Predicting badminton shot types using Convolutional Neural Networks and Multi-Layer Perceptron

1. Introduction

In competitive sports such as badminton, keen observation skills are essential for improving player performance and analyzing opponents' strategies. A critical aspect of this is the accurate identification of shot types. Traditionally, coaches and analysts manually classify shots and analyze players, which is both time-consuming and prone to human error. Implementing machine learning (ML) to automate this process presents a significant opportunity to enhance the analysis of player techniques. By classifying shot types from images of badminton rallies, ML can offer real-time feedback or detailed post-match analysis, assisting both players and coaches in refining strategies. Looking forward, this model could be applied to analyze entire rallies, matches, or even tournaments, generating valuable insights such as shot frequency, shot success rates, and individualized playing styles.

This report outlines the application of machine learning for classifying badminton shots. We begin by formulating the problem, followed by a discussion of the dataset and its structure. Next, we discuss two methods – a Convolution Neural Network (CNN) and a Multi-Layer Perceptron (MLP) – in detail and then compare the outcomes of the two methods on the dataset. Finally, in section 5, we conclude the work and discuss future applications.

2. Problem Formulation

The task of classifying badminton shots based on images can be formalized as a supervised classification problem. In this context, each image represents a data point, and the goal is to predict the type of shot being played—such as a serve, smash, clear, or lift shot—based on the visual features extracted from the image.

Each data point corresponds to a single image captured during a badminton rally, showing a player in the act of attempting a shot and another player waiting to return it. The data points are images and thus categorical data from a dataset on RobotFlow Universe (Singhaklangpol, 2023). The dataset contains still images taken from professional level matches, with the players and shot types readily identified. For both the CNN and MLP model, the features of this data will be the individual pixels from each image. Although the data itself is categorical, when fed as an input to either the MLP model or CNN model, the pixels are classified as continuous data. The labels are already available in the dataset and consist of four different badminton shots: clear, lift, serve, and smash.

3. Methods

3.1 Dataset, preprocessing and feature selection

There are 1571 total datapoints, which are still images of badminton rallies from professional level matches. The dataset was downloaded from the source as JPG images, along with a CSV file containing the filenames of images and data on the labels. The data was preprocessed to remove the “null” and “wait for defense” labels. The null labels have no shot assigned to the photo and thus are not useful in training the model. The “wait for defense” labels are present in most images and are assigned to the player who is not actively performing a shot. These labels introduce unnecessary noise into the data and can also heavily bias our models.

Although the images were uniformly sized at 640 x 640 pixels, they were resized to 240 x 240 pixels to reduce computational costs and enable the model to train within a reasonable timeframe. For the Convolutional Neural Network (CNN) specifically, we applied data augmentation techniques, including random jitter, to enhance the dataset. Figure 1 provides an example of a datapoint with its corresponding label.

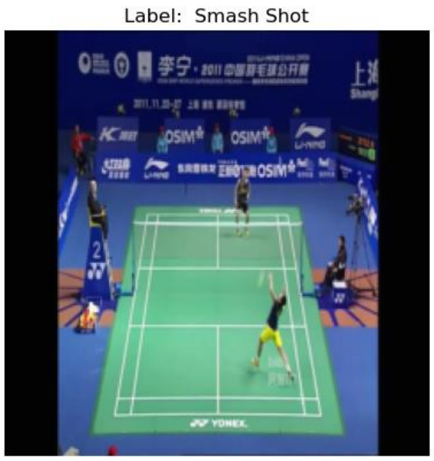


Figure 1 – An example image in the dataset, labelled as a smash shot

The features for both models are the pixels themselves. In a CNN, the feature extraction process is done automatically through convolutional layers. Hence, no manual feature engineering was required. The images and their pixels serve as the input data and the convolutional layers will detect patterns such as edges, textures, and shapes, which are hierarchically combined to represent more complex features like player posture, racket positioning and shot motion. No manual feature processing was required for the MLP either, it takes the raw pixels and flattens them resulting in a feature vector of 12 288 dimensions (64 x 64 x 3 channels) and then normalizes each pixel value, resulting in values between 0 and 1.

3.2 Models

As stated previously, the two models we will be using are a Convolutional Neural Network and a Multi-Layer Perceptron.

A CNN is well-suited for image classification tasks, as it automatically extracts spatial hierarchies of features through convolution and pooling layers. The hypothesis space of a CNN includes non-linear predictors, which are necessary to capture the complex variations in the posture, racket angle, and other subtle visual cues that distinguish different types of shots in badminton. Unlike traditional MLPs, CNNs are specifically designed to deal with grid-like data, like images.

The CNN Model we constructed consists of two parts: the feature extractor and classifier. The feature extractor is constructed from 3 convolutional layers, each followed by ReLU activation and maxpooling. The classifier consists of adaptive average pooling, flattening and two fully connected layers with a dropout layer in between.

