A Machine Learning Approach to Classifying Billiards Balls

September 20, 2024

1 Introduction

2 Problem Formulation

Our dataset of billiards balls contains 960 high-quality 3552x3552 pixel, 24-bit RGB images of billiards balls, 60 for each of the 16 different categories. Each image only depicts one ball. The data is therefore categorical with 16 different classes, one representing each type of ball.

We have cropped and resized the images to 64 by 64 pixels. We have also scaled the color 8-bit color channels, whose values range from 0 to 256, to floating-point values between 0 and 1. After preprocessing the image dataset, each image will constitute a 3×4096 feature matrix of floating-point values. A machine learning model, specifically SVC, is trained using this labeled dataset, constituting a supervised machine learning task, with the goal of accurately predicting the labels of input images.

3 Method

A large literature of image classification projects exist and we have chosen to largely follow the method described in [1]. We have also used tensorflow's computer vision tutorial extensively in the dataset loading, preprocessing and implementation of the model.[2]

3.1 Dataset

The dataset of billiards ball images was created specifically for the goal of predicting the correct labels of billiards balls at our table location. We took 60 photos of each of the 16 balls from various angles, resulting in 960 total images. The images taken were 3552 by 3552 pixel jpg images with a bit depth of 24. The dataset was split into training, validation and test datasets. The final training dataset had 720 images, validation dataset 80 images and test dataset 160 images, corresponding roughly to a 75-8-17 split. We chose this split to have most of the data be used in training, while setting aside separate datasets for validation to prevent overfitting, and testing to evaluate the final performance of the model. The reasoning for a commonly used 75-10-15 split is explained in [3]. The training, validation and test datasets were randomly separated using scikit-lean's train_test_split-function.

3.2 Data Preprocessing and Feature Selection

Our initial dataset consists of unnecessarily large RGB images. Our preprocessing has three steps: Cropping the images, resizing them to 64x64 pixels and scaling the 8-bit integer RGB values down to a single floating-point number between 0 and 1. This is done to reduce the size of inputs to the model from an integer matrix of size $3 \times (3552^2)$, to a more manageable 3×4096 floating-point values. The color value scaling was done because machine learning methods generally work better with small input values. [4]. Now each feature is a 3 by 4096 matrix of floating-point numbers between 0 and 1. The dimensions of the matrix represent the red, green and blue color channels of the image. The features are labeled with a number between 0 and 15, representing the ground truth, which is the number of the billiards ball in the image. The cue ball is number 0.

3.3 Data Augmentation

Since our initial dataset is somewhat small, we have extended it using dataset augmentation to increase the quantity and diversity of the training set. We have augmented the training dataset by applying random rotations, image flips and brightness adjustments to the preprocessed training dataset images. The augmentation was performed using keras' layers utility.

3.4 Model

For this task of recognizing billiard balls from images, we have selected the Support Vector Classifier (SVC) as our initial ML model. SVC is ideal for handling high-dimensional data, like images, by finding a hyperplane that maximises the margin between different classes. In this case, the classes are different billiard balls. SVC works by identifying the optimal weight vector and bias term to separate these classes. If the data isn't linearly separable, kernel functions such as the Radial Basis Function (RBF) can be used to map the data to a higher-dimensional space, allowing more accurate classification. [1], [5]

$$h(x) = sign(wx + b) \tag{1}$$

In this case, $h(x)$ assigns a class label to the input data point x, returning $+1$ when $wx + b$ is greater than or equal to 0, and -1 when it is less than 0.

SVC was chosen because of its ability to efficiently manage image-based tasks with numerous pixel features, and its effectiveness in separating classes through margin maximization. It's also robust against overfitting and performs well with smaller datasets like the couple thousand images used in this project, making it a practical and reliable choice. [1], [5]

3.5 Loss Function

For this project, we have selected the hinge loss function as our initial loss function, which is commonly used with Support Vector Machines (SVMs). Hinge loss encourages the model to maximize the margin between different classes, penalizing predictions that fall too close to the decision boundary or are misclassified. This ensures that the SVC model not only classifies the billiard balls correctly but does so with confidence, reducing the likelihood of errors. [1], [6]

$$L(y, f(x)) = max(0, 1 - y \cdot f(x)) \tag{2}$$

Here, y denotes the actual class label (+1 or -1), while $f(x)$ refers to the classifier's decision function for the given data point x.

Hinge loss is computationally efficient and aligns perfectly with SVC's margin-maximizing objective, making it an ideal choice. By penalizing near-boundary predictions, hinge loss ensures that the model maintains clear and accurate boundaries between billiard ball classes, improving overall classification performance. [1], [6]

4 Aknowledgements

Use of generative AI

Parts of this text were generated using ChatGPT, but all prompts were crafted and texts edited and reviewed by students. Additionally, ChatGPT was used for assistance in data processing and coding.

