Rock-Paper-Scissors CNN Model - Final project

Itamar Cohen - 209133826
Eliyahu Greenblatt - 316302934
Arad Ben Menashe - 207083353

GitHub Repository: [1] Dataset on Kaggle: [2] Kaggle Notebook: [3]

Abstract

This project presents a deep learning approach to classifying rock, paper, and scissors hand signs using a Convolutional Neural Network (CNN). We progressed from a basic softmax model (73% accuracy) through a neural network with softmax (76% accuracy), to our final CNN model with dropout and dynamic data augmentation, including rotations between -30 and 30 degrees for each image. This advanced approach substantially improved our model's performance, achieving 97.9% accuracy. Our findings highlight the effectiveness of combining dropout and data augmentation in complex neural network models for image classification, demonstrating significant improvements in accuracy and model robustness.

1 Introduction

This project focuses on the classification of hand signs representing rock, paper, and scissors, employing a dataset of images with a resolution of 300x200x3 pixels (RGB). Our objective is to develop an advanced deep learning model capable of accurately classifying these images, with broader implications for gesture recognition systems and complex image recognition applications. We began with a basic softmax regression model, progressed to a neural network integrated with softmax, and finally implemented a Convolutional Neural Network (CNN). The CNN was further enhanced with dropout for regularization and dynamic data augmentation, wherein each image was replicated with varying rotations. This strategy significantly improved our model's performance, as evidenced by a leap in classification accuracy shown in Table 1.

This report details shows and discusses the methodologies, results, and key learnings. It contributes to the broader understanding of the application of advanced neural network techniques in image classification, particularly highlighting the effectiveness of dropout and data augmentation in improving model performance.

No.	Epochs	Augmented Data	Learning Rate	Drop Out	Final Accuracy %
1	100	T	0.0001	X	93.6
2	100	T	0.001	0.5	95.6
3	100	F	0.001	0.5	92.9
4	50	T	0.001	0.3	**97.94**
5	50	T	0.001	X	92.92
6	50	F	0.001	0.3	93.83
7	50	F	0.001	X	92.46

Table 1: Model Performance Across Different Parameters

2 Related Work and Required Background

2.1 Related Work

The use of neural networks in image classification, particularly Convolutional Neural Networks (CNNs), has been pivotal in advancing the field of machine learning. Influential studies, such as Krizhevsky et al.'s work on AlexNet[4], have shown the power of CNNs in image recognition tasks.

2.2 Required Background

A fundamental understanding of neural networks, particularly Convolutional Neural Networks (CNNs), is crucial for comprehending our project's methodologies. CNNs are renowned for their effectiveness in visual imagery analysis, utilizing convolution operations and layered structures to process input images. In addition, the softmax function is integral to our model, facilitating the classification of images into one of three categories. This function converts a vector of values into a probability distribution, which is vital for addressing multi-class classification challenges like ours. Moreover, an in-depth knowledge of deep learning, a subset of machine learning in artificial intelligence, is essential. Deep learning employs layered structures of algorithms, referred to as neural networks, to process data in complex ways, learning to perform tasks by considering examples. This approach is particularly effective in identifying patterns and making intelligent decisions based on large amounts of data. Our project leverages these deep learning principles, particularly in neural network architectures and algorithms, to advance the capabilities of our image classification model.

3 Project Description

3.1 Model Architecture

Our project employs a Convolutional Neural Network (CNN) for image classification, implemented using PyTorch. The CNNModel [see code in Figure 3] features two convolutional layers with 8 and 16 filters, kernel size of 3, and max pooling layers. It includes two fully connected layers, leading to a 3-class output

for rock, paper, and scissors. Xavier uniform initialization is used for weight initialization, enabling fair comparison across different model settings [see code in Figure 4].

