Case Study in Celestial Body Classification

CSCI S-89: Introduction to Deep Learning, Harvard Summer School
Sathvika Iyengar

ABSTRACT

With the growing amount of observational data from observatories and space telescopes, more automated methods for classifying celestial bodies are essential for scientific discovery. This case study presents an investigation into the development and evaluation of a deep learning-based classification system for identifying various types of celestial bodies, including stars, galaxies, and quasars.

The study begins by collecting data from the Sloan Digital Sky Survey (SDSS), one of largest public astronomical datasets. Through SQL queries submitted to the SDSS database server, a dataset of 250,000 celestial object observations was used for training, validating, and testing the neural network built. Each observation had 17 feature columns and 1 class column indicating the type of celestial body (star, galaxy or quasar).

Next, the dataset was prepared for processing by dropping irrelevant columns, such as identifiers. In addition, one-hot encoding was applied to the class column, where dummy variables are created for each categorical value. The dataset was scaled and split into training (60% of the dataset), validation (20%), testing (20%) sets, to assess the model's performance on unseen data.

Then, a feedforward neural network model was constructed with two hidden layers, along with batch normalization and dropout layers to prevent overfitting. The dropout rates for the layers were determined through finding the optimal validation accuracy from sample dropout rate values. With the model trained with the training set, the accuracy of the model was evaluated on the unseen testing set. This resulted in a test loss of 4.26% and test accuracy of 98.85%, indicating the high accuracy performance of the model.

The model was then used to predict the class labels for the unseen validation dataset, for which a Confusion Matrix displayed significant results for the class types predicted correctly. In addition, using Principal Component Analysis (PCA) shed light on the reasoning behind the model's high prediction accuracy. The first two principal components (PCs) explained 98.15% of the variance in the data, which were the most informative features for the neural network to learn the patterns of classification in the data. It was seen that original plate and run features contributed the most to each principal component, respectively.

Overall, the deep learning-based classification system proved to be effective in identifying celestial bodies with high accuracy. With an automized classification process, this approach can significantly aid in the handling and processing of observational data.

Link to Youtube Video Presentation: https://youtu.be/pHDDyVNmpSw

I. INTRODUCTION

As the universe continues to expand, our understanding of celestial bodies undergoes constant evolution. The criteria used to categorize these entities, whether they are planets, stars, or galaxies, experience frequent revisions with each significant astronomical discovery. Space exploration and cutting-edge technologies have expedited the discovery of new celestial bodies, uncovering new knowledge almost every day.

This research project delves into three distinctive celestial bodies: galaxies, stars, and quasars. Galaxies are vast collections of stars and interstellar matter that offer insights into cosmic evolution. Stars are luminous celestial bodies that shape the structure of the universe. Quasars are luminous objects that are powered by supermassive blackholes.

The primary objective of this study revolves around addressing two pivotal questions:

1. What specific criteria determine the classification of a celestial body as a galaxy, star, or quasar?
2. Can these criteria be utilized to accurately predict and classify celestial bodies that have yet to be definitively categorized?

By delving into these inquiries, this study aspires to illuminate the classification of celestial bodies and make a valuable contribution to the field of astronomy, providing insights that can pave the way for future discoveries.

II. DATASET DESCRIPTION

This study relies on data sourced from the Sloan Digital Sky Survey (SDSS), utilizing a wide-angle optical telescope at the Apache Point Observatory to meticulously map the universe. The dataset, obtained from SkyServer SQL Search and downloaded as a csv with the following command below:

```
-- This query does a table JOIN between the imaging (PhotoObj) and spectra
-- (SpecObj) tables and includes the necessary columns in the SELECT to upload
-- the results to the SAS (Science Archive Server) for FITS file retrieval.
SELECT TOP 250000
p.objid,p.ra,p.dec,p.u,p.g,p.r,p.i,p.z,
p.run, p.rerun, p.camcol, p.field,
s.specobjid, s.class, s.z as redshift,
s.plate, s.mjd, s.fiberid
FROM PhotoObj AS p
JOIN SpecObj AS s ON s.bestobjid = p.objid
WHERE
p.u BETWEEN 0 AND 19.6
AND g BETWEEN 0 AND 20
```

