Neural Network Classification of Top Quark Production at the Large Hadron Collider

Raveel Tejani

April 17, 2024

PHYS 310

University of British Columbia

Contents

List of Tables

List of Figures

1 Introduction

At the Large Hadron Collider (LHC), where protons are collided at a staggering rate of 40 MHz across various points within the LHC ring, the ATLAS experiment plays a pivotal role as a general-purpose detector at one of these collision sites. It precisely tracks particles emitted from these collisions, an essential task for understanding fundamental physics. The integration of machine learning, specifically neural networks, is instrumental in distinguishing between different physics processes: "signals" and the more prevalent "background" events. Achieving accurate discrimination between these is crucial due to the relatively rare occurrence of signal events. The focus of this project was to explore several key objectives within the context of high-energy physics:

1. Training a Classifier: Develop a neural network classifier to accurately distinguish between ttZ (the production of a pair of top quarks associated with a Z boson) and WZ (the production of a W and a Z boson) events. This distinction is critical as both ttZ and WZ can produce similar detector-level features, yet represent different underlying physics processes.

2. Feature Selection: Experiment with the addition and removal of various detector-level features to evaluate how each alteration affects the classifier's performance. This analysis helps in understanding which features are most predictive and thus critical for accurate event classification.

3. Optimization of Neural Network: Enhance the architecture and training parameters of the neural network to maximize its performance in classifying ttZ and WZ events. This includes adjustments in network structure, learning rates, and other hyperparameters to refine the model efficacy.

4. Feature Ranking: Determine and rank the importance of these features in the neural network to prioritize data collection and processing strategies at the detector level.

To accomplish these goals, simulated data samples of ttZ and WZ events, produced using Monte Carlo simulations, were utilized to train the neural network. This training enables the classifier to learn the distinctive patterns of each event type. Once optimized, the classifier is then applied to actual recorded data from the LHC, using its outputs for further statistical analysis and insights. This project not only advances the accuracy and efficiency of particle identification at the ATLAS experiment but also contributes significantly to the broader field of particle physics, enhancing our understanding of particle interactions and the fundamental forces of nature.

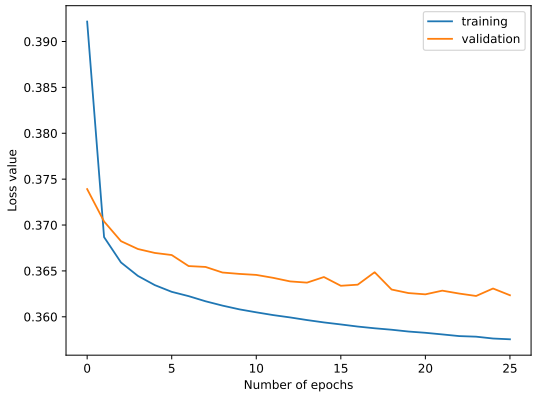
2 Methods

The initial phase of the project involved utilizing a predefined neural network (NN) model, constructed using Keras's Sequential API, for the classification of ttZ and WZ events. The base model was configured to operate with a subset of features, specifically 9 out of a possible 18 features, selected based on prior analyses which identified them as significant. We will perform the following steps to hopefully enhance the model's performance:

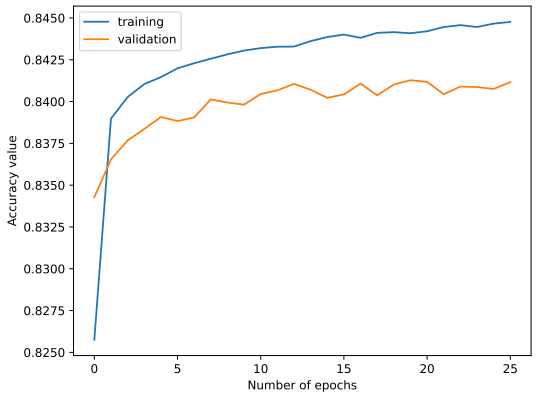
1. **Feature Selection:** To enhance the model's performance, a Sequential Feature Selector (SFS) was employed to methodically identify the most impactful features. This process resulted in two new feature sets: the "best" feature list comprising 11 of the 18 total features, and a "small" feature list which achieved comparable accuracy with only 7 out of 10 features. These subsets were chosen based on their ability to optimize the predictive accuracy of the models while minimizing complexity. Subsequent to feature selection, a comparative analysis was conducted using the base model structure with three different sets of features: the original base feature list, the newly identified best feature list, and the small feature list. This comparison was aimed at determining the impact of each feature set on model performance in terms of accuracy and efficiency.

2. **Parameter Optimization via GridSearchCV:** Following the feature comparison, GridSearchCV was implemented to systematically explore and optimize several hyperparameters for each model variant (base, best feature list, and small feature list). Each of the three feature sets was then evaluated using their respective optimized models, as determined by the GridSearchCV results. The effectiveness of each model was assessed based on the optimized feature list and model parameters, focusing on classification accuracy and computational efficiency.

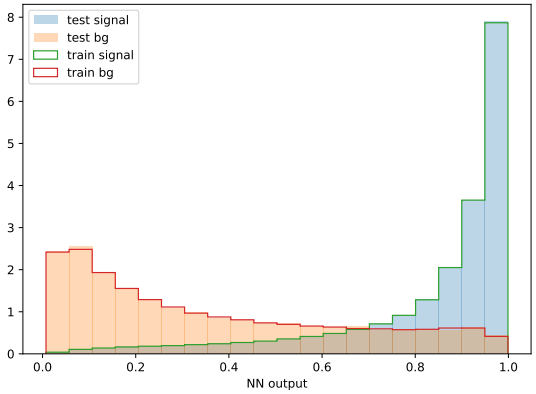
3. **Feature Importance Analysis:** Finally, to understand the contribution of individual features within the optimized models, a permutation feature importance analysis was performed. This method involved systematically altering the values of each feature in the dataset and observing the resultant impact on model performance. This analysis helped to highlight which features were most critical to the model's predictive capabilities, providing insights into the underlying data structure and the physics phenomena being modeled.



(a) Loss reduction as a function of epochs



(b) Accuracy performance as a function of epochs



(c) Probability distribution of test and train cases, based on trained Neural network

Figure 1: Perfromance of The Base Neural Network Model Using Base Feature Selection. (a) Loss curves for the training and validation datasets, demonstrating model convergence. (b) Accuracy scores for the training and validation datasets, reflecting model effectiveness. (c) NN output distributions for both training and testing datasets, highlighting the model's differentiation between signal and background classes.

3 Results

3.1 Feature Selection

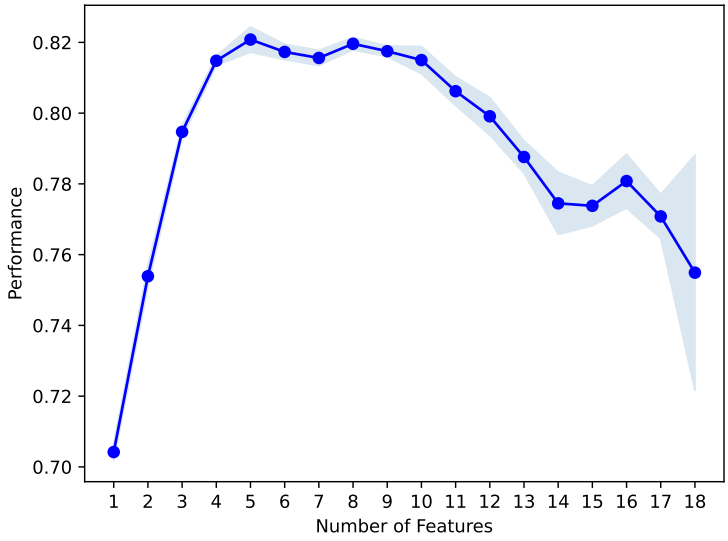
In this subsection, we present our findings from applying Sequential Feature Selection (SFS) to our dataset. SFS, while powerful, is notably computationally intensive as it involves multiple iterations over the feature list. The process begins with a model devoid of any features, progressively incorporating features based on their contribution to performance enhancement. Given our dataset includes 18 features, this amounts to 171 evaluations:

$$18 + 17 + ... + 1 = \frac{18 \times 19}{2} = 171$$

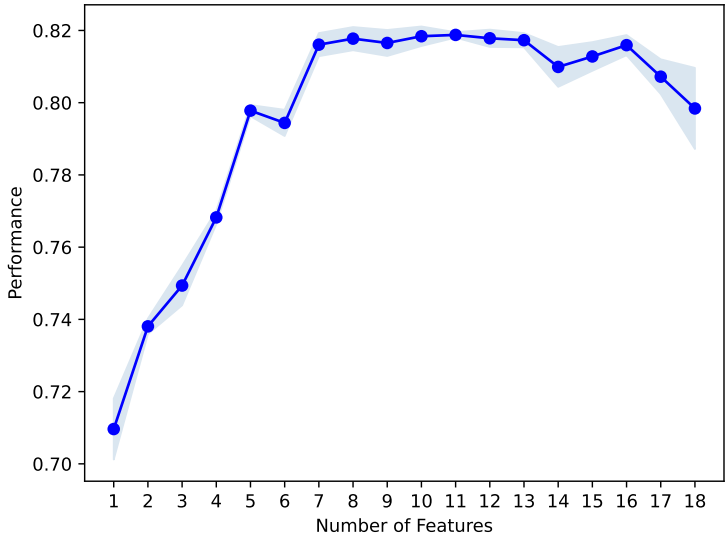
Additionally, we employ cross-validation during each evaluation to enhance the robustness of our findings, leading to a total of $171 \times 5 = 855$ model fits. To manage the computational demands, we selectively use smaller data subsets—10,000 and 30,000 samples—and execute SFS on these subsets. The implementation details can be found in Appendix B.

The outcomes, are summarized in Figure 2 and Table 1. This approach not only elucidates the efficacy of selected features but also helps in optimizing processing time. Table 1 outlines the features selected during each iteration and their corresponding average CV accuracy, illustrating how each feature contributes to the overall model performance. The results are directly related to plot (b) in Figure 2, providing a detailed view of the feature selection impact within this specific subset.

We have opted to advance with three distinct feature lists for our analysis. The first list is the initially suggested base list, the second is the best-performing list as determined by the Sequential Feature Selector (SFS), and the third is the smallest list that maintains at least 80% accuracy. The latter two lists are highlighted in green in Table 1. Utilizing the full dataset, we produced plots similar to those in Figure 1 using the base NN architecure provided, and our findings are showcased in Figure 3. The relevant code is documented in Appendix C.



