Guide for VBF Analysis Through ANN

(Procedure Adopted on CMS-AN-18-120)

Miqueias Melo de Almeida

December 23, 2018

Abstract

This guide brings a brief description of the codes developed for the isolated VBF XS measurement using the reduced ntuples framework developed by prof. Nicola De Filippis. The guide covers the procedures developed for: building up Artificial Neural Networks (ANNs) using Keras, estimating Z+X background (thanks to Reham) and producing a statistical Analysis using Higgs Combine tool. This guide will describe what does and how to use the codes contained into the AnalysisCodes folder (which should contain this guide). Each python codes contains some explanation of their commands line so that the user can find more details. Feel free to also modify the structure of the codes according to your needs.

I ANNs Training using Keras an RECAS Computing Farm

In order to develop the Artificial Neural Networks (ANNs) a framework was developed using Keras, which is a set of python libraries that allows one to create any ANN structure (as known so far) using few lines of codes (all the objects defined using Keras can be fully hand-coded but this is time consuming and not the focus for Physics). For full details see `https://keras.io/` which is the Keras web page containing the full documentation about all functions and classes defined within this package. Keras needs another set of python libraries which handled the tensorial operations associated to the building and training of a ANN. Nowadays Keras accept two of those: the Tensorflow (which should become the official one in the future) and Theano (which is no longer maintained by its community - it'll become obsolete at some point). These configuration issues are not a trouble for the user anymore because CERN has adopted Keras into his set of libraries within the cvmfs environment. So, in order to have Keras working at RECAS one just needs to run the following commands:

Once Keras is ready the first thing to do is to prepare the datasets (events from MC) to be used as examples, to train the ANNs, and as probes, to test them. So, a python dictionary is a good object to store arrays containing subset of information from the reduced ntuples. The **Keras/FormatROOTs.py** and **Keras/runFormatROOTs.sh** macros are used for this purpose. What it does could be done in a single run but since it takes time is better to make this separately just once and not every time you run an ANN training. The runFormatROOTs.sh simplifies the running procedure of FormatROOTs.py which takes some commands line. The runFormatROOTs.sh is where the user should specify the list of MC files which one wants to use, the number of jets (since for VBF 2016 analysis there were created a separated ANN for 2 and 3 jets case). Fig. I shows how that macro looks like. Its commands line are:

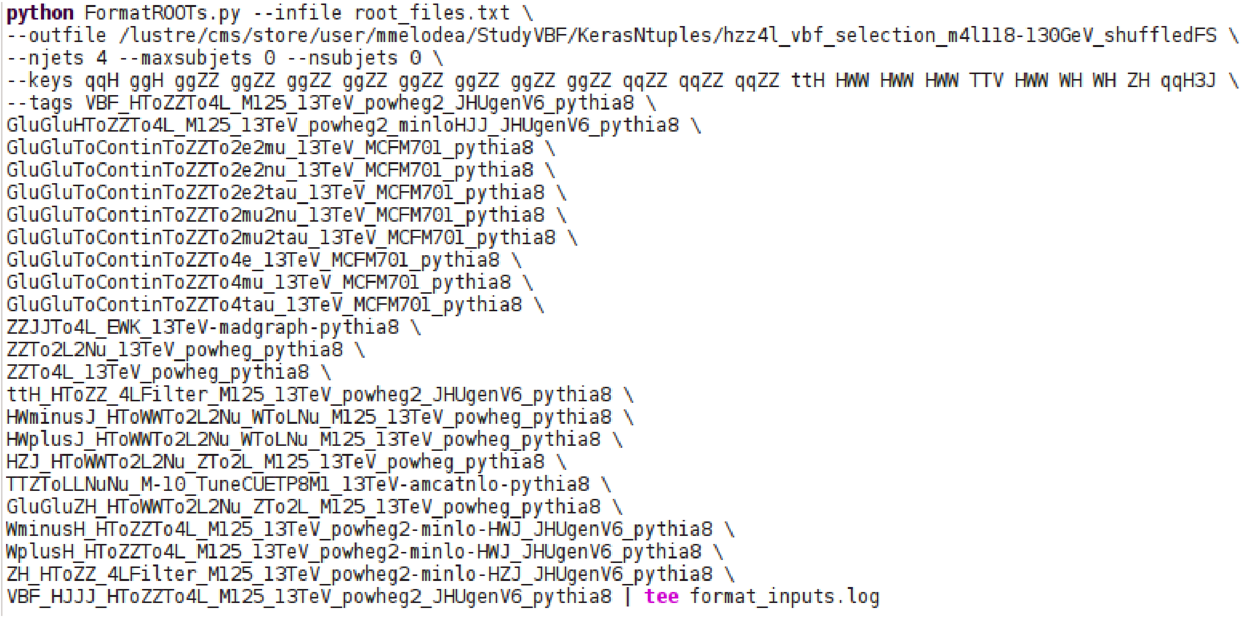
- infile: receives a list of the MC files name;
- outfile: the name of the output file used into Keras (the dictionary created by the code);
- njets: specifies the number of jets (one can creates separated dictionaries for each case, please see the FormatROOTs.py to understand how the jets are handled);
- maxsubjets and nsubjets: it can be ignored if there's no interest in studying jet image (leave the options set like in the figure for fast running);
- keys: are the names one wants to label each MC file within the dictionary;
- tags: are the names used to identify the MC files specified into the file passed to infile command line (be aware, they must be different for each file in order to avoid double counting - note that in the figure they are the names of the MC files).

