Wide, Deep, and Recurrent: An End-to-End Neural Architecture for Automatic Fashion Product Labeling

Binbin Xiong, Minfa Wang
Stanford University
{bxiong,minfa}@stanford.edu

Abstract

In this article, we propose a novel neural network architecture for the iMaterialist Challenge (Fashion) at FGVC5[12]. We leverage transfer-learned Xception[2] as our base image feature extractor which we call it deep. On top of it, we propose two novel module components. One is to use recurrent network with pre-trained label embeddings to explicitly measure label dependency, the other is to combine our generalized deep model with shallow yet wide features such as color histogram and HOG as a way to enhance memorization. We use residual connection for the Wide, Deep and Recurrent modules so that the model could learn to weigh or even ignore each module automatically. To deal with class imbalance, we also leveraged label sampling and per-class threshold selection techniques which prove to be simple and effective. Our best single model achieves 0.648 F1 score on validation set and 0.643 test set. Our ensembled model achieves 0.656 F1 score on test set. The paper is organized as follows, we will first introduce related work followed by analysis of the dataset, then we introduce our methodology and show experimental results at last.

1. Introduction

Automatic labeling and classification for product photos has been a constant effort for online merchandise companies and could potentially improve the shopping experience for shoppers drastically. However, it is not an easy task to tackle because of the complex settings: different lighting, angles, backgrounds, and levels of occlusion, etc. The FGVC workshop created a dataset and invited the computer vision community to brainstorm for advancing this problem.

In this paper, we will work with the provided dataset to tackle down the problem of multi-label image classification. The input to our model is a product image, and the output should be the labels that the image belongs to. We propose a combined Deep, Wide and Recurrent model where Deep is for image understanding, Wide is for pattern memorization, and Recurrent is for label dependency understanding, to predict the probability for each of the class that the image belongs to, then we use validation set to find the optimum per-class thresholds to convert the output probability to actual labels. As required by the competition holder, we will be using F1 metric (micro-average) for our model evaluation.

2. Related Work

As the large-scale hand-labeled datasets such as ImageNet[5] being developed, the progress of image classification has been speeded up vastly. Traditional heuristic based feature extraction classification methods were soon surpassed by end to end deep neural net architectures with giant and deep convolutional layers. Specifically, in the area of multi-label classification, where each image could be classified as multiple labels where labels may be or not be mutual exclusive, there are a few different techniques each emphasize on slight different areas.

2.1. Deep Convolutional Networks

The basic trend towards a better multi-label image classifier is to go deeper, with hundreds of convolution layers. Ever since the success of AlexNet[13] in 2012, it has been proven effective well to stack many convolutional layers. However, modern computation efforts limited the depth and number of parameters, thus in VGG[19], the authors propose to use 3x3 filters as against to 7x7 layers to improve model parameter efficiency. Later on, InceptionNet[20] has further decreased the classification error by introducing inception unit. While the current state of art technique is from ResNet[8, 10, 3], where they use residual connection over layers to help them successfuly train more than 100 layers deep networks.

Such networks mainly focus on more effectively understand the images, while they don't focus on dealing with learning label or (label, image) dependencies, which is inevitably an important area for boosting the model perfor-

mance especially when label is sparse and skewed.

2.2. Explicity learn label dependency

In 2016, Wang el[22] utilized RNNs to explicitly learn laben dependencies between labels and proposed a CNN-RNN framework which learns a joint image-label embedding to characterize the semantic label dependency as well as the image-label relevance, and it can be trained end-to-end from scratch to integrate both information in a unified framework.

2.3. Classification through Learning to Ranking

Another technique which focused on multi class image classification is to treat classification as a learning to rank task[23], where a common technique is to rank the labels and then select a subset, either through top-k, or through thresholding. In 2017, Li el[14] proposed a method where they introduce a smoother loss function that allows a deep net to converge faster. In their proposal, they also treat the threshold for each class as a parameter to optimize for. They showed that their methods achieved best results on a variety of datasets at the time of publication.

3. Data Preparation

The challenge provides a dataset of over one million (fashion related) product photos for training, 40K images for testing and 10K images for validation. As shown in figure 1, each of the product photo is associated with multiple labels denoting special attributes with the product. The labels are given as encoded integers instead of strings, which makes the debugging work a bit more difficult.

