**Report Title : SVHN Digit Recognition using Convolutional Neural Networks**

**Name : Asad Nazir**

**Email :** [**asadnazir1419@gmail.com**](mailto:asadnazir1419@gmail.com)

**Github Link : https://github.com/asadnazir14/-DEEP-LEARNIG-PROJECTS**

**1. Introduction**

The task of digit recognition is an essential problem in the field of computer vision, having applications in areas such as automated postal systems, bank check processing, and even the interpretation of street view images. The **Street View House Numbers (SVHN)** dataset, obtained from images of house numbers in real-world street views, presents a unique challenge due to the varying scales, orientations, and lighting conditions of the digits. This report details the process of building a Convolutional Neural Network (CNN) to perform digit classification on the SVHN dataset, achieving an accuracy of **96%**.

This project aims to demonstrate the application of deep learning, specifically CNNs, in image classification tasks, using TensorFlow as the core library for model implementation.

**2. Dataset Description**

The **SVHN dataset** is a real-world image dataset that contains over 600,000 labeled digits in color images, which are obtained from house numbers in Google Street View images. It consists of both training and test sets:

* **Training set**: 73,257 images
* **Test set**: 26,032 images
* **Extra set**: 531,131 images (used for additional training data if needed)

Each image in the dataset is a 32x32 RGB image with a single digit (0-9). The labels are in the range [0-9], where 0 represents the digit '0', and so forth.

**3. Methodology**

**3.1. Data Preprocessing**

Before feeding the images into the neural network, several preprocessing steps are required to standardize the data:

1. **Normalization**: The pixel values of each image, which originally range from 0 to 255, are scaled to a range of 0 to 1 to improve the performance of the neural network. This normalization step helps the model converge faster during training.
2. **One-Hot Encoding**: The labels, which are in integer form, are converted to one-hot encoded vectors. For example, if a label is 5, the corresponding vector would be [0, 0,

**3.2. Model Architecture**

The Convolutional Neural Network (CNN) architecture used for this task consists of three main convolutional layers, followed by max-pooling, batch normalization, dropout layers to prevent overfitting, and fully connected dense layers for the final classification.

**Layer Breakdown**:

* **Conv2D Layers**: Extract important features from the images such as edges, corners, and textures.
* **MaxPooling Layers**: Reduce the dimensionality of the feature maps, retaining the most critical information.
* **Batch Normalization**: Helps in normalizing the activations to reduce internal covariate shifts.
* **Dropout**: Prevents overfitting by randomly dropping a fraction of neurons during each training step.
* **Fully Connected Dense Layers**: Used for final classification.

**3.3. Model Compilation**

The model is compiled with the following configuration:

* **Optimizer**: Adam, which is an efficient gradient-based optimizer for minimizing the loss function.
* **Loss Function**: Categorical cross-entropy is used since the task involves multiclass classification.
* **Evaluation Metric**: Accuracy is used to evaluate the performance of the model during **3.4. Training the Model**

The model is trained on the training dataset using a validation split of 20%. The model is trained for 20 epochs, and the batch size is set to 64.

**4. Results and Discussion**

After training the model, it was evaluated on the test dataset to check its generalization performance. The following results were obtained:

* **Training Accuracy**: 96%
* **Validation Accuracy**: 92%
* **Test Accuracy**: 91%

The high training accuracy indicates that the model learned the features of the training data well. However, the slight gap between the validation and test accuracies suggests potential overfitting, although the use of dropout layers and batch normalization helped reduce it.

**4.1. Visualizing Training and Validation Accuracy**

The training and validation accuracies were plotted over the 20 epochs to visualize how the model improved over time:

The plot indicates that while the model continued to improve on the training data, the validation accuracy leveled off after a certain number of epochs, suggesting early stopping could be a good strategy for future experiments.

**4.2. Confusion Matrix**

To gain deeper insight into the model's performance, a confusion matrix was generated to visualize the correct and incorrect predictions across different digit classes.

The confusion matrix revealed that most errors occurred between similar-looking digits, such as 1 and 7, indicating the model's struggle with subtle visual distinctions in certain cases.

**5. Conclusion**

In this project, a Convolutional Neural Network (CNN) was successfully implemented to classify digits from the **SVHN dataset**. The final model achieved a test accuracy of **91%**, which demonstrates the effectiveness of CNNs in image classification tasks.

**6. Future Work**

There are several avenues for further improvements to the model:

1. **Data Augmentation**: Using techniques like rotation, zooming, and shifting can help the model generalize better by introducing more variations in the training data.
2. **Hyperparameter Tuning**: Experimenting with different architectures, optimizers, and learning rates could yield better results.
3. **Transfer Learning**: Utilizing pre-trained models such as VGG or ResNet could improve accuracy, especially on the extra SVHN dataset.

Overall, the project provided valuable insights into the workings of CNNs and their application in real-world digit recognition tasks.