Classifying Counter-Strike maps from images using Machine Learning

September 2024

1 Introduction

Counter-Strike 2 is one of the most popular first-person shooter video games. Counter-Strike is played online in two teams of five players. Matches are played on different maps, all of which have a different look and feel. In this project we attempt to train a machine learning model to accurately recognize Counter-Strike 2 maps from images of them.

Although our project doesn't have many real-life application domains, one could be to help new players learn to recognize each map.

In this report the process of training a machine learning model to recognize a map from an image of it is described. Section 2 discusses the formulation of the problem. Section 3 describes the dataset, and the creation of it. Section 4 describes the two machine learning methods to solve this problem and the reasons behind choosing them. Section 5 discusses the results of this project as well as which of these methods we chose as the final method. Section 6 then gives a conclusion of the entire project.

2 Problem Formulation

Our goal is to recognize a map from the current active-duty map pool from an image of the map. This map pool consists of seven different maps: Mirage, Vertigo, Nuke, Ancient, Anubis, Inferno and Dust 2. Each of them is unique and has some recognizable features. As we are trying to recognize these maps from images, this problem is a classification problem.

3 Dataset

3.1 Creation of the dataset

To train the model, we needed a dataset consisting of images from all the maps that we wanted the model to recognize. As there was no suitable dataset to be found online, we created our own dataset.

Using a recording software, we recorded video in-game in each of the seven maps, covering all major areas. Images were extracted from the video with VLC media player. Then, depending on the lengths of the recorded videos we took every fifth or third frame of the videos so that there was around a thousand images of each map to make the dataset balanced.

The images were shuffled before splitting them into training, validation and testing sets. We used 70% of the images as the training set, 15% of the images as the validation set and the remaining 15% of the images as the test set. Similar split ratio is commonly used in basic machine learning applications and in our testing, it produced the most accurate results, so we decided to use it.

3.2.1 Pre-processing of the dataset for CNN

For our two models the preprocessing of the dataset was slightly different. For CNN, we downscaled the images to a lower resolution so that the processing of the photos was faster. The final resolution was 64x64. Then with Keras, we created two NumPy arrays. One of them is a one-dimensional NumPy array that has all the labels of the photos represented with integers. The second NumPy array will be a multidimensional array with a shape of (batch size, horizontal size, vertical size, color channels). So, for example with a thousand 64 by 64 RGB-photos the array's shape would be (1000, 64, 64, 3). We also normalize the values so that the maximum brightness for a pixel is 1.0 and minimum is 0.0 so that the processing would be faster.

To put simply, in the final vector each subpixel of a photo represents a single datapoint. For example, when we used 64 by 64 images to train the model, there were 12288 datapoints in total for one image.

3.2.2 Pre-processing of the dataset for SVC

For SVC the preprocessing for the photos was a little different. Firstly, we downscaled the images to a resolution of 32 by 32 after which we transformed them into one-dimensional NumPy arrays which house all of the brightnesses of the subpixels. In our case the length of the array and the amount of the datapoints would be 32 x 32 x 3 = 3072. Then we again normalized the values of the array and added them into a data-array that houses all the processed photos.

3.3 Labels

Both of our selected models are supervised learning models, which meant that we had to input the labels into the model. We decided to use the names of the Counter-Strike 2 maps as the labels.

The photos were divided into seven different folders by the map they were taken from, and the name of the folder was the name of the maps inside. That way we could use the names of the folders as labels.

4 Methods

4.1 Convolutional Neural Network (CNN)

The first machine learning method we applied to this problem is convolutional neural network (CNN). We chose CNN, as it is the standard deep learning approach to computer vision problems. The best algorithms for image recognition problems are currently based on convolutional networks [1]. As a loss function, we have chosen categorical cross-entropy. It is used for measuring the performance of a multi-class classification model. This loss function is widely used for multi-class classification tasks, which makes it a safe choice for our model.

