MUSHROOM EDIBILITY PREDICTION

CEN 451 01: Artificial Neural Networks

Eren KIZILIRMAK
210704025

Tunahan Türker ERTÜRK
200706019

Berru HANEDAR
200706028

CONTENTS

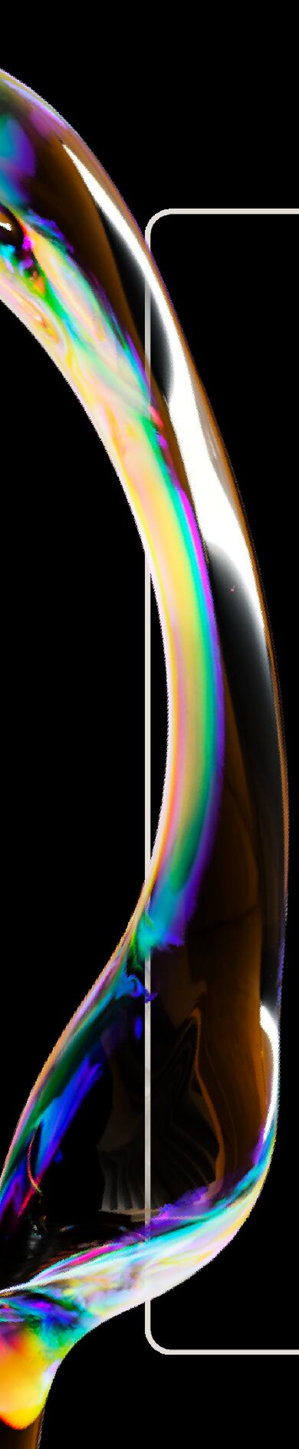
INTRODUCTION

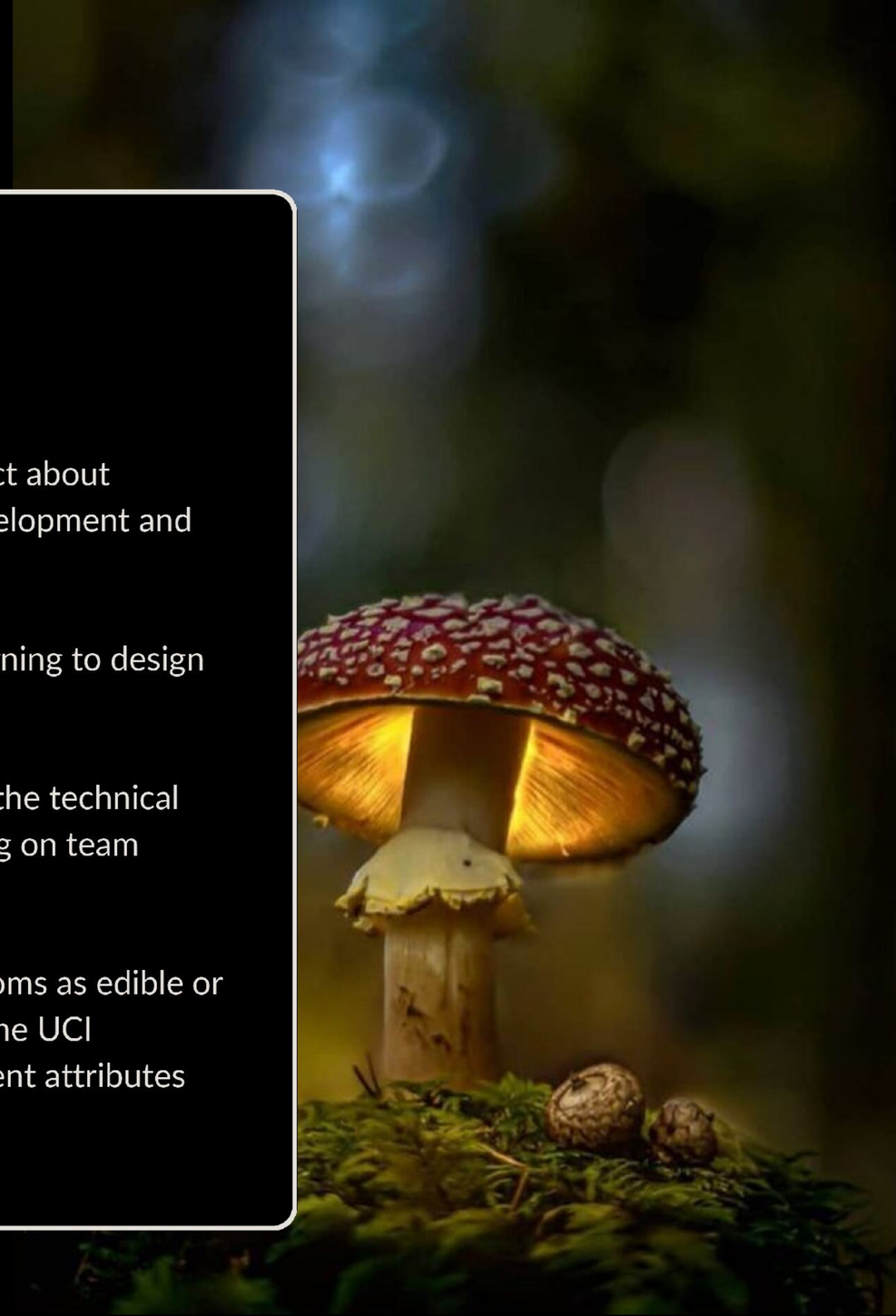
This project summarizes CEN 451 01: Artificial Neural Networks' course project about Mushroom Edibility Prediction, which we gained experience in ML project development and testing.

