CS420: Machine Learning Project Report

Zhiwen Qiang, 515030910367 Leqi Zhu, 515020910272 Yulun Wu, 5140719008

I. Project Description

THIS project is about building a classifier for a modified version of MNIST data. The dataset is consist of two parts, the train data_label and test data_label. Each data is a 45*45 array and the value is [0,255], which means the single-channel color. Each label is a number in [0,9], representing the exact number in the image. The train size is 60000 and the test size is 10000.

For the data itself, we can find that for each image, there is one main number and several other spots and impurities, while the remaining space is in black. Our task is to decide which number is in the image, avoiding the influence of other spots.

So one natural thought is to get rid of these disturbance term and just pick the numbers out and then conduct the classification job. Actually it's the way we finished this project. We first using pretreatment to exclude the disturbance term and then trying to use traditional and deep learning methods to conduct the classification task.

Our project code can be downloaded in:
$https://github.com/QLightman/mnist-classifiers/$.

II. Pretreatment

The project's goal is to build classifiers for a modified version of MNIST data. Following the problem oriented train of thought, pretreatment is of great importance since the data is very noisy and the size of each figure can vary. In this project, our pretreatment is mainly threeflod:

A. Noise Reduction and Size Adjustment

Here we implemented the method to reduce the noise and adjust the size of each image. The algorithm detail is shown in **Alg. 1**. The reason why we choose the two parameters as 0.45% and $(20, 20)$ are that after many controlled experiments, we find that these two can achieve the best results among others.

Part of the output images arer shown in **Fig. 1**. Here we can see that the result successfully reduced the noise and maintained the shape of the figure at the same time.

Algorithm 1 Noise Reduction and Size Adjustment

Input: Original images I of size $60000 * 45 * 45$.
Output: Images S of size $60000 * 20 * 20$

```
for each image i in I do
    Calculating connected component in i
    index=1
    while the index largetset connected component of i is smaller than the 45% of the whole do
        index++
        Add the index largest connected component.
    end while
end for
for each image i in I do
    Cutting the margin of i.
    Adding black margin to make i a square. Resize the i to (20 * 20).
end for
```

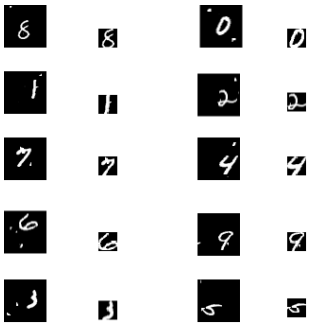


Fig. 1: Results of pretreatment. The larger one is the original image, the smaller one is its corresponding pretreatment image.

B. Histogram of Oriented Gradients

We implemented the Histogram of Oriented Gradients (HOG) method, which is a feature descriptor used in computer vision and image processing for the purpose of object detection. The algorithm mainly includes the following steps:

- **Gradient computation**
  Computation of the gradient values.
- **Orientation binning**
  Creating the cell histograms. In this experiment, we use unsigned gradients in conjunction with 9 histogram channels.
- **Descriptor blocks**
  Ggrouping the cells together into larger, spatially connected blocks so that the gradient strengths can be locally normalized to account for changes in illumination and contrast.

Fig. 2 is the general view of how HOG works. It is worth noting that after HOG process, the size of the image is $18*18$, so it can be viewed as a dimensionality reduction algorithm as well.

C. Data Augmentation

We also tried the method of data augmentation, which is to let each image randomly rotate a small angle, in this experiment, we use 15° as its maximum rotating angle. **Fig. 3**

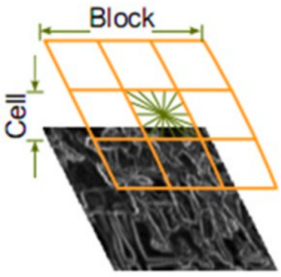


Fig. 2: General view of HOG's precedure.

are part of the results we obtained. We can see that the output image rotates a small angle compared to the original image.

