Machine Learning Algorithms and Particle Physics Searches

João Pedro Pino Gonçalves[1]

Based on JHEP 01 (2021) **076**, JHEP 01 (2022) **154** and EPJC **82**, 826 (2022)

[1] Physics Department and Centre for Research and Development in Mathematics and Applications (CIDMA), University of Aveiro, Portugal.

3rd Workshop on Compact Objects, Gravitational Waves and Deep Learning - University of Minho

The Standard Model (**SM**) is the basis by which all subatomic interactions are described. **However**, some unanswered questions remain ...

- Inability to explain the observed particle spectra (family replication, masses and couplings hierarchies, neutrino masses);
- Lack of a Dark matter (**DM**) candidate;
- Hiearchy problem;
- $(g-2)_\mu$ anomaly, R_{K,K^*} anomalies, matter/anti-matter asymmetry;
- Naive quantization of gravity leads to a non-renormalizable theory, etc;

A simple observation $\Longrightarrow$ **SM is not the ultimate theory**

The story of SM is incomplete, **but we have not found anything new!** That means

- New physics is **heavy**, i.e., of the TeV-PeV order (or beyond);
- New physics is **weakly coupled to the SM**, i.e., low couplings;

As constraints on new physics become increasingly tighter, computational resources become more and more important.

We need

- **Powerful, reliable and proven** methods to deal with weak signals in the ocean of the SM background;
- The methods **must be comfortable** in dealing with **large** datasets.

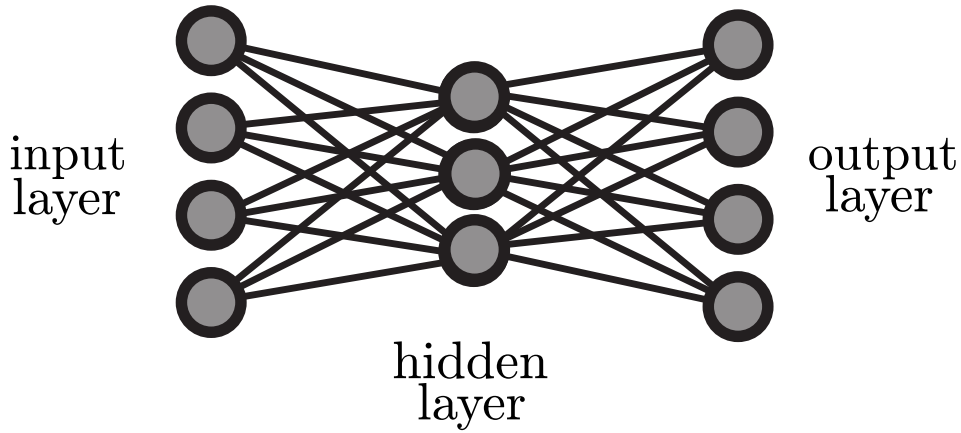
Machine learning to the rescue !

Deep Learning (DL) → Extracting high-level features from input data

Universal approximation theorem [Kurt Hornik, Neural Networks. **4** (2): 251–257]

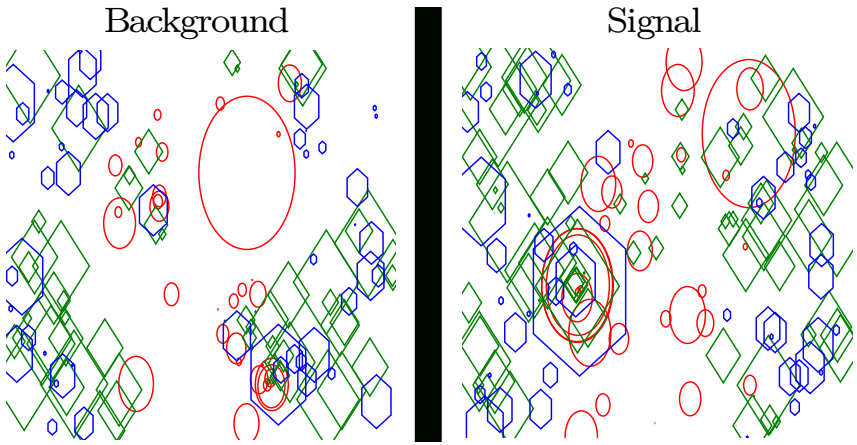
⇓

Approximate any function, for an arbitrary number of layers!