(a) 10 000 samples



(b) 30 000 samples

Figure 2: Accuracy Performance of the Sequential Feature Selector on Two Data Subsets.

Average CV Score	Feature Names
70.96%	('bjet_1_pt',)
73.81%	('bjet_1_pt', 'n_jets')
74.94%	('bjet_1_pt', 'n_jets', 'n_bjets')
76.82%	('jet_1_twb', 'bjet_1_pt', 'n_jets', 'n_bjets')
79.78%	('jet_1_twb', 'jet_2_twb', 'bjet_1_pt', 'n_jets', 'n_bjets')
79.44%	('jet_3_pt', 'jet_1_twb', 'jet_2_twb', 'bjet_1_pt', 'n_jets', 'n_bjets')
81.61%	('jet_3_pt', 'jet_1_twb', 'jet_2_twb', 'jet_3_twb', 'bjet_1_pt', 'n_jets', 'n_bjets')
81.77%	('jet_3_pt', 'jet_3_eta', 'jet_1_twb', 'jet_2_twb', 'jet_3_twb', 'bjet_1_pt', 'n_jets', 'n_bjets')
81.65%	('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')
81.84%	('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')
81.88%	('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')
81.78%	('jet_1_pt', 'jet_3_pt', 'jet_1_eta', 'jet_3_eta', 'jet_1_twb', 'jet_2_twb', 'jet_3_twb', 'bjet_1_pt', 'lep_1_pt', 'n_jets', 'n_bjets', 'n_leptons')
81.73%	('jet_1_pt', 'jet_3_pt', 'jet_1_eta', 'jet_2_eta', 'jet_3_eta', 'jet_1_twb', 'jet_2_twb', 'jet_3_twb', 'bjet_1_pt', 'lep_1_pt', 'n_jets', 'n_bjets', 'n_leptons')
80.99%	('jet_1_pt', 'jet_3_pt', 'jet_1_eta', 'jet_2_eta', 'jet_3_eta', 'jet_1_twb', 'jet_2_twb', 'jet_3_twb', 'bjet_1_pt', 'lep_1_pt', 'n_jets', 'n_bjets', 'n_leptons', 'H_T')
81.28%	('jet_1_pt', 'jet_3_pt', 'jet_1_eta', 'jet_2_eta', 'jet_3_eta', 'jet_1_twb', 'jet_2_twb', 'jet_3_twb', 'bjet_1_pt', 'lep_1_pt', 'n_jets', 'n_bjets', 'n_leptons', 'met_met', 'H_T')
81.59%	('jet_1_pt', 'jet_3_pt', 'jet_1_eta', 'jet_2_eta', 'jet_3_eta', 'jet_1_twb', 'jet_2_twb', 'jet_3_twb', 'bjet_1_pt', 'lep_1_pt', 'lep_2_pt', 'n_jets', 'n_bjets', 'n_leptons', 'met_met', 'H_T')
80.72%	('jet_1_pt', 'jet_3_pt', 'jet_1_eta', 'jet_2_eta', 'jet_3_eta', 'jet_1_twb', 'jet_2_twb', 'jet_3_twb', 'bjet_1_pt', 'lep_1_pt', 'lep_2_pt', 'lep_3_pt', 'n_jets', 'n_bjets', 'n_leptons', 'met_met', 'H_T')
79.84%	('jet_1_pt', 'jet_2_pt', 'jet_3_pt', 'jet_1_eta', 'jet_2_eta', 'jet_3_eta', 'jet_1_twb', 'jet_2_twb', 'jet_3_twb', 'bjet_1_pt', 'lep_1_pt', 'lep_2_pt', 'lep_3_pt', 'n_jets', 'n_bjets', 'n_leptons', 'met_met', 'H_T')

Table 1: Iterative Feature Selection and Corresponding Average Cross-Validation Accuracy Scores by Sequential Feature Selector (SFS) for the 30,000 Sample Subset.

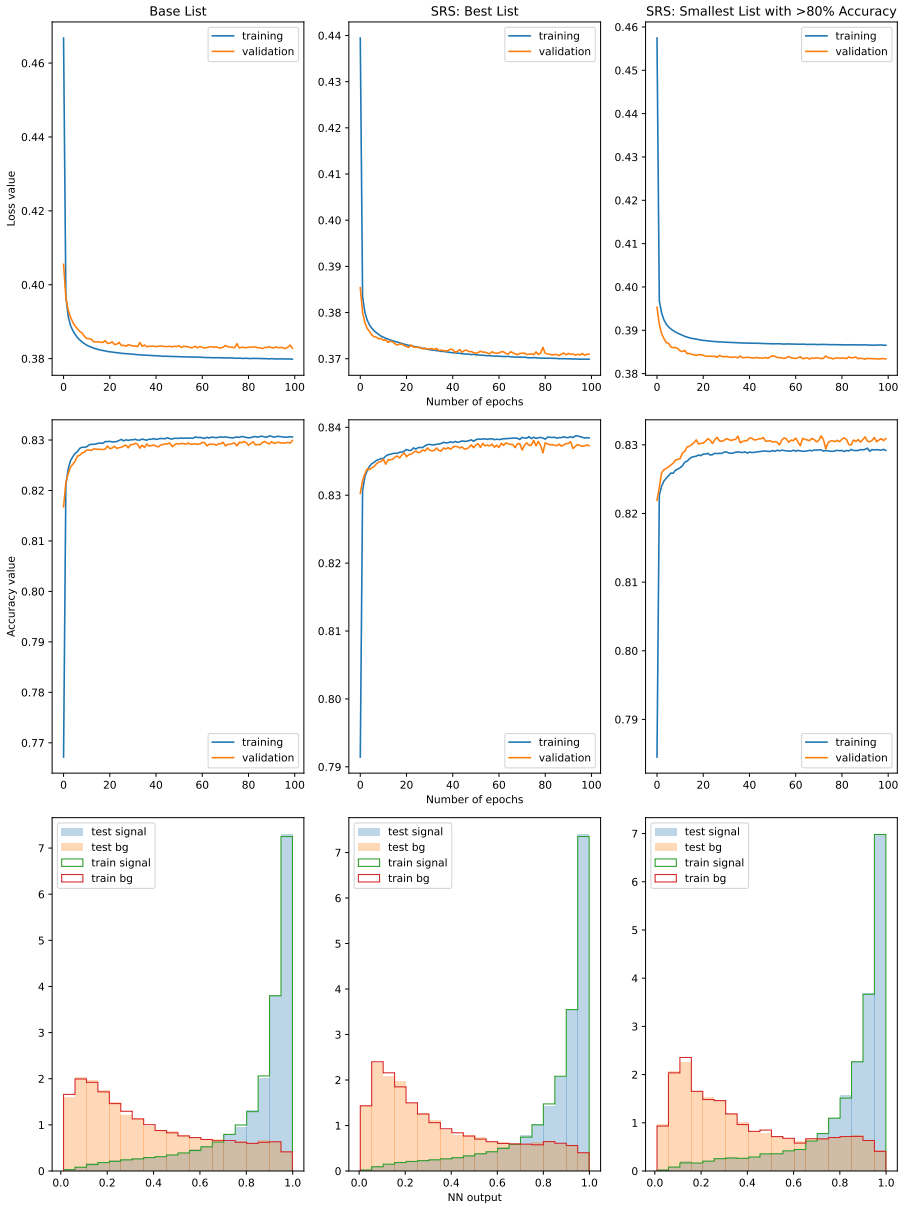


Figure 3: Performance Comparison of Neural Network Models Using Different Feature Lists. Top panels show loss curves for the base list, SFS best list, and smallest list with over 80% accuracy across training and validation phases over 100 epochs. Middle panels depict corresponding accuracy values. Bottom panels illustrate NN output distributions for training and testing datasets, highlighting model behavior for signal and background classes.

3.2 Parameter Optimization via GridSearchCV

In this subsection, we detail our findings from using GridSearchCV on three feature lists identified previously. The optimization adjusted parameters including the number of nodes in three neural network layers, learning rate, and batch size, with three possible values for each of these five parameters, as shown in Table 2. Notably, parameters in Set 2 are those used in the base model. This exhaustive search, involving $3^7 = 2187$ fits due to the parameters, threefold cross-validation, and three feature sets, demanded significant computational resources, leading us to use a smaller dataset of 30,000 samples.

The results are presented in Table 3, which displays the best parameter configurations and their corresponding performances. Each configuration was evaluated through 3-fold cross-validation, and the 'Best Score' column indicates the average accuracy across the folds, identifying the most effective settings. We then applied these optimal parameters to generate the visualizations in Figure 4, akin to those in Figure 3. Relevant code is available in Appendices D and E.

Parameter	Set 1	Set 2	Set 3
Nodes: Dense Layer 1	25	50	100
Nodes: Dense Layer 2	12	25	50
Nodes: Dense Layer 3	5	10	15
Batch Size	100	150	300
Adam Learning Rate	0.002	0.0002	0.00002

Table 2: Parameter configurations for GridSearchCV, listing sets of values for nodes in each dense layer, batch sizes, and learning rates. Configuration 2 corresponds to the parameters of the base model.

Parameters	Base List	SRS: Best List	SRS: Best Small List
Nodes: Dense Layer 1	50	25	50
Nodes: Dense Layer 2	12	25	50
Nodes: Dense Layer 3	10	15	5
Batch Size	300	100	150
Adam Learning Rate	0.002	0.002	0.0002
Best Score	82.96%	83.39%	82.7%

Table 3: Optimized Parameter Sets and Corresponding Performance Scores from GridSearchCV for our Three Feature Lists.

