# Implementation of Four Convolution Neural Network and Their Visualization

**Enze Ma**
Department of Computer Science
University of California San Diego
e1ma@ucsd.edu

**Yixin Jiang**
Department of Computer Science
University of California San Diego
yij011@ucsd.edu

**Fangqi Yuan**
Department of Computer Science
University of California San Diego
fayuan@ucsd.edu

## Abstract

We implement four convolution neural networks. The first model is the most basic one which called the baseline model with the worst performance on the food-101 classification task is around 46.2%. In the second model, which called the custom model, we make some improvements and the test accuracy is around 55.5%. It improves more than 20% compared with the baseline which is a tremendous improvement. In the third model and the fourth model, we use the resnet18 and vgg16 to implement transfer learning by replacing the last full connect layer, both models' accuracy is around 68% which is way better than the first two models.

## 1 Introduction

Comparing the four models' different performances, we have found that the thicker the depth of the model, the better the performance the model will have. Even though it may not be correct for all models, during our experiments, we find this may be true. Through the visualization process, we will visualize the weight of the first convolution layer and the feature maps of three different position convolution layer for all the models except baseline. We find that the better the model performance is, the more simple its weight and feature map will be (simple means more colorless), where a lot of parts of the image will be black instead of gray. We assume it because each weight or feature map is more specific and represents a more important part of each class with less ambiguity, which is represented by gray color.

## 2 Related Work

Wang, F., Kong, T. , Liu, H., Zhang, R., & Li, H. (n.d.). Papers with code - self-supervised learning by estimating twin class distributions. Self-Supervised Learning by Estimating Twin Class Distributions — Papers With Code. Retrieved February 21, 2022, from https://paperswithcode.com/paper/self-supervised-learning-by-estimating-twin-1

This paper introduces a self-supervised representation learning method called TWIST that minimizes the distribution entropy of each sample so that each sample can enforce the category prediction and maximizes the entropy of the average distribution so that the prediction of different samples can maintain the original diversity and make the prediction more accurate.

Lee, K.-H., He, X., Zhang, L., & Yang, L. (n.d.). Papers with code - cleannet: Transfer learning for scalable image classifier training with label noise. CleanNet: Transfer Learning for Scalable Image Classifier Training with Label Noise — Papers With Code. Retrieved February 21, 2022, from https://paperswithcode.com/paper/cleannet-transfer-learning-for-scalable-image

This paper studies the problem of learning image classification models with label noise. In order to reduce the amount of manual supervision for label noise cleaning, the research team integrated CleanNet and traditional convolutional neural network classifier into a framework for image classification learning. The validity of the algorithm is proved in the tasks of label noise detection and noise data image classification.

## 3 Models

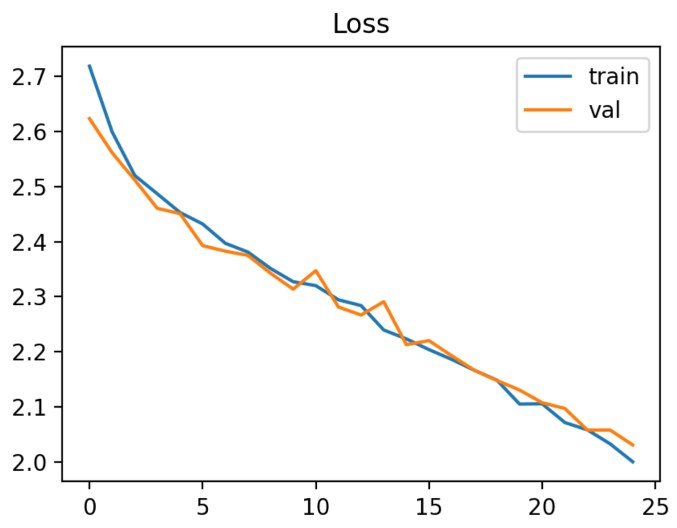
For the baseline model, we have 4 convolution layers. Between each convolution layer, we have a batch normalization layer and a ReLU activation layer. Between conv3 and conv4, we add a 3x3 maxpool after a ReLU activation layer. After conv4, we send the output to the adaptive_avg_pool with output size 1x1. Then, we send the result to fc1 layer. The output from fc1 will go through a dropout layer with p=0.5 and a ReLU activation layer. Finally, the output will be sent to the fc2 layer which will have 20 output units. The best accuracy on the test set we got 46.22%.
For the custom model. We have 6 convolution layers. Same as the baseline model, between each convolution layer, we have a batch normalization layer and a ReLU activation layer. Between both conv2 and conv3, conv4 and conv5, we have an extra 3x3 maxpool layer. Between conv5 and conv6, we have a dropout layer with p=0.5. After conv6, we send the output to the adaptive_avg_pool with output size 1x1. Then, we send the result to fc1 layer. The output from fc1 will go through a dropout layer with p=0.5 and a ReLU activation layer. Finally, the output will be sent to the fc1 layer which will have 20 output units. The best accuracy on the test set we got 55.56%.

