Higgs Boson Using Deep Learning

Laura Lazarescou, John Rodgers, Maysam Mansor and Mel Schwan

March 29, 2021

Introduction

In this case study our team designed a Neural Network to replicate the “Searching for Exotic Particles in High-energy Physics with Deep Learning” paper architecture using Tensorflow. We trained our model using the dataset located at the https://archive.ics.uci.edu/ml/datasets/HIGGS (https://archive.ics.uci.edu/ml/datasets/HIGGS) in the UCI Machine Learning Respository. Our objective is to tune our model to obtain AUC results as close to the result in Table 1.

Technique	Low-level	High-level	Complete
BDT	0.73 (0.01)	0.78 (0.01)	0.81 (0.01)
NN	0.733 (0.007)	0.777 (0.001)	0.816 (0.004)
DN	0.880 (0.001)	0.800 (<0.001)	0.885 (0.002)

Performance of Higgs Benchmark (Table 1)

The Dataset

Data Set Information:

The data has been produced using Monte Carlo simulations. The first 21 features (columns 2-22) are kinematic properties measured by the particle detectors in the accelerator. The last seven features are functions of the first 21 features; these are high-level features derived by physicists to help discriminate between the two classes. There is an interest in using deep learning methods to obviate the need for physicists to manually develop such features. Benchmark results using Bayesian Decision Trees from a standard physics package and 5-layer neural networks are presented in the original paper. We descreased the size of the data set to 1,000,000 records. This subset of the data was split 80/20 for our train and test sets.

Attribute Information:

The first column is the class label (1 for signal, 0 for background), followed by the 28 features (21 low-level features then 7 high-level features): lepton pT, lepton eta, lepton phi, missing energy magnitude, missing energy phi, jet 1 pt, jet 1 eta, jet 1 phi, jet 1 b-tag, jet 2 pt, jet 2 eta, jet 2 phi, jet 2 b-tag, jet 3 pt, jet 3 eta, jet 3 phi, jet 3 b-tag, jet 4 pt, jet 4 eta, jet 4 phi, jet 4 b-tag, m_jj, m_jjj, m_lv, m_jlv, m_bb, m_wbb, m_wwbb. For more detailed information about each feature see the original paper.

Methods

Replication of Model

Our modeling process began by updating the methods that were used by Baldi, Sadowski, and Whiteson. These models were created using Tensorflow based on the original model developed in PyLearn2. The model utilizes SGD with a learning rate of .05 for optimization of the model. This model contains 5 dense layers with 300 neurons each and a TANH activation. The output layer contains a single neuron and utilizes a Sigmoid activation to evaluate the classification values of 0 for background and 1 for signal. Each layer includes L2 regularization with a weight decay of 0.00001. A total of 10 epochs were evaluated with a batch size of 100, with early stopping enabled based on the accuracy score if 2 subsequent epochs did not improve the model. This model resulted in an accuracy score of 0.7468 and an AUC score of 0.8287. The model improved over all 10 epochs, meaning the early stopping did not occur. Because early stopping did not occur, the recreation of the original model was modified to increase the number of epochs from 10 to 20 and to decrease the batch size from 100 to 50. This model processed for 15 epochs before early stopping occurred with the 13th epoch performing with the best accuracy. The best accuracy for this model was 0.7508 and an AUC score of 0.8326.

Optimization of Tensor Flow Neural Networks

Expanding on the models of Baldi, et al, we manipulated a number of modeling parameters to see if we could improve Accuracy or increase Area Under the Curve or AUC. The ideal values of Accuracy and AUC are 1, and our first models performed well, but we hoped to increase the performance. We also wanted to see how a change in different aspects of the model would change our results. Deep neural networks are very effective at imitating the data they are given. This means they are capable of building more accurate models with a risk of overfitting so these models all applied a 70/30 train/test split within the model fit process.

In order to speed up processing, the majority of our subsequent models 2-14 were run on a subset of the total dataset. The number of records did have an effect on model performance. As we look at **Methods Table 2** and **Results Table 3** we see that models using one million records had consistently lower accuracy and AUC than models based on eleven million records. This is a strength of neural networks – the more data you add to the model, the more accurate the model will become. In these studies, we did not vary our batch size. Given the volume of data, we chose to use a batch size of 50 across all models. This allowed us to see the influence of other changes such as loss function, optimizer and activation function. We also varied the number of layers and the number of nodes in each layer. In Models 3 and 4 we chose to reduce the number of layers, reduce the number of nodes and add dropout points in our process. We also selected the Adam optimizer instead of the SGD optimizer that was used in our first models. Model 3 resulted in extremely good performance, considering the model only contained one million records. **Model 3** is likely the optimal model if we consider the cost/accuracy tradeoff of time and processing cost.

Model	# M Rows	Optimizer	Activation(s)	# Layers	Nodes
1	11	SGD	tanh,sigmoid	6	300x5,1
2	11	SGD	tanh,sigmoid	6	300x5,1
3	1	Adam	tanh,sigmoid	4	200,100x2,1
4	1	Adam	tanh,sigmoid	6	1000x2, 500x2,1
5	1	Adam	tanh,sigmoid	4	200,100X2,1

Model	# M Rows	Optimizer	Activation(s)	# Layers	Nodes
6	1	Adam	tanh,sigmoid	4	200,100X2,1
7	1	Adamax	tanh,sigmoid	4	200,100X2,1
8	1	SGD	tanh,sigmoid	4	200,100X2,1
9	1	Adam	tanh,sigmoid	4	200,100X2,1
10	1	Adam	tanh,sigmoid	4	600,300,150,1
11	1	Adam	swish, sigmoid	4	600,300,150,1
12	1	Adam	swish, sigmoid	4	200,100X2,1
13	1	Adam	softplus, sigmoid	4	600,300,150,1
14	11	Adam	swish, sigmoid	4	200,100X2,1

Methods Settings (Table 2)

We varied the loss function from binary_crossentropy to Hinge in Model 6. This reduced accuracy by 27 percent (decreasing from .7052 to .5300) as compared to Model 5. AUC also declined from .7772 to .5031. Clearly, for this dataset, the binary_crossentropy loss function is a more effective tool for modeling loss. This holds to reason, since binary_crossentropy is designed for binary classifications and our model is distinguishing signal from background. With the exception of Model 9, most of our variations yielded accuracy values less than 70%. We did see a positive effect in Model 9 relative to Model 8 when we removed dropout points. It is not clear why this would improve performance, other than to imagine that with a dataset of one million records, having more data in the model would improve accuracy relative to models that drop values within the model fitting process. We varied our optimization methods to analyze tanh, swish and softplus with all models using Sigmoid for the final layer. Swish did seem to improve accuracy relative to tanh, so it was chosen as the best activation function for our “full model”.

Results

Table 3 below shows the different models with the resulting accuracy and AUC values. In our experiment we looked for the optimal number of Epochs to process this data set.

Neural Networks Optimization Results

Model #	Accuracy	AUC	Best Epoch
1	0.7468	0.8287	10
2	0.7508	0.8326	13
3	0.7365	0.8154	4
4	0.7004	0.7700	3
5	0.7052	0.7772	5
6	0.5300	0.5031	1
7	0.6746	0.7388	2
8	0.6335	0.6782	10
9	0.7145	0.7879	10
10	0.6944	0.7695	6
11	0.6864	0.7553	2
12	0.6967	0.7651	3
13	0.6915	0.7601	7
14	0.7527	0.8342	6

Study Results (Table 3)

Conclusion

Deep Learning Advancements

Deep Learing has advanced since 2014 when the reference paper “Searching for Exotic Particles in High-energy Physics with Deep Learning” was published. The impovements in machine learning in the last seven years have been:

- **Transfer Learning is the idea of overcoming the isolated learning paradigm and utilizing knowledge acquired for one task to solve related ones**
- **Generative Adversarial Network (GAN) to transform facial images into video sequences**
- **Generative Pre-Training (GPT) to generate synthetic text automatically**

Suggested Improvements To Model

Our study confirmed that different datasets have different responses to the levers available in neural network models. For this dataset, the influence of the volume of data was the greatest factor. We achieved more accurate models using the binary_crossentropy loss function and the Adam or Swish optimizers. We also saw that “less is more” in that our models with fewer layers and fewer nodes were more accurate than models with more layers and more nodes. In the end, leveraging all eleven million data records, our most accurate model was Model 14 which used the reduced layer structure of Model 3, a reduced number of nodes relative to Models 4, 10, 11, 13 and the Adam optimizer like most of our models. The model did take significantly more time to run than Model 3 and the improvement on accuracy and AUC was only .0162 and .0188 respectively. Depending on the frequency the model is refreshed and the business value that 2% improvement of accuracy represents, the end-user may choose between models 3 and 14.

References

- [1] Baldi, P., P. Sadowski, and D. Whiteson. “Searching for Exotic Particles in High-energy Physics with Deep Learning.” Nature Communications 5 (July 2, 2014). https://arxiv.org/pdf/1402.4735.pdf (https://arxiv.org/pdf/1402.4735.pdf)

Appendix

Code

In [2]:
```
import pickle

import pandas as pd
import numpy as np
from scipy.stats import ttest_1samp
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras import initializers
from tensorflow.keras import callbacks
from tensorflow.keras import backend as K
from tensorflow.keras.regularizers import l2
from tensorflow.keras.models import Model
print(tf.__version__)

auc_score = tf.keras.metrics.AUC()
seed = 42
```

```
2.4.1
```

In [3]:
```
# Set Directory for image files - Comment out if you are not LL or MM
#os.chdir("C://Users/18322/OneDrive - Southern Methodist University/Desk
top/QOW/Case Study 10")
#os.chdir('C:\\SMU_Local')
```

In [15]:
```
import pandas as pd
import numpy as np
col_names = ['class_label','lepton_pT','lepton_eta','lepton_phi','missin
g_energy_magnitude','missing_energy_phi','jet_1_pt','jet_1_eta','jet_1_p
hi','jet_1_b-tag','jet_2_pt','jet_2_eta','jet_2_phi','jet_2_b-tag','jet_
3_pt','jet_3_eta','jet_3_phi','jet_3_b-tag','jet_4_pt','jet_4_eta','jet_
4_phi','jet_4_b-tag','m_jj','m_jjj','m_lv','m_jlv','m_bb','m_wbb','m_wwb
b']
data = pd.read_csv(r"./data/HIGGS.csv.gz", header=None, names=col_names,
nrows=1000000, dtype=np.float32, compression='gzip')
```

