Machine Learning in Practice

iMaterialist Challenge (Fashion) at FGVC5

Alex Suciu Brigel Pineti Laurens Kuiper Ankur Ankan Luca Parolo
Rick Dijkstra Tristan de Boer

June 2018

1 Introduction

In the field of Machine Learning, image recognition and image object recognition are popular topics of research. Both topics are challenging, as pictures can be taken in different lighting, angles, backgrounds and levels of occlusion. Many approaches tackling these problems involve neural network models. In the more recent years, applications demand more capabilities of these models, such as detecting- and correctly naming multiple objects and features present in the image. Images often need to be classified in more than one class. This process is called multilabel classification, which is the topic of this research. Multilabel classification often brings more challenges to the play, as multilabel datasets are often prone to imbalance [2].

We take part in a Kaggle machine learning competition on multilabel classification of iMaterialist Fashion images, where the goal is to correctly classify as many labels per image as possible.

In Section 2, we discuss the challenge and the evaluation of the competition. In Section 3, Related Work is treated and in Section 4, we discuss the used methods, of which results are reported in Section 5. Finally, in Section 6, our results are discussed, before concluding in Section 7.

2 Challenge: iMaterialist Challenge (Fashion) at FGVC5

The goal of this challenge is to create an automated product classification system for fashion products, by tagging them with labels. Each fashion product contains one or more labels, thus this problem is a multilabel classification problem. The exact meaning of each of the 228 labels is unknown, to avoid classification by hand. The challenge is particularly difficult because there is a huge variation in the method of data collection. The training set has images in different lighting conditions, angles, backgrounds, and levels of occlusion. Apart from the variation in data collection method, some classes, like dresses and skirts, are quite difficult to distinguish from each other.

Because of these issues, the Conference on Computer Vision and Pattern Recognition has partnered with Google, Wish and Malong Technologies to create this challenge for the Fine-Grained Visual Categorization workshop (FGVC5).

2.1 Data

Kaggle has provided the data in three separate *.json*-files, one for each training, validation, and test datasets. The files contain the image ids with an URL to the image. A script was used to extract the URLs from these files and download all images. The training set contains $1,014,544$ images, the validation set $9,897$, and the test set $39,706$. The complete dataset is approximately 65GBs in size. In total, $12,753$ images were either corrupted or had an invalid URL, and were ignored as it only contributed to 1.2% of the training set. In addition, the train and validation dataset contained a list of labels associated with each image id. There are 228 labels in total and each label is represented using an integer value ranging from 1 to 228.

The dataset is heavily imbalanced, as seen in Fig. 1. For example, label-66 occurs $743,250$ times, while label-84 occurs only 52 times. With imbalanced classes, class labels assigned to each instance are not equally assigned, which could affect the mean F_1 score [16], as all classes are weighted equally.

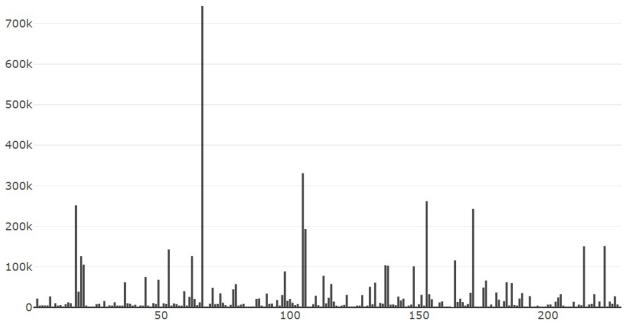

Fig. 1: Training set label distribution, with instances per class

The first $9,897$ images of the test set are identical to the validation set, thus the test set is more similar to the validation set rather than the training set. This might have been an accident on Kaggle's part.

2.2 Evaluation

The metric used by Kaggle to evaluate the predictions on the test set is micro-averaged mean F_1 score [16]. F_1 score is the harmonic mean of precision and recall, and is given by:

$$F_1 = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (1)$$

where precision and recall are defined as:

$$\begin{aligned} \text{precision} &= \frac{TP}{TP+FP} \\ \text{recall} &= \frac{TP}{TP+FN} \end{aligned} \tag{2}$$

where TP, TN, FP, and FN are the number of true positives, true negatives, false positives, and false negatives respectively. Precision and recall are also referred to as positive predictive value and true positive rate (or sensitivity) respectively.

