Math 6380O Final Project Report: Semiconductor Classification by Making Decisions with Deep Features

Shichao Li
Department of Computer Science and Engineering, HKUST
slicd@connect.ust.hk

Ziyu Wang
Department of Computer Science and Engineering, HKUST
zwangbm@connect.ust.hk

Zhenzhen Huang
Department of Mathematics, HKUST
zhuangby@connect.ust.hk

Abstract

We explored both unsupervised and supervised anomaly detection in this project using deep learning techniques. For unsupervised experiments, we extract deep features from semiconductor images and utilized several unsupervised classification techniques. For supervised experiments, we train a deep neural decision forest (NDF) to detect anomalies. NDF is an ensemble of decision trees with soft decision functions, where we use densenet features to help it make decisions. The final AUC for the in-class competition is 0.962. The code is released at repository (supervised)[1] and repository (unsupervised)[2].

1 Unsupervised anomaly detection

1.1 Feature extraction

During unsupervised learning, we used the training images but did not use the labels. In the pre-processing step, we used ResNet-18 model [1] which was pre-trained on ImageNet [2] to extract features from the training images. The output feature vector length is 1000.

1.2 Methodology

We used three unsupervised machine learning methods: K-Means clustering [3], One-class SVM (Support Vector Machines) [4] [5] and Isolation Forest [6] which are implemented in [7].

K-means clustering separates the dataset into k clusters such that the data belongs to the cluster whose mean is closest to it.

One-class SVM is an unsupervised outlier detection method. It estimates the distribution of the data which has one class and predicts whether a new instance belongs to the data or not.

[1] https://github.com/Nicholasli1995/VisualizingNDF
[2] https://github.com/zuewang/MATH6380

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Table 1: Performance of unsupervised methods using 10-fold cross-validation.

Method	Average training accuracy	Average testing accuracy	Average area under ROC score
K-Means	0.498	0.500	0.502
One-class SVM	0.521	0.519	0.559
Isolation Forest	0.828	0.829	0.526

Isolation Forest is based on decision trees. This method uses random partitioning during training. Because the probability of selecting an outlier is far less than that of a normal data (which is also true for the dataset used in this project), after the random partitioning, the anomaly data will be closer to the tree root.

1.3 Experiment

During the experiments, we split the training dataset into 10 folds and calculated the AUC for ROC scores. For K-Means clustering, we set the number of clusters to be 2, and trained the model with 1000 iterations. For One-class SVM and Isolation forest, we used the default parameters during training.

The results are shown in Table 1. The training and testing accuracies of K-Means are close to 0.5, which indicates that the model can not distinguish bad semi-conductors with pre-trained Resnet18 features. One-class SVM performed a little better than K-Means. Isolation Forest had the best accuracy among the three methods yet is still unsatisfactory.

2 Supervised anomaly detection using deep neural decision forest

The un-supervised decision forest using pre-trained feature extractor in Section 1 does not perform well, here we use supervised decision forest with learn-able deep features.

2.1 Model description

Deep neural decision forest (NDF) was proposed in [8] to combine the representation power of deep convolutional neural networks (CNNs) and the divide-and-conquer idea of decision trees. Traditional decision tree assumes the features of training data are already computed, and it is constructed by heuristic node-splitting rules such as entropy minimization. During training, the feature vectors stay the same but the tree topology changes (grows). On the contrary, NDF uses deep networks to extract features from training data, which is used by decision trees with pre-defined topology for decision-making. During training the network parameters (and hence the feature vectors) change but the tree topology is fixed.

NDF is in nature an ensemble of decision trees that make soft decisions based on features extracted by deep networks. Each tree consists of splitting nodes and leaf nodes. In general each tree can have unconstrained topology but here we use full binary trees for simplicity. We can index the nodes sequentially with integer i as shown in Fig. 1.

A splitting node $\mathcal{S}_i$ is associated with a recommendation (splitting) function $\mathcal{R}_i$ that extracts deep features from the input $\mathbf{x}$ and gives the recommendation score $s_i = \mathcal{R}_i(\mathbf{x})$ which represents the probability that the input is recommended (routed) to its left sub-tree.

