Sparsity Invariant CNNs Reproducibility on classification

COURSE NAME

CS4240 Deep Learning

Authors:

Zhixuan Ge (5927633)
Jiarui Zhou (6017002)
Junchi Liu (5983533)

April 14, 2024

Contents

Overview

Our code and blog: https://github.com/RangerCoF/SparseConv
Reproduced paper: https://arxiv.org/abs/1708.06500

1 Introduction

In our project, we aim to reproduce the sparse convolution proposed in Sparsity Invariant CNNs. The original work was done on depth dateset. It proved traditional convolution networks performed poorly when the input had missing pixels, and the proposed sparse convolution was invariant to the level of sparsity. We reproduce the sparse convolution on a different job, classification, with dataset MNIST and Caltech 101. However, the result is not as satisfying as the original work.

2 Convolutional Neural Networks, CNN

A typical CNN model architecture is shown as:

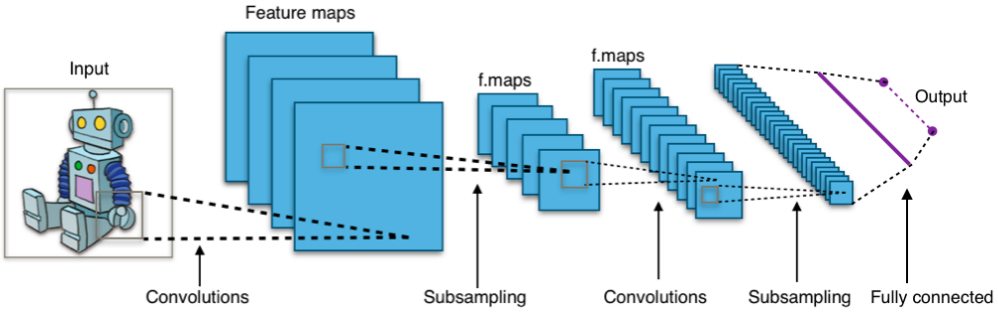


Figure 1: Traditional CNNs

A convolutional neural network (CNN) comprises an input layer, hidden layers, and an output layer, every input feature is connected to every output feature through a convolution operation.[4] [5] Within the hidden layers are one or more convolutional layers, where each layer conducts convolutions. Typically, this involves computing the dot product of a convolution kernel with the input matrix using the Frobenius inner product. ReLU is commonly employed as the activation function for these operations. As the convolution kernel traverses the input matrix, it generates a feature map, contributing to the subsequent layer's input. Additional layers like pooling, fully connected, and normalization layers follow.

3 Sparse Convolutions

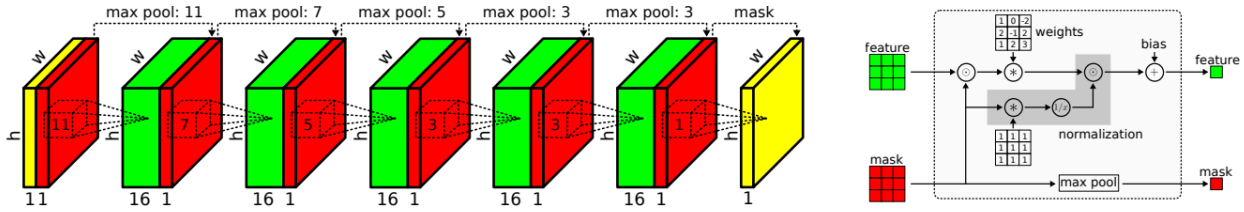
In sparse convolutions, the locations of missing points are known. They are saved as a mask binary matrix $\mathbf{o}$. An observed pixel on location (u, v) is represented as $o_{u,v} = 1$, otherwise, $o_{u,v} = 0$ for the missing pixel. [3] The sparse image input is denoted by $\mathbf{x}$. The image $\mathbf{x}$ with sparsity and the corresponding mask matrix $\mathbf{o}$ compose the actual input of the network. The sparse convolution can be represented as

$$f_{u,v}(\mathbf{x}, \mathbf{o}) = \frac{\sum_{i,j=-k}^{k} o_{u+i,v+j} x_{u+i,v+j} w_{i,j}}{\sum_{i,j=-k}^{k} o_{u+i,v+j} + \epsilon} + b \tag{1}$$

in which $\mathbf{w}$ is the convolution kernel. Compared with traditional convolution

