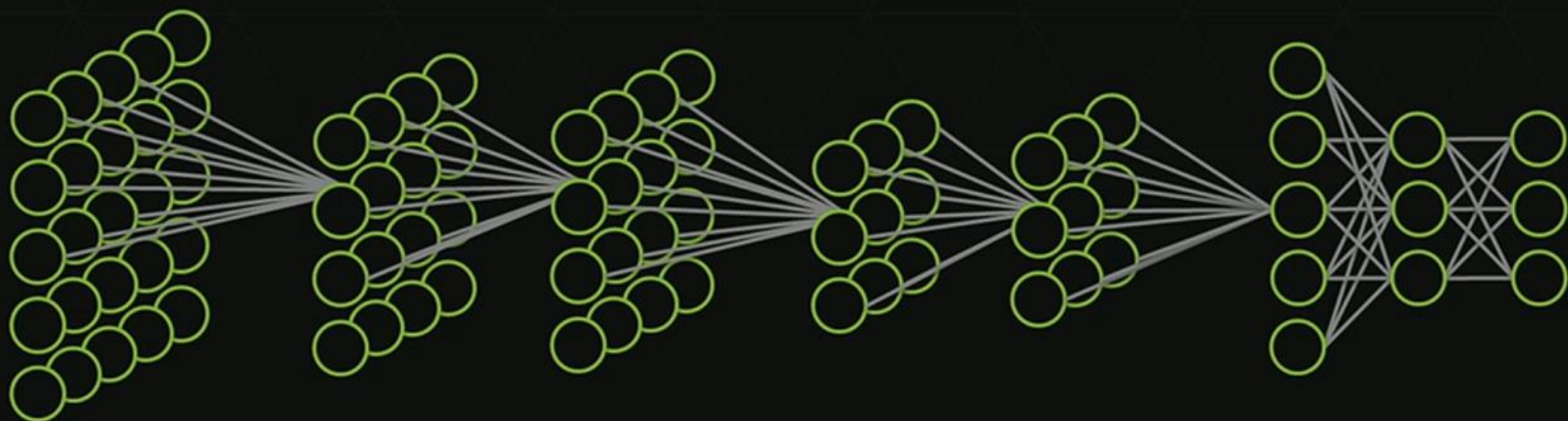
KERAS

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

HOW A DEEP NEURAL NETWORK SEES

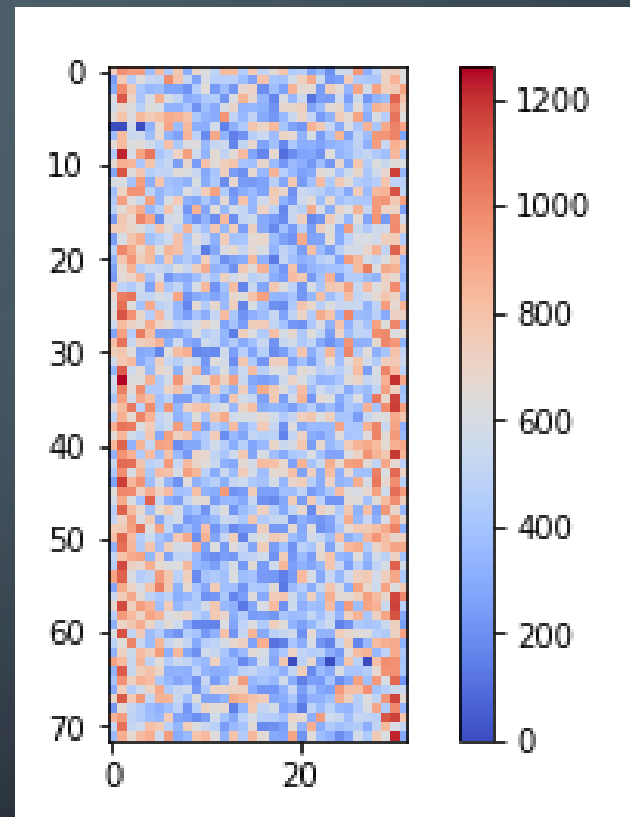
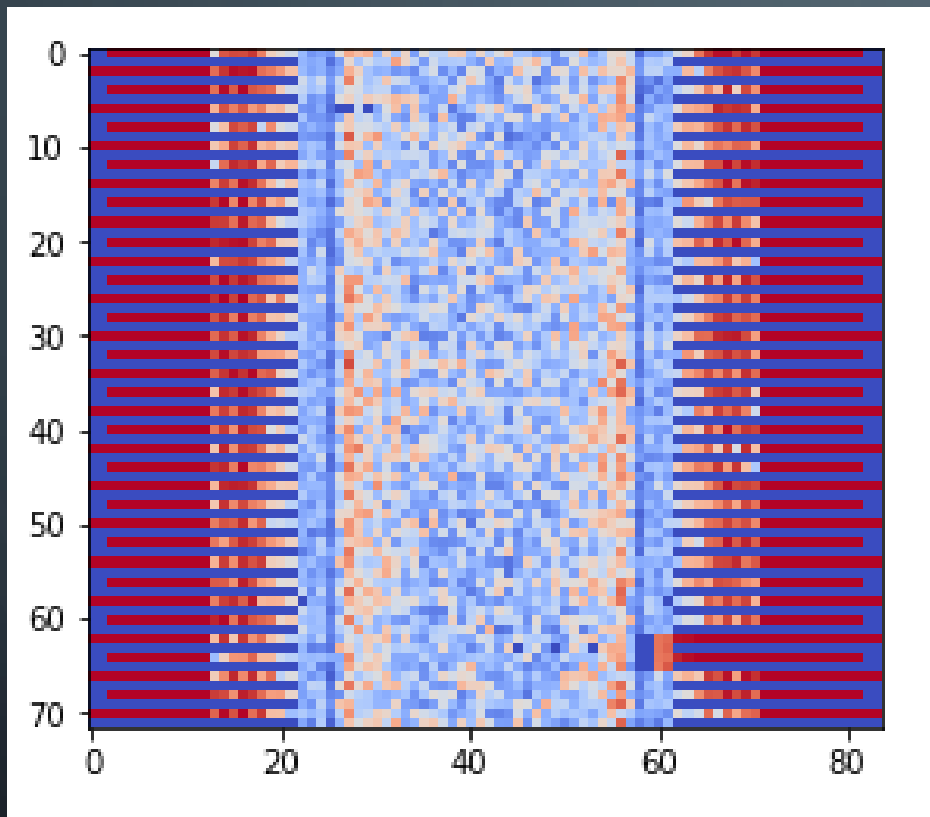


Image

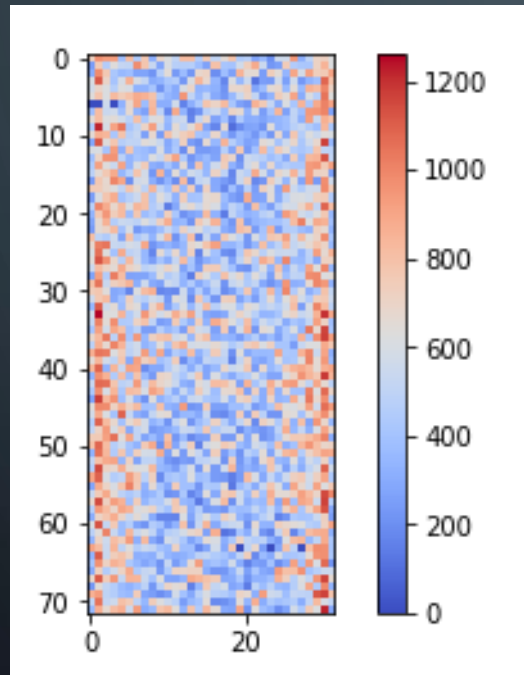


"Audi A7"

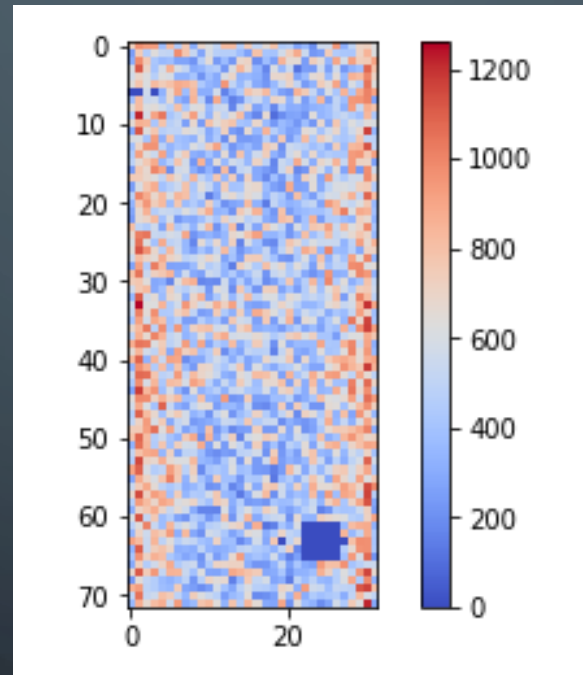
Images used



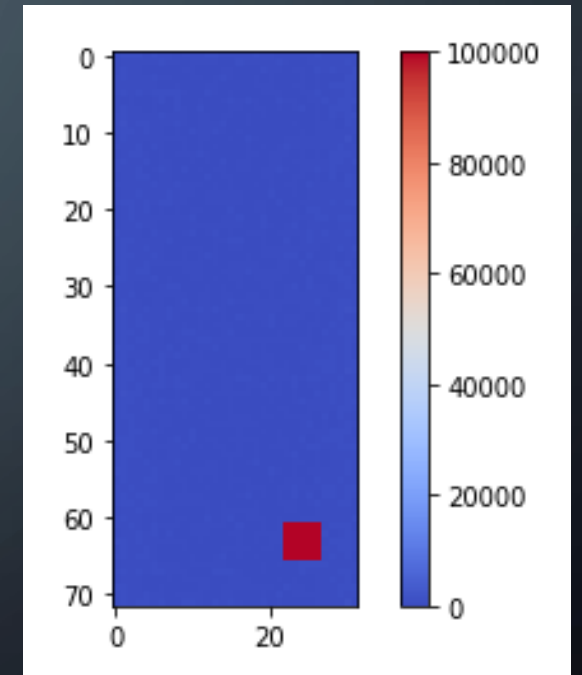
5x5 (readout channels) problematic region with fixed location
5x5 problematic region with random location



Good



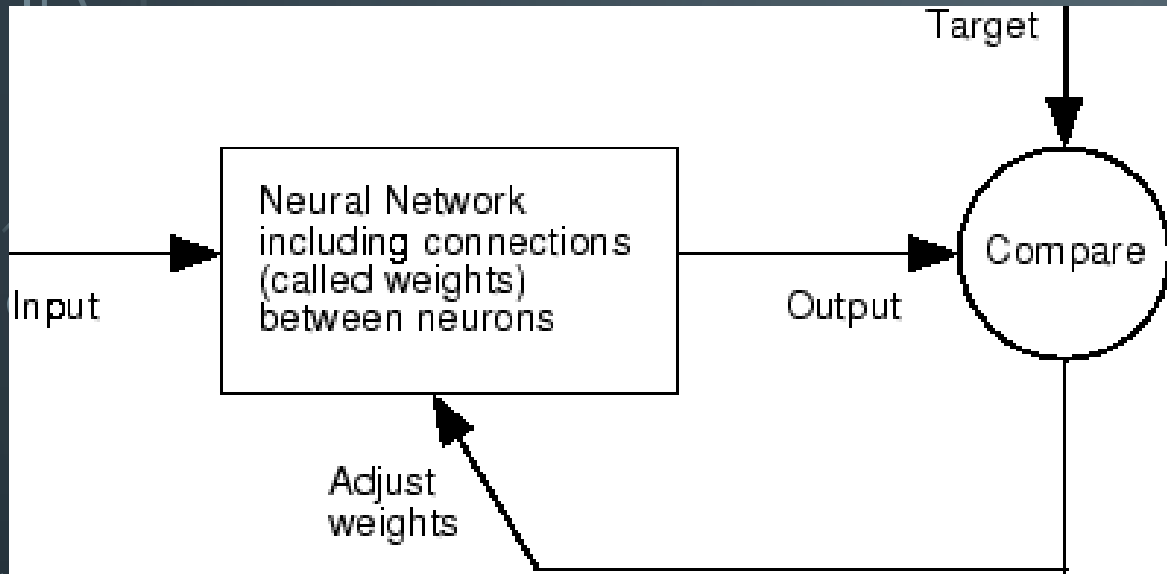
Dead



Hot

Images like these were used as inputs

How to train a model



Gradient Descent


The "Learning" in Machine Learning.