There are several challenges we found in this dataset. One is the sizes of images vary a lot. In order to be compatible with pretrained models as well as increase training efficiency, we do the following to normalize each image: first we pad each image with pure white (255,255,255) to square according to its longest side, then resize them to 299*299*3 and rescale each pixel to range from 0 to 1. We choose padding instead of corping because objects appear at very different locations from image to image, and we choose white pixel padding because this is the background color for most of the image. The other problem is class imbalance. As shown in figure 3, among the 228 classes, some of the images appear much more frequently. In the next section, we will describe how we use weighted sampling to alleviate this imbalance issue.

3.1. Class imbalance Adjustments

The fashion dataset contains 228 unique labels. From 3 we could see that the frequency of different labels showing on the dataset could vary drastically. This type of data imbalance imposes challenge of model generalization. In

Figure 1: Four example training images. Images vary in size, and could contain multiple product objects.

our experiment, we observed that when we were training the model with uniform weights across all examples, then in the validation set over 70 labels did not appear in the predictions even once.

In order to address the class imbalance issue, we referred to prior art and took a similar approach that GloVe[11] used to train the word embeddings. In their approach, the overall weight of a word would increase smoothly with its increase of frequency but in a slower pace, until it reaches a threshold and becomes plateau. In our case, we didn't use the maximum threshold because the label imbalance for image tagging is not as drastic as word occurrences. Formally, the adjusted weight satisfies the following equations:

$$\begin{cases} (w_{l_1} \cdot f_{l_1})/(w_{l_2} \cdot f_{l_2}) = (f_{l_1}/f_{l_2})^{\alpha} \ \ \forall l_1, l_2 & (1) \\ \sum_l (w_l \cdot f_l) = \sum_l f_l & (2) \end{cases}$$

In the equations, l denotes a label. w_l denotes the weight of a label, and f_l the total frequency of a label. The first equation is the part inspired by GloVe[11], and we use the same $\alpha = 3/4$. The second equation is to make sure the overall weights across all examples on all of their occurrences stay the same. It will subsequently ensure that the average image weight will be 1, and the loss will be at the same scale. So no additional hyper-parameter tuning is needed for training the network.

The weight of an image W_I is simply the average of weight of labels in that image (given image I contains n_I

labels):

$$W_I = 1/n_I \sum_{l \in I} (w_l)$$

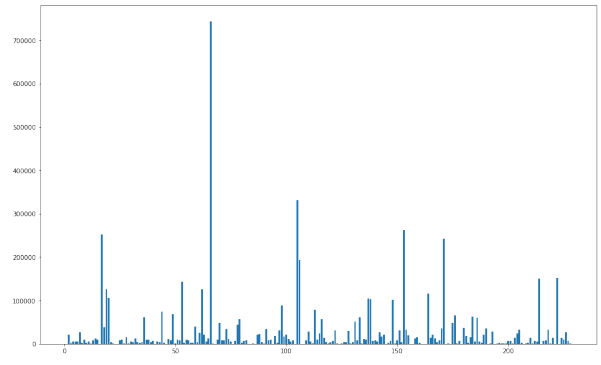

Figure 2: Data distribution of different labels.

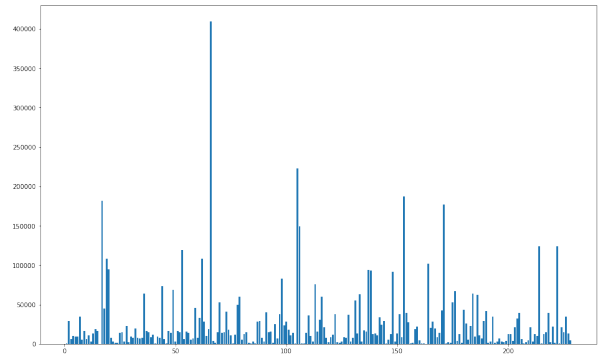

Figure 3: Data distribution of different labels after weight adjustment.

3.2. Label Counts Imbalance

Besides the imbalance of appearance count of unique labels, the distribution imbalance of the number of labels tagged in each image, illustrated in fig4, could also cause bias to the model.

The model performance may suffer most if we choose to have a recurrent layer at the end to make predictions. However, intuitively the dependency between different fashion tags would not be strong enough to predict them sequentially. Hence our model chose a fully-connected layer at the top, and we apply a fine grained per-class threshold selection algorithm detailed at later sections that makes the model less prone to this type of imbalance. So we chose to not directly adjust the weight of images based on label counts imbalance.