Detects small deviations in classes ⇒ Perfect for **classfication tasks**!

(a) For cats and dogs



(b) For detector images

```
neurons = 512
activ = 'sigmoid'
initl = 'RandomNormal'
loss  = 'binary_crossentropy'
metric = 'accuracy'
a1    = 1.e-7
a2    = 1.e-7
alp   = 0.1
nb_classes = 5

def NN_model():
    model = Sequential()
    #Input layer
    nn = model.add(Dense(neurons, input_dim=X_train.shape[1], kernel_initializer=initl,
                   kernel_regularizer=regularizers.l1_l2(l1=a1, l2=a2)))
    model.add(Activation(activ))

    #2nd layer
    model.add(Dense(neurons, kernel_initializer=initl,
              kernel_regularizer=regularizers.l1_l2(l1=a1, l2=a2)))
    model.add(Activation(activ))

    #3rd layer
    model.add(Dense(neurons, kernel_initializer=initl,
              kernel_regularizer=regularizers.l1_l2(l1=a1, l2=a2)))
    model.add(Activation(activ))

    #4th layer
    model.add(Dense(neurons, kernel_initializer=initl,
              kernel_regularizer=regularizers.l1_l2(l1=a1, l2=a2)))
    model.add(Activation(activ))

    # Output layer
    model.add(Dense(nb_classes, init=initl , activation=activ ))

    model.compile(loss=loss, optimizer=Adam(), metrics=[metric])
    return model
```

```
neurons = 512
activ = 'sigmoid'
initl = 'RandomNormal'
loss  = 'binary_crossentropy'
metric = 'accuracy'
a1    = 1.e-7
a2    = 1.e-7
alp   = 0.1
nb_classes = 5
```

- Number of neurons: **Arbitrary**;
- Activation functions: **10** + in `Keras` (with tunable parameters);
- Initializers: **10** + in `Keras` (with tunable parameters);
- ...

```
def NN_model():
    model = Sequential()
    #Input layer
    nn = model.add(Dense(neurons, input_dim=X_train.shape[1], kernel_initializer=initl,
                    kernel_regularizer=regularizers.l1_l2(l1=a1, l2=a2)))
    model.add(Activation(activ))

    #2nd layer
    model.add(Dense(neurons, kernel_initializer=initl,
              kernel_regularizer=regularizers.l1_l2(l1=a1, l2=a2)))
    model.add(Activation(activ))

    #3rd layer
    model.add(Dense(neurons, kernel_initializer=initl,
              kernel_regularizer=regularizers.l1_l2(l1=a1, l2=a2)))
    model.add(Activation(activ))

    #4th layer
    model.add(Dense(neurons, kernel_initializer=initl,
              kernel_regularizer=regularizers.l1_l2(l1=a1, l2=a2)))
    model.add(Activation(activ))

    # Output layer
    model.add(Dense(nb_classes, init=initl , activation=activ ))

    model.compile(loss=loss, optimizer=Adam(), metrics=[metric])
    return model
```

```
neurons = 512
activ = 'sigmoid'
initl = 'RandomNormal'
loss  = 'binary_crossentropy'
metric = 'accuracy'
a1    = 1.e-7
a2    = 1.e-7
alp   = 0.1
nb_classes = 5
```

