MATH6380P Project II: Image Classification of Semiconductors

Yue Guo
(20490720)
HKUST
`yguoar@connect.ust.hk`

Hao He
(20751497)
HKUST
`hheat@connect.ust.hk`

He Cao
(20748414)
HKUST
`hcaoaf@connect.ust.hk`

Haoyi Cheng
(20715302)
HKUST
`hchengaj@connect.ust.hk`

Abstract

Inspecting the quality of semiconductors is laborious and time-intensive, which motivates us to model this task as an image classification problem, using deep learning methods to classify the defective products from the good products. In this task, we want to find out the best classification model achieving the highest AUC score. We fine-tuned 7 different pre-trained classification models using our training set, and designed novel model refinement methods including data augmentation, stacked pooling and label smoothing tricks, and training refinement methods including progressive resizing, two-stage training and cosine annealing to boost the performance of the single models. We further do the model ensemble to improve the generalization ability. Our extensive experiments find out that EfficientNet-b7 is the most powerful single model, and the result of every single model can be further improved by the ensemble model.

1 Introduction

Image classification task has been greatly improved due to the development of deep learning and is now widely applied in industrial inspection to automatically detect unqualified products. In the semiconductor industry, because semiconductors have high demand in various fields which could be very distinctive from each other, it is important to ensure that functionality matches the desired function during production stages. In order to effectively find out the defected semiconductors among a large number of products with high precision, we need an improved image classification algorithm. In this project, we work on the semiconductor image classification task using the dataset provided by Nexperia. In this task, we are provided 34459 images of semiconductors, labeled as good or defective, and our goal is to use this dataset to train an image classifier such that it can generalize well on the unseen data.

To start our project, we begin with an Exploration Data Analysis, reporting the class imbalance problem and finding different kinds of noise in the dataset, which guides us to do the data preprocessing procedure. Then we set up the model using the pre-trained ResNet50, and fine-tune the model by the training set. We involve many refinement methods including stacked pooling, data augmentation, progressive resizing, two-stage training and cosine annealing. We also conduct a detailed study on the ensemble models. By aggregating different models, we can alleviate the problem that a single model may fall into the local minimum and overfit the training data. By doing lots of experiments, we find out that the EfficientNet-b7 model can achieve the best result among the single models, and the ensemble of ResNet50+Dense121+Dense169+EfficientNet-b5+EfficientNet-b7 can achieve the overall best result.

To conclude, we make the following contributions:

- We have a detailed Exploration Data Analysis(EDA), and report the features of the dataset.
- We solve the image classification task by fine-tuning the pre-trained models and use many model refinement methods to improve the model performance.
- We study the results of ensemble models, and find out that ensemble models can improve the generalization.
- We conduct a large number of experiments, and empirically find out the best single model and the best ensemble model.

This report is organized as follows: Section 2 provides the data description, section 3 describes the model setup, and section 4 gives the experimental results and analysis. Finally, we conclude in section 5.

2 Dataset Description

2.1 Exploration Data Analysis

During our initial exploration of the dataset, we have noticed that there are quite a few noisy training data. Therefore, we have cleaned the dataset accordingly. Here are some EDA tricks we used to clean the dataset.

2.1.1 Unlabeled data

There are 7039 defects images, however, only 6696 of them have bounding box labels. We have checked the 343 unlabeled data, here are some examples from the unlabeled data.

Most of the defective semiconductor chips have a scratch in the middle, and look apart from the below images, therefore, for our final model, we have removed them to reduce the noisy label. Shown in Figure 1

