Room Cleanliness Detection Using Deep Learning Network

Xuanmao Huang
Khoury College of Computer Sciences
Northeastern University
huang.xuanm@northeastern.edu

Charlie (Cunxi) Huang
Khoury College of Computer Sciences
Northeastern University
huang.cun@northeastern.edu

Abstract

The goal of this project is to evaluate the messy probability of a given room picture. For an input image, our model will output a messy probability, indicating how messy the room is. We tune parameters for our models to achieve the best test probability compared to ground truth. We show how different models perform on this task, providing training time, test accuracy, etc.

Index Terms: SVM, CNN, ResNet

1 Introduction

Room cleanliness detection can be modeled as a typical computer vision task. In this project, we view this problem as a binary classification. For an input image with 3 RGB channels, we will determine whether the room in the image is clean or messy. To achieve that, 3 models are implemented, including SVM, CNN, and ResNet. We compare the performance of our models based on training time, evaluation results, and parameter tuning.

One difficult thing to do is to allow the models to "see" room images. To achieve this, we need to use TorcVision to transform images into arrays, then feed the resulting arrays to models.

Our motivation for doing this project is that it allows us to see the best model for dealing with image categorization. In addition to this, we can apply a similar approach in the future, which can help sort images in different scenarios.

2 Methods

2.1 SVM

For the Support Vector Machine (SVM) to apply to image-like input, we need to convert the image matrix into a flat vector form. And for each image, we have an associated label:[1] clean or dirty. Thus, we can use SVM to tackle this problem. Denote input as: $X = \{x_1, \dots, x_n\}$, output as $Y = \{y_1, \dots, y_n\}$, where n is the number of all data points. SVM aims to find a hyperplane to correctly classify the two sets. The goal function is given by:

$$\operatorname{argmin}_{w,b} \frac{1}{2} ||w||_2^2, \quad \text{s. t.} \;\; y_i(w^T x_i + b) \geq 1. \tag{1}$$

This optimization problem can be transformed into a dual form by KKT constraints. The dual form is given by:

$$\operatorname{argmin}_{\alpha} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j K(x_i \cdot x_j) - \sum_{i=1}^{n} \alpha_i \,, \quad \text{s. t.} \;\; 0 \leq \alpha_i \leq C, \sum_{i=1}^{n} y_i \alpha_i = 0. \tag{2}$$

The $K(x_i \cdot x_j)$ is a kernel function. In this project, we implement three kernel modes: "linear", "rbf", and "poly".

2.2 CNN

Convolutional Neural Network (CNN) is a commonly used method to deal with image classification problems.[2] In our task, we need to let our model output a messy probability to represent the information encoded in the dataset. To achieve this goal, we design the neural network based on the AlexNet structure. The input 3-channel matrix is firstly fed to a batch normalization layer for regularization. Then the data goes through two convolutional layers, each associated with a max-pooling layer. The output is linked to three fully connected layers to be mapped to the dimension of two. This final output is activated by the softmax function to represent the probability of messy or clean:

$$\sigma(z_i) = \frac{\exp(z_i)}{\sum_{i=1}^{n} \exp(z_i)}. \tag{3}$$

2.3 ResNet

A residual neural network (ResNet) is an artificial neural network (ANN) of a kind that builds on designs adapted from pyramidal cells in the cerebral cortex. Residual neural networks implement this by utilizing skip connections, or shortcuts to jump over some layers.[3] Typical ResNet models are implemented with triple-layer skips that contain nonlinearities (ReLU) and batch normalization in between. In our implementation, we load a ResNet18 pre-trained on the ImageNet dataset. The output is then mapped to the dimension of two through three fully connected layers to represent the two probabilities.

3 Related Works

While image detection isn't new in the field of deep learning, some similar works have been done by other colleagues. For example, Chum et al 2008 researched near-duplicate image detection. This paper proposes two novel image similarity measures for fast indexing via locality-sensitive hashing. What's like our project is that they also converted images to vectors/arrays to analyze the images in a better way.