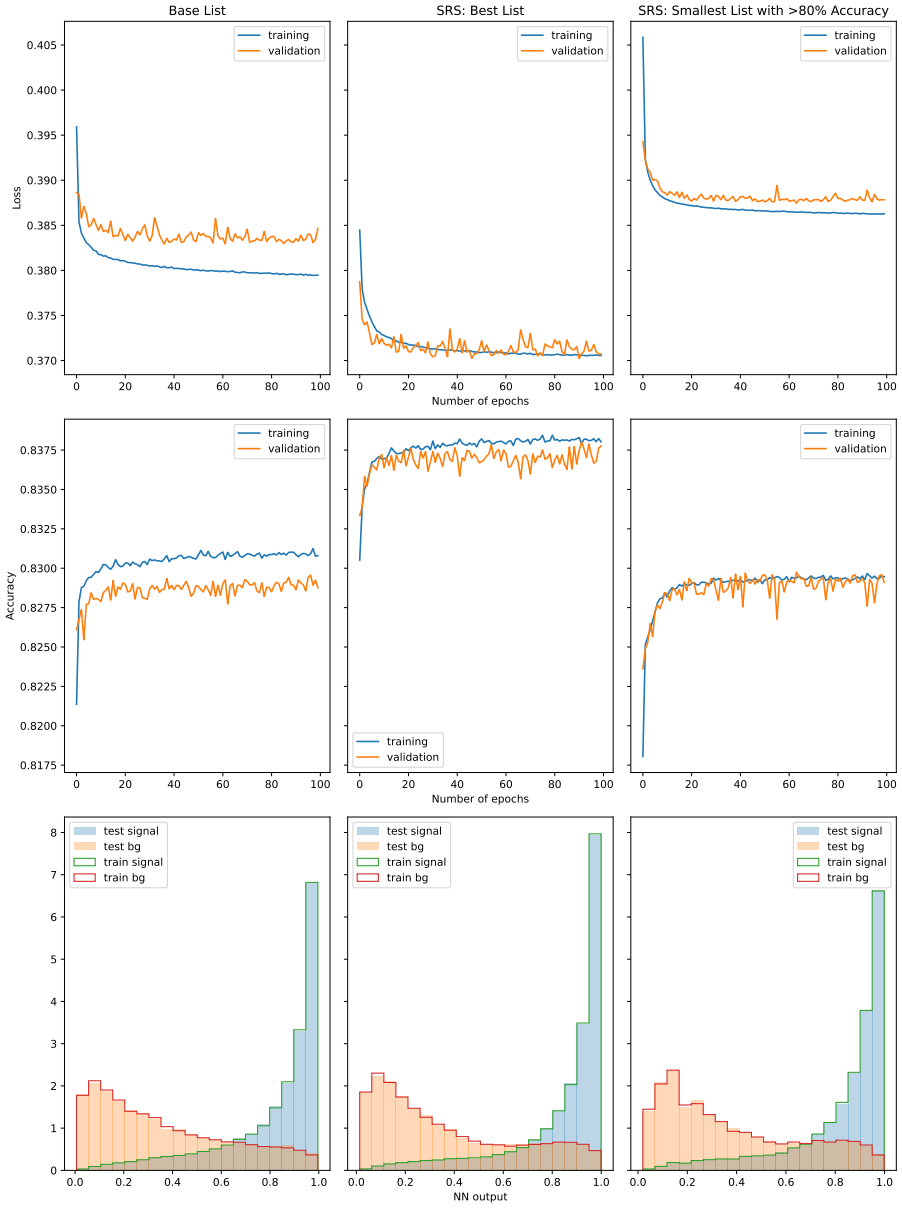


Figure 4: Performance Comparison of Optimized Neural Network Models Using Different Feature Lists. This figure displays the outcomes of models individually optimized using GridSearchCV for the base list, the SFS best list, and the smallest list with over 80% accuracy. The top panels illustrate loss curves for both training and validation phases over 100 epochs. The middle panels show the corresponding accuracy values, and the bottom panels provide NN output distributions for the training and testing datasets, emphasizing the model's differentiation between signal and background classes.

3.3 Feature Importance Analysis

In this subsection, we present our findings from conducting permutation feature importance analysis on the optimal models developed in the previous subsection. Permutation feature importance evaluates the impact of each feature on the model's performance by randomly shuffling individual feature values and observing the change in model accuracy. This process disrupts the relationship between the feature and the outcome, quantifying the importance of each feature based on the decrease in model performance. Our findings are summarized in Figure 5. This figure displays the relative importance of features across all three feature lists, highlighting how each feature contributes to model performance when utilizing the best parameters identified for each list. The relevant code is documented in Appendix E.

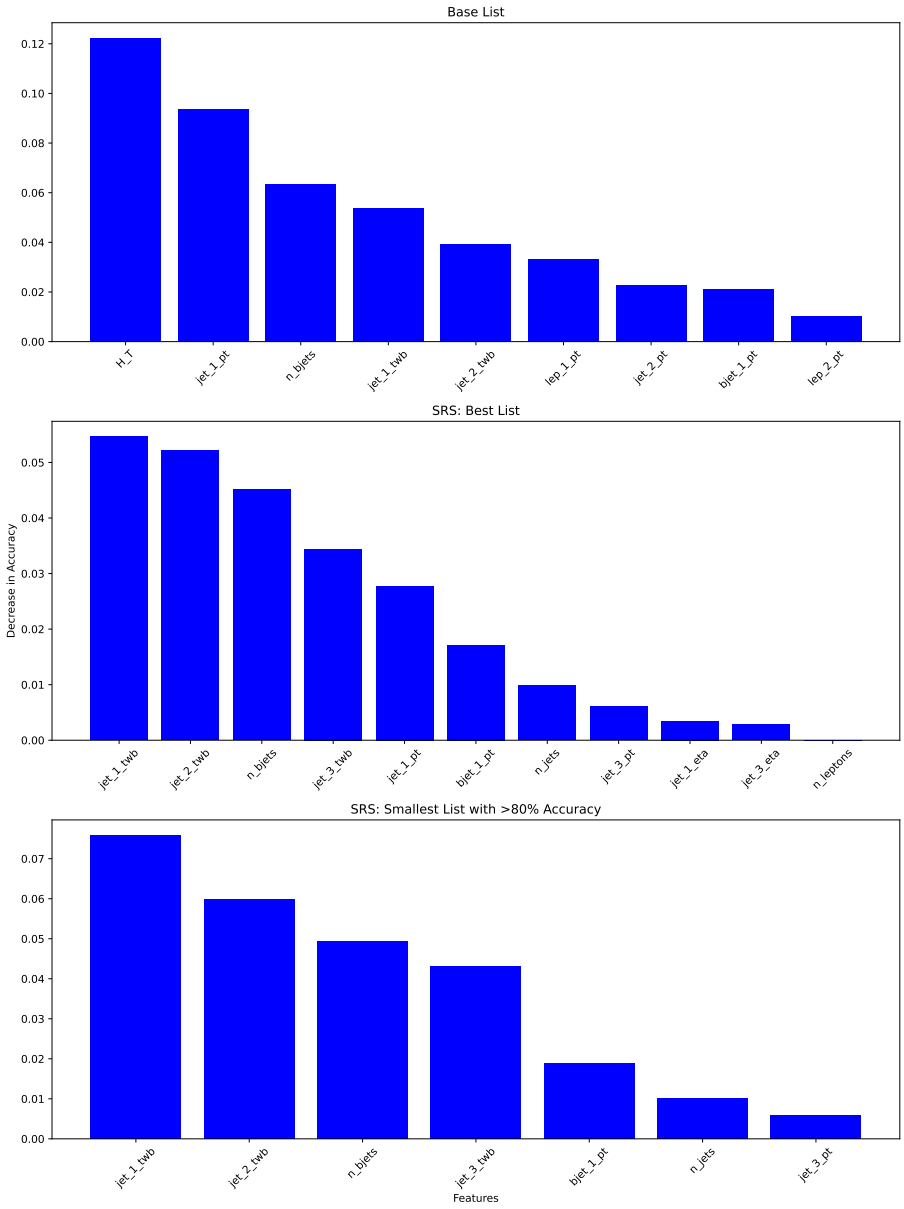


Figure 5: Feature Importances for Each Feature List Based on Optimized Model Parameters.

4 Discussion and Summary

In this study, while we did not achieve a significant improvement in accuracy, we successfully demonstrated the ability to reduce the dimensionality of our feature space without compromising accuracy. Notably, our smallest feature list comprising only 7 features performed comparably to the base list of 9 features and the optimal list of 11 features identified by Sequential Feature Selector (SFS).

An interesting observation emerged from the feature importance analysis. Contrary to expectations, the ranking of features in the smallest list did not align perfectly with those in the best list derived from SFS. For example, the feature jet_1_pt was ranked higher in the best list's feature importance despite not being in the top 7 of the SFS selections. Additionally, the base model identified H_T as the most crucial feature, a finding SFS did not recognize until the 14th iteration, as shown in Table 1. This discrepancy suggests that the forward iteration approach of SFS may not have been optimal, and exploring alternative methods such as backward elimination could potentially yield different insights into feature significance.

Although SFS did not fail (provided results comparable to the base model), it also did not furnish a significant enhancement in performance. The grid search over various parameters similarly did not yield a substantial increase in accuracy, with most results hovering around 80%.

This study underscores the complexity of feature selection and the potential limitations of certain algorithms in capturing the most predictive features. Future work could explore more dynamic feature selection methods or hybrid approaches that combine forward and backward elimination to potentially uncover more effective feature combinations. Additionally, experimenting with different model architectures or more advanced machine learning algorithms might also lead to improvements in performance. Overall, the findings suggest a continued exploration into more nuanced aspects of feature selection and model optimization to better understand and enhance predictive accuracies in complex datasets.

A Code: Base Model

In this section, we provide the code used to implement the base model introduced at the beginning of this project. This code was used in generating the visualizations displayed in Figure 1.

