Deep Learning Reproducibility Project

Sparsity Invariant CNNs

Group 66:

Shipeng Liu
S.Liu-56@student.tudelft.nl
5956560

Zhewen Gao
Z.Gao-13@student.tudelft.nl
5955378

Taoyue Wang
T.Wang-32@student.tudelft.nl
5914795

Mingkang Tang
M.Tang-3@student.tudelft.nl
5824117

Code link:
https://github.com/TaoyueWang/sparsityCNN/tree/main

Task Division

- Create the sparse version of datasets - Taoyue Wang
- Data preprocessing - Taoyue Wang
- Implementing network in PyTorch - Taoyue Wang
- Sparse convolution module and model architecture design - Taoyue Wang
- Ablation Study of Model on Caltech-101 - Shipeng Liu
- Hyperparameter Tuning of Model on Caltech-101 - Shipeng Liu, Mingkang Tang
- Hyperparameter Tuning of Model on MNIST - Zhewen Gao, Mingkang Tang
- Sparse ConvNet and ConvNet Testing for MNIST - Zhewen Gao
- Write section 1 of the blog post - Mingkang Tang
- Write sections 2.1 and 2.2 of the blog post - Taoyue Wang
- Write sections 2.3 and 2.4 of the blog post - Shipeng Liu, Zhewen Gao
- Write section 3 of the blog post - Shipeng Liu, Zhewen Gao, Mingkang Tang
- Write sections 4 and 5 of the blog post - Shipeng Liu, Zhewen Gao

1. Introduction

In most cases, the input to a traditional CNN is an image or video represented by a dense tensor. When only a small proportion of the pixels carry valid information, the data is said to be sparse.

The naïve approach to handling sparse input data by setting all invalid pixels to a default value leads to suboptimal results as the invalid pixels act as distortion. Moreover, the approach is sensitive to the level of sparsity, in the sense that the network is expected to be applied to data with a similar level of sparsity as the training data.

The paper we reproduced introduced a novel sparse convolutional layer that takes additionally a mask as input, which indicates the validity of each position of the image or video. The mask is conveyed across layers by max-pooling. This simple yet effective model, which we will refer to as SparseConvNet in the following, can handle large levels of sparsity without significant loss of accuracy and is invariant to the sparsity levels. In the paper, an experiment on the KITTI depth dataset was conducted, and the results in Table 3 of the paper showed effectiveness at all levels of sparsity.

In our project, we reproduced Table 3 of the original paper on MNIST and Caltech101 datasets. We confirmed that by reasonably adjusting the structure of the original SparseConvNet model, it can effectively handle the generalized sparse data from those two datasets.

2. Experimental Setup

2.1 Dataset and Data Prepossessing

Different from the paper, which mainly uses the KITTI dataset for the depth upsampling task, this project focuses on the image classification task and utilizes two new datasets: MNIST and Caltech-101, which are well organized as popular research benchmarks. However, instead of using these datasets directly, we add some sparsity to the images and create a sparse version of the datasets as in real-world applications. In this section, what information these datasets contain and how we generate the sparse version of datasets will be well illustrated.

1) The MNIST Dataset

   The MNIST digits classification dataset contains a total of 70,000 28 x 28 grayscale images of handwritten digits 0-9 split into a train set of 60,000 images and a test set of 10,000 images. The labels are one-hot encoded.

   In our experiments, the first step is to create a sparsified version of the dataset. We perform this by generating a random mask for each image in the dataset. The mask is a binary array, with zero representing an invalid position and one representing a valid position. After multiplying each original image with the random mask, the sparse images are generated. For the MNIST dataset, we create a dataset class

"SparseDatasetMNIST" inherited from the torch dataset class *"torch.utils.data.Dataset*". The "__getitem__" function is modified by generating a random mask for each image and processing the image with the function "*masked_fill*". By passing in different sparsity levels, a dataset with different sparsities can be created. After that, the processed image and the corresponding label are returned. This part of the code is shown as follows:

```
# generate the random mask
        tensor_length = 28 * 28
        ones_count = int(tensor_length * self.sparsity_level)
        ones_indices = torch.randperm(tensor_length)[:ones_count]
        random_mask = torch.zeros(tensor_length).bool()
        random_mask[ones_indices] = 1
# generate the sparse image
        processed_image = image.masked_fill(random_mask, 0).numpy()
```