References

[1] unknown, "A Machine Learning Approach to Classifying Bangla Handwritten Characters," Tech. Rep., Sep. 2023.

[2] *Computer vision with TensorFlow — TensorFlow Core*, `https://www.tensorflow.org/tutorials/images`. Accessed: Sep. 20, 2024.

[3] V. R. Joseph, "Optimal Ratio for Data Splitting," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 15, no. 4, pp. 531–538, Aug. 2022, ISSN: 1932-1864, 1932-1872. DOI: `10.1002/sam.11583`. arXiv: `2202.03326 [cs, stat]`. Accessed: Sep. 19, 2024.

[4] *Importance of Feature Scaling*, `https://scikit-learn/stable/auto\_examples/preprocessing/plot\_scaling\_importance.html`. Accessed: Sep. 20, 2024.

[5] W. S. Noble, "What is a support vector machine?" *Nature Biotechnology*, vol. 24, no. 12, pp. 1565–1567, Dec. 2006, ISSN: 1087-0156, 1546-1696. DOI: `10.1038/nbt1206-1565`. Accessed: Sep. 20, 2024.

[6] P. L. Bartlett and M. H. Wegkamp, "Classification with a Reject Option using a Hinge Loss," Aug. 2008.

A preprocessing.pdf

preprocessing

September 20, 2024

1 Crop and resize images

The images are 3552x3552 pixels and contain a lot of empty space at the edges. Here we crop the images to 2048x2048 toward the center. Then they are resized to 64x64.

[ ]:
```
from PIL import Image
import os
from pprint import pprint

dirs = [ (f'.\\data\\raw_data\\{num}', f'.\\data\\processed_64\\{num}') for num␣
↪in range(16) ]
# dirs = [("./data/sanity_check_raw", "./data/sanity_check")]

pprint(dirs)
for (in_dir, out_dir) in dirs:
    os.makedirs(out_dir, exist_ok=True)

original_size = 3552
crop_size = 2048
target_size = 64
max_files = 60 # We want the same number of images of each ball
```

[ ]:
```
left = (original_size - crop_size) // 2
top = (original_size - crop_size) // 2
right = (original_size + crop_size) // 2
bottom = (original_size + crop_size) // 2

for (in_dir, out_dir) in dirs:
    files = os.listdir(in_dir)
    print(f"Processing {in_dir} ...")

    counter = 0
    for filename in files[:max_files]:
        if filename.lower().endswith(".jpg"):
            counter += 1
            # Open the image
            img_path = os.path.join(in_dir, filename)
            output_path = os.path.join(out_dir, filename)
            Image.open(img_path
                ).crop((left, top, right, bottom)
                ).resize((target_size, target_size), Image.Resampling.LANCZOS
                ).save(output_path)

            # print(f"Cropped and saved: {output_path} ({counter} of␣
 ↪{max_files})")
```

2 Split the data into training, validation and testing datasets

We have 960 images (60 of each class). We will split these into

720 training images (75 %)
80 validation images (8.33 %)
160 testing images (16.67 %)

[20]:
```
import numpy as np
import keras
from sklearn.model_selection import train_test_split
import tensorflow as tf

data_dir = './data/processed_64'
batch_size = 25
img_width = 64
img_height = 64

dataset = keras.utils.image_dataset_from_directory(
    data_dir,
    image_size=(img_height, img_width),
    batch_size=batch_size,
    label_mode='int'
)

class_names = dataset.class_names # ['0', '1', '10', '11', '12', '13', '14',␣
 ↪'15', '2', '3', '4', '5', '6', '7', '8', '9']

image_batches = []
label_batches = []

for images, labels in dataset:
    image_batches.append(images)
    label_batches.append(labels)

X = np.concatenate(image_batches)
y = np.concatenate(label_batches)
```

```
print("Splitting data into training, validation and testing datasets...")

X_train, X_rest, y_train, y_rest = train_test_split(X, y, test_size = 0.25,␣
 ↪random_state = 0)

X_val, X_test, y_val, y_test = train_test_split(X_rest, y_rest, test_size = (2/
 ↪3), random_state = 0)

print("X_train", X_train.shape)
print("y_train", y_train.shape)
print("X_val", X_val.shape)
print("y_val", y_val.shape)
print("X_test", X_test.shape)
print("y_test", y_test.shape)

train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train))
val_ds = tf.data.Dataset.from_tensor_slices((X_val, y_val))
test_ds = tf.data.Dataset.from_tensor_slices((X_test, y_test))
```

```
Found 960 files belonging to 16 classes.
Splitting data into training, validation and testing datasets…
X_train (720, 64, 64, 3)
y_train (720,)
X_val (80, 64, 64, 3)
y_val (80,)
X_test (160, 64, 64, 3)
y_test (160,)
```

3 Data augmentation

We can use `keras` layers utility to apply random flips, rotations and brightness adjustment to prepare the training dataset.