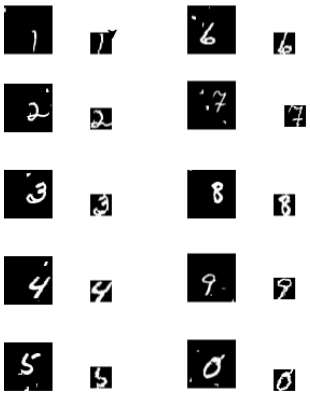
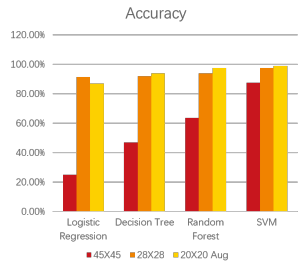


Fig. 3: Results of Data Augmentation. The larger one is the original image, the smaller one is its corresponding rotate image.

III. Traditional Methods

In this section we examine the performance of traditional classifying methods on both the orginal data and our pretreated dataset. A brief accuracy comparison of different tradtional algorithms on the original 45×45 dataset, the official 28×28 dataset and the pretreated 20×20 dataset with augmentation is shown in Figure 4.

While the result showed a significant improvement on accracy with the pretreated dataset, it also demonstrated that



(a) Figure

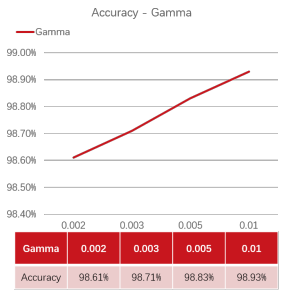
	Logistic	Decision Tree	Random Forest	SVM
45X45	25.12%	46.9%	63.62%	87.64%
28X28	91.37%	91.85%	93.8%	97.61%
20X20 w/ Aug	87.05%	93.99%	97.5%	98.87%

(b) Data table

Fig. 4: Comparison of Traditional Algorithms on Differently Treated Datasets

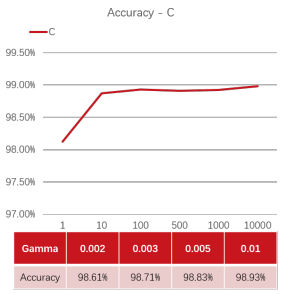
SVM is the best fit for the MNIST dataset among traditional methods.

Thus, we focused on SVM as the traditional method of our interest, and tried to obtain the best performance by testing the effect of different parameters. We varied Gamma(for RBF), C and kernel functions in our experiment. The former two decides SVM's judgement and tolerance of the error, respectively; kernel determines the mapping space. The result is shown in Figure 5



Gamma	0.002	0.003	0.005	0.01
Accuracy	98.61%	98.71%	98.83%	98.93%

(a) Accuracy - Gamma



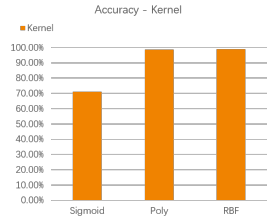
Gamma	0.002	0.003	0.005	0.01
Accuracy	98.61%	98.71%	98.83%	98.93%

(b) Accuracy - C

Fig. 5: Performance examination of SVM by varying Gamma and C

From the result, we saw a strong positive correlation between classification accuracy and Gamma value, because SVM judges the distance between two data points more strictly in the mapping space of RBF as Gamma goes up. Note that higher Gamma could also result in the non-convergence of the algorithm.

As for parameter C, in theory the curve should be of an arch shape. It is easy to comprehend that if C is too small, i.e. the tolerance towards the error is too loose, the accuracy will fall. But if C is too big, it may also causes over-fitting problem. We found no evidence of the latter incidence in our experiment, possibly due to the fact that MNIST dataset is relatively large and irregular, also our pretreatment method of data augmentation reduced the chance of over-fitting from occuring. The result shows that our pretreatment methods balanced the accuracy and over-fitting problem really well.