Some examples of sparse images, with a sparsity level of 70%, are shown in Figure 1:

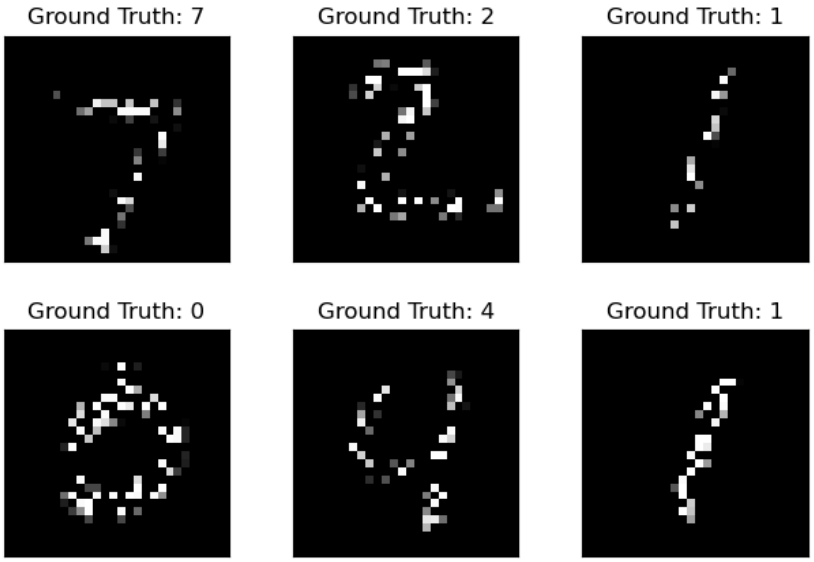


Figure 1: examples of sparse images in MNIST (sparsity_level = 70%)

2) The Caltech-101 Dataset

The Caltech-101 dataset contains a total of 9,148 pictures of objects belonging to 101 categories, each with 40 to 800 images sized roughly 300 x 200 pixels. In this project, after loading the images and labels, images are resized to 224 x 224 pixels, and normalized with the mean [0.485, 0.456, 0.406] and standard deviation [0.229, 0.224, 0.225].

Because the dataset capacity increases, we adopt a different and more parallelized method to make the dataset sparse. We first choose a sparsity level and create masks for all images using the Pytorch function “torch.bernoulli”. Masks are generated for the training set, validation set, and test set, respectively:

```
# generate random masks for all images
sparsity_level = 0.7 #0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7
train_mask =
torch.bernoulli(torch.full(torch.Size([len(x_train),224,224]), 1 -
sparsity_level))
validation_mask =
torch.bernoulli(torch.full(torch.Size([len(x_validation),224,224]), 1 -
sparsity_level))
test_mask = torch.bernoulli(torch.full(torch.Size([len(x_test),224,224]),
1 - sparsity_level))
```

After generating the mask, we pass the mask to the dataset class “CustomDataset”:

```
# process the image with the mask
train_data = CustomDataset(x_train, y_train, train_mask)
validation_data = CustomDataset(x_validation, y_validation,
validation_mask)
test_data = CustomDataset(x_test, y_test, test_mask)
```

Inside the dataset class, images multiply with the mask to become sparse images:

```
for i in range(len(self.images)):
            self.images[i] = self.images[i] * self.mask[i]
```

Some examples of sparse images with a sparsity level of 30% are shown in Figure 2:

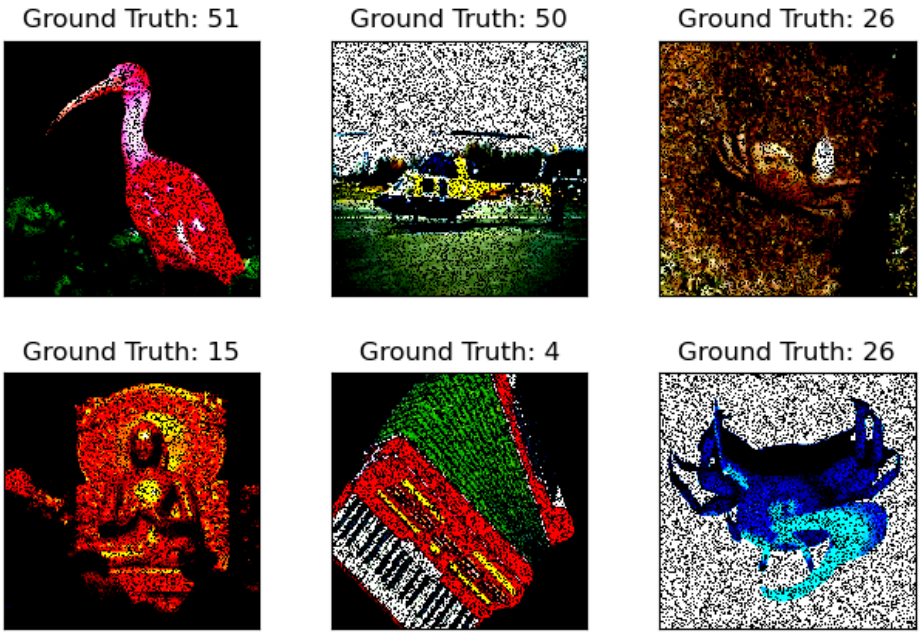


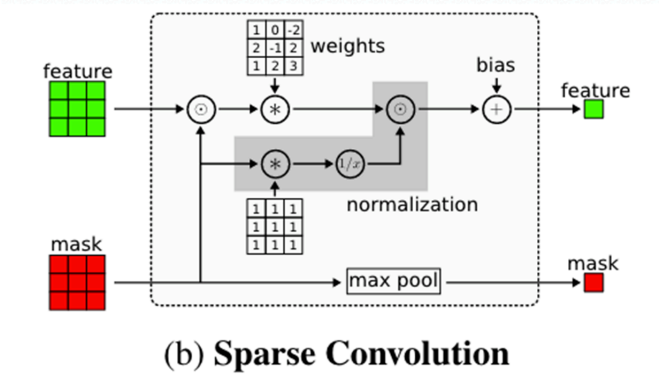
Figure 2: examples of sparse images in Caltech-101 (sparsity_level = 30%)

2.2 Sparse Convolution Neural Network

The sparse convolution module is the key to model design and implementation. Figure 3 shows the architecture of the sparse convolution module in the paper. Different from the normal convolution, the sparse convolution takes the mask into account. The input and output are both two parts: feature map and mask. As in the following equations, it only performs convolution on valid points to get the output, ignoring the invalid pixels. Moreover, the result is normalized by the number of valid points. To avoid division by zero, a small positive number is added to the denominator, which in our implementation is 1e-8. To track the output mask, the module uses a max-pooling layer to act on the input mask.

$$f_{u,v}(\boldsymbol{x}, \boldsymbol{o}) = \frac{\sum_{i,j=-k}^{k} o_{u+i,v+j}\, x_{u+i,v+j}\, w_{i,j}}{\sum_{i,j=-k}^{k} o_{u+i,v+j} + \epsilon} + b$$

$$f^{o}_{u,v}(\boldsymbol{o}) = \max_{i,j=-k,..,k} o_{u+i,v+j}$$



(b) **Sparse Convolution**

Figure 3: the sparse convolution module in the paper [1].