In [16]:
```
y = data.class_label
```

In [17]:
```
data.drop(['class_label'], axis=1, inplace=True)
```

In [18]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 28 columns):
 #   Column                    Non-Null Count    Dtype  
---  ------                    --------------    -----  
 0   lepton_pT                 1000000 non-null  float32
 1   lepton_eta                1000000 non-null  float32
 2   lepton_phi                1000000 non-null  float32
 3   missing_energy_magnitude  1000000 non-null  float32
 4   missing_energy_phi        1000000 non-null  float32
 5   jet_1_pt                  1000000 non-null  float32
 6   jet_1_eta                 1000000 non-null  float32
 7   jet_1_phi                 1000000 non-null  float32
 8   jet_1_b-tag               1000000 non-null  float32
 9   jet_2_pt                  1000000 non-null  float32
 10  jet_2_eta                 1000000 non-null  float32
 11  jet_2_phi                 1000000 non-null  float32
 12  jet_2_b-tag               1000000 non-null  float32
 13  jet_3_pt                  1000000 non-null  float32
 14  jet_3_eta                 1000000 non-null  float32
 15  jet_3_phi                 1000000 non-null  float32
 16  jet_3_b-tag               1000000 non-null  float32
 17  jet_4_pt                  1000000 non-null  float32
 18  jet_4_eta                 1000000 non-null  float32
 19  jet_4_phi                 1000000 non-null  float32
 20  jet_4_b-tag               1000000 non-null  float32
 21  m_jj                      1000000 non-null  float32
 22  m_jjj                     1000000 non-null  float32
 23  m_lv                      1000000 non-null  float32
 24  m_jlv                     1000000 non-null  float32
 25  m_bb                      1000000 non-null  float32
 26  m_wbb                     1000000 non-null  float32
 27  m_wwbb                    1000000 non-null  float32
dtypes: float32(28)
memory usage: 106.8 MB
```

In [21]:
```
# Scale Data
# Neural Networks are especially sensitive to data scaling.  Nearly all
 the activation functions saturate at (0,1) or (-1,1)from sklearn.prepro
cessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))
scaled_train = scaler.fit_transform(data)
```

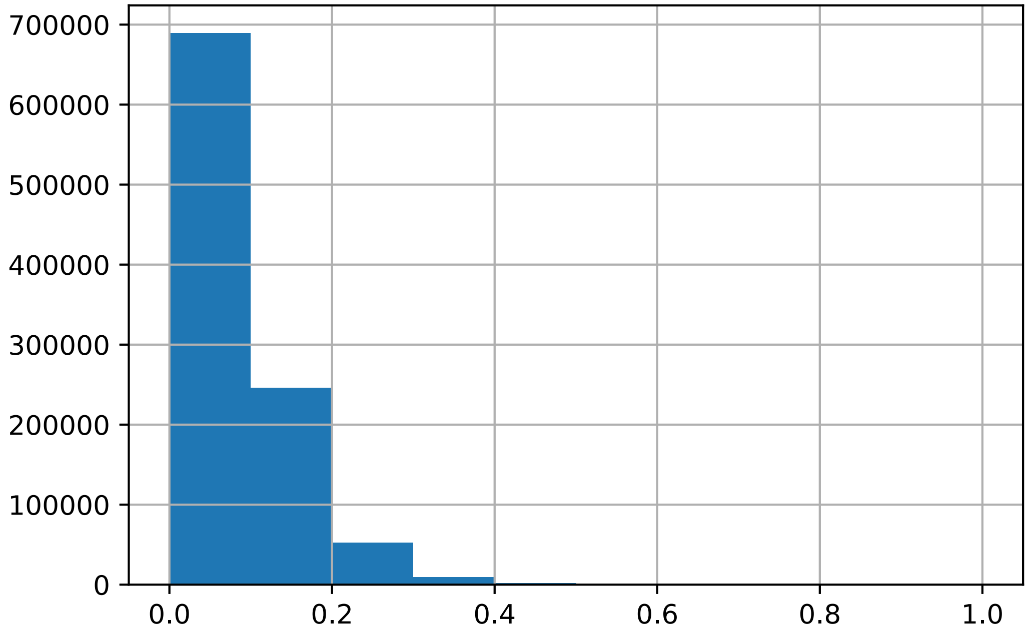
In [22]:
```
# Print out the adjustment that the scaler applied to the total_earnings
column of data
print("Note: median values were scaled by multiplying by {:.10f} and add
ing {:.6f}".format(scaler.scale_[7], scaler.min_[7]))
multiplied_by = scaler.scale_[7]
added = scaler.min_[7]