After running runFormatROOTs.sh macro (like "bash runFormatROOTs.sh"), which takes about 10 min, a file with ".pkl" extension will be created in the folder where the macro was ran. Just as quick description of this kind of file, the python dictionary is a container storing arrays which indexes can be "names". So, for instance, once the dictionary is loaded, the p_T of the first lepton (which are sorted by p_T) can be accessed in a event using the syntax "array[0]['f_lept1_pt']" (the labels at the moment have been defined as the branch names of the variables stored in the TTree in the reduced ntuples). The splitting of the events have been defined in such way that one can get a certain fraction of each MC file and the events are randomized in order to mix the four-lepton final states and the MC process.

The next step is to run the ANNs training. With Keras the user writes his (her) own code and there are several parameters which needs to be set up. This means there are several configurations one can build up and, writing the code and training each of them separately is for sure not efficient (and also don't allow great coverage of the models that can be built). In order to be able to run several ANN models in parallel the **Keras/runEvaluateNeuralNetwork.sh** and **Keras/EvaluateNeuralNetwork.py** macros were created. The work similarly to previous ones. The ".sh" script is where the user specifies commands line used by the python code. The commands line accepted at this stage are (please, see the appendix to understand the meaning of the parameters here):

- infile: receives the address of the input file (the pkl dictionary);
- keys: receives the labels of the MC processes that one wants to use in the training (the MC not specified is not used, even if it's contained into the dictionary);
- nninputs: receives the variables to be used as inputs for the neural network;

Figure 1: A view of runFormatROOTs.sh macro.



```
python FormatROOTs.py --infile root_files.txt \
--outfile /lustre/cms/store/user/mmelodea/StudyVBF/KerasNtuples/hzz4l_vbf_selection_m4l118-130GeV_shuffledFS \
--njets 4 --maxsubjets 0 --nsubjets 0 \
--keys qqH ggH ggZZ ggZZ ggZZ ggZZ ggZZ ggZZ ggZZ ggZZ qqZZ qqZZ qqZZ ttH HWW HWW HWW TTV HWW WH WH ZH qqH3J \
--tags VBF_HToZZTo4L_M125_13TeV_powheg2_JHUgenV6_pythia8 \
GluGluHToZZTo4L_M125_13TeV_powheg2_minloHJJ_JHUgenV6_pythia8 \
GluGluToContinToZZTo2e2mu_13TeV_MCFM701_pythia8 \
GluGluToContinToZZTo2e2nu_13TeV_MCFM701_pythia8 \
GluGluToContinToZZTo2e2tau_13TeV_MCFM701_pythia8 \
GluGluToContinToZZTo2mu2nu_13TeV_MCFM701_pythia8 \
GluGluToContinToZZTo2mu2tau_13TeV_MCFM701_pythia8 \
GluGluToContinToZZTo4e_13TeV_MCFM701_pythia8 \
GluGluToContinToZZTo4mu_13TeV_MCFM701_pythia8 \
GluGluToContinToZZTo4tau_13TeV_MCFM701_pythia8 \
ZZJJTo4L_EWK_13TeV-madgraph-pythia8 \
ZZTo2L2Nu_13TeV_powheg_pythia8 \
ZZTo4L_13TeV_powheg_pythia8 \
ttH_HToZZ_4LFilter_M125_13TeV_powheg2_JHUgenV6_pythia8 \
HWminusJ_HToWWTo2L2Nu_WToLNu_M125_13TeV_powheg_pythia8 \
HWplusJ_HToWWTo2L2Nu_WToLNu_M125_13TeV_powheg_pythia8 \
HZJ_HToWWTo2L2Nu_ZTo2L_M125_13TeV_powheg_pythia8 \
TTZToLLNuNu_M-10_TuneCUETP8M1_13TeV-amcatnlo-pythia8 \
GluGluZH_HToWWTo2L2Nu_ZTo2L_M125_13TeV_powheg_pythia8 \
WminusH_HToZZTo4L_M125_13TeV_powheg2-minlo-HWJ_JHUgenV6_pythia8 \
WplusH_HToZZTo4L_M125_13TeV_powheg2-minlo-HWJ_JHUgenV6_pythia8 \
ZH_HToZZ_4LFilter_M125_13TeV_powheg2-minlo-HZJ_JHUgenV6_pythia8 \
VBF_HJJJ_HToZZTo4L_M125_13TeV_powheg2_JHUgenV6_pythia8 | tee format_inputs.log
```

