```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from matplotlib import pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
```

```
data = pd.read_csv('magic_gamma_telescope/magic04.csv')
```

```
columns = ['fLength', 'fWidth', 'fSize', 'fConc', 'fConc1', 'fAsym', 'fM3Long', 'fM3Trans', 'fAlpha', 'fDist', 'class']
data.columns = columns
```

```
data
```

	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	fAlpha	fDist	class
0	31.6036	11.7235	2.5185	0.5303	0.3773	26.2722	23.8238	-9.9574	6.3609	205.2610	g
1	162.0520	136.0310	4.0612	0.0374	0.0187	116.7410	-64.8580	-45.2160	76.9600	256.7880	g
2	23.8172	9.5728	2.3385	0.6147	0.3922	27.2107	-6.4633	-7.1513	10.4490	116.7370	g
3	75.1362	30.9205	3.1611	0.3168	0.1832	-5.5277	28.5525	21.8393	4.6480	356.4620	g
4	51.6240	21.1502	2.9085	0.2420	0.1340	50.8761	43.1887	9.8145	3.6130	238.0980	g
...	...	...	...	...	...	...	...	...	...	...	...
19014	21.3846	10.9170	2.6161	0.5857	0.3934	15.2618	11.5245	2.8766	2.4229	106.8258	h
19015	28.9452	6.7020	2.2672	0.5351	0.2784	37.0816	13.1853	-2.9632	86.7975	247.4560	h
19016	75.4455	47.5305	3.4483	0.1417	0.0549	-9.3561	41.0562	-9.4662	30.2987	256.5166	h
19017	120.5135	76.9018	3.9939	0.0944	0.0683	5.8043	-93.5224	-63.8389	84.6874	408.3166	h
19018	187.1814	53.0014	3.2093	0.2876	0.1539	-167.3125	-168.4558	31.4755	52.7310	272.3174	h

19019 rows × 11 columns

Preprocessing

```
X = data[columns[:-1]]
y = data['class']

features_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler()),
])

labelencoder = LabelEncoder()

X = features_pipeline.fit_transform(X)
y = labelencoder.fit_transform(y)

y = y.astype(float)

X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.2)

X_train, X_val = train_test_split(X_train_full, test_size=0.2)
y_train, y_val = train_test_split(y_train_full, test_size=0.2)
```

Converting to Tensorflow dataset

```
X_train = tf.convert_to_tensor(X_train)
X_val = tf.convert_to_tensor(X_val)
X_test = tf.convert_to_tensor(X_test)
y_train = tf.convert_to_tensor(y_train)
```

```
y_val = tf.convert_to_tensor(y_val)
y_test = tf.convert_to_tensor(y_test)
```

Building the model

```
def build_model(n_hidden=1, n_neurons=30, input_shape=[10,], optimizer="adam"):
    model = keras.models.Sequential()
    model.add(keras.layers.Flatten(input_shape=input_shape))
    for _ in range(n_hidden):
        model.add(keras.layers.Dense(n_neurons, activation="relu"))
    model.add(keras.layers.Dense(10, activation="softmax"))
    model.compile(optimizer=optimizer,
        loss="sparse_categorical_crossentropy",
        metrics=["accuracy"])
    return model
```