[1]:
```
import h5py
import matplotlib.pyplot as plt
import numpy as np
import tensorflow.keras as K
from numpy.lib.recfunctions import structured_to_unstructured
from sklearn.model_selection import train_test_split
import pandas as pd
```

```
2024-04-16 16:46:04.583009: 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"][:]
```

2 'Unstructuring' Data

[4]:
```
signal_data = structured_to_unstructured(signal_data)
bg_data = structured_to_unstructured(bg_data)
```

3 Train - Validation - Test Splits

[5]:
```
X = np.concatenate([signal_data, bg_data])
y = np.concatenate([np.ones(signal_data.shape[0], dtype=int), np.zeros(bg_data.
 ↪shape[0], dtype=int)])

x_train, x_rem, y_train, y_rem = train_test_split(X, y, train_size=0.8)
x_val, x_test, y_val, y_test = train_test_split(x_rem, y_rem, train_size=0.5)
```

4 Data Preprocessing

[6]:
```
preprocessing_layer = K.layers.Normalization()
preprocessing_layer.adapt(x_train)
```

5 Model training

[7]:
```
model = K.Sequential(
    [
        preprocessing_layer,
        K.layers.Dense(50, activation="relu", name="hidden1"),
        K.layers.Dense(25, activation="relu", name="hidden2"),
        K.layers.Dense(10, activation="relu", name="hidden3"),
        K.layers.Dense(1, activation="sigmoid", name="output"),
    ]
)

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 18)               37
 n)

 hidden1 (Dense)             (None, 50)                950

 hidden2 (Dense)             (None, 25)                1275

 hidden3 (Dense)             (None, 10)                260

 output (Dense)              (None, 1)                 11

=================================================================
Total params: 2,533
Trainable params: 2,496
Non-trainable params: 37
_________________________________________________________________
```

[8]:
```
model.compile(
    optimizer=K.optimizers.Adam(learning_rate=0.0002),
    loss=K.losses.BinaryCrossentropy(),
    metrics=[K.metrics.BinaryAccuracy()],
)
```

[9]:
```
early_stopping_callback = K.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=10,
    min_delta=0.002,
    restore_best_weights=True,
    verbose=1,)
```

[10]:
```
fit_history = model.fit(
        x_train,
        y_train,
        batch_size=100,
        epochs=100,
        validation_data=(x_val, y_val),
        callbacks=[early_stopping_callback],
    )

print("Printing summary of the trained model:")
print(model.summary())
```

```
Epoch 1/100
1159/1159 [==============================] - 2s 1ms/step - loss: 0.4256 -
binary_accuracy: 0.8084 - val_loss: 0.3800 - val_binary_accuracy: 0.8312
        ⋮
Epoch 29/100
1159/1159 [==============================] - 1s 1ms/step - loss: 0.3594 -
binary_accuracy: 0.8435 - val_loss: 0.3579 - val_binary_accuracy: 0.8433
Epoch 30/100
1127/1159 [============================>.] - ETA: 0s - loss: 0.3591 -
binary_accuracy: 0.8439Restoring model weights from the end of the best epoch:
20.
1159/1159 [==============================] - 2s 1ms/step - loss: 0.3592 -
binary_accuracy: 0.8438 - val_loss: 0.3577 - val_binary_accuracy: 0.8437
Epoch 30: early stopping
Printing summary of the trained model:
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 18)               37
 n)

 hidden1 (Dense)             (None, 50)                950

 hidden2 (Dense)             (None, 25)                1275

 hidden3 (Dense)             (None, 10)                260

 output (Dense)              (None, 1)                 11

=================================================================
Total params: 2,533
Trainable params: 2,496
Non-trainable params: 37
_________________________________________________________________
```

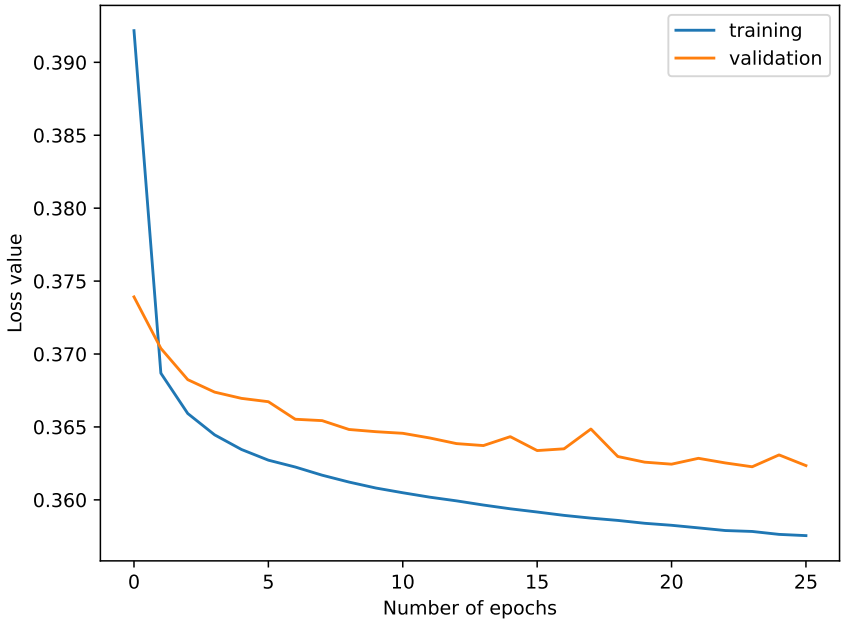
```
None
```

[11]:
```
save_name = "my_model"
print(f'Storing model with name "{save_name}" now. You can convert '
        'this to ONNX format with the "tf2onnx" command-line utility.')
model.save(save_name)
```

```
Storing model with name "my_model" now. You can convert this to ONNX format with
the "tf2onnx" command-line utility.

WARNING:absl:Found untraced functions such as _update_step_xla while saving
(showing 1 of 1). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: my_model/assets

INFO:tensorflow:Assets written to: my_model/assets
```

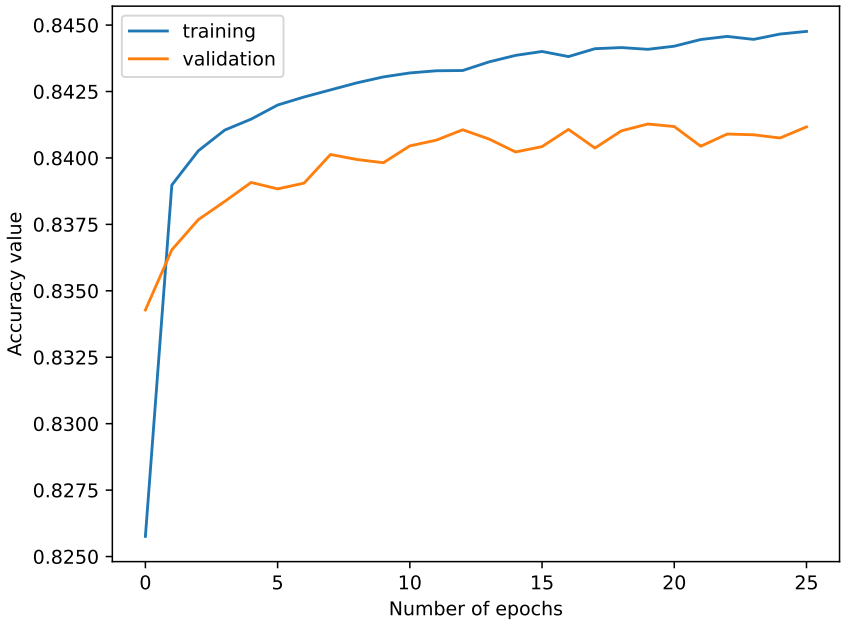
The following code produces a plot of the loss evolution for training and validation:

[12]:
```
plt.plot(fit_history.history["loss"], label="training")
plt.plot(fit_history.history["val_loss"], label="validation")
plt.xlabel("Number of epochs")
plt.ylabel("Loss value")
plt.legend()
plt.tight_layout()
plt.show()
```



The following code produces a plot of the accuracy evolution per epoch for training and validation:

[13]:
```
plt.figure()
plt.plot(fit_history.history["binary_accuracy"], label="training")
plt.plot(fit_history.history["val_binary_accuracy"], label="validation")
plt.xlabel("Number of epochs")
plt.ylabel("Accuracy value")
plt.legend()
plt.tight_layout()
#plt.savefig(output_file)
```

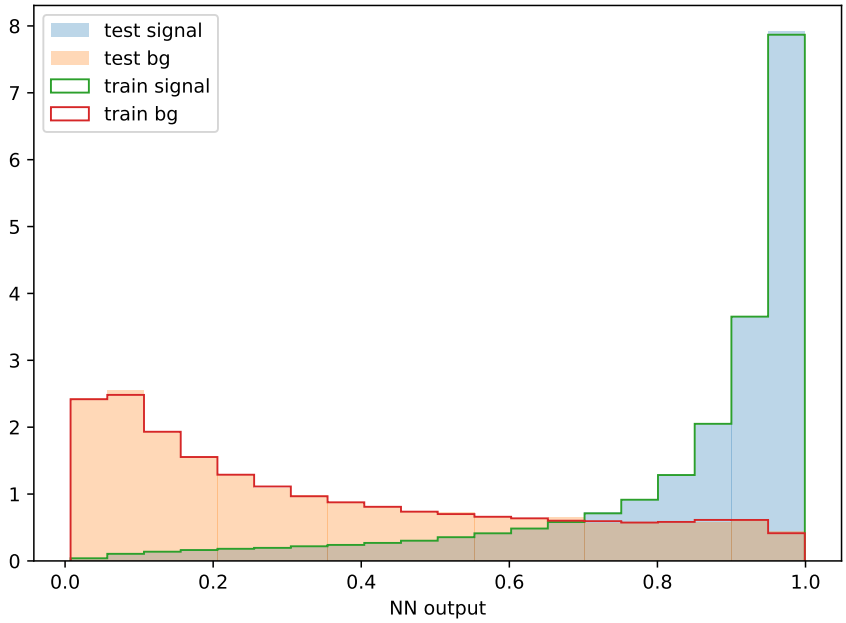


The following code plots the distribution of the neural-network output node for both training and test data to check for possible differences between the two. The datasets are sliced according to their truth labels (i.e. the two classes).

[14]:
```
"""Creates a plot of the NN output for training and test data.

    The function slices both training and test data according to the truth
    labels, so that the displayed spectra can be split into "background" and
    "signal" events (i.e. the two classes the model was trained on).

    """

_, bins, _ = plt.hist(model.predict(x_test[y_test.astype(bool)]), bins=20,␣
↪alpha=0.3, density=True, label="test signal")
plt.hist(model.predict(x_test[~y_test.astype(bool)]), bins=bins, alpha=0.3,␣
↪density=True, label="test bg")
```

```
plt.hist(model.predict(x_train[y_train.astype(bool)]), bins=bins, density=True,␣
 ↪histtype="step", label="train signal")
plt.hist(model.predict(x_train[~y_train.astype(bool)]), bins=bins, density=True,␣
 ↪histtype="step", label="train bg")
plt.xlabel("NN output")
plt.legend()
plt.tight_layout()
plt.show()
#plt.savefig(output_file)
print("Created plots of loss, accuracy, and NN output.")
```

```
1595/1595 [==============================] - 1s 422us/step
723/723 [==============================] - 0s 420us/step
12717/12717 [==============================] - 5s 429us/step
5823/5823 [==============================] - 3s 446us/step
```




```
Created plots of loss, accuracy, and NN output.
```

B Code: Feature Selection

In this section, we present the code utilized for the Sequential Feature Selector. This code facilitated the creation of the plots shown in Figure 2 and the data compiled in Table 1.