A lot of free parameters to tune in architectural building

⇓

Calls for some optimization procedure → **Genetic algorithms!**

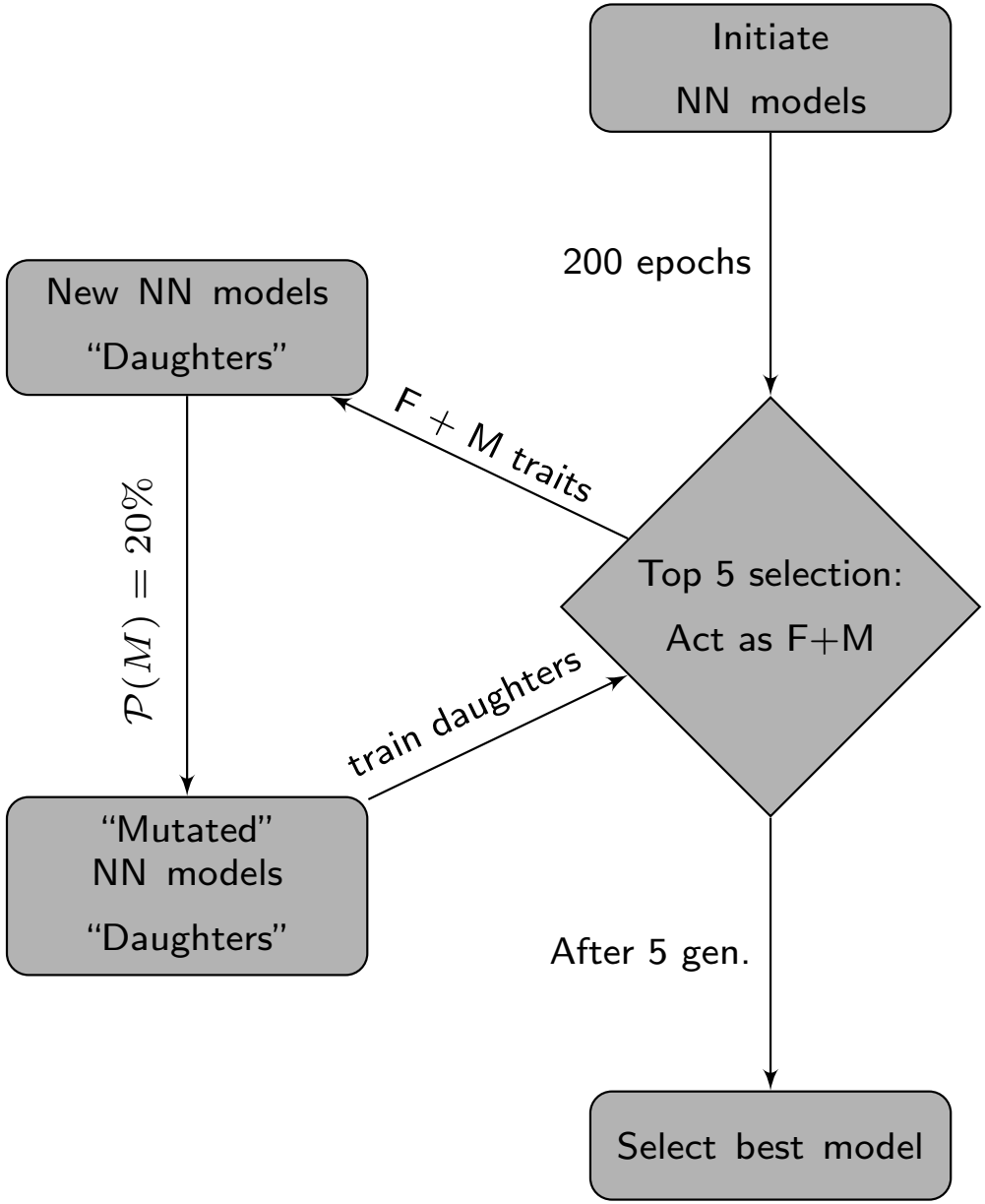
```
def NN_model():
    model = Sequential()
    #Input layer
    nn = model.add(Dense(neurons, input_dim=X_train.shape[1], kernel_initializer=initl,
                    kernel_regularizer=regularizers.l1_l2(l1=a1, l2=a2)))
    model.add(Activation(activ))

    #2nd layer
    model.add(Dense(neurons, kernel_initializer=initl,
              kernel_regularizer=regularizers.l1_l2(l1=a1, l2=a2)))
    model.add(Activation(activ))

    #3rd layer
    model.add(Dense(neurons, kernel_initializer=initl,
              kernel_regularizer=regularizers.l1_l2(l1=a1, l2=a2)))
    model.add(Activation(activ))

    #4th layer
    model.add(Dense(neurons, kernel_initializer=initl,
              kernel_regularizer=regularizers.l1_l2(l1=a1, l2=a2)))
    model.add(Activation(activ))

    # Output layer
    model.add(Dense(nb_classes, init=initl , activation=activ ))

    model.compile(loss=loss, optimizer=Adam(), metrics=[metric])
    return model
```

Algorithm Felipe F. Freitas et. al JHEP 01 (2021) 076:

- Randomly generate N models, by pooling a list of hyper-parameters;
- Train: Top 5 models are used to breed daughter networks;
- Add mutation probability. Train daughters and iterate the cycle.