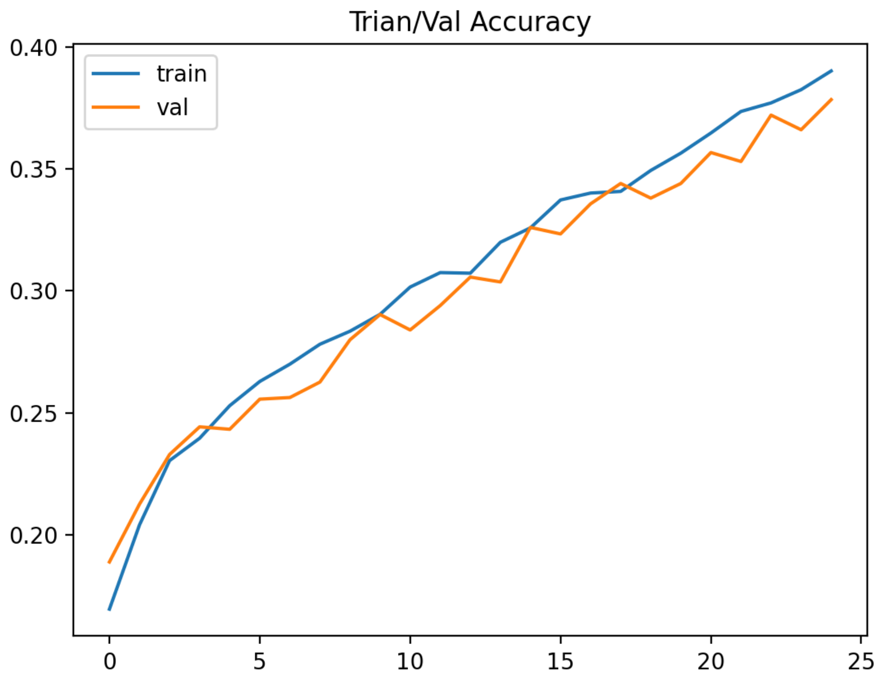
| Layer Name | Dimension |
|---|---|
| input | 224x224x3 |
| Conv1 | In-channel:3 Out-channel:64 Kernel Size:3 |
| BN1 | In-channel:64 |
| ReLU | |
| Conv2 | In-channel:64 Out-channel:128 Kernel Size:3 |
| BN2 | In-channel:128 |
| ReLU | |
| Maxpool1 | Kernel Size:3 |
| Conv3 | In-channel: 128 Out-channel:168 Kernel Size:3 |
| BN3 | In-channel:168 |
| ReLU | |
| Conv4 | In-channel: 168 Out-channel:168 Kernel Size:3 |
| BN4 | In-channel:168 |
| ReLU | |
| Maxpool2 | Kernel Size:3 |
| Conv5 | In-channel: 168 Out-channel:168 Kernel Size:3 |
| BN5 | In-channel:168 |
| ReLU | |
| Dropout | P=0.5 |
| Conv6 | In-channel: 168 Out-channel:128 Kernel Size:3 |
| BN5 | In-channel: 128 |
| ReLU | |
| Adaptive average pool | Output size:1x1 |
| Fc1 | 128->20 |

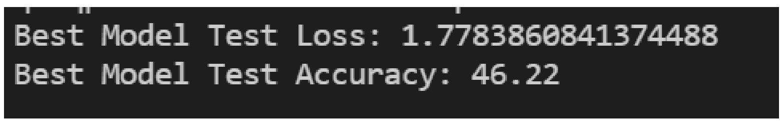
For the vgg16 and resnet18, we find that they are using two different ways to create the model which the latter one is sequential and the former one is a vgg object. When we try to access each layer and replace the last fc layer, we use different ways to implement. For both models, we tried to freeze all layers except the last layer we added. The final accuracy on the test set is pretty high which is around 68%. When we attempt to train some of them by not freezing them, it actually makes the model worse. Adjusting hyperparameter, change loss or optimizer didn't help at all either. Thus, in the end, we are not going to train any layer weight during our training process and change any hyperparameter, we just use the weight from the pre-trained model and the same setting hyperparameter as the custom model.