An MLP is better suited for multi-class classification problems than many other models as it has multiple layers of neurons and non-linear activation functions that can approximate highly non-linear decision boundaries. This is crucial in a task such as this one, as boundaries separating classes are complex and not linearly separable. Our MLP architecture looks like this:

Input Layer (12,288 nodes) -> Dense(256) -> Dense(128) -> Dense(64) -> Output(4)

The model includes ReLU activation functions for hidden layers, softmax activation for the output layer and dropout layers (0.3) between the dense layers for regularization. Dropout layers help in preventing overfitting and gradually decreasing layer sizes help in learning hierarchical representations.

3.3 Loss function

The cross-entropy loss function (log loss) was used for both models. This is because it is appropriate for multi-class classification tasks, where the goal is to predict the likelihood of an image belonging to a particular class (in this case, a shot type). It is a generalization of logistic loss for multi-class classification problems. The loss is calculated for each class and summed across all classes by using the following equation (Fortuner, 2017):

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

Where:

- M is the number of classes
- y is the binary indicator (0 or 1) if class label c is the correct classification for observation o
- p – predicted probability observation o is of class

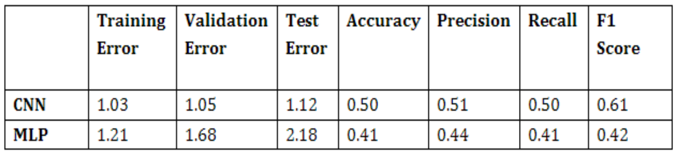
Cross-entropy minimizes the difference between the predicted probabilities and the actual class labels, making it an ideal loss function for a CNN and MLP, in this context (Fortuner, 2017). For the MLP the loss function was already implemented in the MLPClassifier provided by the Sklearn library, making it easier to use than other measures of loss.

3.4 Model validation

The datapoints were split as follows (for both models): 1357 images for training, 142 validation, and 75 for testing (approximately 86% training, 9% validation, and 5% test). This is a single-split approach, and we chose to split the data in exactly this manner as it ensures our model has enough data to learn from while also maintaining separate sets for hyperparameter tuning (validation) and performance evaluation (testing). Also, MLP's typically require more training data as they do not have built in image specific inductive biases. Additionally, as we have a small data set, it is even more important to have a large training set. This is also the split that the dataset initially suggested.

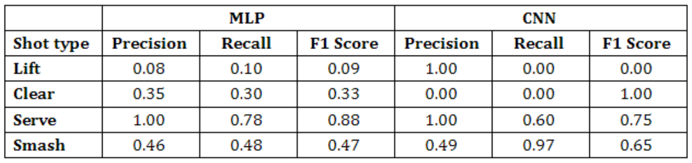
4. Results

To assess the performance of our models in this multi-class classification problem, we employ the metrics of accuracy, precision, recall, and F1-score (Table 1 & 2). Additionally, we track training error, validation error, and test error as supplementary evaluation metrics. While these latter metrics provide useful insights, the primary evaluation metrics—accuracy, precision, recall, and F1-score—are generally considered more appropriate for a problem of this nature.



	Training Error	Validation Error	Test Error	Accuracy	Precision	Recall	F1 Score
CNN	1.03	1.05	1.12	0.50	0.51	0.50	0.61
MLP	1.21	1.68	2.18	0.41	0.44	0.41	0.42

Table 1 – evaluation metrics



	MLP			**CNN**		
Shot type	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
Lift	0.08	0.10	0.09	1.00	0.00	0.00
Clear	0.35	0.30	0.33	0.00	0.00	1.00
Serve	1.00	0.78	0.88	1.00	0.60	0.75
Smash	0.46	0.48	0.47	0.49	0.97	0.65

Table 2 – Metrics by shot type

According to Table 1, the CNN model shows a small difference between training and validation errors (0.02), while the MLP shows a larger gap (0.47), along with a high test error of 2.18. This indicates that the MLP has overfitted on the training set, whereas the CNN generalizes better and performs well on the test set. Overall, CNN outperforms the MLP across most metrics, with 9 percentage points higher accuracy and a 19 percentage point advantage in F1-score.

However, a closer look at Table 2 reveals that the MLP performs better on specific shot types, such as clears, and predicts all shot types, while the CNN fails to predict any lift or clear shots (recall score of 0.00). CNN also tends to classify most shots as smashes, as is made clear in the confusion matrix in the appendix.

Ultimately, we selected the CNN as our final model. It demonstrated higher overall accuracy, more consistent performance across all classes, reduced overfitting, and better generalization to the test data. The test set was created by partitioning the original dataset of 1,571 images into training (86%), validation (9%), and test (5%) sets. As a result, the test set consists of 75 images used for the final evaluation of the trained model. The chosen model, our CNN, achieved a test error of 1.12 on this test set, as shown in Table 1.

