DiceMNIST - A Convolutional Neural Network for recognizing dice digits.

Yashovardhan Srivastava

27 October,2021

Introduction

Convolutional neural network(CNN) have been firmly established as most commonly used methods for computer vision tasks such as image and video recognition,classification and segmentation. A benchmark result to test CNNs is training it on the MNIST handwritten digits. In this work, we test the results in recognizing digits on a dice using our model - DiceMNIST.

Background

Data Collection

We collected all the images of dices with our own resources. We used Canon EOS 1300D for capturing images with a white background to avoid any noise. For the initial build, we used only 2 dice - with complimentary colors on the numbers in gray-scale.We took images from various angles to increase variability(initially 0^o and 45^o)

Data Processing

All of the images collected were edited using GNU Image Manipulation Tool(GIMP). Each image was first cropped to highlight the subject, then converted to gray-scale and finally resized to 28x28 pixels. Each image after this modification was flipped horizontally,vertically,rotated $90o$ clockwise, 90^o counter-clockwise, and 180^0 to induce randomness.

Database Creation

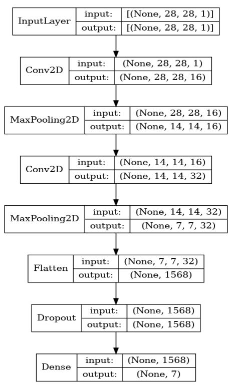
From the previous 2 processes, we able to manage 144 images which were then splitted into testing and training datasets and then carefully labeled. They were then saved as Numerical Python (NumPy) data file.

For the second version of DiceMNIST, the size of dataset is increased by a significant amount with the help of Keras ImageDataGenerator. Using this we were able to create a diverse dataset of around 1500 images(including both training and testing).

Model Architecture

We followed a straightforward model for testing on DiceMNIST. It consisted of 2 Conv2D layers, followed by pooling layers. For regularization, we used residual dropout after flattening the layer. Most hyperparameters were adjusted after fine tuning with Keras Tuner.

In the second version,the model bottleneck analysis revealed that it spent too much time of Input Data Generation, so we opted to use generators to get the data directly from their respective directories(using flow from directory function of Keras). This helped substantially reduce readings of Input Pipeline Analyzer from Tensorboard.



Training

Hardware

For our initial batch, we trained our model on our personal machine using NVIDIA MX100 GPU. For the hyperparameters described in this paper, each training step took around 0-1 second. We trained it for 100-200 steps which took about 3 minutes.

Optimizer and Losses

For our model, we used Adam(Adaptive Moment Estimation) optimizer with learning rate 0.001,$\beta_1 = 0.90$, $\beta_2 = 0.999$ and $\epsilon = 10^{-9}$. We did not used any learning rate scheduler for our initial model. Since our labels were categorically distributed, we used Categorical Crossentropy as our loss function.

Results

DiceMNIST, by using the hyperparamters given in the paper, gave average accuracy of 87.5 and loss of 0.3. We aim to increase the accuracy by adding more data in our dataset. All the other analysis for the model can be accessed using TensorBoard Dev from the link:
`https://tensorboard.dev/experiment/TzCA1ZjpREaEcgx3BVZmGQ/`
For using the model, the github repository is:
`https://github.com/yash-srivastava19/SpicyDicey.git`