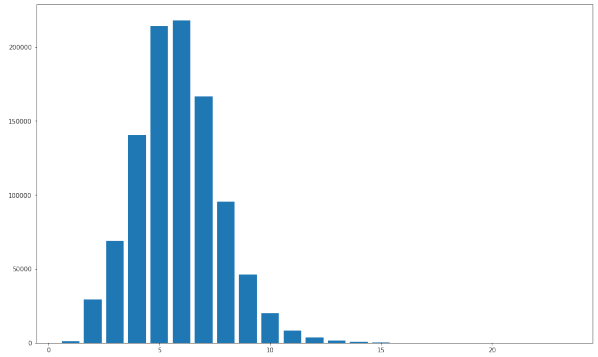

Figure 4: Distribution of number of labels tagged on an image.

4. Approach

Figure 5 shows our proposed architecture. As shown, we use concatenation to combine the output from each module, and use a final dense layer with sigmoid activation to map the feature into our 228 output class predictions. The concatenation for all has a few advantages over depending the output on only a subset of the module outputs. After concatenation, the final output $\hat{y}_i$ for each class i can be expressed as a linear combination of each module's outputs: $\hat{y}_i = \sum_j W_{ij} X_j$, where X_j denotes sub module's output. This gives the model the ability to learn how to weigh different module automatically. More over, since the deep feature is also put as the input for recurrent module, the concatenation is acting as a residual connection, which could future help regularize the recurrent module. In extreme cases, if some sub modules are of less use, the model could possibly even learn an all zero weight matrix, meaning totally ignoring the sub module.

In this section we elaborate on the details of how and why we choose to build the Deep, Wide and Reccurent modules in our network, and we also introduced the algorithm that we used for per-class threshold selection.

4.1. Deep: Transfer Learned Xception

Among a variety of modern DNN architectures for image classification, Xception[2] networks stands out in that it achieves state-of-art accuracy while with only around 23M parameters in 126 layers. For this reason, we chose Xception as our base deep module in this project. Xception takes 299*299 images as input and encode them into 2048 feature vectors before the final prediction layer. We use the 2048 feature vector as the output of Deep Module, then feed it to later networks. As shown in Figure 5, deep feature is feed into two modules: one for recurrent module, one for label prediction dense module. When passing to recurrent module, we first convert feature dimension by passing it through a dense layer. Later on, we use this image embedding as the

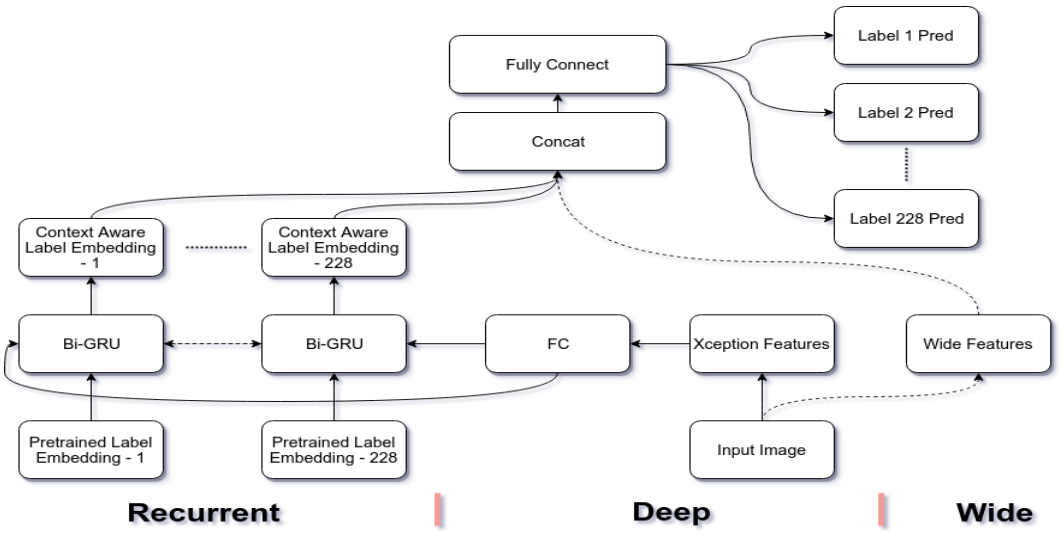

Figure 5: Our proposed Wide Deep and Recurrent Architecture.

initial hidden state for our Recurrent module. Learned from ResNet[8], where the authors show that residual connection is an effective technology for boosting model performance along with model complexity, we further connect the output of deep module directly to the final prediction dense layer as we believe that this residual connection could help regularize the recurrent module and make our combined module stronger.