5. Conclusion

In this report, the performance of Convolutional Neural Networks (CNN) was compared to that of Multi-Layer Perceptron's (MLP) in predicting shot types from professional-level still images of badminton rallies, using various evaluation metrics. The results indicated that the CNN generally outperformed the MLP in terms of accuracy, precision, recall, F1-score, and exhibited lower training, validation, and test errors. These findings align with existing literature, which consistently demonstrates that CNNs tend to outperform MLPs in image classification tasks due to their more robust and stable performance. MLPs, by contrast, are more susceptible to overfitting when applied to image data.

However, neither model performed particularly well. Comparing Cross-Entropy Loss values to literature benchmarks highlights the CNN's shortcomings, especially its failure to predict clear and lift shots, while overpredicting smash shots. This is likely due to class imbalance, as clear shots are underrepresented, and smash shots are overrepresented in the dataset. This imbalance leads the model to bias toward smash shots, especially when distinguishing between visually similar clear and smash shots. Future improvements should focus on increasing the dataset size and balancing the distribution of shot types, which would benefit both models.

To further improve the CNN, class weights can be adjusted to handle imbalance, the model's complexity can be increased, and transfer learning from a pre-trained model could be employed. For the MLP, using pose estimation libraries like OpenPose to extract key points representing body parts, instead of using raw images, could provide more effective input features in numerical formats such as CSV or JSON.

6. Bibliography

CloudFactory Computer Vision Wiki. (2024). *Cross-Entropy Loss*. [online] Available at: https://wiki.cloudfactory.com/docs/mp-wiki/loss/cross-entropy-loss [Accessed 8 Oct. 2024].

Hidalgo, G., Cao, Z., Simon, T., Wei, S.-E., Raaj, Y., Joo, H. and Sheikh, Y. (2020). *OpenPose*. [online] cmu-perceptual-computing-lab.github.io. Available at: https://cmu-perceptual-computing-lab.github.io/openpose/web/html/doc/index.html.

Fortuner, B. (2017). *Loss Functions — ML Glossary documentation*. [online] ml-cheatsheet.readthedocs.io. Available at: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html#cross-entropy [Accessed 20 Sep. 2024].

Marttinen, Sigg, Moen (2024) 'Machine Learning D MyCourses' [Online course material], *CS-CS240 Machine Learning*. Aalto University. [Accessed 20 Sep. 2024]

Sivakorn Singhaklangpol (2023). *Badminton project V2 Object Detection Dataset*. [online] Roboflow Universe. Available at: https://universe.roboflow.com/117sivakorn-singhaklangpol-niqct/badminton-project-v2/dataset/1 [Accessed 20 Sep. 2024].

7. Appendix

Data Preprocessing

```
# Importing images
import zipfile
import os

# Path to zip file and destination folder
zip_file_path = 'zipped_images.zip'  # Replace with the actual zip file path
destination_folder = 'unzipped_images'       # Folder to extract files into

Create destination folder if it doesn't exist
if not os.path.exists(destination_folder):
    os.makedirs(destination_folder)

# Unzip folder
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(destination_folder)

print(f"Unzipped files to: {os.path.abspath(destination_folder)}")
```

```
# Load the CSV file to drop the last available column -- already done, commented out
csv_file = 'classes.csv'
df = pd.read_csv(csv_file)

label_columns = df.columns
print(label_columns) # Print column names

df = df.drop(columns=[' Wait for defense']) # Drop the 'Wait for defense' column
df['non_zero_count'] = (df[label_columns] != 0).sum(axis=1) # Calculate sum of label values
df_filtered = df[df['non_zero_count'] != 1] # Drop any rows without one label (e.g. no label or two
labels)
df_filtered = df_filtered.drop(columns=['non_zero_count'])
df_filtered.to_csv('filtered_classes.csv', index=False) # Save datarframe to new csv
```

Convolutional Neural Network

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from PIL import Image
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
```

```python
# Modify the dataset class to return a single label
class BadmintonDataset(Dataset):
    def __init__(self, csv_file, img_dir, transform=None):
        self.annotations = pd.read_csv(csv_file)
        self.img_dir = img_dir
        self.transform = transform

        print("Columns in the CSV file:")
        print(self.annotations.columns)

        # Identify shot type columns (excluding 'filename')
        self.classes = [col for col in self.annotations.columns if col != 'filename']

        print("\nIdentified shot type columns:")
        print(self.classes)

        # Print sample data
        print("\nSample data (first 5 rows):")
        print(self.annotations.head())

        # Print data types of columns
        print("\nData types of columns:")
        print(self.annotations.dtypes)

    def __len__(self):
        return len(self.annotations)

    def __getitem__(self, index):
        img_name = self.annotations.iloc[index, 0]  # 'filename' column
        img_path = os.path.join(self.img_dir, img_name)

        if not os.path.exists(img_path):
            raise FileNotFoundError(f"Image file not found: {img_path}")

        image = Image.open(img_path).convert("RGB")

        # Get the index of the correct class (1 in the one-hot encoding)
        label = torch.tensor(self.annotations.iloc[index][self.classes].values.astype(np.float32))
        label = torch.argmax(label).item()

        if self.transform:
            image = self.transform(image)

        return (image, label)