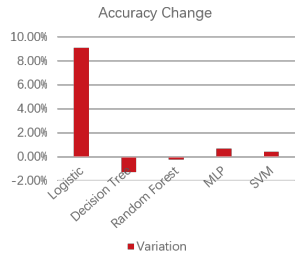


Kernel	Sigmoid	Poly	RBF	Linear
Accuracy	70.97%	98.76%	98.93%	-

Fig. 6: Performance examination of SVM by varying kernel functions

As for the kernel, linear SVM ended up with no outcome after approximately 10 hours, showing that the dataset is most possibly not linear separable. The other three algorithms all converged, among which poly and RBF performed decently.

However, there was still one problem we had the urge to address, that the straightened 0-1 dataset failed to capture the spacial structure of the original images, which is actually a very important information in deciding what numbers they are. To address this, we treated the 20×20 dataset with HOG. The effect of this modification on SVM and other algorithms is shown in Figure 7.



(a) Figure

	Logistic	Decision Tree	Random Forest	MLP
20X20	87.05%	93.99%	97.5%	98.48%
20X20 w/ HOG	96.11%	92.69%	97.27%	99.16%

SVM	Gamma= 0.01	Gamma= 0.1	Kernel = linear
20X20	98.93% 34m39s	- >10h	- >10h
20X20 w/ HOG	98.56% 7m21s	99.33% 7min33s	97.55% 17m2s

(b) Table

Fig. 7: Examination on the effect of HOG

While not so significant on other algorithms, the effect of HOG on linear algorithms is remarkable. It greatly improved Logistic regression, and made linear kernel viable for SVM. For SVM, it also increased the efficiency by a very large margin, and made it possible to raise up Gamma value. We finally yield an accuracy of 99.33% on the HOG treated data.

Besides the elimination of noises by dimensional reduction, this drastic improvement is also caused by that, as mentioned earlier, it keeps a part of the spatial structure information in the dataset by capturing the second-order interactions between local pixels(see [1]).

IV. Deep Learning Methods

In this section, we tried to use deep learning methods in this task. We first introduce a simple implement of Convolutional Neural Network (CNN). Then we tried to use some recently popular deep neural network designs to conduct the classic MNIST classification problem.

A. Convolutional Neural Network(CNN)

1) Algorithm Introduction:

- **Convolutional Neural Network (CNN)** is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units.[2]

- **Dropout.** Because a fully connected layer occupies most of the parameters, it is prone to overfitting. One method to reduce overfitting is dropout. At each training stage, individual nodes are either "dropped out" of the net with probability $1-p$ or kept with probability$p$, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed. Only the reduced network is trained on the data in that stage. The removed nodes are then reinserted into the network with their original weights.[3]. By avoiding training all nodes on all training data, dropout decreases overfitting. The method also significantly improves training speed. This makes model combination practical, even for deep neural nets. The technique seems to reduce node interactions, leading them to learn more robust features that better generalize to new data.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

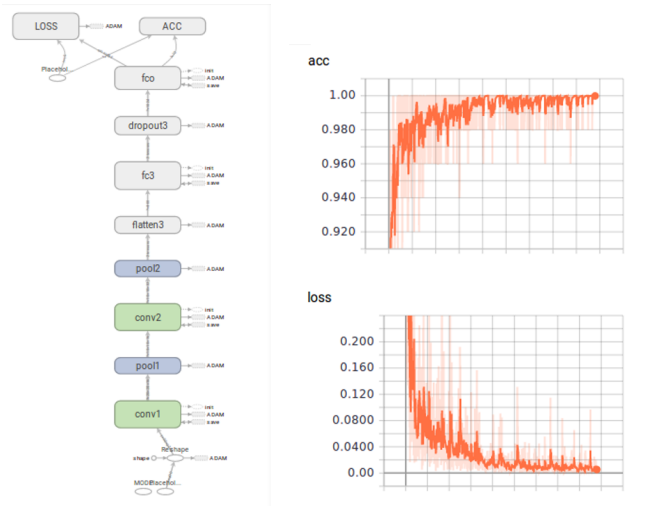
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Fig. 8: The Algorithm of Batch Normalization.

