iMaterialist Fashion - Report

Mick van Hulst Dennis Verheijden Roel van der burg Brian Westerweel
Joost Besseling

June 20, 2018

1 Problem statement

This report describes the results for the second competition of the course Machine Learning in Practice. We as team MLoB, consisting of Joost Besseling, Roel van der Burg, Mick van Hulst, Dennis verheijden and Brian Westerweel, have chosen the imaterialist fashion challenge [1] as the second project for the course Machine Learning in Practice. This challenge consisted of a data set of approximately 1.1 million images which had to be assigned to one or multiple labels. Totalling up to 228 labels that could be assigned.

2 Data set

The data set that the challenge provides consists of a training, test and validation set, which in turn consists of:

- 1.014.544 training images
- 228 labels in the training dataset
- 39.706 test images
- 9.897 validation images
- 225 labels in validation set

We're interested in the distributions of the data such that we can prepare our models accordingly. Figure 1 and 2 visualise the distribution of the labels for the training and validation dataset respectively. We observe that the labels aren't evenly distributed, but the distribution between the training and validation dataset seem comparable.

To get a feeling for the dataset we also visualise some images and observe that for some labels there are no clear characteristics that define the corresponding label (see Figure 3). From this observation we conclude that it would be hard for a human to differentiate between the labels without knowing what the labels mean. We believe that there's some structure to the labels, meaning that there might be some hierarchical structure that explains

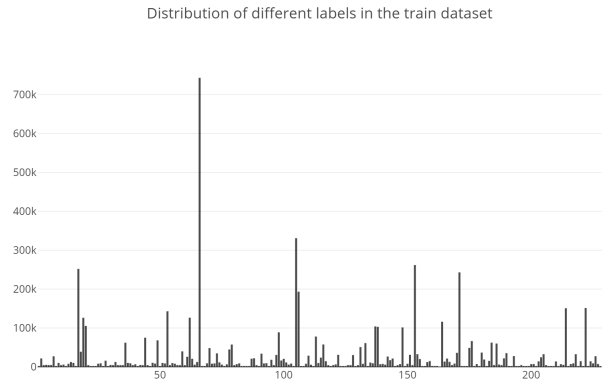


Figure 1: Data distribution training set

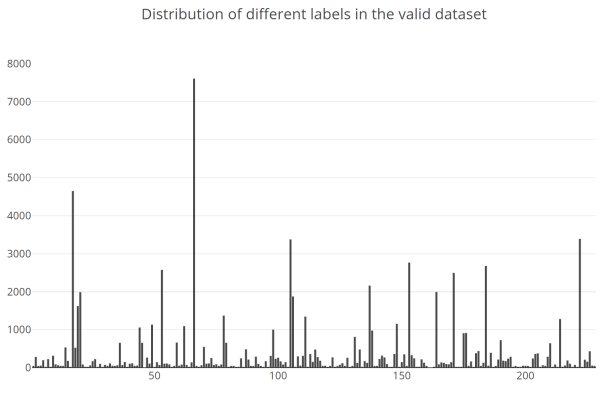


Figure 2: Data distribution validation set

[1] https://www.kaggle.com/c/imaterialist-challenge-fashion-2018

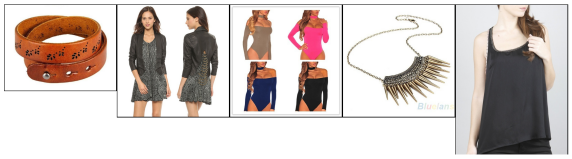


Figure 3: Examples of image-label 24

the different combinations of the labels, however, this is not something that the challenge provides so we cannot use this to e.g. define class-weights when classifying.

3 Data handling

The data for this challenge had to be pre-processed as the images might be corrupt, due to the link to the image being dead, different aspect ratios or different sizes. To counter this, we had to resize the images and use white-padding to maintain aspect ratios (figure 4b). We used white-padding as the majority of the images had a white background. Corrupt images within the train and validation set are not used as they can only negatively impact training.

It is computationally infeasible to load all the images in the network at once, so we developed a batch generator. The batch generator allows us to load a static number of images at a time. To counter the class imbalance we tried probability sampling, however, this didn't render any desirable results as this is a multi-label problem and probability sampling resulted in an identical distribution (i.e. this was most likely due to the top 10 classes occurring for almost every class). We also experimented with custom loss functions, where correctly predicting less common classes had a bigger impact than correctly prediction common classes. If we look at the results in table 1, we can see that the effect is marginal. This might be due to the fact that correctly predicting less common classes has a lower impact than focusing on the common classes.

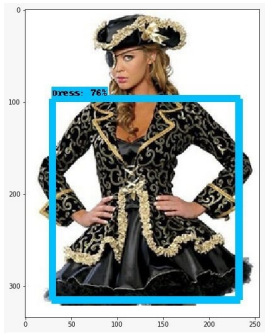
One thing that stands out is the large size of the training data set. During training we noticed that generating batches took a long time as we had to continuously download and store the images temporarily. To counter this we preprocessed the images as much as possible and uploading the entire data set to GCP (Google Cloud Platform) with a 85% quality. This decreased the file size significantly, thus decreased the total memory footprint of one batch while also decreasing the time needed to generate one batch.