[1]:
```
import h5py
import matplotlib.pyplot as plt
import numpy as np
import tensorflow.keras as K
import pandas as pd
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
import pandas as pd
```

```
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.
```

```
2024-04-14 01:18:28.522266: 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"][:]
```

2 Subset of the Data

[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})

# 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()
```

[3]:

	index	jet_1_pt	jet_2_pt	jet_3_pt	jet_1_eta	jet_2_eta
469349	469349	110.017174	89.244843	42.692078	-1.638234	-1.114363
8248	8248	183.934067	125.170509	93.043922	0.185653	-1.264277
699594	190559	195.334396	155.822617	39.977455	-0.860027	1.231296
149760	149760	165.566879	136.009689	124.258598	0.903266	-0.343321
72006	72006	128.334076	54.510422	45.616039	-0.975434	-2.489865

	jet_3_eta	jet_1_twb	jet_2_twb	jet_3_twb	bjet_1_pt	lep_1_pt
469349	0.334825	3	1	1	110.017174	165.685593
8248	1.190622	1	4	1	125.170509	68.110207
699594	2.197089	1	1	1	35.366051	184.187210
149760	0.041760	5	5	1	165.566879	113.711166
72006	-1.147104	1	1	1	32.256153	109.551361

	lep_2_pt	lep_3_pt	n_jets	n_bjets	n_leptons	met_met
469349	81.492973	32.393501	3	1	3	64.430573
8248	35.741417	12.237628	4	1	3	104.903961
699594	82.709946	36.546856	5	1	3	48.988052
149760	112.550560	34.406342	6	2	3	172.116821
72006	100.125435	48.681091	6	2	3	29.693062

	H_T	ttZ
469349	585.956726	1.0
8248	654.210388	1.0
699594	807.437683	0.0
149760	1007.604309	1.0
72006	606.260437	1.0

[4]:
```
# 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
X_sub = subset_df[['jet_1_pt', 'jet_2_pt', 'jet_3_pt', 'jet_1_eta', 'jet_2_eta',
       'jet_3_eta', 'jet_1_twb', 'jet_2_twb', 'jet_3_twb', 'bjet_1_pt',
       'lep_1_pt', 'lep_2_pt', 'lep_3_pt', 'n_jets', 'n_bjets', 'n_leptons',
       'met_met', 'H_T']]
y_sub = subset_df['ttZ']

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

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

3 Sequential Feature Selector

https://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/#example-9-selecting-the-best-feature-combination-in-a-k-range

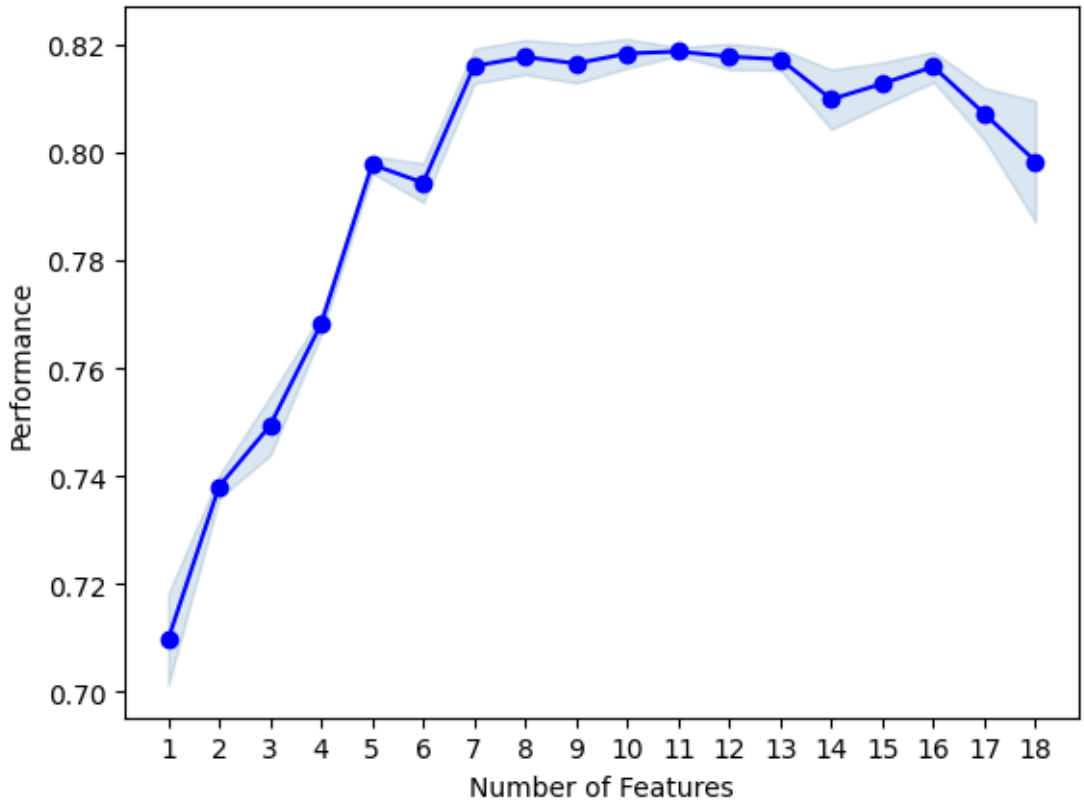
[5]:
```
# model for classification
def build_model(input_dim=None):
    model = K.Sequential([
        K.layers.Normalization(),
        K.layers.Dense(50, activation="relu", input_dim=input_dim),
        K.layers.Dense(25, activation="relu"),
        K.layers.Dense(10, activation="relu"),
        K.layers.Dense(1, activation="sigmoid")
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# wrapper function
model = KerasClassifier(build_fn=lambda: build_model(input_dim=X_sub.shape[1]),␣
 ↪epochs=10, batch_size=32, verbose=0)
```

```
/var/folders/3x/lv7sddxn2gg8mq0dwdcn1wg40000gn/T/ipykernel_29259/2265007354.py:1
3: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras
(https://github.com/adriangb/scikeras) instead. See
https://www.adriangb.com/scikeras/stable/migration.html for help migrating.
  model = KerasClassifier(build_fn=lambda:
build_model(input_dim=X_sub.shape[1]), epochs=10, batch_size=32, verbose=0)
```

[6]:
```
# SFS model, will check performance from 1 feature to all 18
sfs = SFS(model,
          k_features = (1,18),
          forward=True,
          floating=False,
          scoring='accuracy',
          cv=5,
          verbose=0)

sfs = sfs.fit(X_sub, y_sub)
```

[7]:
```
# plotting performance improvement with each additional feature

print('best combination (ACC: %.3f): %s\n' % (sfs.k_score_, sfs.k_feature_idx_))
plot_sfs(sfs.get_metric_dict(), kind='std_err');
plt.savefig('feature_num_performance_30k.pdf')
```

```
best combination (ACC: 0.819): (0, 2, 3, 5, 6, 7, 8, 9, 13, 14, 15)
```



[8]:
```
# saving performance of all feature sets

feature_performance_df = pd.DataFrame.from_dict(sfs.get_metric_dict()).T
feature_performance_df.to_csv('performance_30k.csv', index=False)
feature_performance_df
```

[8]:

	feature_idx
1	(9,)
2	(9, 13)
3	(9, 13, 14)
4	(6, 9, 13, 14)
5	(6, 7, 9, 13, 14)
6	(2, 6, 7, 9, 13, 14)
7	(2, 6, 7, 8, 9, 13, 14)
8	(2, 5, 6, 7, 8, 9, 13, 14)
9	(2, 3, 5, 6, 7, 8, 9, 13, 14)
10	(2, 3, 5, 6, 7, 8, 9, 13, 14, 15)
11	(0, 2, 3, 5, 6, 7, 8, 9, 13, 14, 15)
12	(0, 2, 3, 5, 6, 7, 8, 9, 10, 13, 14, 15)
13	(0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 14, 15)
14	(0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 14, 15, 17)
15	(0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 14, 15, 16...
16	(0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15...
17	(0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...

```
18  (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...

                                              cv_scores avg_score  \
1   [0.7248333333333333, 0.6886666666666666, 0.721...  0.709633
2   [0.7351666666666666, 0.7353333333333333, 0.735...  0.738067
3   [0.7548333333333334, 0.765, 0.7485, 0.747, 0.7...  0.749367
4   [0.765, 0.7721666666666667, 0.7733333333333333...  0.768233
5   [0.7918333333333333, 0.8003333333333333, 0.8, ...    0.7978
6   [0.7828333333333334, 0.8046666666666666, 0.794...    0.7944
7   [0.8116666666666666, 0.8283333333333334, 0.810...  0.816067
8   [0.8115, 0.8275, 0.8211666666666667, 0.81, 0.8...  0.817733
9   [0.815, 0.8263333333333334, 0.812, 0.8065, 0.8...  0.816533
10  [0.8146666666666667, 0.8246666666666667, 0.811...    0.8184
11  [0.8196666666666667, 0.8185, 0.817, 0.8175, 0...  0.818767
12  [0.8168333333333333, 0.8203333333333334, 0.816...  0.817833
13  [0.8201666666666667, 0.8226666666666667, 0.811...    0.8173
14  [0.8206666666666667, 0.8223333333333334, 0.810...    0.8099
15             [0.8215, 0.82, 0.8055, 0.8155, 0.8015]    0.8128
16  [0.817, 0.8243333333333334, 0.811, 0.808666666...  0.815933
17  [0.8131666666666667, 0.789, 0.806, 0.815333333...    0.8072
18  [0.7715, 0.8163333333333334, 0.8115, 0.8218333...    0.7984

                                        feature_names  ci_bound   std_dev  \
1                                         (bjet_1_pt,)  0.021713  0.016894
2                                (bjet_1_pt, n_jets)  0.005534  0.004306
3                        (bjet_1_pt, n_jets, n_bjets)  0.014078  0.010953
4              (jet_1_twb, bjet_1_pt, n_jets, n_bjets)  0.004797  0.003732
5   (jet_1_twb, jet_2_twb, bjet_1_pt, n_jets, n_bj...  0.003981  0.003097
6   (jet_3_pt, jet_1_twb, jet_2_twb, bjet_1_pt, n_...  0.009342  0.007268
7   (jet_3_pt, jet_1_twb, jet_2_twb, jet_3_twb, bj...  0.008231  0.006404
8   (jet_3_pt, jet_3_eta, jet_1_twb, jet_2_twb, je...  0.008258  0.006425
9   (jet_3_pt, jet_1_eta, jet_3_eta, jet_1_twb, je...  0.009256  0.007201
10  (jet_3_pt, jet_1_eta, jet_3_eta, jet_1_twb, je...  0.006957  0.005413
11  (jet_1_pt, jet_3_pt, jet_1_eta, jet_3_eta, jet...  0.001938  0.001508
12  (jet_1_pt, jet_3_pt, jet_1_eta, jet_3_eta, jet...  0.006169  0.004799
13  (jet_1_pt, jet_3_pt, jet_1_eta, jet_2_eta, jet...  0.005066  0.003942
14  (jet_1_pt, jet_3_pt, jet_1_eta, jet_2_eta, jet...  0.014196  0.011045
15  (jet_1_pt, jet_3_pt, jet_1_eta, jet_2_eta, jet...  0.010215  0.007947
16  (jet_1_pt, jet_3_pt, jet_1_eta, jet_2_eta, jet...  0.007186  0.005591
17  (jet_1_pt, jet_3_pt, jet_1_eta, jet_2_eta, jet...  0.012362  0.009618
18  (jet_1_pt, jet_2_pt, jet_3_pt, jet_1_eta, jet_...  0.028888  0.022476

     std_err
1   0.008447
2   0.002153
3   0.005477
4   0.001866
5   0.001549
6   0.003634
7   0.003202
8   0.003213
9   0.003601
10  0.002706
```

11	0.000754
12	0.0024
13	0.001971
14	0.005523
15	0.003974
16	0.002796
17	0.004809
18	0.011238

C Code: Feature Comparison

In this section, we present code was utilized to generate the plots in Figure 3, which compare the performance of the base neural network model across the three feature lists.