## 4 Experiments

### 4.1 Baseline

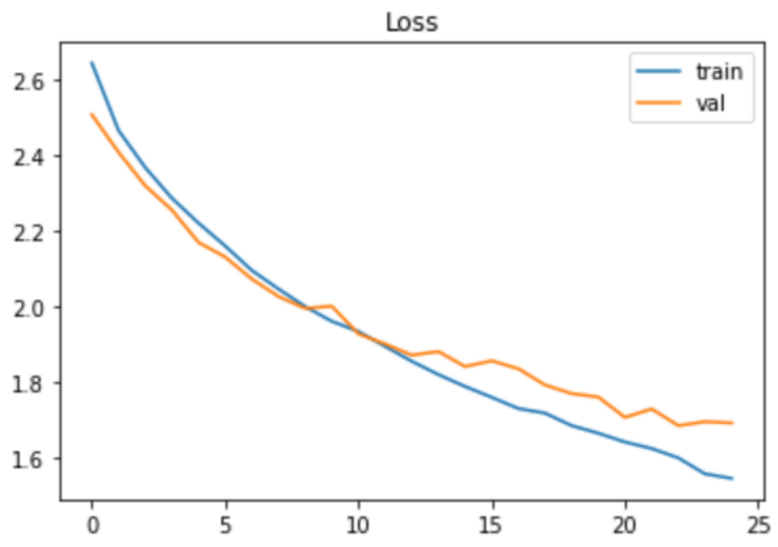


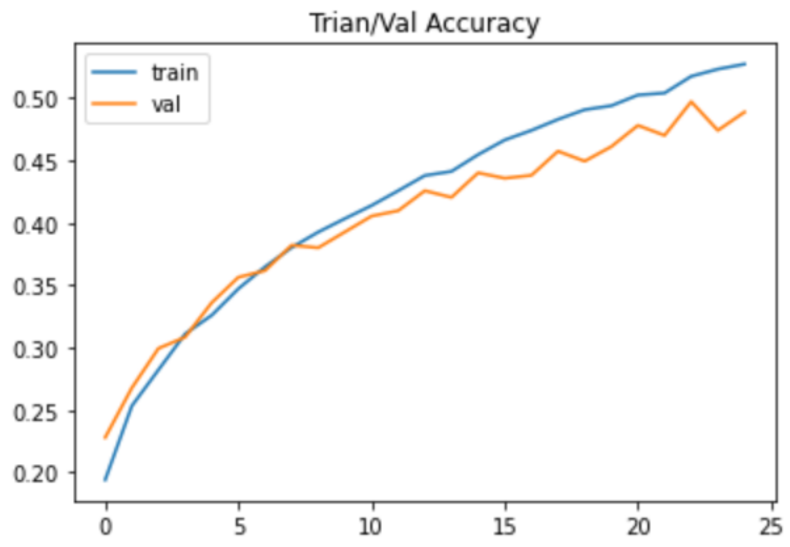


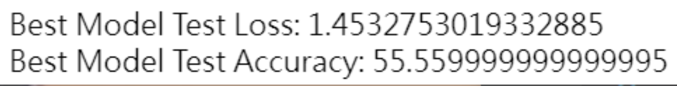


```
Best Model Test Loss: 1.7783860841374488
Best Model Test Accuracy: 46.22
```

### 4.2 Custom







```
Best Model Test Loss: 1.4532753019332885
Best Model Test Accuracy: 55.559999999999995
```

| Model Name | Change Made | Train Accuracy | Val Accuracy | Test Accuracy | Affect |
|---|---|---|---|---|---|
| New_ini | Change Initialization Method to Normal Distribution | 38% | 37% | 42% | Negative |
| New_p | Change the p value for drop out to 0.4 or 0.6 | 40.5% | 37% | 43% | Negative |
| I_lr | Increase Learning Rate to 1.1e-3 | 35% | 34% | 42.28% | Negative |
| D_lr | Decrease Learning Rate to 0.95e-3 | 38% | 37% | 44.62% | Positive |
| New_arc | Keep changing the architecture of the model, then finally get a better one which is describe as above. | 41% | 40% | 50.1% | Positive |
| Ron_flip | Add horizontal randomize flip during the training data transpose process | 40% | 39% | 47.4% | Positive |
| custom | Combine the change for Ron_flip, D_lr, and New_arc | 53% | 51% | 55.55% | Positive |

#### 4.2.1 New_ini

We already give a brief idea about what kind of change we made and how we implement them through the model. The first trial we do is actually change the way we initialize the filter weight for each convolution layer, instead Xavier we use the normal distribution. As the table showed, it negatively affects the model.

#### 4.2.2 New_p

The second change we made is changing the parameter p for the dropout layer, we have two trials. One is increasing the p by 0.1 and the other is decreasing the p by 0.1. By our experiment, both adjustment will not help the model. So, we assume it will not be helpful and we will try adjusting other stuff.

#### 4.2.3 I_lr

Since the learning epoch is low, we think that maybe increasing the learning rate will help the model training sooner and quicker. Thus, we set it to 1.1e-3 and we can find it is a negative decision.

#### 4.2.4 D_lr

A low learning rate will help the model reduce the overfitting problem, our model is complex, so we assume this will help our model. So we set it to 0.95e-3, and we find it did improve the model performance.

#### 4.2.5 New_arc

Our baseline model architecture is pretty simple, so maybe more depth and complex architecture will help the model, after a few trials, the performance is not improved obviously. According to the most common convolution neural network, we just add two more conv layers and one max pool and change the dropout pool position, then it's the same as the custom model' architecture we use at last. Which improve the model performance a lot.