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

$$X = X - \eta \nabla(X)$$

X: weights or biases

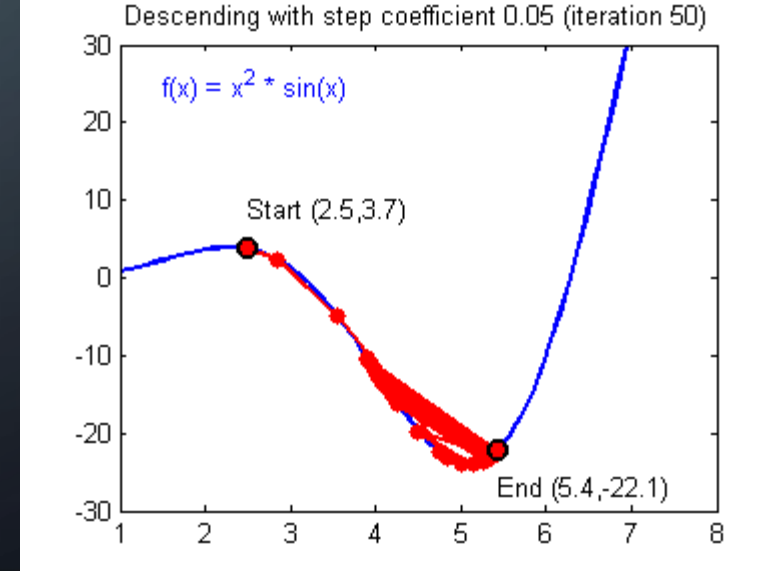
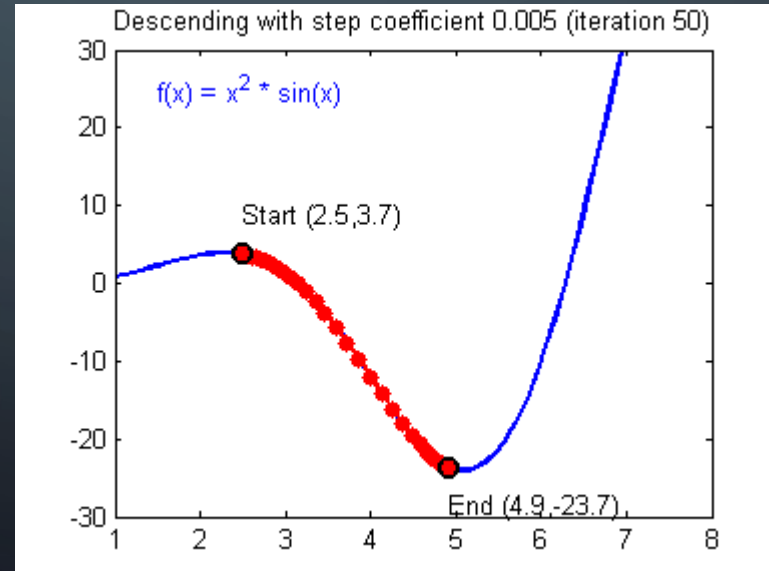
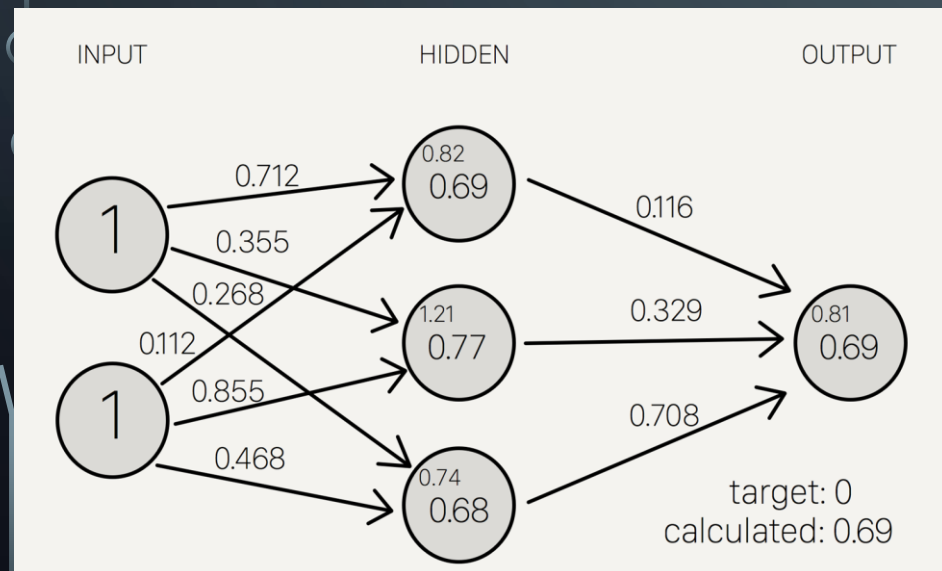
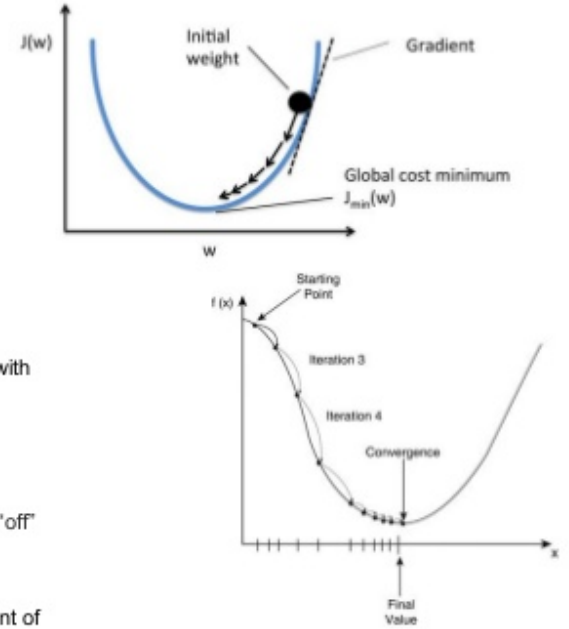
η : Learning Rate (typically 0.01 to 0.001)

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

$\nabla(x)$: 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



Models used

- Supervised - Convolutional Neural Network
 - Results
- Semi Supervised- Auto Encoder
 - Results