$$f_{u,v}(\mathbf{x}, \mathbf{o}) = \sum_{i,j=-k}^{k} x_{u+i,v+j} w_{i,j} + b \tag{2}$$

only observed points are calculated in the sparse convolution by multiplying $\mathbf{x}$ and $\mathbf{o}$ element-wisely. Then, the result of the weighted sum is normalized by the number of non-sparse entries in $\mathbf{o}$ under the convolution kernel. A very small number ϵ is added to the denominator to prevent the 0 divided by 0 case when all entries are 0. The bias b is added at the end. When the size of feature maps changes, the mask is also max pooled with the same kernel size and stride to keep track of the visibility state and maintain the same shape as the image.



(a) **Network Architecture** (b) **Sparse Convolution**

Figure 2: Architecture

4 Our Work

Sparse convolutions can be realized by quotient of 2 unbiased convolutions. The numerator in Equation 1 is the unbiased convolution of $\mathbf{o} \odot \mathbf{x}$ with learnable kernels. The denominator is the unbiased convolution of $\mathbf{o}$ with all one kernels. After division, learnable parameters $\mathbf{b}$ are added.

We found some existing implementations on GitHub, including Pytorch and Tensorflow. However, we noticed that these repositories create an extra convolution layer with the same number of channels as the nominator for the denominator and fix all weights to 1. This strategy is correct, but it doubles the number of parameters in the convolution layer and most of them have little use. Besides, there are $\texttt{in\_channels} \times \texttt{out\_channels}$ times of convolution of $\mathbf{o}$ with all one kernel in forward propagation. To optimize this, we rewrite the sparse convolution from scratch, simplifying the denominator with one input channel and one output channel. Then, the denominator can be expanded to the same dimension as the nominator via broadcasting in the division.

In our project, we tested SpareseConv on MNIST and Caltech 101. The architecture is based on the classification test. A regular convolution net was built first. Then we replaced it with sparse convolutions. We will introduce them separately.

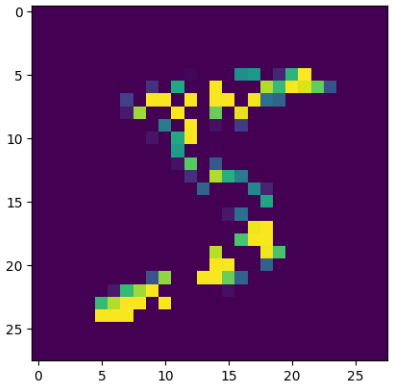


Figure 3: Sparse MNIST ($p = 50\%$)

4.1 MNIST

We import MNIST from Torchvision. There are 60000 handwriting digits in the training set and 10000 in the test set. All of them are 28×28 and grayscale. In a regular ConvNet for sparse images, sparse pixels are set to a default value, usually 0. Also, in SparseConvNet, $o_{i,j}x_{i,j}$ equals 0 when $o_{i,j}$ is 0. Hence, we choose to scale the image to the range $[0, 1]$, ensuring that sparse values are the lowest of all entries.

To realize sparsity, we create a mask matrix sharing the size with the image. All entries of the mask are independent and identically distributed and are under a Bernoulli distribution. So each entry has a possibility of p to be 0. When creating a dataset, we generate a mask for each image. Only the element-wise product of the mask and the image is solved, rather than the non-sparse image, to avoid access to original images.

4.2 Architecture

MNIST is a simple dataset, so a simple convolution net is enough. We use 2 convolution layers and 3 fully connected layers. Parameters can be seen in Table 1. Same padding is used in the convolution to keep the size of the image. Because we will replace convolutions with sparse convolutions later, it is necessary to hold

the image size. Otherwise, the size of the images does not match that of the masks. We can solve this problem by adding an extra maxpooling layer for the mask, with the same pooling kernel size and the stride, but this is much more complicated and not tested in [6]. Hence, we just maintain the output size.

We also normalize the outputs of the convolution layer by BatchNorm and use ReLU as the activation function after normalization. Though BatchNorm seems unnecessary for such a simple job, but it is also very useful in the SpareConvNet, so we keep it to control variables. We find a SpareConvNet without BatchNorm cannot learn anything from the batch. In this case, the loss remains unchanged and the accuracy is always 10%, which is the accuracy of random guesses from 10 uniformly distributed answers. It can be attributed to the denominator in sparse convolutions. Take non-sparse inputs with $p = 0$ as an example, the kernel size of convolutions is 5×5, so the results are always divided by a factor of 25, which indicates gradient vanishing. Normalization solves the problem in SparseConvNet.