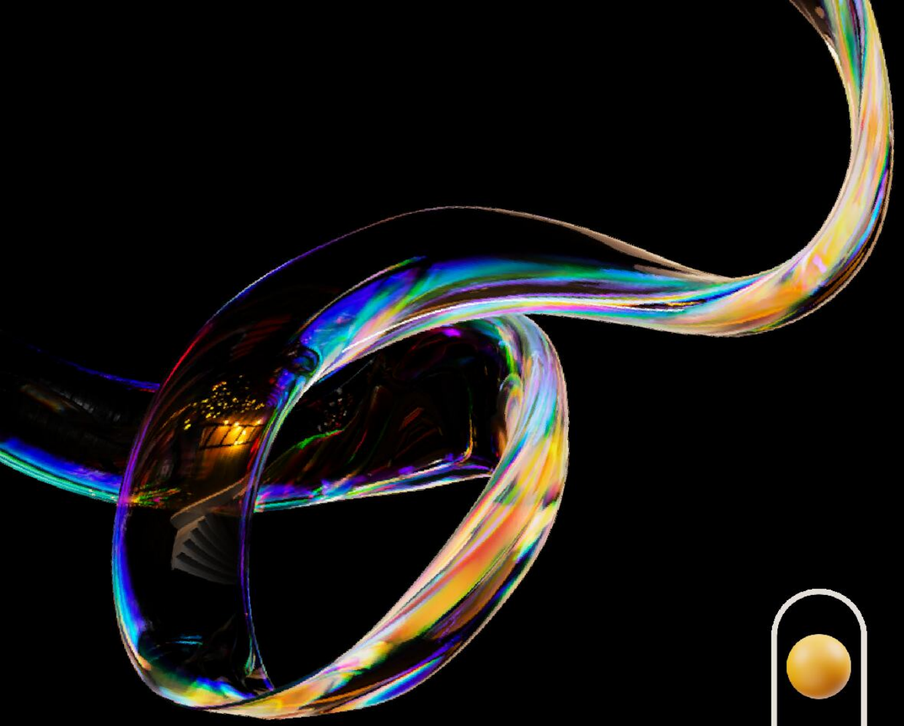
The main goal of this project is to gain experience in MLPs. We did that by learning to design and implement a 3-layered MLP Neural Network.

It was also important to improve our programming skills in Python. Besides all the technical work we have also learned to work in a team and learned to divide tasks among on team members.

This project focuses on developing a neural network model to classify mushrooms as edible or poisonous based on their physical characteristics." "The implementation uses the UCI Mushroom Dataset, which gives a collection of mushroom samples with different attributes that can be used to predict edibility.







PROJECT OVERVIEW & TECHNOLOGIES

Throughout development of the project, We learned Pytorch for building the core ML blocks, Matplotlib & Seaborn for plotting and monitoring, Scikit-learn for applying different metrics, Numpy for array operations and Pandas for statistical exploratory data analysis.

We firstly selected a dataset to build our model on, in our case Mushroom dataset. Then, we designed an abstract prediction model. Then, we learned to follow our diagram by using Pytorch.

Towards the end part of the project, We discovered more advanced topics such as Model Monitoring, Batch Normalization for better performance and applying applying ADAM optimizer for momentum and dynamic learning rate control.

Python

Python is a popular choice "for data science, machine learning, and web development.

Git

Git, a distributed version control system... is designed to handle projects of any size with speed and efficiency." Git "enables multiple developers to work on the same project simultaneously, tracking changes and managing different code versions.

GitHub

GitHub... is a platform for collaboration, code sharing, and version control, offering features like pull requests, issue tracking, and project management tools.

Visual Studio Code

Visual Studio Code supports a wide range of programming languages and includes features such as Integrated Terminal, Debugging Support, and Extensions for Additional Functionality.