Supervised Model

```
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'))

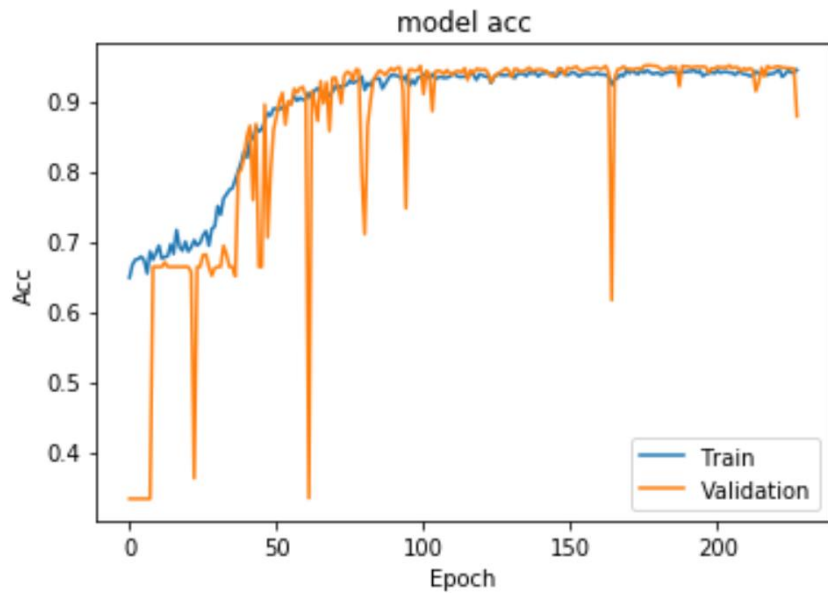
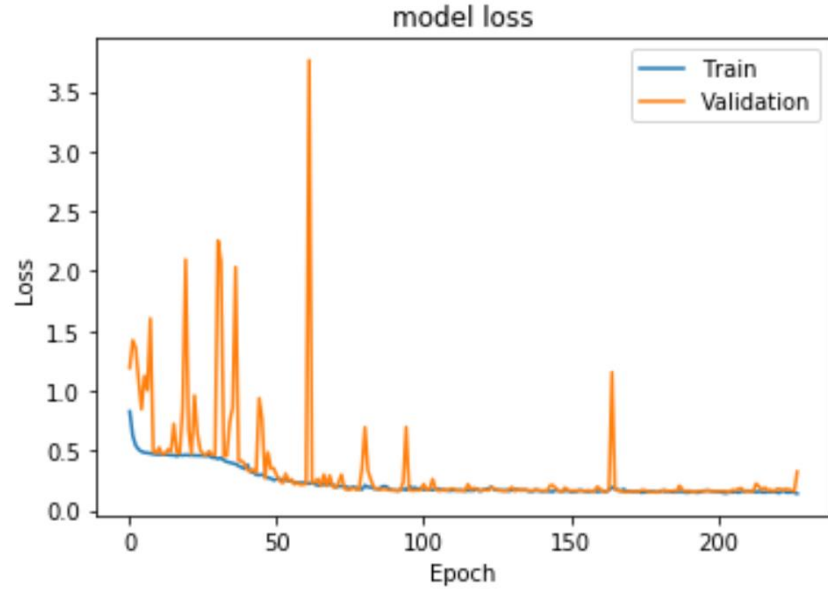
model.add(Dropout(0.25))
model.add(Flatten())

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

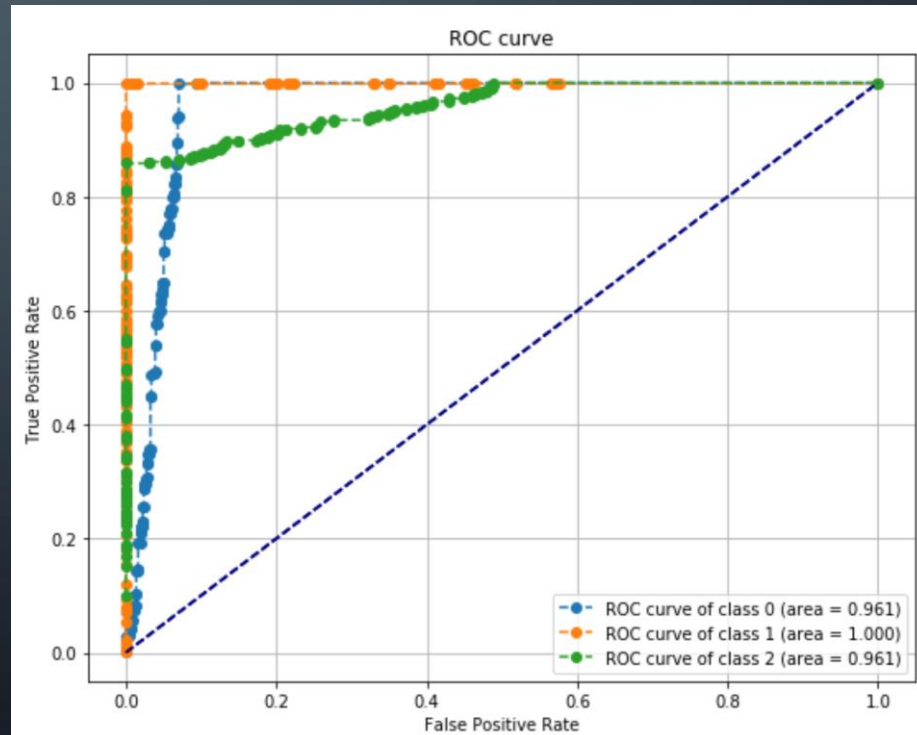
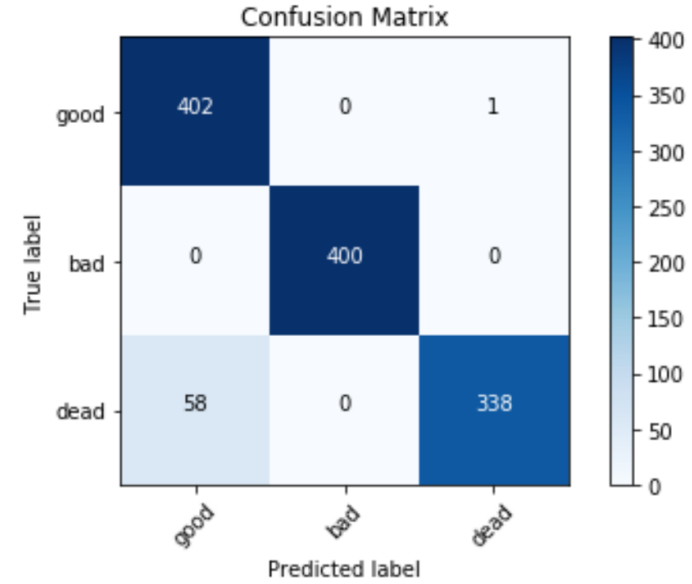
model.add(Dense(3, activation='softmax'))

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

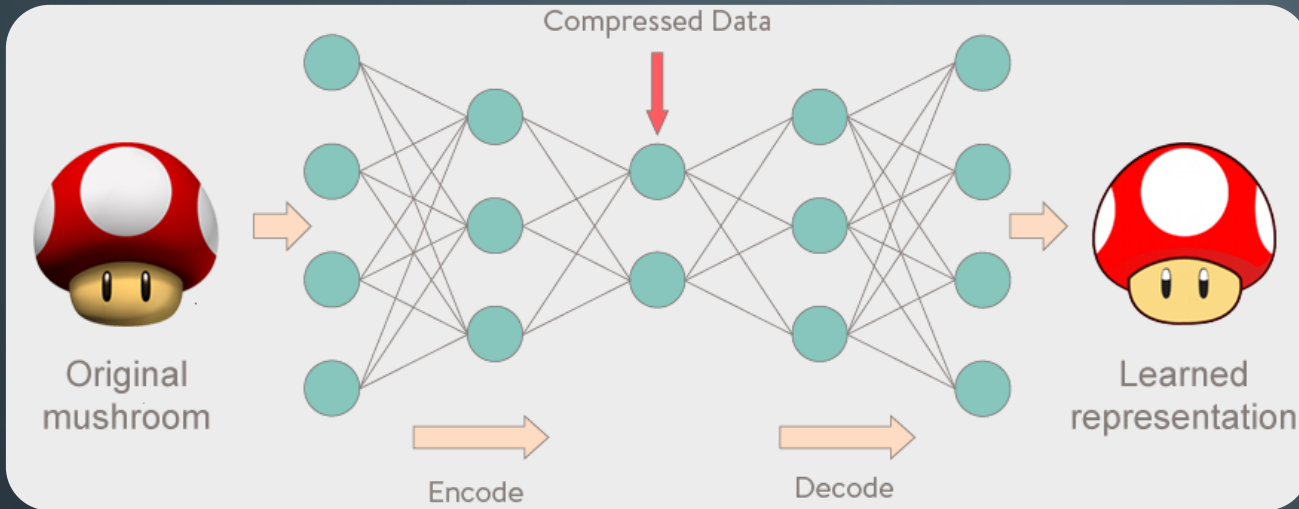
RESULTS



accuracy score: 0.950792326939

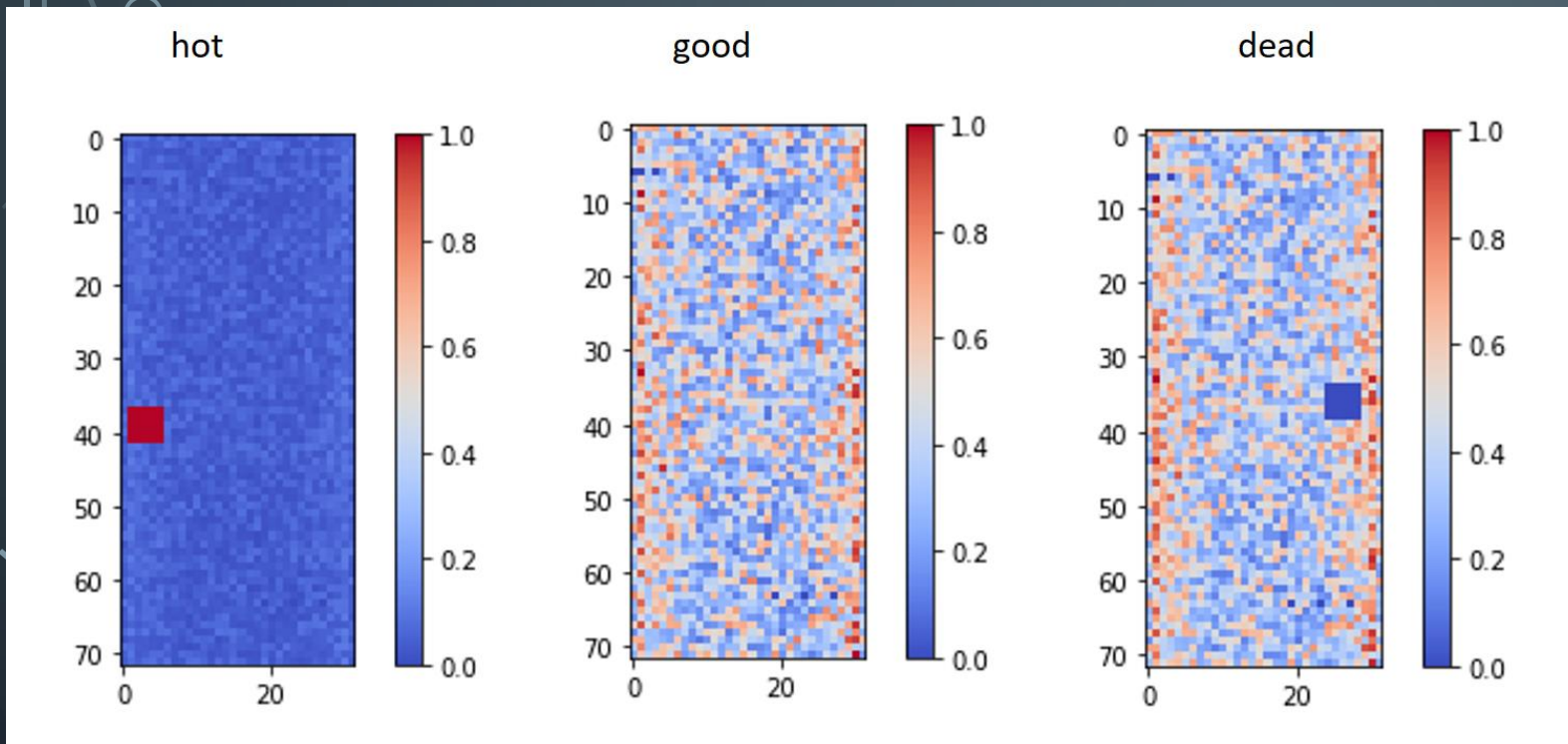


SEMI-SUPERVISED MODEL



Auto-Encoder

- 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



- Trained only on good images
- Expected to see better reconstruction for good images and a much different reconstruction for bad images.
- Bad images have 5x5 bad regions
 - Hot
 - Dead
- Images have been normalized

SETTINGS FOR THE MODEL

ARCHITECTURE

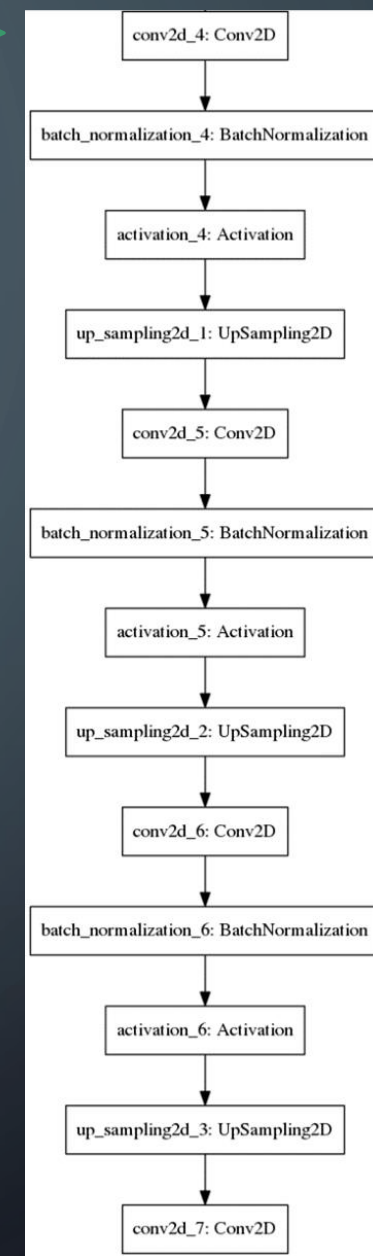
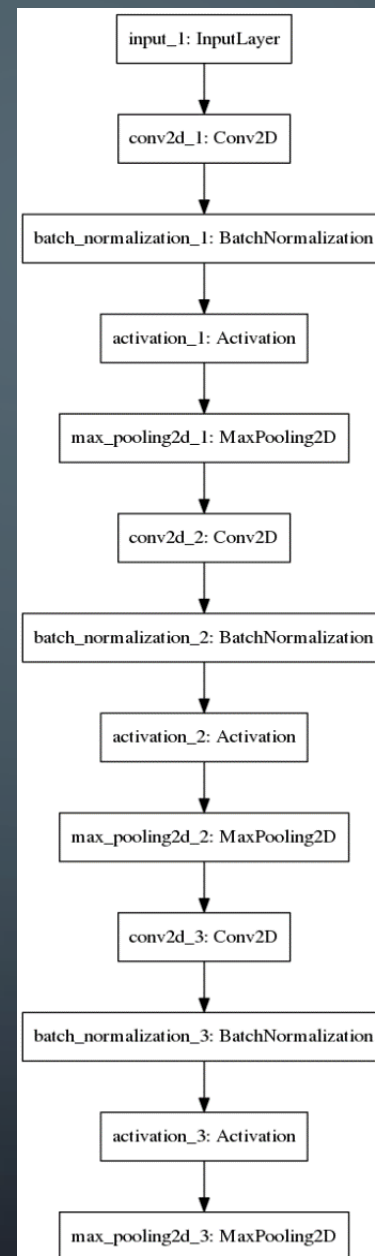
```
input_img = Input(shape=input_shape) # adapt this if using `channels_first`
```