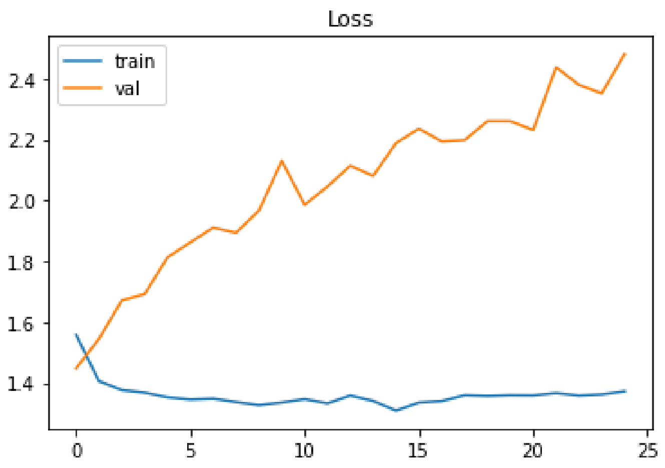
#### 4.2.6 Ron_flip

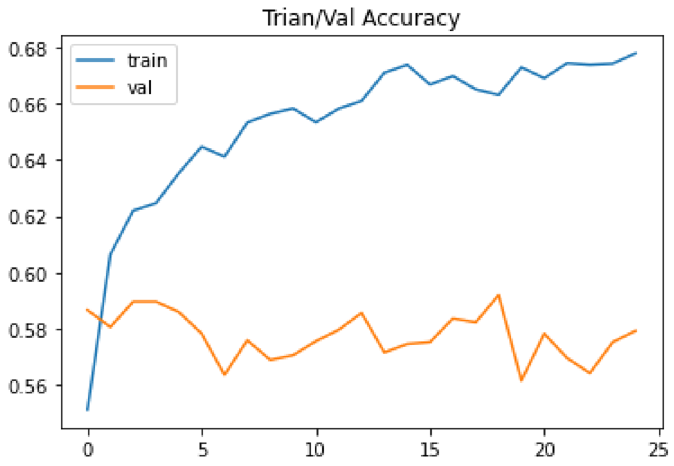
Because decreasing the learning rate did help the model, we assume we can improve the performance of the model by reducing the overfitting problem. We decide to add more random factors which may help, so we add the random horizontal flip for training data transpose. Images now are not consistent, each image may be horizontally flipped. We can find this to help the model too.

#### 4.2.7 Custom

At last, we just combine three positive factors that may improve the performance of the model as the final version custom model, the performance did get improved more than each one and tremendous compare with baseline model. So, we believe the custom model succeeded.

### 4.3 vgg16



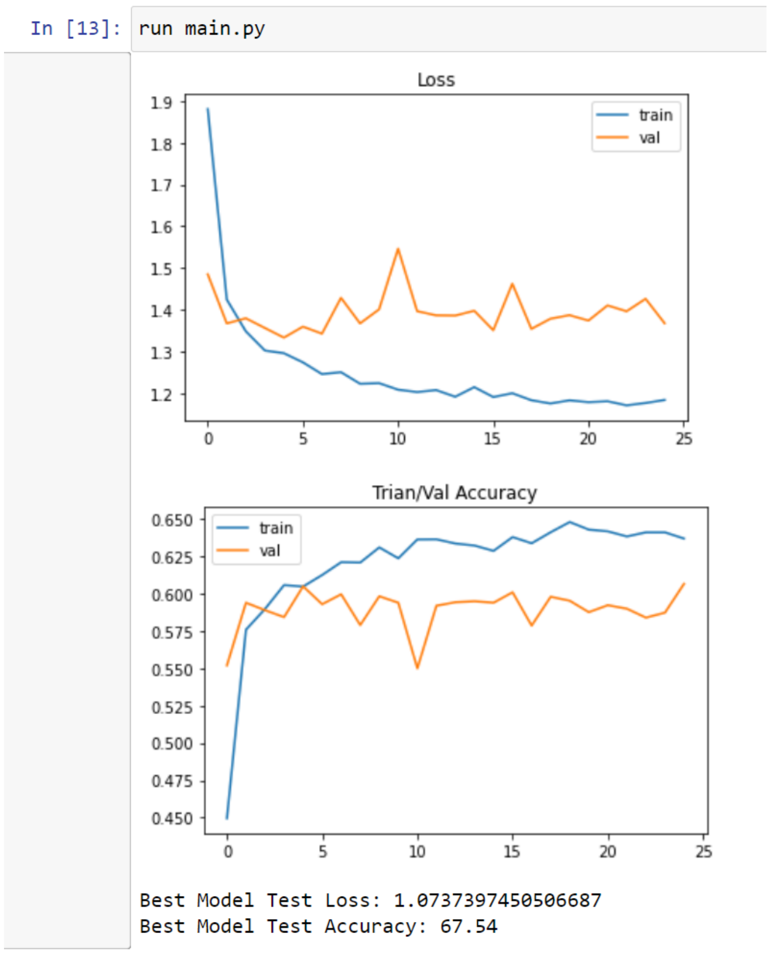


```
Best Model Test Loss: 1.2512294242102653
Best Model Test Accuracy: 68.88
```

| Model Name | Change Made | Train Accuracy | Val Accuracy | Test Accuracy | Affect |
|---|---|---|---|---|---|
| 3x | Didn't Freeze all conv layers %3=0 | 90% | 40% | 47% | Negative |
| New_lr | Same as custom model change learning rate from 1e-3 to 0.95e-3 | 68% | 60% | 68.88% | Positive |

For this part, we are trying to adjust the model that differs from the custom model because similar changes may generate a similar effect. So we just unfreeze different layers, so they can be trained during our training process just for the final fc layer. We tried to unfreeze all the layers that can be divided by 3, but as the result we got is pretty a disaster result, so we discard this experiment from the final vgg model. So, instead we assume that adjust learning rate will be helpful according to the custom model, and it did help the result.