DATASET DESCRIPTION

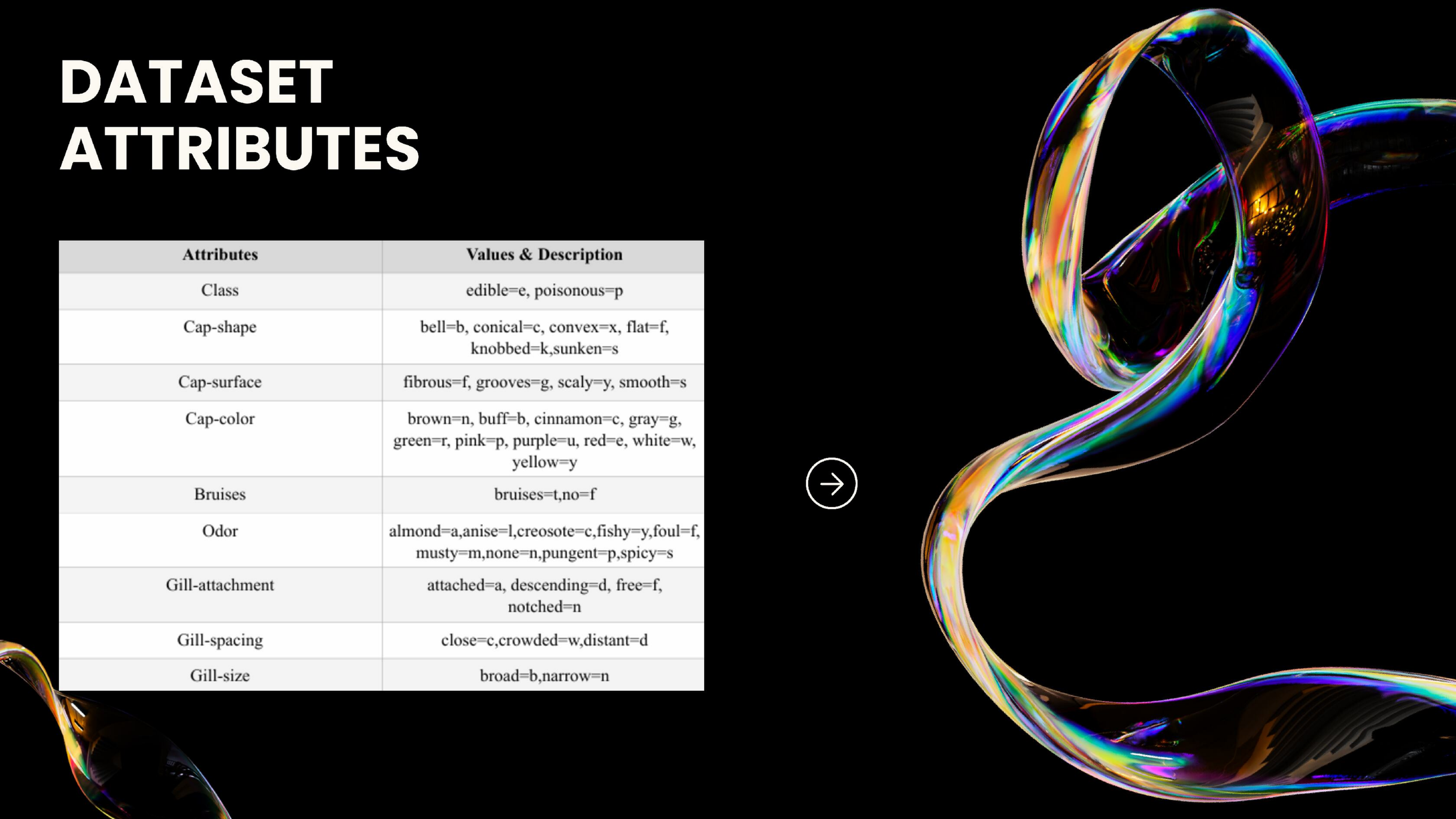
The UCI Mushroom Dataset, created by Jeff Schlimmer in 1981, consists of 8,124 mushroom samples from 23 species of gilled mushrooms in the Agaricus and Lepiota families." "Each mushroom in the dataset is labeled as edible or poisonous based on field guides.

The dataset has no missing values and consists only of categorical attributes.

The dataset contains 4,208 edible samples and 3,916 poisonous samples a balanced distribution of the target classes.