In CNN, different layers are used to extract features from the images. We used convolution and max pooling layers, as well as some dense layers on top in a similar way as the TensorFlow tutorial for image recognition does [3]. As a result, a feature vector that is a combination of the most important learned features from all the layers is extracted from the data. This feature vector was used to train the model to classify the images.

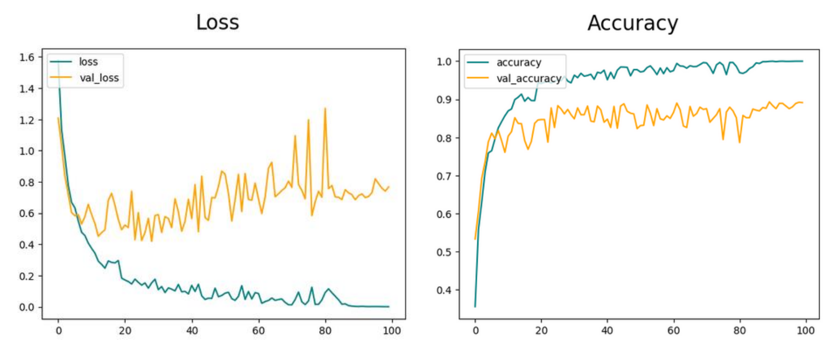


Figure 1: Training and validation loss and accuracy in relation to the amount of epochs

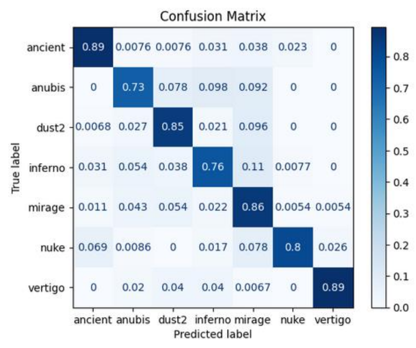
In CNN an epoch is a full training cycle through the entire training dataset, where the weights are updated based on the validation set. We decided to use 25 epochs, because as seen in figure x, the validation accuracy did not improve noticeably after this point and the validation loss starts increasing.

4.2 Support Vector Classifier (SVC)

The second machine learning method we applied to our problem is supervised learning algorithm support vector classifier (SVC). We chose this method since it's very effective with high-dimensional arrays, which we need in our project to classify images. As an SVC kernel we chose to use radial basis function kernel (RBF) due to it giving us the best accuracy among other kernels. As a loss function, we used Hinge Loss in our application. Hinge Loss can effectively maximize the distance between different classes, which helps us in our application because we wanted results that fit clearly in one class, not in between.

5. Results

For CNN, the accuracy is 83% and error is 17% for both the training and validation datasets. The training accuracy of the model is 93%, which indicates there might be some overfitting. Categorical cross-entropy testing loss was 0.60, which is quite high and means there is still room for improvement. These numbers differ slightly in each run. Figure 1 shows a confusion matrix for the testing dataset.



	ancient	anubis	dust2	inferno	mirage	nuke	vertigo
ancient	0.89	0.0076	0.0076	0.031	0.038	0.023	0
anubis	0	0.73	0.078	0.098	0.092	0	0
dust2	0.0068	0.027	0.85	0.021	0.096	0	0
inferno	0.031	0.054	0.038	0.76	0.11	0.0077	0
mirage	0.011	0.043	0.054	0.022	0.86	0.0054	0.0054
nuke	0.069	0.0086	0	0.017	0.078	0.8	0.026
vertigo	0	0.02	0.04	0.04	0.0067	0	0.89

Figure 2: Confusion matrix for the testing dataset with CNN

For our SVC model, the training accuracy is 66% and the training error is 34% with a validation accuracy of 68% with an error of around 32% which is significantly lower than our CNN model. The confusion matrix for our SVC testing dataset is shown in figure 3. The hinge loss was about 1.04 which is very high. It is worth noting that even though SVC uses lower resolution images, it did not perform noticeably better with the 64x64 resolution used with CNN and larger photos would require a lot more processing power than we have for this project.