Follow suggestions by Yosinski el[24], we use transfer learning techniques for the deep module. It's worth mentioning that we actually started this project with partially fixed fine tune, and we have similar observation as the authors of the paper, that is fine tune learning all layers with pretrained weights performs the best, while fine tuning only half layers with pretrained weights performs poor. We will elaborate more on this in section 5.

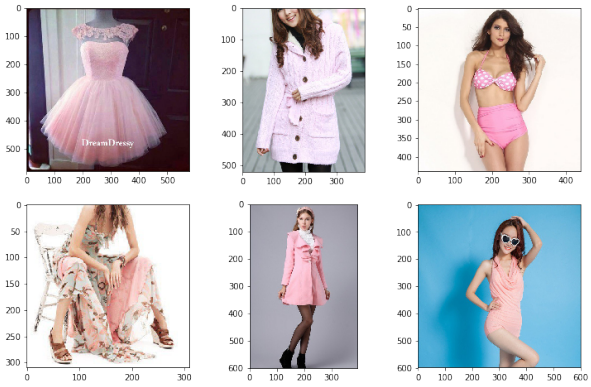

Figure 6: Sample images tagged with label-131.

4.2. Wide: Color and Gradient for Memorization

Even though the competition does not disclose the meaning the labels directly. By grouping images based on their labels together, we could reverse engineering and infer the meaning of some labels. In our observation, some labels may simply be the indications of some low level features of the image like color of the clothing. For example, Figure 6 is some sample images that tagged with label-131. It is likely for this label to be something related to pink color. This pattern inspire us to increase the strength of memorization of the network by adding additional shallow but wide image features to the network. Some prior art[9] has applied this technique in recommendation system and proven its effectiveness.

The features that we extracted from the images are oriented gradients (HOG)[17] and color histogram[18, 21]. These features will just be applied with a fully-connected layer and then concatenate to the last layer before making the final predictions.

When extracting color histograms of the image, we observed that for a large number of images, their pixel color is dominated by the white background. Hence we came up with a trick to ignore white background pixels in the histogram computation. The trick is that in real world it is rare for a pixel hue to be fully saturated. So we could simply drop the pixels that reach maximum hue. Despite its simpleness, this approach is effective as shown in Figure 7.

The feature extracted from the wide layers could be formally denoted as:

$$\mathbf{F}_{HOG} = FC(HOG(\mathbf{I}))$$
$$\mathbf{F}_{CH} = ColorHistogram(\mathbf{I})$$
$$\mathbf{F}_{W} = [\mathbf{F}_{HOG}; \mathbf{F}_{CH}]$$

The symbol $\mathbf{I}$ is the input image. For the color histogram, we simply take its output with the number of bins to be 50 as features denoted as $\mathbf{F}_{CH}$. For the histogram of gradients, we apply another fully connected layer on top to shrink its dimension to 100, and output its feature vector $\mathbf{F}_{HOG}$. The additional fully connected layer is used to balance out the influence of the two feature vectors to the final predictions. Then final feature vector $\mathbf{F}_{W}$ of the wide layers will be the concatenation of the two vectors.

Figure 7: Effect of background cropping. The left is original image, and the right is image with background cropped (shown in gray).

4.3. Recurrent: Label Dependency Through Sequence to Bag-of-Labels

With cross entropy loss and none weight sampling, the learning process treat each label the same independently. However, as shown in the previous section, our dataset has very sparse and skewed label distribution. As a result, for many classes, only a few thousands examples out of one million are positive impression, this strong imbalance may hurt the minority classes a lot. In order to alleviate this issue, besides using sample weights, we would also like to leverage the label dependencies. For example, when the "jeans" label appears in the image, then the "shorts" label would be less likely to appear in the same image, and "blue" label might be more likely to appear in the image. As has been shown effective in identifying sequential data in many applications [15], recurrent network is very good at capturing the potential data dependencies.

