[1]:

```
import h5py
import matplotlib.pyplot as plt
import numpy as np
import tensorflow.keras as K
import pandas as pd
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import GridSearchCV
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R)
SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math Kernel
Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(R) AVX)
instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R)
SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math Kernel
Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(R) AVX)
instructions.
```

```
2024-04-16 23:38:19.943206: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: SSE4.1 SSE4.2, in other operations,
rebuild TensorFlow with the appropriate compiler flags.
```

1 Loading the Data

[2]:

```
with h5py.File("info/data/output_signal.h5", "r") as file:
    signal_data = file["events"][:]

with h5py.File("info/data/output_bg.h5", "r") as file:
    bg_data = file["events"][:]
```

[3]:

```
# storing signal and background data in panda DataFrame
signal = pd.DataFrame(signal_data)
background = pd.DataFrame(bg_data)

# concatenating the data frames to be part of one big set
df = pd.concat([signal, background])

# reseting the indicies
df = df.reset_index()

# creating the labels for the data sets i.e; signal = 1, background = 0 for␣
 ↪classification
labels = np.concatenate([np.ones(signal.shape[0]), np.zeros(background.
 ↪shape[0])])
labels = pd.DataFrame({'ttZ': labels})
```

```python
# adding labels as a column at the end of the DataFrame
df = df.join(labels)

# shuffling the DataFrame
df_shuffled = df.sample(frac=1, random_state=42)  # 'random_state' for␣
 ↪reproducibility
df_shuffled.head()


# taking the first 30 000 rows from the shuffled DataFrame
subset_df = df_shuffled.iloc[:30000]

# splitting the labels from the rest of the dataset
y = subset_df['ttZ']

# we can check the class distribution in the subset
print('ttZ events: {:.2f}%'.format(np.sum(y)/len(y) * 100))
print('WZ events: {:.2f}%'.format((1 - np.sum(y)/len(y)) * 100))
```

```
ttZ events: 68.69%
WZ events: 31.31%
```

2 Various Feature Lists

[4]:

```python
# base input list suggested by the project
input_list = [ "H_T",
              "jet_1_pt",
              "jet_2_pt",
              "lep_1_pt",
              "lep_2_pt",
              "n_bjets",
              "jet_1_twb",
              "jet_2_twb",
              "bjet_1_pt"]

# best input list suggested by SRS
input_list_2 = ['jet_1_pt',
                'jet_3_pt',
                'jet_1_eta',
                'jet_3_eta',
                'jet_1_twb',
                'jet_2_twb',
                'jet_3_twb',
                'bjet_1_pt',
                'n_jets',
                'n_bjets',
                'n_leptons']
```

```
# smallest input list that still has an accuracy >80%, suggested by SRS
input_list_3 = ['jet_3_pt',
                'jet_1_twb',
                'jet_2_twb',
                'jet_3_twb',
                'bjet_1_pt',
                'n_jets',
                'n_bjets']

input_lists = [input_list, input_list_2, input_list_3]
```

3 Grid Search through Various Parameters

[5]:

```
# model for classification
def build_model(lr=0.002, layer1=25, layer2=12, layer3=5):
    model = K.Sequential([
        preprocessing_layer,
        K.layers.Dense(layer1, activation="relu"),
        K.layers.Dense(layer2, activation="relu"),
        K.layers.Dense(layer3, activation="relu"),
        K.layers.Dense(1, activation="sigmoid")
    ])
    model.compile(optimizer=K.optimizers.Adam(learning_rate=lr),␣
 ↪loss='binary_crossentropy', metrics=['accuracy'])
    return model

# wrapper function
model = KerasClassifier(model=build_model, verbose=0, epochs = 30)

# Parameters to search through
param_grid = {'model__optimizer__lr': [0.002, 0.0002, 0.00002],
              'model__layer1': [25, 50, 100],
              'model__layer2': [12, 25, 50],
              'model__layer3': [5, 10, 15],
              'batch_size': [100, 150, 300]}

# Create GridSearchCV
gridCV = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1, cv=3,␣
 ↪verbose = 1)

# GridSearch through all parameters and all suggested feature lists
names = ['base list', 'SRS: best list', 'SRS: best smallest list']
for i, feature_list in enumerate(input_lists):
    X = subset_df[feature_list]
    preprocessing_layer = K.layers.Normalization()
    preprocessing_layer.adapt(X)
    grid_result = gridCV.fit(X, y)

# saving results into a DataFrame
    if i == 0:
        df = pd.DataFrame.from_dict(grid_result.best_params_, orient='index')
        df.loc['best score'] = grid_result.best_score_
        df = df.rename(columns={0: names[i]})
    else:
        df2 = pd.DataFrame.from_dict(grid_result.best_params_, orient='index')
        df2.loc['best score'] = grid_result.best_score_
        df2 = df2.rename(columns={0: names[i]})
        df = pd.concat([df, df2], axis=1)

df.to_csv('model_selection.csv', index=True)
df
```

```
Fitting 3 folds for each of 243 candidates, totalling 729 fits
Fitting 3 folds for each of 243 candidates, totalling 729 fits
Fitting 3 folds for each of 243 candidates, totalling 729 fits
```

[5]:

```
                      base list  SRS: best list  SRS: best smallest list
batch_size             300.0000        100.0000                 150.0000
model__layer1           50.0000         25.0000                  50.0000
model__layer2           12.0000         25.0000                  50.0000
model__layer3           10.0000         15.0000                   5.0000
model__optimizer__lr     0.0020          0.0020                   0.0002
best score               0.8296          0.8339                   0.8270
```