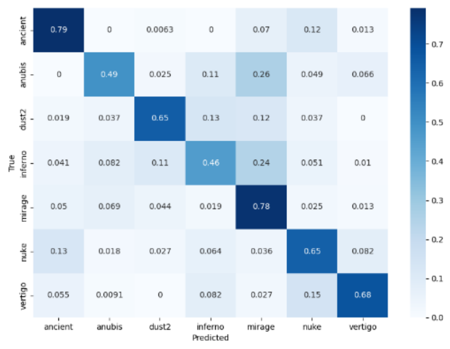


Figure 3: Confusion matrix for the testing dataset for SVC

Based on these results, CNN has superior accuracy and lower error in both validation and testing sets compared to SVC. This is why we chose CNN as our final machine learning method.

6. Conclusion

In our project we tested two different machine learning methods, CNN and SVC, for classifying Counter-Strike 2 maps by images taken from them. We first trained both of them with a dataset that we created with images taken from different CS2 maps and then tested how well the algorithms performed. We came to the conclusion that CNN performs better than SVC in classifying images. CNN had lower training and validation error and higher validation and testing accuracy than SVC.

We optimized both algorithms to the best of our knowledge but there could be some improvement to be made with CNN model with the current dataset as our accuracy was under 90% with a quite high loss. Also, with SVC our accuracy was only about 65% with an even larger loss thus we think that it could be improved slightly by for example increasing the resolution of the photos. Also both models would benefit from increasing the size of the training dataset, especially CNN performs better on large datasets.

We used AI tool ChatGPT to provide assistance in training our machine learning models as well as preprocessing the dataset.

References

[1] **Wikipedia contributors.** (n.d.). *Computer vision: Recognition*. Wikipedia. Retrieved September 20, 2024, from https://en.wikipedia.org/wiki/Computer_vision#Recognition

[2] **Peyrone, N.** (n.d.) *Comparing SVM and CNN in Recognizing Handwritten Digits: An Overview.* Medium. Retrieved September 20, 2024, from https://peyrone.medium.com/comparing-svm-and-cnn-in-recognizing-handwritten-digits-an-overview-5ef06b20194e

[3] **TensorFlow.** (n.d.). *Convolutional Neural Network (CNN)*. TensorFlow. Retrieved September 18, 2024, from https://www.tensorflow.org/tutorials/images/cnn

Appendix

October 9, 2024

1 Code for training CNN

[ ]:
```python
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import os
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import numpy as np
```

[ ]:
```python
gpus = tf.config.list_physical_devices('GPU')

for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)

data = tf.keras.utils.image_dataset_from_directory('maps')
data_iterator = data.as_numpy_iterator()

batch = data_iterator.next()
fig, ax = plt.subplots(ncols=7, figsize=(20,20))
for idx, img in enumerate(batch[0][:7]):
    ax[idx].imshow(img.astype(int))
    ax[idx].title.set_text(batch[1][idx])
```

[ ]:
```python
dataset_path = 'maps'
img_height = 64
img_width = 64
batch_size = 32

full_dataset = tf.keras.utils.image_dataset_from_directory(
    dataset_path,
    image_size=(img_height, img_width),
    batch_size=batch_size
)

total_samples = len(full_dataset) * batch_size
train_size = int(0.7 * total_samples)
val_size = int(0.15 * total_samples)
```

```
data_iterator = full_dataset.as_numpy_iterator()
all_images = []
all_labels = []

for images, labels in data_iterator:
    all_images.append(images)
    all_labels.append(labels)

all_images = np.concatenate(all_images)
all_labels = np.concatenate(all_labels)

x_train = all_images[:train_size]
y_train = all_labels[:train_size]
x_val = all_images[train_size:train_size + val_size]
y_val = all_labels[train_size:train_size + val_size]
x_test = all_images[train_size + val_size:]
y_test = all_labels[train_size + val_size:]

train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train)).
 ↪batch(batch_size)
val_dataset = tf.data.Dataset.from_tensor_slices((x_val, y_val)).
 ↪batch(batch_size)
test_dataset = tf.data.Dataset.from_tensor_slices((x_test, y_test)).
 ↪batch(batch_size)
```