We train 2 epochs on the dataset and test it on the testset. The accuracy is 98%, indicating this model is already well-trained. Hence, we use epoch=2 for the latter experiments.

Table 1: Model for MNIST architecture

Layer	kernel	stride
Conv	[5,5]$\times$ 6	1
Maxpooling	[2,2]	2
Conv	[5,5]$\times$16	1
Maxpooling	[2,2]	2
Linear	120	
Linear	84	
Linear	10	

4.3 SparseConvNet

We replace convolutions with sparse convolutions. Results are shown in Figure 5. With BatchNorm, SpareseConvNet performs as well as ConvNet when $p = 0$. It makes sense because the 2 nets are equivalent when the images are non-sparse. However, when sparsity increases, SparseConvNet does not perform better than the regular one. Instead, the accuracy is lower when sparsity is large. Another interesting fact is models for classification seem not sensitive to sparsity. When sparsity is 0, the accuracy is 98%, but it only drops to 91% when the sparsity reaches 80%. The digit 7 with sparsity 80% is shown in Figure 4. We can still recognize the number from human eyes. One possible explanation is the model can still extract enough features similarly for classification in this case, as long as receptive fields are large enough. But we are not certain about it. Hence, we apply SparseConvNet on Caltech 101 to check the results on another dataset.

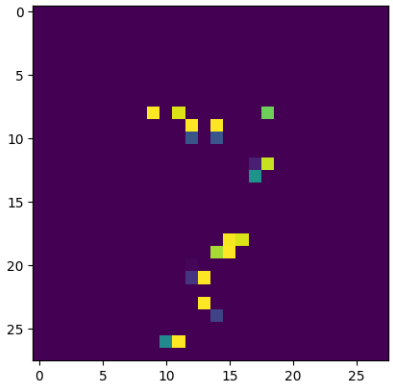


Figure 4: Sparse MNIST ($p = 80\%$)

4.4 Caltech 101

Caltech 101 contains pictures of objects belonging to 101 categories.[1] About 40 to 800 images per category. Most categories have about 50 images. The images are resized to 224×224. Due the default value of sparse pixels, we rescale them into $[0, 1]$ too. (We also tried the popular normalization for RGB images by mean $= [0.485, 0.456, 0.406]$, std $= [0.229, 0.224, 0.225]$. The performance is similar.) The mask is the same size

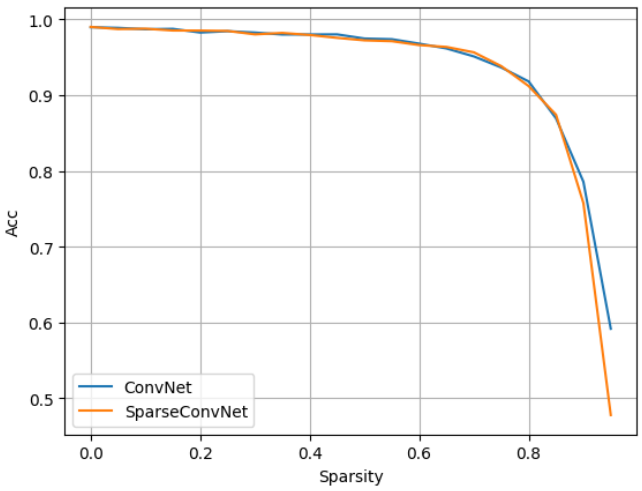


Figure 5: MNIST acc

as the image but has only one channel. The 3 channels of the image are multiplied by the same mask, so the locations of sparse points are the same for R, G and B channels. Figure 7 is an image with sparsity 30%. 80% of the images are split into the training set, and the rest are the testset.

4.4.1 Resnet18

For the architecture used for classification on Caltech 101, we first tried the well designed and famous network - Resnet[2]. We choose Resnet18 as our baseline, which consists of 18 layers. We implement the network architecture with normal convolutions and then replace the 2 convolution layers in the residual blocks with sparse convolution layers. Because there are residual blocks doubling channels and halving the size, in which skip connections are realized by convolutions with kernel size 1 and stride 2. We replace such structure with MaxPooling and sparse convolutions with kernel size 1 and stride 1.