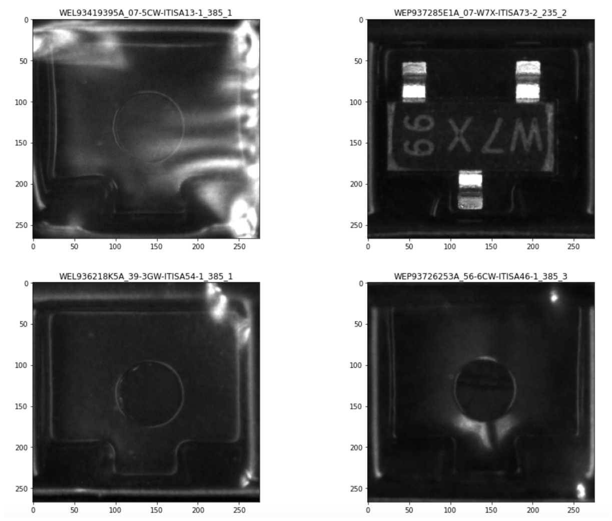

WEL93419395A_07-5CW-ITISA13-1_385_1
WEP937285E1A_07-W7X-ITISA73-2_235_2
WEL936218K5A_39-3GW-ITISA54-1_385_1
WEP93726253A_56-6CW-ITISA46-1_385_3
0
50
100
150
200
250

Figure 1: Unlabeled Data

2.1.2 Class Imbalance

We have 34459 training images in the dataset, good vs defect has a roughly 4:1 ratio. If we are only outputting good class, we can have about 80% accuracy. Therefore, accuracy will not be a good evaluation metrics. To address this issue, in our final model, we have implemented a weighted dataloader for batch-wise oversampling. Shown in Figure 2

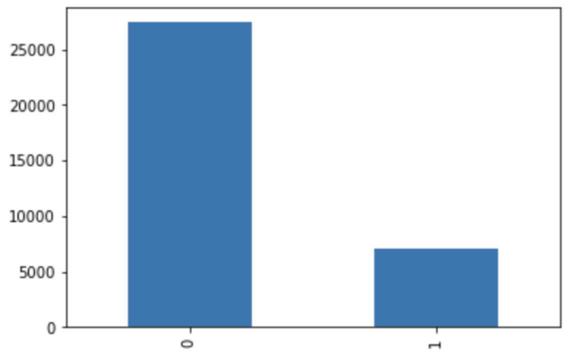


Figure 2: Good V.S. Defect in Dataset

2.1.3 Remove Duplicates

In order to find duplicates in the dataset, we use md5 hash function to hash the entire dataset. The idea is that if two images are identical, their hash value will be the same. By hashing the entire 38289 images through python thread, we have found 36 duplicates in 44 seconds. We also noticed that most of the duplicates are coming from the image with a name containing APG_ITIS tags. For our final model, we have removed those duplicates. Shown in Figure 3

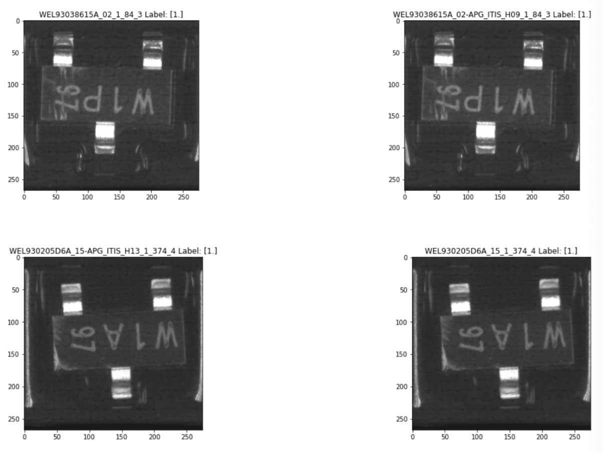

WEL93038615A_02_1_84_3 Label: [1.]
WEL93038615A_02-APG_ITIS_H09_1_84_3 Label: [1.]
WEL930205D6A_15-APG_ITIS_H13_1_374_4 Label: [1.]
WEL930205D6A_15_1_374_4 Label: [1.]

Figure 3: Duplicates in Dataset

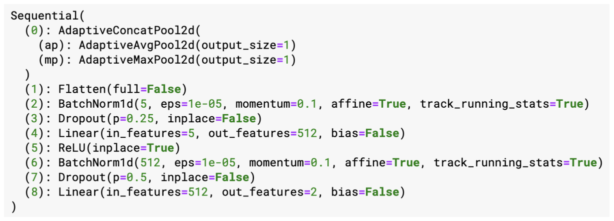
3 Model

3.1 Baseline

In this section, we will first introduce how we set up the baseline model, and follow by a few model adjustment that we used to reduce the noisy labels in the dataset. Finally, we will discuss some training refinements that we implemented to reproduce our results.