The micro-averaged mean that is used for the challenge is the sum of F_1 score for each image, divided by the number of images. This differs from the macro-averaged mean, which is the average of F_1 scores per label, instead of per image.

3 Related Work

Related research on image classification can be seen as older techniques and newer techniques that use neural networks. Older techniques that use Binary Relevance (BR) in combination with SVMs [14, 22] or perceptrons [15]. In the more recent years, as more computing power became available, a lot of progress has been made in the field of image classification with the rise of neural networks. Convolutional neural networks (CNNs) have reduced the classification error rates on various datasets compared to earlier techniques as early as 2012 [13]. Nowadays, many different CNN architectures are available, and research has shown that CNNs give good results for multi-label classification as well [21].

3.1 Transfer Learning

Transfer Learning is a method in machine learning where knowledge of a specific domain is used for a second task. A well performing model can be used on a different (but related) dataset. An often-used example of this is training a state-of-the-art model on the ImageNet dataset, saving the weights, then fine-tuning it on something else.

Since 2010, ImageNet hosts the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where researchers submit their classification algorithms for a large 1000-class dataset. These models are publicly available, along with their pre-trained weights. Deep learning library *Keras* [4] has implementations of these models, which makes it easy to directly import and use them. Since ILSVRC gets state of the art algorithm submissions for image classification, models that perform best on the ImageNet dataset are initially explored. Table 1 shows the performances of models over given dataset for this competition. First, we experimented with the plain pre-trained models to find which pre-trained network performs best for this challenge. After an initial test, further improvements were made to the best performing network. This approach is described in Section 4.

In this section, we further elaborate on the pre-trained models that were explored for the challenge.

Table 1: ImageNet results for pre-trained models

Model	Top-1	Top-5	Parameters	Depth
VGG-16	0.715	0.901	138,358K	23
InceptionV3	0.788	0.944	23,852K	159
InceptionResNetV2	**0.804**	**0.953**	55,874K	572
Xception	0.790	0.945	22,910K	126

3.1.1 VGG-16

VGG-16 [17] is a simpler network compared to the more recent networks for image classification. VGG-16 uses a small convolution filter of size 3×3 with a network depth of 16 layers. VGG-16 implements convolution layers and max-pooling.

3.1.2 Inception

The Inception [19] micro-architecture (or module) was introduced with the goal to scale up neural networks without increasing computational costs. An inception module computes multiple transformations over the same input in parallel, and concatenating their results into a single output. The computation consists of a 1×1 convolution layer, a 3×3 layer, a 5×5 one, and a max-pool.

3.1.3 InceptionResNet

The ResNet [9] architecture allows the network layers to access inputs to lower layer using residuals. The authors argue that residual connections are important for training deep networks. The InceptionResNet [18] model applies the concept of residual connections to the Inception architecture.

3.1.4 Xception

Xception [5] aims to reconstruct the way convolution networks are presented. Unlike Inception, which partitions input data into several compressed chunks, Xception maps the spacial correlations for each output channel separately. It then performs a 1×1 depth-wise convolution to capture cross-channel correlations.

3.2 Label Distributions

The the distributions of the training set and validation set are different. between On average, an image in the training set has 5.84 labels associated with it, with all labels occurring at least once. While the label distribution of the training and validation sets are very similar, the average amount of labels per image in the validation set is higher, at 8.03 labels per image, and covers only 225 out of 228 labels. The frequency distribution of labels in the training and validation sets are shown in Fig. 2 and Fig. 3 respectively.

3.3 Label Imbalance

As seen in Fig. 1, the distribution of labels in the dataset is heavily imbalanced. Some labels occur very often, with label-66 occurring on almost every 3 out of 4 images, whereas some labels have only a handful of

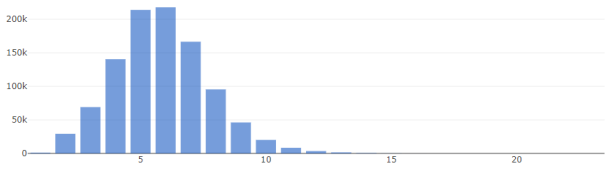

Fig. 2: Frequency distribution of labels in the training set

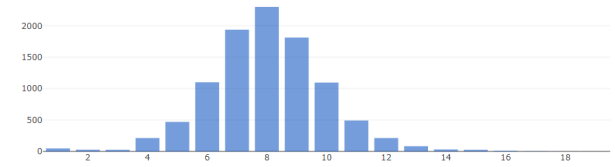

Fig. 3: Frequency distribution of labels in the validation set

images associated with them. The ten most occurring labels are shown in Fig. 4.