Training the model with hyperparams

```
from sklearn.model_selection import KFold

param_grid = {
    'n_hidden': [1,2],
    'n_neurons': [10, 50, 100],
    'optimizer': ['adam', 'sgd']
}

kfold = KFold(n_splits=5, shuffle=True)

n_epochs = 5

results = []
for n_hidden in param_grid['n_hidden']:
    for n_neurons in param_grid['n_neurons']:
        for optimizer in param_grid['optimizer']:
            model = build_model(n_hidden=n_hidden, n_neurons=n_neurons, optimizer=optimizer)
            fold_scores = []
            for train_idx, val_idx in kfold.split(X):
                X_train, X_val = X[train_idx], X[val_idx]
                y_train, y_val = y[train_idx], y[val_idx]
                model.fit(X_train, y_train, epochs=n_epochs, validation_data=(X_val, y_val))
                _, accuracy = model.evaluate(X_val, y_val, verbose=0)
                fold_scores.append(accuracy)
            avg_score = sum(fold_scores) / len(fold_scores)
            results.append({'n_hidden': n_hidden, 'n_neurons': n_neurons, 'optimizer' : optimizer, 'score': avg_score, 'model': model})
```

```
Epoch 1/5
476/476 [==============================] - 1s 2ms/step - loss: 0.9987 - accuracy: 0.6891 - val_loss: 0.5299 - val_accuracy: 0.7852
Epoch 2/5
476/476 [==============================] - 1s 1ms/step - loss: 0.4620 - accuracy: 0.8021 - val_loss: 0.4453 - val_accuracy: 0.8036
Epoch 3/5
476/476 [==============================] - 1s 1ms/step - loss: 0.4191 - accuracy: 0.8143 - val_loss: 0.4233 - val_accuracy: 0.8144
Epoch 4/5
476/476 [==============================] - 1s 1ms/step - loss: 0.4022 - accuracy: 0.8231 - val_loss: 0.4102 - val_accuracy: 0.8223
Epoch 5/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3906 - accuracy: 0.8314 - val_loss: 0.4014 - val_accuracy: 0.8270
Epoch 1/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3879 - accuracy: 0.8365 - val_loss: 0.3685 - val_accuracy: 0.8454
Epoch 2/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3801 - accuracy: 0.8421 - val_loss: 0.3626 - val_accuracy: 0.8525
Epoch 3/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3738 - accuracy: 0.8463 - val_loss: 0.3573 - val_accuracy: 0.8541
Epoch 4/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3678 - accuracy: 0.8506 - val_loss: 0.3538 - val_accuracy: 0.8515
Epoch 5/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3636 - accuracy: 0.8538 - val_loss: 0.3512 - val_accuracy: 0.8599
Epoch 1/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3580 - accuracy: 0.8557 - val_loss: 0.3496 - val_accuracy: 0.8583
Epoch 2/5
476/476 [==============================] - 1s 2ms/step - loss: 0.3545 - accuracy: 0.8563 - val_loss: 0.3474 - val_accuracy: 0.8583
Epoch 3/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3516 - accuracy: 0.8582 - val_loss: 0.3455 - val_accuracy: 0.8567
Epoch 4/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3489 - accuracy: 0.8595 - val_loss: 0.3436 - val_accuracy: 0.8609
```

```
Epoch 5/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3464 - accuracy: 0.8602 - val_loss: 0.3420 - val_accuracy: 0.8628
Epoch 1/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3414 - accuracy: 0.8613 - val_loss: 0.3513 - val_accuracy: 0.8575
Epoch 2/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3386 - accuracy: 0.8621 - val_loss: 0.3504 - val_accuracy: 0.8559
Epoch 3/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3361 - accuracy: 0.8627 - val_loss: 0.3489 - val_accuracy: 0.8573
Epoch 4/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3343 - accuracy: 0.8639 - val_loss: 0.3484 - val_accuracy: 0.8596
Epoch 5/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3329 - accuracy: 0.8648 - val_loss: 0.3479 - val_accuracy: 0.8559
Epoch 1/5
476/476 [==============================] - 1s 2ms/step - loss: 0.3371 - accuracy: 0.8607 - val_loss: 0.3218 - val_accuracy: 0.8733
Epoch 2/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3354 - accuracy: 0.8608 - val_loss: 0.3213 - val_accuracy: 0.8712
Epoch 3/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3344 - accuracy: 0.8613 - val_loss: 0.3189 - val_accuracy: 0.8722
Epoch 4/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3329 - accuracy: 0.8613 - val_loss: 0.3187 - val_accuracy: 0.8719
Epoch 5/5
476/476 [==============================] - 1s 1ms/step - loss: 0.3325 - accuracy: 0.8628 - val_loss: 0.3189 - val_accuracy: 0.8719
Epoch 1/5
476/476 [==============================] - 1s 2ms/step - loss: 1.1666 - accuracy: 0.6528 - val_loss: 0.6577 - val_accuracy: 0.7392
Epoch 2/5
476/476 [==============================] - 1s 1ms/step - loss: 0.5356 - accuracy: 0.7776 - val_loss: 0.5011 - val_accuracy: 0.7776
Epoch 3/5
476/476 [==============================] - 1s 1ms/step - loss: 0.4512 - accuracy: 0.8060 - val_loss: 0.4625 - val_accuracy: 0.7931
Epoch 4/5
```

Evaluation of the model

```
best_result = max(results, key=lambda x: x['score'])
best_params = {'n_hidden': best_result['n_hidden'], 'n_neurons': best_result['n_neurons'], 'optimizer': best_result['optimizer']}
best_score = best_result['score']
best_model = best_result['model']

print("Best Parameters: ", best_params)
print("Best Score: ", best_score * 100)
```

```
Best Parameters:  {'n_hidden': 2, 'n_neurons': 100, 'optimizer': 'adam'}
Best Score:  87.38631963729858
```

```
best_model.summary()
```