- layers: receives the ANN topology (for instance, 10 10 10 means a ANN with 3 layers containing 10 neurons each);

- batchsize: receives the size of the batch sample of events used to train the events;

- patience: receives the number of epochs to keep training without improvement;

- scaletrain: receives a label to specify which kind of weight to be used when scaling the contribution of each MC process during the training (see appendix for more detail on the reason to use this option);

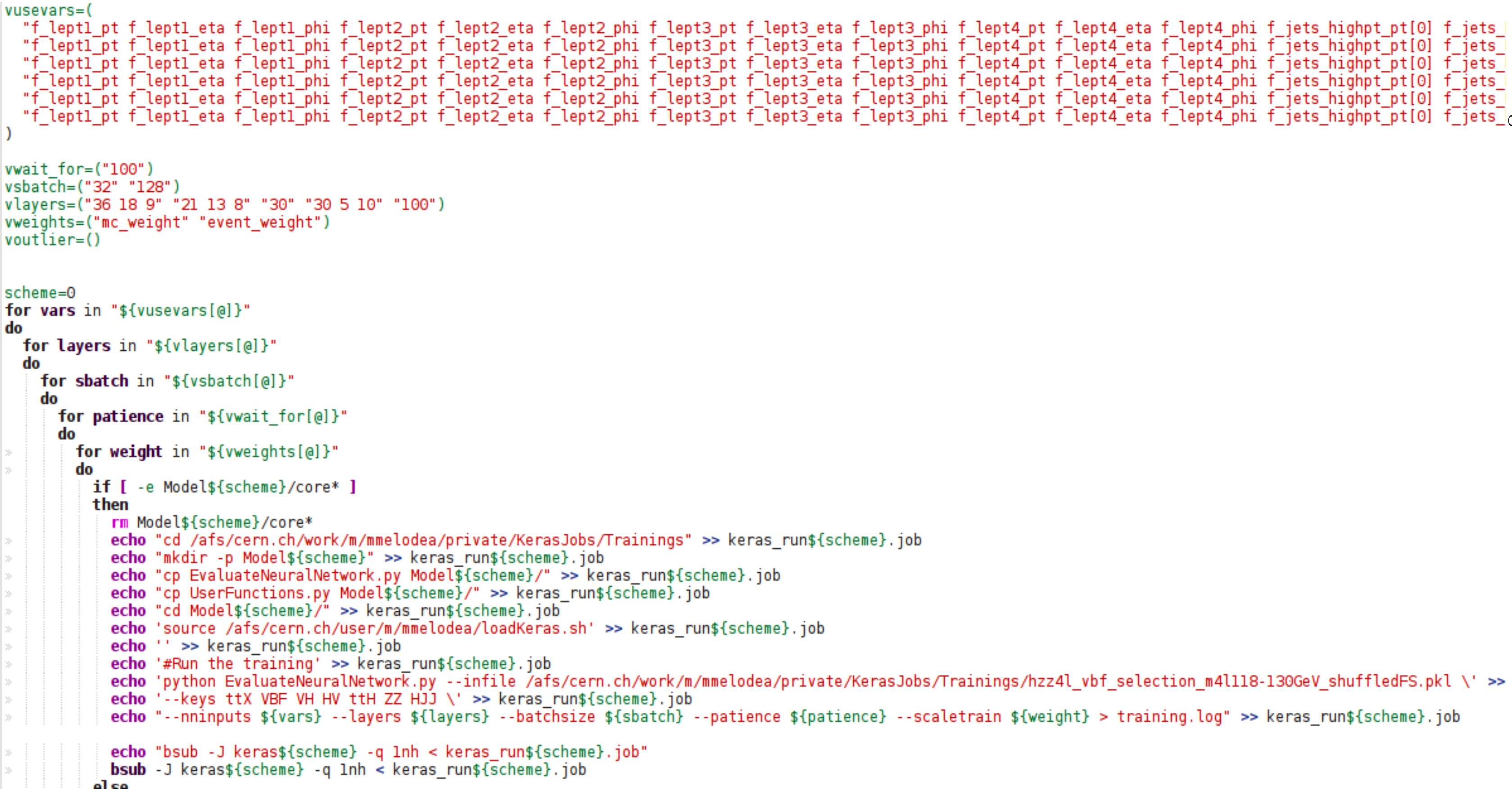
- neuron: receives the type of neuron to be used (the activation function, see Keras web page to see some examples). The ReLU and SeLU type are recommended.