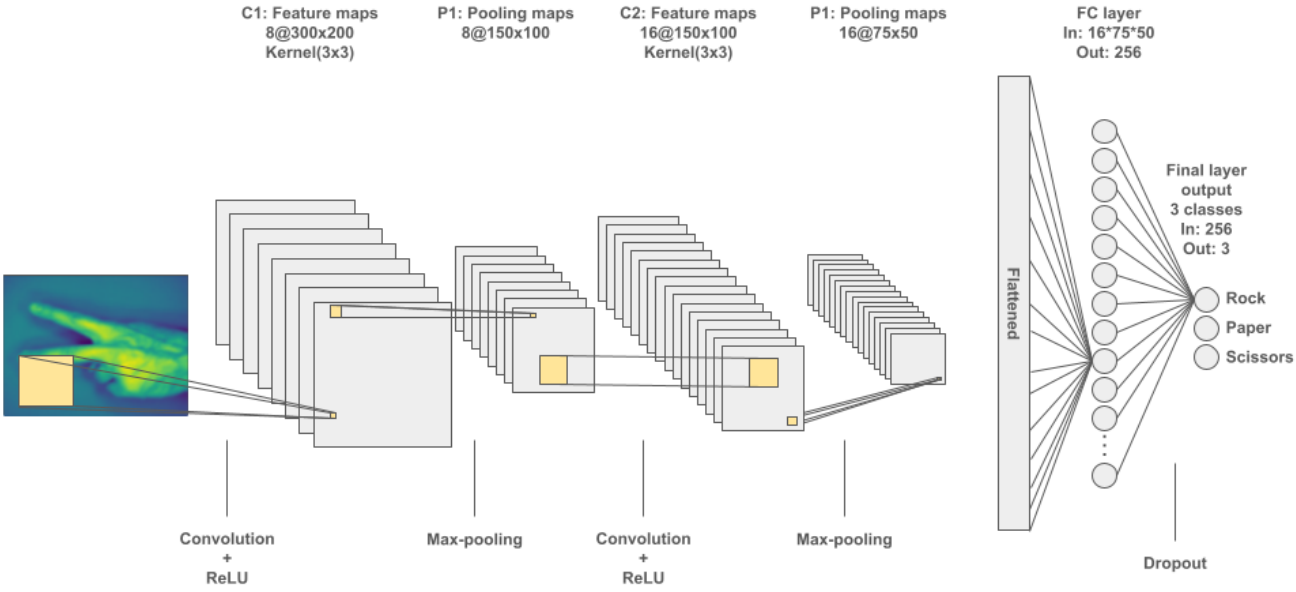

Figure 1: Model Summary

3.2 Data Preparation and Training

The dataset consists of RGB images (300x200 pixels) converted to grayscale. Data augmentation, involving rotation of images, enhances generalization. The dataset is split into 80% training and 20% testing, with training using cross-entropy loss and Adam optimizer over 50 epochs. Real-time performance metrics are available in Figure 7 and the final model's accuracy calculated by running the test set is shown in Figure 6.

4 Results and Observations

The CNNModel achieved 97.9% accuracy on the test set, outperforming previous models (73% and 76% accuracy). Optimal performance was observed with a 0.3 dropout rate, emphasizing the importance of regularization. Rotation-only data

augmentation was more effective than combined methods, as depicted in Figure 8. These results illustrate the balance between model complexity and feature extraction in image classification.