```
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

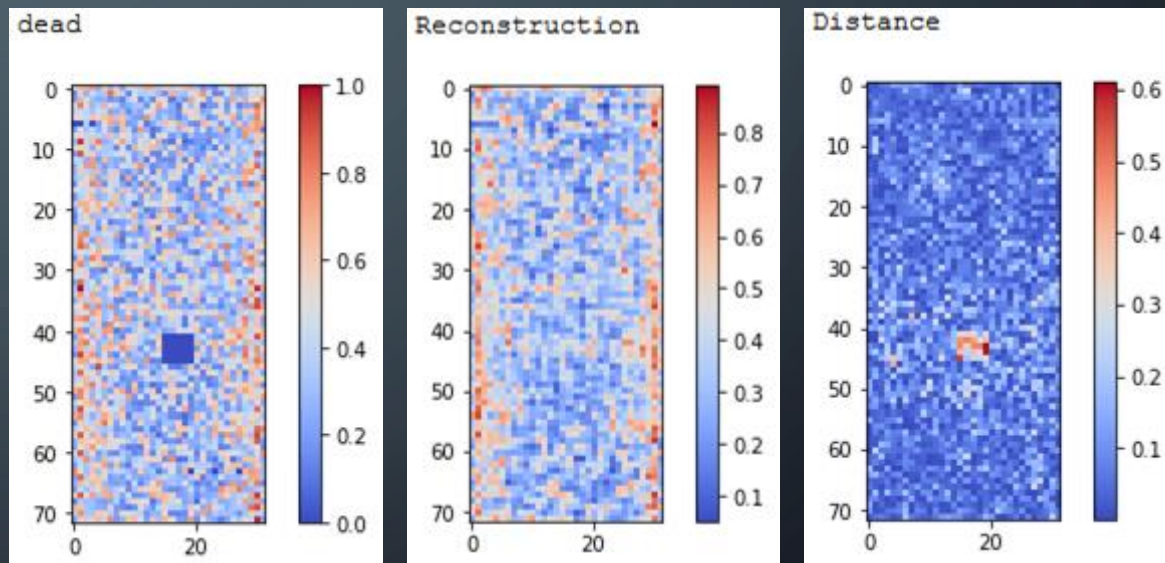
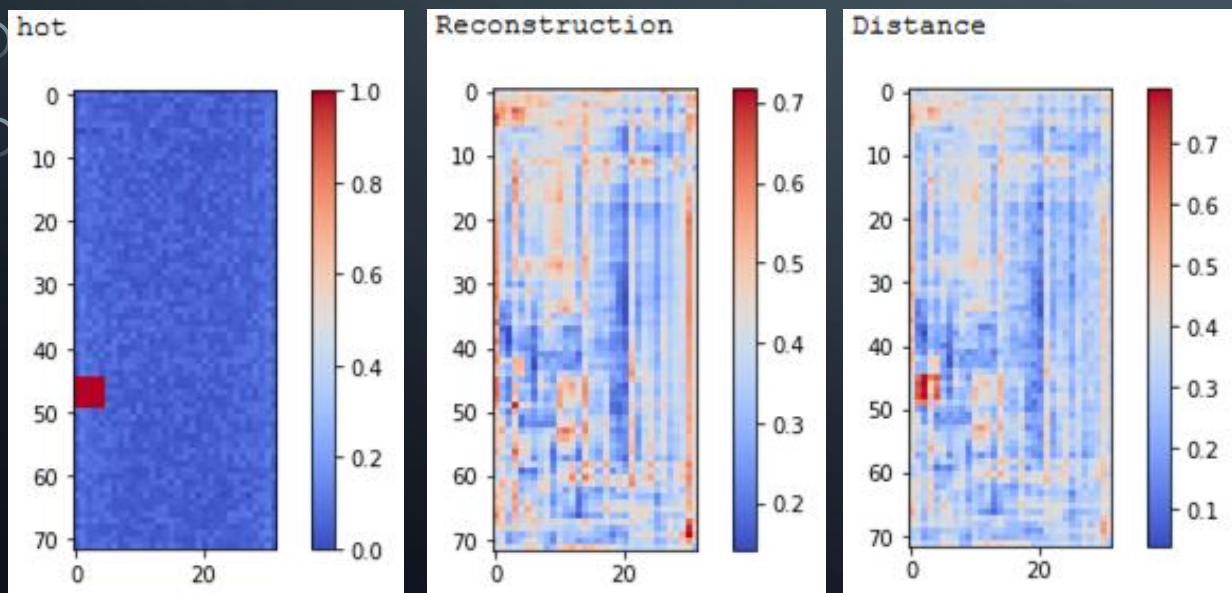
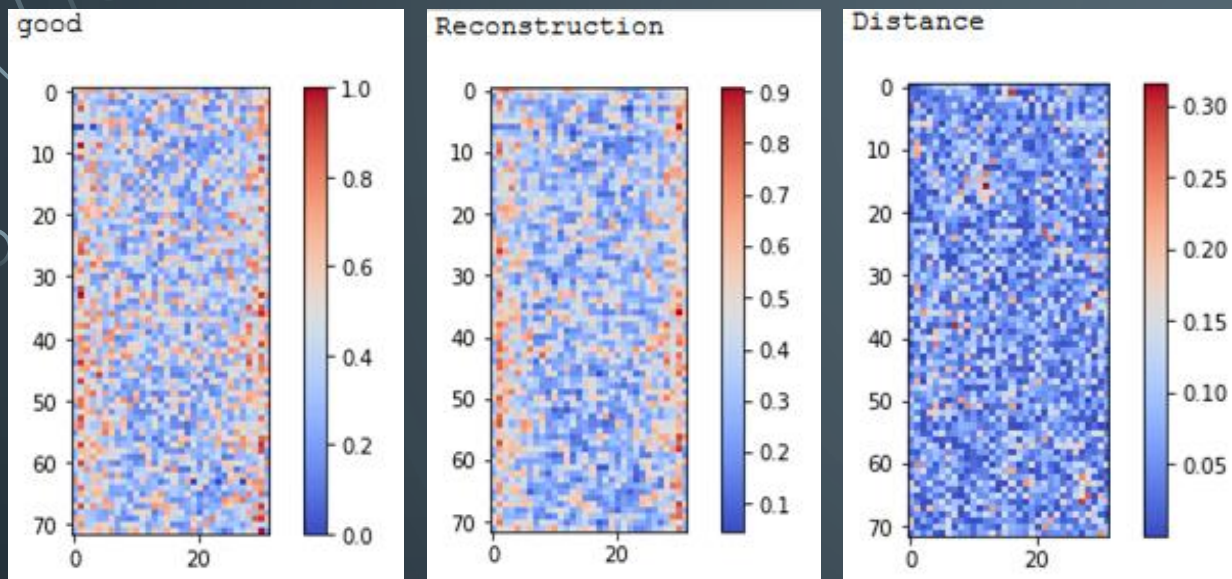
```
x = Conv2D(32, (3, 3), padding='same')(encoded)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(64, (3, 3), padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(86, (3, 3), padding='same')(x)
x = 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')
```



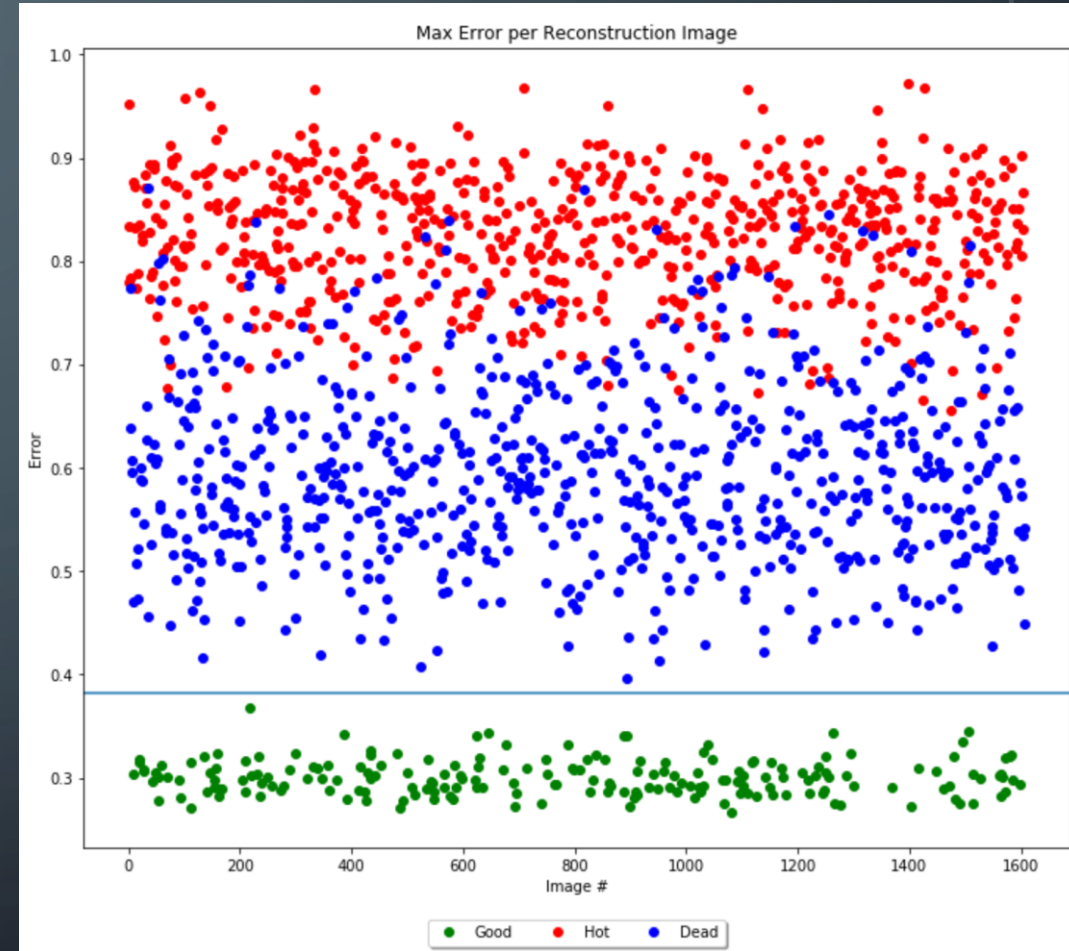
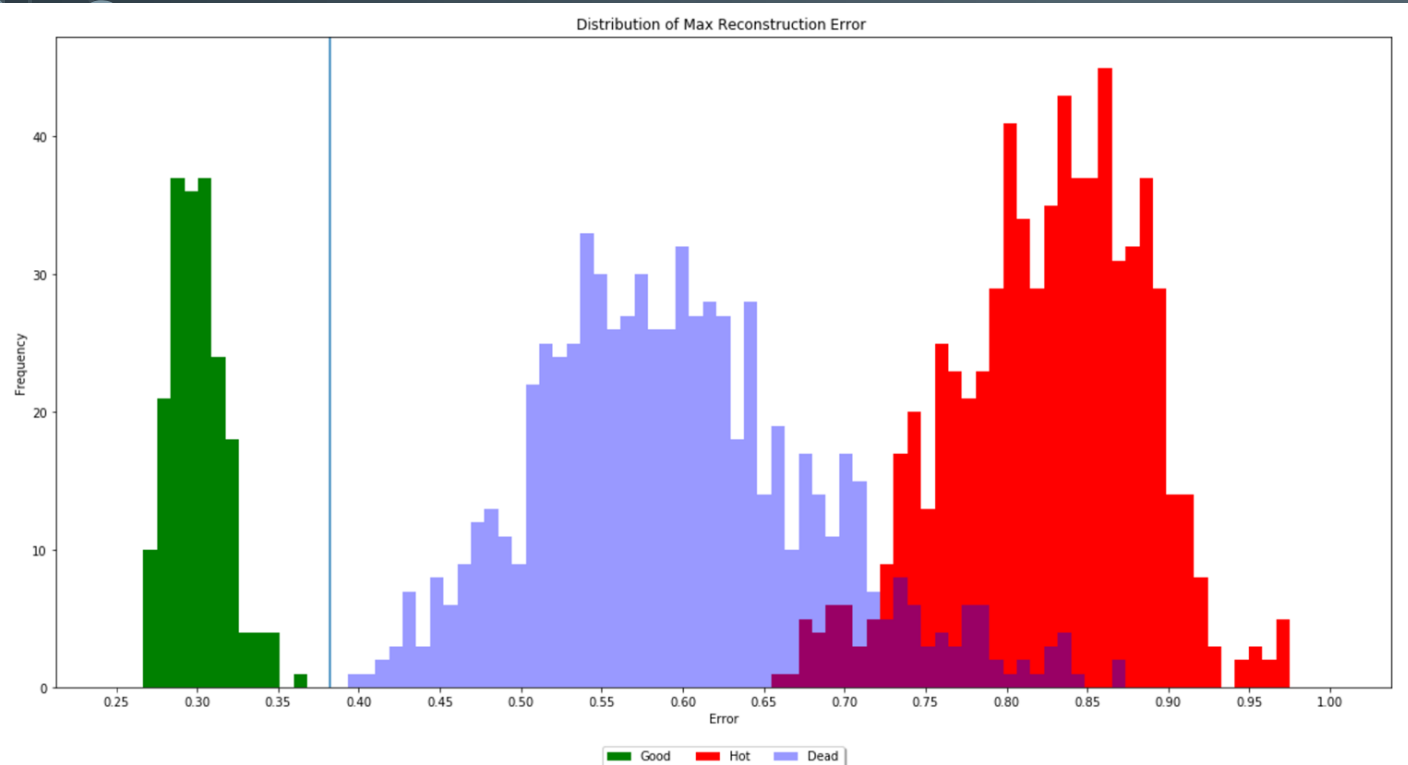
RESULTS