As it can be seen in Fig. I, the ".sh" script makes a loop over any configuration the user wants and send multiples jobs to RECAS computing farm. It just needs to be specified in the arrays in the script. Once a jobs is finished, the folder containing such job will contain a set of plots which shows the performance of the trained ANN (which by the way, allows the user to check if there was overfitting on the ANN training - note that this trouble is avoided when ModelCheckPoint parameter, which is default in the macros, is used in Keras). Since the VBF analysis has the MELA discriminant from CMS, the plots show a comparison with MELA performance in the same MC events used to test the trained ANN. The script **Keras/checkTrainings.py** can be used to run over the dictionary also created at this stage. This dictionary contains variables describing the performance of each ANN and the script will outputs such performances in terms of the $\epsilon.\pi$ (efficiency.purity) and the significance of the signal.

Once the training of the ANN training is finished there will be a file with ".h5" extension in the jobs folder. This file is the one which contains the ANN function. This file can't be used to run a on-flying analysis. If the user wants to use it, then the procedure is to use a python code to store the ANN outputs per event into a new TTree. The **Keras/exportNetwork.py** macro was created to convert the Keras h5 format into a ".h" macro, which allows one to plug such file into another analysis codes. This macro is executed as (see an example of the ".h" file in the Keras folder)

python exportNetwork.py –infile keras_model –outfile output_file.

