# Studying jet quenching phenomenon using deep learning on low level variables

Filipe Cunha

*filipe.miranda.cunha@tecnico.ulisboa.pt*
Supervised by: Nuno Castro, Miguel Romão

September 11, 2020

# Table of Contents

# Jets

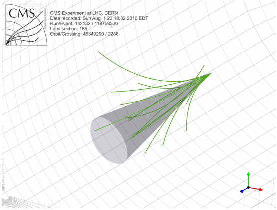



Figure 1: Visualization of a jet

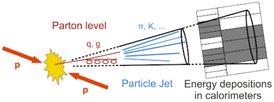



Figure 2: Sketch of pp-collision and resulting collimated spray of particles, a jet

## Jet-Quenching

In heavy ion collisions, a dense medium called the quark gluon plasma is created, and jets can interact with this medium reducing their energy.

# Table of Contents

# Convolutional Neural Networks

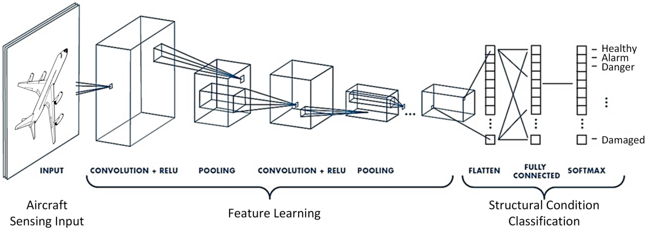



Figure 3: CNN structure

# Convolutions

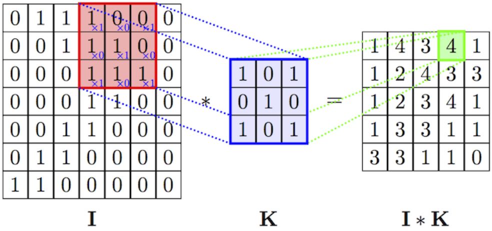



Figure 4: Convolution Operation Schema

## Mathematical definition

$$w_{ijk}^{(q)} = \sum_{r=1}^{F_q} \sum_{s=1}^{F_q} \sum_{k=1}^{d_q} w_{rsk}^{(p,q)} h_{i+r-1,j+s-1,k}^{(q)}$$

# Preventing Overfitting

## $L^p$-Regularization

- Adds a penalty $\lambda||\bar{W}||^p$ to the loss function;
- Constrains a model to use fewer non-zero parameters;

## Early Stopping

- Stops gradient descent after consecutive iterations without improving;

## Dropout

- Each neuron has a probability p of being dropped out of training for each iteration;
- Prevents co-dependency between neurons;

# Processing Data

## Dataset

- Very Sparse Matrices
- 1165404 Images, separated in Training, Validation and Test Sets

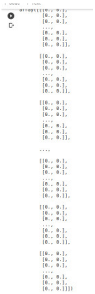


Figure 5: Image example

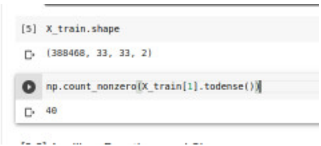



Figure 6: Dataset

# Processing Data

```
class Generator(tf.keras.utils.Sequence):
    "Generates data for Keras"

    def __init__(self,
                 X,
                 y,
                 weights,
                 batch_size=32,
                 shuffle=True,
                 normalise=True):
        "Initialization"
        self.X = X
        self.y = y
        self.batch_size = batch_size
        self.weights = weights
        self.shuffle = shuffle
        self.normalise = normalise
        self.on_epoch_end()

    def __len__(self):
        "Denotes the number of batches per epoch"
        return math.ceil(self.X.shape[0] / self.batch_size)

    def __getitem__(self, index):
        "Generate one batch of data"
        # Generate indexes of the batch
        indeces = self.indeces[index * self.batch_size : (index + 1) * self.batch_size]

        X_tmp = self.X[indeces].todense()
        y_tmp = self.y[indeces]
        w_tmp = self.weights[indeces]

        if self.normalise:
            X_tmp /= X_tmp.max(1).max(1).shape

        return X_tmp, y_tmp

    def on_epoch_end(self):
        "Updates indexes after each epoch"
        self.indeces = np.arange(len(self.X))
        if self.shuffle == True:
            np.random.shuffle(self.indeces)
```