The dataset comprises information on 250,000 celestial objects, each with 17 distinctive features and a corresponding class variable. A dataset size of 250,000 was chosen as it provided a large and diverse enough dataset to train a model for classification. As seen below, the dataset has a

significant amount of observations for galaxies, stars, and quasars, which would allow the neural network to be exposed to thousands of different celestial objects.

```
df_celestial["class"].value_counts()
```

```
GALAXY    127117
STAR       96116
QSO        26767
Name: class, dtype: int64
```

The features provided for each observation are as follows:

• objid = Object Identifier
• u = Ultraviolet filter in the photometric system
• ra = Right Ascension angle (at J2000 epoch)
• dec = Declination angle (at J2000 epoch)
• g = Green filter in the photometric system
• r = Red filter in the photometric system
• i = Near-Infrared filter in the photometric system
• z = Infrared filter in the photometric system
• run = Run Number used to identify the specific scan
• rerun = Rerun Number specifying image processing details
• camcol = Camera column identifying the scanline within the run
• field = Field number used for identification
• specobjid = Object Identifier for optical spectroscopic objects
• class = Object class (galaxy, star, or quasar)
• redshift = Redshift value based on the increase in wavelength
• plate = Plate ID, identifying each plate in SDSS
• mjd = Modified Julian Date indicating when SDSS data was taken
• fiberid = Fiber ID pointing the light at the focal plane in each observation

This hardware used for this study was an Apple M1 MacBook Pro and Jupyter Notebook.

III. DATASET PREPROCESSING

To begin, the dataset is converted from a csv to a dataframe, using the Panda's function read_csv.

```
# reading downloaded csv from SQL search
df_celestial = pd.read_csv('Skyserver250k.csv')
df_celestial
```

	objid	ra	dec	u	g	r	i	z	run	rerun	camcol	field	specobjid	
0	1237661976015274033	196.362072	7.667016	19.32757	19.20759	19.16249	19.07652	18.86196	3842	301	4	102	2020027785916999680	
1	1237661362373066810	206.614664	45.924279	18.95918	17.09173	16.25019	15.83413	15.55686	3699	301	5	121	1649585252231833600	GA
2	1237661360767238272	220.294728	40.894575	17.75587	16.54700	16.67694	16.77780	16.88097	3699	301	2	194	3812387877359296512	
3	1237665440983416884	206.315349	27.438152	19.29195	19.12720	19.03992	18.76714	18.73874	4649	301	2	152	6762291282364878848	
4	1237665531717812262	228.092653	20.807371	19.19731	18.26143	17.89954	17.76130	17.68726	4670	301	3	201	4454292673071960064	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
249995	1237661360767565997	221.101714	40.641045	19.30748	18.22145	17.61426	17.32240	17.02841	3699	301	2	199	1572955889466370048	GA
249996	1237667783903084693	171.645089	22.797546	19.19911	17.79553	17.03988	16.63705	16.31786	5194	301	6	381	2814852916416899072	GA
249997	1237648704591233226	215.751103	0.044486	18.88386	17.51738	16.89393	16.39914	16.07888	752	301	4	482	342430828837496832	GA
249998	1237660634386923630	163.501339	44.470389	18.49867	16.73666	15.88036	15.48524	15.14631	3530	301	1	286	1614541342266910720	GA
249999	1237668271376236954	236.495641	12.142432	19.39894	18.40550	18.06466	17.93771	17.87524	5308	301	2	295	5517031135849500672	

250000 rows × 18 columns

Then, the dataframe is checked to see if there were any null values that would need to be replaced NaN.

```
# checking to see if there are any null values
df_celestial.isnull().sum()
```

```
objid        0
ra           0
dec          0
u            0
g            0
r            0
i            0
z            0
run          0
rerun        0
camcol       0
field        0
specobjid    0
class        0
redshift     0
plate        0
mjd          0
fiberid      0
dtype: int64
```

Since neural networks will be used for multi-class classification, one-hot encoding is used to create dummy variable for each categorical value. This allows the output variable to take on the form of a matrix, where 0's and 1's represent whether each observation has a particular class value. In this case, the three class values are GALAXY, STAR, and QSO. With the addition of dummy variables, the 'class' column can be dropped from the dataframe. The dataset also has two unique object identifier features, objid and specobjid, which were dropped from the dataframe.

```
# using one hot encoding to generating 0's and 1's for each body type
df_celestial = pd.concat([df_celestial, pd.get_dummies(df_celestial["class"], prefix="class type")], axis=1)
# dropping class feature for classification
df_celestial.drop(columns=["class"], inplace=True)
df_celestial
```