3.1.1 Experiment Setup

We use ImageNet pre-trained ResNet-50 as our baseline model. However, we empirically find a specific model head that will be more suitable for the dataset. Therefore, we have removed the pre-trained ResNet-50 head, and a customized head is attached to the ResNet-50 backbone.



```
Sequential(
  (0): AdaptiveConcatPool2d(
    (ap): AdaptiveAvgPool2d(output_size=1)
    (mp): AdaptiveMaxPool2d(output_size=1)
  )
  (1): Flatten(full=False)
  (2): BatchNorm1d(5, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (3): Dropout(p=0.25, inplace=False)
  (4): Linear(in_features=5, out_features=512, bias=False)
  (5): ReLU(inplace=True)
  (6): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (7): Dropout(p=0.5, inplace=False)
  (8): Linear(in_features=512, out_features=2, bias=False)
)
```

Figure 4: Customized Model Head

3.1.2 Sampling Strategy

In the training dataset, the ratio between good and defect is approximately 4:1. Therefore, if we uniformly sample a training batch from the dataset, our model learned from such dataset will be biased towards the good class, and cannot learn the feature of defect class effectively. To address this problem, we involve a weighted dataloader during the training procedure. Specifically, we sample the image of good class with probability $1/\#good$, and sample the image of defect with probability $1/\#bad$, where the $\#good$ and $\#bad$ denote the total number of images of good class and bad class respectively. With this balancing strategy, we can make the class ratio of the training batch approximately 1:1.

3.1.3 Model refinement

Stacked Pooling In this study, we investigate and compare two typical kinds of pooling layers for the downstream classification tasks: average and max pooling. Average pooling involves calculating the average of the given feature map, by using the AdaptiveAvgPool in PyTorch, we are able to output 1024 features for any given size image. Max pooling involves getting the max value out of any given feature map, by using the AdaptiveMaxPool, we are able to output 1024 features for any given size image from the ResNet-50 backbone. In our experiment, we found that stacking the two pooling layers boost our result by about 1%. We think this model refinement gives the downstream model head twice much about information from the backbone feature extraction, therefore, yielding a better classification result.

Data Augmentation In addition to the model refinement, another key observation we noticed from the experiment setup section is the set of Augmentation operations. As we pointed out in Section 3, since the dataset is heavily imbalanced, oversampling is implemented in the later models. Data argumentation is key to avoid overfitting as we have to duplicate the minority class in order to match the majority class. We found the following argumentation parameter works the best for our experiment.

- Random flip vertically and horizontally
- Rotating the image by 10 degree
- Zoom the image by 1.1
- Increase image contrast and lightning by 0.2
- Apply a perspective warping by 0.2

All argumentation is applied by 50% chance, and the original image is applied if none of the argumentation is selected.

Label Smoothing CrossEntropy As we noticed in Section 3, the dataset is imbalanced and contains noisy labels. We will address the class imbalance by oversampling and data argumentation, however, mislabeled data will significantly impact our model performance. By applying Label Smoothing Crossentropy loss, we are able to give room for the mislabeled mistakes that resulting in overconfidence prediction. We can also reduce the overfitting since the correct prediction are scaled by the hyper-parameter ϵ.

$Label\,Smoothing\,Crossentropy\,Loss = (1-\epsilon) * ce(i) + \epsilon * \sum \frac{ce(j)}{N}$

The i and j here are different labels, therefore, the model is able to leave room for mistakes. By using the ϵ, we can constrain the model from overconfidence and thus boosting our model performance.