In our proposal, we treat the classification problem as a "Sequence to Bag of Labels" model. Following the convention of typical sequence to sequence model, we treat the image embedding feature as the input sequence, and then output a bag of labels sequence with learned dependency among them. The output of the deep model is passed as the initial hidden state vector, and we then treat each label as a time stamp for the output sequence, according to their label id. In each time stamp, the recurrent unit reads in a hidden state, the current label embedding, then output a context aware label embedding. We later on concatenate all of the context aware label embeddings along with other module features to the final dense layer for prediction. Since our output is more of bag of words rather than strict sequence, we use bidirectional recurrent network. For the recurrent unit, we choose GRU[1] here because it has relatively few parameters while acts as a top performing RNN unit.

4.3.1 Pretrained Label Embedding

Label embedding is a key module in the recurrent network. One simple way to construct the label embedding is to initialize them with random weights, and let gradient back propagation to adjust them accordingly. However, since we have 228 labels, and we simply put them in sequence according to their label id, it might be very hard for the recurrent unit to learn proper dependencies for such long spans, for example, label 1 and lable 228 could be similar, but since they are far away from each other, even though GRU is know to be effective against vanishing gradients, in practice it may still be hard to train.

Inspired by embedding training in natrual language processing[11, 16], we would like to pretrain the label embedding, so that similar, or co-occurrent labels would be close to each other in the embedding space, and let the recurrent module learn from there instead of randomness. Learned from GloVe[11], we train our label embedding based on their co-occurrence, Figure 8 shows the co-occurrence map for all 228 labels in log space. For simplicity, we use matrix factorization[4, 7] to learn label embeddings X_i to approximate

$$F_{ij} = X_i X_j$$

We learn such embeddings through minimizing the following square loss:

$$e_{ij}^2 = (F_{ij} - \hat{F_{ij}})^2 = (F_{ij} - \sum_{k=1}^{K} X_{ik} X_{jk})^2$$

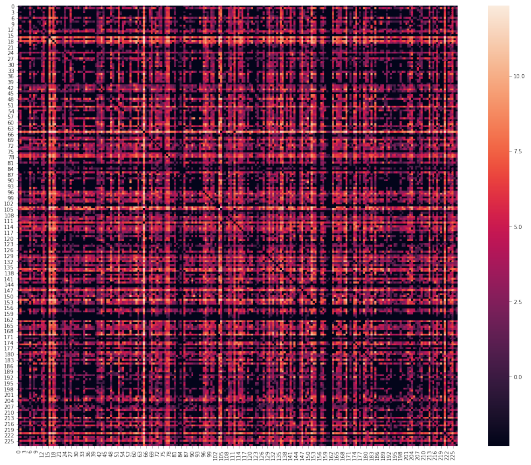

Figure 8: Label co-occurrence map in log space. Dark means not co-occurrent, red means co-occurrent. We can see that some class pairs are very co-occurent.

This way, we make sure that the more likely labels are co-occurrent, the closer they are in the embedding space.

4.3.2 Alternative Recurrent Architecture

One alternative architecture for recurrent module is to instead of concatenating the context aware label embeddings, we treat them separately. Since we assume after the recurrent module, each label embedding already contains the dependency information as we describe them as context aware. Compared to concatenation, this architecture significantly shrink the model size and is expected to be easier to train. As shown in Figure 9, the highlighted squares are the alternative parts.

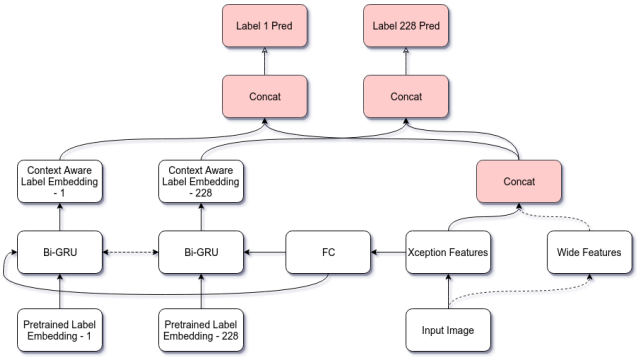

Figure 9: An alternative RNN architecture. Parts replaced are highlighted.

4.4. Per-class Threshold Selection

In the exit flow of our network, we use a dense layer to map the final input encoding to a vector of length 228 with sigmoid activation, which produces a vector of [0,1] values indicating the probability of the appearance for each label in the input class. Since we are using micro F1 in evaluation, threshold selection could play a crucial role. In this project, we leverage the simple "cyclic optimization procedure" as described in [6], where on the validation set, we iterate through each class, do a grid search for all possible thresholds, keep the one that results in the biggest overall micro F1, and repeat this process until converge. The reason for iteration is that micro F1 depends on each of the thresholds, unlike for macro F1, we could treat and optimize thresholds for each class independently. This simple technique has been proven effective on our dataset, and we well elaborate more in the experimental section.