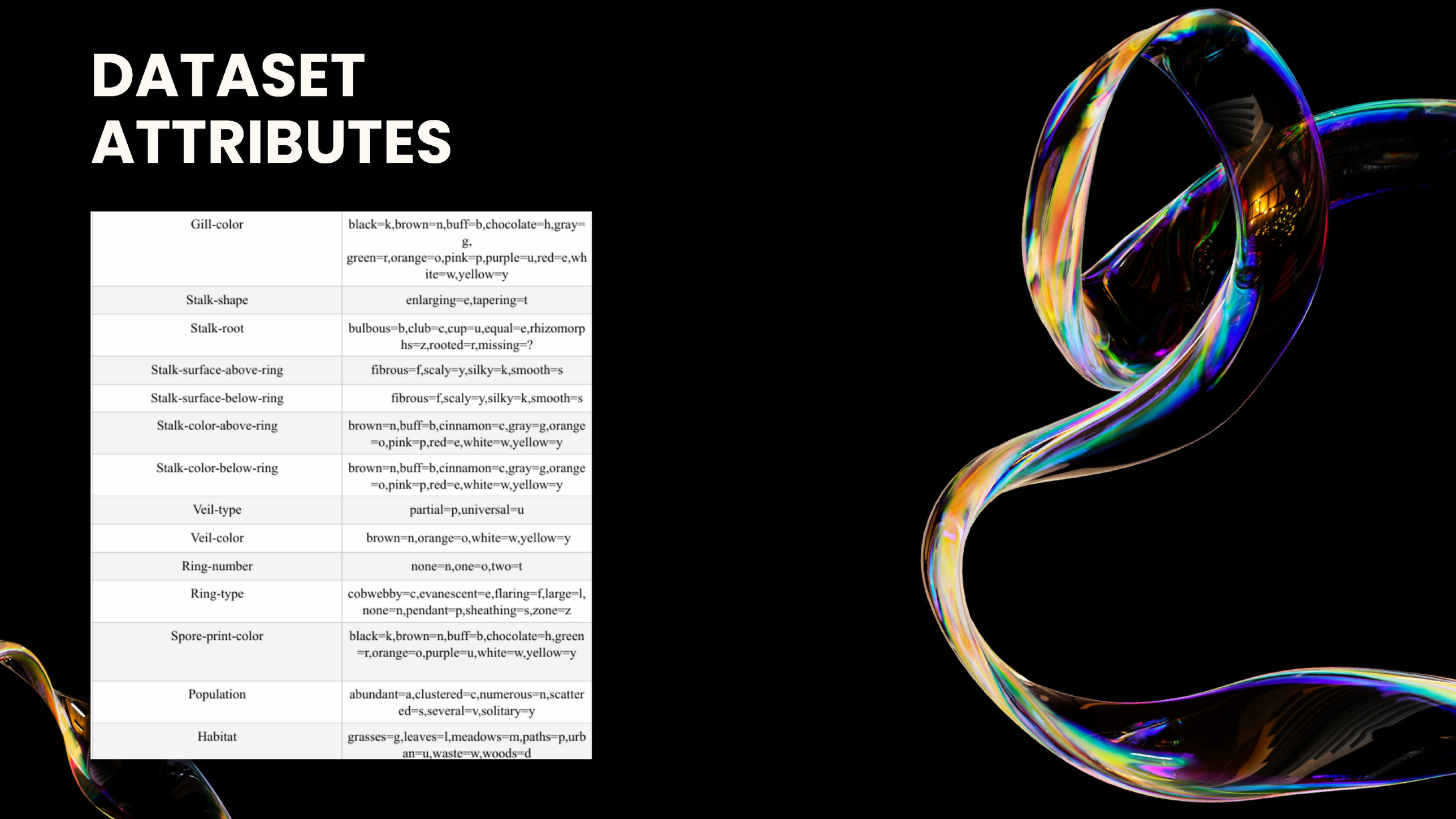
DATASET ATTRIBUTES

Attributes	Values & Description
Class	edible=e, poisonous=p
Cap-shape	bell=b, conical=c, convex=x, flat=f, knobbed=k,sunken=s
Cap-surface	fibrous=f, grooves=g, scaly=y, smooth=s
Cap-color	brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y
Bruises	bruises=t,no=f
Odor	almond=a,anise=l,creosote=c,fishy=y,foul=f, musty=m,none=n,pungent=p,spicy=s
Gill-attachment	attached=a, descending=d, free=f, notched=n
Gill-spacing	close=c,crowded=w,distant=d
Gill-size	broad=b,narrow=n



DATASET ATTRIBUTES

Gill-color	black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e,white=w,yellow=y
Stalk-shape	enlarging=e,tapering=t
Stalk-root	bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=?
Stalk-surface-above-ring	fibrous=f,scaly=y,silky=k,smooth=s
Stalk-surface-below-ring	fibrous=f,scaly=y,silky=k,smooth=s
Stalk-color-above-ring	brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
Stalk-color-below-ring	brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
Veil-type	partial=p,universal=u
Veil-color	brown=n,orange=o,white=w,yellow=y
Ring-number	none=n,one=o,two=t
Ring-type	cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p,sheathing=s,zone=z
Spore-print-color	black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white=w,yellow=y
Population	abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y
Habitat	grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d



DATA PREPROCESSING & EDA

Exploratory Data Analysis (EDA) We applied EDA technique for the dataset. This includes using plotting to check distributions of the columns, correlations, checking missing data and unique values for each columns.

Distribution of the dataset was: 4,208 edible (51.8%) and 3,916 poisonous (48.2%) samples.

Even though no missing values were present in the dataset, Figure 2. shows there was a column, veil-type, which had only 1 unique value for the entire dataset. We dropped that column because it hadn't provided any useful information.

For neural network, preprocessing steps including one-hot encoding are necessary to convert the categorical data into numerical format (float32).

We created 3 sets of data using the dataset. first one, training, is only used for both forward propagation and back propagation. It's the 70% of the dataset. second one, validation, is only used during training to calculate the validation accuracy and validation loss to make model robust to unseen data. It's the 15% of the data. Third one, test, is created for only testing which is important to test the model's final robustness. It's the 15% of the data.

DATA PREPROCESSING & EDA

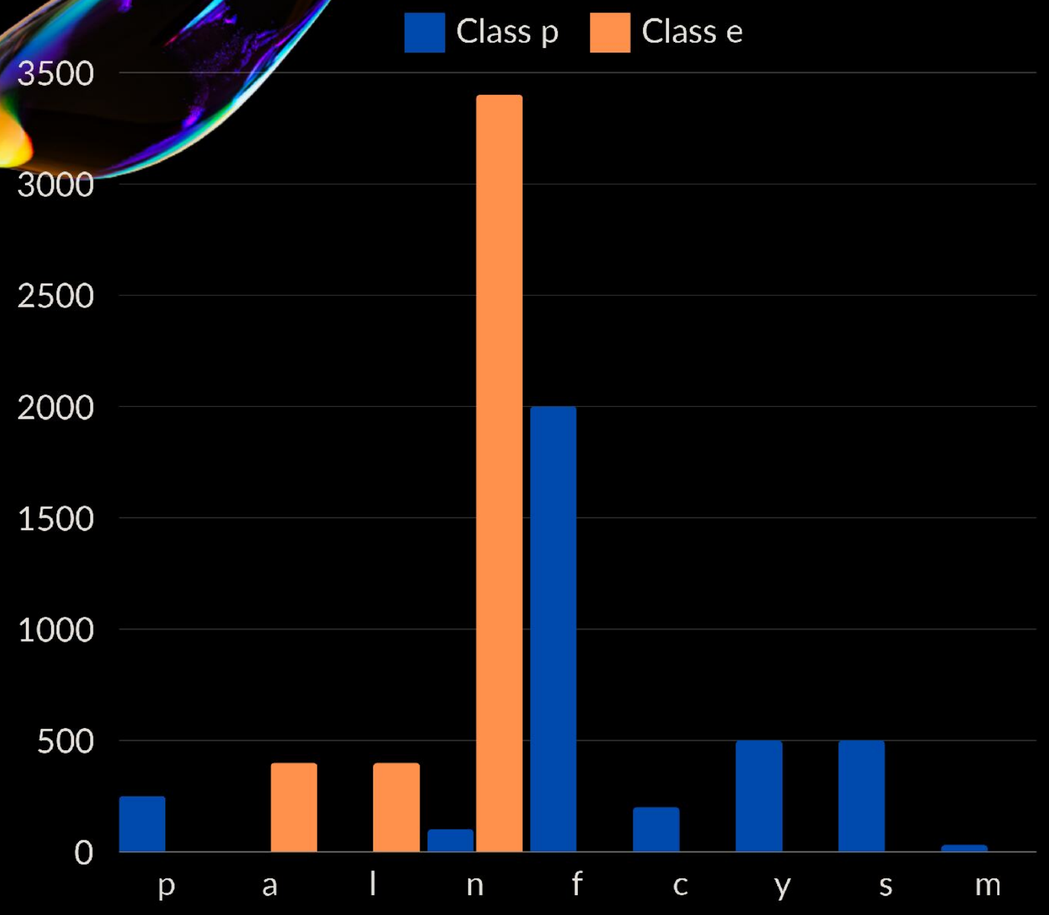
Figure 2. Unique values for each column

```
a = mushroom.describe()[1:2]
a = a.transpose()
display(a) #as we can see the veil-type is irrel

a["unique"].sum()

mushroom = mushroom.drop(columns="veil-type")
```