Figure 7: Generator class

# Table of Contents

# Basic Networks

## Training the model

- Sigmoid activation in the last layer.
- Binary cross-entropy as the loss function.
- Early Stopping after 5 consecutive iterations without improving.

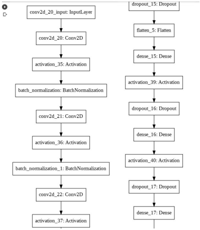



Figure 8: AlexModelV2

| | Train AUC | Val Loss | Val AUC |
|---|---|---|---|
| activation: relu<br>strides: 2<br>lr: 0.264609 | 0.4986 | 22.2533 | 0.5000 |
| activation: selu<br>strides: 4<br>lr: 0.077206 | 0.5014 | 4.6076 | 0.5000 |
| activation: relu<br>strides: 2<br>lr: 3.31145 | 0.8249 | 0.8246 | 0.6331 |
| activation: selu<br>strides: 4<br>lr: 0.010146 | 0.5011 | 1.3742 | 0.5000 |

Table 1: Tweaked AlexNet

# VGG

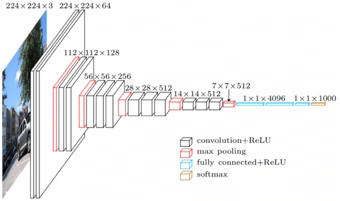



Figure 9: VGG Schema

| Depth | Train AUC | Val Loss | Val AUC |
|---|---|---|---|
| 10 | 0.6531 | 0.7563 | 0.6321 |
| 16 | 0.7125 | 0.6875 | 0.6799 |
| 21 | 0.7041 | 0.6385 | 0.6888 |

Table 2: VGG Models

## VGG

- Is defined by the recurring presence of a convolution block, with pooling, with an increasing number of filters.

# ResNet

## Residual Layers

Information can skip certain weight layers, allowing for deeper networks.

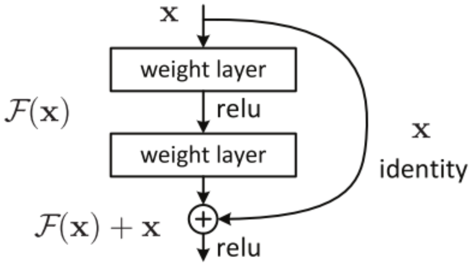



Figure 10: Skip Connection

# ResNet

| Depth | Train AUC | Val Loss | Val AUC |
|---|---|---|---|
| 32 (v1) | 0.7147 | 0.6356 | 0.6984 |
| 38 (v1) | 0.7051 | 0.6414 | 0.6986 |
| 70 (v1) | 0.7456 | 0.6128 | 0.7123 |
| 56 (v2) | 0.7796 | 0.71656 | 0.6820 |
| 110 (v2) | 0.8346 | 0.7056 | 0.6983 |

Table 3: ResNet Models

# Inception

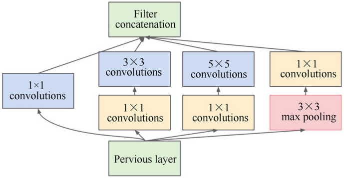
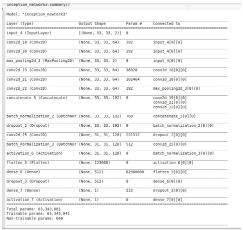



Figure 11: Basic Inception Module



inception_network3.summary()

Model: "inception_newtork3"