scaled_train_df = pd.DataFrame(scaled_train, columns=data.columns.values
)
```

```
Note: median values were scaled by multiplying by 0.2871342599 and addi
ng 0.499969
```

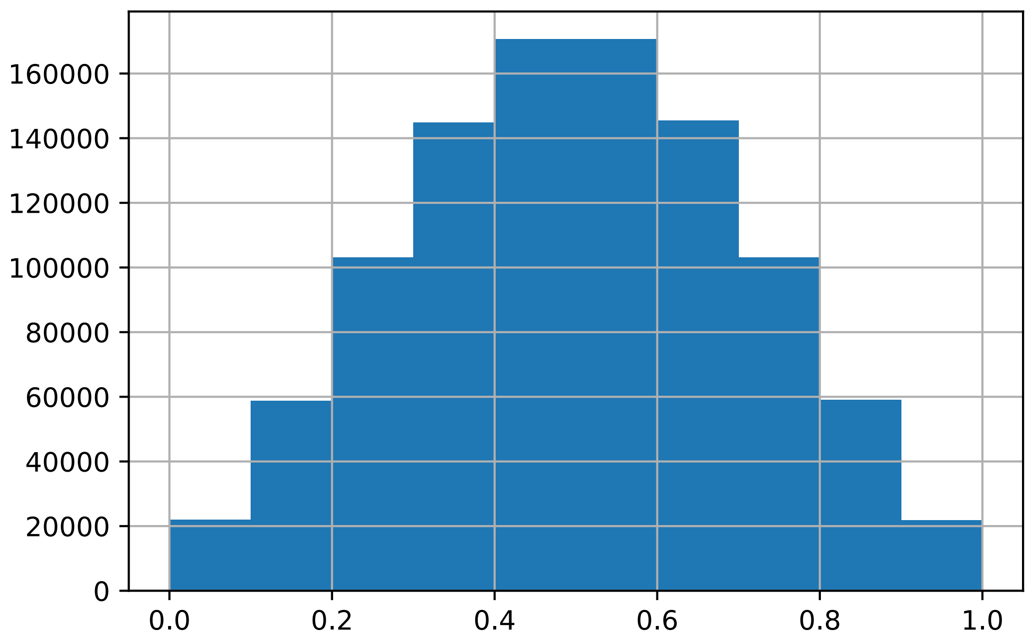
In [ ]:
```
```

In [23]:

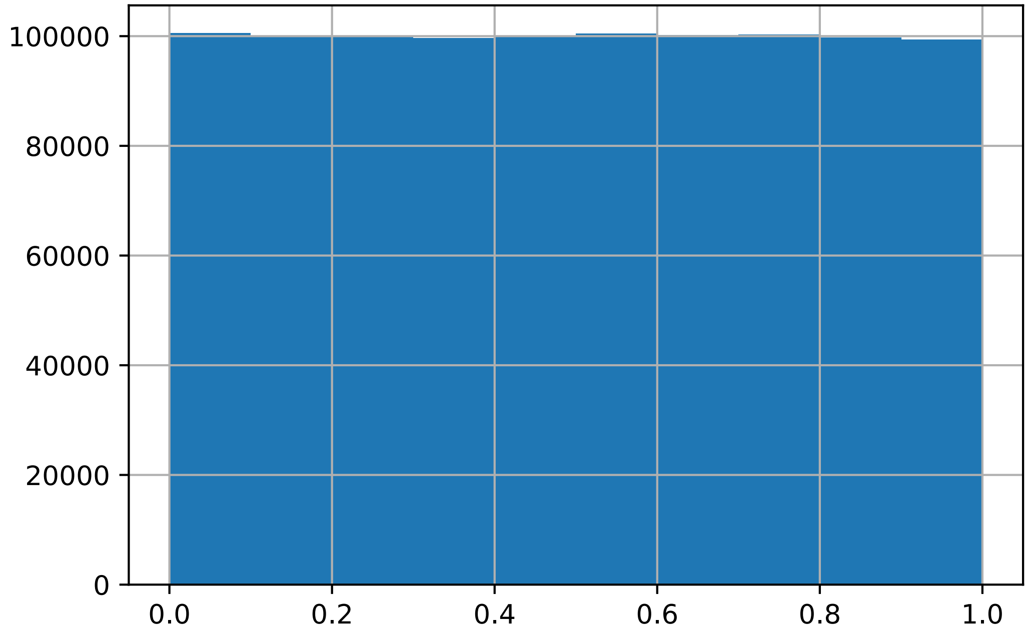
```
import matplotlib.pyplot as plt
%matplotlib inline
for i in scaled_train_df:
    scaled_train_df[i].hist()
    plt.show()
```

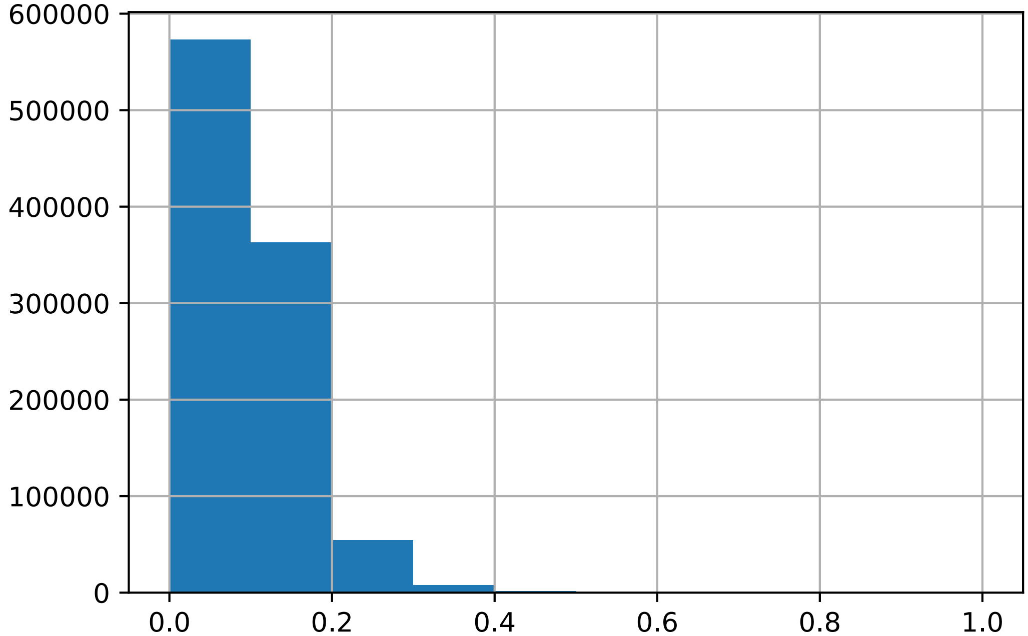


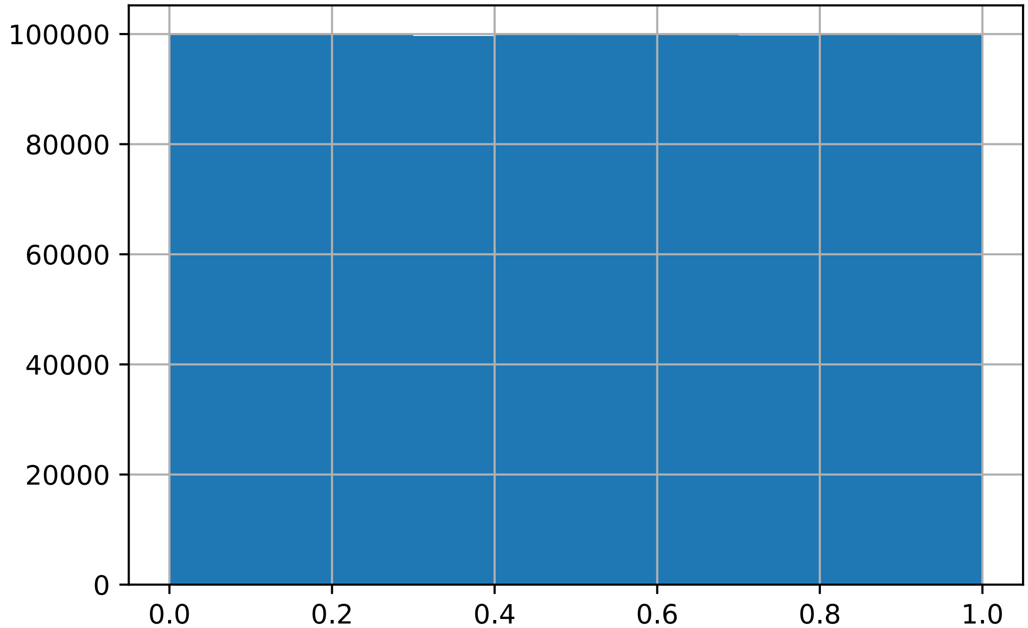
700000
600000
500000
400000
300000
200000
100000
0
0.0
0.2
0.4
0.6
0.8
1.0

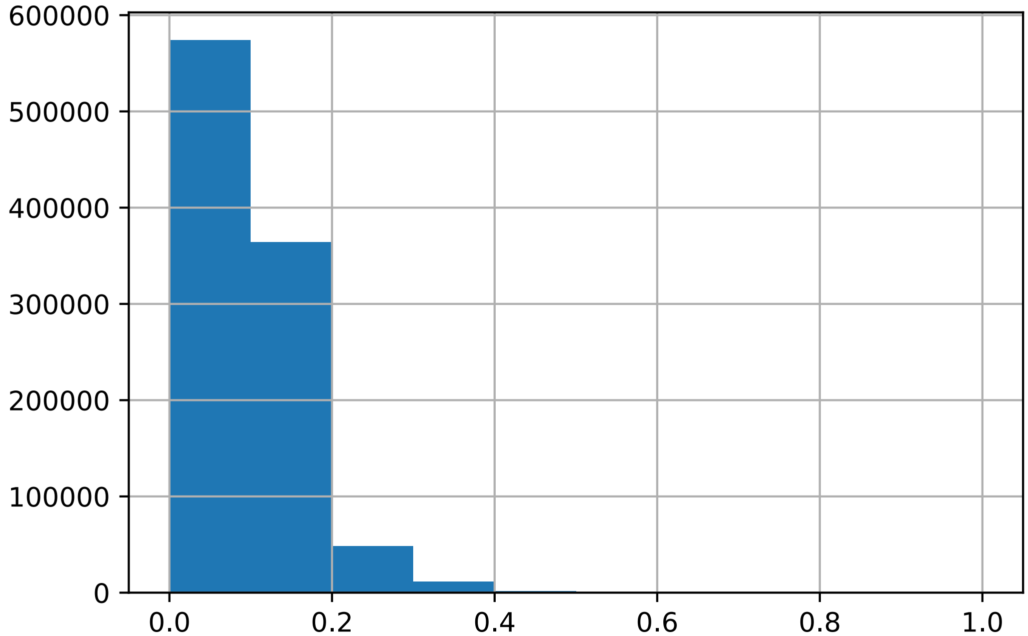


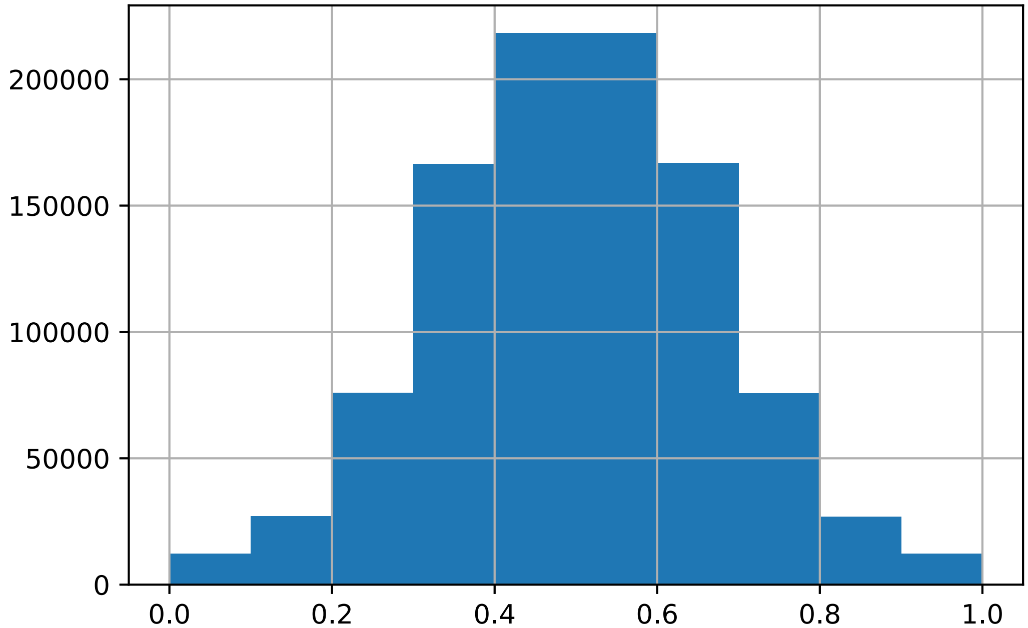
160000
140000
120000
100000
80000
60000
40000
20000
0
0.0
0.2
0.4
0.6
0.8
1.0

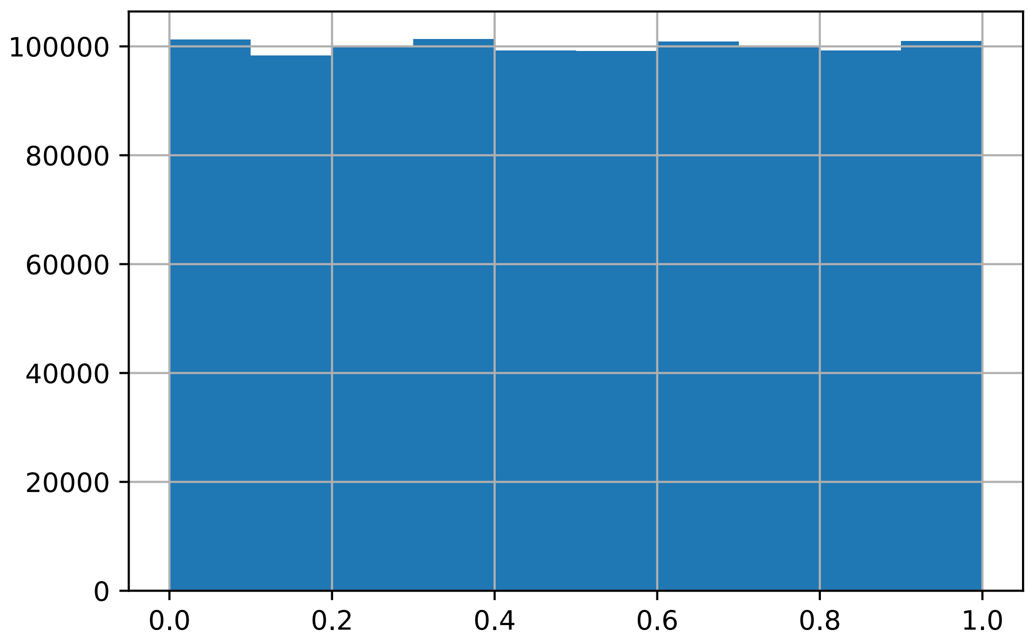

100000
80000
60000
40000
20000
0
0.0
0.2
0.4
0.6
0.8
1.0