Deep learning can also be useful in medical image analysis. For example, Shen et al 2017 proposed a way to use CNNs to analyze many medical images like CT, MRI, PET, and X-ray. They said the structural characteristics of these medical images allow CNNs to reach a great interest in the field.

Not only we can use deep learning in image detection but also in image segmentation. Minaee et al 2021 provided a comprehensive review of the recent literature, covering the spectrum of efforts in semantic and instance segmentation, one of them is the convolutional pixel-labeling network. The paper proposed many pioneering ways to do image detection, allowing us to study them in the future.

Deep learning can also make changes to images: super-resolution can be one's best friend in upscaling an image for his/her desktop wallpaper. Wang et al 2020 roughly categorized image super-resolution into 3 categories: supervised super-resolution, unsupervised super-resolution, and domain-specific super-resolution. All of these can use machine learning ways to achieve.

Other than super-resolution, denoising is another field that uses deep learning to make a difference in images. Tian et al 2018 completely reviewed and summarized the deep learning technologies for image denoising in recent years. They manipulated conventional machine learning methods for image denoising.

4 Related Implementations

As we use a public dataset on Kaggle, the two highest voted implementations using the same dataset are valuable to be discussed. The first one is *Image Classification with Logistic Regression* by Gulsah Demiryurek.[9] In her code, she used logistic regression to classify images into either clean or messy category. The code is relatively simple because the algorithm is simple. She also wrote her logistic regression class to categorize the images. In the end, she showed that using her method, the training accuracy reached 100% and test accuracy got to 93.75%. This seems like a successful implementation.

The second implementation comes from Maxim Gerasimov: custom_imagenet.[10] In his code, we notice that he used the TorchVision library as we do. This implies that similarities may exist in some parts of our code implementation: we both used TorchVision's image-to-array transformation method. However, the model he used is a convolutional model called ConvNeXT. The ConvNeXT model was proposed in 2020 by Liu et al.[11] This model is an advanced model that's beyond the course material.

In all, our choices of the model are suitable for the class as well as for the image classification task.

5 Dataset

We use the public dataset, Messy vs Clean Room, from Kaggle:
https://www.kaggle.com/datasets/cdawn1/messy-vs-clean-room. This dataset mainly contains 3 folders: Training, Validation, and Testing. The training folder consists of 2 parts: clean and messy, each has 96 images. The validation folder consists of 2 parts: clean and messy, each has 10 images. The testing folder also consists of 2 parts: clean and messy, each has 10 images.

5.1 Data Analysis

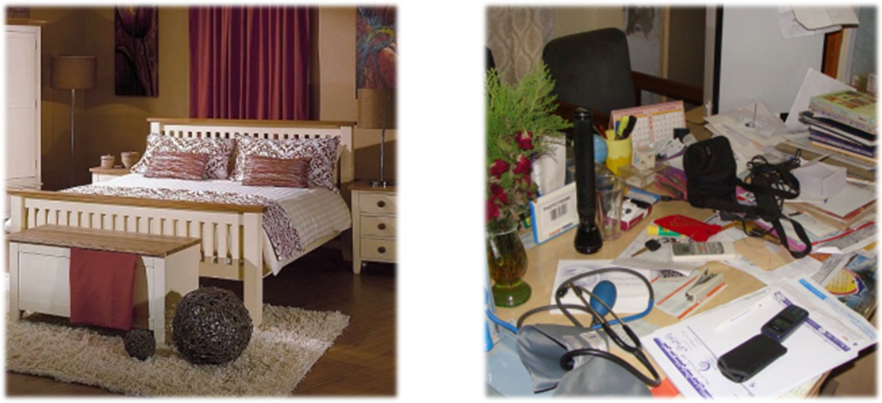

Figure 1: Clean and messy rooms.