A characteristic of mean F_1 score is that all labels are weighted equally in calculating the performance. The imbalance of the distribution of labels might make it difficult to train the model and thus lead to a bad mean F_1 score. Solving the class imbalance issue will lead to a better mean F_1 score.

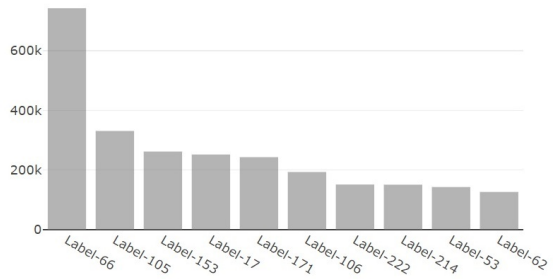

Fig. 4: Ten most occurring labels in the training set

3.3.1 Dealing with labels imbalance by Entropy for Multi-Label classification (DEML)

DEML [7] deals with a label imbalance issue. An important measure used to quantify the imbalance in the article that introduces this method, is the *LabelEntropy* of the dataset. This measure is calculated with the normalized entropy, which is calculated from the entropy of the labels. The entropy of a set of labels is given by:

$$E(labelset) = -\sum p_i log_2(p_i) \tag{3}$$

Where p_i denotes the probability of an element of the dataset being labeled with the i^{th} class in the *labelset*. The normalized entropy can then be calculated as:

$$NE(labelset) = \frac{E(labelset)}{log_2(c)} \tag{4}$$

Where c is the number of classes appearing in the *labelset*. Finally, the label entropy measure can be determined with:

$$LEnt(D) = \frac{1}{q}\sum_{i=1}^{q} NE(y_i) \tag{5}$$

Where D denotes the dataset, q denotes the number of labels, and y_i denotes the labels of the i^{th} element of the dataset.

DEML works well on datasets with higher label entropies and becomes computationally expensive in case of a low label entropy and will perform worse. The iMaterialist Fashion dataset has a low label entropy of 0.011, as calculated using Equations 3-5, therefore *DEML* is not suitable for this challenge.

3.3.2 Random k-Labelsets (RAkEL)

Another method to deal with label imbalance is *Random k-Labelsets* (*RAkEL*) [20], which performs reasonably well on low-entropy datasets, and is computationally less expensive than *DEML*. However, *RAKEL* will not perform significantly better than the baseline *binary relevance* (*BR*) method on datasets that are similar to the Kaggle dataset. The *BR* method suggests to train an independent classifier for each label. *RAkEL* takes much longer to run than *BR* for low-entropy datasets [7], and the dataset is very large. On top of that, labels are suspected to be correlated (such as 'dress' and 'woman'), and thus implement some kind of dependency, with which the *BR* method can not deal.

As both *DEML* and *RAkEL* do not seem fit for the iMaterialist Fashion dataset, we designed a new approach that is able to deal with label imbalance on the fly. This new approach will be discussed in Section 4.3.1.

4 Approach

In this section different experiments for finding the best model for the challenge are described.

4.1 Choosing a pre-trained model

First, the performance of various pre-trained models on the dataset are compared, as seen in Section 3. For the initial performance comparison two fully-connected layers of 1024 neurons with *ReLU* activation were appended to the pre-trained model. Also, a classification layer with 228 neurons (one for each label) was appended after the fully-connected layers. This final layer uses sigmoid activation and was trained with binary cross-entropy as the loss function [21]. Each of these models were trained over only 25% of the training data because of time and computation constraints. In the training, Adam optimizer [12] was used with learning rate set to 0.01, with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Training was done using the evaluation metric as described in Section 2.2. The mean F_1 score shown in Table 2

is obtained by submitting the results to Kaggle after thresholding the prediction outputs of the network at 0.3. The threshold is set in such a way that all images have at least one label associated to them.

The performance of various pre-trained models on the dataset are shown in Table 2.

Table 2: Deep learning model comparison, where x-Frozen indicates how many layers are frozen during training.