After testing different parameters this architecture seems to perform best for us.



RESULTS

The distribution of the error in the reconstruction confirms that they are differentiable and so we can simply place a cut and classify accordingly



The background is a dark blue gradient. In the four corners, there are decorative white line-art patterns resembling circuit traces or neural network connections. These patterns consist of straight lines of varying lengths and angles, ending in small white circles.

Thank you!

BACKUP

Acknowledgments

- Federico De Guio , Ph.D
- Nural Akchurin, PH.D
- Sudhir Malik , Ph.D
- The US State Dept.
- The University of Michigan
- CERN/CMS
- Steven Goldfarb, Ph.D
- Jean Krisch, Ph.D

WITH SIMPLE (BINARY, FIXED LOCATION) PROBLEM SIMPLE ARCHITECTURE

```
model = Sequential([
    Conv2D(64, kernel_size=(2, 2), activation='relu', input_shape=(input_shape), data_format='channels_last'),
    Flatten( ),
    Dense(2, activation='softmax')
])
model.compile(loss='categorical_crossentropy', optimizer='adadelta', metrics=['accuracy'])
```

Epoch 20/20

1198/1198 [=====] - 5s 4ms/step - loss: 8.0321 - acc: 0.5017 - val_loss: 8.0993 - val_acc: 0.4975

NEXT ARCHITECTURE

```
model = Sequential([
    Conv2D(64, kernel_size=(2, 2), activation='relu', input_shape=(input_shape), data_format='channels_last'),
    Conv2D(64, (2, 2), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    Conv2D(64, (2, 2), activation='relu'),

    Flatten( ),
    Dense(2, activation='softmax')
])
model.compile(loss='categorical_crossentropy', optimizer='adadelta', metrics=['accuracy'])
```

Epoch 20/20

1198/1198 [=====] - 66s 55ms/step - loss: 8.0860 - acc: 0.4983 - val_loss: 8.0188 - val_acc: 0.502

Architecture

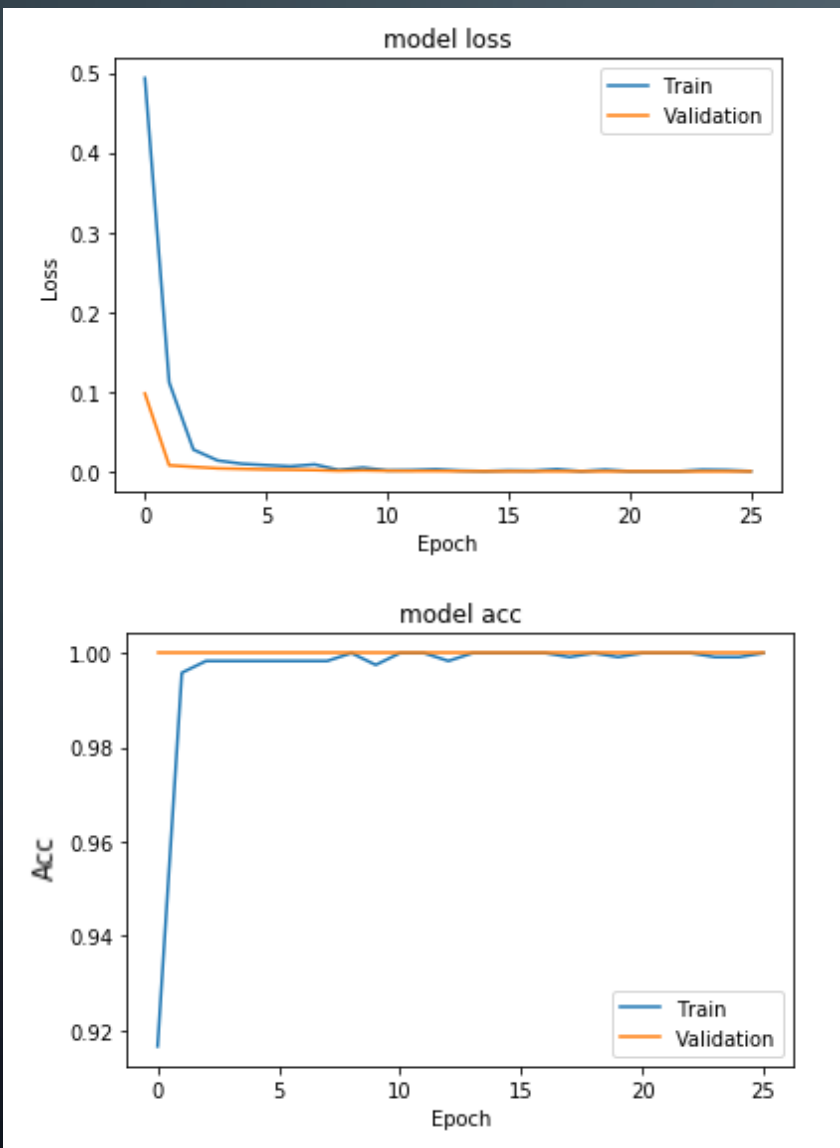
```
model = Sequential([
    BatchNormalization(input_shape=input_shape))
    Conv2D(8, kernel_size=(3, 3), strides=(2, 2), activation='relu')
    Conv2D(8, kernel_size=(3, 3), strides=(2, 2), activation='relu')

    Dropout(0.25)

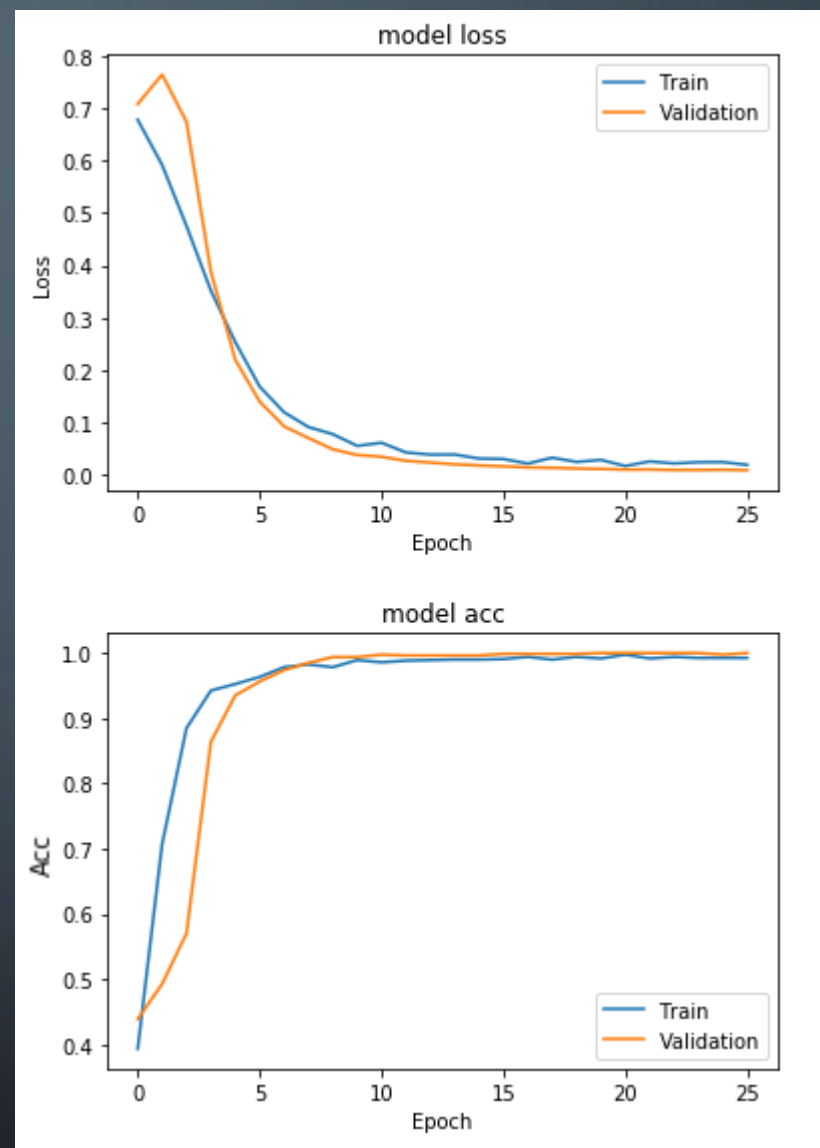
    Flatten()

    Dense(2, activation='softmax')
])
```