Figure 6 shows the classification accuracy on Caltech 101 dataset with normal and sparse Resnet. The stable accuracy for ConvNet is 75%. We can see the accuracy drops significantly when we replace the convolution blocks. If we increase epochs, SparseConvNet can reach an accuracy of 50%, which is still far from ConvNet. We think this is because sparse convolutions are used twice in a residual block with the same mask. It can be assumed that sparsity has been fixed after one time of sparse convolution. Otherwise, the mask and the image are downsampled to get a less sparse version to get further sparsity fixation. However, the output of sparse convolution is convoluted again with the same mask, adding extra errors to the output.

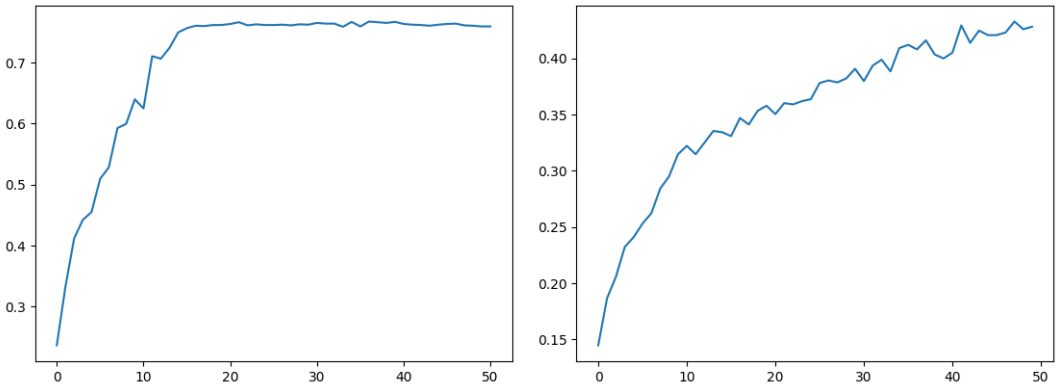


Figure 6: Resnet performance on Caltect 101 (sparsity = 0). Left: ConvNet, Right: SparseConvNet

4.4.2 Simple CNN

After the failure of Resnet18, we decided to focus on a simpler architecture. We choose to use the combination of convolution layers and fully connected layers. It is the deeper version of architecture in subsection 4.1. Structures are listed in Table 2. We train this architecture with 50 epochs and find it converges at epoch $= 10$.

Hence, we choose 10 as the number of epochs. Compared with Resnet18, the accuracy of our new model is relatively low. But our objective is to analyze how sparse convolution replacement can influence results, rather than higher accuracy, so it is reasonable to select the architecture that fulfills an acceptable accuracy. The optimizer is Adam with default parameters. In the first 5 epochs learning rate is $1e-3$, and it is set to $1e-4$ in the next 5 epochs.