g	r	i	z	run	rerun	camcol	field	specobjid	redshift	plate	mjd	fiberid	class type_GALAXY	class type_QSO	class type_STAR
9.20759	19.16249	19.07652	18.86196	3842	301	4	102	2020027785916999680	1.984419	1794	54504	594	0	1	0
7.09173	16.25019	15.83413	15.55686	3699	301	5	121	1649585252231833600	0.064456	1465	53082	516	1	0	0
6.54700	16.67694	16.77780	16.88097	3699	301	2	194	3812387877359296512	-0.000509	3386	54952	330	0	0	1
9.12720	19.03992	18.76714	18.73874	4649	301	2	152	6762291282364878848	1.882893	6006	56105	496	0	1	0
8.26143	17.89954	17.76130	17.68726	4670	301	3	201	4454292673071960064	-0.000295	3956	55656	846	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8.22145	17.61426	17.32240	17.02841	3699	301	2	199	1572955889466370048	0.158805	1397	53119	268	1	0	0
7.79553	17.03988	16.63705	16.31786	5194	301	6	381	2814852916416899072	0.034430	2500	54178	375	1	0	0
7.51738	16.89393	16.39914	16.07888	752	301	4	482	342430828837496832	0.078681	304	51609	572	1	0	0
6.73666	15.88036	15.48524	15.14631	3530	301	1	286	1614541342266910720	0.048519	1434	53053	3	1	0	0
8.40550	18.06466	17.93771	17.87524	5308	301	2	295	5517031135849500672	0.000864	4900	55739	442	0	0	1

The dataframe was then split into input values X, which consisted of all rows from 15 features, as well as the output value y, which consisted of all rows from the three class values, "class type_GALAXY", "class type_QSO", and "class type_STAR."

Using the train_test_split function, the input and output values were then split into the training set (60% of the data), validation set (20%), and test set (20%). This ratio was chosen as to allow the model to have enough data to train, to evaluate the model using test data, and then to make predictions using the validation data. All three training, validation, and testing input arrays were

scaled using the StandardScaler(), which normalizes the data to be on the same scale for each feature.

```
# input values, dropping the two object identifiers
X = df_celestial.drop(columns=["objid", "specobjid"])
X = X.iloc[:,0:15]
# output values
y = df_celestial[["class type_GALAXY", "class type_QSO","class type_STAR"]]

# splitting data into training set (60%), validation set (20%), and test set (20%)
X_train, X_tmp, y_train, y_tmp = train_test_split(X, y, test_size=0.4, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_tmp, y_tmp, test_size=0.5, random_state=42)

# scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

IV. CREATING THE NEURAL NETWORK MODEL

The model is a feedforward neural network, where the input data flows from the input layer through two hidden layers to the output layer.

```
model = Sequential()
model.add(Dense(128, input_dim=X_train_scaled.shape[1], activation="relu"))
model.add(BatchNormalization())  # batch normalization layer
model.add(Dropout(0.1))
model.add(Dense(64, activation="relu"))
model.add(BatchNormalization())  # batch normalization layer
model.add(Dropout(0.1))
model.add(Dense(3, activation="softmax"))
```

The first hidden layer contains 128 neurons with the Rectified Linear Unit (ReLU) activation function, followed by batch normalization and a dropout layer. With 16 features in the input layer and the size of the dataset, 128 neurons were used for the first Dense layer in order to make sure that network could learn more intricate patterns. The second hidden layer consists of 64 neurons with ReLU activation, followed by another batch normalization and dropout layer. The output layer has three neurons, representing the three class types, and uses the classification activation function, softmax, to provide probabilities for each class.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_18 (Dense)            (None, 128)               2048

 batch_normalization_2 (Bat  (None, 128)               512
 chNormalization)

 dropout_12 (Dropout)        (None, 128)               0

 dense_19 (Dense)            (None, 64)                8256

 batch_normalization_3 (Bat  (None, 64)                256
 chNormalization)

 dropout_13 (Dropout)        (None, 64)                0

 dense_20 (Dense)            (None, 3)                 195

=================================================================
Total params: 11267 (44.01 KB)
Trainable params: 10883 (42.51 KB)
Non-trainable params: 384 (1.50 KB)
_________________________________________________________________
```