[1]:
```
import h5py
import matplotlib.pyplot as plt
import numpy as np
import tensorflow.keras as K
from sklearn.model_selection import train_test_split
import pandas as pd
```

```
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.
```

```
2024-04-17 15:29:47.130479: 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.
```

[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"][:]
```

1 Loading the Data

[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})

# 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()

# splitting the labels from the rest of the dataset
y = df_shuffled['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: 68.65%
WZ events: 31.35%
```

2 Various Feature Lists

[4]:
```
# 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 Running Base Model for all Feature Lists

[5]:
```
fig, axs = plt.subplots(3, 3, figsize=(12, 16))

for i, feature_list in enumerate(input_lists):
    names = ['base list', 'SRS: best list', 'SRS: best smallest list']
    X = df_shuffled[feature_list]
    x_train, x_rem, y_train, y_rem = train_test_split(X, y, train_size=0.8)
    x_val, x_test, y_val, y_test = train_test_split(x_rem, y_rem, train_size=0.5)

    preprocessing_layer = K.layers.Normalization()
    preprocessing_layer.adapt(x_train)

    model = K.Sequential(
    [
        preprocessing_layer,
        K.layers.Dense(50, activation="relu", name="hidden1"),
        K.layers.Dense(25, activation="relu", name="hidden2"),
        K.layers.Dense(10, activation="relu", name="hidden3"),
        K.layers.Dense(1, activation="sigmoid", name="output"),
    ]
    )

    model.summary()

    model.compile(optimizer=K.optimizers.Adam(learning_rate=0.0002),
                  loss=K.losses.BinaryCrossentropy(),
                  metrics=[K.metrics.BinaryAccuracy()])

    fit_history = model.fit(
        x_train,
        y_train,
        batch_size=512,
        epochs=100,
        validation_data=(x_val, y_val),
        verbose = 0)


    print("Printing summary of the trained model:")
```

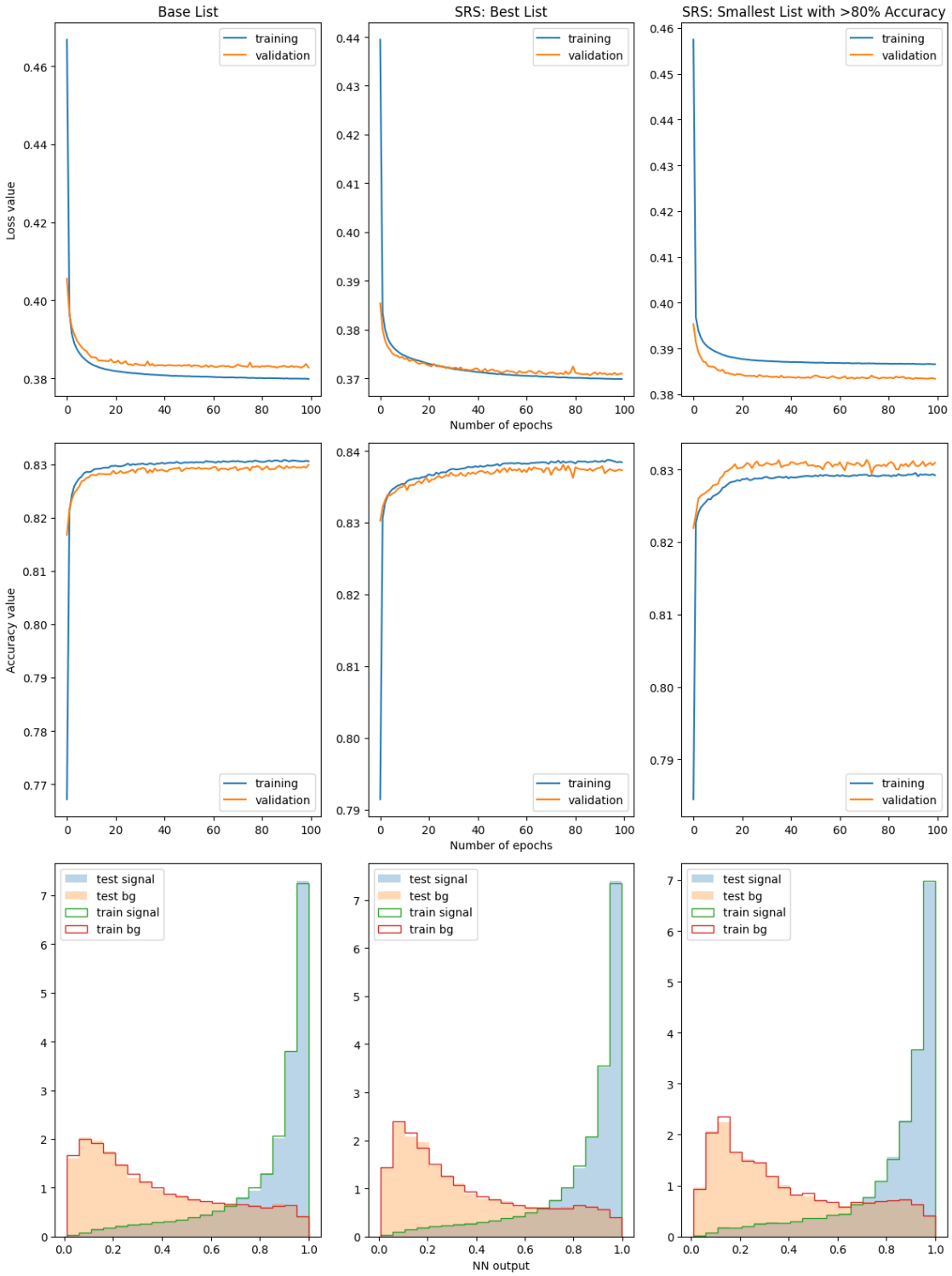
```
    print(model.summary())

    titles = ['Base List', 'SRS: Best List', 'SRS: Smallest List with >80%␣
↪Accuracy']

    axs[0, i].plot(fit_history.history["loss"], label="training")
    axs[0, i].plot(fit_history.history["val_loss"], label="validation")
    axs[0, i].legend()
    axs[0, i].set_title(titles[i])

    axs[1, i].plot(fit_history.history["binary_accuracy"], label="training")
    axs[1, i].plot(fit_history.history["val_binary_accuracy"],␣
↪label="validation")
    axs[1, i].legend()

    _, bins, _ = axs[2, i].hist(model.predict(x_test[y_test.astype(bool)]),␣
↪bins=20, alpha=0.3, density=True, label="test signal")
    axs[2, i].hist(model.predict(x_test[~y_test.astype(bool)]), bins=bins,␣
↪alpha=0.3, density=True, label="test bg")
    axs[2, i].hist(model.predict(x_train[y_train.astype(bool)]), bins=bins,␣
↪density=True, histtype="step", label="train signal")
    axs[2, i].hist(model.predict(x_train[~y_train.astype(bool)]), bins=bins,␣
↪density=True, histtype="step", label="train bg")
    axs[2, i].legend()

    if i == 0:
            axs[0, i].set_ylabel("Loss value")
            axs[1, i].set_ylabel("Accuracy value")

    elif i == 1:
        axs[0, i].set_xlabel("Number of epochs")
        axs[1, i].set_xlabel("Number of epochs")
        axs[2, i].set_xlabel("NN output")

fig.tight_layout()
plt.savefig("feat-comparison.pdf")
plt.show()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 9)                19
 n)

 hidden1 (Dense)             (None, 50)                500
```

```
 hidden2 (Dense)             (None, 25)                1275

 hidden3 (Dense)             (None, 10)                260

 output (Dense)              (None, 1)                 11

=================================================================
Total params: 2,065
Trainable params: 2,046
Non-trainable params: 19
_________________________________________________________________
Printing summary of the trained model:
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 9)                19
 n)

 hidden1 (Dense)             (None, 50)                500

 hidden2 (Dense)             (None, 25)                1275

 hidden3 (Dense)             (None, 10)                260

 output (Dense)              (None, 1)                 11

=================================================================
Total params: 2,065
Trainable params: 2,046
Non-trainable params: 19
_________________________________________________________________
None
1595/1595 [==============================] - 1s 375us/step
723/723 [==============================] - 0s 375us/step
12723/12723 [==============================] - 5s 388us/step
5817/5817 [==============================] - 2s 380us/step
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization_1 (Normalizat  (None, 11)               23
 ion)

 hidden1 (Dense)             (None, 50)                600

 hidden2 (Dense)             (None, 25)                1275
```

```
 hidden3 (Dense)             (None, 10)                260

 output (Dense)              (None, 1)                 11

=================================================================
Total params: 2,169
Trainable params: 2,146
Non-trainable params: 23
_________________________________________________________________
Printing summary of the trained model:
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization_1 (Normalizat  (None, 11)               23
 ion)