Model	x-Frozen	Mean F1 score
VGG16	4	0.41930
Inception Resnet V2	400	0.47681
Inception V3	249	0.49091
Xception	126	0.49224
Xception	50	**0.51219**

With pre-trained models, freezing some of the layers of the network [1] (i.e. weights of these layers are not updated during training) is a common practice that reduces training time. Normally, the first few layers of the network are frozen, since they are expected to learn very small structures of the images. The unfrozen layers can then learn to recognize the bigger structures in the images. In Table 2, the x-Frozen column indicates that the first x layers of the network are frozen in the experiment. Given that there is no definitive way of deciding the number of layers that are to be frozen during training, numbers based on the network depth and the training dataset size as suggested in articles [10], [11], [6] were used.

As shown in Table 2, Xception clearly outperforms all other models on the dataset. The deeper network architecture allows the model to extract more features than a shallower one, which is presumably better for this challenge. Since Xception performed best, it was used for all further experiments.

4.2 Thresholding

The model's prediction for an image comes in the form of an array of 228 values ranged between 0 and 1. During training, the predictions are thresholded at 0.5 to compute the training and validation F_1 scores. After training our first model, non-thresholded predictions for labels that occur more often were higher on average, even if the predictions were wrong. Therefore, a method which optimizes on F_1 score is used that finds optimal threshold for each label.

For this, the algorithm shown in Algorithm 1 is used. For optimization, the predictions of the network on the validation set was used. For each label, the most optimal threshold value between 0 and 1 is found such that it maximizes the F_1 score, iterating over all the labels multiple times until our F_1 score converges.

Then, the optimal thresholds is applied to the model's predictions on the test set before submitting to Kaggle. This technique really helped in improving our score on the Kaggle leaderboard. For the first fully trained network, the optimal thresholds improved the results from 0.538 to 0.574.

Algorithm 1 Linear threshold search

Input: $true, pred$ ▷ Truth, predictions
Output: $thresh$ ▷ Thresholds
procedure FindThresholds($true, pred$)
    $thresh \leftarrow 0.5$ ▷ For each label
    $score \leftarrow F_1(true, pred, thresh)$
    for *epochs* **do** ▷ Until $score$ converges
        $epthresh \leftarrow thresh$
        for $l \in labels$ **do**
            $propvals \leftarrow \{0.1, 0.2, \ldots, 1\}$
            for $val \in propvals$ **do**
                $epthresh[l] \leftarrow val$
                $new \leftarrow F_1(true, pred, epthresh)$
                if $new > score$ **then**
                    $thresh[l] \leftarrow val$
                    $score \leftarrow new$
    return $thresh$

4.3 Pre-Processing

Other than the label imbalance in the dataset, one another major problem is that the images are of different sizes and aspect ratios. The image sizes vary from 32×32 to 600×600 and in total there are 1768 different sizes in the training set. In this section we describe the pre-processing steps we took to deal with these issues.

4.3.1 Batch Balancing

In order to deal with label imbalance, a batch generator that generates mini-batches from an internal macro-batch is used. The macro-batch contains at least one image for each label type, and is filled with random images until it reaches a specified amount. Mini-batches are taken from the macro-batch until it is exhausted, at which point a new macro-batch is generated. This method was simple to implement, does not slow down training by much, and improved results slightly, as shown in Table 3.

Table 3: Macro-batch comparison: macro-batch size of 1024 contains at least one image for each label. Layers 2x Dense 1024 refers to two fully connected layers of 1024 neurons before the classification layer.

macro-batch	**Layers**	**Mean F1 score**
None	2x Dense 1024	0.49224
None	1x Dense 1024	0.51219
1024	2x Dense 1024	0.49646
1024	1x Dense 1024	**0.52924**
512	1x Dense 1024	0.52419

As seen in Table 3, having only one fully connected layer outperforms having two, and macro-batch sizes 1024 and 512 have similar results, and are better than not having macro-batches. The results are gathered from training the model for 5 epochs.

4.3.2 Image Manipulation

The images in the dataset have many different dimensions and aspect ratios, while the pre-trained model expects square images of size 299×299.

Stretching vs. cropping and padding: Rescaling every image to the same size stretches and distorts the objects in the images, which would make it difficult for the network to learn structures in the image. To avoid this, every image was sized down until the largest axis of the image fits within the square size that the model expects. Then, white borders were added around it. This is done during batch generation in order to avoid duplicating the entire dataset on disk. Both padding and stretching do not show much difference in the results. The stretching approach actually seemed to work out better than cropping and padding, as shown in Table 4.