[233]

	unique
class	2
cap-shape	6
cap-surface	4
cap-color	10
bruises	2
odor	9
gill-attachment	2
gill-spacing	2
gill-size	2
gill-color	12
stalk-shape	2
stalk-root	5
stalk-surface-above-ring	4
stalk-surface-below-ring	4
stalk-color-above-ring	9
stalk-color-below-ring	9
veil-type	1

DISTRIBUTION OF ODOR BY CLASS



Odor emerged as a potentially strong predictor, with certain odors (e.g., foul, pungent) strongly correlated with poisonous mushrooms. Figure 3. shows the distribution of odor and it's relation to poisonous class.

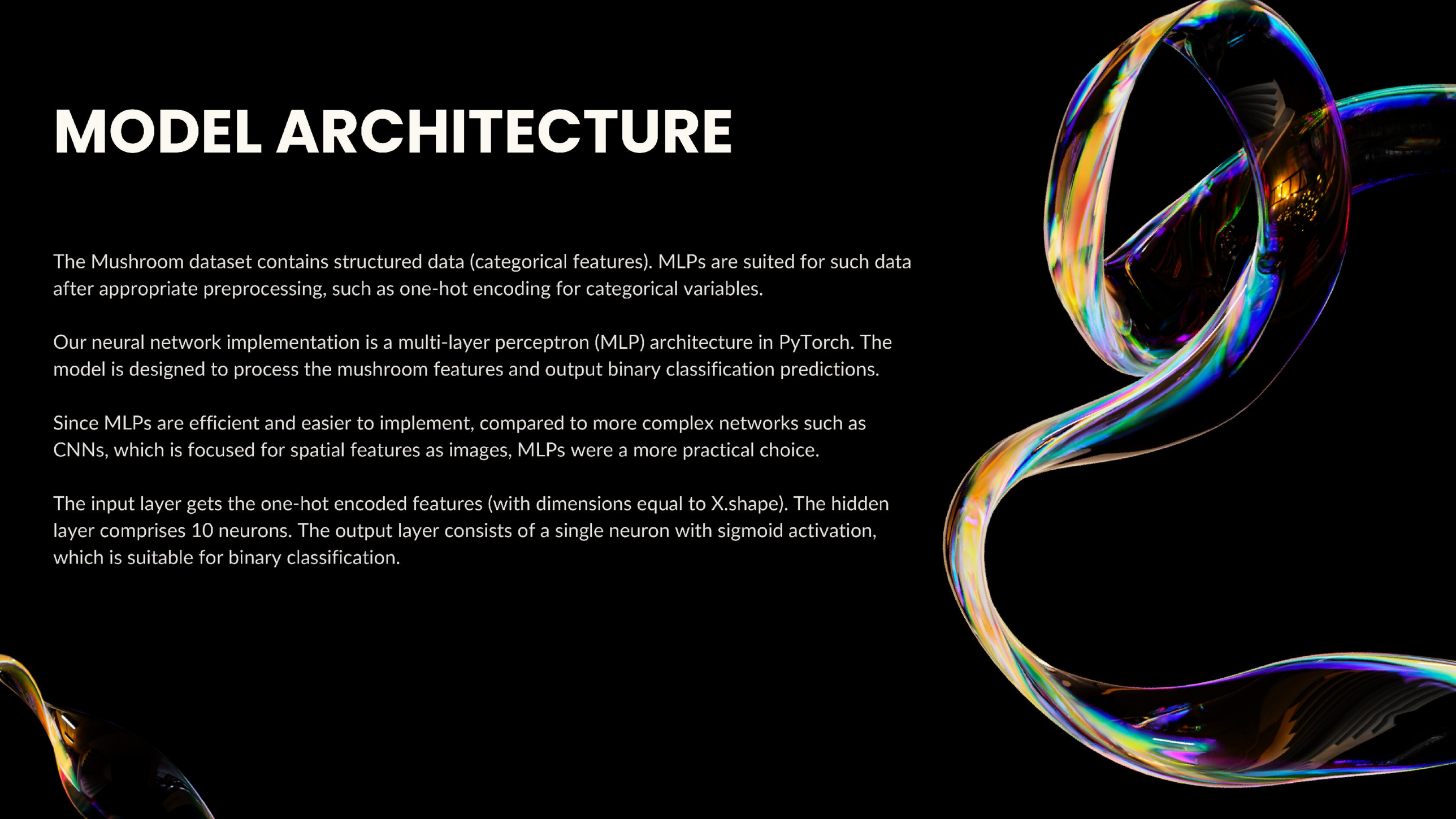
MODEL ARCHITECTURE

The Mushroom dataset contains structured data (categorical features). MLPs are suited for such data after appropriate preprocessing, such as one-hot encoding for categorical variables.