# Data Transforms: Reduce dimensions
data_transforms = transforms.Compose([
    transforms.Resize((240, 240)),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Load Data
dataset = BadmintonDataset(csv_file='filtered_classes.csv', img_dir='images',
transform=data_transforms)

# Function to display the first image with its labels
def display_first_image(dataset):
    image, label = dataset[0]
    image = image.permute(1, 2, 0)  # Change from (C, H, W) to (H, W, C)
    image = image * torch.tensor([0.229, 0.224, 0.225]) + torch.tensor([0.485, 0.456, 0.406])  #
Denormalize
    image = image.clip(0, 1)  # Clip values to [0, 1] range


    plt.imshow(image)
    title = f"Label: {dataset.classes[label]}"
    plt.title(title)
    plt.axis('off')
    plt.show()

# Display the first image
display_first_image(dataset)
print("First image displayed.")
```

```
Columns in the CSV file:
Index(['filename', ' Clear Shot', ' Lift shot', ' Serve', ' Smash Shot'], dtype='object')

Identified shot type columns:
[' Clear Shot', ' Lift shot', ' Serve', ' Smash Shot']

Sample data (first 5 rows):
                                            filename   Clear Shot   Lift shot  \
0  SAIT_D2_1_321_jpg.rf.43f6d2c31e7648a1d38f4f0a3...            0           0
1  SAIT_D2_2_276_jpg.rf.078d95f2eb914dfc59961f095...            0           1
2  SAIT_D2_2_563_jpg.rf.45e6ea0a3949a0064a9d00d51...            0           1
3  SAIT_D2_4_48_jpg.rf.15a2248da0998628738a7cf939...            0           0
4  SAIT_D2_2_25_jpg.rf.3afb88e1c92a0b5137f0b154b5...            0           1

    Serve   Smash Shot
0       0            1
1       0            0
2       0            0
3       1            0
4       0            0
```

```
Data types of columns:
filename       object
 Clear Shot     int64
 Lift shot      int64
 Serve          int64
 Smash Shot     int64
dtype: object
```

Label: Smash Shot

```
First image displayed.
```

```python
# Split the dataset
train_size = int(0.86 * len(dataset))
val_size = int(0.09 * len(dataset))
test_size = len(dataset) - train_size - val_size
train_dataset, val_dataset, test_dataset = torch.utils.data.random_split(dataset, [train_size,
val_size, test_size])


# Create DataLoaders
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True) # initial batch size 32,
changed to 16
val_loader = DataLoader(val_dataset, batch_size=16, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)
```

```python
class BadmintonCNN(nn.Module):
    def __init__(self, num_classes):
        super(BadmintonCNN, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )
        self.classifier = nn.Sequential(
            nn.AdaptiveAvgPool2d((1,1)),
            nn.Flatten(),
            nn.Linear(256, 512),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(512, num_classes)
            # No activation function here; we'll use CrossEntropyLoss which includes LogSoftmax
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x
```

```
# Training loop
def train_model(model, train_loader, val_loader, criterion, optimizer, num_epochs, device):
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0

        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        # Validation phase
        model.eval()
        val_loss = 0.0
        correct = 0
        total = 0
        with torch.no_grad():
```

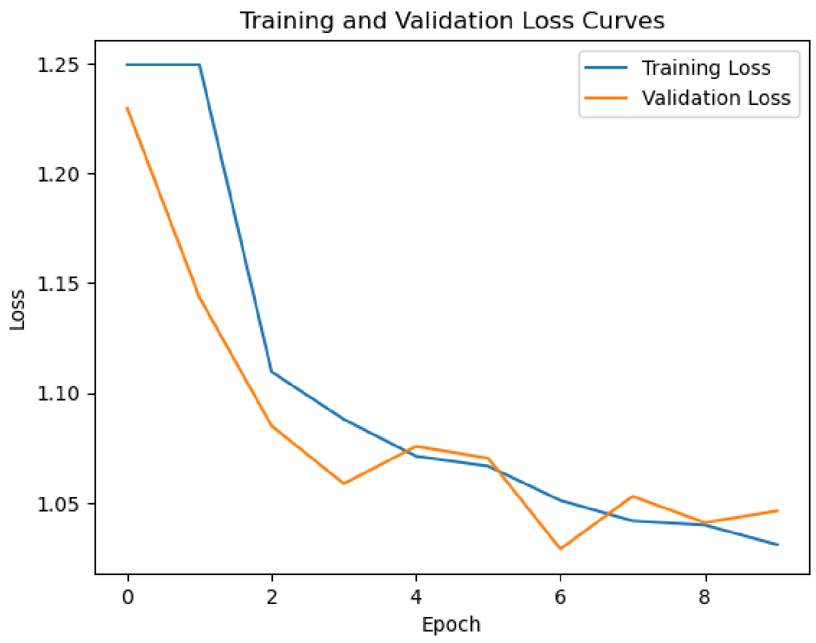
```python
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item()