Batch normalization and dropouts were two regularization techniques used to prevent the model from overfitting. Batch normalization normalizes the activations of each neural network, with stabilizes and speeds up training. Dropouts randomly deactivate neurons during training to prevent neurons from relying on specific inputs. The dropout rate was determined by observing the highest validation accuracy for sample dropout rates from 0.1 to 0.5. The dropout rate of 0.1 provided the best validation accuracy of 98.91%.

```
# dropout rates to test
dropout_rates = [0.1, 0.2, 0.3, 0.4, 0.5]

# dict to store val accuracies
dropout_val_accuracies = {}

# iterating over each dropout rate
for dropout_rate in dropout_rates:
    # creating model with dropout rate
    model = Sequential()
    model.add(Dense(128, input_dim=X_train_scaled.shape[1], activation="relu"))
    model.add(Dropout(dropout_rate))
    model.add(Dense(64, activation="relu"))
    model.add(Dropout(dropout_rate))
    model.add(Dense(3, activation="softmax"))

    # compile
    model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

    # training the model with the training set
    model.fit(X_train_scaled, y_train, epochs=100, batch_size=32, verbose=0)

    # evaluate the model on the validation set
    _, val_accuracy = model.evaluate(X_val_scaled, y_val, verbose=0)

    # store the validation accuracy for each dropout rate
    dropout_val_accuracies[dropout_rate] = val_accuracy

# finding the dropout rate with the max validation accuracy
best_dropout_rate = max(dropout_val_accuracies, key=dropout_val_accuracies.get)

print("Validation Accuracies for Different Dropout Rates:")
for dropout_rate, val_accuracy in dropout_val_accuracies.items():
    print(f"Dropout Rate: {dropout_rate}, Validation Accuracy: {val_accuracy:.4f}")

print(f"\nBest Dropout Rate: {best_dropout_rate}")
```

```
Validation Accuracies for Different Dropout Rates:
Dropout Rate: 0.1, Validation Accuracy: 0.9891
Dropout Rate: 0.2, Validation Accuracy: 0.9874
Dropout Rate: 0.3, Validation Accuracy: 0.9874
Dropout Rate: 0.4, Validation Accuracy: 0.9881
Dropout Rate: 0.5, Validation Accuracy: 0.9831

Best Dropout Rate: 0.1
```

The model is compiled with loss function typically used for classification, categorical cross-entropy loss and the Adam optimizer, and its performance is evaluated using accuracy during training. The model is trained using the training data (X_train_scaled and y_train). The training process consists of 100 epochs, where each epoch represents one full pass through the entire training dataset. During each epoch, the model updates its weights to minimize categorical cross-entropy using the Adam optimizer. The batch_size parameter 32 determines the number of samples to be used in each update of the model's weights.

V. TESTING + RESULTS

The trained model's performance is then evaluated on test set (X_test_scaled and y_test), which the model has not seen during the training. The test loss was observed to be 4.26%, whereas the test accuracy was about 98.95%. This indicated that the model performed well in correctly classifying samples from the test set.