The dataset acquired from Kaggle is clean, we do not need to do any data cleaning before starting training. All images have identical dimensions and the relative room size in every image is about the same.

To allow our models “see” these room images, we need to use TorcVision to transform images into Tensor-like arrays, then feed the resulting arrays to the models. There’s nothing special about the data itself before we use the models to gather some collective information.

Unlike some other previous works which mainly focused on the grayscale channel of an image, we innovated a little on our project to let the models analyze on all three RGB channels.

6 Method Analysis and Setup

6.1 SVM

1) Training phase: To minimize the dual form, the core idea is to use alternating minimization. Each time we only update one pair of (α_i, α_j) in equation 2 and set others as constants. We update this pair according to the following rule:

1. For a pair (α_i, α_j), fix α_j, minimize the function by finding α_i;
2. Fix α_i, minimize the function by finding α_j;
3. Repeat steps 1 and 2 above until convergence.

2) Testing phase: We apply the trained model to the test set to derive the predicted label $\hat{Y} = \{\hat{y}_1, \dots, \hat{y}_n\}$. Then evaluate the accuracy.

6.2 CNN

1) Training phase: To train the network we designed, a few techniques are introduced. The loss function is defined by Cross-Entropy Loss:

$$L = -(y \log p + (1 - y) \log(1 - p)). \tag{4}$$

We update our model by Back Propagation. Its efficiency makes it feasible to use gradient methods for training multilayer networks, and updating weights to minimize loss; In our settings, stochastic gradient descent is used.

2) Testing phase: After training our model and tuning the hyperparameters, we directly apply the model to the input image from the test set. Our model will generate a messy probability and use the max function to determine whether the room is tidy or not.

6.3 ResNet

1) Training phase: Same as CNN, we also use Cross-Entropy Loss as the loss function and backpropagation to update weights.

2) Testing phase: After training our model and tuning the hyperparameters, we directly apply the model to the input image from the test set. Our model will generate a messy probability and use the max function to determine whether the room is tidy or not. The figure below shows the basic structure of our model.

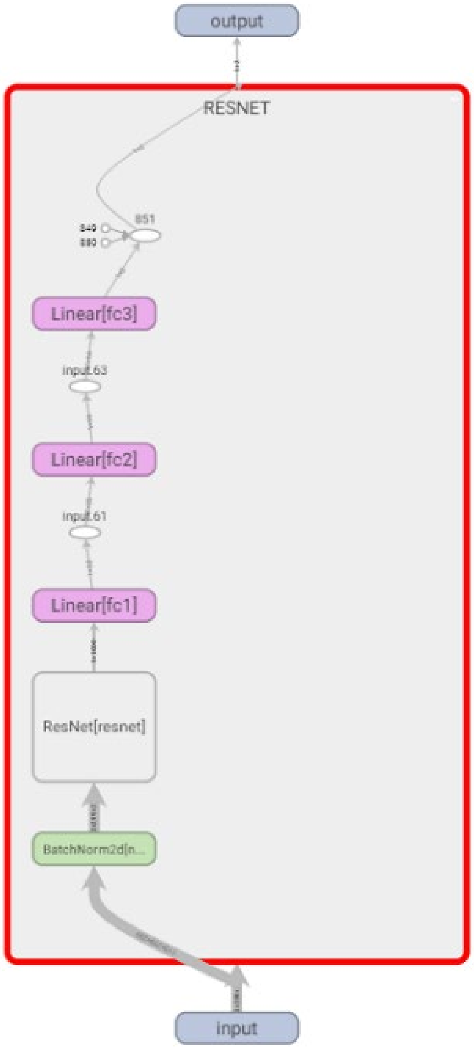

Figure 2: The structure of our model.