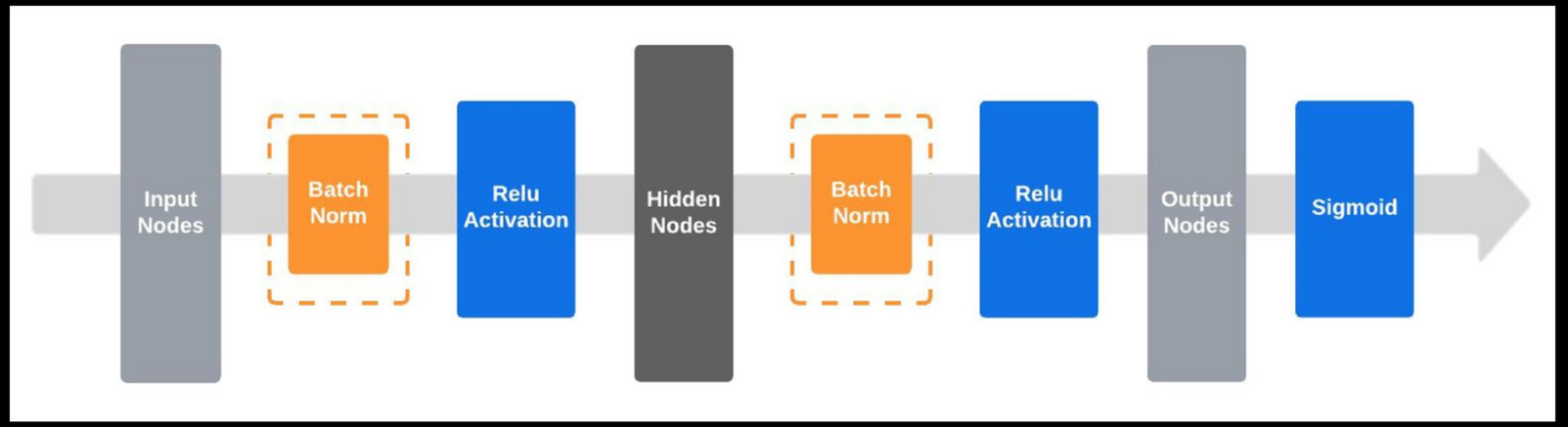
Our neural network implementation is a multi-layer perceptron (MLP) architecture in PyTorch. The model is designed to process the mushroom features and output binary classification predictions.

Since MLPs are efficient and easier to implement, compared to more complex networks such as CNNs, which is focused for spatial features as images, MLPs were a more practical choice.

The input layer gets the one-hot encoded features (with dimensions equal to X.shape). The hidden layer comprises 10 neurons. The output layer consists of a single neuron with sigmoid activation, which is suitable for binary classification.



MODEL ARCHITECTURE



MODEL IMPLEMENTATION (CODE HIGHLIGHTS)

```python
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()

        self.input_layer = nn.Linear(X.shape[1], 10)
        self.norm1 = nn.BatchNorm1d(10)
        self.relu1 = nn.ReLU()

        self.linear1 = nn.Linear(10, 10)
        self.norm2 = nn.BatchNorm1d(10)
        self.relu2 = nn.ReLU()



        self.linear2 = nn.Linear(10, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.input_layer(x)
        x = self.norm1(x)
        x = self.relu1(x)
        x = self.linear1(x)
        x = self.norm2(x)
        x = self.relu2(x)
        x = self.linear2(x)
        x = self.sigmoid(x)
        return x

model = MyModel()
model.to("mps")
```

Each layer uses Linear Transformations which is implemented using non-Linear. Hidden layers and Input layer Uses Batch Norm to prevent the in covariate shift and a RELU activation to prevent vanishing gradients for faster calculation and non-linearity. Output layer only consists of a linear transformation nodes and a sigmoid Activation which is applied finally to predict the given output. Since it's a binary classification problem, using sigmoid function is suitable for scaling the output for probability."

Forward pass function implements how data flows through the network. Since we are creating an MLP, there is a casual forward pass without skipping layers.

TRAINING & VALIDATION

The training process was feeding our preprocessed data through the network in 64 batches. During each training iteration, The input data passes through the network, with batch normalization applied between layers. The ReLU activation function processed the normalized outputs. The Binary Cross-Entropy loss is calculated between the model's predictions and the true labels. And Adam optimizer updated the model parameters based on the computed gradients. This process was repeated for 15 epochs until the model achieved satisfactory performance on both training and validation sets.

During the training, each batch of data passes through the network in a forward pass, at the end Binary Cross-Entropy loss is calculated between predictions and true labels. The model then performs back propagation and updates parameters using the Adam optimizer. This process repeats for each batch in epoch.

The validation phase checks the model's performance on unseen data without computing gradients. This is for monitoring the model's robustness capability and to detect potential overfitting. During validation, we compute the same metrics as training but don’t apply back propagation.

PERFORMANCE MONITORING & VISUALIZATION

```python
epochs = 15
for epoch in range(epochs):
    total_acc_train = 0
    total_acc_val = 0
    total_loss_train = 0
    total_loss_val = 0

    for data in train_dataloader:


        ######################
        #this part is training the model with batches.

        inputs, labels = data

        prediction = model(inputs).reshape(len(labels)) #we need thiş beacuse prediction is 8,1 dimension

        #print(prediction)

        batch_loss = criterian(prediction, labels)

        total_loss_train += batch_loss.item()

        acc = (prediction.round() == labels).sum().item() #calculating accuracy of a batch item !!!

        total_acc_train += acc

        batch_loss.backward()
        optimizer.step()
        optimizer.zero_grad()


        ############################

    #this part is for validation during training.
    with torch.no_grad():
        for data in validation_dataloader:
            inputs, labels = data
            prediction = model(inputs).reshape(len(labels))
            batch_loss = criterian(prediction, labels)
            total_loss_val += batch_loss.item()

            acc = (prediction.round() == labels).sum().item()
            total_acc_val += acc
```

FIGURE 8. TRAINING LOOP

Figure 8. tracks loss and accuracy performance metrics throughout training. The training and validation losses measure the model's prediction error on their datasets, while accuracies show the percentage of correct predictions.