### 4.4 resnet18


```
In [13]: run main.py
```

```
Best Model Test Loss: 1.0737397450506687
Best Model Test Accuracy: 67.54
```

| Model Name | Change Made | Train Accuracy | Val Accuracy | Test Accuracy | Affect |
|---|---|---|---|---|---|
| 10x | Didn't Freeze all layers %10=0 | 65% | 60% | 66.34% | Negative |
| Ron_flip | Same as custom model add random image horizontal flip | 65% | 61% | 67.42% | Positive |

For this part, we are trying to do a similar thing with vgg, but instead of 3 we unfreeze all the layers that can be divided by 10, which did not have a large effect on the model but still lower the model performance, so we discard it. Instead of learning rate we try another method in custom which is adding randomized horizontal flip, it did slightly improve our model, so we keep it.

## 5 Feature map and weights analysis

For this section we will show only 64 images even some layers may have more than 64 filters, we keep 64 images just for the convenience to display and comparison. We believe that 64 images will be enough for our analysis.

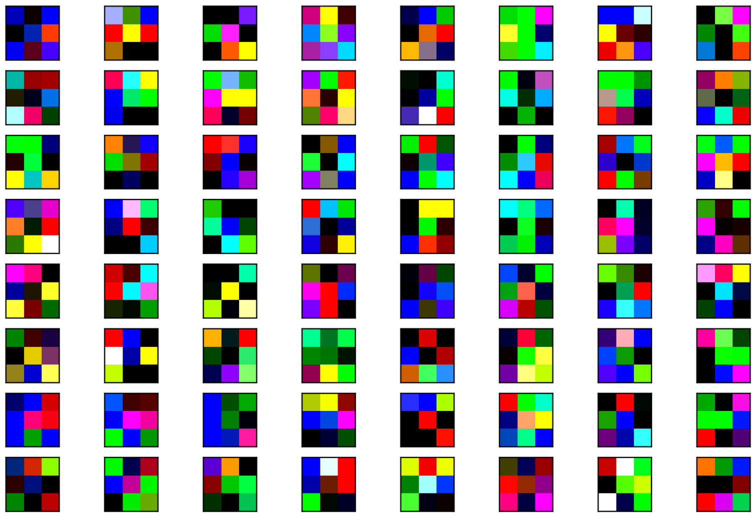


Figure 1: custom first conv layer weight visualization

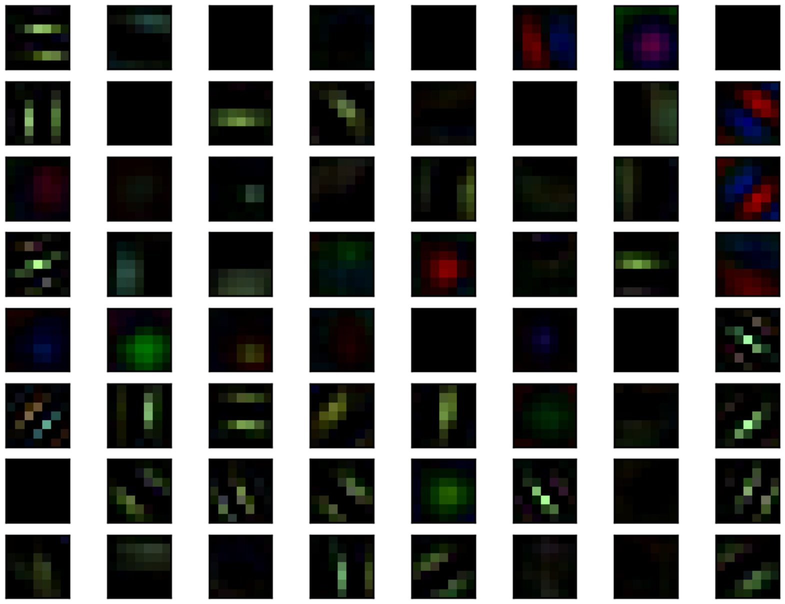


Figure 2: resnet first conv layer weight visualization

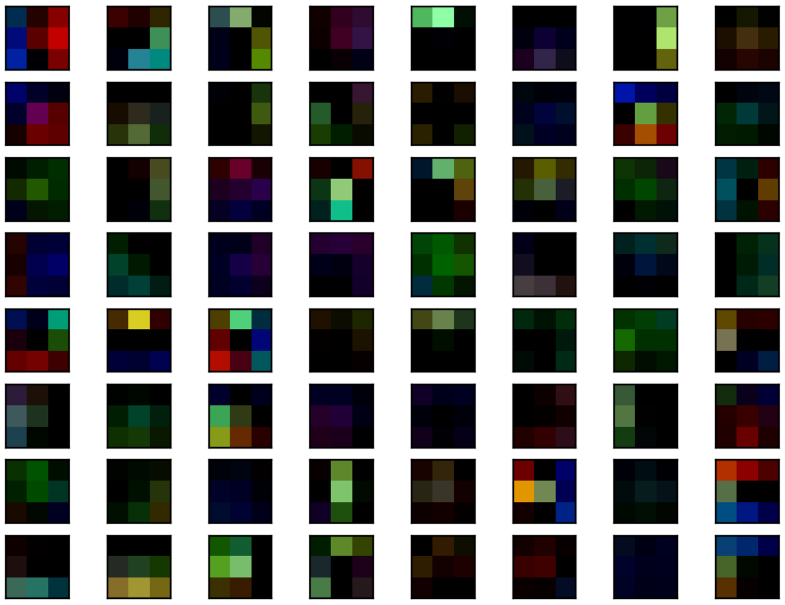


Figure 3: vgg first conv layer weight visualization

We find the weight get from the vgg16 and resnet18 will be darker, and each cub's color will be more similar to the cube around it. Instead, the weight-trained by ourselves will be much more bright than vgg and resnet. The contrast of the color will be more obvious. We think this is because the other two models are more stable after learning from a large amount of data, and their understanding of each feature is more complete.

The feature map will be displayed from first convolution layer to last convolution layer.