7 Results and Discussion

7.1 SVM

For validation accuracy, we tested three parameters: kernel mode, degree (for poly), and C, which is the punishment coefficient. The following is the test result:

C	Kernel	Degree	Accuracy
0.1	linear		1
0.1	poly	3	0.61
0.1	poly	5	0.6
0.1	rbf		0.79
0.01	linear		1
0.01	poly	3	0.54
0.01	poly	5	0.52

0.01	rbf		0.79
0.001	linear		1
0.001	poly	3	0.54
0.001	poly	5	0.52
0.001	rbf		0.79

Table 1: SVM Result.

Among all settings, the rbf kernel performs the best. Changing C does not affect the result significantly for many settings.

7.2 CNN

For validation accuracy, we tested three hyperparameters: learning rate, number of epochs, and criteria (different loss functions). The following is the test result:

Learning rate	**Num_epochs**	**Criteria**	**Accuracy**
0.01	10	CrossEntropyLoss()	0.5
0.01	10	NLLLoss()	0.5
0.01	20	CrossEntropyLoss()	0.5
0.01	20	NLLLoss()	0.5
0.01	30	CrossEntropyLoss()	0.5
0.01	30	NLLLoss()	0.5
0.001	10	CrossEntropyLoss()	0.75
0.001	10	NLLLoss()	0.5
0.001	20	CrossEntropyLoss()	0.7
0.001	20	NLLLoss()	0.5
0.001	30	CrossEntropyLoss()	0.65
0.001	30	NLLLoss()	0.6
0.0001	10	CrossEntropyLoss()	0.8
0.0001	10	NLLLoss()	0.65
0.0001	20	CrossEntropyLoss()	0.65
0.0001	20	NLLLoss()	0.6
0.0001	30	CrossEntropyLoss()	0.55
0.0001	30	NLLLoss()	0.55
0.00001	10	CrossEntropyLoss()	0.7
0.00001	10	NLLLoss()	0.6
0.00001	20	CrossEntropyLoss()	0.75
0.00001	20	NLLLoss()	0.6
0.00001	30	CrossEntropyLoss()	0.75
0.00001	30	NLLLoss()	0.7

Table 2: CNN Result.

In the end, our model has an accuracy of no greater than 0.8 when using the best setup. This is roughly about the same as a finely tuned SVM. This exemplifies that a shallow network may not be enough for dealing with image-like information with tons of features. Thus, we need to import a more complicated network to perform better.

7.3 ResNet

For validation accuracy, we tested three hyperparameters: learning rate, number of epochs, and criteria (different loss functions). The following is the test result:

Learning rate	Num_epochs	Criteria	Accuracy
0.01	10	CrossEntropyLoss()	0.5
0.01	10	NLLLoss()	0.5
0.01	20	CrossEntropyLoss()	0.5
0.01	20	NLLLoss()	0.5
0.01	30	CrossEntropyLoss()	0.5
0.01	30	NLLLoss()	0.5
0.001	10	CrossEntropyLoss()	1
0.001	10	NLLLoss()	0.85
0.001	20	CrossEntropyLoss()	0.65
0.001	20	NLLLoss()	0.8
0.001	30	CrossEntropyLoss()	0.9
0.001	30	NLLLoss()	0.95
0.0001	10	CrossEntropyLoss()	0.95
0.0001	10	NLLLoss()	0.95
0.0001	20	CrossEntropyLoss()	1
0.0001	20	NLLLoss()	0.95
0.0001	30	CrossEntropyLoss()	1
0.0001	30	NLLLoss()	1
0.00001	10	CrossEntropyLoss()	1
0.00001	10	NLLLoss()	1
0.00001	20	CrossEntropyLoss()	1
0.00001	20	NLLLoss()	0.95
0.00001	30	CrossEntropyLoss()	1
0.00001	30	NLLLoss()	1

Table 3: ResNet Result.

As we can see, once we gradually increase the learning rate, the generated label goes as close as the ground truth. It demonstrates that ResNet fully captures the features within the dataset and thus can make inferences based on the hidden information.