- **Batch Normalization** Training Deep Neural Networks is complicated by the fact that the distribution of each layers inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. This phenomenon is called internal covariate shift, and address the problem by normalizing layer inputs. Batch Normalization draws its strength from making normalization a part of the model architecture and performing the normalization for each training mini-batch. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout.[4]

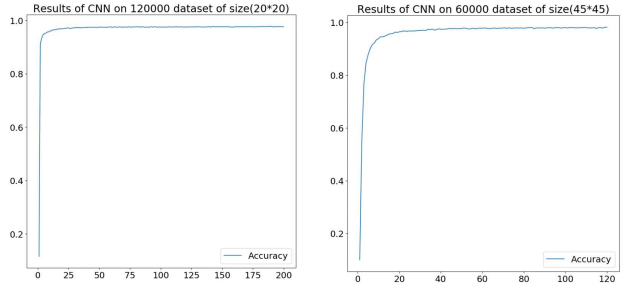
2) Network structure: Our CNN model is consist of 2 convolution layers, two pooling layers, one fully-connected layer and the input/output layer. All the neural network layers use batch normalization.

- Input layer: (20*20*120000 or 45*45*600000)
- First convolution layer: kernel[5,5], channel 32.
- First pooling layer
- Second convolution layer: kernel[5,5], channel 48.
- Second pooling layer
- Full-connected layer, equipped with dropout.
- Output layer.



(a) The structure of CNN (b) The accuracy and loss while training

Fig. 9: The structure of our model and the parameters while training.



(a) pretreated Dataset (b) Original Dataset

Fig. 10: The validation accuracy of CNN in pretreated dataset and original dataset respectively.

3) Experimental Results: We conduct the experiment on the original 45*45*600000 dataset and pretreated 20*20*120000 dataset. The test accuracy of pretreated dataset is **99.33%** and **98.04%** for original dataset. As we can see, the simple CNN model works quite well in both datasets. Unlike the traditional methods, CNN can have a high performance with original dataset.

B. Capsule Network

1) Algorithm Introduction: In this part we tried to use the idea of Hinton's Capsule Network [5]. in our project. A Capsule Neural Network (CapsNet) is a machine learning system that is a type of artificial neural network (ANN) that can be used to better model hierarchical relationships. The approach is an attempt to more closely mimic biological neural organization.

The idea is to add structures called capsules to a convolutional neural network (CNN), and to reuse output from several of those capsules to form more stable (with respect to various perturbations) representations for higher order capsules. The output is a vector consisting of the probability of an observation, and a pose for that observation. This vector is similar to what is done for example when doing classification with localization in CNNs.

A function called squashing function is used to ensure the short vector get shrunk to almost zero length and long vectors get shrunk to a length slightly below 1 while keep the direction the same.

$$v_j = \frac{\| s_j \|^2}{1+\| s_j \|^2} \frac{s_j}{\| s_j \|}$$

The loss function used in CapsNet is the Margin loss L_k.:

$$L_k = T_k \, max(0, m^+ - \| v_k \|)^2 + \lambda(1 - T_k) max(0, \| v_k \| - m^-)^2$$

2) Advantages: From the definition and the calculating concepts of CapsNet, combined with some experiment results, we can conclude the advantages of it by:

- Requires less training data
- Position and pose information are preserved.
- Routing by agreement is great for overlapping objects.
- Capsule activations nicely map the hierarchy of parts
- Offers robustness to affine transformations
- Activation vectors are easier to interpret

3) Network Constructure: A simple model is shown in 11 introduced by [5]. The architecture is shallow with only two convolutional layers and one fully connected layer. Conv1 has 256, 9×9 convolution kernels with a stride of 1 and ReLU activation. This layer converts pixel intensities to the activities of local feature detectors that are then used as inputs to the primary capsules.