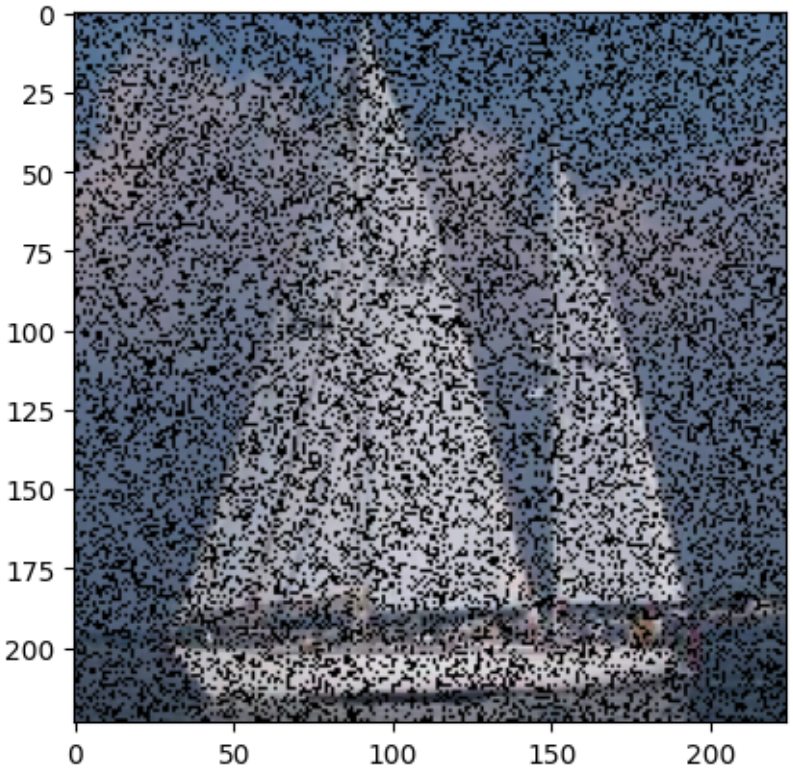


Figure 7: Sparse Caltech 101 $(p = 50\%)$

Table 2: Model for Caltech 101 architecture

Layer	kernel	stride
Conv	[7,7]*6	1
Maxpooling	[2,2]	2
Conv	[5,5]*16	1
Maxpooling	[2,2]	2
Conv	[3,3]*16	1
Maxpooling	[2,2]	2
Conv	[3,3]*16	1
Maxpooling	[2,2]	2
Linear	240	
Linear	120	
Linear	101	

The results are shown in Figure 9. Due to randomness, there is some oscillation in the accuracy curve, but it can still be seen that the performance of SparseConvNet does not exceed that of ConvNet. Besides, the accuracy drops about 13% when the sparsity increases from 0 to 80%. The decrease is higher than MNIST, but considering the sparsity of 80%, 55% is still a fine result for 101 categories. Figure 10 is an image with sparsity 80%. We can still find patterns indicating it is a ship. Hence, we can still conclude that convolution classification is not sensitive to sparsity.

