**RPS5**

**(Rock-Paper-Scissors-Gun-Donut)**

1.Introduction

The main objective of my work was to implement a playable version of RPS5 using a camera connected to the computer, using my own image classification model. The version of RPS5 in question is one invented by me, which adds 2 new options, gun, and donut. The rules are simple rock beats scissors and gun, paper beats rock and donut, scissors beats paper and donut, gun beats paper and scissors, donut beats rock and gun. For more visual people, (there is not rock emoji, so rock is represented by 🤟)

🤟→✂️ and 🔫

📄 → 🤟 and 🍩

✂️→ 📄 and 🍩

🍩 → 🤟 and 🔫

🔫 →✂️ and 📄

The model in question was made and trained by me using pytorch.

2. The dataset/datasets used

As said before I initially used outside rps3 datasets for 3 gestures and my own for the other 2 initially, the final dataset includes a mix of my own and outside datasets for rock, paper, scissors and completely my own for the other two. All images were resized to 300\*200 or 200\*300 as I didn’t want huge files to have to train on and all of their backgrounds were removed. My reasoning for removing backgrounds on all of them is backgrounds did not contain any relevant information and just meant more potential confusion for the model. Each class has 400-500 images in its training the training set.

3. Dataset and Data Preprocessing:

As I believe it to be common practice the dataset is organized in a training and a testing folder. The train directory contains images used for model training, while the test directory contains images used for evaluating the model's performance.

The images used in the dataset are a combination of images taken my me, some images taken by friends and family and some images taken from other projects I found on Kaggle and GitHub. All of the images are images of the gestures taken from a birds eye view of the top of the hand, my thinking was that because the donut gesture and the rock gesture are basically the same when looked at from the front this view would resolve that confusion. I also removed the background from all the images using rembg as they don’t really serve a purpose in determining the gesture.

Each image in the dataset is resized to a fixed size (128x128 pixels) to ensure uniform input dimensions for the model. During the training phase, data augmentation techniques (transforms) are applied to enhance the model's ability to generalize to various gestures. And these are:

Random Horizontal Flip: This transformation randomly flips images horizontally with a probability of 0.5. It literally mirrors the image so the model can recognize gestures from both hands, even if it were trained on only images of gestures of one hand (not the case).

Random Vertical Flip: Similar to horizontal flips, vertical flips randomly flip images vertically with a 50% probability. This is to ensure that the model can understand the gesture even if the hand were placed from the wrong side of the camera.

Random Rotation: Images are randomly rotated up to 45 degrees. This transformation helps the model handle variations in the angle of the hand that is feed into it, also people aren’t perfect and it is probable that not all images that the model will be feed are going to be completely straight.

Random Perspective: The application of random perspective transformations further increases the variety in the dataset, allowing the model to recognize gestures even when they are captured from different perspectives.

Model Architecture:

The heart of this project is the custom CNN architecture, that I called "MyTinyVGG." This model is a simplified version of the popular VGG (Visual Geometry Group) network. It consists of three convolutional layer blocks, followed by a classifier block. Each convolutional layer block consists of two convolutional layers, followed by batch normalization, ReLU activation, and max-pooling. The model's structure effectively finds important details in the pictures, which are crutial for figuring out what hand gestures are being made.

The classifier block includes a flattening layer that transforms the 2D feature maps into a 1D tensor, and a fully connected layer for final classification. The number of output units in the final layer matches the number of gesture classes in the dataset.

Training and Evaluation:

The training is performed using the built in Pytorch functions. The training loop runs a specified number of epochs (in this case, 30 epochs). During each epoch, the model is trained on the training data, and then its performance is evaluated on the testing dataset. Key training metrics, including training and testing losses and accuracies, are recorded and printed during the training process.

The train\_step function handles the forward and backward passes and updates the model's parameters using gradient descent. The test\_step function evaluates the model's performance on the testing dataset without updating its weights. This separation ensures that the model generalizes well and helps detect if the model overfits on the train data.

Results and Model Saving:

The trained model is then saved at the end to future use. This is obviously so the model can be used in my rps5 game. The saved model could also be used as a starting point for further fine-tuning or transfer learning tasks.

Final Model results:

The final version of the model after a good amount of fine tuning has given me this result: 95% training and %97% testing accuracy on its final epoch.

A graph of loss and loss

Description automatically generatedA graph with blue lines and white text

Description automatically generated

3.The Gui and the game

My project introduces an object-oriented implementation of a RPS5. The game is encapsulated within the RPS5Game class, featuring methods to play rounds, retrieve scores, reset scores, and determine the computer's choice. Game rules adhere to interactions among these five options, and each round's outcome is displayed. I felt that the object-oriented design would enhance code organization and readability.

The project also includes a simple user-friendly GUI that combines the RPS5Game class and the model. The Gui Loads the model and the class and lets you start playing immediately. The Gui has your camera feed open and initially has an image of a robot, which after playing a round becomes the choice Chance(opponent) has made. All you have to do to play is make the gesture to the camera in the specified way and press the “play round” button. The Gui also keeps track of your wins losses and ties. And if you decide you want to start from the beginning you can reset the score with its button.

The way the game takes in the camera image and feed it to the model is that it first goes through rembg to remove the background because again, the model was trained on images without backgrounds and then feeds it to the model.

4.Conclusuins

In conclusion, this project successfully achieved its main objective, which was to create a playable version of Rock-Paper-Scissors-5 (RPS5) using a camera connected to a computer, while using a custom image classification model for gesture recognition. The image classification model, "MyTinyVGG," was designed and trained effectively using PyTorch, demonstrating its ability to recognize and classify gestures accurately. The dataset and data preprocessing practices, including resizing and data augmentation, contributed to the model's robustness and adaptability.

The project adopts an object-oriented approach, encapsulating the game logic within the "RPS5Game" class, which provides a user-friendly and interactive platform for playing and keeping score. The graphical user interface (GUI) enhances the gaming experience by combining the RPS5Game class and the model, allowing users to play rounds, view computer choices, and track their wins, losses, and ties. The project's successful training results, with a final accuracy of 95% in training and 97% in testing, demonstrate the model's effectiveness.

Overall, this project not only provides an entertaining game but also showcases the potential of image classification models in real-time applications, teaching me lessons in the fields of computer vision and gaming. The combination of innovative game design, effective image classification, and user-friendly interface design results in a cohesive and engaging gaming experience.

I believe the model still has room for improvement as I believe the dataset that I am using not varied enough and the model still occasionally makes mistake in its predictions.

resources:

<https://github.com/paveldat/rock_paper_scissor>

<https://poloclub.github.io/cnn-explainer/>

<https://github.com/DrGFreeman/rps-cv>

<https://www.kaggle.com/datasets/glushko/rock-paper-scissors-dataset>

<https://www.youtube.com/watch?v=V_xro1bcAuA&t=89269s>