Augmentation: A common pre-processing technique is to augment the existing dataset with perturbed versions of existing images, as used by Goodfellow et al. [8]. This is done to expose the neural network to possible image variations, which makes it less likely to overfit on a small dataset. The dataset is large, therefore it could be argued that it does not need any augmentation. But as shown, some the labels do not have many images associated with them, and with the macro-batch approach, the network sees these specific images many times. This could possibly lead to overfitting for these labels, therefore a horizontal flip augmentation with 50% chance was used.

Table 4: Stretching vs. Padding with different macro-batch sizes, with label-wise thresholding

macro-batch size	Stretching	Padding
1024	**0.59799**	0.58992
512	0.58631	**0.58767**

4.3.3 Multiple networks

In another approach, the labels were divided into two separate clusters in order to solve the imbalance issue. Different models were trained on these separate clusters. The first network was trained on labels having less than 7000 images associated with them, and the second network on rest of the labels. Our intuition for dividing the labels based on the number of images was that it would help in balancing the dataset, as each network will see comparable numbers of images for each label. We came up with the specific number of 7000 by examining the performance of our baseline model on each label. For these labels, our baseline model was giving much worse results than other labels.

However, this multiple network approach resulted in a score of 0.58403 on the public Kaggle leaderboard, thus was not able to improve results.

5 Results

Most of the results have already been shown to discuss the performance improvements of some of the techniques that were used. All these experiments lead to believe that the pre-trained model Xception with the first 50 layers frozen and one fully connected layer of 1024 neurons is the best model for this challenge. Training this model with a batch generator that somewhat balances the batches, and applying label-wise thresholding before submitting gives the best results. The best single-model results with this model has already been discussed in Table 4. By ensembling these models, more robust predictions were acquired, and lead to an increase in the score to ≈ 0.62 on the public leaderboard.

A total of 42 submissions were made during the competition. All results mentioned so far were evaluated on the public Kaggle leaderboard, which was calculated only on 30% of the test data. On May 30^{th} the competition ended, and the final score was calculated on the remaining 70% of the data, which was then displayed on the private leaderboard.

Table 5: Best results on the private and public Kaggle leaderboards

Model	Public	Private
Stretch-1024	0.59799	0.59427
Stretch-512	0.58631	0.58360
Pad-1024	0.58992	0.58647
Pad-512	0.58767	0.58130
Ensemble-Stretches	0.60725	0.60343
Ensemble-All	**0.62095**	**0.61531**

Results of different approaches on both public and private leaderboards are shown in Table 5. The first four models in the table are the same as those referred to in Table 4. The last two models were ensembled by simply adding up their predictions and dividing by the number of models in the ensemble. *Ensemble-Stretches* is the ensemble of *Stretch-1024* and *Stretch-512* in the table, and *Ensemble-All* is the ensemble of all four models in the table, with sharper label-wise thresholding. For the Ensemble-All model, a more precise thresholding was done by decreasing the step size to 0.05 of the threshold search algorithm.

The final submission on Kaggle reached position number 16 on the public leaderboard, and position #18 on the private leaderboard, with 42 total submissions, out of 212 total participants.

6 Discussion

Our final result on the public leaderboard of 0.62095 is not much higher than our first submission of 0.53833, but our experiments have certainly improved results by a significant amount.

The dataset is so imbalanced that many techniques from the literature are not applicable. Label-wise

thresholding improved results much more than balancing the mini-batches did. This is expected, because adjusting the threshold can make or break a prediction, while changing the mini-batches only changes training slightly.

We expected that cropping and padding the images to fit the model would give better results than stretching them, however, stretching seemed to be slightly better. This is presumably because most images had the right aspect ratio to start with, and stretching the smaller images actually helped with the prediction, whereas cropping and padding the smaller ones resulted in wide white image borders with little content in the center.

Using ensembling methods, which were also employed in the best performing method of the competition, we were able to increase the score even more. These work because different models often learn to recognize different things, and they complement each other when ensembled.

The number 1 ranking submission on the Kaggle leaderboards was by user *"radek"*, who submitted predictions 64 times, and ended with a score of 0.72815 on the public leaderboard, and 0.72483 on the private leaderboard. He revealed his approach one day after the challenge ended, and we were pleased to see that his approach was very similar to ours. His best results were also with pre-trained CNNs, with ResNet performing the best. To optimize thresholds, he used XGBoost [3] instead of doing a linear search as we did. Using either of these would probably given a slight improvement to the results. However, the most significant thing he did to get a higher score was to not train on the training set, but on the validation set by using his own cross-validation, and ensembling the models trained on each fold. The validation set is quite small, consisting of only 10k images, which needed heavy augmentation in order to reduce overfitting.