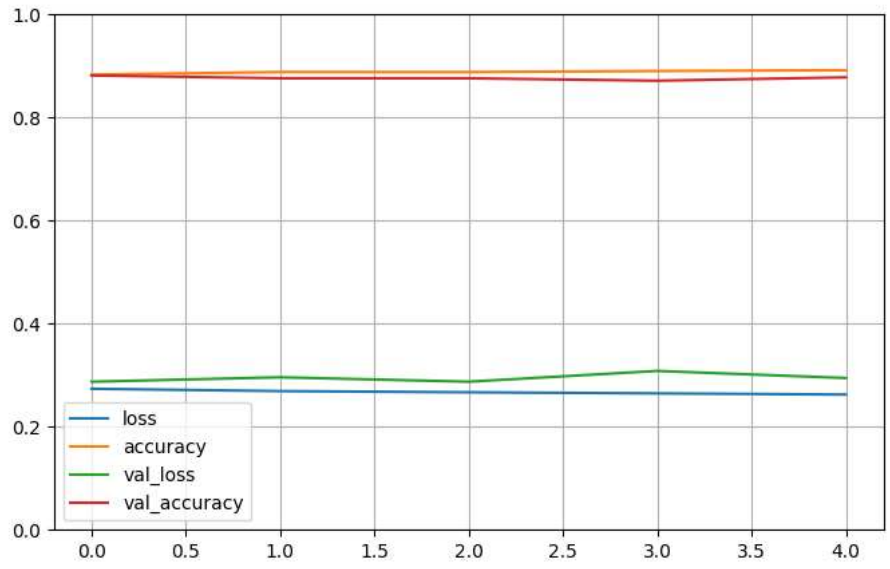
```
Model: "sequential_61"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_61 (Flatten)        (None, 10)                0

 dense_153 (Dense)           (None, 100)               1100

 dense_154 (Dense)           (None, 100)               10100

 dense_155 (Dense)           (None, 10)                1010

=================================================================
Total params: 12210 (47.70 KB)
Trainable params: 12210 (47.70 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
history = best_model.fit(X_train, y_train, epochs=n_epochs, validation_data=(X_val, y_val))
```

```
Epoch 1/5
476/476 [==============================] - 1s 2ms/step - loss: 0.2738 - accuracy: 0.8832 - val_loss: 0.2874 - val_accuracy: 0.8817
Epoch 2/5
476/476 [==============================] - 1s 1ms/step - loss: 0.2692 - accuracy: 0.8883 - val_loss: 0.2960 - val_accuracy: 0.8762
Epoch 3/5
476/476 [==============================] - 1s 2ms/step - loss: 0.2669 - accuracy: 0.8882 - val_loss: 0.2875 - val_accuracy: 0.8762
Epoch 4/5
476/476 [==============================] - 1s 1ms/step - loss: 0.2648 - accuracy: 0.8902 - val_loss: 0.3083 - val_accuracy: 0.8714
Epoch 5/5
476/476 [==============================] - 1s 1ms/step - loss: 0.2625 - accuracy: 0.8918 - val_loss: 0.2946 - val_accuracy: 0.8780
```

```
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
```

```
(0.0, 1.0)
```




```
probs = best_model.predict(X_test)
```

```
119/119 [==============================] - 0s 700us/step
```

```
predications = labelencoder.inverse_transform(np.argmax(probs, axis=1))
print(np.argmax(probs, axis=1))
```

```
[1 0 1 ... 0 0 1]
```

```
predictions_X = pd.DataFrame(X_test)
predictions_y = pd.DataFrame(predications)
predictions = pd.concat([predictions_X, predictions_y], axis=1)
predictions.columns = ['fLength', 'fWidth', 'fSize', 'fConc', 'fConc1', 'fAsym', 'fM3Long', 'fM3Trans', 'fAlpha', 'fDist', 'class']
predictions
```

	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	
0	0.708269	0.647182	1.456577	-0.548792	-0.456589	0.232240	0.938598	0.844809	-
1	-0.765708	-0.879450	-1.228799	2.283061	3.238979	0.353241	0.158795	-0.305313	
2	-0.965064	-0.548321	-1.218431	1.617901	1.193024	0.357808	-0.410851	-0.396737	
3	-0.694786	-0.687261	-1.029475	1.282039	1.606559	0.532790	-0.333495	-0.355373	-
4	-0.534338	-0.311326	-0.477630	0.588981	0.277276	0.481808	0.280226	-0.578694	-
...	...	...	...	...	...	...	...	...	
3799	0.204253	0.705620	0.138328	-1.263730	-1.250177	0.859752	0.723738	-1.015658	-
3800	0.289610	0.839720	0.804435	-1.210671	-1.205838	0.145943	-1.101123	1.755559	
3801	-0.441326	-0.809102	-0.544706	0.836228	0.714338	0.779501	0.232750	0.169495	-
3802	-0.191104	-0.684623	0.012428	0.220846	0.080915	1.010432	0.515834	-0.256205	-
3803	-0.341542	-0.100158	0.060672	-0.747356	-0.769681	-0.161135	-0.111469	0.445584	-

3804 rows × 11 columns