4.5. Loss and Evaluation Metric

The challenge evaluates the model using F1 score. However, the F1 score is not differentiable and it imposes difficulty for the model to run gradient descent on. So in the training process, we used cross entropy loss as our objective to minimize:

$$CE = -\sum_i y_i log(\hat{y}_i)$$

where y_i is represents whether the *i*-th label is appeared in the image, and $\hat{y}_i$ denotes the probability of i-th label the model predicts it to appear in the image.

As required by the competition holder, we use Micro Averaged F1 for model evaluation, which is described as:

$$F1 = \frac{2PR}{P+R}, P = \frac{tp}{tp+fp}, R = \frac{tp}{tp+fn}$$

where tp is global true positive, fp is global false positive, and fn is global false negative.

5. Experiments

In this section we describe our experimental settings, experimental results, and model error analysis. If otherwise not stated, we train models on single GPU(Nvidia Tesla P100) with the full training set (about 1M images, excluding empty images). For all the experiments, we chose the following set of hyper-parameters:

- batch size: 64
- learning rate: 3e-4
- optimizer: Nesterov Adam

When loading the preprocessed images for training, we further applied image augmentation as a way to enhance our training data. Due to the data specialty, we choose only to

randomly flip the image instead of any rotation and color adjustment, since the prediction label is very color and texture sensitive.

5.1. Transfer Learning

We first use transfer learning to fine tune the Xception[2] network. At the beginning, we fix the network weights, and pass the 2048 feature encoding to the exit dense layer directly. This gives us about 0.51 validation F1. After the model is stable, we started to fine tune the model. We first tried to only fine tune half of the module. However, as shown in table 1, this doesn't give us any increase in the F1 score. Later on when we learned from [24], instead of gradually fine tune, we open all layers for tuning from the very beginning. In contrast, we easily achieved 0.61 validation F1 score with Xception. To examine the effectiveness of weighted sampling, we also fine-tuned another xception model with all layer trainable with our proposed sampling logic. This model only achieve slightly better F1 score than our transfered learned baseline, which from one hand indicates that the weighting method is effective, while on the other hand indicates that the weighting method may still be not strong enough. Also, we observed that with weighted sampling, the optimum threshold for F1 score increases (if choose universally), as it is 0.27 as compared to 0.23. Since ideally, threshold should be close to 0.5, even though the improvement is small, we still believe that weighted sampling method is effective, thus we apply this method to all later model trainings.

5.2. Training Wide and Recurrent Modules

For wide and recurrent model training, since our training data is over 1M, due to time limit, we choose to fix the deep module, and only train wide or recurrent weights with random initialization. For the deep module, we use weights trained in previous section, instead of using weights trained from ImageNet. As shown in table 1, adding wide feature slightly improves the baseline model, while adding RNN also slightly improves the baseline model. When combining them together, our single model achieves 0.62 F1 score on validation set. And if we train the per class threshold on the validation set, it can give us another 2.8% gain in F1 score, achieving 0.648. This result give support to our previous reasoning in designing our architecture, that is Wide helps with memorization, and Recurrent helps with learning complext label dependency.

Model	ValF1	TestF1
Deep(fixed all)	0.511	0.512
Deep(fixed half)	0.520	0.516
Deep(tune)	0.610	0.609
Deep+Weighted Sampling	0.612	0.610
Deep(tune)+Wide	0.615	0.613
Deep(tune)+Recurrent	0.618	0.615
Deep(tune)+Wide+Recurrent	0.620	0.616
Deep(tune)+Wide+Recurrent*	0.648	0.643
Ensembled	0.642	0.640
Ensembled*	**0.661**	**0.656**
Team@5th	-	0.699
Team@1st	-	0.72

Table 1: F1 score for each model on validation set and on test set, * means the result is further boosted with per label threshold selection algorithm.