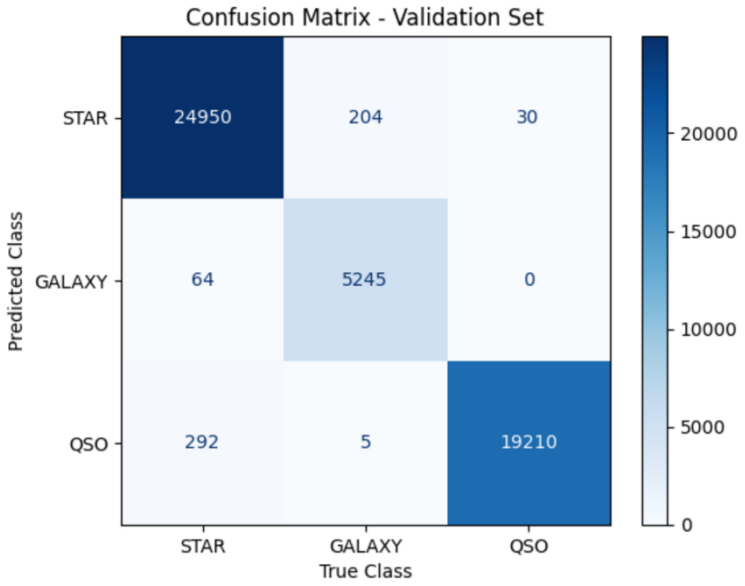
```
# compiling
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

# training the model with the training set
model.fit(X_train_scaled, y_train, epochs=100, batch_size=32)

# evaluating model on test dataset
loss, accuracy = model.evaluate(X_test_scaled, y_test)
print(f"Test loss: {loss:.4f}, Test accuracy: {accuracy:.4f}")
```

```
4688/4688 [==============================] - 4s 806us/step - loss: 0.0444 - accuracy: 0.9889
Epoch 93/100
4688/4688 [==============================] - 4s 779us/step - loss: 0.0442 - accuracy: 0.9886
Epoch 94/100
4688/4688 [==============================] - 6s 1ms/step - loss: 0.0436 - accuracy: 0.9889
Epoch 95/100
4688/4688 [==============================] - 5s 1ms/step - loss: 0.0449 - accuracy: 0.9887
Epoch 96/100
4688/4688 [==============================] - 4s 896us/step - loss: 0.0439 - accuracy: 0.9889
Epoch 97/100
4688/4688 [==============================] - 4s 795us/step - loss: 0.0445 - accuracy: 0.9886
Epoch 98/100
4688/4688 [==============================] - 4s 834us/step - loss: 0.0442 - accuracy: 0.9888
Epoch 99/100
4688/4688 [==============================] - 4s 885us/step - loss: 0.0441 - accuracy: 0.9887
Epoch 100/100
4688/4688 [==============================] - 4s 770us/step - loss: 0.0454 - accuracy: 0.9882
1563/1563 [==============================] - 1s 449us/step - loss: 0.0426 - accuracy: 0.9895
Test loss: 0.0426, Test accuracy: 0.9895
```

Using the trained model, predictions for the class labels for the validation set are computed. A confusion matrix is created to display 9 different combinations of predicted and actual values. The diagonal of the 3x3 matrix corresponds to the true positives, where 24950 stars, 5245 galaxies, and 19210 quasars have been predicted correctly. It can be observed that 204 galaxies and 30 quasars have been predicted as stars, 64 stars have been predicted as galaxies, 292 stars and 5 galaxies have been predicted as quasars. Overall, the number of celestial objects correctly predicted is significantly higher than the number incorrectly predicted, proving the neural network is doing a rather good job at classifying.



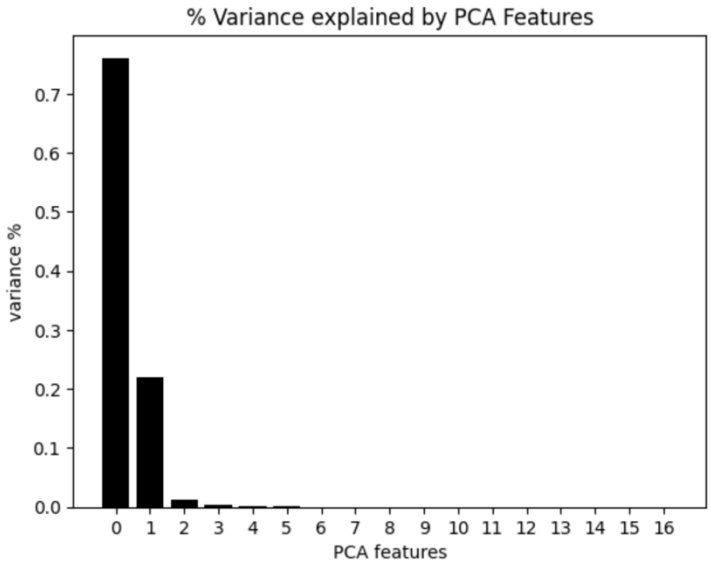
VI. ANALYSIS

Given the fact that the model has such a high classification accuracy, it can be inferred that there are certain features significantly contribute the model's classification ability. Principal Component Analysis (PCA) can help identify which features have the most effect for classification, through the directions of maximum variance in the data. PCA transforms the original data into a set of principal components (PCs), which are uncorrelated features, that capture the most variations in the data. Features that have a high variance contribute the most to the total variance explained by each principal component. These features have a larger spread of data and implies that the original features have a substantial level of correlation.