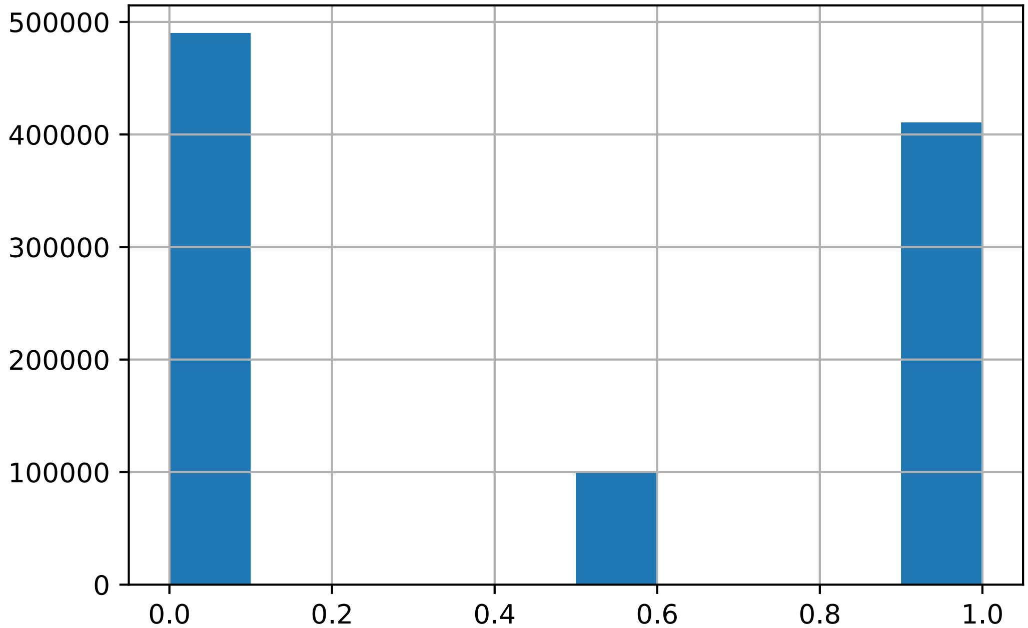

600000
500000
400000
300000
200000
100000
0
0.0
0.2
0.4
0.6
0.8
1.0


100000
80000
60000
40000
20000
0
0.0
0.2
0.4
0.6
0.8
1.0

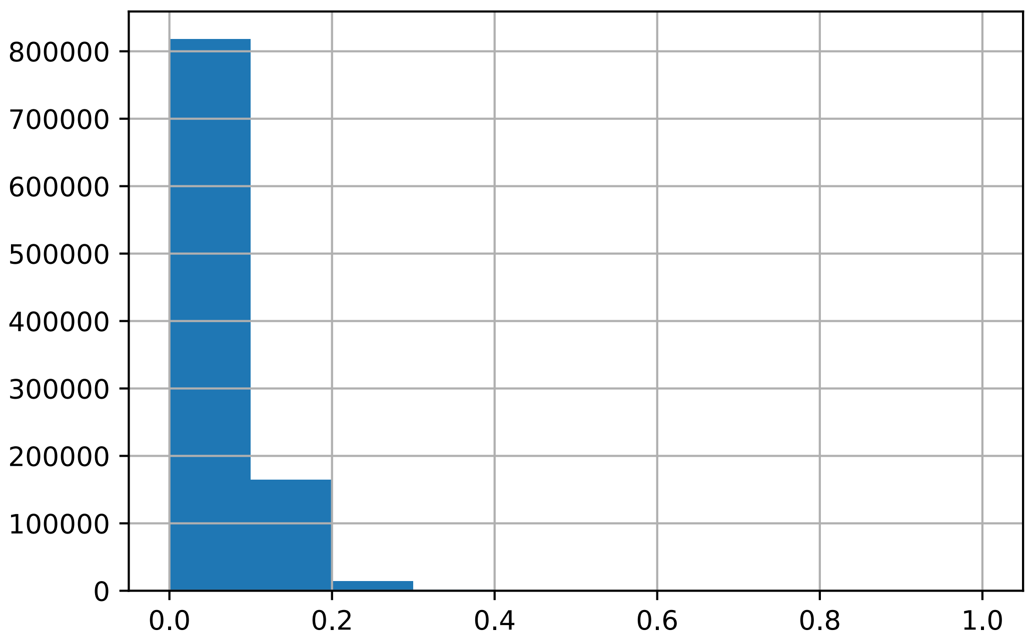

600000
500000
400000
300000
200000
100000
0
0.0
0.2
0.4
0.6
0.8
1.0


200000
150000
100000
50000
0
0.0
0.2
0.4
0.6
0.8
1.0

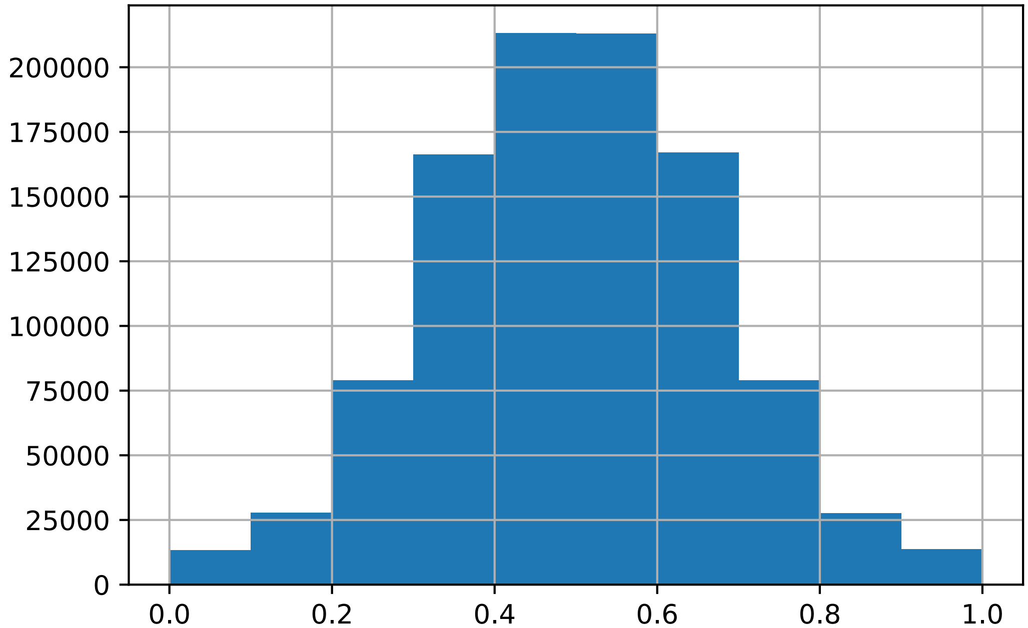

100000
80000
60000
40000
20000
0
0.0
0.2
0.4
0.6
0.8
1.0

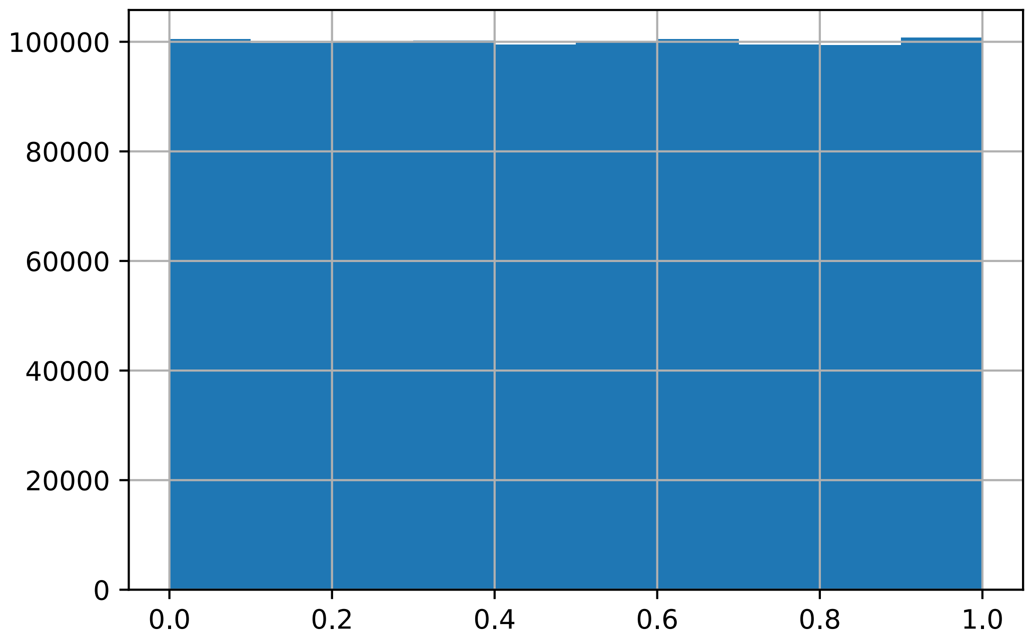


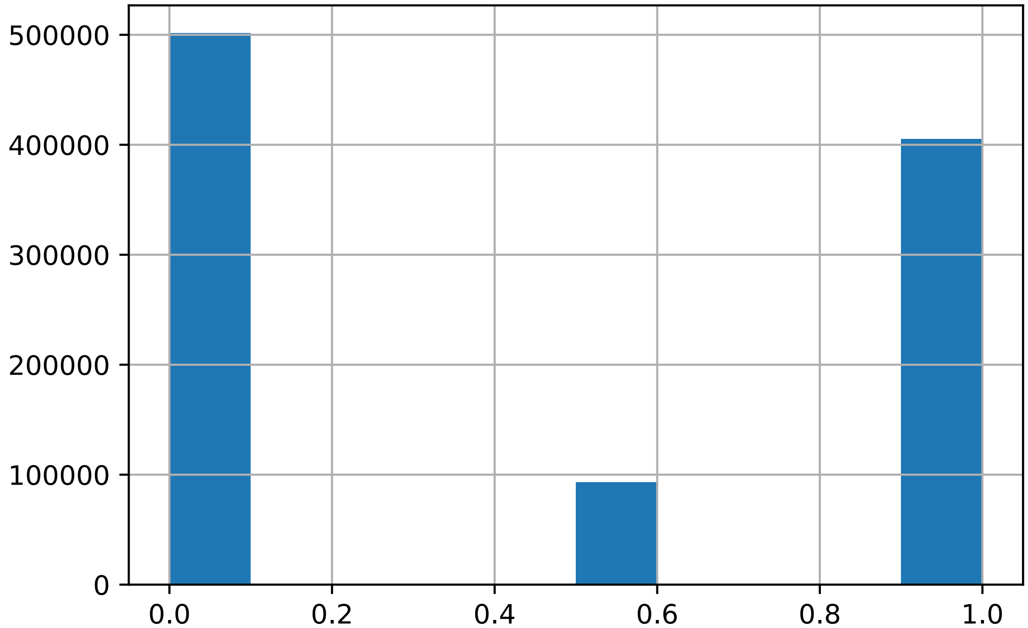
500000
400000
300000
200000
100000
0
0.0
0.2
0.4
0.6
0.8
1.0

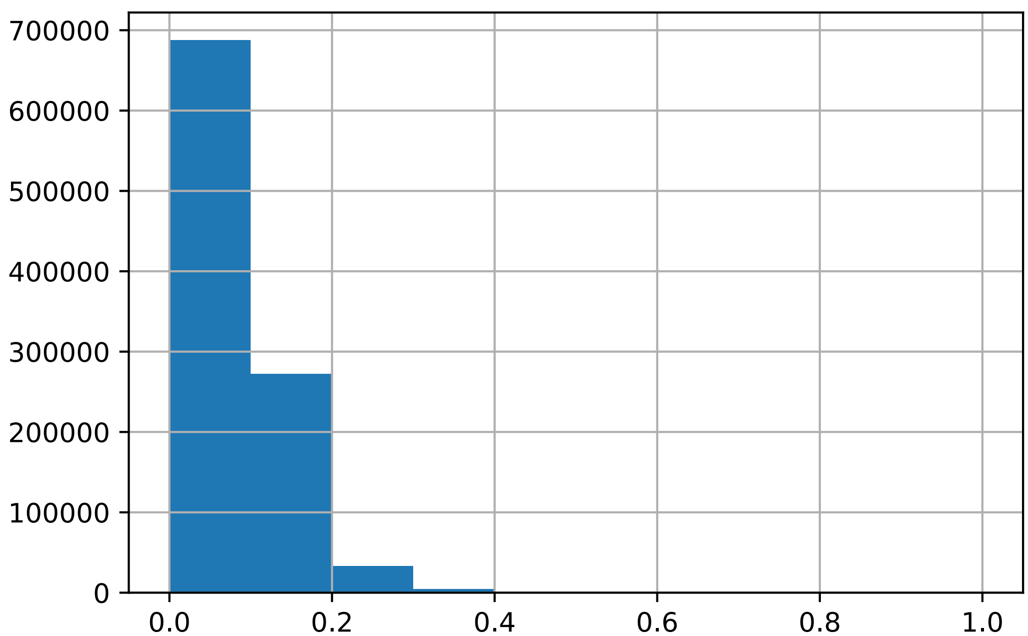


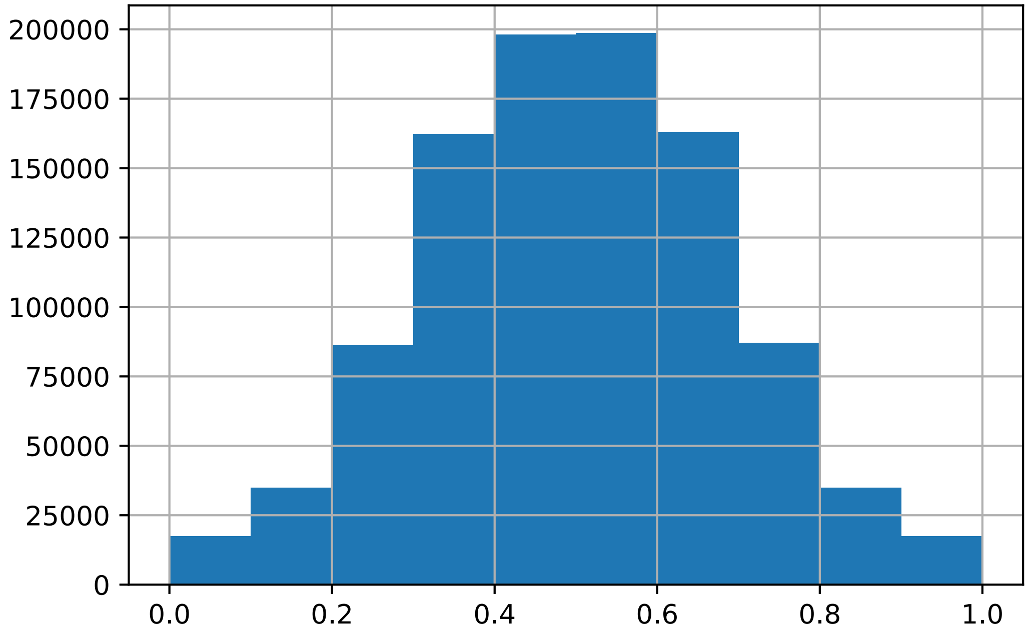
800000
700000
600000
500000
400000
300000
200000
100000
0
0.0
0.2
0.4
0.6
0.8
1.0


200000
175000
150000
125000
100000
75000
50000
25000
0
0.0
0.2
0.4
0.6
0.8
1.0