In our project, we designed the sparse convolution module based on the existing open-source code: https://github.com/chenxiaoyu523/Sparsity-Invariant-CNNs-pytorch. Due to the differences in data complexity and capacity, we design separate sparse convolution modules for the MNIST dataset and the Caltech-101 dataset. For MNIST, because the size of each input image is only 28*28 pixels, pooling is not required for each module. For Caltech-101, we add the batch normalization layer after the sparse convolution operation for input and add the 2 x 2 max-pooling layer after the activation function ReLU. The batch normalization helps to stabilize and accelerate the training process by reducing internal covariate shifts. The function of the pooling layer is to reduce the spatial dimensions (width and height) of the input images while retaining important information. For the sparse dataset, the max pooling type is suitable, since it can capture the maximum value, generally the valid information, in the operating region, which is beneficial for future learning and representations.

The sparse convolution is suitable for both single-channel input images and multi-channel input images. For multi-channel images, each of the three channels uses the same mask. So in terms of the Caltech-101 dataset, we generate one channel mask to perform on all channels using Pytorch broadcasting.

2.3 Model architecture

The model architectures for both the MNIST and Caltech-101 datasets are illustrated in Figures 4 and 5. For the MNIST dataset, the model architecture includes simple convoluted layers and linear layers. The model architecture for Caltech-101, however, has introduced additional batch normalization layers, pooling layers, and dropout for linear layers.

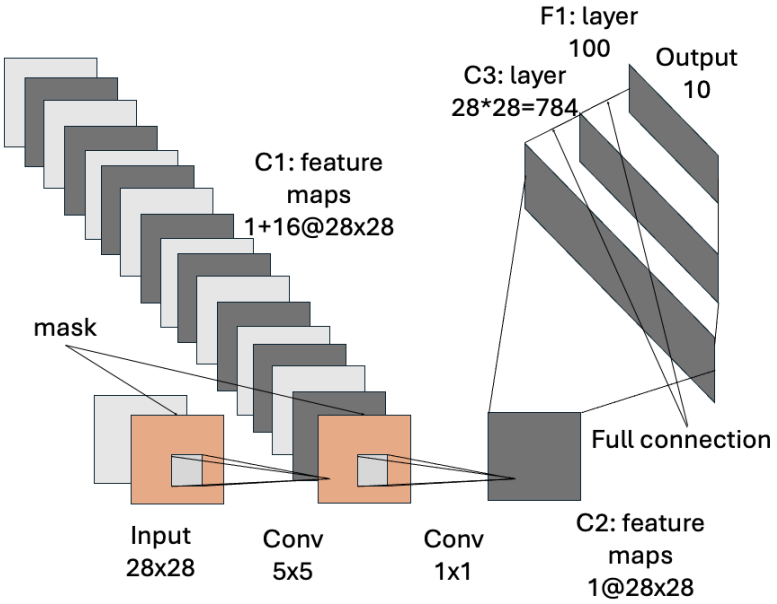


Figure 4: The Sparse CNN for the MNIST Dataset.