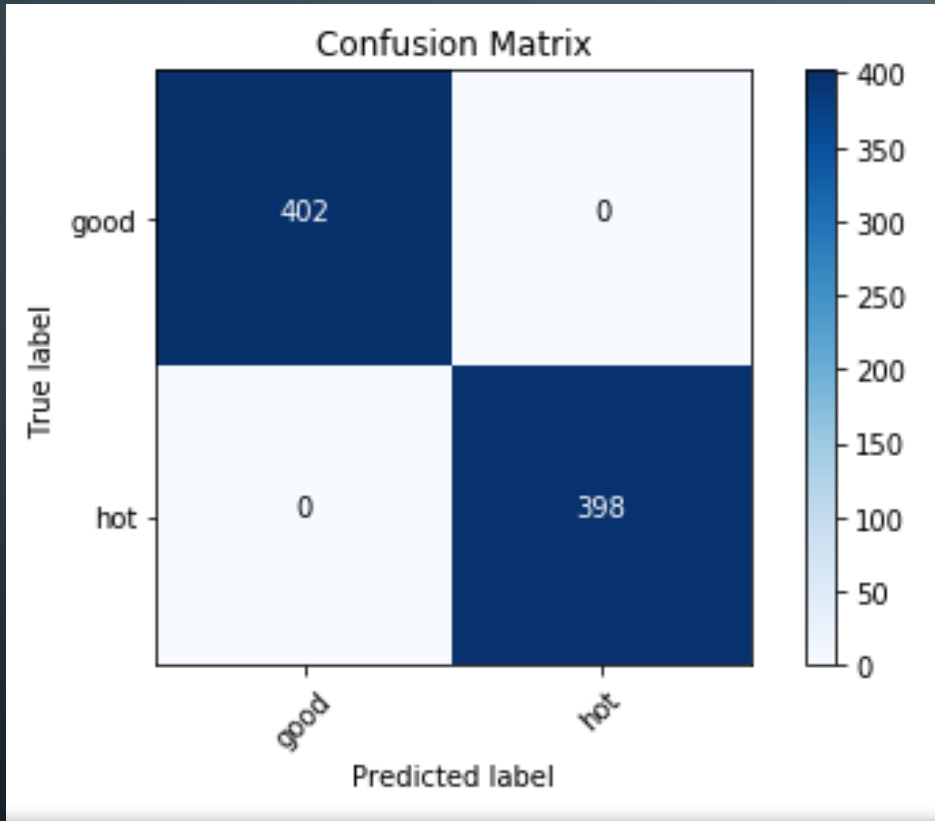
RESULTS



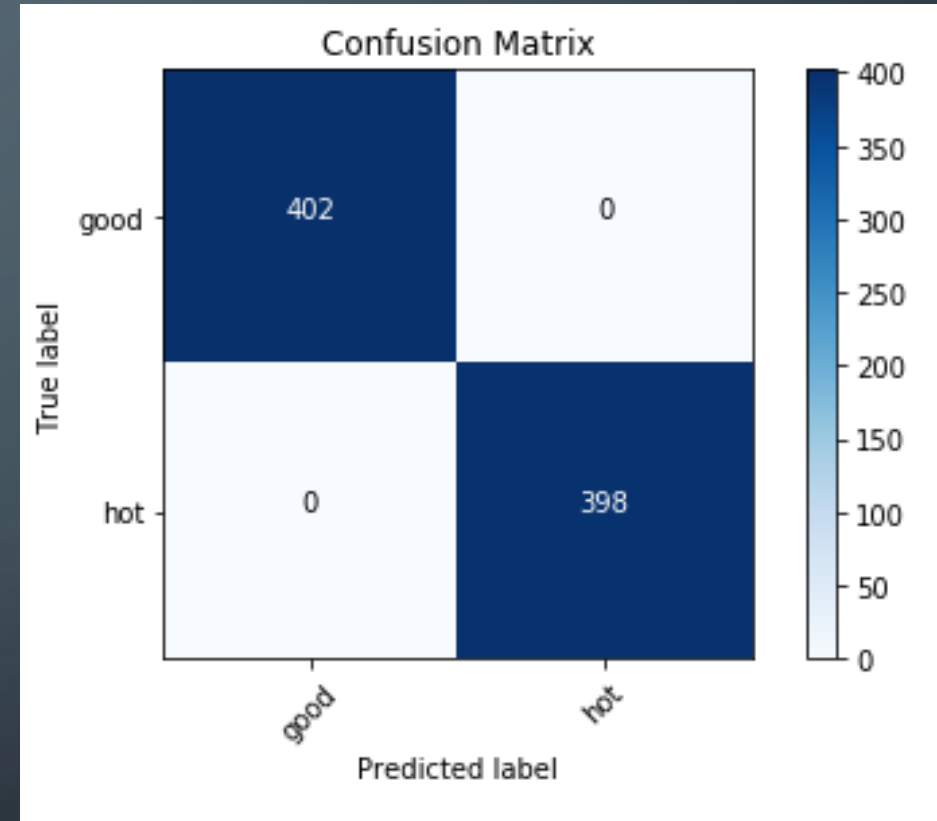
Fixed Location



Random Location



Fixed Location



Random Location

NOW LET'S MAKE THE PROBLEM A BIT HARDER

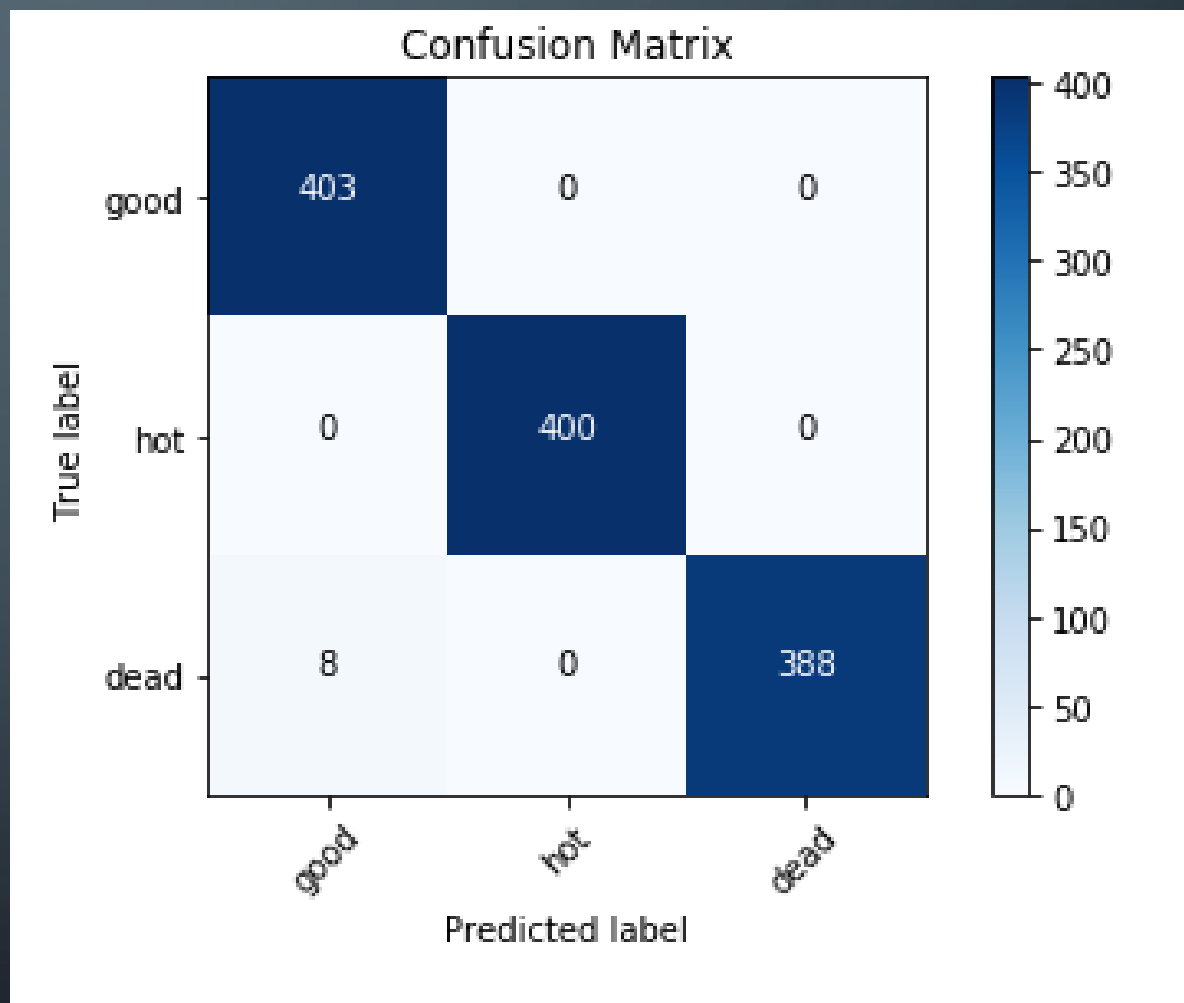
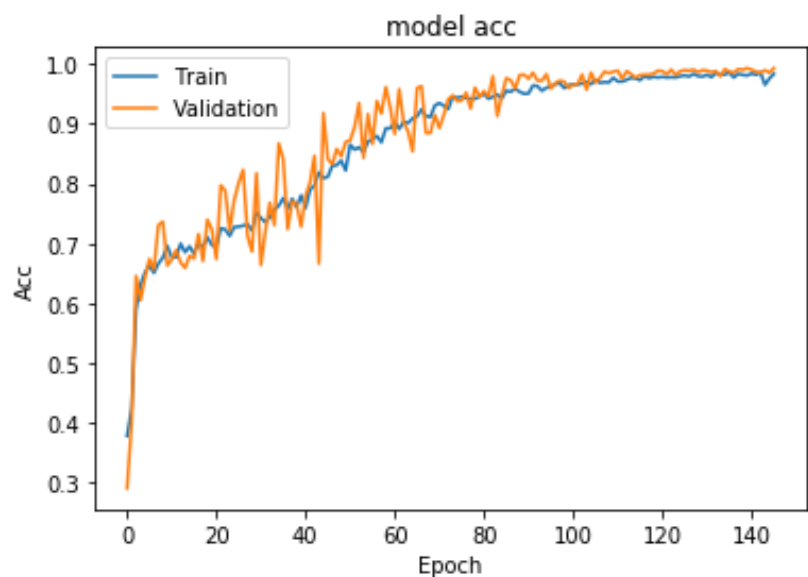
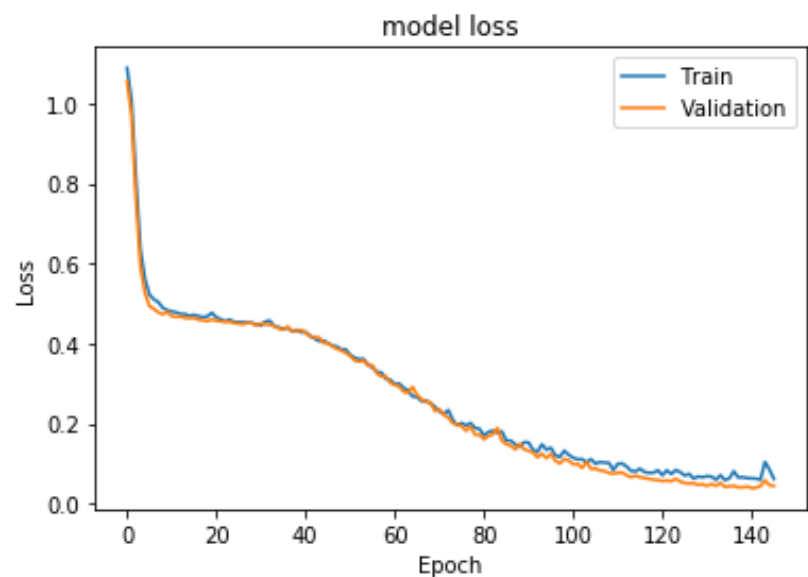
- With random position
- Multiclass problem (good ,hot, dead)
- Same arch.