Nice **advantages**:

- Simplifies network construction. Simple way to find the best hyperparameters;
- Straightforward way to maximize distinct metrics.



The best neural model is chosen based on two distinct metrics

- Asimov significance defined as

$$\mathcal{Z}_A = \left[2 \left((s+b) \ln \left(\frac{(s+b)(b+\sigma_b^2)}{b^2 + (s+b)\sigma_b^2} \right) - \frac{b^2}{\sigma_b^2} \ln \left(1 + \frac{\sigma_b^2 s}{b(b+\sigma_b^2)} \right) \right) \right]^{1/2}$$

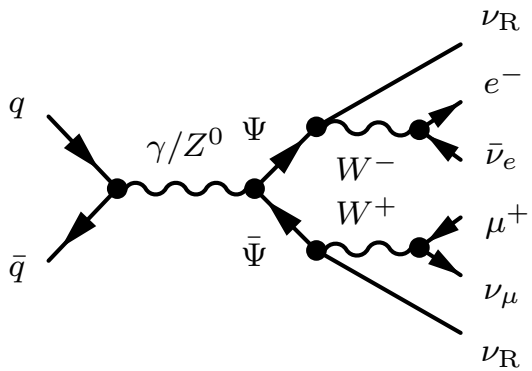
  Loss function is defined as $L = 1/(\mathcal{Z}_A + \epsilon)$. ϵ regularizes the loss function.
  Adam Elwood and Dirk Krücker arXiv:1806.00322

- Accuracy with binary cross-entropy loss function

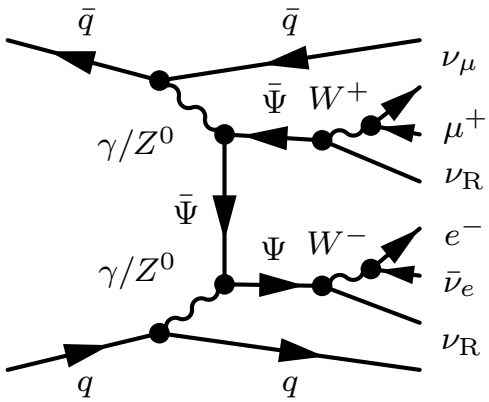
$$L = -\frac{1}{N} \sum_{i=1}^{N} y_i \log(p_{\hat{y}_i}) + (1 - y_i) \log(1 - p_{\hat{y}_i}),$$

  with N being the number of points, y the ground truth label (0 if background and 1 if signal) and p_{y_i} the probability of being signal.

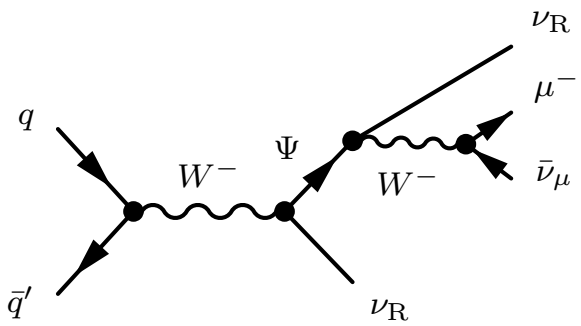
Single and pair-production topologies at the LHC. ν_{R} in the keV range and acts as missing energy.