                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        val_accuracy = 100 * correct / total
        print(f"Epoch [{epoch+1}/{num_epochs}], "
              f"Training Loss: {running_loss/len(train_loader):.4f}, "
              f"Validation Loss: {val_loss/len(val_loader):.4f}, "
              f"Validation Accuracy: {val_accuracy:.2f}%")
```

```python
# Main execution
num_classes = len(dataset.classes)
model = BadmintonCNN(num_classes)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

train_model(model, train_loader, val_loader, criterion, optimizer, num_epochs=10, device=device)
```

```
Epoch [1/10], Training Loss: 1.2494, Validation Loss: 1.2295, Validation Accuracy: 39.85%
Epoch [2/10], Training Loss: 1.1512, Validation Loss: 1.1435, Validation Accuracy: 46.62%
Epoch [3/10], Training Loss: 1.1096, Validation Loss: 1.0850, Validation Accuracy: 49.62%
Epoch [4/10], Training Loss: 1.0880, Validation Loss: 1.0587, Validation Accuracy: 49.62%
Epoch [5/10], Training Loss: 1.0712, Validation Loss: 1.0757, Validation Accuracy: 49.62%
Epoch [6/10], Training Loss: 1.0667, Validation Loss: 1.0703, Validation Accuracy: 50.38%
Epoch [7/10], Training Loss: 1.0511, Validation Loss: 1.0290, Validation Accuracy: 49.62%
Epoch [8/10], Training Loss: 1.0418, Validation Loss: 1.0530, Validation Accuracy: 49.62%
Epoch [9/10], Training Loss: 1.0399, Validation Loss: 1.0410, Validation Accuracy: 49.62%
Epoch [10/10], Training Loss: 1.0310, Validation Loss: 1.0463, Validation Accuracy: 48.87%
```

```python
import torch
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, precision_score,
recall_score, f1_score
```

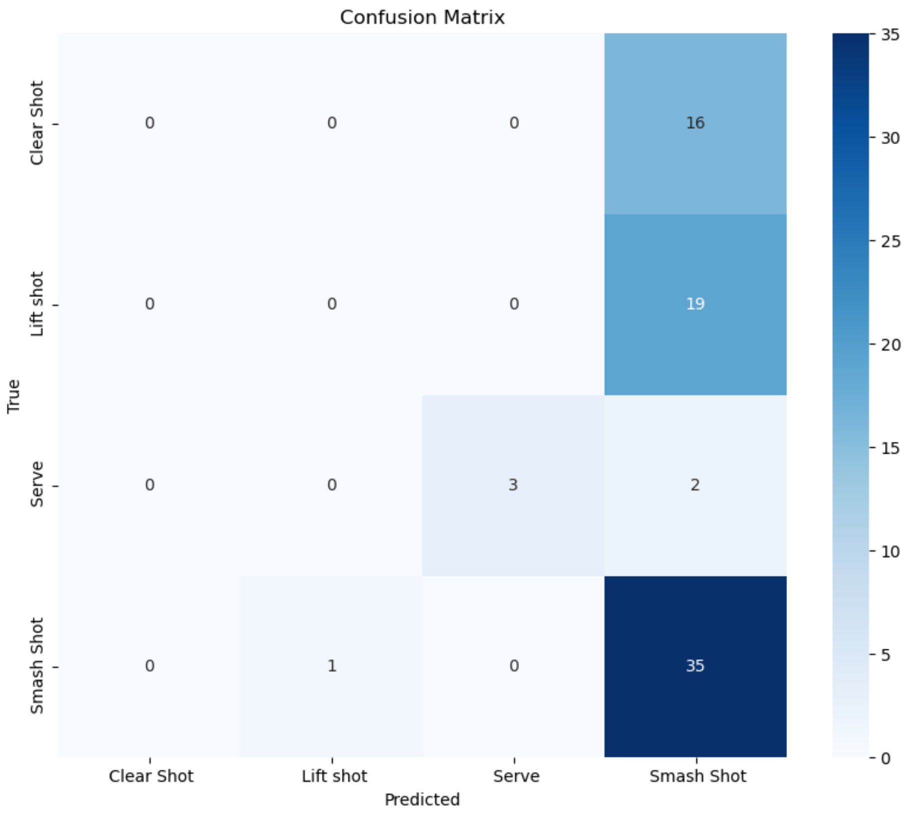
```python
# Plot the losses after training
train_losses = [1.2494, 1.2494, 1.1096, 1.0880, 1.0712, 1.0667, 1.0511, 1.0418, 1.0399, 1.0310]
val_losses = [1.2295, 1.1435, 1.0850, 1.0587, 1.0757, 1.0703, 1.0290, 1.0530, 1.0410, 1.0463]

plt.plot(train_losses, label='Training Loss')
plt.plot(val_losses, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss Curves')
plt.legend()
plt.show()
```