5.3. Ensemble and Per-class Threshold Selection

For this project, we didn't explore complex ensemble techniques, rather, we simply ensemble our combined model with control models (Wide+Deep, Recurrent+Deep, Wide+Deep, Deep, Wide+Recurrent+Deep). We calculate the average sum over all the model prediction probabilities, then use the threshold selection algorithm we described in previous section on our validation set to set per class threshold. Compared to single model, we observed that this simple ensemble algorithm gives us around 2% performance gain, and the simple threshold selection algorithm gives us another 2.5% F1 gain, resulting in our final 0.656 F1 score. It is worth mentioning that on the kaggle kernel discussion, one top performer mentioned their best single model achieved 0.64% F1 score before ensemble and could achieve 0.71 F1 score with ensemble, while our non-tuned best single model achieved 0.648. We believe with hyper parameter fine-tuning and more advanced model ensemble, we could possibly reach 0.7 F1 score on the validation set.

5.4. Per-Label Error Analysis

In order to find out how well our model performs on labels with different frequency, we made plots that combined per-label precision/recall with its frequency. Figure 10 contains labels with frequency greater than 2000 in the validation set and figure 11 contains labels with frequency less than 20 in the validation set. From the figures we could see that for labels that show up very frequently, the model generally tends to be more aggressive on recalls, whereas the pattern of low frequent labels is very noise and in general the model tends to be a lot more conservative.

5.5. Saliency Map

Intuitively, when making predictions for different labels, the model should focus on different regions on the image.

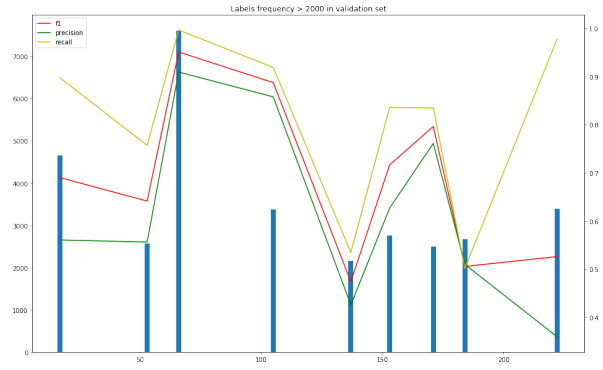

Figure 10: Per-label F1 and frequency for popular labels.

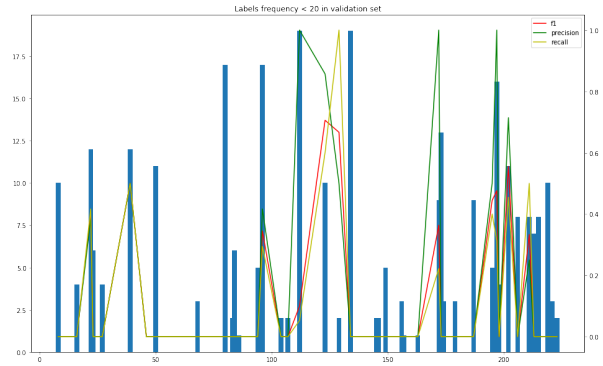

Figure 11: Per-label F1 and frequency for rare labels.

We used saliency map to help visualize and debug this behavior. Figure 12 shows an example of where the model pays attention to when it is making 8 distinct predictions. The bottom right image on that figure is the original image. We could see that despite of all the background noise, the model is smart enough to only focus on the main figure of the image. And depending on which label the model is trying to predict, it focuses on the different regions of the human body.

6. Conclusion and Future Work

In this paper, we proposed and experimented a novel Wide, Deep and Recurrent architecture for automatic fashion product labeling on the FGVC5 fashion dataset with over one million images. Deep for input image understanding, Wide for image label memorization, and Recurrent for label dependency understanding. As compared to the transfered learned Xception baseline model, our combined model showed slight improvement in F1 score. Our best single model achieves 0.648 F1 score on validation set which ranks 6 on the public leaderboard, while the 5th team has 0.699 F1 score. (Note that most of the work was done after the competition deadline, thus on the public board our