3.1.4 Training Refinement

Progressive Resizing Transfer learning is widely used in computer vision tasks, the key idea is that by using the pre-trained weights, the model is able to extend its knowledge learned from other datasets to the domain-specific dataset. We extend this idea by first feeding the model with a smaller size image (resize to 128), and training the model until overfitting. Now we consider the current model has extended its knowledge from ImageNet to our dataset. We reapply the model that trained on the smaller size image to a bigger size image (resize to 224), and retrain the model. We found that we are able to boost the model accuracy by another 1%. The key observation behind this finding is that we have created a pre-trained model on the specific dataset domain, and by leveraging different size images, we increase the training datapoints and are able to reduce overfitting.

Two stage training In our baseline model, we have implemented a specific model head with a couple more linear layers. Therefore, during all our training steps, we have implemented a two-stage training process. We first freezing the backbone from the pre-trained ResNet-50, and warming up the special model head we are implemented by 3 epochs. After the first training is done, we unfreeze the backbone and finetuning the entire model by about 10 epochs. We apply this two-stage training process to all our models and observe all models have better performance under this training refinement.

Cosine annealing Learning rate annealing is important for better convergence. During our experiment, we have found that cosine-annealing works best for our experiment. For our hyper-parameter setup, the learning rate is set at 1e-3 for stage-1, and the group of learning rate is split for finetuning (5e-7,5e-6,5e-5) for stage-2. In addition, the cosine learning rate is the first increase in the 30% of the training steps and 70% decrease for the rest of the training steps. The intuition behind this training refinement is allowing the model to train at a higher learning rate while in the later part allow it to converge. During our experiment, we noticed by applying such an annealing method, the model requires fewer epochs to converge than other annealing methods.

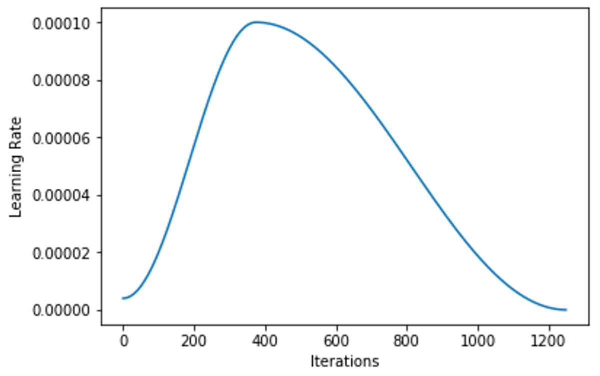


Figure 5: Cosine Learning Rate

3.2 Ensemble Model

Ensemble modeling is a process where multiple diverse models are created to predict an outcome, either by using many different modeling algorithms or using different training data sets. The ensemble model then aggregates the prediction of each base model and results in one final prediction for the unseen data(test data). The motivation for using ensemble models is to reduce the generalization error of the prediction. Since when training a single model, the overfitting problem might be a big issue. As long as all the base models are diverse and independent, the prediction error of the model decreases when the ensemble approach is used. Ensemble models have been a popular technique applied in many data-mining scenarios, so here we try to apply this technique and compare the ensemble one with single models we picked.

Neural network models are nonlinear and might have a high variance, which can be frustrating when preparing a final model for making predictions. The reason is neural network training usually learns via a stochastic training algorithm which means that they are sensitive to the specifics of the training data and may find a different set of weights each time they are trained, which in turn produce different predictions. By training multiple base models and bind their results together can reduce the variance. The prerequisite for multi-model ensembling to work is that each model has its own characteristics, and the error predicted by each model is different. Combining multiple models makes the final prediction result will add a bias, and this bias will be offset with the variance of the neural network so that the model's prediction will not be too sensitive to the details of the training data, the choice of training scheme and the contingency of a single training run.

The oldest and still most commonly used ensembling approach is called a "committee of networks". A collection of networks with the same configuration and different initial random weights is trained on the same dataset. Each model is then used to make a prediction and the final prediction is calculated as the average of the predictions. In this section, we follow this simple idea by choosing 5 different base models, including *ResNet50*,*ResNet34*,*DenseNet121*,*DenseNet169*,and traditional famous network *AlexNet*. We trained the five models separately using the same training schemes, such as setting the same learning rate and fixed epoch number,5. Then according to each network's performance on test data, we manually design different combinations and average the predictions of componential networks to make the final prediction. Based on the experiment results, we observed that the ensemble methods can effectively reduce the variance of a single model and can achieve a higher performance than any single model. Detailed experiment results will be shown in Section 5.