We denote the unique path from the root node to a leaf node $\mathcal{L}_i$ a computation path $\mathcal{P}_i$. Each leaf node stores one function $\mathcal{M}_i$ that maps the input into a prediction vector $\mathbf{p}_i = \mathcal{M}_i(\mathbf{x})$. To get the final prediction $\mathbf{P}$, each leaf node contributes its prediction vector weighted by the probability of taking its computation path as

$$\mathbf{P} = \sum_{i \in \mathcal{N}_l} w_i \mathbf{p}_i \tag{1}$$

and $\mathcal{N}_l$ is the set of all leaf nodes. The weight can be obtained by multiplying all the recommendation scores given by the splitting nodes along the path. Assume the path $\mathcal{P}_i$ consists of a sequence of q

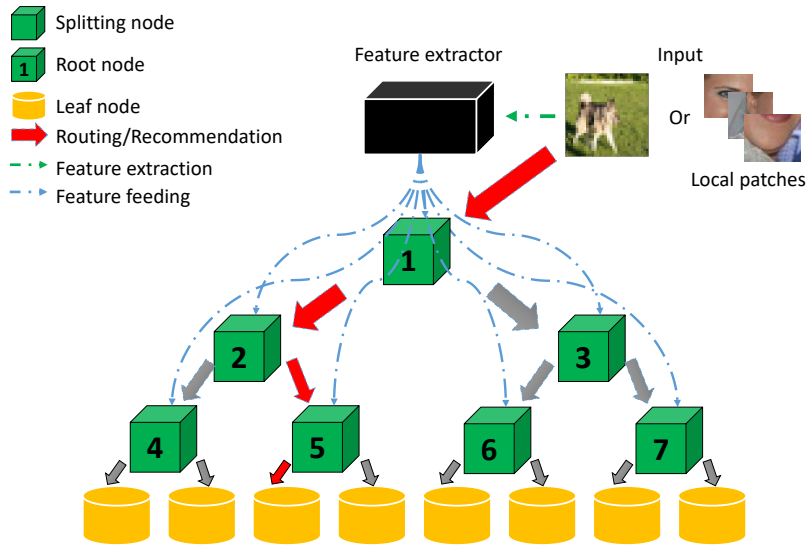

Figure 1: Figure taken from [9]. Illustration of the decision-making process in deep neural decision forest. Input images are routed (red arrows) by splitting nodes and arrive at the prediction given at leaf nodes. The feature extractor computes deep representation from the input and send it (blue arrows) to each splitting node for decision making. Best viewed in color.

splitting nodes and one leaf node as $\{\mathcal{S}_{i_1}^{j_1}, \mathcal{S}_{i_2}^{j_2}, \ldots, \mathcal{S}_{i_q}^{j_q}, \mathcal{L}_i\}$, where the superscript for a splitting node denotes to which child node to route the input. Here $j_m = 0$ means the input is routed to the left child and $j_m = 1$ otherwise. Then the weight can be expressed as

$$w_i = \prod_{m=1}^{q} (s_{i_m})^{\mathbb{1}(j_m=0)} (1 - s_{i_m})^{\mathbb{1}(j_m=1)} \tag{2}$$

Note that the weights of all leaf nodes sum to 1 and the final prediction is hence a convex combination of all the prediction vectors of the leaf nodes. Also note here all computation paths will contribute to the final prediction, unlike traditional decision tree where only one path is taken for each input. In addition, we assume the recommendation and mapping functions mentioned above are differentiable and parametrized by Θ_i at node i. Due to the formulation of soft routing in Equation (2), the final prediction is a differentiable function with respect to all Θ_i. A loss function defined upon the final prediction can hence be minimized with gradient-based algorithms.

In the special case where the mapping functions are constants (assumed in this project), the leaf node parameters can be updated according to leaf node update rules for certain loss functions [8]. These leaf node update rules guarantee to find the global minimizer for leaf node parameter when the splitting node parameters are fixed. In this case, the splitting node and leaf node parameters are optimized alternately where the former can be optimized by gradient descent and the latter by update rules.

2.2 Training process

We use the alternate optimization method mentioned above, where the splitting node parameters are optimized by gradient descent and the leaf node parameters optimized by update rules.

For a splitting node $\mathcal{S}_i$, we denote nodes in its left and right sub-trees as node sets $\mathcal{S}_i^l$ and $\mathcal{S}_i^r$, respectively. We denote the probability of recommending the input $\mathbf{x}$ to a leaf node $\mathcal{L}_i$ as $\mathbb{P}(\mathcal{L}_i|\mathbf{x})$. The optimization target is to minimize the negative log-likelihood loss over the whole training set containing N instances $\mathbb{D} = \{\mathbf{x}_t, y_t\}_{t=1}^N$,

$$L(\mathbb{D}) = -\sum_{t=1}^{N} \log(\mathbb{P}(y_t|\mathbf{x}_t)) \tag{3}$$

Table 2: Performance of NDF for Kaggle semiconductor classification.