(a) ZA topology



(b) VBF topology



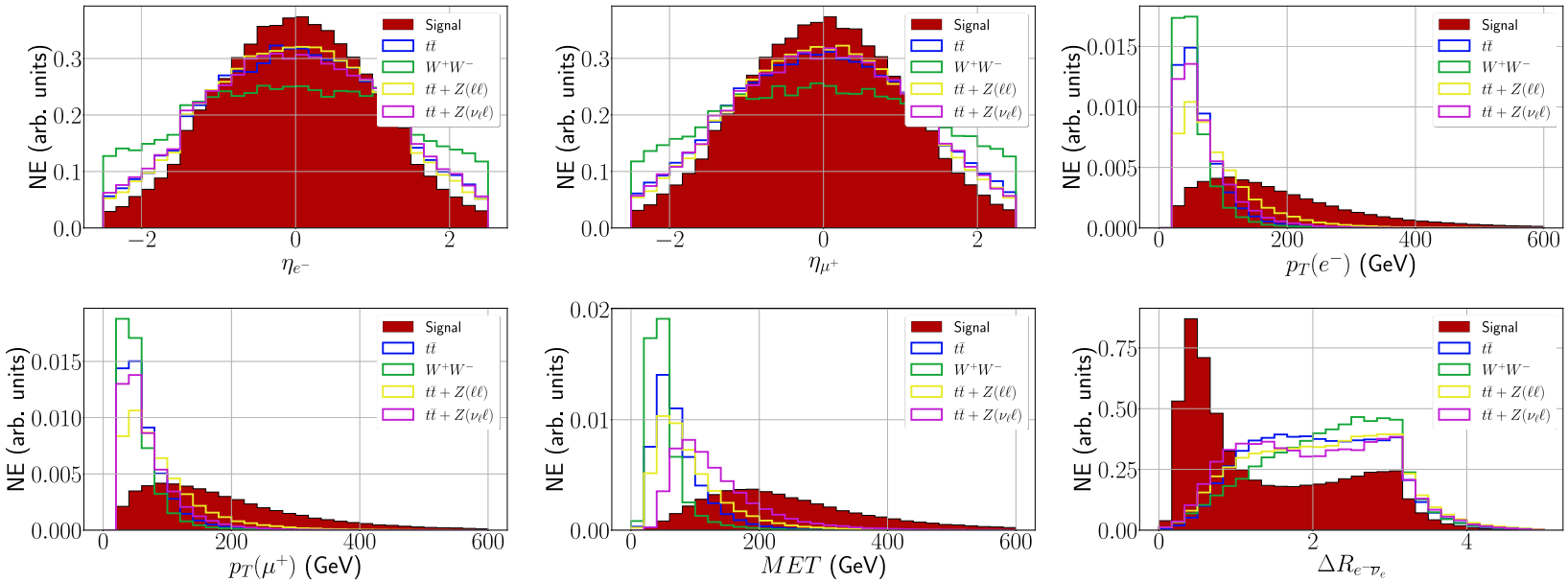
(c) VLBSM topology

For simplicity, we consider flavour opposite final states. Event selection via simple cuts:

1. Charged leptons with $p_T > 25$ GeV and $|\eta| \leq 5$;
2. Missing transverse energy $\not{E}_T > 15$ GeV;
3. Jets: $\Delta R = 1.0$, $p_T > 35$ GeV and $|\eta| \leq 5$.

Event generation flow: `SARAH` $\rightarrow$ `MadGraph` $\rightarrow$ `Pythia8` $\rightarrow$ `Delphes` $\rightarrow$ `ROOT`.

From paper: Phenomenology of vector-like leptons with Deep Learning at the Large Hadron Collider. **J. Gonçalves**, Felipe F. Freitas, António P. Morais, Roman Pasechnik. doi: 10.1007/JHEP01(2021)076