PERFORMANCE MONITORING & VISUALIZATION

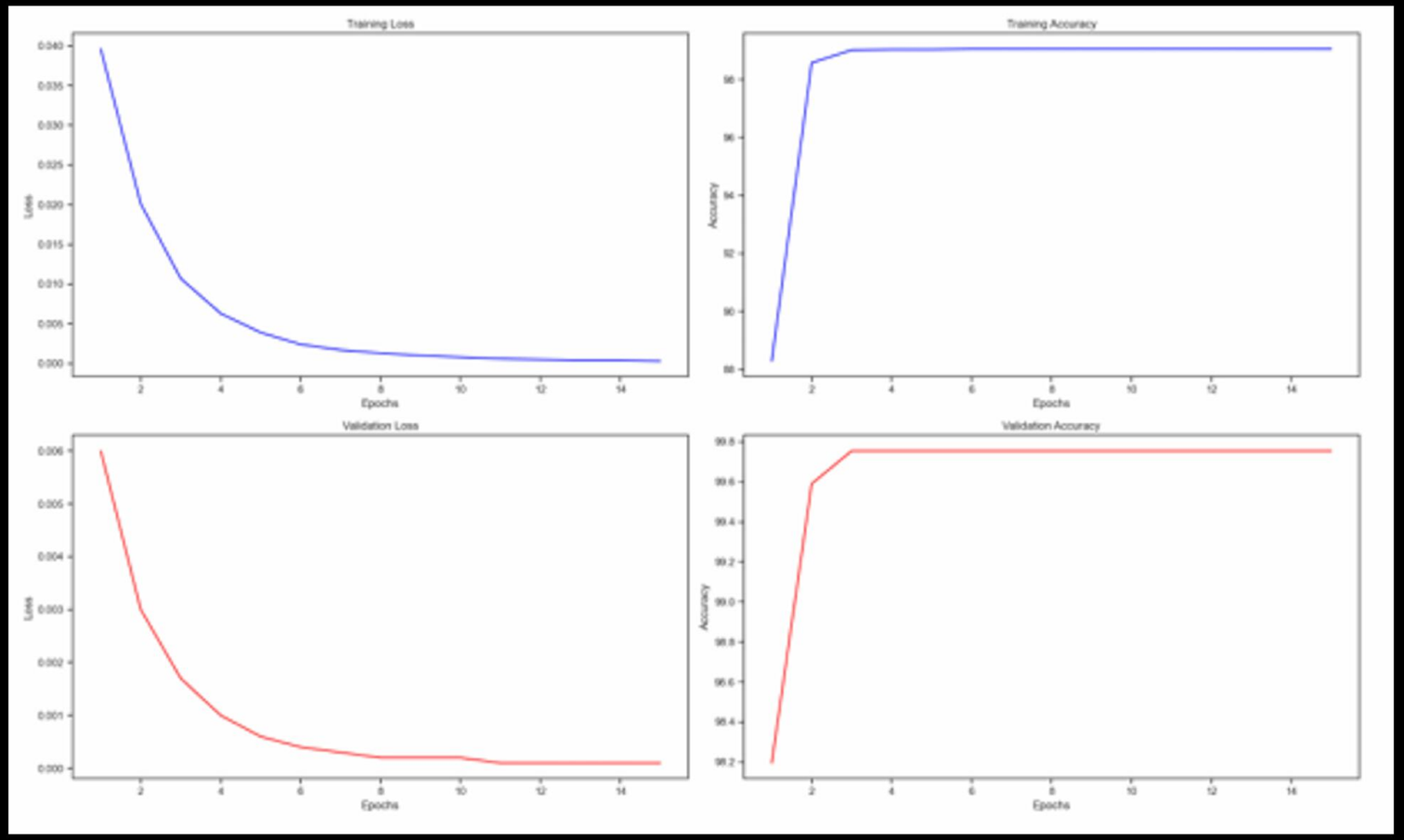


Figure 10. By maintaining different training and validation metrics, we can see how the model generalizes to unseen data and make decisions about when to stop training or adjust its parameters.

MODEL TESTING & RESULTS

After completing the training process, we tested our model's performance on the unseen test dataset to assess its generalization capability and predictive accuracy. Test implementation is the same as validation.

```
from sklearn.metrics import classification_report

matrix = classification_report(y_true=yss_arr, y_pred=preds_arr)
print(matrix)
```

```
85]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       607
         1.0       1.00      1.00      1.00       609

    accuracy                           1.00      1216
   macro avg       1.00      1.00      1.00      1216
weighted avg       1.00      1.00      1.00      1216
```

Performance Metrics: "The model achieved performance on the test dataset with the its metrics Test Loss was 0.0001 and the Test Accuracy is 99.75%.