There were two possible input sizes that may be used for images, 224x224 and 299x299. Initially we chose to work with the smallest size, to increase the

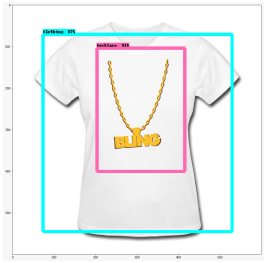
(a) Padded example image 2

(b) Padded example image 2

Figure 4: Examples of Image Padding



(a) 'dress'



(b) 'necklace' 'clothing'

Figure 5: Examples of Object Detection

speed in which images could be processed by the models. However, this came with the cost of image quality. When the pipeline and routines were finished, an experiment was done with the large images. This experiment proved that using the larger images would produce better results than using the smaller images (see table 1).

Besides the huge data set, one can also observe that there are a large number of labels. To counter this, we decided to ignore labels that occurred less than 500 times as we wanted to avoid training the classifiers on labels that did notn occur frequently.

4 Results

In contrast to the last project, we decided to use pre-trained models for this challenge, instead of fully training the models ourselves. The models used in this project are pre-trained using ImageNet [1]. ImageNet is a collection of over 100.000 concepts with an average of 1000 images per concept. ImageNet is often used as a reference data set for establishing convolutional neural network benchmarks. For this project MobileNet, InceptionV3, Xception and ResNet50 are chosen because these model are considered state-of-the-art when it comes to individual performance for image classification problems. Therefore, we think that combining the

Method	F1	#params (Million)
MobileNet	0.21	3.46M
MobileNet_customLoss	0.20	3.46M
InceptionV3_small	0.35	22.3M
Xception_small	0.38	21.3M
ResNet_small	0.38	26M
Xception_large	0.41	21.3M
Xception_large_precompute	0.504	23.7M
Ensemble - Mean	0.47	227.1M
Ensemble - Harmonic Mean	0.41	227.1M
Ensemble - PSO	0.506	227.1M

Table 1: Summary of the achieved results, per model and with their corresponding F1 on Kaggle. The *_small* suffix denotes that the network is trained on 224x224 images and *_large* on 299x299 images.

models with ensemble learning methods will yield even better results. Besides the well-known models we created a model based on object detection. By using the pre-trained weights for the different models the models are already aware of the high-level abstraction for images. These abstractions have proven to work well on image classification tasks [2]. For training these models, we have chosen to remove the last prediction layers from the original model and add our own prediction layers. When performing back-propagation, we froze the original pre-trained layers and only trained the prediction layers. After training converged, we unfroze the last N layers of the pre-trained model (depending on the model) to adapt the pretrained-model to our domain using a low learning rate.

Binary cross entropy was used as the loss for all models, because the models have to output a probability for each label. To measure performance of the models, a micro-averaged F1-score is used, because this is also the performance metric used by Kaggle. Hyperparameters were optimized by creating callbacks, these monitor the training process of the models and change the process accordingly. One of the callbacks adapt the learning rate of the optimizer when the validation loss did not decrease over a set number of epochs. Another callback was: early stopping, which stopped training if the validation loss did not increase after a set number of epochs.

4.1 Object detection

As we had the suspicion that there was probably some relation between the labels, we thought it would be interesting to try an object detection algorithm. We tried predicting objects for each image using a pretrained ResNet50 [3]. This network was trained on the open image data set [4]. For each image we wanted to use the discovered objects as additional features for making predictions. These features could be integrated into the model by performing Late Fusion, which proved to be helpful for classification [5].

The object detection network detected reasonable objects in some cases, but in the majority of the cases it detected objects such as 'clothing' which are meaningless within the iMaterialist challenge (see Figure 5a and 5b). Furthermore, the model took around two minutes to detect objects in an image, meaning that we'd need around two million minutes to apply the model to the entire training set. Due to these reasons, we decided to not pursue this method any further.

4.2 ResNet50

The ResNet we used is based on Keras' version [3], where the last layers were unfrozen and fine-tuned to our target data set. Resnet-50 achieved the first place during the ILSVRC 2015 classification competition on imagenet. It solves a degregation problem of deeper networks: when network depth increases, accuracy could get saturated. The degradation problem suggest that solvers might have difficulties in approximating identity mappings by multiple nonlinear layers [6]. By dividing the network in residual blocks, it is easier to map these identity mappings by multiple nonlinear layers.

4.3 MobileNet

MobileNet is a convolutional network by (Howard et al, 2017) [7]. This network is designed to be used on mobile devices, which makes it an ideal candidate to test out certain parts of our pipeline. During the project it was used to experiment with certain hyperparameters and for testing if the batch generator was working correctly. In this case only the last 5 layers were unfrozen for training. MobileNet presents an efficient and powerful deep-learning model by focusing on depth-wise separable convolutions.

4.4 InceptionV3

InceptionV3 [8] is a convolutional neural network developed by Google. Inception differs from other convolutional neural networks in how its lower-level layers are built.