4 Experimental Results and Analysis

4.1 Overall Result

We conduct experiments with 7 pre-trained models, namely, *ResNet50*, *ResNet34*, *DenseNet121*, *DenseNet169*, *AlexNet*, *EfficientNet-b5* and *EfficientNet-b7*. We search the best hyperparameters according to the performance on validation set. The best result is reported in table 1 and all results are reported in table 2. For the single model, *EfficientNet-b7* has the best performance, achieving AUC score of 0.99750. The best model overall is the ensemble of *ResNet50+dense121+DenseNet169+EfficientNet-b5+EfficientNet-b7*.

We can have some interesting observations from table 2. Firstly, comparing within a family of the models (i.e. ResNet34 vs ResNet50, DenseNet121 vs DenseNet169, EfficientNet-b5 vs EfficientNet-b7), the larger model always has better performance than the small model. This result echos the double descent theory: with the improvement of the model complexity, deep learning models would not overfit the data, instead, they can have better generalization performance. This finding encourages us to use complex deep models to do the image classification task.

EfficientNets beat other image classification models in our experiments. EfficientNets are developed based on AutoML and Compound Scaling. In particular, the AutoML Mobile framework is used to develop a mobile-size baseline network, named EfficientNet-B0; Then, EfficientNet-B1 to B7 are built by compound scaling method to scale up this baseline. EfficientNets family has comparable model complexity (in terms of the number of parameters) with ResNet and DenseNet, but has better performance, which indicates the efficiency of the newly designed methods.

Table 1: Best Single model and Ensemble model performances

Model	AUC Score
ResNet50(baseline)	0.99608
EfficientNet-b7	0.99750
res50+dense121+dense169+Eff-b5 + Eff-b7	**0.99873**

4.2 Ensembling result

In the experiment, to prove the effectiveness of the ensemble methods, compared to the base model *ResNet50*, we did not choose recent years' top network structures. In contrast, we choose*ResNet34*, *DenseNet121*, *DenseNet169* and *AlexNet*,*EfficientNet-b5*. Some of above networks are older and have simpler structures than *ResNet50*, can be defined as weaker classifiers. But our ensemble experiment results still prove the ensemble methods can gain better classification accuracy than any single model. The details of result shown in Table1. The best ensemble combination attributes to ensemble of (*ResNet50*,*DenseNet121*,*DenseNet169*,*EfficientNet-b5*), whcih gains the best auc score of 0.99841. Compared to other two combinations, we can draw the conclusion that *AlexNet* and *ResNet34* will lag the final performance.

Table 2: Single model and Ensemble models' performances

Model	AUC Score	Notes
ResNet50	0.99608	baseline
ResNet34	0.99157	
DenseNet121	0.99597	
DenseNet169	0.99654	+
AlexNet	0.95444	
EfficientNet-b5	0.99686	+
EfficientNet-b7	<u>0.99750</u>	+
all*	0.99684	+
res50+res34+dense121+dense169	0.99663	+
res50+dense121+dense169	0.99834	+
res50+dense121+dense169+Eff-b5	0.99841	+
res50+dense121+dense169+Eff-b5 + Eff-b7	**0.99873**	+

* + means models' performance gains improvement compares to base model *ResNet50*.

5 Conclusion

In this project, we solve the image classification task on semi-conductor inspection. We test the performance of different models through a large number of experiments and involves model refinement methods, training refinement models, and model ensemble to boost the performance.

References

[1] M. Tan & Q. V Le. (2019) EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.

[2] A. Krizhevsky, I. Sutskever &G. E. Hinton. (2012) ImageNet Classification with Deep Convolutional Neural Networks.

[3] K. He, X. Zhang, S. Ren &J. Sun. (2015) Deep Residual Learning for Image Recognition.