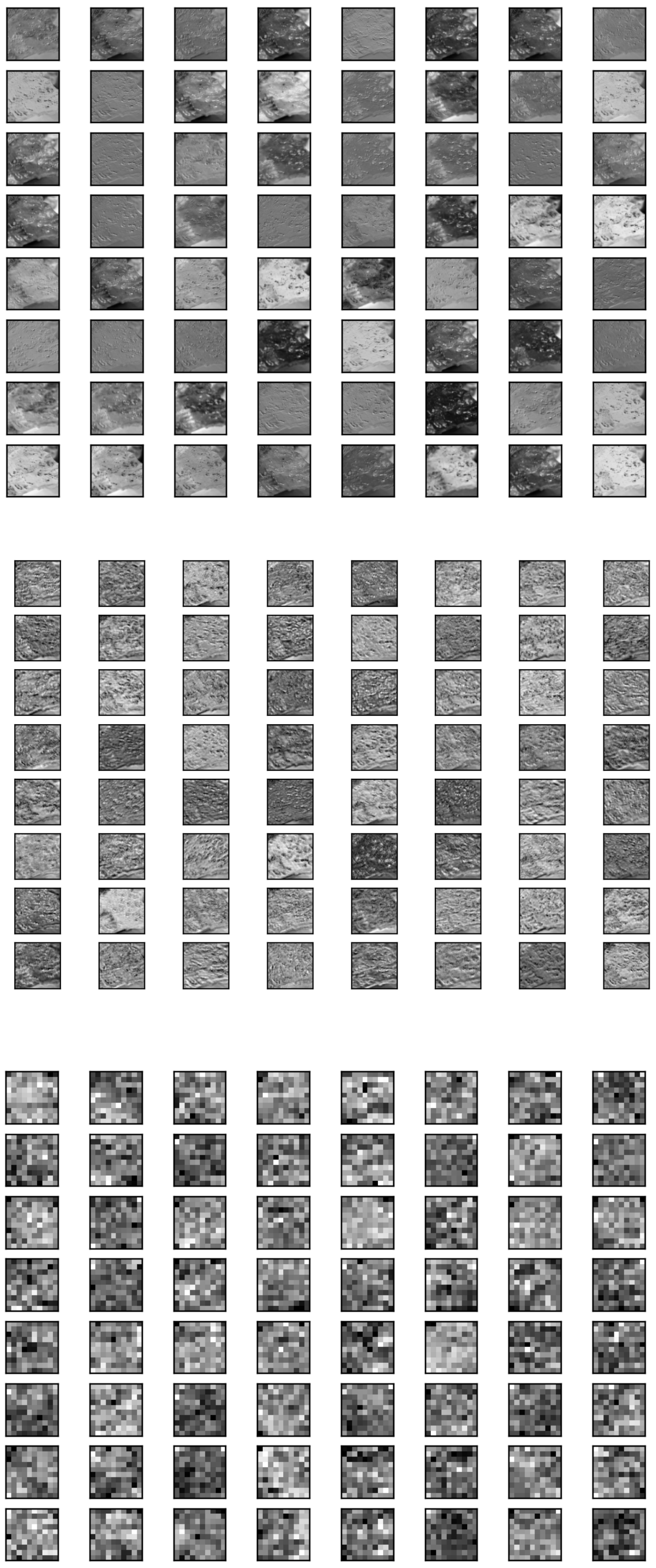


Figure 4: custom feature map visualization

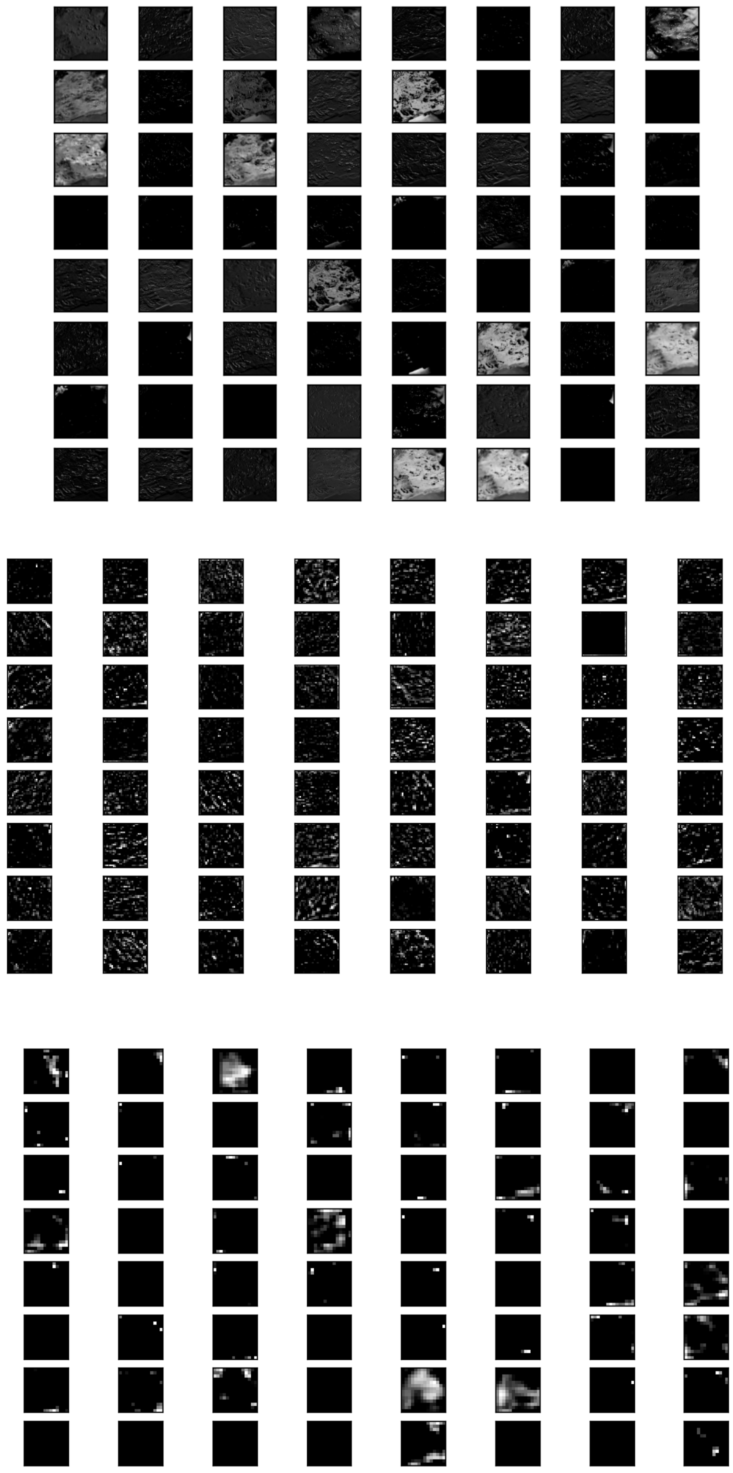


Figure 5: vgg feature map visualization

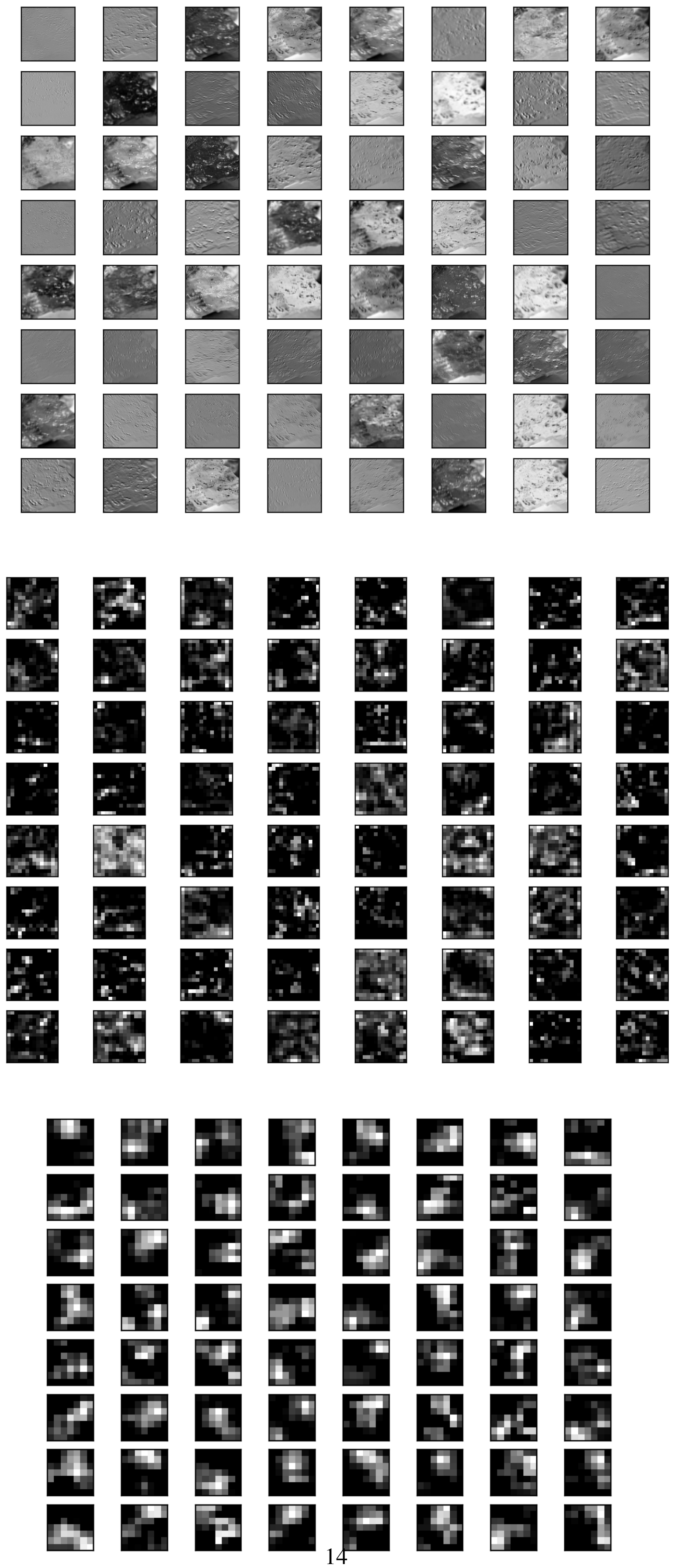


Figure 6: resnet feature map visualization

Comparing each model, we can still find the same situation, the feature map of our self-trained model will be brighter, and most places are grayish. But the feature maps of the other two models will be much darker, which is especially obvious at the last conv layer, the area of white and gray space is much smaller than our self-trained model. Comparing the feature maps of the model itself, we can see that the feature map is getting blurry and darker which may represent the model are learning the feature that we can't understand.

## 6 Discussion

### 6.1 Compare the baseline, custom, vgg16 and resnet18 models.

Compare to the baseline and custom, the vgg16 and resnet18 model is better. Since the depth of the vgg16 and resnet18 model is deeper and these models are pre-trained by a large amount of data which means they are more stable, and a large amount of data will significantly reduce the probability of overfitting problems happen. Thus, the model will be better, so that the accuracy of resnet18 and vgg16 is higher than baseline and custom. And our custom model performs better because we increase the depth of the model compared with the baseline, and we try a lot of methods to reduce the overfitting problem.