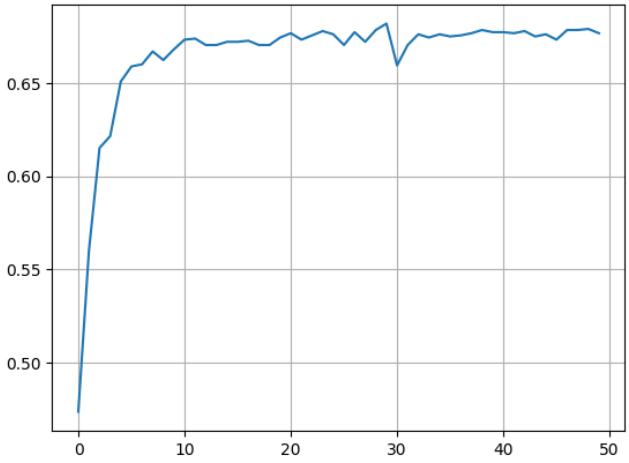


Figure 8: Accuracy of ConvNet on non-sparse dataset vs epochs

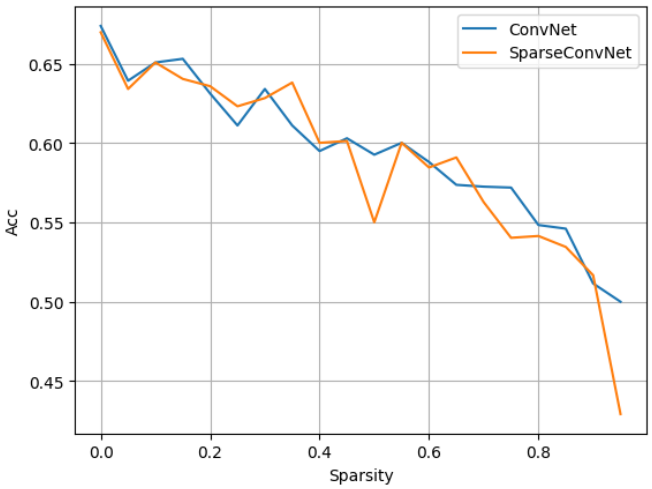


Figure 9: Accuracy of ConvNet and SparseConvNet with different sparsity levels

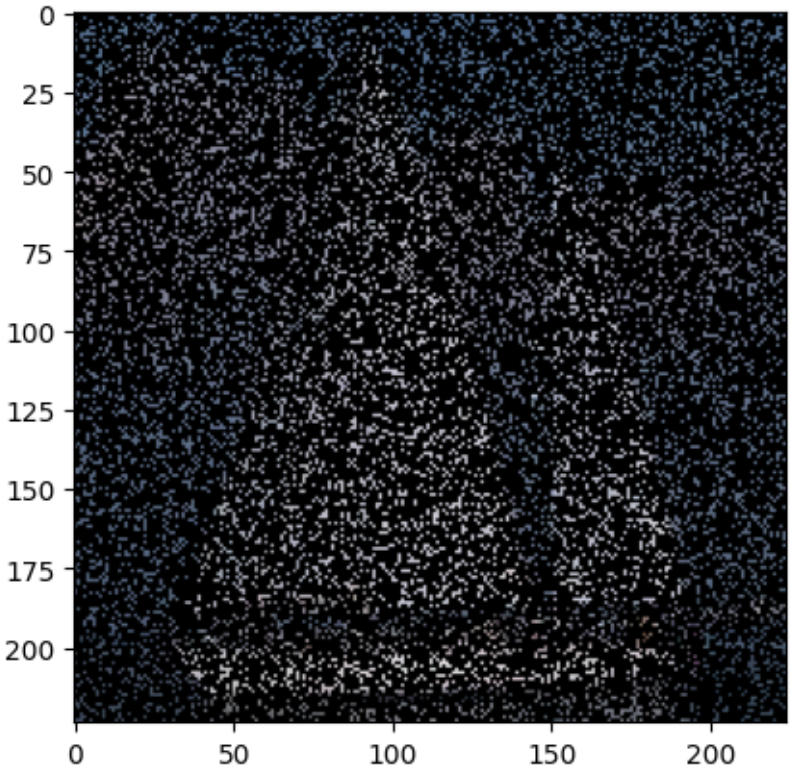


Figure 10: Sparse Caltech 101 $(p = 80\%)$

5 Discussion

If we compare the task of [6] and our work, we can find reasons for our results. In the original work, sparse convolutions are applied to depth maps and MAE is used to evaluate results. We also checked citations of this paper and found almost all of them were about depth maps. By contrast, our task is a classification and we use accuracy. In MAE, the result of each pixel matters, because it contributes to the total MAE and errors can be accumulated. However, in classification, our model just needs to make a decision between 10 or 101 choices. It extracts available features and makes the most likely selection. Sparsity can ruin some patterns, especially in a small neighborhood, but as long as the receptive fields are large enough, the net can always catch some features. Then, it does not need to calculate errors of each point or feature. Rather, existence of pattern combinations predicting an object is enough to make the decision. In general, calculating exact values of abundant values is hard, while making decisions is easy. That is why the model in our project is not sensitive to sparsity. Based on that, the sum of the mask entries in sparse convolutions might introduce interference. The normalizing of the mask may fix huge data errors in depth maps, but it interferes with feature extraction in classification. Table 3 and Table 4 are the reproduces of Table 3 in [6].

Table 3: Performance comparison (Accuracy) of ConvNet and SparseConvNet on different sparsity levels of MNIST

Sparsity	5%	10%	20%	30%	40%	50%	60%	70%	80%	90%
ConvNet	98.88%	98.70%	98.26%	98.26%	98.01%	97.47%	96.80%	95.11%	91.81%	78.58%
SparseConvNet	98.72%	98.77%	98.52%	98.02%	97.94%	97.23%	96.61%	95.65%	91.20%	75.77%

Table 4: Performance comparison (Accuracy) of ConvNet and SparseConvNet on different sparsity levels of Caltech 101

Sparsity	5%	10%	20%	30%	40%	50%	60%	70%	80%	90%
ConvNet	63.94%	65.09%	63.13%	63.42%	59.50%	59.27%	58.81%	57.26%	54.84%	51.15%
SparseConvNet	63.42%	65.09%	63.59%	62.85%	60.02%	55.01%	58.47%	56.28%	54.15%	51.67%

References

[1] Bansal, M., Kumar, M., Sachdeva, M., and Mittal, A. (2023). Transfer learning for image classification using vgg19: Caltech-101 image data set. *Journal of ambient intelligence and humanized computing*, pages 1–12.

[2] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[3] Liu, B., Wang, M., Foroosh, H., Tappen, M., and Pensky, M. (2015). Sparse convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 806–814.

[4] Sarraf, S. and Tofighi, G. (2016). Classification of alzheimer's disease structural mri data by deep learning convolutional neural networks. *arXiv preprint arXiv:1607.06583*.

[5] Stanković, L. and Mandic, D. (2023). Convolutional neural networks demystified: A matched filtering perspective-based tutorial. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*.

[6] Uhrig, J., Schneider, N., Schneider, L., Franke, U., Brox, T., and Geiger, A. (2017). Sparsity invariant cnns. In *2017 international conference on 3D Vision (3DV)*, pages 11–20. IEEE.