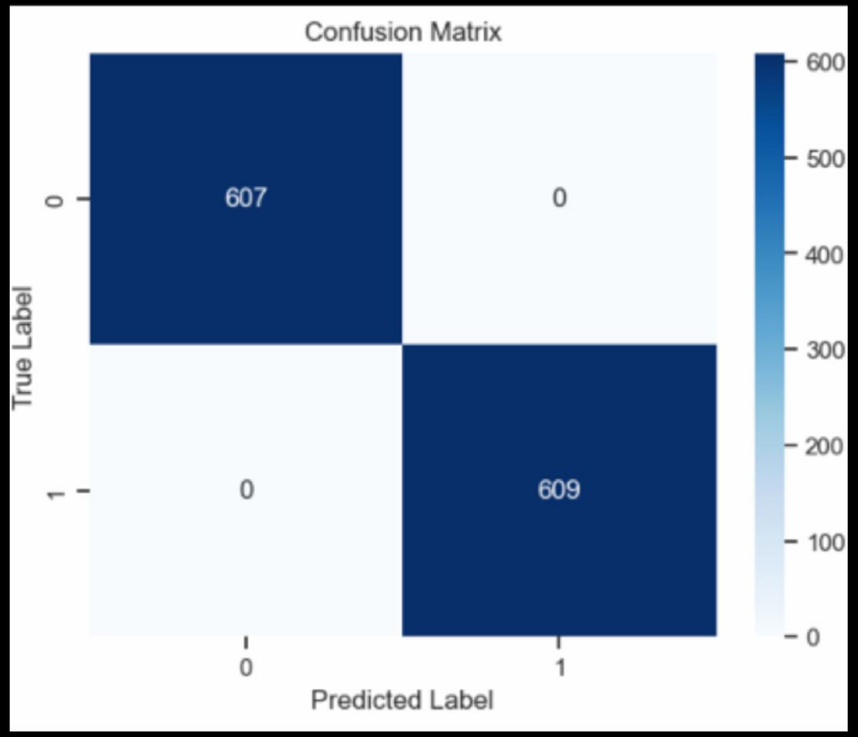
To gain deeper information about model’s performance, we used classification_report from sklearn.metrics. This function gave us Precision, Recall, f-1 and their averages.

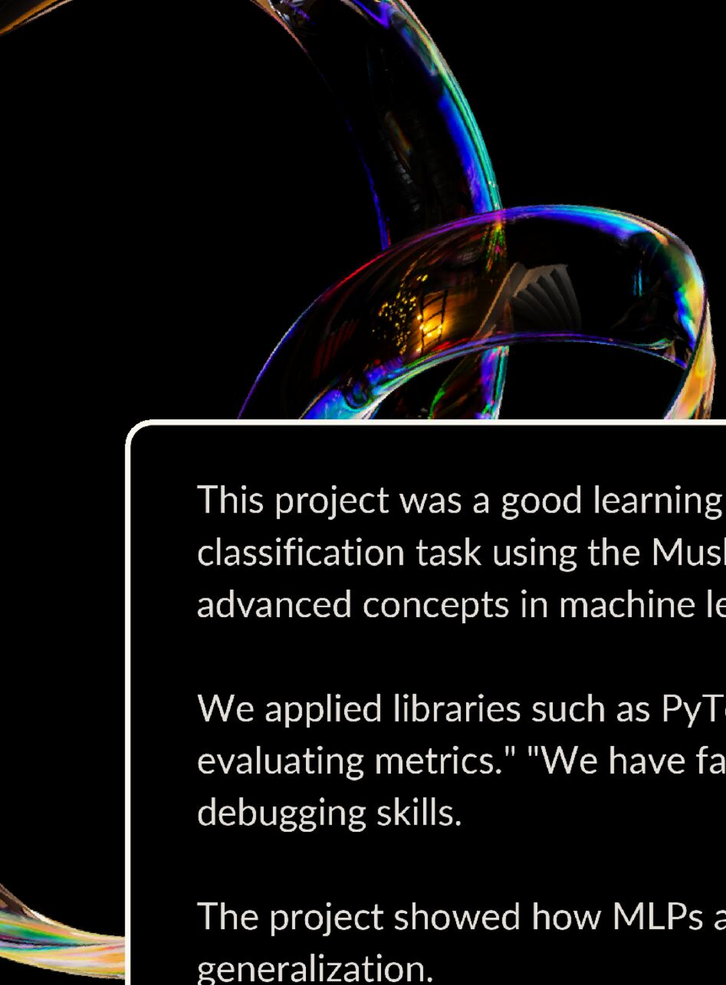
MODEL TESTING & RESULTS

Confusion Matrix Visualization: The confusion matrix visualization provides an aesthetic view of the model's prediction performance. The deep blue colors show strong classification performance. This shows that our model has successfully learned the earning features between edible and poisonous mushrooms from the training data and it’s robust.

```
from sklearn.metrics import confusion_matrix

preds_arr = np.array(preds).reshape(-1)
yss_arr = np.array(yss).reshape(-1)
# display(preds_arr.reshape(-1).__len__())
# display(preds.__len__())
calc = confusion_matrix(y_true=yss_arr, y_pred=preds_arr)
a = sns.heatmap(data=calc,annot=True, fmt="d",cmap=plt.cm.Blues)
a.set_ylabel("True Label")
a.set_xlabel("Predicted Label")
a.set_title("Confusion Matrix")
# sns.heatmap(data=calc)
```

```
xt(0.5, 1.0, 'Confusion Matrix')
```





CONCLUSIONS

This project was a good learning experience in designing, implementing, and optimizing a Multi-Layer Perceptron (MLP) for a binary classification task using the Mushroom dataset. The gaining experience with MLPs was achieved, along with an understanding of advanced concepts in machine learning and neural network architecture.

We applied libraries such as PyTorch for model construction, Matplotlib and Seaborn for visualizations, and Scikit-learn for evaluating metrics." "We have faced issues such as data preprocessing, overfitting, and runtime errors, which improved our debugging skills.

The project showed how MLPs are well suited for binary classification problems, by the final model scoring robust accuracy and generalization.

Regularization techniques such as using ADAM optimizer, improved model’s performance by applying momentum for gradients and dynamically improving learning rate." "Our programming skills in Python, analyzing data, using Pytorch, Scikit-learn Matplotlib, seaborn, are improved.

MEET THE TEAM

Thank you for your time! Reach out to us for questions.

Eren KIZILIRMAK
210704025

Tunahan Türker ERTÜRK
200706019

Berru HANEDAR
200706028

THANK YOU

for your time and attention

CEN 451 01: Artificial Neural Networks