RESULTS

Epoch 141/150

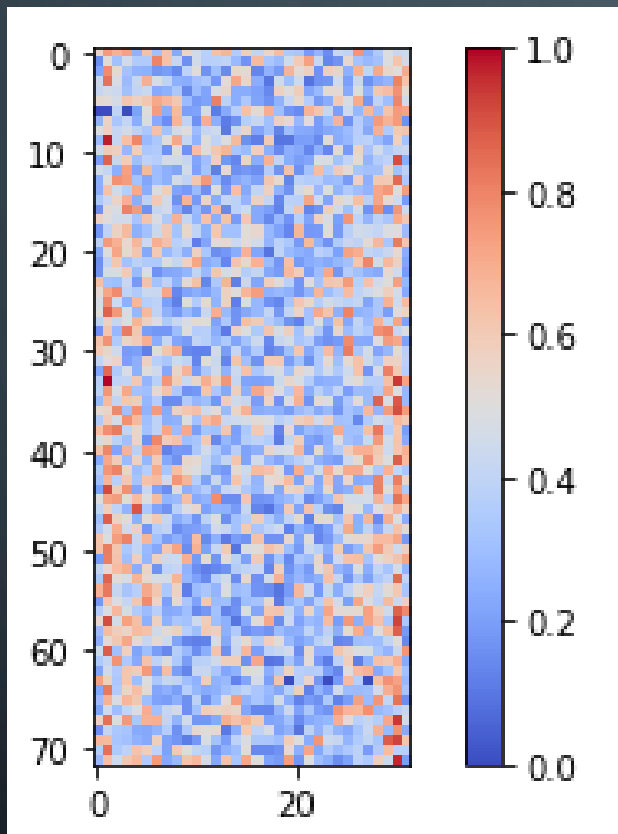
Epoch 00141: val_loss improved from 0.04021 to 0.03862, saving model to best_weights.hdf5

- 1s - loss: 0.0626 - acc: 0.9844 - val_loss: 0.0386 - val_acc: 0.9908

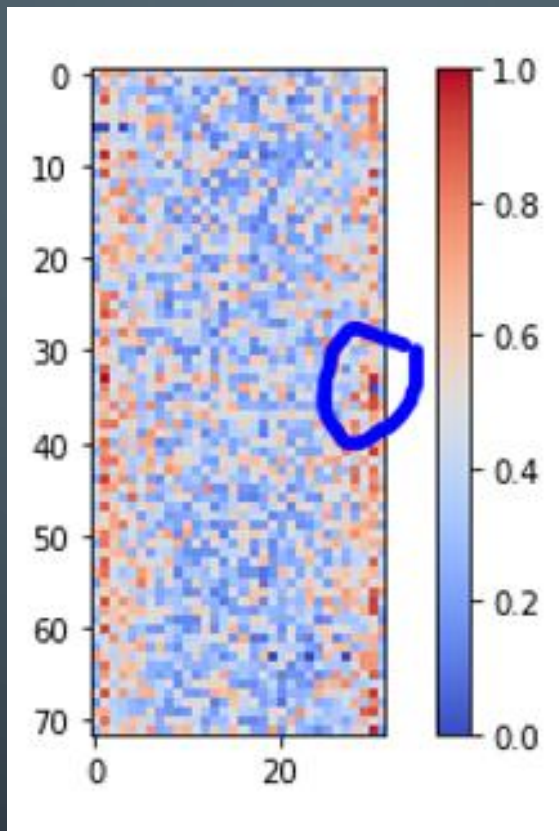


NOW LET'S MAKE THE PROBLEM MORE REALISTIC

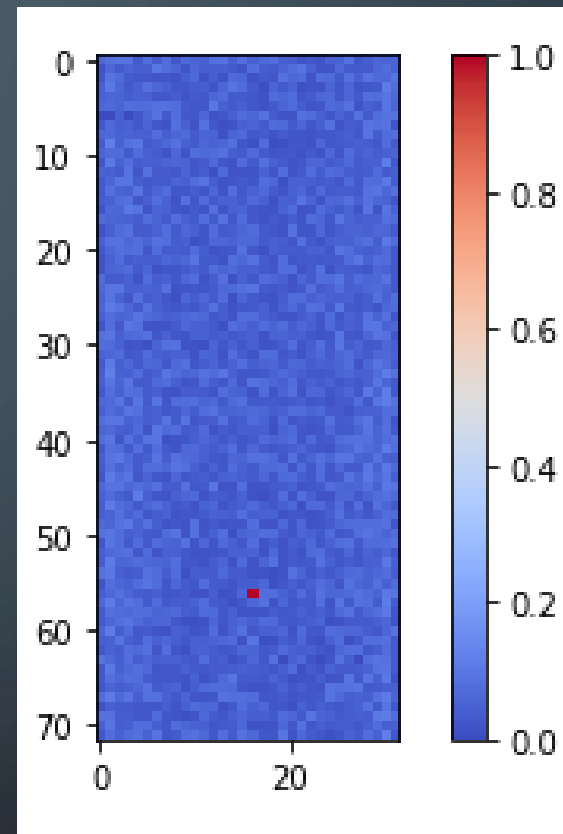
- With random position
- Multiclass problem (good ,hot, dead)
- Same arch.
- Only 1x1 bad channels