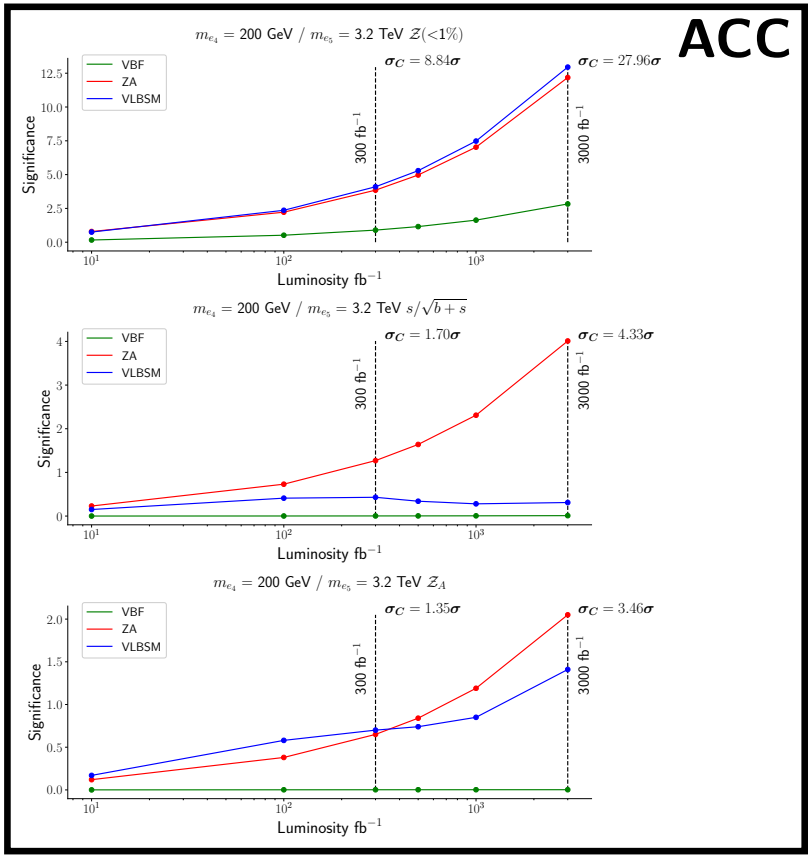


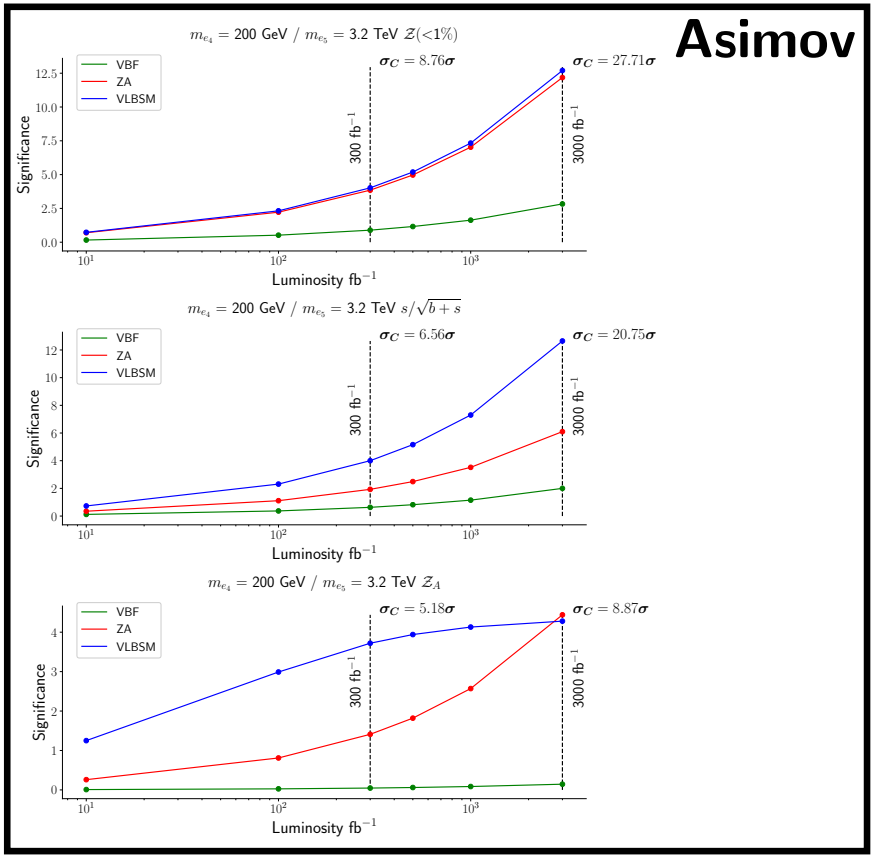
Feed the neural net high-level kinematics (mass distributions, pseudorapidity, transverse momentum, etc) for signal/background topologies.

Some cuts may be imposed to reduce backgrounds $\rightarrow$ **Unbalanced datasets!**

- Generate more Monte-Carlo: **Computational inefficient**;
- Oversample minority classes (e.g. `SMOTE` algorithm N. V. Chawla et. al JAIR: Vol 16, Issue 1, Jan. 2002);