### 6.2 Describe at least 3 changes that improve the performance the most as described in custom model.

1) Change the architecture of the model. To be specific, we have increased the number of convolution layers and added an extra 3x3 maxpool layer between conv4 and 5, change the position of the dropout layer, and so on, the more specific description please read the model section. By increasing the depth of the model and more filters we may extract more features from the image we use to train even we have a small size of the training set.
2) Decrease the learning rate to 0.95e-3, by decreasing the learning rate we can reduce the effect we get from the training set which means it's more unlikely to have the overfitting problem.
3) Add horizontal randomize flip during the training data transpose process, same as the reason of learning rate but instead this time we make some noise to reduce the overfitting problem by increasing the dataset variety manually.

### 6.3 Describe the changes that improve the performance as described in transfer learning.

1) Change the learning rate from 1e-3 to 0.95e-3, by decreasing the learning rate we can reduce the effect we get from the training set, even we have frozen all other layers except for the last fc layer and they are train by large amount of data, but we can still have the overfitting problem during our training process. So we can still reduce the probability of overfitting by adjusting the learning rate
2) Add horizontal randomize flip during the training data transpose process,same as the reason of learning we may still have the overfitting problem even so we can still use the similar method for custom model which is increasing the dataset variety manually.

### 6.4 Other design choices/building blocks that helped in improving the model

## 7 Team contributions

1) Enze Ma works on part of the model.py, data.py, engine.py, main.py and write the report.
2) Yixin Jiang works on part of the model.py, data.py, engine.py, main.py and write the report.
3) Fangqi Yuan works on part of the model.py, data.py, engine.py, main.py and write the report.

## 8 References

Wang, F., Kong, T. , Liu, H., Zhang, R., & Li, H. (n.d.). Papers with code - self-supervised learning by estimating twin class distributions. Self-Supervised Learning by Estimating Twin Class Distributions — Papers With Code. Retrieved February 21, 2022, from https://paperswithcode.com/paper/self-supervised-learning-by-estimating-twin-1

Lee, K.-H., He, X., Zhang, L., & Yang, L. (n.d.). Papers with code - cleannet: Transfer learning for scalable image classifier training with label noise. CleanNet: Transfer Learning for Scalable Image Classifier Training with Label Noise — Papers With Code. Retrieved February 21, 2022, from https://paperswithcode.com/paper/cleannet-transfer-learning-for-scalable-image