Good



Dead



Hot

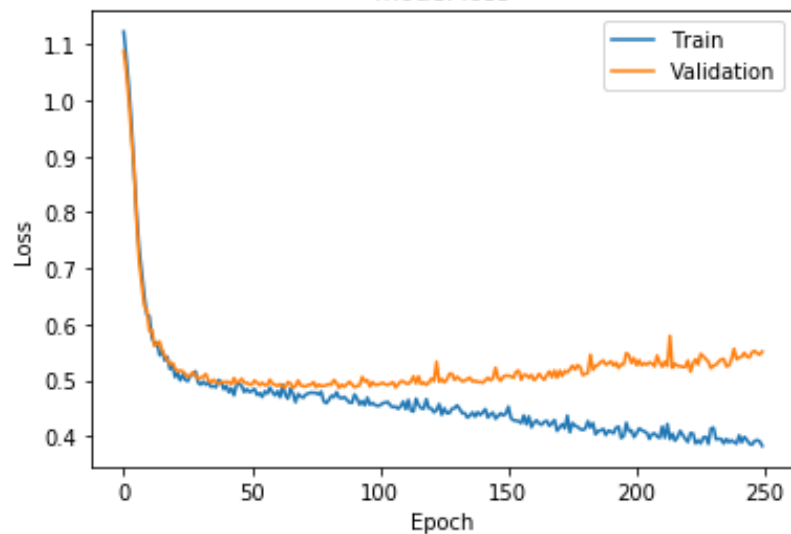
Images like these were used as inputs

Epoch 250/250

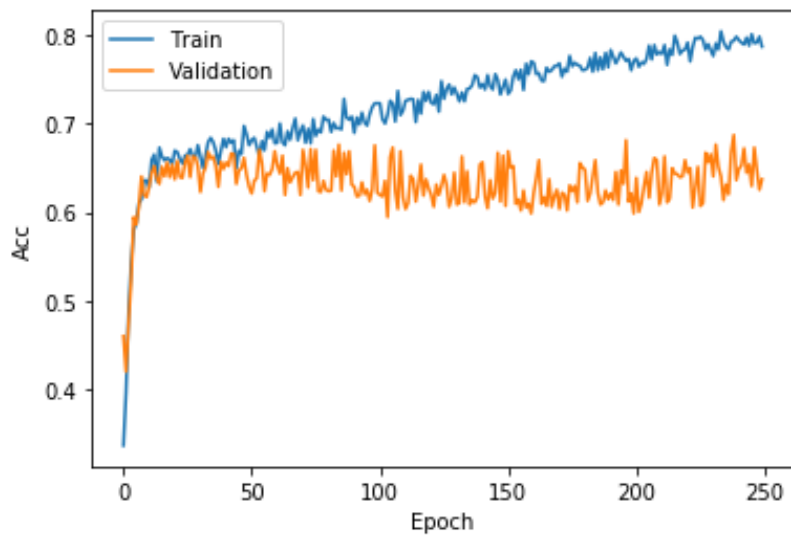
Epoch 00250: val_loss did not improve

- 1s - loss: 0.3824 - acc: 0.7870 - val_loss: 0.5509 - val_acc: 0.6372

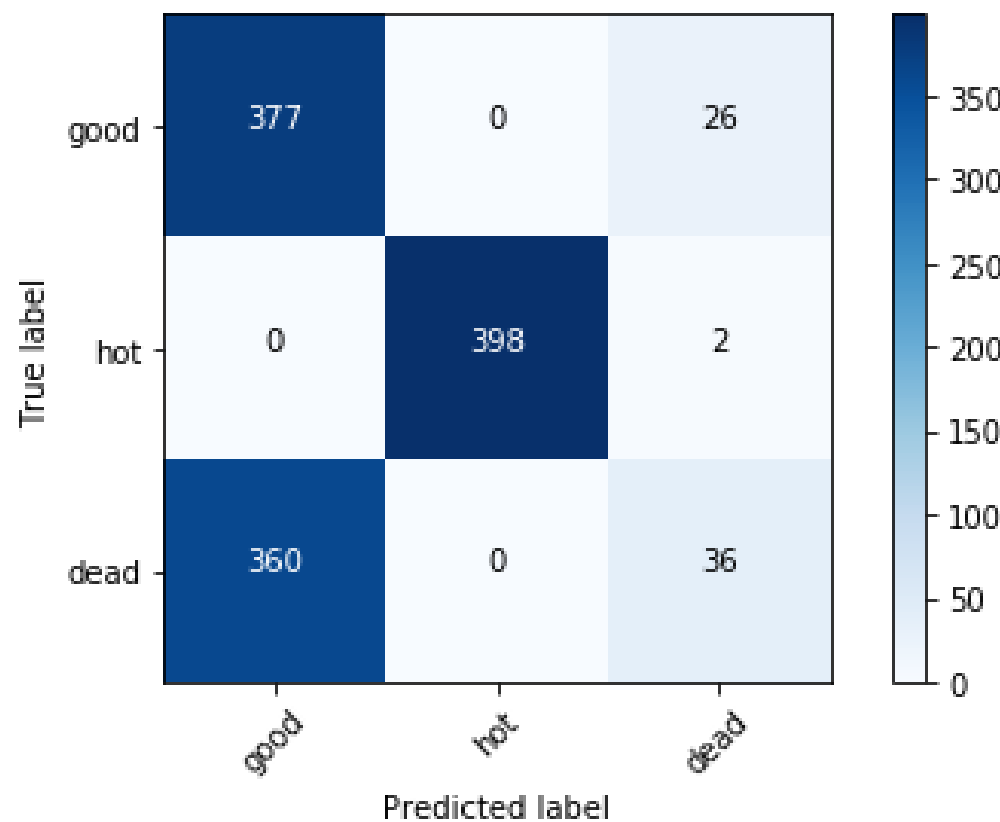
model loss



model acc



Confusion Matrix



ADD A COUPLE OF LAYERS

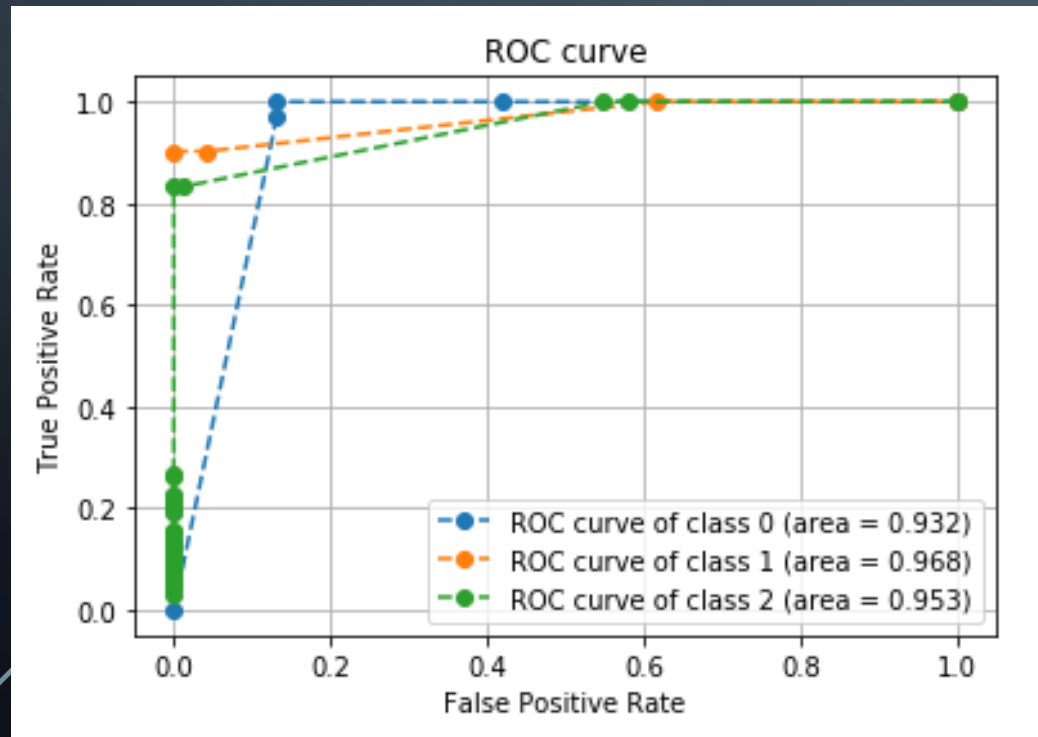
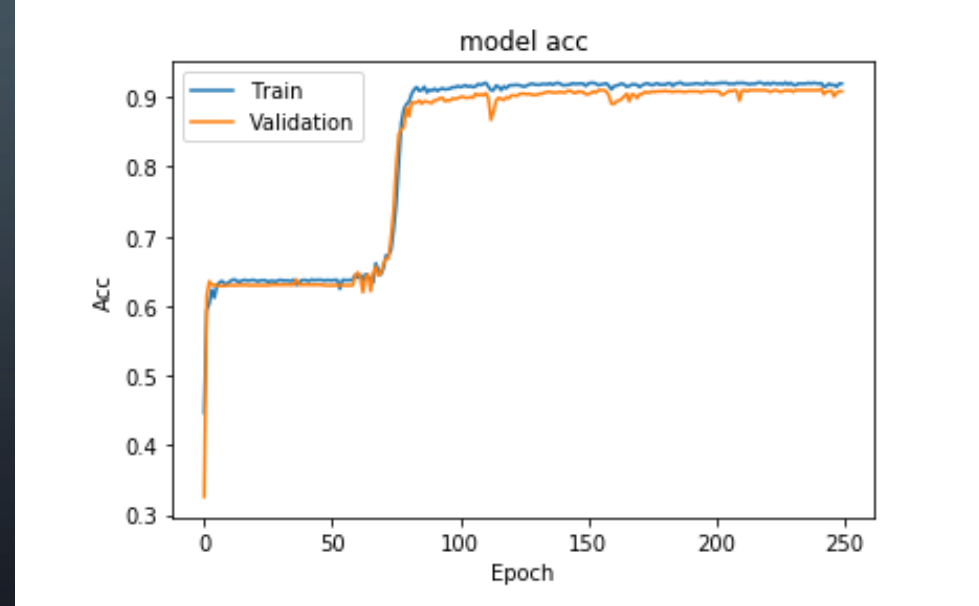
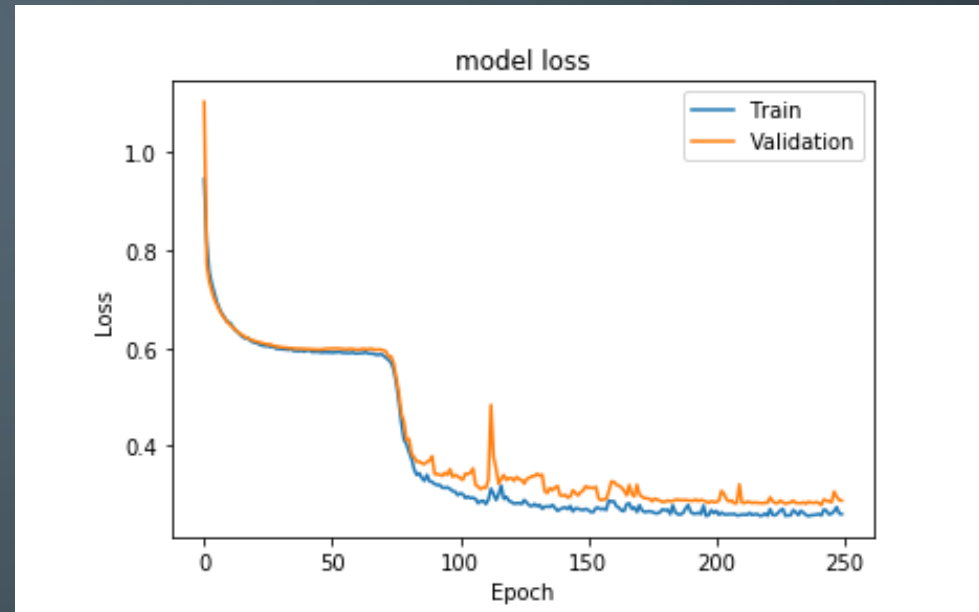
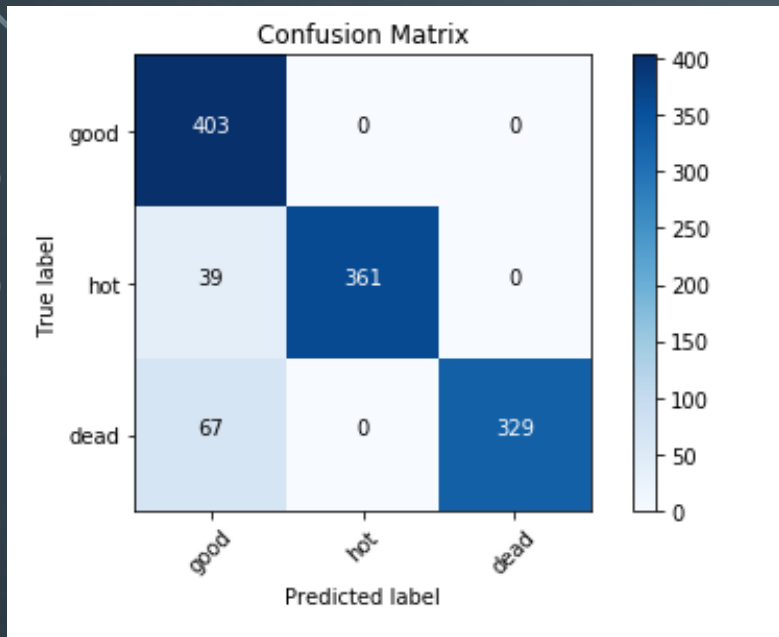
```
model = Sequential([
    Conv2D(10, kernel_size=(2, 2), activation='relu', strides=(1, 1), input_shape=input_shape),
    MaxPooling2D(pool_size=(2, 2)),
    BatchNormalization(),
    Conv2D(8, kernel_size=(3, 3), activation='relu', strides=(1, 1)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(8, kernel_size=(1, 1), activation='relu'),
    Dropout(0.25),
    Flatten(),

    Dense(8, activation='relu'),
    Dense(3, activation='softmax')
])
```

Epoch 250/250

Epoch 00250: val_loss did not improve

- 7s - loss: 0.2612 - acc: 0.9194 - val_loss: 0.2887 - val_acc: 0.9083



TUNING

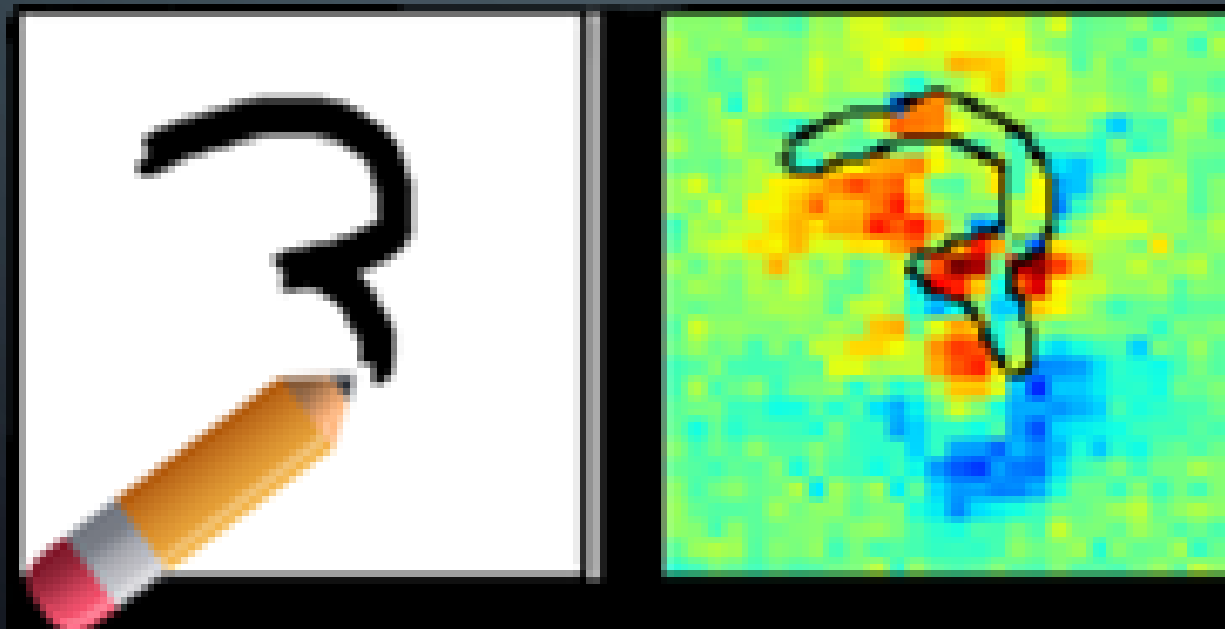
- Now we have to find a way to a make the model perform slightly better
- Minimize the amount of false positives

HOW TO FIX THIS?



WHAT'S NEXT?

- Why and exactly what is it learning?
- 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)



- 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

TOOLS AND DATA PREPARATION

- Have been familiarized with the following tools:
 - Working with data stored as hdf5 files
 - Familiarize with NumPy arrays
 - Working env: Juypiter python notebook
 - Matplotlib is used for plotting results
- Data comes in form of occupancy maps for HCAL
 - Flow of one map each lumisection for every lumisection.

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