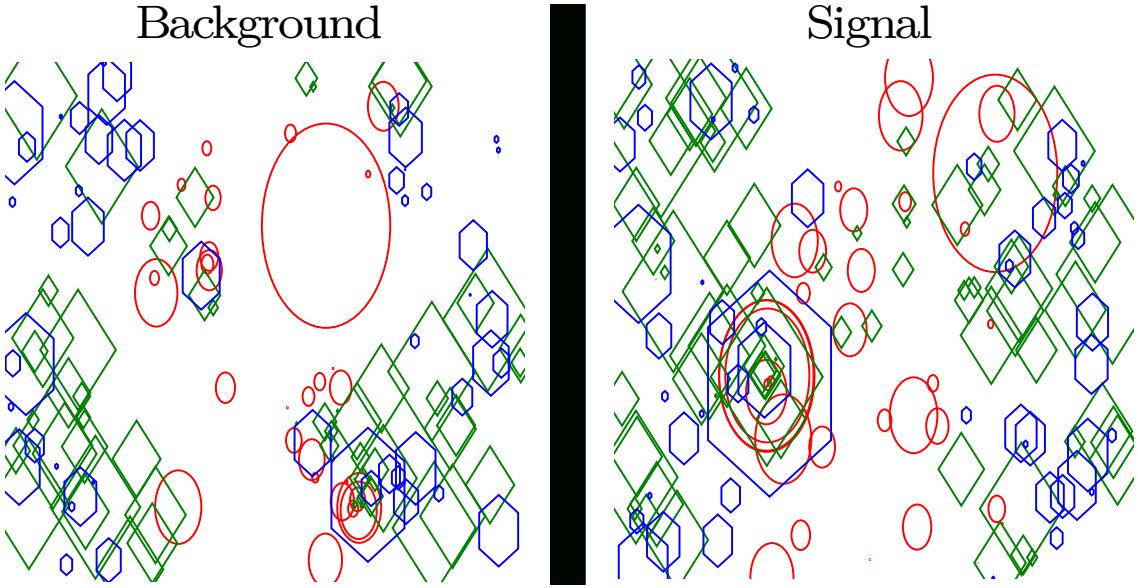
From paper: Phenomenology of vector-like leptons with Deep Learning at the Large Hadron Collider. **J. Gonçalves**, Felipe F. Freitas, António P. Morais, Roman Pasechnik. doi: 10.1007/JHEP01(2021)076





- Significance as a function of luminosity. $300\ \mathrm{fb}^{-1} \rightarrow$ Run-III;
- Utilizing the Asimov metric in the genetic algorithm, we can already obtain results above 5σ **for all** three metrics. We can **already probe them** at run-III of the LHC.

From paper: Phenomenology of vector-like leptons with Deep Learning at the Large Hadron Collider. **J. Gonçalves**, Felipe F. Freitas, António P. Morais, Roman Pasechnik. doi: 10.1007/JHEP01(2021)076

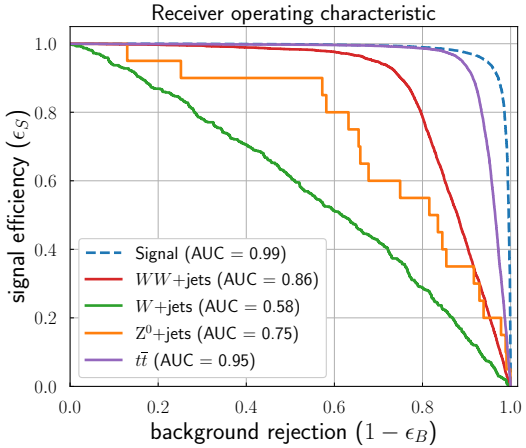


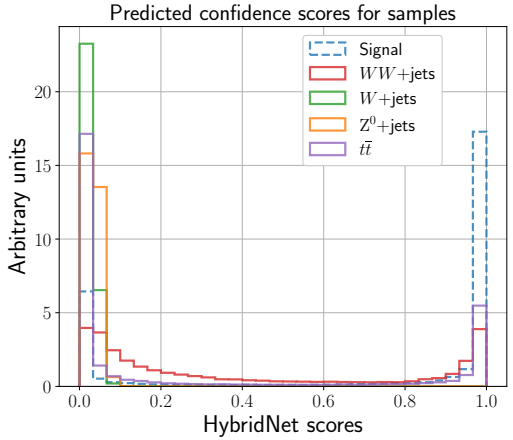
New physics at TeV order $\rightarrow$ Highly boosted decay products that needs to separated from multijet background at hadron colliders;

Jet Images: Associate energy deposited in the calorimeter with a pixel in the (η, ϕ) plane [Luke de Oliveira et. al JHEP 07 (2016) 069].