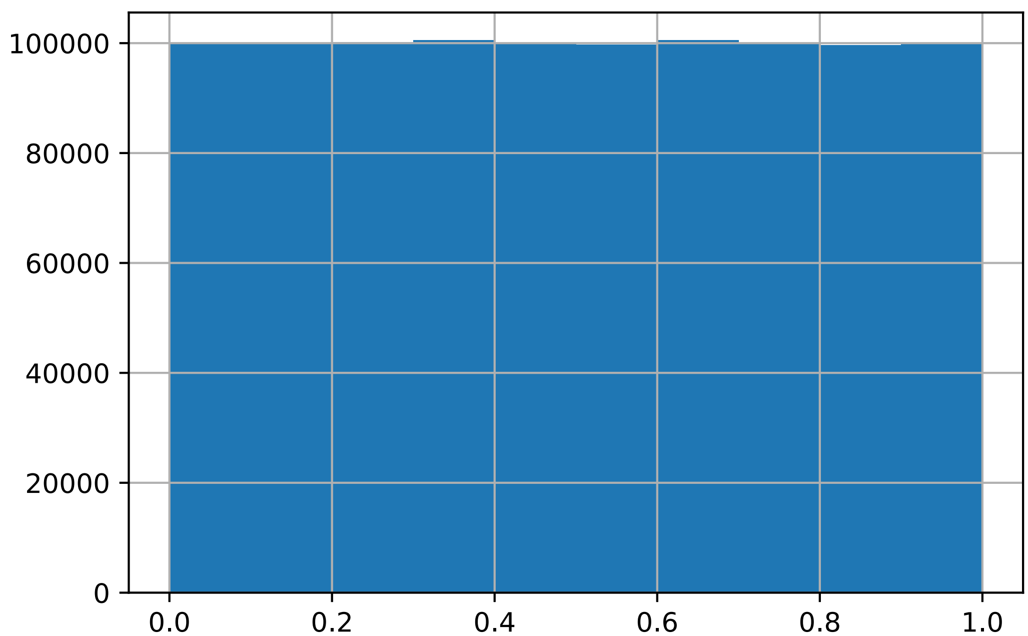

100000
80000
60000
40000
20000
0
0.0
0.2
0.4
0.6
0.8
1.0


500000
400000
300000
200000
100000
0
0.0
0.2
0.4
0.6
0.8
1.0

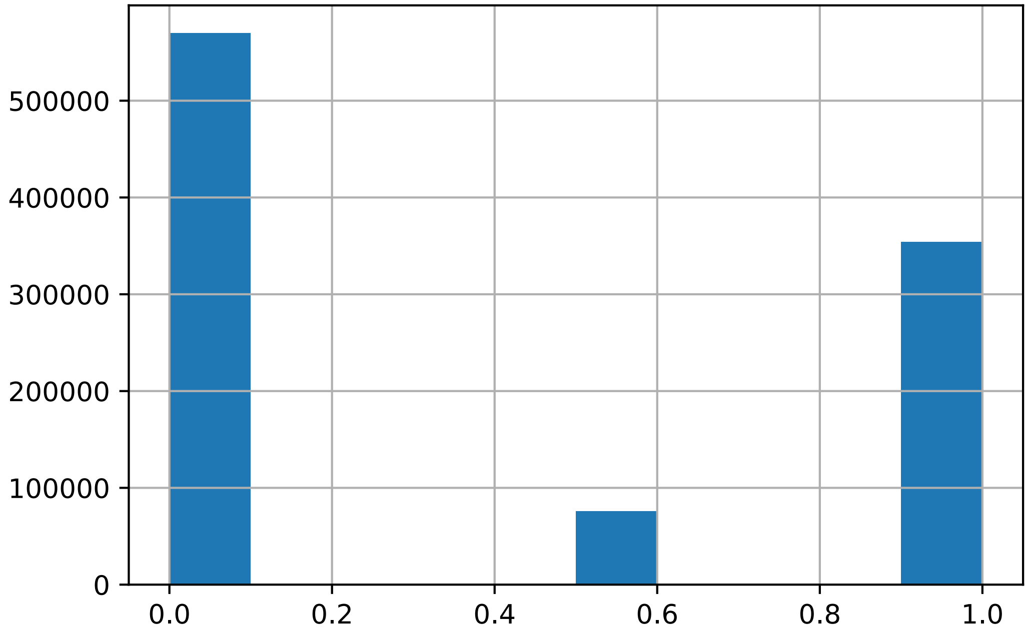

700000
600000
500000
400000
300000
200000
100000
0
0.0
0.2
0.4
0.6
0.8
1.0

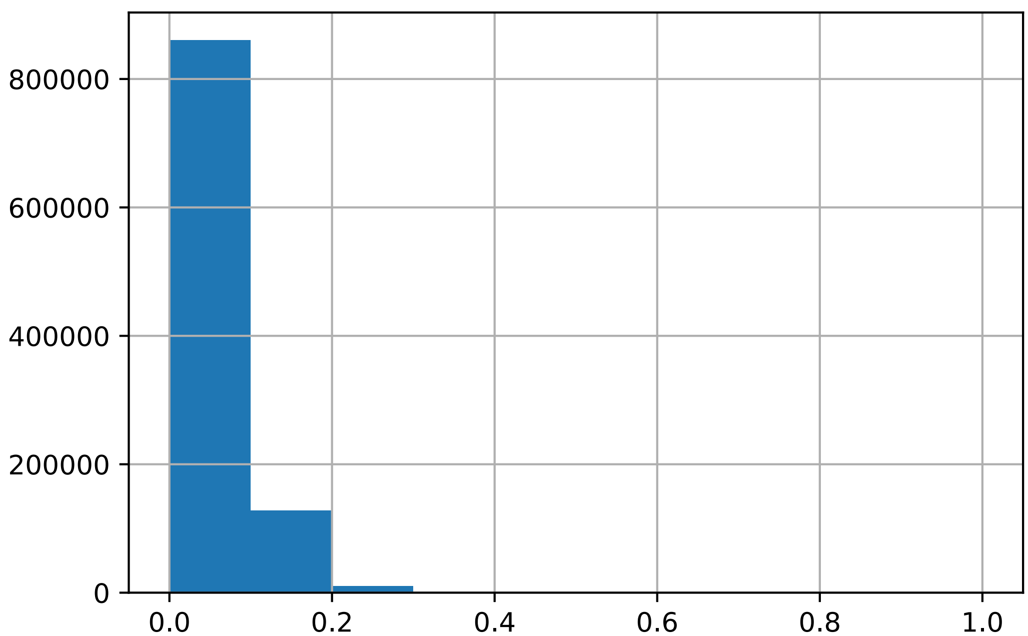


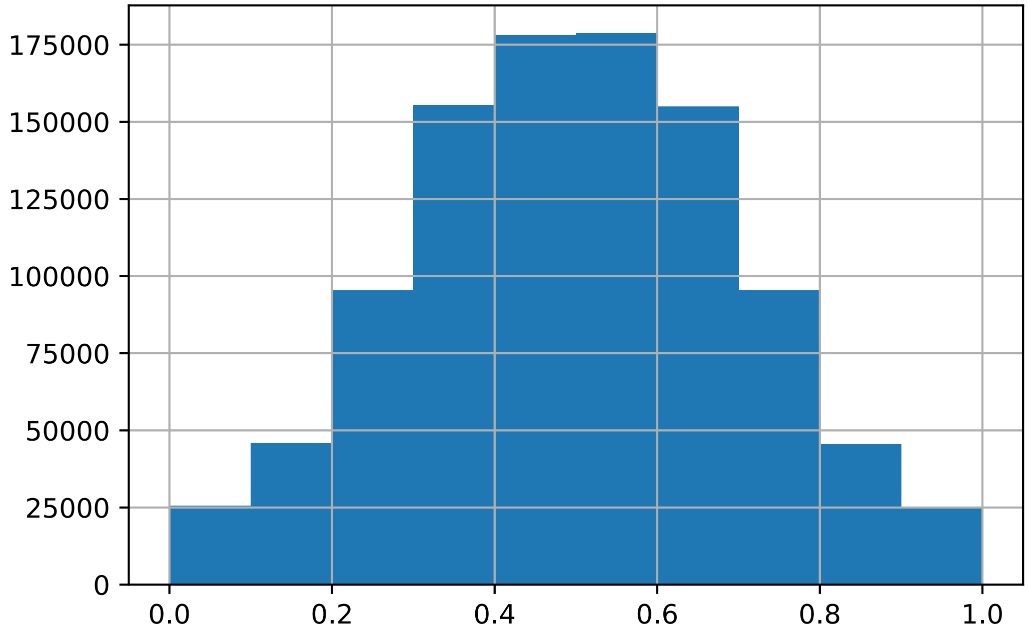
200000
175000
150000
125000
100000
75000
50000
25000
0
0.0
0.2
0.4
0.6
0.8
1.0

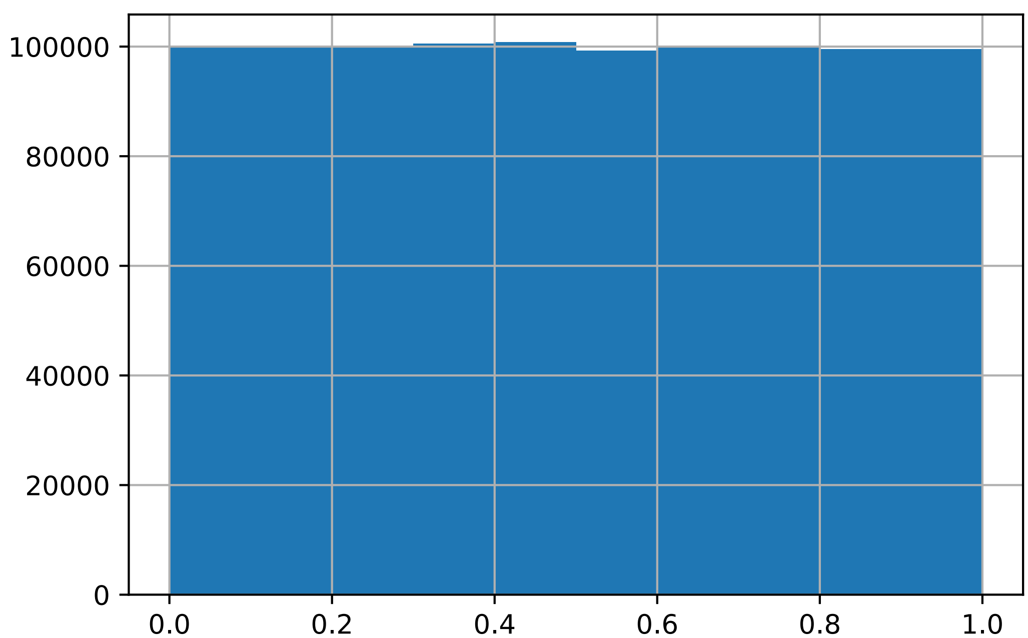


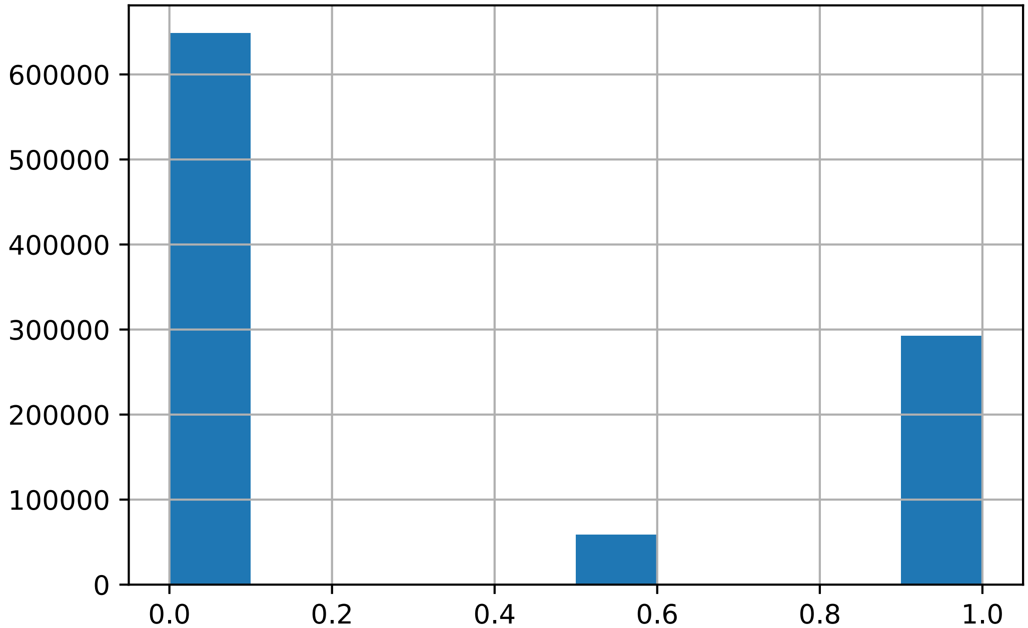
100000
80000
60000
40000
20000
0
0.0
0.2
0.4
0.6
0.8
1.0


500000
400000
300000
200000
100000
0
0.0
0.2
0.4
0.6
0.8
1.0

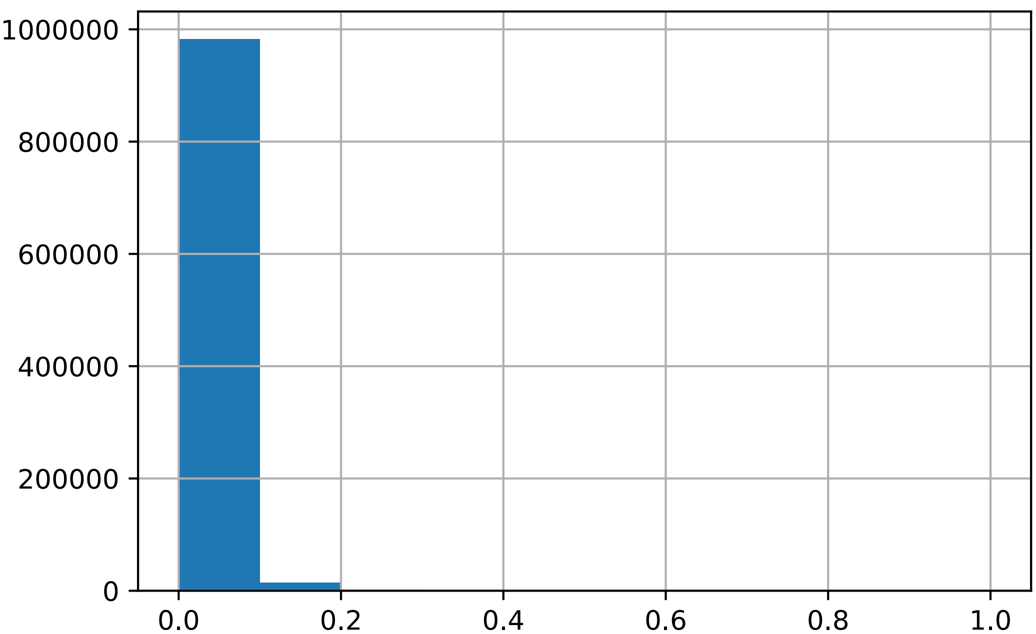

800000
600000
400000
200000
0
0.0
0.2
0.4
0.6
0.8
1.0


175000
150000
125000
100000
75000
50000
25000
0
0.0
0.2
0.4
0.6
0.8
1.0

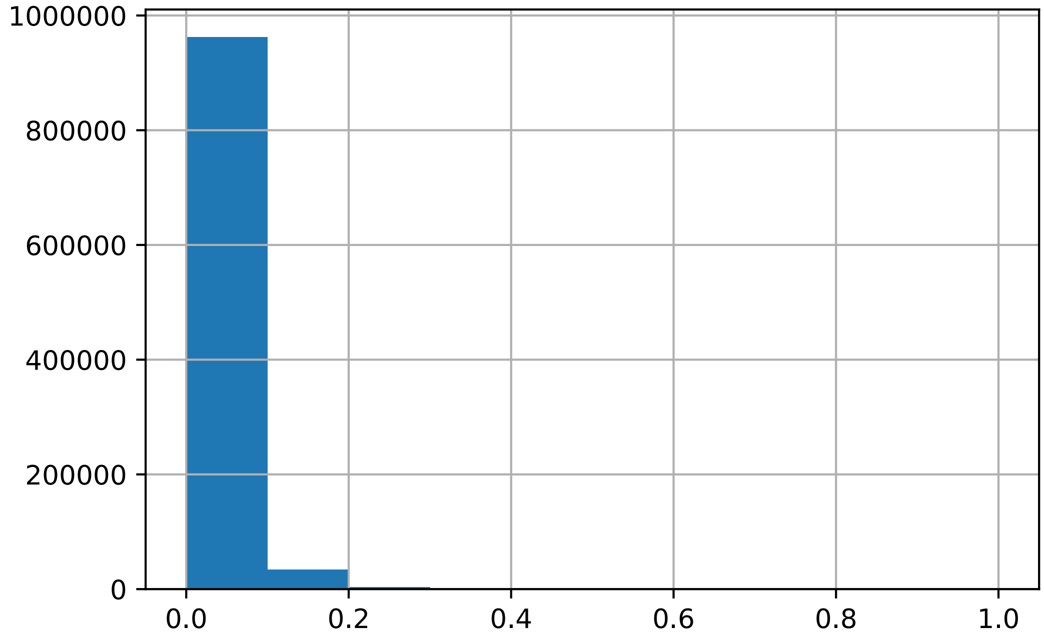