Applying PCA to the dataframe X that contains the original features, the plot below displays the percentage of variance in the data explained by each principal component. It can be observed in the plot that the first PC explains about 75% of variance in the data, with the second PC explaining about 20% of the variance. Using the cumulative sum of explained variance ratios, the first two PCs explain about 98.15% of the variance. This indicates that the first two PCs have the most effect on classification.

```
from sklearn.decomposition import PCA
# applying PCA on original dataframe
pca = PCA(whiten=True)
y = pca.fit_transform(X.values)

# plotting all PCA features
features = range(pca.n_components_)
plt.bar(features, pca.explained_variance_ratio_, color='black')
plt.xlabel('PCA features')
plt.ylabel('variance %')
plt.title('% Variance explained by PCA Features')
plt.xticks(features);
```




```
pca.explained_variance_ratio_.cumsum()
```

```
array([0.7611583 , 0.98153111, 0.99307609, 0.99730108, 0.99925855,
       0.99984371, 0.99992236, 0.99996177, 0.99999948, 0.99999973,
       0.99999996, 0.99999999, 1.        , 1.        , 1.        ,
       1.        , 1.        ])
```

Focusing on the first two PC's, the features that have the most significant influence on each PC can be found. The PCA features can be mapped to the original features, where the feature with the highest absolute value weight contributes the most to each PC.

```
# focusing on the first two PC's
pca = PCA(n_components=2, whiten=True)
y = pca.fit_transform(X.values)

# names of original features
feature_names = X.columns

# names of the first two PCA features
pca_feature_names = ['PCA_1', 'PCA_2']

# storing the mapping of PCA features to original features
pca_to_original_features_df = pd.DataFrame(pca.components_, columns=feature_names, index=pca_feature_names)
pca_to_original_features_df
```

	ra	dec	u	g	r	i	z	run	rerun	camcol	field	redshift	plate	mjd	fibe
PCA_1	-0.001680	-0.000047	-4.520028e-07	0.000044	0.000073	0.000091	0.000121	0.322864	0.0	-0.000007	-0.003483	0.000025	0.776598	0.540617	0.019
PCA_2	0.001434	0.001376	-3.925545e-06	0.000042	0.000070	0.000090	0.000096	-0.945022	-0.0	0.000050	0.011546	0.000021	0.294865	0.140522	0.010

For PCA_1, the plate feature (Plate ID, identifying each plate in SDSS) contributes the most. For PCA_2, the run feature (Run Number used to identify the specific scan) contributes the most. This shows that these two features are the most informative on the pattern of the data, which the model is able to effectively learn for classification.

```
# finding feature with the highest weight (absolute value) for each PC
highest_weight_features = {
    # finding the index of the max absolute weight
    pca_feature: pca_to_original_features_df.loc[pca_feature].abs().idxmax()
    for pca_feature in pca_feature_names
}
# for the first two PC's
for pca_feature, feature_name in highest_weight_features.items():
    print(f"For {pca_feature}, the feature with the highest weight is: {feature_name}")
```

```
For PCA_1, the feature with the highest weight is: plate
For PCA_2, the feature with the highest weight is: run
```

VII. CONCLUSION + LESSONS LEARNED

In this case study, a deep learning-based classification system was successfully developed and evaluated for identifying different types of celestial bodies, including stars, galaxies, and quasars. The system was trained on a dataset of 250,000 celestial object observations obtained from the Sloan Digital Sky Survey (SDSS) database. Through a feedforward neural network with batch normalization and dropout layers, an impressive test accuracy of 98.85% was achieved, demonstrating the model's ability to accurately classify celestial bodies.

Using the unseen validation dataset, the Confusion Matrix provided valuable insights into the model's performance, revealing significant results for the correct classification of class types. Through PCA, it was determined that the first two principal components explained 98.15% of the variance in the data, where the plate and run features are the most informative on the pattern of the data. Since the two PC's are the most informative features, the neural network was able to learn the patterns in classification efficiently.

One of the lessons learned throughout this study was that testing the prediction accuracy of the model using unseen dataset allowed for an evaluation from an unbiased model and prevented data leakage. Another lesson learned was that it was important to use batch normalization and dropout layers to prevent overfitting and enhance generalization of the model. This allowed for the high accuracy of the model to not be necessarily from overfitting, but from sophistication of the data that allowed the model to predict class types efficiently. Using PCA also taught that original dataset had two principal components that contributed the most to the variance to the data. This explained why the model has such a high prediction accuracy, as it was able to use these two features to understand the data rather quickly.

Overall, this case study demonstrates a successful implementation of an automized classification system for celestial bodies.