7.4 Overall

The following table shows the best parameters with the test accuracy:

model				**acc**
svm	'C': 0.1	'kernel': 'linear'	'degree': none	0.6
cnn	'learning_rate': 0.0001	'num_epochs': 10	'criterion': CrossEntropyLoss()	0.7
resnet	'learning_rate': 0.001	'num_epochs': 10	'criterion': CrossEntropyLoss()	1.0

Table 4: Model Comparisons.

Note that we train the neural network on GPU so that it is accelerated. In real-life, the computation cost for the neural network is significantly larger than traditional machine learning methods like SVM. From these results, we can interpret that ResNet is the best model of all. It shows that a deep neural network can capture the hidden information within the dataset and can make good inferences based on that. Besides, SVM also has the power of feature extraction, and its result is equivalent to shallow network models with one or two convolutional layers.

We also plotted iteration times vs cross-entropy loss for both CNN and ResNet:

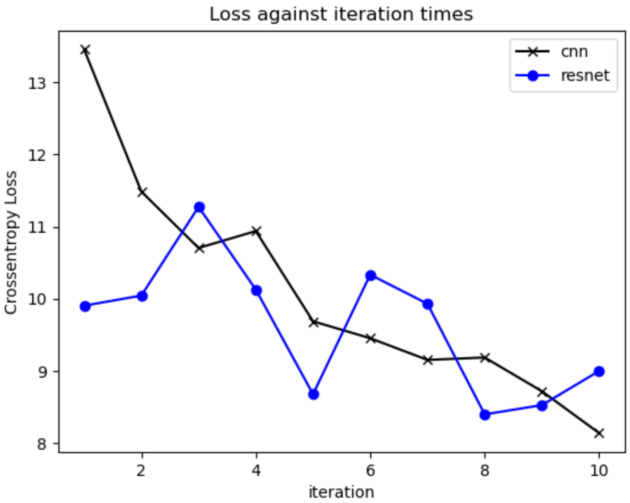

Figure 3: Iteration times vs Cross-entropy Loss.

8 Conclusion

In our project, we implement three models to detect room cleanliness. Among all three models, ResNet is the most promising one, correctly classifying all test data points. All models require fine-tuning parameters. Though a neural network needs more computation resources, a deep model can be trained to generate the best result.

As a reference, in the future, we could use more resources to improve the performance of our models. For example, we can use more room images—the training set has 96 images for each clean and messy category, in real-life this won't be enough. We can also acquire more data for pre-processing. In addition to this, it would be better if the models can be used in real-life scenarios. For example, we can apply the models to create mobile applications to create an image object detector.

Many courses and tutorials have recently drawn new machine learning engineers' attention to image classification. We suspect, this trend will likely continue to be for the foreseeable future.

9 References

[1] Harris Drucker, Christopher J Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. Support vector regression machines. *Advances in neural information processing systems*, 9:155–161, 1996.

[2] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[4] Chum, Ondrej, James Philbin, and Andrew Zisserman. "Near duplicate image detection: Min-hash and TF-IDF weighting." Bmvc. Vol. 810. 2008.

[5] Shen, Dinggang, Guorong Wu, and Heung-Il Suk. "Deep learning in medical image analysis." *Annual review of biomedical engineering* 19 (2017): 221-248.

[6] Minaee, Shervin, et al. "Image segmentation using deep learning: A survey." *IEEE transactions on pattern analysis and machine intelligence* (2021).

[7] Wang, Zhihao, Jian Chen, and Steven CH Hoi. "Deep learning for image super-resolution: A survey." *IEEE transactions on pattern analysis and machine intelligence* 43.10 (2020): 3365-3387.

[8] Tian, Chunwei, et al. "Deep learning for image denoising: A survey." *International Conference on Genetic and Evolutionary Computing.* Springer, Singapore, 2018.

[9] Demiryurek, Gulsah. "Image Classification with Logistic Regression." Kaggle, 29 Apr. 2019, www.kaggle.com/code/gulsahdemiryurek/image-classification-with-logistic-regression.