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_4 (InputLayer) | [(None, 33, 33, 2)] | 0 | |
| conv2d_18 (Conv2D) | (None, 33, 33, 64) | 192 | input_4[0][0] |
| conv2d_20 (Conv2D) | (None, 33, 33, 64) | 192 | input_4[0][0] |
| max_pooling2d_3 (MaxPooling2D) | (None, 33, 33, 2) | 0 | input_4[0][0] |
| conv2d_19 (Conv2D) | (None, 33, 33, 64) | 36928 | conv2d_18[0][0] |
| conv2d_21 (Conv2D) | (None, 33, 33, 64) | 102464 | conv2d_20[0][0] |
| conv2d_22 (Conv2D) | (None, 33, 33, 64) | 192 | max_pooling2d_3[0][0] |
| concatenate_3 (Concatenate) | (None, 33, 33, 192) | 0 | conv2d_19[0][0]<br>conv2d_21[0][0]<br>conv2d_22[0][0] |
| batch_normalization_2 (BatchNor | (None, 33, 33, 192) | 768 | concatenate_3[0][0] |
| dropout_2 (Dropout) | (None, 33, 33, 192) | 0 | batch_normalization_2[0][0] |
| conv2d_25 (Conv2D) | (None, 31, 31, 128) | 221312 | dropout_2[0][0] |
| batch_normalization_3 (BatchNor | (None, 31, 31, 128) | 512 | conv2d_25[0][0] |
| activation_6 (Activation) | (None, 31, 31, 128) | 0 | batch_normalization_3[0][0] |
| flatten_3 (Flatten) | (None, 123008) | 0 | activation_6[0][0] |
| dense_6 (Dense) | (None, 512) | 62980608 | flatten_3[0][0] |
| dropout_3 (Dropout) | (None, 512) | 0 | dense_6[0][0] |
| dense_7 (Dense) | (None, 1) | 513 | dropout_3[0][0] |
| activation_7 (Activation) | (None, 1) | 0 | dense_7[0][0] |

Total params: 63,343,681
Trainable params: 63,343,041
Non-trainable params: 640

Figure 12: Inception Module Example

# Inception

| | Train AUC | Val Loss | Val AUC |
|---|---|---|---|
| 1 module 3 cols | 0.7518 | 0.6891 | 0.6544 |
| 1 module 4 cols | 0.6692 | 0.6587 | 0.6742 |
| 2 modules 3 cols | 0.7231 | 0.6252 | 0.7156 |
| 2 modules 4 cols | 0.7274 | 0.6035 | 0.6952 |

Table 4: Inception Models

# XCeption

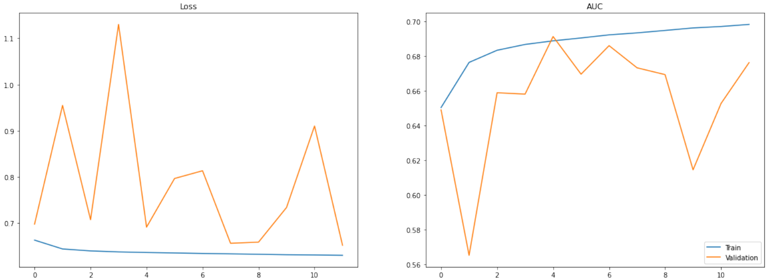



Figure 13: Xception results

# Table of Contents

# Final Word

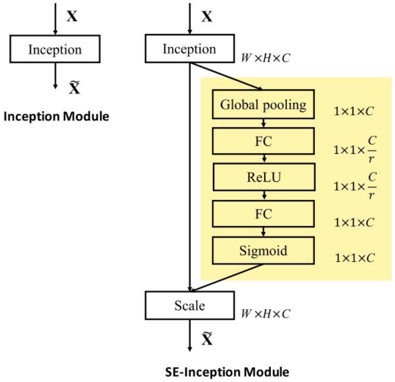



Figure 14: SeNet Inception Module

# Thank you!