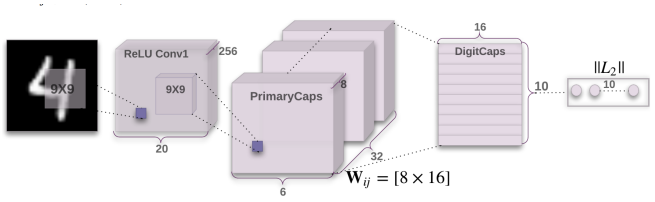
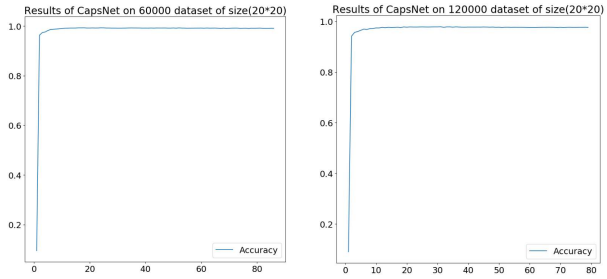


Fig. 11: A simple CapsNet with 3 layers.

The primary capsules are the lowest level of multi-dimensional entities and, from an inverse graphics perspective, activating the primary capsules corresponds to inverting the rendering process. This is a very different type of computation than piecing instantiated parts together to make familiar wholes. The second layer (PrimaryCapsules) is a convolutional capsule layer with 32 channels of convolutional 8D capsules (i.e. each primary capsule contains 8 convolutional units with a 9×9 kernel and a stride of 2). Each primary capsule output sees the outputs of all 256×81 Conv1 units whose receptive fields overlap with the location of the center of the capsule. In total PrimaryCapsules has $[32 \times 6 \times 6]$ capsule outputs (each output is an 8D vector) and each capsule in the $[6 \times 6]$ grid is sharing their weights with each other. One can see PrimaryCapsules as a Convolution layer. The final Layer (DigitCaps) has one 16D capsule per digit class and each of



(a) 20*20*600000 Dataset (b) 20*20*120000 Dataset

Fig. 12: The validation accuracy of CapsNet in pretreated dataset and original dataset respectively.

these capsules receives input from all the capsules in the layer below.

The network we used is similar with the one discussed above. We use 20×20 image as input instead. The implement of CapsNet is finished in Tensorflow.

4) Experimental Results: Considering the complex network structure and resource consuming calculating when applying 45*45 dataset, we just apply the 20*20 data for 60000 and 120000. The test accuracy is **99.23%** and **99.28%** respectively. As we can see, the performance is almost the same as CNN model and the 120000 dataset is better than that of 60000.

C. DenseNets

1) Algorithm Introduction: As CNNs become increasingly deep, information about the input or gradient passes through many layers, it can vanish and "wash out" by the time it reaches the end (or beginning) of the network. Motivated by this research problem, the [6] proposed DenseNets, which can obtain significant improvements over the state-of-the-art on *CIFAR-10, CIFAR-100, SVHN, and ImageNet*. We believe that this kind of architecture can work well on modified MNIST data classification problem as well, so we studied, implemented it and evaulate its result in the following section. The features of DenseNets is that it connects each layer to every other layer in a feed-forward fashion. So this network with L layers has $\frac{L(L+1)}{2}$ direct connections. As is shown in Fig. 13p. For each the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers.

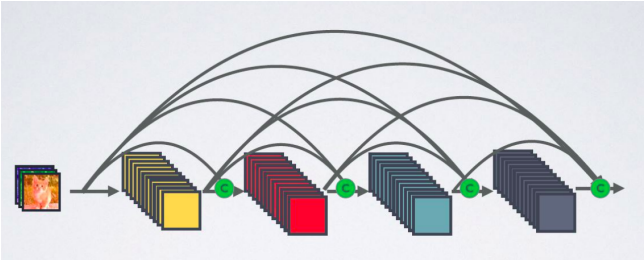


Fig. 13: General view of Dense Connectivity.

2) Advantages: The advantages of DenseNets are as follows:

- Alleviate the vanishing-gradient problem.
- Strengthen feature propagation.
- Encourage fea- ture reuse.
- Substantially reduce the number of parameters.

3) Structure: Fig. 14 shows the architecture of DenseNets, the input images first processed by a convolutional layer, then a dense block, a convolutional layer and a pooling layer, this process goes on until it meets the last dense block. The pooling layer can reduce the feature map sizes, while the feature map sizes match within each block.