100000
80000
60000
40000
20000
0
0.0
0.2
0.4
0.6
0.8
1.0



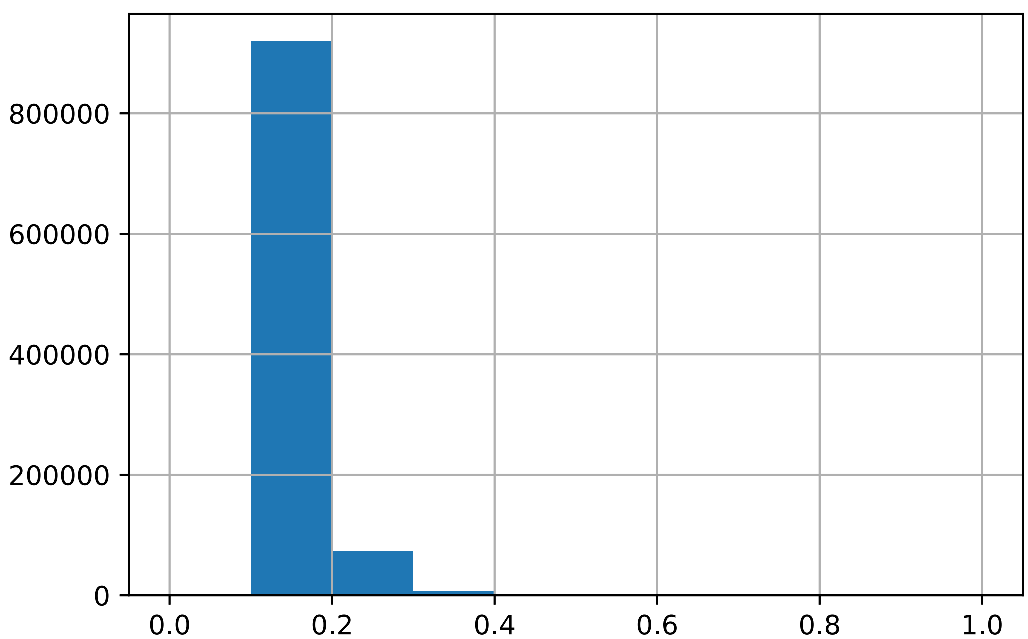
0
100000
200000
300000
400000
500000
600000
0.0
0.2
0.4
0.6
0.8
1.0



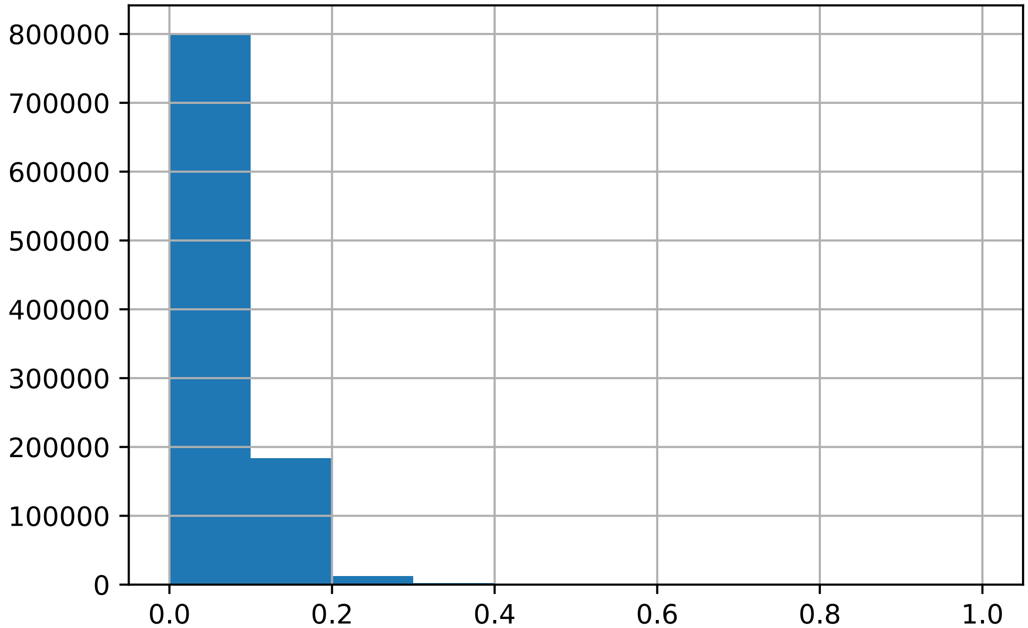
0
200000
400000
600000
800000
1000000
0.0
0.2
0.4
0.6
0.8
1.0



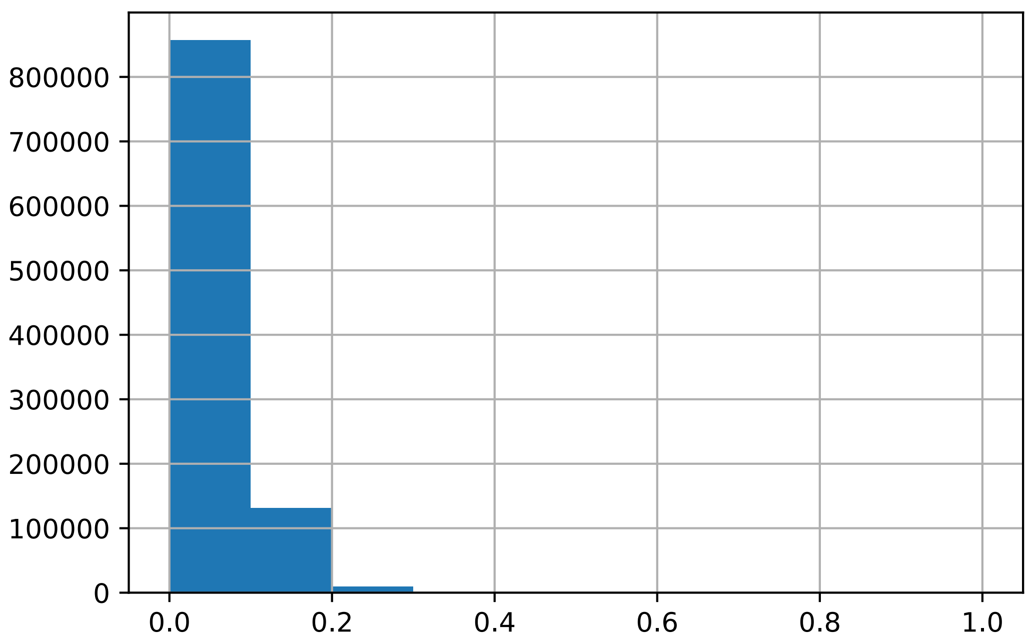
1000000
800000
600000
400000
200000
0
0.0
0.2
0.4
0.6
0.8
1.0



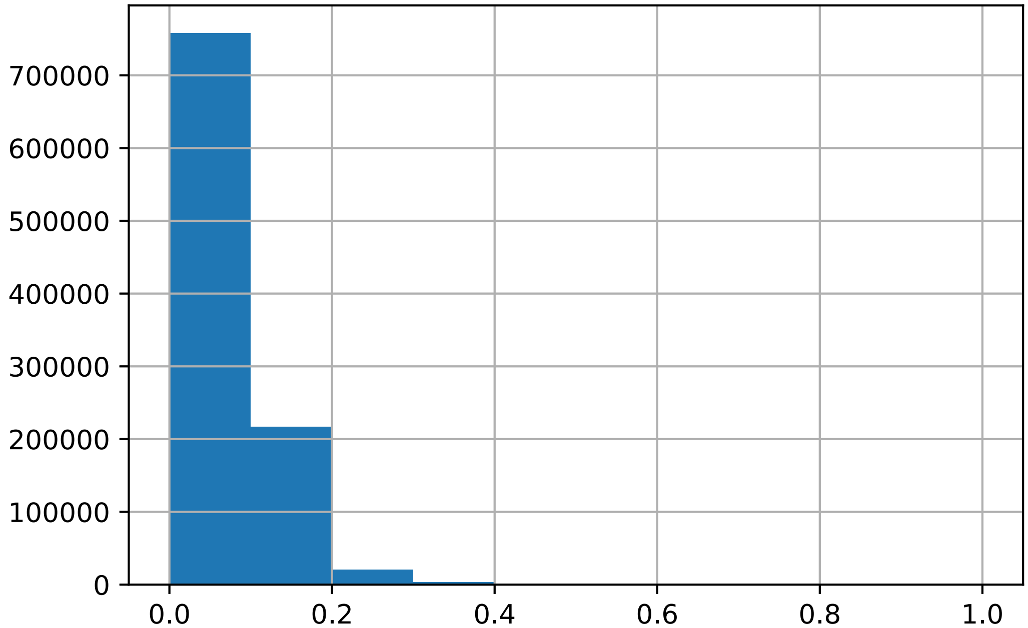
800000
600000
400000
200000
0
0.0
0.2
0.4
0.6
0.8
1.0

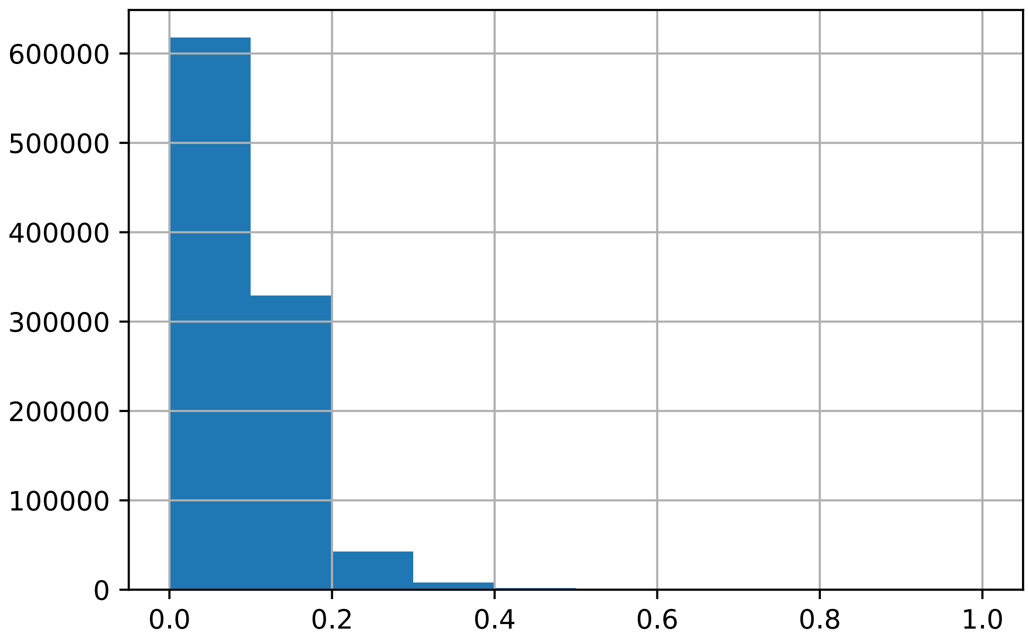


800000
700000
600000
500000
400000
300000
200000
100000
0
0.0
0.2
0.4
0.6
0.8
1.0



800000
700000
600000
500000
400000
300000
200000
100000
0
0.0
0.2
0.4
0.6
0.8
1.0





In [24]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(scaled_train_df, y,
test_size=0.20, random_state=1776)
x_train.shape
```