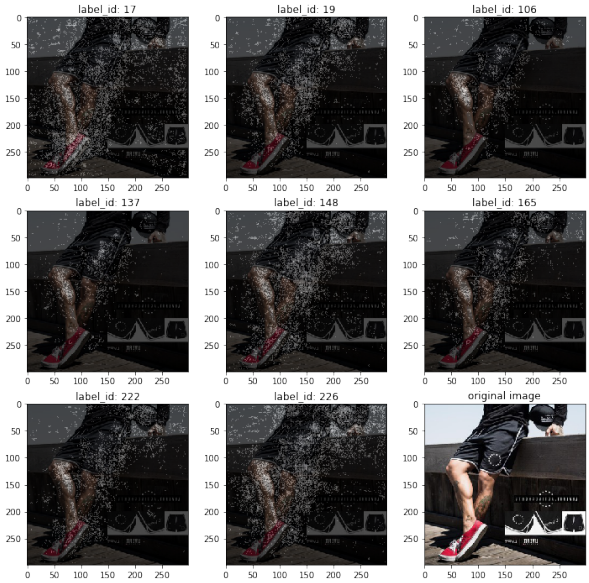

Figure 12: Saliency map of a fashion image. The bottom right is the original image. Two highlights: 1) the model could distinguish background noise from main figure, 2) the model focuses on different regions of the body when making different predictions.

team don't appear on 6th.)

For future work, we would like to experiment with more model combination, model training techniques such as distributing loss between each of the sub network, and more ensemble techniques, and hyper parameter tuning. Due to time and budget limitation, we haven't fine tune any of the hyper parameters and for some of the ideas that we proposed in this paper, we don't have chance to fully experiment on them, such as the alternative recurrent network, or more tailored label embedding training. We believe our proposed architecture could achieve much higher performance on this dataset.

7. Contributions And Acknowledgements

Overall, ideas in this projects were discussed by both team members and they contributed equally in this project.

Contribution for Binbin Xiong:

- Designed and implemented the training evaluation testing pipeline.
- Trained and fine-tuned the baseline model with transfer learning.
- Implemented per-class threshold selection.
- Implemented Recurrent module.

- Designed and implemented label embedding training.

Contribution for Minfa Wang:

- Implemented wide feature extractor module.
- Co-implemented the recurrent module.
- Implemented model ensemble.
- Designed and implemented class imbalance weight adjustments.
- Implemented various debugging and visualization techniques like saliency map.

For data downloading, we modified based on public code in the https://www.kaggle.com/nlecoy/imaterialist-downloader-util For wide feature extraction, we referenced code from the CS231N assignment 1. For salience map analysis, we referenced code from CS231N assignment 3.

At the end, we would like to thank the instructor and TAs for all the help during the course. It is the tailoredand well designed course material and projects that helped us to learn and reach this far.

References

[1] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[2] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint*, 2016.

[3] V. V. A. A. Christian Szegedy, Sergey Ioffe. Inception-v4, inception-resnet and the impact of residual connections on learning. 2016.

[4] A. Cichocki and A.-H. Phan. Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 92(3):708–721, 2009.

[5] 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.

[6] R.-E. Fan and C.-J. Lin. A study on threshold selection for multi-label classification. *Department of Computer Science, National Taiwan University*, pages 1–23, 2007.

[7] C. Févotte and J. Idier. Algorithms for nonnegative matrix factorization with the β-divergence. *Neural computation*, 23(9):2421–2456, 2011.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] J. H. T. S. T. C. H. A. G. A. G. C. W. C. M. I. R. A. Z. H. L. H. V. J. X. L. H. S. Heng-Tze Cheng, Levent Koc. Wide deep learning for recommender systems. *arXiv:1606.07792*, 2016.

[10] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, volume 1, page 3, 2017.

[11] R. S. Jeffrey Pennington and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[12] Kaggle. imaterialist challenge (fashion) at fgvc5 dataset, 2018. `https://www.kaggle.com/c/imaterialist-challenge-fashion-2018`.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[14] Y. Li, Y. Song, and J. Luo. Improving pairwise ranking for multi-label image classification. *CoRR*, abs/1704.03135, 2017.

[15] L. Medsker and L. Jain. Recurrent neural networks. *Design and Applications*, 5, 2001.

[16] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[17] B. T. N. Dalal. Histograms of oriented gradients for human detection. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference*, 2005.

[18] X. M. Rishav Chakravarti. A study of color histogram based image retrieval. *Information Technology: New Generations, 2009. ITNG '09. Sixth International Conference*, 2009.

[19] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015.

[21] H. N. T. T. Van Hieu Vu, Quynh Nguyen Huu. Content based image retrieval with bin of color histogram. *Audio, Language and Image Processing (ICALIP), 2012 International Conference*, 2012.

[22] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu. Cnn-rnn: A unified framework for multi-label image classification. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2285–2294. IEEE, 2016.

[23] J. Weston, S. Bengio, and N. Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, volume 11, pages 2764–2770, 2011.

[24] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.