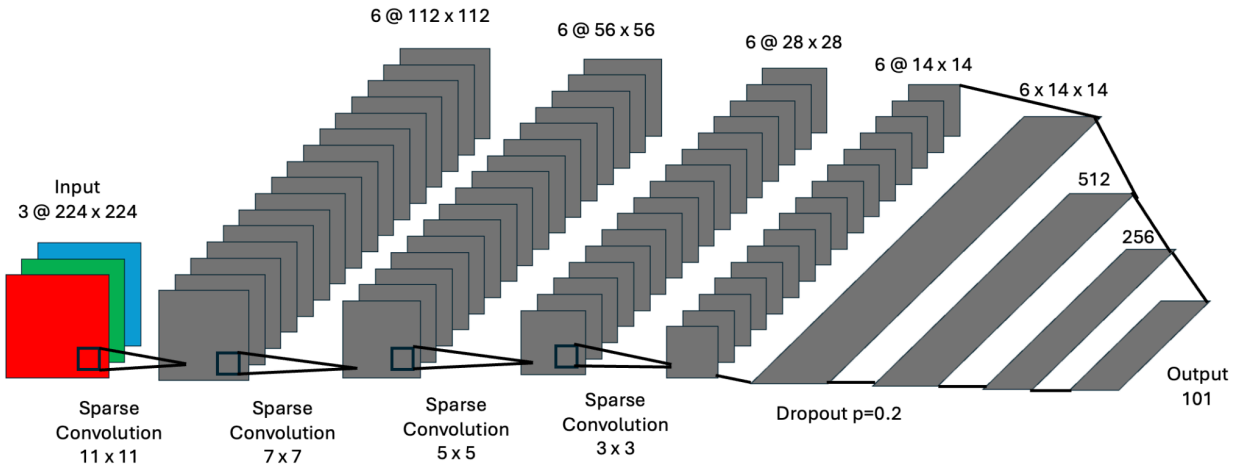


Figure 5: The Sparse CNN for Caltech-101 Dataset.

2.4 Training and Testing

1) MNIST

The MNIST is a simple but comprehensive collection of handwritten digits extensively used for training image processing systems and machine learning models. The model training is straightforward, and the hyperparameter settings for the training and test procedures are listed in Table 1 below.

Table 1: Hyperparameter settings for Training sparse ConvNet on MNIST.

Training Batch Size	Test Batch Size	Epoch	Learning Rate	Optimizer	Weight Decay
64	1000	10	5e-3	Adam	1e-3

2) Caltech-101

The Caltech101 data set is more complex than the MNIST data set in general. However, we still found that the original model from Jonas *et al.* cannot perform classification tasks well. We then started training and testing the model used on the MNIST data set and gradually increased the model complexity by increasing the number of layers. We found that a batch normalization layer boosts the model performance on both training and test data sets.

This indeed increased the performance, but we also discovered that the model became overfitting on data, with almost 100% accuracy on training data but lower than 20% on test data. We then added a dropout layer for the first linear layer and decreased the number of input channels in this layer. This actually works. After that, we began to decrease the learning rate and batch size. The final hyperparameters for the training procedure are illustrated in Table 2.

Table 2: Hyperparamters for Training sparse ConvNet on Caltech-101.

Learning Rate	Training Batch Size	Epoch	Weight Decay
1e-3	128	15	0.3

3. Result

Test accuracy

Table 3: Test Accuracy with Different Sparsity Levels of SparseConvNet and ConvNet for the MNIST dataset.

Sparsity	10%	20%	30%	40%	50%	60%	70%	80%	85%	90%	95%
Sparse ConvNet	96.57%	95.80%	94.14%	95.38%	94.59%	94.39%	92.77%	86.09%	84.61%	76.23%	56.81%
ConvNet	96.36%	95.72%	96.34%	95.41%	95.07%	94.17%	90.43%	85.32%	84.52%	69.27%	56.22%

Table 4: Test Accuracy with Different Sparsity Levels of SparseConvNet and ConvNet for the Caltech-101 dataset.

Sparsity	5%	10%	20%	30%	40%	50%	60%	70%
Sparse ConvNet	62.80%	61.98%	61.29%	62.68%	60.65%	60.94%	59.32%	59.26%
ConvNet	43.93%	41.90%	40.34%	39.18%	38.83%	38.07%	38.60%	37.84%

The results of test accuracy for both the MNIST and Caltech101 datasets are illustrated in Tables 3 and 4. Based on the same number of epochs for training, we test the classification performance of Sparse ConvNet and ConvNet on images with different sparsity levels. For the

MNIST dataset, it can be shown in Table 3 that the implementation of sparse ConvNet doesn't show notable priority compared to conventional ConvNet, where both networks' testing accuracy dropped with the increase in sparsity levels. In contrast, regarding the Caltech-101 dataset, it can be seen that for all the sparse levels, Sparse ConvNet performs better. Also worth noticing is that as the sparsity level increases, the accuracy of ConvNet decreases with each increase, but that of Sparse ConvNet is stable, which is consistent with the paper.