ATTENTION, at the epoch of this guide, the Keras package doesn't have the functionality to export the trained ANN in a given format that can be used on fly. The exportNetwork.py macro is not optimal and thus can't convert Keras ANNs trained using Dropout or other activation functions different from ReLU and SeLU.



```
vusevars=(
  "f_lept1_pt f_lept1_eta f_lept1_phi f_lept2_pt f_lept2_eta f_lept2_phi f_lept3_pt f_lept3_eta f_lept3_phi f_lept4_pt f_lept4_eta f_lept4_phi f_jets_highpt_pt[0] f_jets_
  "f_lept1_pt f_lept1_eta f_lept1_phi f_lept2_pt f_lept2_eta f_lept2_phi f_lept3_pt f_lept3_eta f_lept3_phi f_lept4_pt f_lept4_eta f_lept4_phi f_jets_highpt_pt[0] f_jets_
  "f_lept1_pt f_lept1_eta f_lept1_phi f_lept2_pt f_lept2_eta f_lept2_phi f_lept3_pt f_lept3_eta f_lept3_phi f_lept4_pt f_lept4_eta f_lept4_phi f_jets_highpt_pt[0] f_jets_
  "f_lept1_pt f_lept1_eta f_lept1_phi f_lept2_pt f_lept2_eta f_lept2_phi f_lept3_pt f_lept3_eta f_lept3_phi f_lept4_pt f_lept4_eta f_lept4_phi f_jets_highpt_pt[0] f_jets_
  "f_lept1_pt f_lept1_eta f_lept1_phi f_lept2_pt f_lept2_eta f_lept2_phi f_lept3_pt f_lept3_eta f_lept3_phi f_lept4_pt f_lept4_eta f_lept4_phi f_jets_highpt_pt[0] f_jets_
  "f_lept1_pt f_lept1_eta f_lept1_phi f_lept2_pt f_lept2_eta f_lept2_phi f_lept3_pt f_lept3_eta f_lept3_phi f_lept4_pt f_lept4_eta f_lept4_phi f_jets_highpt_pt[0] f_jets_
)

vwait_for=("100")
vsbatch=("32" "128")
vlayers=("36 18 9" "21 13 8" "30" "30 5 10" "100")
vweights=("mc_weight" "event_weight")
voutlier=()


scheme=0
for vars in "${vusevars[@]}"
do
  for layers in "${vlayers[@]}"
  do
    for sbatch in "${vsbatch[@]}"
    do
      for patience in "${vwait_for[@]}"
      do
        for weight in "${vweights[@]}"
        do
          if [ -e Model${scheme}/core* ]
          then
            rm Model${scheme}/core*
            echo "cd /afs/cern.ch/work/m/mmelodea/private/KerasJobs/Trainings" >> keras_run${scheme}.job
            echo "mkdir -p Model${scheme}" >> keras_run${scheme}.job
            echo "cp EvaluateNeuralNetwork.py Model${scheme}/" >> keras_run${scheme}.job
            echo "cp UserFunctions.py Model${scheme}/" >> keras_run${scheme}.job
            echo "cd Model${scheme}/" >> keras_run${scheme}.job
            echo 'source /afs/cern.ch/user/m/mmelodea/loadKeras.sh' >> keras_run${scheme}.job
            echo '' >> keras_run${scheme}.job
            echo '#Run the training' >> keras_run${scheme}.job
            echo 'python EvaluateNeuralNetwork.py --infile /afs/cern.ch/work/m/mmelodea/private/KerasJobs/Trainings/hzz4l_vbf_selection_m4l118-130GeV_shuffledFS.pkl \' >>
            echo '--keys ttX VBF VH HV ttH ZZ HJJ \' >> keras_run${scheme}.job
            echo "--nninputs ${vars} --layers ${layers} --batchsize ${sbatch} --patience ${patience} --scaletrain ${weight} > training.log" >> keras_run${scheme}.job

            echo "bsub -J keras${scheme} -q 1nh < keras_run${scheme}.job"
            bsub -J keras${scheme} -q 1nh < keras_run${scheme}.job
          else
```

Figure 2: A view of runEvaluateNeuralNetwork.sh macro.

II The Z+X Estimation

The Z+X estimation follows very close the procedure adopted in the analysis reported at AN-16-328 and few modifications were introduced in the codes in order to store the events used in the Z+X estimation. Such events present the same variables as the MC files which allows one to use them in the ANN training and for deriving the Z+X shape via the ANNs.

The first step for deriving the Z+X background is to compute the so called fake rates. In order to do that one needs to select samples of Z+lepton, where the Z are build using two tight leptons and the third lepton can be loose or tight (see AN-16-328 or AN-18-120 for the requirements defining the loose and tight leptons). The fake rate is then the ratio between the number of events with the third lepton being tight and the number of events with the third lepton being loose. For the VBF analysis it was decided to compute the fake rate as a function of the third lepton p_T and η. The second step is the application of the fake rate into two control regions of four-lepton events composed of two tight leptons plus, either two loose leptons (2P2F CR) or one tight and one loose lepton (3P1F CR).

The **ZplusX/ComputeFRandCRsOS.cc** macro is the macro which does this two steps in a single run. This macro needs to be compiled by running the script **ZplusX/compileference.sh**, which will produce the executable **ZplusX/ComputeFRandCRs**. This executable receives as arguments the name of the big ntuples (small ntuples obtained from MiniAODs and the ones from which the reduced ntuples are produced), the number of events before and after the HLT selections and the XS of the given process (if Data the argument should be set to 1). The script **ZplusX/submit_jobs.sh** can be used to send jobs to RECAS computing farm. Once the jobs are finished, then one needs to prepare the datasets for the Z+X estimation (see the **ZplusX/*Spring16*.txt**). The **ZplusX/IncludeKeras.cc** macro is used to insert in the TTrees produced for the Z+X estimation the ANN output for each event.