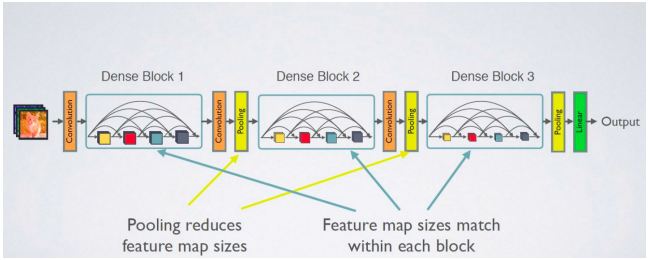
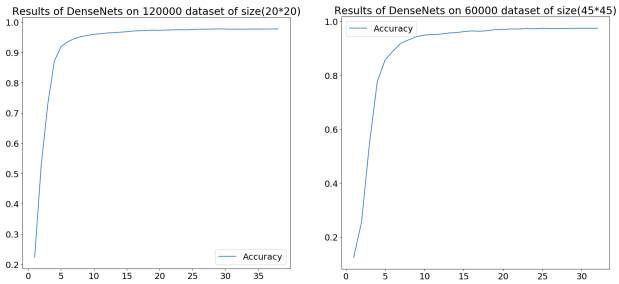


Fig. 14: Architecture of DenseNets we implemented.

4) Experimental Results: Fig. 15(a) is the results of DenseNets on 120000 pretreated dataset of size 20*20. The best accuracy is **97.76%**. Fig. 15(b) is the results of DenseNets on 60000 original modeified MNIST dataset of size 45*45. The best accuracy is **97.46%**.



(a) 20*20*600000 Dataset (b) 20*20*120000 Dataset

Fig. 15: The validation accuracy of CapsNet in pretreated dataset and original dataset respectively.

As we can see, the results obtained by DenseNets aren't promising conpared to the other deep learning methods. We think that bacause of the network architecture is very complex, it contians hundreds of layers, so there may exist some overfitting problem.

D. Comparing Three Deep Learning Results

As we can see in Table 1, these three deep learning methods all have good results in this classification task. Among them, CNN works best, CapsNet is slightly weaker than CNN and the DenseNet becomes last. Considering the complex structure of DenseNet and the relatively small dataset, we think it's probably owe to the overfitting problem.

Another phenomenon is that for the deep learning methods, the influence of pretreatment is not so huge as that of traditional methods. Without pretreatment, they can still have a acceptable accuracy, which might be one strength of them.

TABLE I: The results of Deep Learning Algorithms

Methods	***Test Accuracy***
CNN(20*20,12w)	0.9933
CNN(45*45,6w)	0.9804
CapsNet(20*20,12w)	0.9923
DenseNet(20*20,12w)	0.9776
DenseNet(45*45,6w)	0.9746

V. Conclusion

In this project, we first conduct pretreatment on the whole dataset, and then using traditional and deep learning network methods to finish the classification problem. The best accuracy of these methods is shown in Table 2. Among them, the SVM and CNN works best. From this we can know that with the help of data pretreatment, the traditional machine learning methods can have a rather satisfying result, which even might be better then some deep learning algorithms.

TABLE II: The Best Results of Each Algorithm

Methods	***Test Accuracy***
Logistic Regression	0.8705
Decision Tree	0.9399
Random Forest	0.9750
SVM	0.9933
CNN	0.9933
CapsNet	0.9923
DenseNet	0.9776

References

[1] H. Bristow and S. Lucey, "Why do linear svms trained on hog features perform so well?" *arXiv preprint arXiv:1406.2419*, 2014.

[2] http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/.

[3] https://en.wikipedia.org/wiki/Convolutional_neural_network#Dropout.

[4] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[5] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," *CoRR*, vol. abs/1710.09829, 2017. [Online]. Available: http://arxiv.org/abs/1710.09829

[6] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016. [Online]. Available: http://arxiv.org/abs/1608.06993