STL-10 Image Recognition with ResNet-18

Ömer Faruk Zeybek
Department of Artificial Intelligence and Data
Istanbul Technical University
Istanbul, Turkey
Email: zeybeko22@itu.edu.tr
June 11, 2024

***Abstract*—This project explores the use of contrastive learning for embedding generation using a pre-trained ResNet18 model, followed by fine-tuning a linear classifier for image classification. The project addresses the problem of class imbalance and proposes a hypothesis that utilizing contrastive loss can improve the representation learning, leading to better classification performance. The model is trained and tested on a STL-10 dataset, with results show significant improvements in classification accuracy.**

***Index Terms*—resnet-18, image, contrastive loss, linear classifier**

CONTENTS

I. INTRODUCTION

This report documents the project conducted for the BLG 454E course in Spring 2024. In the project using the ResNet-18 model, the model is trained using contrastive loss, after the training of the model, the model is frozen and a linear classifier is added on top of it and the accuracy is calculated and then evaluated with the test set.

II. PROBLEM STATEMENT - HYPOTHESIS - LITERATURE SURVEY

A. Problem Statement

The task is to enhance image classification performance by addressing the challenges of class imbalance and representation learning.

B. Hypothesis

Utilizing contrastive loss for embedding generation followed by a fine-tuned linear classifier will improve classification accuracy compared to traditional end-to-end training methods.

C. Literature Survey

- **Contrastive Learning:** Recent advances in contrastive learning, such as SimCLR and MoCo, have shown promising results in representation learning by maximizing agreement between differently augmented views of the same data point.
- **ResNet18:** The ResNet architecture, particularly ResNet18, has been widely used for various image classification tasks due to its effectiveness and efficiency.
- **Class Imbalance:** Techniques like weighted random sampling and data augmentation are commonly used to address class imbalance in training datasets.

III. METHOD(S) - DATA - RESULTS

A. Method(s)

1) **Data Agumentation:**
   - Applied transformations including resizing, random flips, rotations, color jitter, and grayscale conversion to enhance data variability.

   Code

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomRotation(20),
    transforms.RandomResizedCrop(224,
        scale=(0.6, 1.0)),
    transforms.ColorJitter(brightness=0.5,
        contrast=0.5, saturation=0.5, hue=0.2)
    transforms.RandomGrayscale(p=0.2),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5),
        (0.5, 0.5, 0.5))
])
```

2) **Embedding Model:**

- A pre-trained ResNet18 model is modified to generate 128-dimensional embeddings.
- A dropout layer and a fully connected layer are added to the ResNet18 model.

Code

```
class EmbeddingModel(nn.Module):
    def __init__(self, base_model,
    embedding_size):
        super(EmbeddingModel,
        self).__init__()
        self.base_model =
        nn.Sequential(
        *list(
        base_model.children())[:-1])
        self.fc =
        nn.Linear(
        base_model.fc.in_features,
        embedding_size)
        self.dropout = nn.Dropout(
        0.5)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.base_model(x)
        x = x.view(x.size(0), -1)
        x = self.dropout(x)
        x = self.fc(x)
        x = self.relu(x)
        return x
```

3) **Contrastive Loss:**
   - Pairs of images are created, and contrastive loss is used to train the embedding model by minimizing the distance between embeddings of similar images and maximizing the distance between embeddings of dissimilar images.

Code

```
class ContrastiveLoss(nn.Module):
    def __init__(self, margin=1.0):
        super(ContrastiveLoss, self)
        .__init__()
        self.margin = margin

    def forward(self, output1, output2,
    label):
        euclidean_distance =
        F.pairwise_distance(
        output1,output2)
        loss_contrastive =
        torch.mean((1 - label) *
        torch.pow(
        euclidean_distance, 2) +
        label * torch.pow(
        torch.clamp(self.margin -
        euclidean_distance, min=0.0),
        2))
        return loss_contrastive
```

4) **Linear Classifier:**
   - A linear classifier with batch normalization and ReLU activation is added on top of the embedding model.
   - The classifier is fine-tuned using cross-entropy loss.

Code

```
classifier = nn.Sequential(
    nn.Linear(in_features=128,
    out_features=512),
    nn.BatchNorm1d(512),
    nn.ReLU(),
    nn.Linear(in_features=512,
    out_features=10)
).to(device)

class FullModel(nn.Module):
    def __init__(self, embedding_model,
    classifier):
        super(FullModel,
        self).__init__()
        self.embedding_model =
        embedding_model
        self.classifier = classifier

    def forward(self, x):
        x = self.embedding_model(x)
        x = self.classifier(x)
        return x
```

B. Data

While doing this project, separate folders were created for training and test data in the downloaded dataset, then separate folders were created for each class label and 100 images were added to each of them.

- **Class Labels** : Airplane, bird, car, cat , deer, dog, horse, monkey, ship , truck
- **Training Data:** STL-10 dataset with class labels for training
- **Test Data:** STL-10 dataset with class labels for testing
- **Data Source:** STL-10 dataset https://www.kaggle.com/datasets/jessicali9530/stl10

C. Results

1) **Training Phase:**
   - Loss curves for contrastive loss and classifier loss demonstrate the model's learning progression.
2) **Linear Classifier Phase:**
   - The linear classifier implemented using Cross Entropy achieved an accuracy of up to 74% and loss fell below 1

3) **Test Phase:**
   - Achieved significant test accuracy with small loss
   - Confusion matrix analysis to showcase performance.

IV. Discussion

The project methodology, which combines contrast learning with a fine-tuned classifier, has shown promising results in improving image classification performance. The following discussion details key aspects of the project's approach and results:

A. Effectiveness of Contrastive Learning

Contrastive learning proved to be highly effective in improving the quality of embeddings. By training the model to maximize agreement between differently augmented views of the same image and minimize agreement between views of different images, the model learned more discriminative features. This led to improve model's ability to distinguish between classes

B. Class Imbalance

Data augmentation increased the variability of the training data, making the model more robust and reducing overfitting. Weighted random sampling ensured that each class contributed proportionately to the training process, preventing the model from being biased.

C. Future Work

Further experimentation with different contrastive learning techniques could yield even better results. Additionally, exploring the use of more complex architectures for the embedding model and classifier could further improve the model's generalization and performance across diverse datasets and tasks.

V. Conclusion

In conclusion, the project's methodology showcased the effectiveness of contrastive learning in improving embedding quality and addressing class imbalance issues in image classification. The combination of data augmentation, weighted sampling, and fine-tuning techniques contributed to significant enhancements in classification accuracy, demonstrating the potential of this approach for real-world applications in computer vision and beyond.

Appendix

- **Code:** Complete code of the project can be accessed via the following repository link: https://github.com/itu-itis22-zeybeko22/Learning-From-Data-Project