During our data analysis in Section 2.1 we discovered that the average amount of labels of the training set images was lower when compared to those in the validation set, and that Kaggle leaked information about the test set by accidentally containing all of the validation images in the test set. The images in the validation and test sets are much more alike, and are presumably labelled much better than the images in the training set, which helps training significantly.

7 Conclusion

The fact that this challenge is mainly a computer vision task, and judging from our own results and the top results on the leaderboard, pre-trained CNNs are the obvious choice. Despite the fact that these models work well even at baseline, it seemed that the biggest task of this challenge was dealing with the label imbalance of the dataset, not applying the model. We have spent time looking at the literature to deal with such imbalance, but did not find promising methods to apply in this challenge. In the end, the biggest change we could have made to improve predictions was to train on the validation set. We are disappointed that we missed this opportunity, and it seems that at least the top 5 submissions used it, as the 2nd place submission revealed they did the same, and there is a steep drop-off in the scores after the top 5. Still, we are pleased to have reached the top 10% of the leaderboard once again.

8 Acknowledgements

This research was supported by Radboud University Nijmegen. We would like to thank Zaheer Babar, MSc, for his supervision during this course, and his feedback on the report.

We would also like to show our gratitude to Prof. Dr. Elena Marchiori for organizing this course, and to Wish and Kaggle for providing us with the dataset and the challenge.

Individual contributions

We split our work into different tasks. Our individual contributions to those tasks are as follows:

Alex: Image Manipulation

Brigel: Image Manipulation

Laurens: Label-wise thresholding, Data Exploration, Label Imbalance, Batch Balancing

Ankur: Label-wise thresholding, Multi Network approach, Image Manipulation

Luca: Pre-trained models

Rick: Image Manipulation, experimenting on implementing k-max pooling

Tristan: Google Cloud Platform, Pre-trained models

Links

The project sourcecode is available at Github:
`https://github.com/alexthe2nd/machinelearninginpractice`

More information about the Kaggle competition can be found here:
`https://www.kaggle.com/c/imaterialist-challenge-fashion-2018`

References

[1] Andrew Brock, Theodore Lim, JM Ritchie, and Nick Weston. Freezeout: Accelerate training by progressively freezing layers. *arXiv preprint arXiv:1706.04983*, 2017.

[2] Francisco Charte, Antonio J Rivera, María J del Jesus, and Francisco Herrera. Addressing imbalance in multilabel classification: Measures and random resampling algorithms. *Neurocomputing*, 163:3–16, 2015.

[3] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[4] François Chollet et al. Keras. `https://keras.io`, 2015.

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

[6] Greg Chu. How to use transfer learning and fine-tuning in keras and tensorflow to build an image recognition..., May 2017.

[7] Ming Fang, Yuqi Xiao, Chong-Jun Wang, and Junyuan Xie. Multi-label classification: Dealing with imbalance by combining labels. *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, pages 233–237, 2014.

[8] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] Prakash Jay. Transfer learning using keras – prakash jay – medium, Apr 2017.

[11] Andrej Karpathy. Transfer learning.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[14] Xuchun Li, Lei Wang, and Eric Sung. Multilabel svm active learning for image classification. In *Image Processing, 2004. ICIP'04. 2004 International Conference on*, volume 4, pages 2207–2210. IEEE, 2004.

[15] Gulisong Nasierding, Grigorios Tsoumakas, and Abbas Z Kouzani. Clustering based multi-label classification for image annotation and retrieval. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pages 4514–4519. IEEE, 2009.

[16] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979.

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[18] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[19] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. Going deeper with convolutions. 2015.

[20] G. Tsoumakas, I. Katakis, and I. Vlahavas. Random k-labelsets for multilabel classification. *IEEE Transactions on Knowledge and Data Engineering*, 23(7):1079–1089, July 2011.

[21] Nikita Uvarov. Multi-label classification of a real-world image dataset. Master's thesis, NTNU, Norway, 2017.

[22] Zheng-Jun Zha, Xian-Sheng Hua, Tao Mei, Jingdong Wang, Guo-Jun Qi, and Zengfu Wang. Joint multi-label multi-instance learning for image classification. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.