```python
def evaluate_model(model, test_loader, device):
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, preds = torch.max(outputs, 1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    return np.array(all_preds), np.array(all_labels)

def compute_metrics(y_true, y_pred, classes):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted', zero_division=1)
    recall = recall_score(y_true, y_pred, average='weighted', zero_division=1)
    f1 = f1_score(y_true, y_pred, average='weighted', zero_division=1)

    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")

    print("\nClassification Report:")
    print(classification_report(y_true, y_pred, target_names=classes, zero_division=1))

def plot_confusion_matrix(y_true, y_pred, classes):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

def display_sample_predictions(model, test_loader, device, classes, num_samples=5):
    model.eval()
    fig, axes = plt.subplots(num_samples, 2, figsize=(12, 4*num_samples))

    with torch.no_grad():
        for i, (images, labels) in enumerate(test_loader):
            if i >= num_samples:
                break

            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, preds = torch.max(outputs, 1)

            for j in range(2):
                img = images[j].cpu().permute(1, 2, 0)
                img = img * torch.tensor([0.229, 0.224, 0.225]) + torch.tensor([0.485, 0.456, 0.406])
                img = img.clip(0, 1)

                axes[i, j].imshow(img)
                axes[i, j].set_title(f"True: {classes[labels[j]]}\nPred: {classes[preds[j]]}")
                axes[i, j].axis('off')

    plt.tight_layout()
    plt.show()
```

```python
# Evaluation
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
model.eval()

y_pred, y_true = evaluate_model(model, test_loader, device)

# Compute and display metrics, plot confusion matrix, display samples
compute_metrics(y_true, y_pred, dataset.classes)
plot_confusion_matrix(y_true, y_pred, dataset.classes)
display_sample_predictions(model, test_loader, device, dataset.classes)
```

```
Accuracy: 0.5000
Precision: 0.5066
Recall: 0.5000
F1 Score: 0.6064

Classification Report:
              precision    recall  f1-score   support

  Clear Shot       1.00      0.00      0.00        16
   Lift shot       0.00      0.00      1.00        19
       Serve       1.00      0.60      0.75         5
  Smash Shot       0.49      0.97      0.65        36