The Z+X estimation uses an equation (which can be found in the CMS HZZ4L ANs) that combines the contribution coming from two components: one from 2P2F and one from 3P1F. m_{4l} shapes in each of these CRs can be derived using the **ZplusX/plotCRs.cc** macro (run root -l -q $plotCRs.cc \backslash (\backslash"vbf \backslash", \backslash"4mu \backslash", \backslash"FRmap.root \backslash"\backslash)$ to plot those shapes[1]), where vbf stands for VBF-SR, 4mu is the final state and FRmap.root is the file containing the fake rate map) while the FR maps can be plotted using the **ZplusX/plotFR.cc** macro (run root -l $plotFR.cc \backslash (\backslash"MergedData.root \backslash", \backslash"MergedZZ.root \backslash"\backslash)$ to plot the FR maps), where MergedData.root and MergedZZ.root stands for the files created after merging the data and ZZ files produced when running the ComputeFRandCRs executable).

In order to properly derive yields and shapes, the Z+X equation have been implemented through histograms and the estimation ultimately uses only Data and MC events from ZZ processes. The macro plotCRs.cc is the one to be used, which will outputs on the terminal screen the values of Z+X in each four-lepton final state (4μ, $4e$, $2e2\mu$ and $2\mu2e$). The final numbers into $4l$ were estimated by combining the yields derived in each final state. The $4l$ yields are the sum of the yields in each final state and its uncertainty has been estimated by quadratically summing their uncertainty. Also, note that, for final states where the muons are the loose leptons (4μ and $2e2\mu$) the Z+X is not derived from the full equation but instead it is just the component coming from 2P2F. One can compute the final Z+X yields and uncertainties by hand but there's is the **ZplusX/computeFinalZX.cc** which can be used for that, the user just needs to state on it the yields given by the plotCRs.cc macro.

The derivation of Z+X shapes for a given trained ANN is straightforward since the same histograming procedure mentioned above is executed. The events selected in each of the CRs are just fed to the ANN and the histograms are used. Please, check the plotCRs.cc macro and the Z+X equation to better understand this procedure. Since the events selection for the Z+X estimation takes time, the **ZplusX/IncludeKeras.cc** macro has been created to produce a new version of the output files of the ComputeFRandCRs executable, containing the ANN output value for each event. So, this macro should be ran before the plotCRs.cc macro.

Once these steps have been executed the last thing to do is just to merge the files created. The **ZplusX/combineOSSF.sh** can be used for that. Please, take a look to see how to properly merge the files in order to have the appropriate histograms.

III Statistical Analysis

The statistical analysis has some complex theoretical background and it will not be mentioned here. For complete details please check the CMS-AN-18-120 and the Higgs Combine tool web page. The steps discussed in this section

[1] The "vbf" label can be replaced by "smhiggs" in order to plot the same shapes from the events after SM Higgs selections.

comprehends then, the proposed procedure to estimate the uncertainty on the 3rd jet in the events, the uncertainty on the ANN shape due to the uncertainty on its inputs and the procedures adopted to get some statistical results (focused here on HybridNew method which is the recommended one by the LHC Higgs Combination Group). The summary of the commands and procedures from Z+X estimation to the obtainment of limits are (note that you can adapt them, they just reflect how they were executed previously):

1 run plotCR.cc to generate Z+X shapes;

1.1 copy files to vbf folder and make the merging of the generated files running combineOS*;

2 run plotObjectsProperties.cc to generate Data and remaining BKGs shapes and, create a file including them and Z+X;

2.2 copy the Histograms.root file locally;

3 run plot3rdJetUncertainty.cc to generate the ANN shapes for each channel in the Njets3 category from VBF-H2J and VBF-H3J, getting their ratio (= to the uncertainty)

4 run checkUnc1Sigma*.cc to generate all process and their +/- 1sigma shifts (first set of files used in the limits procedure);

4.1 merge the generated files (hadd Distribution*.root)

5 run create_datacards.sh, which calls the genInputs*.py creating the *.txt and *.root datacards needed for limits;

6 run combine_cards.sh to combine the cards using combine tool;

7 run Limits:

7.1 Best fit: combine ./../combinedCard_Higgs125GeV_VBFStudy_k57nj2_k24nj3_4l.txt -M FitDiagnostics –plots –saveOverallShapes –saveWithUncertainties ¿& combineResults.log

7.2 Best fits: combine ./../combinedCard_Higgs125GeV_VBFStudy_k57nj2_k24nj3_4l.txt -M ChannelCompatibilityCheck –saveFitResult ¿& combineResults.log

7.3 Obs Likelihood: combine ./../combinedCard_Higgs125GeV_VBFStudy_k57nj2_k24nj3_4l.txt -M MultiDimFit -n Obs –algo=grid –points 300 ¿& combineResults.log

7.4 Exp Likelihood: combine ./../combinedCard_Higgs125GeV_VBFStudy_k57nj2_k24nj3_4l.txt -M MultiDimFit -n Exp –algo=grid –points 300 -t -1 –expectSignal=1 ¿& combineResults.log

7.5 Impact plot:

7.5.1 text2workspace.py ../combinedCard_Higgs125GeV_VBFStudy_k57nj2_k24nj3_4l.txt -m 125

7.5.2 python combineTool.py -M Impacts -d ../combinedCard_Higgs125GeV_VBFStudy_k57nj2_k24nj3_4l.root -m 125 –doInitialFit –robustFit 1

7.5.3 python combineTool.py -M Impacts -d ../combinedCard_Higgs125GeV_VBFStudy_k57nj2_k24nj3_4l.root -m 125 –robustFit 1 –doFits

7.5.4 python combineTool.py -M Impacts -d ../combinedCard_Higgs125GeV_VBFStudy_k57nj2_k24nj3_4l.root -m 125 -o impacts_obs_4l_k57nj2_k24nj3.json

7.5.5 python plotImpacts.py -i impacts_obs_4l_k57nj2_k24nj3.json -o impacts_obs_4l_k57nj2_k24nj3

7.6 ExpHybridLimMedian: combine ./../combinedCard_Higgs125GeV_VBFStudy_k57nj2_k24nj3_4l.txt -M HybridNew –LHCmode LHC-limits -H AsymptoticLimits -T 50000 –expectedFromGrid=0.5 –expectSignal=1 ¿& combineResults.log

7.7 ExpHybridLimObs: combine ./../combinedCard_Higgs125GeV_VBFStudy_k57nj2_k24nj3_4l.txt -M HybridNew –LHCmode LHC-limits -H AsymptoticLimits -T 50000 ¿& combineResults.log

7.8 ExpHybridSigMedian: combine ./../combinedCard_Higgs125GeV_VBFStudy_k57nj2_k24nj3_4l.txt -M HybridNew –LHCmode LHC-significance -H Significance -T 50000 –expectedFromGrid=0.5 –expectSignal=1 ¿& combineResults.log

7.9 ObsHybridSigMedian: combine ./../combinedCard_Higgs125GeV_VBFStudy_k57nj2_k24nj3_4l.txt -M HybridNew –LHCmode LHC-significance -H Significance -T 50000 ¿& combineResults.log

For plotting there are some macros in the package and the use as inputs the datacard root files created for the statistical analysis and the root files created by combine when running the previous commands. Their name should be self-explaining with the terminology used within Higgs Combine tool. For extra details on this steps please see the recent workshop realized at Fermilab (some codes were derived based on it): `https://indico.cern.ch/event/747340/timetable/`.

I Appendix: Keras Parameters Meaning

- **pre-processing**: it is a usual practice in Machine Learning field to apply some operation in the NN inputs in order to standardize them (keep them in the same range of values for instance). In the scans done the pre-processing showed negligible effects and it was decided to not adopt such procedure for further studies. Note that, when pre-processing is applied in the training it will be need to also apply the same procedure when using the trained NN (otherwise the results will be mistaken, since the NN are expecting preprocessed inputs). One also should note that preprocessing changes the input variables what may destroy the possibility of the NN reconstruct some important property(e.g. invariant masses);

- **Topology**: the architecture of the NN, that is, the number of hidden layers, neurons and the neuron type. In Keras there are several types of neuron which even includes possible learning parameters (during training). The *Rectified Linear Unity* (ReLU) is the most recommended due to its property of non-vanishing gradient;