5 Experiments/Simulation Results

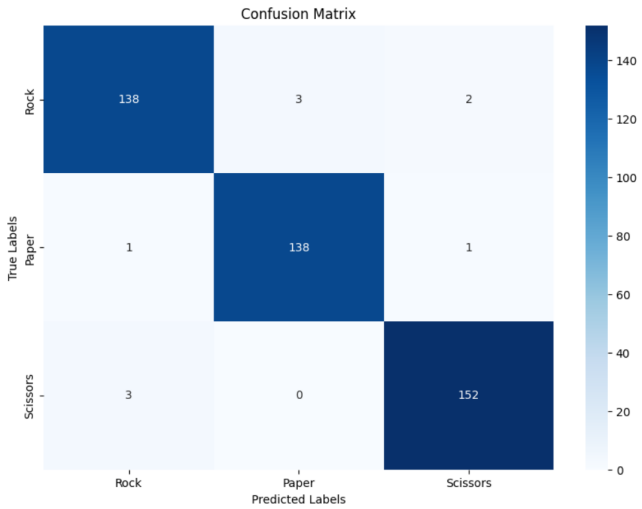

Figure 2: Confusion Matrix

5.1 Optimization of Dropout and Data Augmentation

In our experiments, we investigated the impact of dropout and data augmentation[see code in Figure 5] on the CNNModel's performance. We experimented with dropout rates ranging from 0.3 to 0.5 and found that a rate of around 0.3 yielded the best results. This optimal dropout rate effectively prevented overfitting while maintaining the model's ability to generalize. Additionally, we explored various data augmentation techniques, including rotation, flipping, and adding noise. Our findings indicated that rotational augmentation within a range of -30 to 30 degrees was most beneficial. This selective augmentation approach improved the model's robustness without introducing unnecessary complexity or noise that could hinder learning.

5.2 Network Complexity and Training Parameters

Our experiments also focused on evaluating the impact of network complexity and training parameters. We attempted to scale up the CNNModel by increasing the number of parameters, but this did not lead to significant performance gains. Instead, it resulted in higher computational demands, which were challenging to

meet with the default GPUs available on Kaggle. Additionally, we experimented with various learning rates and epoch counts. We found that a learning rate of 0.001 was optimal, achieving comparable results in 50 epochs to what a lower rate of 0.0001 achieved in 100 epochs. This experimentation underscored the importance of balancing model complexity and resource efficiency, as well as the need to fine-tune training parameters to achieve the best possible performance within practical constraints.

5.3 Testing the Model

We implemented `test_model.py`, a script for real-time model evaluation, loading the CNN and testing it on a random 3 by X image grid to visually compare predictions against actual labels for each class. This direct comparison, exemplified in Figure 8.

6 Previous Attempts

6.1 Recap of Assignment 2

In Assignment 2, the project's initial phase focused on classifying images of rock, paper, and scissors using simpler models. The dataset comprised images converted to grayscale, considering computational efficiency and dimensionality reduction. We employed a basic neural network architecture, which included an input layer representing the flattened grayscale images, a hidden layer with five nodes using ReLU activation, and an output layer with a SoftMax function. Training over 100 epochs with a learning rate of 0.001, using cross-entropy as the loss function, resulted in an accuracy of 76%. This was a modest improvement over a simpler SoftMax model that achieved 73% accuracy.

6.2 Evolution of the Project

The advancement from Assignment 2 to our final project was a pivotal journey toward refining our model's accuracy in classifying hand signs. Initially, we transitioned from simpler models to a Convolutional Neural Network (CNN), a strategic move driven by our need to more effectively handle image data. The CNN's architecture, renowned for its proficiency in feature extraction and learning from spatial hierarchies, initially boosted our accuracy to a commendable 95%. However, we didn't stop there. Recognizing the potential for further enhancement, we integrated dropout techniques and implemented dynamic data augmentation. These additions were critical in our endeavor to combat overfitting and enhance the model's ability to generalize from the training data. The incorporation of dropout and controlled image rotations in our data augmentation strategy was a game-changer, leading us to a remarkable accuracy of 97.9%. This phase of our project was not just about transitioning to a more sophisticated model but also about meticulously fine-tuning it with advanced techniques to achieve optimal performance.

7 Conclusions

7.1 Summary of Findings

In this project, we learned a lot about how to improve a computer model for recognizing hand signs. Our best model, a CNN, reached an accuracy of 97.9% in just 50 training rounds. This was much better than our first simple models, which only got 73% and 76% accuracy. The big improvement came from using a CNN and adding special techniques like dropout and data augmentation (to make our model better at handling different kinds of images).

7.2 Future Work

There's still more to explore in the future. We could try using our CNN model on stronger computers to see if it performs even better. Another goal is to reach an accuracy of over 99%. To do this, we want to experiment with more ways to change the data, like using different image sizes or more augmentation methods. These changes could help our model become even more accurate and reliable. By pushing the limits of our current model, we aim to achieve even higher standards in image classification.