[10] Gerasimov, Maxim. "Custom_imagenet." Kaggle, 30 Jan. 2022, www.kaggle.com/code/mygaps/custom-imagenet.

[11] Liu, Zhuang, et al. "A ConvNet for the 2020s." *arXiv preprint arXiv:2201.03545* (2022).

10 Appendix

10.1 Equations

$$\text{argmin}_{w,b}\frac{1}{2}||w||_2^2, \quad \text{s.t.}\ \ y_i(w^T x_i + b) \geq 1. \tag{1}$$

$$\text{argmin}_{\alpha}\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j K(x_i \cdot x_j) - \sum_{i=1}^{n}\alpha_i\,, \quad \text{s.t.}\ \ 0 \leq \alpha_i \leq C, \sum_{i=1}^{n} y_i \alpha_i = 0. \tag{2}$$

$$\sigma(z_i) = \frac{\exp(z_i)}{\sum_{i=1}^{n}\exp(z_i)}. \tag{3}$$

$$L = -(y \log p + (1-y)\log(1-p)). \tag{4}$$

10.2 Tables

C	Kernel	Degree	Accuracy
0.1	linear		1
0.1	poly	3	0.61
0.1	poly	5	0.6
0.1	rbf		0.79
0.01	linear		1
0.01	poly	3	0.54
0.01	poly	5	0.52
0.01	rbf		0.79
0.001	linear		1
0.001	poly	3	0.54
0.001	poly	5	0.52
0.001	rbf		0.79

Table 5: SVM Result.

SVM-kernel-C	SVM-poly-10	SVM-poly-1	SVM-poly-0.1
training time	15.91	14.95	14.59
accuracy	0.7	0.7	0.7

Table 2: Other SVM Result 1.

SVM-kernel-C	SVM-rbf-10	SVM-rbf-1	SVM-rbf-0.1
training time	14.75	14.94	14.97
accuracy	0.5	0.5	0.5

Table 3: Other SVM Result 2.

SVM-kernel-C	SVM-linear-10	SVM-linear-1	SVM-linear-0.1
training time	14.22	14.52	19.71
accuracy	0.6	0.6	0.6

Table 4: Other SVM Result 3.

Learning rate	**Num_epochs**	**Criteria**	**Accuracy**
0.01	10	CrossEntropyLoss()	0.5
0.01	10	NLLLoss()	0.5
0.01	20	CrossEntropyLoss()	0.5
0.01	20	NLLLoss()	0.5
0.01	30	CrossEntropyLoss()	0.5
0.01	30	NLLLoss()	0.5
0.001	10	CrossEntropyLoss()	0.75
0.001	10	NLLLoss()	0.5
0.001	20	CrossEntropyLoss()	0.7
0.001	20	NLLLoss()	0.5
0.001	30	CrossEntropyLoss()	0.65
0.001	30	NLLLoss()	0.6
0.0001	10	CrossEntropyLoss()	0.8
0.0001	10	NLLLoss()	0.65
0.0001	20	CrossEntropyLoss()	0.65
0.0001	20	NLLLoss()	0.6
0.0001	30	CrossEntropyLoss()	0.55
0.0001	30	NLLLoss()	0.55
0.00001	10	CrossEntropyLoss()	0.7
0.00001	10	NLLLoss()	0.6
0.00001	20	CrossEntropyLoss()	0.75
0.00001	20	NLLLoss()	0.6
0.00001	30	CrossEntropyLoss()	0.75
0.00001	30	NLLLoss()	0.7

Table 5: CNN Result.

No.	Clean prob	Messy prob	Label
0	1.3393e-03	9.9866e-01	1
1	9.9437e-01	5.6297e-03	0
2	5.2131e-03	9.9479e-01	1
3	1.9273e-01	8.0727e-01	1
4	3.8652e-02	9.6135e-01	1
5	7.5680e-02	9.2432e-01	1
6	9.9999e-01	1.2187e-05	0
7	7.6671e-01	2.3329e-01	0
8	9.6782e-01	3.2175e-02	0
9	9.9996e-01	3.5226e-05	0

Table 6: Other CNN Result.