Hyperparameter Tuning

Table 5: Test accuracy of the 10th epoch with different hyperparameters on MNIST

Sparsity	10%			90%		
Batch Size	64	128	256	64	128	256
LR = 1e-2	97.25%	96.97%	96.92%	78.12%	79.57%	79.20%
LR = 5e-3	97.38%	97.12%	97.88%	79.54%	78.77%	79.44%
LR = 1e-3	97.57%	97.59%	97.34%	79.36%	79.23%	78.54%

Table 6: Test accuracy of the 15th epoch with different hyperparameters on Caltech-101

Sparsity	5%			70%			
Batch Size	64	128	256	64	64, the 14th epoch	128	256
LR = 1e-2	48.61%	52.61%	50.12%	47.80%	-	46.11%	45.24%
LR = 1e-3	64.21%	63.69%	62.53%	43.85%	56.26%	54.93%	53.25%
LR = 1e-4	60.38%	58.18%	54.70%	50.70%	-	48.03%	45.48%

As Table 5 shows, the batch size and the learning rate do not significantly influence the test accuracy of the MNIST dataset. For a higher training speed, we choose a batch size of 64.

Table 6 shows the test accuracy of the model in the last epoch with different hyperparameters and sparsity levels on Caltech-101. The learning rate 1e-3 has the most suitable order of magnitude, so we will only discuss the batch size in the following.

When we test batch size 64 on sparsity level 70% caltech-101 datasets, the final epoch outputs worse results than previous epochs, e.g., the 10th epoch, in not only the test accuracy (43.85% < 56.26%), but also the training accuracy (94.29% < 100.00%) and the validation accuracy (44.16% < 56.71%). This implies the decrease in test accuracy may not be caused by overfitting on the whole training set (otherwise the training accuracy could not have fallen). Instead, the small batch size seems sensitive to the randomness of picking training and validation sets and can worsen the model under some circumstances. In other words, it overfitted the chosen random training batch. As a result, though the test accuracy of the model with the best

validation accuracy seems good, we do not select 64 as the batch size because of its inconsistency.

4. Discussion

Model Complexity
Although in this project the model was not tested on the data set from the paper, the results from the MNIST and Caltech101 data sets have illustrated that the model for the latter data set is of much higher complexity than that tested on the former data set.

Sparse Convolution Layer Increases Robustness
From Table 4, it can be seen that sparse ConvNet is more robust to changes in sparsity level. The possible explanation is that the sparse mask deletes the “unnecessary” weights in a convolution mask, though the sparsity patterns are not the same.

MNIST Dataset Limitations
Notably, it can be observed in Table 3 that the choices of either sparse ConvNet or normal ConvNet do not necessarily affect the classification performance for the MNIST dataset. In other words, the magnitude of the decrease in test accuracy with increasing sparsity levels is essentially the same for both networks. One possible reason is that the MNIST dataset is too simple and originally seen as sparse input because of the large amount of black pixels in the background. Such bias makes it negligible to conduct sparse operations on this dataset when the sparse level is relatively low. This explanation can also be partially supported by the fact that, under a 70% sparsity level, the testing accuracy of both networks on the MNIST dataset remains above 90% while being significantly differentiated on the Caltech-101 dataset (59.26% and 37.84% for Sparse ConvNet and ConvNet, respectively). This indicates that the MNIST dataset is not a suitable candidate for the task. The targeting dataset is expected to be more complex in nature, such as Caltech-101, CIFAR-10, and CIFAR-100.

5. Conclusion

In conclusion, two Sparse-Convolution-layer based models with different structures are built to perform classification tasks on MNIST and Caltech101 data sets, respectively. Based on the results from the two data sets, the sparse convolution layer increases the robustness of the model, and this effect can be more obvious for more complex data sets. Although for the image classification task, the performance improvement of the sparse convolution is not as significant as the depth upsampling task in the paper, the project upholds the conclusion in the paper that sparse convolution network generalizes well to novel datasets and is invariant to the level of sparsity in the data.

Reference

[1] Uhrig J, Schneider N, Schneider L, et al. Sparsity invariant cnns[C]//2017 international conference on 3D Vision (3DV). IEEE, 2017: 11-20.
[2] https://github.com/chenxiaoyu523/Sparsity-Invariant-CNNs-pytorch