Out[24]: (800000, 28)

In [26]:
```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.regularizers import l2
from tensorflow.keras import initializers
from sklearn import datasets
import sklearn
```

In [27]:
```
###########################################################
####         Model Design Replicate Higgs             ####
###########################################################
```

In [28]:
```
seed = 42
first_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.1, seed=seed
)
hidden_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.05, seed=seed
)
output_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.001, seed=seed
)

weight_decay = 1 * 10**-5

# build model with 5 layers of 300 neurons, using tanh activation, and l
2 regularization
# final layer uses sigmoid activation to align with 0/1 target values
model = tf.keras.Sequential()
model.add(layers.Dense(300,activation='tanh',kernel_initializer=first_la
yer_init,kernel_regularizer=l2(weight_decay)))  # adds a layer with 300
 neurons, tanh activation
model.add(layers.Dense(300,activation='tanh',kernel_initializer=hidden_l
ayer_init,kernel_regularizer=l2(weight_decay)))  # adds a layer with 300
neurons, tanh activation
model.add(layers.Dense(300,activation='tanh',kernel_initializer=hidden_l
ayer_init,kernel_regularizer=l2(weight_decay)))  # adds a layer with 300
neurons, tanh activation
model.add(layers.Dense(300,activation='tanh',kernel_initializer=hidden_l
ayer_init,kernel_regularizer=l2(weight_decay)))  # adds a layer with 300
neurons, tanh activation
model.add(layers.Dense(300,activation='tanh',kernel_initializer=hidden_l
ayer_init,kernel_regularizer=l2(weight_decay)))  # adds a layer with 300
neurons, tanh activation
model.add(layers.Dense(1, activation='sigmoid',kernel_initializer=output
_layer_init,kernel_regularizer=l2(weight_decay)))  # adds a layer with 1
neurons, sigmoid activation
```

In [29]:
```
auc_score = tf.keras.metrics.AUC() # define AUC score for model output
model.compile(optimizer=optimizers.SGD(lr=.05), loss='binary_crossentrop
y', metrics=['accuracy',auc_score]) # optimized with SGD with a learning
rate of .05
callbacks = [EarlyStopping(patience=2, monitor='val_accuracy', min_delta
=0.00001)]  # stop model after 2 epochs with no improvement based on epo
ch accuracy
```

In [30]:
```
model.fit(x_train, y_train, epochs=20, validation_data=(x_test,y_test),
batch_size=50,callbacks=callbacks) # 20 Epochs with a batch size of 50
```

```
Epoch 1/10
32000/32000 [==============================] - 83s 3ms/step - loss: 0.6
778 - accuracy: 0.5629 - auc_2: 0.5828 - val_loss: 0.6542 - val_accurac
y: 0.6118 - val_auc_2: 0.6683
Epoch 2/10
32000/32000 [==============================] - 77s 2ms/step - loss: 0.6
498 - accuracy: 0.6160 - auc_2: 0.6571 - val_loss: 0.6451 - val_accurac
y: 0.6260 - val_auc_2: 0.6713
Epoch 3/10
32000/32000 [==============================] - 98s 3ms/step - loss: 0.6
449 - accuracy: 0.6248 - auc_2: 0.6667 - val_loss: 0.6423 - val_accurac
y: 0.6259 - val_auc_2: 0.6748
Epoch 4/10
32000/32000 [==============================] - 92s 3ms/step - loss: 0.6
429 - accuracy: 0.6284 - auc_2: 0.6707 - val_loss: 0.6393 - val_accurac
y: 0.6348 - val_auc_2: 0.6794
Epoch 5/10
32000/32000 [==============================] - 95s 3ms/step - loss: 0.6
418 - accuracy: 0.6298 - auc_2: 0.6726 - val_loss: 0.6437 - val_accurac
y: 0.6274 - val_auc_2: 0.6800
Epoch 6/10
32000/32000 [==============================] - 100s 3ms/step - loss: 0.
6399 - accuracy: 0.6326 - auc_2: 0.6752 - val_loss: 0.6347 - val_accura
cy: 0.6405 - val_auc_2: 0.6835
Epoch 7/10
32000/32000 [==============================] - 99s 3ms/step - loss: 0.6
370 - accuracy: 0.6368 - auc_2: 0.6800 - val_loss: 0.6311 - val_accurac
y: 0.6431 - val_auc_2: 0.6938
Epoch 8/10
32000/32000 [==============================] - 89s 3ms/step - loss: 0.6
309 - accuracy: 0.6406 - auc_2: 0.6924 - val_loss: 0.6225 - val_accurac
y: 0.6486 - val_auc_2: 0.7089
Epoch 9/10
32000/32000 [==============================] - 85s 3ms/step - loss: 0.6
207 - accuracy: 0.6524 - auc_2: 0.7090 - val_loss: 0.6088 - val_accurac
y: 0.6677 - val_auc_2: 0.7285
Epoch 10/10
32000/32000 [==============================] - 85s 3ms/step - loss: 0.6
149 - accuracy: 0.6593 - auc_2: 0.7176 - val_loss: 0.6204 - val_accurac
y: 0.6492 - val_auc_2: 0.7085
```

Out[30]:
```
<tensorflow.python.keras.callbacks.History at 0x1a5b67d5d0>
```

In [31]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_5 (Dense)              (25, 300)                 8700      
_________________________________________________________________
dense_6 (Dense)              (25, 300)                 90300     
_________________________________________________________________
dense_7 (Dense)              (25, 300)                 90300     
_________________________________________________________________
dense_8 (Dense)              (25, 300)                 90300     
_________________________________________________________________
dense_9 (Dense)              (25, 1)                   301       
=================================================================
Total params: 279,901
Trainable params: 279,901
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
###########################################################
####         Optimize TensorFlow Design               ####
###########################################################
```

In [ ]:
```
seed = 42
first_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.1, seed=seed
)
hidden_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.05, seed=seed
)
output_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.001, seed=seed
)
model = tf.keras.Sequential()
# model.add(tf.keras.Input(shape=(28,)))
model.add(layers.Dense(300,activation='tanh',kernel_initializer=first_la
yer_init))  # adds a layer with 300 neurons, tanh activation
model.add(layers.Dense(300,activation='tanh',kernel_initializer=hidden_l
ayer_init))  # adds a layer with 300 neurons, tanh activation
model.add(layers.Dense(300,activation='tanh',kernel_initializer=hidden_l
ayer_init))  # adds a layer with 300 neurons, tanh activation
model.add(layers.Dense(300,activation='tanh',kernel_initializer=hidden_l
ayer_init))  # adds a layer with 300 neurons, tanh activation
model.add(layers.Dense(1, activation='sigmoid',kernel_initializer=output
_layer_init))  # adds a layer with 1 neurons, sigmoid activation
```

In [ ]:
```
auc_score = tf.keras.metrics.AUC()
model.compile(optimizer='sgd',
              loss='binary_crossentropy',
              metrics=['accuracy',auc_score])
```

In [ ]:
```
model.fit(x_train, y_train, epochs=10, validation_data=(x_test,y_test),
batch_size=25)
```

In [ ]:
```
model.summary()
```

In [ ]:
```
##########################################################
####          Optimize TensorFlow Design              ####
##########################################################