    accuracy                           0.50        76
   macro avg       0.62      0.39      0.60        76
weighted avg       0.51      0.50      0.61        76
```


Confusion Matrix
True
Predicted
Clear Shot
Lift shot
Serve
Smash Shot
0
0
0
16
0
0
0
19
0
0
3
2
0
1
0
35
Clear Shot
Lift shot
Serve
Smash Shot
35
30
25
20
15
10
5
0

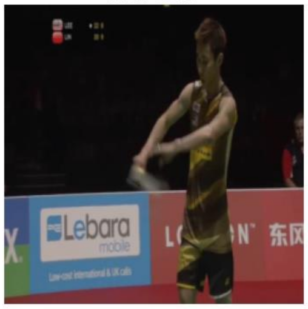
True: Lift shot
Pred: Smash Shot

True: Clear Shot
Pred: Smash Shot

True: Clear Shot
Pred: Smash Shot

True: Clear Shot
Pred: Smash Shot

True: Serve
Pred: Serve



True: Smash Shot
Pred: Smash Shot

True: Lift shot
Pred: Smash Shot

True: Clear Shot
Pred: Smash Shot

True: Clear Shot
Pred: Smash Shot

True: Smash Shot
Pred: Smash Shot

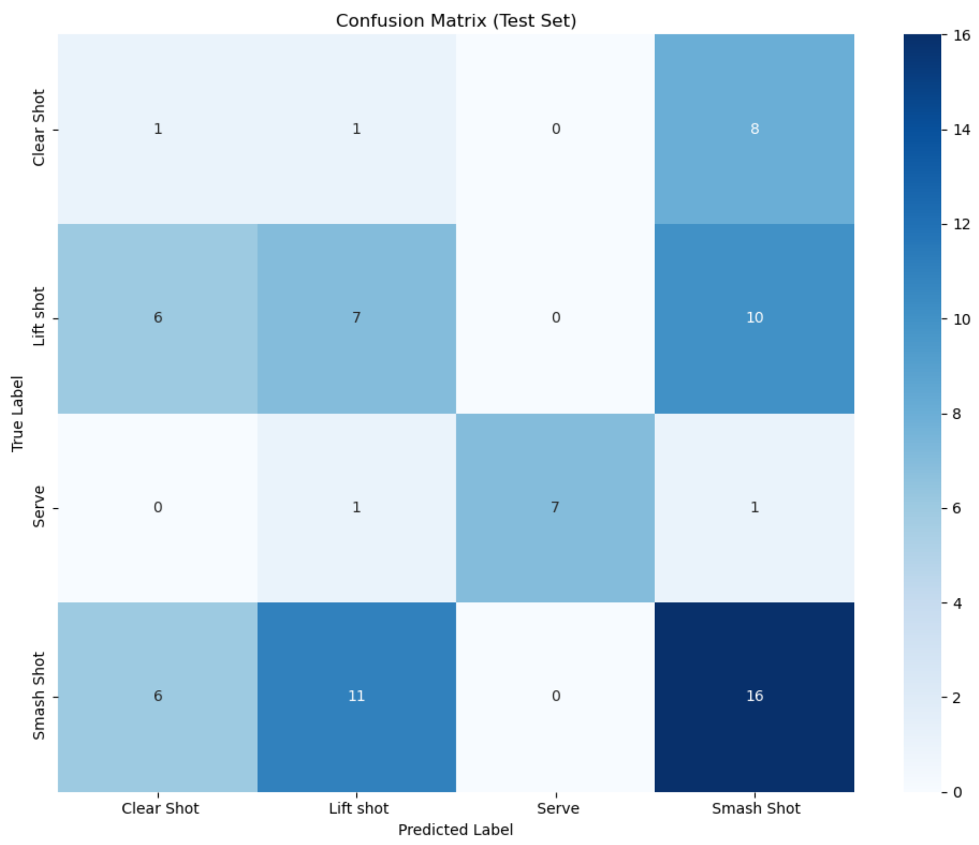
Multi-Layer Perceptron

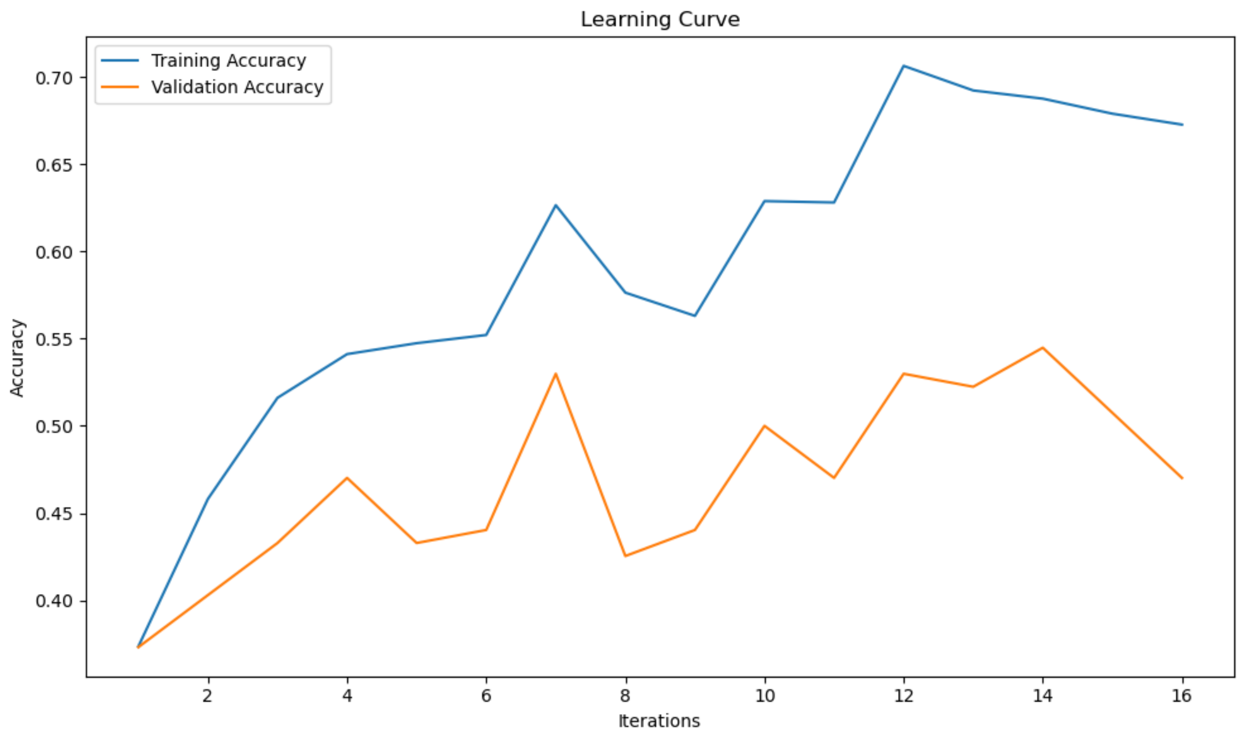
```python
import pandas as pd
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
import os
import warnings

# Filter out convergence warnings
warnings.filterwarnings("ignore", category = UserWarning)

def load_and_preprocess_data(csv_path, image_folder):
    # Load CSV
    df = pd.read_csv(csv_path)

    # Initialize lists to store image data and labels
    X = []
    y = []

    # Define shot types
    shot_types = [' Clear Shot', ' Lift shot', ' Serve', ' Smash Shot']

    # Process each image and create labels
    for _, row in df.iterrows():
        # Load and preprocess image
        img_path = os.path.join(image_folder, row['filename'])
        img = Image.open(img_path)
        img = img.resize((64, 64))  # Resize to a manageable size
        img_array = np.array(img).flatten() / 255.0  # Flatten and normalize
        X.append(img_array)