Method	Backbone	Tree number	Depth	Tesing AUC
NDF	Densenet161	20	6	0.950
NDF + bagging (2)	Densenet161	20	6	0.962

where $\mathbb{P}(y_t|\mathbf{x}_t)$ is the predicted probability that the input belongs to class y_t (the y_tth entry in the final prediction vector $\mathbf{P}$). This loss function is differentiable with respect to each recommendation score s_i, and the gradient $\frac{\partial L(\mathbb{D})}{\partial s_i}$ can be computed as [8],

$$\sum_{t=1}^{N}\left(\frac{\sum_{\mathcal{L}_j\in\mathcal{S}_i^r}\mathbf{p}_j(y_t)\mathbb{P}(\mathcal{L}_j|\mathbf{x}_t)}{(1-s_i)\mathbb{P}(y_t|\mathbf{x}_t)}-\frac{\sum_{\mathcal{L}_j\in\mathcal{S}_i^l}\mathbf{p}_j(y_t)\mathbb{P}(\mathcal{L}_j|\mathbf{x}_t)}{s_i\mathbb{P}(y_t|\mathbf{x}_t)}\right) \tag{4}$$

This gradient is back-propagated into each splitting node to optimize its parameters.

We use constant mapping function $\mathcal{M}_i(\mathbf{x})=\mathbf{p}_i$ at each leaf node, which means we simply store the answers at the leaf nodes. When the splitting nodes are fixed, the answer at each leaf node can be updated iteratively [8],

$$\mathbf{p}_j^{T+1}(y)=\frac{1}{Q_j^T}\sum_{t=1}^{N}\frac{\mathbb{1}(y_t=y)\mathbf{p}_j^T(y_t)\mathbb{P}(\mathcal{L}_j|\mathbf{x}_t)}{\mathbb{P}(y_t|\mathbf{x}_t)} \tag{5}$$

where Q_j^T is a normalization factor to ensure $\sum_{y=1}^{|\mathbf{p}_j|}\mathbf{p}_j^{T+1}(y)=1$. The alternate optimization is carried out repeatedly to train the model.

2.3 Implementation details

In the experiment, we use densenet161 [10] to extract feature vector of length 2048 from the input images. Each splitting node contains a linear layer followed by sigmoid function to map the feature vector into the recommendation score s_i. We use 90% of the training data and the remaining is used for validation. We use Adam optimizer with learning rate 0.001 and train for 150 epochs with batch size 1024. 20 neural decision trees are used where each has a depth of 6, the final output is averaged over the 20 trees. For leaf node update we use a mini-batch of 2240 samples for each iteration. The training is conducted on a Nvidia Titan Xp GPU and takes around 1 hour.

2.4 Quantitative results

The model performance for the in-class Kaggle competition is summarized in Table 2. Since only 90% data is used for training, we re-sample the training data twice and trained two NDFs separately. The final testing prediction is given by the average of the two models, which corresponds to the second row indicated by bagging. In fact, we can achieve >99% training accuracy using all the training data yet the testing AUC is lowered. This indicates that NDF is strong enough to fit all the training data in this competition yet its generalization degrades due to over-fitting.

2.5 Visualization

To understand which part of the input can influence the decision-making of NDF more, authors in [9] takes the gradient of the routing probability with respect to the input and name it *decision saliency map (DSM)* [9],

$$DSM=\frac{\partial s_i}{\partial \mathbf{x}} \tag{6}$$

Some examples of DSM are included in repository (supervised)[3].

[3] `https://github.com/Nicholasli1995/VisualizingNDF`

3 Duty for each team member

Shichao Li is responsible for supervised anomaly detection by implementing and training NDF. Ziyu Wang prepared training data and conducted unsupervised anomaly detection. Zhenzhen Huang joined discussion and helped data preparation.

References

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

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[3] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[4] Marti A. Hearst. Support vector machines. *IEEE Intelligent Systems*, 13(4):18–28, July 1998.

[5] Bernhard Schölkopf, Robert Williamson, Alex Smola, John Shawe-Taylor, and John Platt. Support vector method for novelty detection. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 582–588, Cambridge, MA, USA, 1999. MIT Press.

[6] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 413–422, Washington, DC, USA, 2008. IEEE Computer Society.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[8] P. Kontschieder, M. Fiterau, A. Criminisi, and S. R. Bulò. Deep neural decision forests. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1467–1475, Dec 2015.

[9] Shichao Li and Kwang-Ting Cheng. Visualizing the decision-making process in deep neural decision forest. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 114–117, 2019.

[10] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.