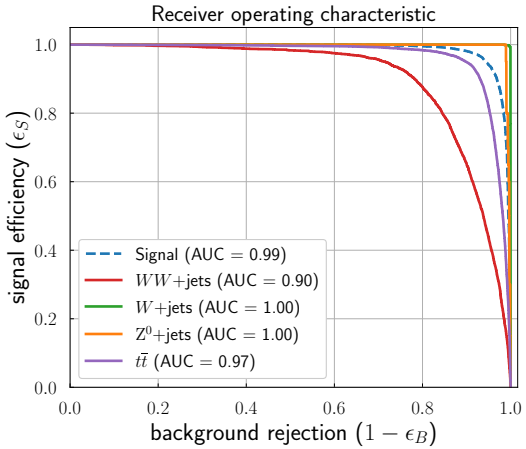
- Enhance classification with **jet kinematics** (multiplicity, mass, ΔR, ...);
- **Abstract Images**: Richer substructure;
- We have obtained efficiencies **greater** than only using kinematic data [Felipe F. Freitas et. al EPJC **82**, 826 (2022)].

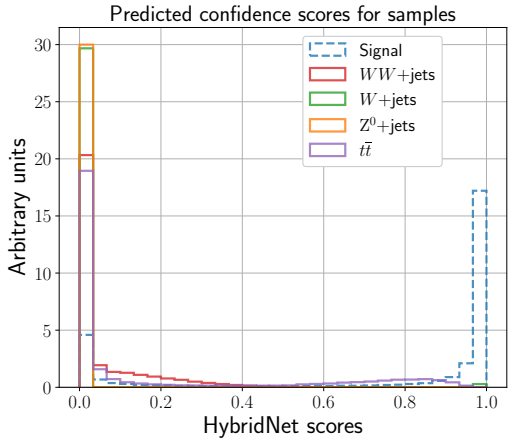
From paper: Phenomenology at the large hadron collider with deep learning: the case of vector-like quarks decaying to light jets. **J. Gonçalves**, Felipe F. Freitas, António P. Morais, Roman Pasechnik. doi: 10.1140/epjc/s10052-022-10799-8



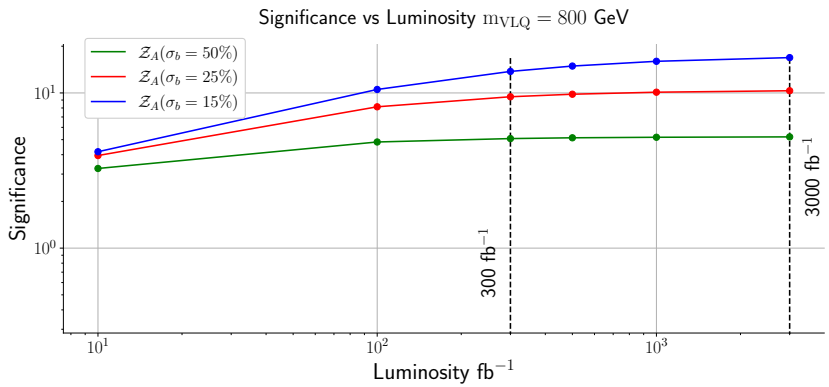


(d) Kinematic





(e) Kinematic + Abstract Images



1. Focusing on VLQ signatures for **decays into light jets**.
2. Use of **Abstract Images** heavily improves the **accuracy** of the neural network
3. Can exclude VLQs at the high-luminosity/run-III phase of the LHC, even for systematics of **50% !**

From paper: Phenomenology at the large hadron collider with deep learning: the case of vector-like quarks decaying to light jets. **J. Gonçalves**, Felipe F. Freitas, António P. Morais, Roman Pasechnik. doi: 10.1140/epjc/s10052-022-10799-8

To summarize . . .

- I have discussed how **Deep learning** algorithms can be used in collider phenomenology of generic BSM models;
- I shown these tools in action for various BSM models, including models with **vector-like fermions** of both quark and lepton types;
- For optimization of neural networks, I have presented a **genetic algorithm** that best maximize the statistical significance;
- In principle, the process itself is **model-independent**, in the sense that the neural models are agnostic to the BSM details. They only need the data.

Machine Learning Algorithms and Particle Physics Searches

Thank you for your attention

dfis universidade de aveiro departamento de física
CIDMA CENTRO DE I&D EM MATEMÁTICA E APLICAÇÕES CENTER FOR R&D IN MATHEMATICS AND APPLICATIONS
FCT Fundação para a Ciência e a Tecnologia
CENTRO 2020
PORTUGAL 2020
UNIÃO EUROPEIA Fundos Europeus Estruturais e de Investimento