        # Create label
        label = shot_types[[row[shot] for shot in shot_types].index(1)]
        y.append(label)

    X = np.array(X)
    y = np.array(y)

    return X, y

def create_and_train_model(X_train, X_val, X_test, y_train, y_val, y_test):
    # Create and train model
    mlp = MLPClassifier(
        hidden_layer_sizes=(256, 128, 64),
        activation='relu',
        solver='adam',
        max_iter=1000,
        random_state=42,
        early_stopping=True,
        validation_fraction=0.1,
        n_iter_no_change=10
    )

    mlp.fit(X_train, y_train)

    return mlp

def evaluate_model(model, X_train, X_val, X_test, y_train, y_val, y_test):
    # Make predictions
    y_train_pred = model.predict(X_train)
    y_val_pred = model.predict(X_val)
    y_test_pred = model.predict(X_test)

    # Print classification reports
    print("Training Set Performance:")
    print(classification_report(y_train, y_train_pred))
    print("\nValidation Set Performance:")
    print(classification_report(y_val, y_val_pred))
    print("\nTest Set Performance:")
    print(classification_report(y_test, y_test_pred))

    # Plot confusion matrix for test set
    cm = confusion_matrix(y_test, y_test_pred)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=np.unique(y_test),
                yticklabels=np.unique(y_test))
    plt.title('Confusion Matrix (Test Set)')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.tight_layout()
    plt.show()

def plot_learning_curve(model, X_train, y_train, X_val, y_val):
    train_scores = []
    val_scores = []

    for i in range(1, len(model.loss_curve_) + 1):
        model_partial = MLPClassifier(
            hidden_layer_sizes=model.hidden_layer_sizes,
            activation=model.activation,
            solver=model.solver,
            max_iter=i,
            random_state=42
        )
        model_partial.fit(X_train, y_train)

        train_scores.append(model_partial.score(X_train, y_train))
        val_scores.append(model_partial.score(X_val, y_val))

    plt.figure(figsize=(10, 6))
    plt.plot(range(1, len(model.loss_curve_) + 1), train_scores, label='Training Accuracy')
    plt.plot(range(1, len(model.loss_curve_) + 1), val_scores, label='Validation Accuracy')
    plt.xlabel('Iterations')
    plt.ylabel('Accuracy')
    plt.title('Learning Curve')
    plt.legend()
    plt.tight_layout()
    plt.show()
```

```
# Set paths
csv_path = 'filtered_classes.csv'
image_folder = 'images'

# Load and preprocess data
X, y = load_and_preprocess_data(csv_path, image_folder)

# Split data into train, validation, and test sets
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.05, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.0947,
random_state=42)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# Create and train model
model = create_and_train_model(X_train_scaled, X_val_scaled, X_test_scaled,
                              y_train, y_val, y_test)

# Evaluate model
evaluate_model(model, X_train_scaled, X_val_scaled, X_test_scaled,
              y_train, y_val, y_test)

# Plot learning curve
plot_learning_curve(model, X_train_scaled, y_train, X_val_scaled, y_val)
```

```
Training Set Performance:
              precision    recall  f1-score   support
```

```
  Clear Shot       0.41      0.30      0.35       225
   Lift shot       0.53      0.56      0.55       373
       Serve       0.94      0.88      0.91       116
  Smash Shot       0.63      0.69      0.66       563

    accuracy                           0.60      1277
   macro avg       0.63      0.61      0.62      1277
weighted avg       0.59      0.60      0.59      1277


Validation Set Performance:
              precision    recall  f1-score   support

  Clear Shot       0.16      0.12      0.14        25
   Lift shot       0.43      0.49      0.46        37
       Serve       0.89      0.80      0.84        10
  Smash Shot       0.56      0.58      0.57        62

    accuracy                           0.49       134
   macro avg       0.51      0.50      0.50       134
weighted avg       0.47      0.49      0.48       134


Test Set Performance:
              precision    recall  f1-score   support

  Clear Shot       0.08      0.10      0.09        10
   Lift shot       0.35      0.30      0.33        23
       Serve       1.00      0.78      0.88         9
  Smash Shot       0.46      0.48      0.47        33

    accuracy                           0.41        75
   macro avg       0.47      0.42      0.44        75
weighted avg       0.44      0.41      0.42        75
```


Confusion Matrix (Test Set)
True Label
Predicted Label
Clear Shot
Lift shot
Serve
Smash Shot
1 1 0 8
6 7 0 10
0 1 7 1
6 11 0 16
0 2 4 6 8 10 12 14 16


Learning Curve
Training Accuracy
Validation Accuracy
Accuracy
Iterations
0.40 0.45 0.50 0.55 0.60 0.65 0.70
2 4 6 8 10 12 14 16