7.3 Final Thoughts

This project was an eye-opener in seeing the difference between simple deep learning networks and more advanced CNNs. It was fascinating to observe how the switch to a CNN significantly boosted our model's performance. Additionally, incorporating techniques like data augmentation and dropout proved to be game-changers. These strategies not only improved our model's accuracy but also provided valuable insights into the importance of adapting and refining our approach. It's clear that in the field of computer science, especially in areas like image recognition, continuous learning and experimentation are key to success.

References

[1] GitHub Repository for the project, `https://github.com/aradbm/rps-model`.

[2] Dataset on Kaggle: "Rock Paper Scissors," `https://www.kaggle.com/datasets/drgfreeman/rockpaperscissors`.

[3] Kaggle Notebook: "Rock Paper Scissors Classification - 97.9% Accuracy," `https://www.kaggle.com/aradbenmenashe/rock-paper-scissors-classification-97-9-acc`.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems 25 (NIPS 2012)*.

8 Code & Figure Appendix

```python
class CNNModel(nn.Module):
    def __init__(self, dropout_rate=0.5):
        super(CNNModel, self).__init__()
        # Convolutional layers
        self.conv1 = nn.Conv2d(1, 8, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(8, 16, kernel_size=3, stride=1, padding=1)
        # Max pooling
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        # Dropout
        self.dropout = nn.Dropout(dropout_rate)
        # Fully connected layers
        self.fc1 = nn.Linear(16 * 75 * 50, 256)
        self.fc2 = nn.Linear(256, 3)
        # Initialize weights
        self.init_weights()

    def forward(self, x):
        # Apply first convolution and pooling
        x = self.pool(F.relu(self.conv1(x)))
        # Apply second convolution and pooling
        x = self.pool(F.relu(self.conv2(x)))
        # Flatten the output
        x = x.view(-1, 16 * 75 * 50)
        # Apply first fully connected layer
        x = F.relu(self.fc1(x))
        # Apply dropout
        x = self.dropout(x)
        # Apply second fully connected layer
        x = self.fc2(x)
        return x
```

Figure 3: Model Code

```python
# Model
##########################################
dp_rate = 0.3
model = CNNModel(dropout_rate=dp_rate).to(device)
augment_bool = True
epochs = 50
learning_rate = 0.001
##########################################
```

Figure 4: Model Settings

```python
def random_rotation(image):
    angle = random.randint(-30, 30)
    rows, cols = image.shape[:2]
    rotation_matrix = cv2.getRotationMatrix2D((cols / 2, rows / 2), angle, 1)
    return cv2.warpAffine(image, rotation_matrix, (cols, rows))


def random_flip(image):
    flip_type = random.choice([-1, 0, 1])  # Horizontal, vertical or both
    return cv2.flip(image, flip_type)
```

Figure 5: Augmentation Functions

```
################################################
Total number of parameters: 15362275
Drop out rate: 0.3
Augment: True
Epochs: 50
Learning rate: 0.001
--Final model accuracy: 0.9794520547945206---
################################################
```

Figure 6: Model Summary

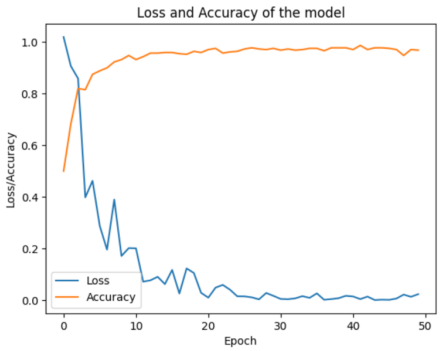

Figure 7: Loss and accuracy in training

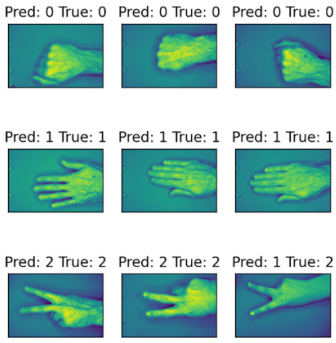

Figure 8: Test output