- **Batch size**: the number of events in a subset from the training set used to compute the gradients and update the w_i's and b in each neuron;

- **Epochs**: the number of iterations over the full training set. The total number of iterations is a combination of the size of the training set, the batch size and the number of epochs. For instance, setting a training of 10 epochs and a batch size of 10 for a training set size of 100 means that Keras performs 10^2 updates on the w_i's and b's;

- **Early stop**: a parameter to set the number of epochs which Keras should wait if not improvement in the loss function is observed. If still not improvement is seen after that number of epochs the training is stopped;

- **Minimizer**: is the method to compute the gradients. There are several options in Keras (SGD, Adam, RMSprop, etc.) and after testing most of them it was decided to keep Adam. Adam stands for Adaptive Momentum and has the property of fast convergence due to a NN updating directed towards the loss function minimum. This minimizer is widely applied in ML studies;

- **Scaling**: Keras allows one to scale the loss function by some weight that can be independent for each training example or the same for a entire class (signal or background, for instance). It was tested the impact of using the expected yield (XS) of each process and the individual weights of each event ($\sigma.\epsilon.BR$).

II Appendix: Scaling Events Contribution in the Training

In the beginning of this analysis the NN trainings were carried out without taking into account any kind of event weight. In such way all events are seen by the NN in an equal basis. Keras has some features that allows one to include scale factor (example weight) which are in general used to balance the training when the number of events from different classes are very different. In the physics scenario one could use the individual MC event weights ($\sigma.\epsilon.BR$) or yet the sum of those weights (which constitutes the expected yields). The advantage of the first approach is that some events, from signal or background, crossing the classes zones defined by the NN could have small weights and would be worthy to allow them to come in with the benefit to possibly improve the final discrimination. The disadvantage is that the individual weights might not keep the hierarchical contribution of each process since the

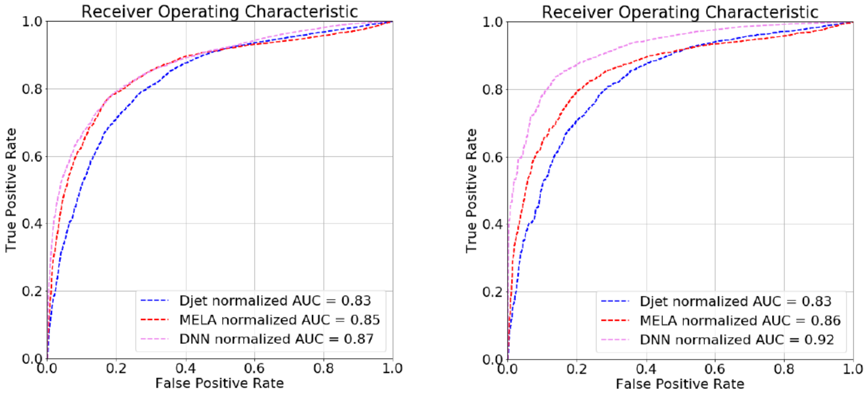


Figure 3: ROC curves comparing the performance of a trained NN, the VBF MELA discriminant and D_{jet}: (a) the performance of NN without scaling the *loss* and (b) performance after scaling the *loss*.

expected cross-section is divided by the number of events (such that if one have a large number of events the weights might become smaller for important processes than the ones for other small processes).

Both approaches were tested. The weight is used by Keras as a scale factor for the *loss* function such that during the training each example is seen by the NN with a different importance. That affects the direction in which the *minimizer* computes the gradients of the *loss* function. The advantage of doing so is that even if a process has few events, which is very frequent in many analysis, it will be properly taken into account with respect to the other process having more events. The Fig. 3 shows the impact of scaling the *loss*. In the beginning of the analysis the sample ggH-minlo was produced slightly different from the other samples: the leptons were sorted by p_T in the 4μ final state. Without weighting the events during the training the typical ROC curves observed were like the one showed in Fig. 3 (left). Once the weights were introduced the discrimination of VBF and ggH processes increased significantly as showed in Fig. 3 (right). Although this is not a physical property and thus had to be fixed, this shown the importance of scaling the events contribution. It shows that the NN may not learn some features from a process if a scaling is not applied. ggH has the biggest yield and has a relatively good number of events but still without weighting it the small difference was not taken into account by the NN. Since there all the further developed studies were done with the events weighted.