Learning rate	**Num_epochs**	**Criteria**	**Accuracy**
0.01	10	CrossEntropyLoss()	0.5
0.01	10	NLLLoss()	0.5
0.01	20	CrossEntropyLoss()	0.5
0.01	20	NLLLoss()	0.5
0.01	30	CrossEntropyLoss()	0.5
0.01	30	NLLLoss()	0.5
0.001	10	CrossEntropyLoss()	1
0.001	10	NLLLoss()	0.85
0.001	20	CrossEntropyLoss()	0.65
0.001	20	NLLLoss()	0.8
0.001	30	CrossEntropyLoss()	0.9
0.001	30	NLLLoss()	0.95
0.0001	10	CrossEntropyLoss()	0.95
0.0001	10	NLLLoss()	0.95
0.0001	20	CrossEntropyLoss()	1
0.0001	20	NLLLoss()	0.95
0.0001	30	CrossEntropyLoss()	1
0.0001	30	NLLLoss()	1
0.00001	10	CrossEntropyLoss()	1
0.00001	10	NLLLoss()	1
0.00001	20	CrossEntropyLoss()	1
0.00001	20	NLLLoss()	0.95
0.00001	30	CrossEntropyLoss()	1
0.00001	30	NLLLoss()	1

Table 7: ResNet Result.

No.	Clean prob	Messy prob	Label
0	8.7576e-01	1.2424e-01	0
1	8.9349e-01	1.0651e-01	0
2	1.2975e-01	8.7025e-01	1
3	8.5477e-01	1.4523e-01	0
4	4.7249e-16	1.0000e+00	1
5	3.8632e-10	1.0000e+00	1
6	8.9842e-01	1.0158e-01	0
7	1.3818e-02	9.8618e-01	1
8	8.4371e-10	1.0000e+00	1
9	8.8990e-01	1.1010e-01	0

Table 8: ResNet Result.

model	SVM-poly-10	SVM-rbf-10	SVM-linear-10
training time	15.91	14.75	14.22
accuracy	0.7	0.5	0.6

Table 9: SVM Comparison.

model	CNN	ResNet
training time	4.28	14.39
accuracy	0.6	1.0

Table 10: DNN Model Comparison.

model				**acc**
svm	‘C’: 0.1	‘kernel’: ‘linear’	‘degree’: none	0.6
cnn	‘learning_rate’: 0.0001	‘num_epochs’: 10	‘criterion’: CrossEntropyLoss()	0.7
resnet	‘learning_rate’: 0.001	‘num_epochs’: 10	‘criterion’: CrossEntropyLoss()	1.0

Table 11: Model Comparisons.

10.3 Graphs

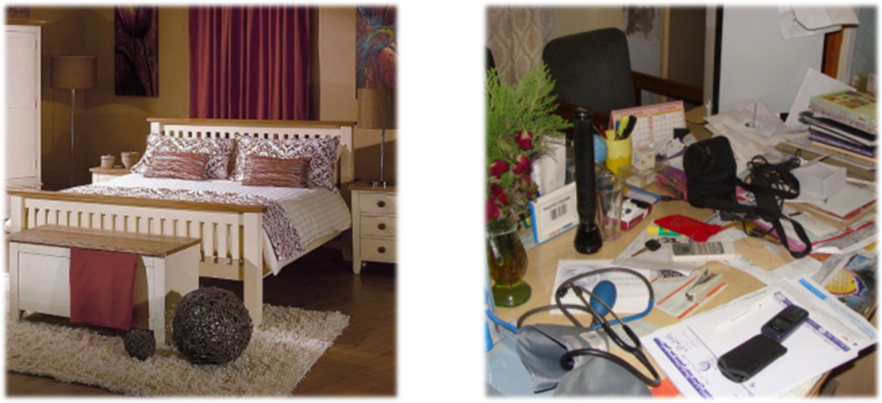

Figure 4: Clean and messy rooms.