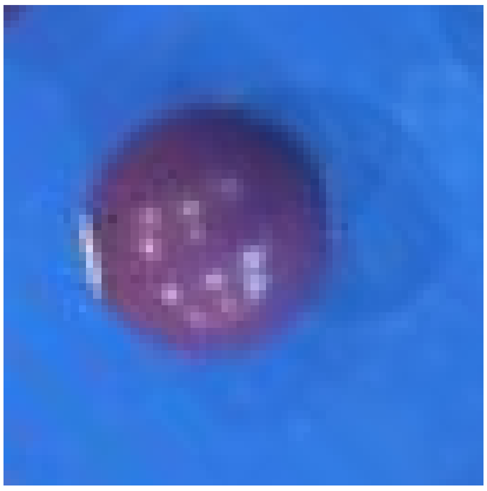
[21]:
```
from keras import layers
import tensorflow as tf
import numpy as np

normalization_layer = keras.layers.Rescaling(1./255) # Rescale RGB values from␣
 ↪0..255 to floats in 0..1

augmentation_layer = keras.Sequential([
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.3),
    layers.RandomBrightness(0.1)
])

batch_size = 50
```

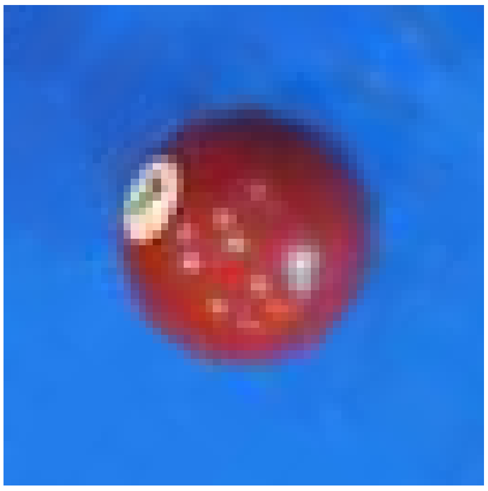
```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.map(
    lambda x, y: (augmentation_layer(x), y)).map(
    lambda x, y: (normalization_layer(x), y)
    ).shuffle(buffer_size=len(X_train), seed=0, reshuffle_each_iteration=True
    ).batch(batch_size
    ).prefetch(buffer_size=AUTOTUNE)

val_ds = val_ds.map(
    lambda x, y: (normalization_layer(x), y)
    ).batch(batch_size
    ).prefetch(buffer_size=AUTOTUNE)

test_ds = test_ds.map(
    lambda x, y: (normalization_layer(x), y)
    ).batch(batch_size
    ).prefetch(buffer_size=AUTOTUNE)
```

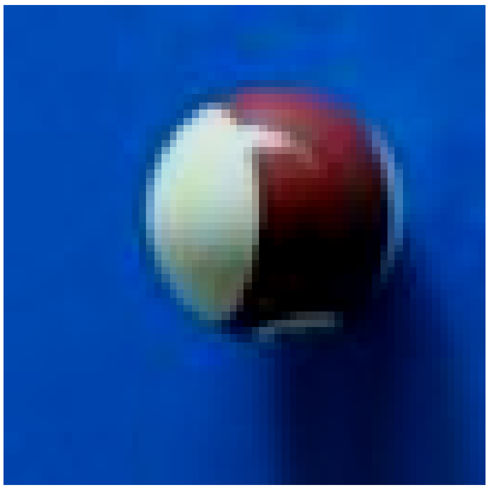
4 Visualize the training dataset images

Here we can see that the images have different brightness levels and rotations

[26]:
```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Select an image from X_train, e.g., the first image
images, labels = next(iter(train_ds.take(1)))

# Create a 2x2 grid of subplots
fig, axs = plt.subplots(2, 2, figsize=(8, 8))

# Display each image
axs[0, 0].imshow(images[0])
axs[0, 1].imshow(images[1])
axs[1, 0].imshow(images[2])
axs[1, 1].imshow(images[3])

# Turn off axes for all subplots
for ax in axs.flat:
    ax.axis('off')

plt.show()
```







```
[ ]:
```