 hidden1 (Dense)             (None, 50)                600

 hidden2 (Dense)             (None, 25)                1275

 hidden3 (Dense)             (None, 10)                260

 output (Dense)              (None, 1)                 11

=================================================================
Total params: 2,169
Trainable params: 2,146
Non-trainable params: 23
_________________________________________________________________
None
1587/1587 [==============================] - 1s 440us/step
731/731 [==============================] - 0s 397us/step
12725/12725 [==============================] - 5s 398us/step
5815/5815 [==============================] - 2s 383us/step
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization_2 (Normalizat  (None, 7)                15
 ion)

 hidden1 (Dense)             (None, 50)                400

 hidden2 (Dense)             (None, 25)                1275

 hidden3 (Dense)             (None, 10)                260
```

```
 output (Dense)              (None, 1)                 11

=================================================================
Total params: 1,961
Trainable params: 1,946
Non-trainable params: 15
_________________________________________________________________
Printing summary of the trained model:
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization_2 (Normalizat  (None, 7)                15
 ion)

 hidden1 (Dense)             (None, 50)                400

 hidden2 (Dense)             (None, 25)                1275

 hidden3 (Dense)             (None, 10)                260

 output (Dense)              (None, 1)                 11

=================================================================
Total params: 1,961
Trainable params: 1,946
Non-trainable params: 15
_________________________________________________________________
None
1589/1589 [==============================] - 1s 385us/step
729/729 [==============================] - 0s 382us/step
12724/12724 [==============================] - 6s 469us/step
5815/5815 [==============================] - 2s 384us/step
```


Base List
SRS: Best List
SRS: Smallest List with >80% Accuracy
training
validation
Loss value
Number of epochs
Accuracy value
test signal
test bg
train signal
train bg
NN output

D Code: Parameter Optimization via GridSearchCV

In this section, you will find the code used for applying GridSearchCV to optimize model parameters. This code was instrumental in producing the data shown in Tables 2 and 3.

[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})
```

```
# 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]:
```
# 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

E Code: Best Models Performance and Feature Importance

In this section, the provided code was used to generate the plots in Figure 4, which compare the performance of individually optimized neural network models across the three feature lists. Additionally, this code facilitated the analysis of permutation feature importance, as displayed in Figure 5.

[1]:
```
import h5py
import matplotlib.pyplot as plt
import numpy as np
import tensorflow.keras as K
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.utils import shuffle
```

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})

# 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()

# splitting the labels from the rest of the dataset
y = df_shuffled['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: 68.65%
WZ events: 31.35%
```

2 Various Feature Lists

[4]:
```
# 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 Defining Model Function

[5]:
```
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
```

4 Importing Best Results

[6]:
```
# from model_selection.ipynb
results = pd.read_csv('model_selection.csv')
results
```

[6]:

	Unnamed: 0	base list	SRS: best list	SRS: best smallest list
0	batch_size	300.0000	100.0000	150.0000
1	model__layer1	50.0000	25.0000	50.0000
2	model__layer2	12.0000	25.0000	50.0000
3	model__layer3	10.0000	15.0000	5.0000
4	model__optimizer__lr	0.0020	0.0020	0.0002
5	best score	0.8296	0.8339	0.8270

5 Feature Importance through Permutations

[7]:
```
# Function to calculate permutation feature importance
def permutation_importance(model, X_valid, y_valid, metric=accuracy_score):

    # Store the baseline accuracy of the model on original data
    baseline_accuracy = metric(y_valid, model.predict(X_valid).round())

    # accuracy decreases will be stored here
    importances = []

    # iterations over each feature
    for i in range(X_valid.shape[1]):
        save = X_valid[:, i].copy()

        # shuffle individual feature
        X_valid[:, i] = shuffle(X_valid[:, i])
        m_accuracy = metric(y_valid, model.predict(X_valid).round())
```

```
        # restore original data
        X_valid[:, i] = save

        # store decrease in accuracy for feature i
        importances.append(baseline_accuracy - m_accuracy)

    return np.array(importances)
```

6 Running Best Models For Each Feature List And Ranking their Feature Importances

[8]:
```
# fig for all best model performances from each feature list
fig, axs = plt.subplots(3, 3, figsize=(12, 16), sharey='row')

#
fig2, axs2 = plt.subplots(3, 1, figsize=(12, 16))

for i, feature_list in enumerate(input_lists):
    names = ['base list', 'SRS: best list', 'SRS: best smallest list']

    ## train, test, val splits
    X = df_shuffled[feature_list]
    x_train, x_rem, y_train, y_rem = train_test_split(X, y, train_size=0.8)
    x_val, x_test, y_val, y_test = train_test_split(x_rem, y_rem, train_size=0.5)

    ## building model
    preprocessing_layer = K.layers.Normalization()
    preprocessing_layer.adapt(x_train)

    model = build_model(lr=results[names[i]][4],
                        layer1=int(results[names[i]][1]),
                        layer2=int(results[names[i]][2]),
                        layer3=int(results[names[i]][3]))

    # fitting model
    fit_history = model.fit(
        x_train,
        y_train,
        batch_size=int(results[names[i]][0]),
        epochs=100,
        validation_data=(x_val, y_val),
        verbose = 0)


    print("Printing summary of the trained model:")
```

```
   print(model.summary())

   titles = ['Base List', 'SRS: Best List', 'SRS: Smallest List with >80%␣
↪Accuracy']

   # plotting performance results
   axs[0, i].plot(fit_history.history["loss"], label="training")
   axs[0, i].plot(fit_history.history["val_loss"], label="validation")
   axs[0, i].legend()
   axs[0, i].set_title(titles[i])

   axs[1, i].plot(fit_history.history["accuracy"], label="training")
   axs[1, i].plot(fit_history.history["val_accuracy"], label="validation")
   axs[1, i].legend()

   _, bins, _ = axs[2, i].hist(model.predict(x_test[y_test.astype(bool)]),␣
↪bins=20, alpha=0.3, density=True, label="test signal")
   axs[2, i].hist(model.predict(x_test[~y_test.astype(bool)]), bins=bins,␣
↪alpha=0.3, density=True, label="test bg")
   axs[2, i].hist(model.predict(x_train[y_train.astype(bool)]), bins=bins,␣
↪density=True, histtype="step", label="train signal")
   axs[2, i].hist(model.predict(x_train[~y_train.astype(bool)]), bins=bins,␣
↪density=True, histtype="step", label="train bg")
   axs[2, i].legend()

   if i == 0:
           axs[0, i].set_ylabel("Loss")
           axs[1, i].set_ylabel("Accuracy")

   elif i == 1:
       axs[0, i].set_xlabel("Number of epochs")
       axs[1, i].set_xlabel("Number of epochs")
       axs[2, i].set_xlabel("NN output")


   # plotting feature rankings through the permutation method
   feature_importances = permutation_importance(model, x_test.to_numpy(),␣
↪y_test.to_numpy())

   # Sorting the features by importance
   df = pd.DataFrame({'Label': x_test.columns, 'Value': feature_importances})
   df = df.sort_values('Value', ascending=False)

   # Plotting the sorted data
   axs2[i].bar(df['Label'], df['Value'], color='blue')
   axs2[i].set_xticklabels(df['Label'], rotation=45)
```

```
    axs2[i].set_title(titles[i])

    if i == 1:
        axs2[i].set_ylabel('Decrease in Accuracy')
    elif i == 2:
        axs2[i].set_xlabel('Features')

    # evaluating test sets
    probs = model.predict(x_test)
    y_pred = (probs > 0.5).astype(int)
    print(names[i])
    print(accuracy_score(y_test, y_pred))

fig.tight_layout()
fig.savefig("best_models.pdf")

fig2.tight_layout()
fig2.savefig("feature_importances.pdf")

plt.show()
```

```
Printing summary of the trained model:
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 9)                19
 n)

 dense (Dense)               (None, 50)                500

 dense_1 (Dense)             (None, 12)                612

 dense_2 (Dense)             (None, 10)                130

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 1,272
Trainable params: 1,253
Non-trainable params: 19
_________________________________________________________________
None
1590/1590 [==============================] - 1s 371us/step
728/728 [==============================] - 0s 374us/step
12727/12727 [==============================] - 5s 409us/step
5812/5812 [==============================] - 2s 416us/step
2318/2318 [==============================] - 1s 447us/step
```

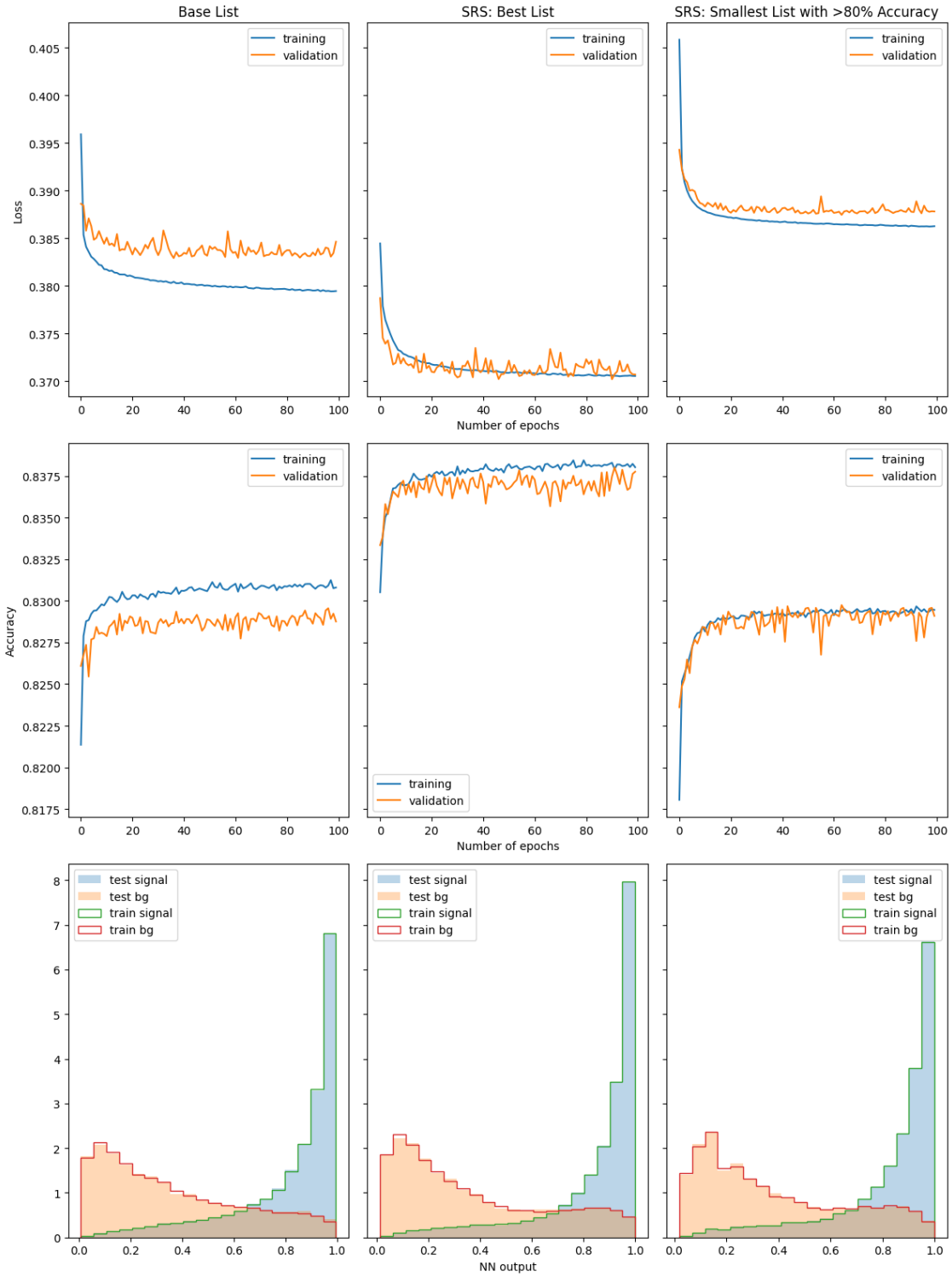
```
2318/2318 [==============================] - 1s 479us/step
2318/2318 [==============================] - 1s 453us/step
2318/2318 [==============================] - 1s 461us/step
2318/2318 [==============================] - 1s 441us/step
2318/2318 [==============================] - 1s 445us/step
2318/2318 [==============================] - 1s 527us/step
2318/2318 [==============================] - 1s 453us/step
2318/2318 [==============================] - 1s 491us/step
2318/2318 [==============================] - 1s 445us/step
 387/2318 [====>...] - ETA: 0s
```