[ ]:
```
class_names = full_dataset.class_names
print("Classes :", class_names)
```

[ ]:
```
model = models.Sequential([
    layers.Input(shape=(img_height, img_width, 3)),
    layers.Rescaling(1./255),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(len(class_names))
])
```

```
[ ]: model.compile(optimizer='adam',
                   loss=tf.keras.losses.
      ↪SparseCategoricalCrossentropy(from_logits=True),
                   metrics=['accuracy'])
```

```
[ ]: history = model.fit(train_dataset, validation_data=val_dataset, epochs=25)
```

```
[ ]: fig = plt.figure()
     plt.plot(history.history['loss'], color='teal', label='loss')
     plt.plot(history.history['val_loss'], color='orange', label='val_loss')
     fig.suptitle('Loss', fontsize=20)
     plt.legend(loc="upper left")
     plt.show()

     fig = plt.figure()
     plt.plot(history.history['accuracy'], color='teal', label='accuracy')
     plt.plot(history.history['val_accuracy'], color='orange', label='val_accuracy')
     fig.suptitle('Accuracy', fontsize=20)
     plt.legend(loc="upper left")
     plt.show()

     train_loss, train_acc = model.evaluate(train_dataset)
     val_loss, val_acc = model.evaluate(val_dataset)
     test_loss, test_acc = model.evaluate(test_dataset)

     print("Train accuracy:", train_acc, "Train loss:", train_loss)
     print("Validation accuracy:", val_acc, "Validation loss:", val_loss)
     print("Test accuracy:", test_acc, "Test loss:", test_loss)
```

```
[ ]: y_true = []
     y_pred = []

     for images, labels in test_dataset:
         preds = model.predict(images)
         y_true.extend(labels.numpy())
         y_pred.extend(np.argmax(preds, axis=1))

     cm = confusion_matrix(y_true, y_pred, normalize='true')

     plt.figure(figsize=(10, 8))
     disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
     disp.plot(cmap=plt.cm.Blues)
     plt.title('Confusion Matrix')
     plt.show()
```

2 Code for training SVC

[ ]:
```
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, hinge_loss
import matplotlib.pyplot as plt
import seaborn as sns
```

[ ]:
```
dataset_path = 'maps'
image_size = (32, 32)
data = []
labels = []
```

[ ]:
```
for folder in os.listdir(dataset_path):
    folder_path = os.path.join(dataset_path, folder)
    if os.path.isdir(folder_path):
        for file in os.listdir(folder_path):
            file_path = os.path.join(folder_path, file)
            image = cv2.imread(file_path)
            if image is not None:
                image = cv2.resize(image, image_size)
                data.append(image.flatten() / 255.0)
                labels.append(folder)
```

[ ]:
```
data = np.array(data)
labels = np.array(labels)
le = LabelEncoder()
labels = le.fit_transform(labels)
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.
 ↪3, random_state=42)
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.5,␣
 ↪random_state=42)
clf = SVC(kernel='rbf')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

[ ]:
```
cm = confusion_matrix(y_test, y_pred, normalize='true')

plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, cmap='Blues', xticklabels=le.classes_,␣
 ↪yticklabels=le.classes_)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
[ ]: accuracy = accuracy_score(y_test, y_pred)
     print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
[ ]: decision_function = clf.decision_function(X_test)
     loss = hinge_loss(y_test, decision_function)
     print(f"Hinge Loss: {loss:.4f}")
```

```
[ ]: y_val_pred = clf.predict(X_val)
     val_accuracy = accuracy_score(y_val, y_val_pred)
     val_error = 1 - val_accuracy
     print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
     print(f"Validation Error: {val_error * 100:.2f}%")
```