Object Recognition

Upenyu Hlangabeza

Table of Contents

Introduction

Overview of the Problem

Evaluating how CNNs would perform on image classification of the CIFAR-100 dataset with an achievable benchmark of 39.43%. Any value higher than this indicates outliers.

Proposed Solution

The proposed solution centres on the analysis of the CIFAR-100 dataset, using Convolutional Neural Networks with TensorFlow for image recognition. Leveraging neural networks, particularly Convolutional Neural Networks (CNN), the focus is on automating image classification—a critical aspect of object recognition. The task involves categorizing images into 100 distinct categories within the dataset, which comprises 500 training images and 100 testing images per category. The fixed dimensions of 32x32x3 (height, width, and colour channels) add complexity to the classification process. The primary goal is to employ advanced machine learning algorithms, specifically CNN, to accurately categorize testing images based on their respective object categories. The report underscores the importance of this approach in addressing the critical need for efficient and correct image classification.

Methodology

Convolutional Neural Network

Convolutional Neural Networks (CNN) seems suited for image recognition, as they are designed to exploit the spatial structure of images. They use convolutional layers that apply filters (kernels) across small regions of the input image. This local connectivity allows CNNs to capture local patterns and features effectively. With multiple layers like convolutional layers, pooling layers, and fully connected layers , these layers form a hierarchical structure that learns increasingly complex and abstract features. Lower layers detect simple features like edges and textures, while higher layers combine these features to detect more complex patterns and objects.

Importing and Loading Dataset

The dataset is loaded from the provided files: fine_labels.csv, trnImage.npy, tstImage.npy, trnLabel_fine.npy, and tstLabel_fine.npy. Data was imported using numpy as well. The data is labelled to understand how the data is structured. Due to the nature of the data, as it was already trained and split there was no need to do so.

Preprocess and Reshaping Data

```
print("Shape of train label",train_Label.shape)
print("Shape of test label",test_Label.shape)
print("Shape of train Image",train_Image.shape)
print("Shape of train Image",test_Image.shape)
```

Executed at 2023.12.14 07:05:24 in 6s 322ms

```
Shape of train label (50000,)
Shape of test label (10000,)
Shape of train Image (32, 32, 3, 50000)
Shape of train Image (32, 32, 3, 10000)
```

Because of the inherent characteristics of the data, it was necessary to transpose the data to align its shape with the corresponding labels. The top image illustrates the original shape, while the bottom image depicts the shape after transposition. Ensuring both the image and label share the same shape is crucial for accurately modelling the data.

```
print("Train Image:",train_Image.shape)
print("Train Label:",train_Label.shape)

print("Test Image:",test_Image.shape)
print("Test Label:",test_Label.shape)
```

Executed at 2023.12.14 08:49:02 in 24ms

```
Train Image: (50000, 32, 32, 3)
Train Label: (50000,)
Test Image: (10000, 32, 32, 3)
Test Label: (10000,)
```

Building CNN

In building our CNN model, we went for a sequential setup. Starting with a 32-filter convolutional layer using a 3x3 kernel and ReLU activation, we processed 32x32x3 input data. We then applied max pooling with a 2x2 pool size for spatial down sampling. This was followed by a similar pattern: a 64-filter convolutional layer, another max-pooling layer, and then flattening the output. The flattened data flowed through a dense layer with 128 neurons and ReLU activation. Finally, a dense layer with 100 neurons and softmax activation handled the categorization into one of 100 classes. This arrangement of convolutional and pooling layers, along with dense layers, forms the core structure of our CNN model for the task at hand.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(100, activation='softmax'))
```

Compile the Model

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(train_Image, train_Label, epochs=10, batch_size=32, validation_data=(test_Image, test_Label))
```

Executed at 2023.12.14 08:35:35 in 3m 6s 992ms

The optimizer and loss function play crucial roles in training a neural network. The Adam optimizer is a popular optimization algorithm that adapts the learning rates of each parameter individually, optimizing the model's ability to converge efficiently during training. On the other hand, the sparse categorical cross-entropy loss function is specifically designed for multi-class classification problems where each instance belongs to only one class.

It measures the difference between the predicted probability distribution and the true distribution of class labels, penalizing the model for incorrect predictions. The "sparse" version is used when the class labels are integers, providing a more memory efficient implementation compared to its categorical counterpart.

Train and Evaluating the Model

After training for 10 epochs with batches of 32 instances, we check how well our model grasps the essence of the data. Using validation data, which includes testing images and labels, we ensure the model not only learns effectively but also generalizes well beyond the training set, highlighting its knack for providing accurate predictions.

Deploying our trained model, we set it to work on the testing images, generating predictions for the corresponding labels. The predicted labels are derived through a straightforward argmax operation on the model's output. Simultaneously, we snag the true labels from the testing dataset for a side-by-side comparison. This step gives us a real-world evaluation, allowing us to gauge how closely our model aligns with the actual labels and highlighting its proficiency on data.

Data Accuracy

We determine the overall accuracy of the model by employing scikit-learn's accuracy score function. The resulting accuracy percentage is then printed, providing a clear measure of the effectiveness of the proposed CNN model.

Experimental Results

The experimental results section will present the findings of the proposed method. This includes the quantitative performance metrics of the trained models on the testing set. Comparative analysis between different methods will be provided, highlighting their efficacy in tackling the object recognition task.

Results

With the benchmark set at 39.43%, my model achieved an accuracy of 33.31%. This means that my model is capable of correctly identifying 84.47% of objects.

Conclusion

This report addresses challenges posed by a surge in image data, proposing a solution centred on Convolutional Neural Networks (CNN) for accurate image classification using the CIFAR-100 dataset. The model's deployment, as evidenced by a confusion matrix and accuracy metrics, highlights its proficiency in handling unseen data. Results indicate increased outlier values, reflecting the model's heightened accuracy. The upcoming experimental results section promises a brief yet insightful comparative analysis of methods for object recognition. The report emphasizes the importance of robust machine learning models in navigating complex image data.

References

A. S. Alharthi, S. U. Yunas and K. B. Ozanyan, “Deep Learning for Monitoring of Human Gait: A Review,” IEEE Sensors Journal, vol. 19, pp. 9575-9591, 2019.

T. Isakava, “A gentle introduction to human activity recognition,” 31 March 2022. [Online]. Available: https://indatalabs.com/blog/human-activity-recognition.