Project Report

Training a CNN on the SVHN Dataset for Classification

AUTHOR: SIKANDAR ALI

Introduction

This report summarizes the process and results of a project that involved training a Convolutional Neural Network (CNN) on the Street View House Numbers (SVHN) dataset for digit classification. The goal of this project was to develop a model capable of accurately recognizing and classifying digit images from the SVHN dataset, which consists of house number images taken from street views.

Dataset Overview

The SVHN dataset contains over 600,000 labeled digits from street view house numbers. The dataset is divided into:

Training set: 73,257 digit images

Testing set: 26,032 digit images

Extra set: 531,131 digit images (not used in this project for simplicity)

Each image is a 32x32 RGB image, and the digits range from 0 to 9.

Data Preprocessing

The data preprocessing step involved:

* Normalization

All pixel values were normalized to a range between 0 and 1 by dividing by 255.One-Hot Encoding: The labels were converted to one-hot encoded format to suit the categorical cross-entropy loss function used in the model.

* Data Augmentation

Techniques such as rotation, width shift, height shift, and horizontal flip were applied using Keras's Image Data Generator to artificially increase the diversity of the training data and prevent overfitting.

Model Architecture

A Convolutional Neural Network (CNN) was built using the following architecture:

* Input Layer: 32x32 RGB images
* Convolutional Layers: Multiple layers with 32, 64, and 128 filters with ReLU activation.
* Max Pooling Layers: Used after each convolutional layer to reduce dimensionality.
* Dropout Layer: Applied dropout (rate of 0.5) to prevent overfitting.
* Flatten Layer: Flattened the 3D feature maps into 1D for input into the fully connected layers.
* Dense Layers: Two dense layers with ReLU activation.
* Output Layer: Softmax activation with 10 units for classification (0-9 digits).

Training Process

* Optimizer: Adam optimizer was used with a learning rate of 0.001.
* Loss Function: Categorical cross-entropy was employed as the loss function.
* Epochs: The model was trained for 10 epochs with a batch size of 64.
* Validation: The model's performance was evaluated using the test dataset at each epoch.

Evaluation and Results

* Confusion Matrix

Below is the confusion matrix generated after the testing phase, which highlights the model’s performance on individual digits:

The confusion matrix shows the model's performance across different digit classes. The high values on the diagonal indicate correct predictions, while the off-diagonal values represent misclassifications. For example, the model correctly classified 4871 images of the digit '0' but misclassified a small number of images as other digits.

* Model Performance

The model achieved a good level of accuracy, but there is room for improvement, especially in distinguishing between certain digits such as 1 and 7, which seem to have a higher number of misclassifications.

Challenges and Improvements

Some challenges faced during the project included:

* Class Imbalance

Certain digits in the SVHN dataset were over-represented, which could have caused bias in the model.

* Misclassifications

Some digits were harder to distinguish due to similarities in their appearance or due to noise in the images.

To improve performance:

* Data Augmentation

Additional augmentation techniques could be employed to increase dataset diversity.

* Model Fine-Tuning

More advanced CNN architectures or transfer learning techniques could be explored.

* Increased Epochs

Training the model for a greater number of epochs might lead to better generalization.

Conclusion

This project demonstrated the effective use of CNNs for digit classification using the SVHN dataset. The model was able to achieve reasonable accuracy, with clear indications of where improvements could be made. Future work may involve experimenting with more complex architectures or using additional data augmentation and fine-tuning strategies to further boost performance

Reference

Github Link: https://github.com/Alexander181