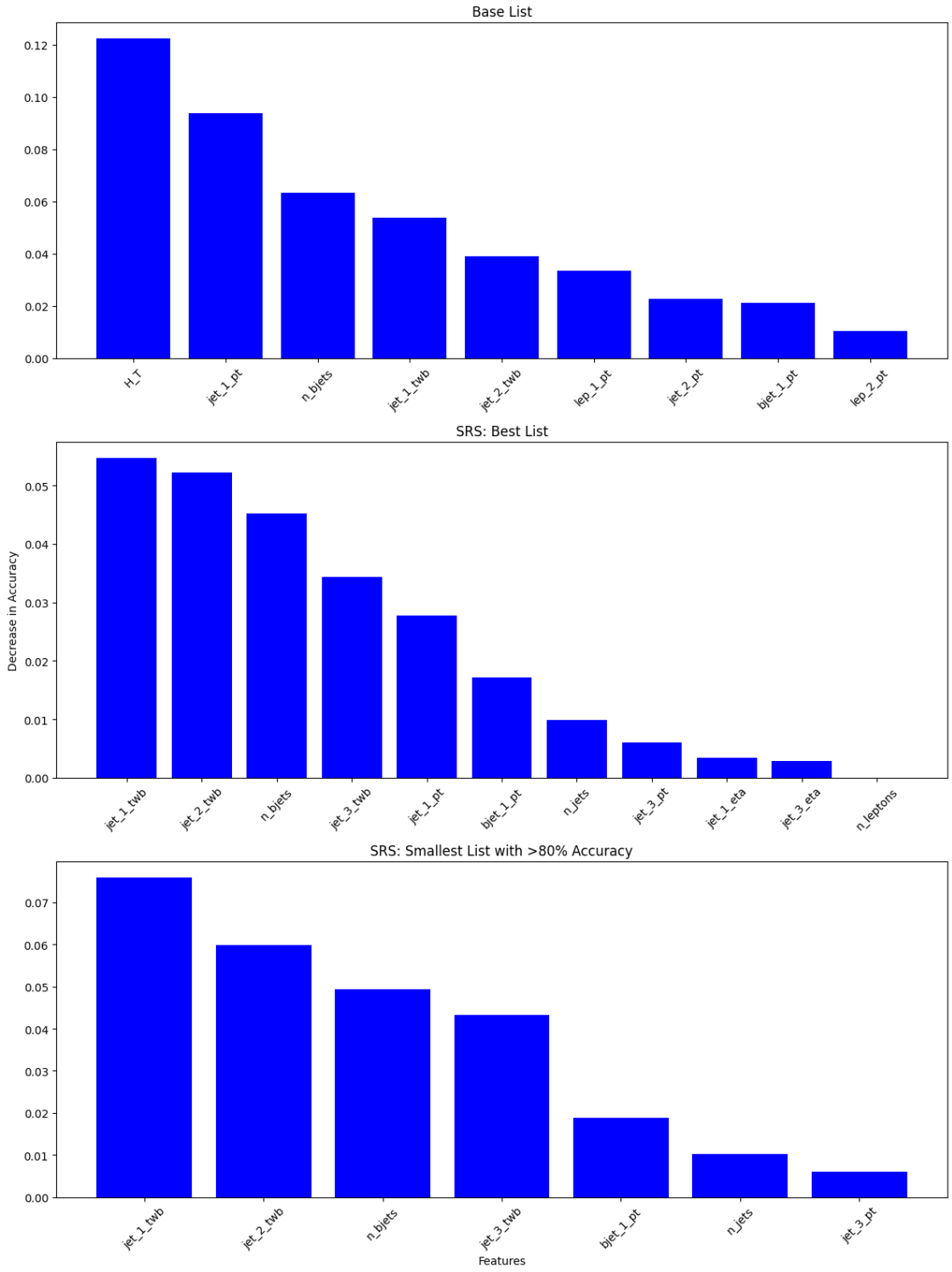
```
/var/folders/3x/lv7sddxn2gg8mq0dwdcn1wg40000gn/T/ipykernel_79200/1070017733.py:7
5: UserWarning: set_ticklabels() should only be used with a fixed number of
ticks, i.e. after set_ticks() or using a FixedLocator.
  axs2[i].set_xticklabels(df['Label'], rotation=45)
```

```
2318/2318 [==============================] - 1s 402us/step
base list
0.8290854168352347
Printing summary of the trained model:
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization_1 (Normalizat  (None, 11)               23
 ion)

 dense_4 (Dense)             (None, 25)                300

 dense_5 (Dense)             (None, 25)                650

 dense_6 (Dense)             (None, 15)                390

 dense_7 (Dense)             (None, 1)                 16

=================================================================
Total params: 1,379
Trainable params: 1,356
Non-trainable params: 23
_________________________________________________________________
None
1589/1589 [==============================] - 1s 371us/step
729/729 [==============================] - 0s 370us/step
12725/12725 [==============================] - 5s 367us/step
5814/5814 [==============================] - 2s 386us/step
2318/2318 [==============================] - 1s 372us/step
2318/2318 [==============================] - 1s 377us/step
2318/2318 [==============================] - 1s 369us/step
2318/2318 [==============================] - 1s 367us/step
```

```
2318/2318 [==============================] - 1s 610us/step
2318/2318 [==============================] - 1s 376us/step
2318/2318 [==============================] - 1s 370us/step
2318/2318 [==============================] - 1s 369us/step
2318/2318 [==============================] - 1s 367us/step
2318/2318 [==============================] - 1s 367us/step
2318/2318 [==============================] - 1s 417us/step
2318/2318 [==============================] - 1s 409us/step
 406/2318 [====>...] - ETA: 0s
/var/folders/3x/lv7sddxn2gg8mq0dwdcn1wg40000gn/T/ipykernel_79200/1070017733.py:7
5: UserWarning: set_ticklabels() should only be used with a fixed number of
ticks, i.e. after set_ticks() or using a FixedLocator.
  axs2[i].set_xticklabels(df['Label'], rotation=45)
2318/2318 [==============================] - 1s 368us/step
SRS: best list
0.839482698168676
Printing summary of the trained model:
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization_2 (Normalizat  (None, 7)                15
 ion)

 dense_8 (Dense)             (None, 50)                400

 dense_9 (Dense)             (None, 50)                2550

 dense_10 (Dense)            (None, 5)                 255

 dense_11 (Dense)            (None, 1)                 6

=================================================================
Total params: 3,226
Trainable params: 3,211
Non-trainable params: 15
_________________________________________________________________
None
1595/1595 [==============================] - 1s 440us/step
723/723 [==============================] - 0s 472us/step
12721/12721 [==============================] - 5s 385us/step
5818/5818 [==============================] - 2s 387us/step
2318/2318 [==============================] - 1s 417us/step
2318/2318 [==============================] - 1s 374us/step
2318/2318 [==============================] - 1s 377us/step
2318/2318 [==============================] - 1s 375us/step
2318/2318 [==============================] - 1s 376us/step
```

```
2318/2318 [==============================] - 1s 375us/step
2318/2318 [==============================] - 1s 375us/step
2318/2318 [==============================] - 1s 374us/step
 400/2318 [====>...] - ETA: 0s

/var/folders/3x/lv7sddxn2gg8mq0dwdcn1wg40000gn/T/ipykernel_79200/1070017733.py:7
5: UserWarning: set_ticklabels() should only be used with a fixed number of
ticks, i.e. after set_ticks() or using a FixedLocator.
  axs2[i].set_xticklabels(df['Label'], rotation=45)

2318/2318 [==============================] - 1s 378us/step
SRS: best smallest list
0.8313240014024867
```


Base List
SRS: Best List
SRS: Smallest List with >80% Accuracy
training
validation
Loss
Accuracy
Number of epochs
test signal
test bg
train signal
train bg
NN output


Base List
0.12
0.10
0.08
0.06
0.04
0.02
0.00
H_T
jet_1_pt
n_bjets
jet_1_twb
jet_2_twb
lep_1_pt
jet_2_pt
bjet_1_pt
lep_2_pt
SRS: Best List
Decrease in Accuracy
0.05
0.04
0.03
0.02
0.01
0.00
jet_1_twb
jet_2_twb
n_bjets
jet_3_twb
jet_1_pt
bjet_1_pt
n_jets
jet_3_pt
jet_1_eta
jet_3_eta
n_leptons
SRS: Smallest List with >80% Accuracy
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
jet_1_twb
jet_2_twb
n_bjets
jet_3_twb
bjet_1_pt
n_jets
jet_3_pt
Features