# Model
# We will use the Sequential() class to build all models.
```

In [ ]:
```
# Model 5:  Base Model
```

In [ ]:
```
# MM Base Model
seed = 42
first_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.1, seed=seed
)
hidden_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.05, seed=seed
)
output_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.001, seed=seed
)
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(28,)))
model.add(layers.Dense(200,activation='tanh'))
keras.layers.Dropout(0.4),
model.add(layers.Dense(100,activation='tanh'))
keras.layers.Dropout(0.4),
model.add(layers.Dense(100,activation='tanh')),
model.add(layers.Dense(1, activation='sigmoid'))
```

In [ ]:
```
# MM
auc_score = tf.keras.metrics.AUC()
model.compile(optimizer=optimizers.Adam(learning_rate=.001),
              loss='binary_crossentropy',
              metrics=['accuracy',auc_score])  #removed 'binary_crossent
ropy','mean_absolute_error'
```

In [ ]:
```
# Fit model
# Now it is time to train
#MM
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlatea
u, EarlyStopping
callbacks = [EarlyStopping( patience=1)]
model.fit(x_train,y_train, epochs=20, validation_data=(x_test,y_test), b
atch_size=50,callbacks=callbacks)
```

In [ ]:
```
#MM
model.summary()
```

In [ ]:
```
train_loss = model.history.history['loss']
val_loss = model.history.history['val_loss']
plt.plot(train_loss,label='train_loss')
plt.plot(val_loss,label='valid_loss')
plt.title("Higgs Model with Adam Opt and Binary Cross-Entropy Loss")
legend = plt.legend(loc='upper center', shadow=True, fontsize='x-large')
plt.show();
```

In [ ]:
```

```

In [ ]:

```
## Model 6: Base Model, Change Loss Function from binary_crossentropy to
Hinge

# MM Model
seed = 42
first_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.1, seed=seed
)
hidden_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.05, seed=seed
)
output_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.001, seed=seed
)
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(28,)))
model.add(layers.Dense(200,activation='tanh'))
keras.layers.Dropout(0.4),
model.add(layers.Dense(100,activation='tanh'))
keras.layers.Dropout(0.4),
model.add(layers.Dense(100,activation='tanh')),
model.add(layers.Dense(1, activation='sigmoid'))

# MM Hinge variation
auc_score = tf.keras.metrics.AUC()
model.compile(optimizer=optimizers.Adam(learning_rate=.001),
              loss='Hinge',
              metrics=['accuracy',auc_score])

#MM
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlatea
u, EarlyStopping
callbacks = [EarlyStopping( patience=1)]
model.fit(x_train,y_train, epochs=10, validation_data=(x_test,y_test), b
atch_size=50,callbacks=callbacks)

#MM
model.summary()

# Visualize
train_loss = model.history.history['loss']
val_loss = model.history.history['val_loss']
plt.plot(train_loss,label='train_loss')
plt.plot(val_loss,label='valid_loss')
plt.title("Higgs Model with Adam Opt and Hinge Loss")
legend = plt.legend(loc='upper center', shadow=True, fontsize='x-large')
plt.show();

# Model 7: Base Model, Change Optimizer Function from Adam to Adamax, Lo
ss = binary_crossentropy

# MM Model
seed = 42
first_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.1, seed=seed
)
```

```
hidden_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.05, seed=seed
)
output_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.001, seed=seed
)
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(28,)))
model.add(layers.Dense(200,activation='tanh'))
keras.layers.Dropout(0.4),
model.add(layers.Dense(100,activation='tanh'))
keras.layers.Dropout(0.4),
model.add(layers.Dense(100,activation='tanh')),
model.add(layers.Dense(1, activation='sigmoid'))

# MM Adamax variation
auc_score = tf.keras.metrics.AUC()
model.compile(optimizer=optimizers.Adamax(learning_rate=.001),
              loss='binary_crossentropy',
              metrics=['accuracy',auc_score])

#MM
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlatea
u, EarlyStopping
callbacks = [EarlyStopping( patience=1)]
model.fit(x_train,y_train, epochs=10, validation_data=(x_test,y_test), b
atch_size=50,callbacks=callbacks)

#MM
model.summary()

# Visualize
train_loss = model.history.history['loss']
val_loss = model.history.history['val_loss']
plt.plot(train_loss,label='train_loss')
plt.plot(val_loss,label='valid_loss')
plt.title("Higgs Model with Adamax Opt and Binary Cross-Entropy Loss")
legend = plt.legend(loc='upper center', shadow=True, fontsize='x-large')
plt.show();

# Model 8: Change Optimizer to SGD with binary_crossentropy loss

# MM Model
seed = 42
first_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.1, seed=seed
)
hidden_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.05, seed=seed
)
output_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.001, seed=seed
)
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(28,)))
model.add(layers.Dense(200,activation='tanh'))
keras.layers.Dropout(0.4),
```

```
model.add(layers.Dense(100,activation='tanh'))
keras.layers.Dropout(0.4),
model.add(layers.Dense(100,activation='tanh')),
model.add(layers.Dense(1, activation='sigmoid'))

# MM SGD variation
auc_score = tf.keras.metrics.AUC()
model.compile(optimizer=optimizers.SGD(learning_rate=.001),
              loss='binary_crossentropy',
              metrics=['accuracy',auc_score])

#MM Fit
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlatea
u, EarlyStopping
callbacks = [EarlyStopping( patience=1)]
model.fit(x_train,y_train, epochs=10, validation_data=(x_test,y_test), b
atch_size=50,callbacks=callbacks)

# Model 9: Remove Dropout, Adam(learning_rate=.001) optimizer,  binary_c
rossentropy loss

# LL Model - MM remove Dropout
seed = 42
first_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.1, seed=seed
)
hidden_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.05, seed=seed
)
output_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.001, seed=seed
)
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(28,)))
model.add(layers.Dense(200,activation='tanh'))
model.add(layers.Dense(100,activation='tanh'))
model.add(layers.Dense(100,activation='tanh')),
model.add(layers.Dense(1, activation='sigmoid'))

# Adam+binary_crossentropy variation
auc_score = tf.keras.metrics.AUC()
model.compile(optimizer=optimizers.Adam(learning_rate=.001),
              loss='binary_crossentropy',
              metrics=['accuracy',auc_score])

# LL Fit
model.fit(x_train,y_train, epochs=20, validation_data=(x_test,y_test), b
atch_size=50,callbacks=callbacks)

# Model 10 - Nodes = 600, 300, 150 instead of 200, 100, 100, 4 Layers, A
dam + binary_ce

# LL Model - MM remove Dropout, Changed Neurons from 300,100,100 to 600,
300,150
seed = 42
first_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.1, seed=seed
```

```
)
hidden_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.05, seed=seed
)
output_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.001, seed=seed
)
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(28,)))
model.add(layers.Dense(600,activation='tanh'))
model.add(layers.Dense(300,activation='tanh'))
model.add(layers.Dense(150,activation='tanh')),
model.add(layers.Dense(1, activation='sigmoid'))

# Adam+binary_crossentropy variation
auc_score = tf.keras.metrics.AUC()
model.compile(optimizer=optimizers.Adam(learning_rate=.001),
              loss='binary_crossentropy',
              metrics=['accuracy',auc_score])

# LL Fit
model.fit(x_train,y_train, epochs=10, validation_data=(x_test,y_test), b
atch_size=50,callbacks=callbacks)

# Visualize
train_loss = model.history.history['loss']
val_loss = model.history.history['val_loss']
plt.plot(train_loss,label='train_loss')
plt.plot(val_loss,label='valid_loss')
plt.title("Higgs Model with Adam Opt and Binary Cross-Entropy Loss")
legend = plt.legend(loc='upper center', shadow=True, fontsize='x-large')
plt.show();

# Model 11: Vary Activation Function - from tanh to swish

# LL Model - MM remove Dropout, Changed Neurons from 300,100,100 to 600,
300,150
seed = 42
first_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.1, seed=seed
)
hidden_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.05, seed=seed
)
output_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.001, seed=seed
)
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(28,)))
model.add(layers.Dense(600,activation='swish'))
model.add(layers.Dense(300,activation='swish'))
model.add(layers.Dense(150,activation='swish')),
model.add(layers.Dense(1, activation='sigmoid'))

# Adam+binary_crossentropy variation
auc_score = tf.keras.metrics.AUC()
model.compile(optimizer=optimizers.Adam(learning_rate=.001),
```

```
              loss='binary_crossentropy',
              metrics=['accuracy',auc_score])

# LL Fit
model.fit(x_train,y_train, epochs=10, validation_data=(x_test,y_test), b
atch_size=50,callbacks=callbacks)

# Visualize
train_loss = model.history.history['loss']
val_loss = model.history.history['val_loss']
plt.plot(train_loss,label='train_loss')
plt.plot(val_loss,label='valid_loss')
plt.title("Higgs Model with Adam Opt and Binary Cross-Entropy Loss")
legend = plt.legend(loc='upper center', shadow=True, fontsize='x-large')
plt.show();

# Model 12:  Base Model with 200, 100, 100 neurons.  Change tanh to swis
h

# MM Model
seed = 42
first_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.1, seed=seed
)
hidden_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.05, seed=seed
)
output_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.001, seed=seed
)
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(28,)))
model.add(layers.Dense(200,activation='swish'))
model.add(layers.Dense(100,activation='swish'))
model.add(layers.Dense(100,activation='swish')),
model.add(layers.Dense(1, activation='sigmoid'))

# Adam+binary_crossentropy variation
auc_score = tf.keras.metrics.AUC()
model.compile(optimizer=optimizers.Adam(learning_rate=.001),
              loss='binary_crossentropy',
              metrics=['accuracy',auc_score])

# LL Fit
callbacks = [EarlyStopping( patience=1)]
model.fit(x_train,y_train, epochs=20, validation_data=(x_test,y_test), b
atch_size=50,callbacks=callbacks)

# Model 13: Change Nodes to 600,300,150 and change swish to softplus

# LL Model - MM remove Dropout, Changed Neurons from 300,100,100 to 600,
300,150
seed = 42
first_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.1, seed=seed
)
hidden_layer_init = initializers.RandomNormal(
```

```
    mean=0.0, stddev=0.05, seed=seed
)
output_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.001, seed=seed
)
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(28,)))
model.add(layers.Dense(600,activation='softplus'))
model.add(layers.Dense(300,activation='softplus'))
model.add(layers.Dense(150,activation='softplus')),
model.add(layers.Dense(1, activation='sigmoid'))

# Adam+binary_crossentropy variation
auc_score = tf.keras.metrics.AUC()
model.compile(optimizer=optimizers.Adam(learning_rate=.001),
              loss='binary_crossentropy',
              metrics=['accuracy',auc_score])

# LL Fit
callbacks = [EarlyStopping( patience=1)]
model.fit(x_train,y_train, epochs=20, validation_data=(x_test,y_test), b
atch_size=50,callbacks=callbacks)

# Model 14: Full Dataset, Swish Activation, Adam Optimizer, binary_cross
entropy

# All Data Model
seed = 42
first_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.1, seed=seed
)
hidden_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.05, seed=seed
)
output_layer_init = initializers.RandomNormal(
    mean=0.0, stddev=0.001, seed=seed
)
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(28,)))
model.add(layers.Dense(200,activation='swish'))
model.add(layers.Dense(100,activation='swish'))
model.add(layers.Dense(100,activation='swish')),
model.add(layers.Dense(1, activation='sigmoid'))

# Adam+binary_crossentropy variation
auc_score = tf.keras.metrics.AUC()
model.compile(optimizer=optimizers.Adam(learning_rate=.001),
              loss='binary_crossentropy',
              metrics=['accuracy',auc_score])

# LL Fit
callbacks = [EarlyStopping( patience=1)]
model.fit(x_train_all,y_train_all, epochs=100, validation_data=(x_test_a
ll,y_test_all), batch_size=50,callbacks=callbacks)

# Visualize
train_loss = model.history.history['loss']
```

```
val_loss = model.history.history['val_loss']
plt.plot(train_loss,label='train_loss')
plt.plot(val_loss,label='valid_loss')
plt.title("Full Higgs Dataset with Adam Opt, Binary Cross-Entropy Loss a
nd Swish Activation")
legend = plt.legend(loc='upper center', shadow=True, fontsize='x-large')
plt.show();
```