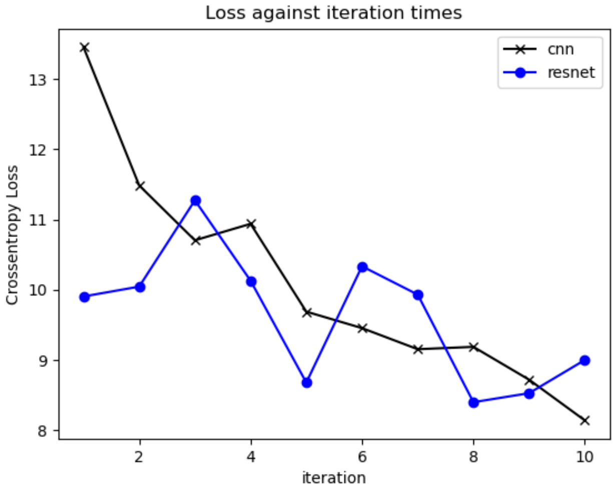

Figure 2: Iteration times vs Cross-entropy Loss.

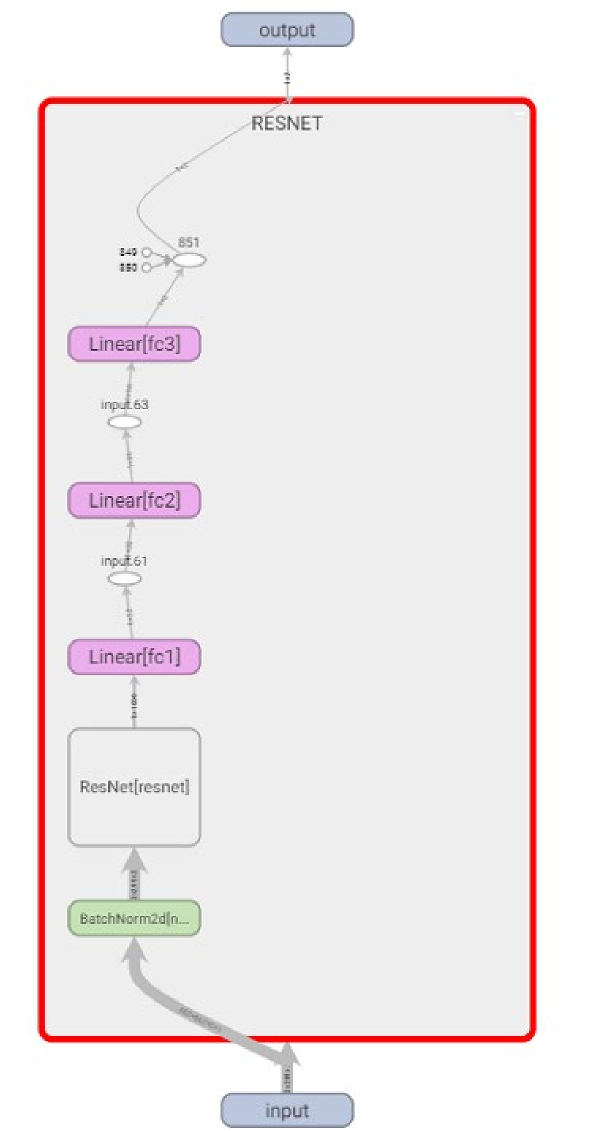

Figure 3: The structure of our model.

11 Statement of Contributions

During the project implementation, each group member contributed the work equally. Xuanmao Huang mainly focused on the deployment of the models as well as tuning, Charlie Huang mainly did the research and report write-up. Each member had a wonderful time in collaboration.