Usually during the creation of a network, a user has to define the types of layers and the corresponding structure (i.e 5x5, 3x3 or Max-Pooling). Inception refuses to choose between these types and decides to use all of them within a single layer. During training time these different types are combined into a single activation map. This architecture makes the network very time expensive to train, because it is basically brute-forcing the different options that are available.

The InceptionV3 network was trained three times. The first network trained was an Inception v3 network which we trained on the entire data set. The second and third network were used to counter the highly imbalanced data set (see Figure 1 and 2 for distribution classes). The second classifier is only trained on the labels that occur at least 100.000 times. The third classifier is trained on labels that occur less than 100.000. These two classifiers were combined by adding their predictions.

4.5 Xception

The XCeption network is a network proposed by Chollet in 2016 [9] building on the already existing InceptionV3 networks. It changed the inception modules to use depthwise separable convolutions This network was trained once and trained on the entire data set.

4.6 Precomputation

Training on the whole dataset proved to be a time-consuming process. One training epoch over the whole dataset, using an SSD for fast read-write and CUDA (GPU-acceleration), took two hours and twenty minutes. To speed up this process, we could use precomputed features of the images, instead of the actual images. This eliminates the Xception model for training, losing roughly 21 million parameters per forward sweep. Using these features, we drastically reduced the training time for models. Using a classifier of roughly 3 million parameters, one training epoch over the whole dataset takes only twenty minutes. However, using this method, we lose the possibility of data augmentation and finetuning some Xception blocks.

The final model that was used in the ensemble effectively saw at least ten times as much data as any other model in the ensemble. Judging from table 1, we can observe that this trade-off was worth it, as it achieved the highest F1 score of the individual models.

4.7 Ensembling

After our previous project, one of our takeaways was to apply ensembling during this project. Ensembling is used to combine several models and thus combine their respective strengths. For our ensembling process we combined the aforementioned InceptionV3 and Xception networks. We ensembled the networks after training the models and were ensembled using the following methods:

- **Arithmetic mean:** Takes the mean of all classifier's results.
- **Harmonic mean:** The Harmonic mean is calculated by dividing the number of observations by the reciprocal of each number in the series. This means it mitigates the effect of large outliers in a dataset when compared to the arithmetic mean.
- **Particle Swarm Optimization:** Using PSO (Particle Swarm Optimization) we were able to find weights which maximized the F1-score given the validation set. Naturally this means we assume that the weights which maximised the validation results would also maximize the results of the test data set.

4.8 Thresholding

To submit to Kaggle we had to threshold our results. After using a standard threshold of 0.5, we also decided to try custom thresholding for each label. We tried finding thresholds for each label by trying several thresholds and measuring which threshold would maximize the F1-score.

The results after thresholding are portrayed in Table 1. We can observe that the results of the ensemble using the PSO algorithm are the most convincing. If we look at the weights that are generated by this strategy, we can see that the combined classifier is relatively important. This validates our idea that we could counteract the imbalance by training two separate classifiers. The difference between our best single classifier and ensemble is minimal, this tell us that the remainder of our single classifiers weren't strong enough. This corresponds with what is stated in section 4.6, as this is the only classifier that is trained on the entire dataset for several iterations.

5 Conclusion

5.1 Google Cloud platform

Many of the implemented models were used on the GCP for final training. Including the InceptionV3 and Xception networks, during the final weeks of the project. Together with the entire dataset that was already uploaded

on the GCP-framework we were able to adjust our batch generator to directly load the images from GCP instead of the downloading we had been doing until then. Unfortunately GCP prove to be very slow, as it took about 40 hours to go over the entirety of the training dataset. After talking to other project groups, who experienced the same, we believe that something was configured incorrectly. Other groups who used Compute Engine with an SSD, instead of CloudML, did not experience such problems. In hindsight, we believe that Compute Engine would've enabled us to increase our training time. We classify this as a take-away for future projects.

5.2 Simplicity vs. Over-Engineering

During the start of the second half of the project we enthusiastically started with discussing several convolution networks, an extensive training pipeline, (weighted) probability sampling, per-class thresholding, potential object recognition for extra features and ensembling methods. Many of these methods did prove to be beneficial. For example, per-class thresholding and ensembling did (significantly) improve our results, especially when used with models all having had the same relative training time. However, object recognition seemed to be an avenue worth exploring, but we concluded this didn't prove to be useful (as elaborated in section 4.1).

5.3 Stick with the Basics... initially

There's a trade-off between wanting to learn complex things and achieving desired results. The extra merit from fine tuning, or using different models usually pales in comparison with the basic model that is simply trained longer and has seen more of the data. The best models in our ensembling method, for example, did not need probability sampling. This does not mean complex methods are always inferior to simple ones. Though it did learn us to first stick with a strong simple baseline model, which can be used as a checkpoint.

6 Who did what

During the project each of the team members were assigned specific tasks. Do take note that our approach can be considered agile as we switched roles constantly and helped each other out where we could. For the second competition we planned weekly meetings on Wednesday to work together. This significantly decreased overhead caused by communications.

- **Mick van Hulst:** During the first competition Mick focused on the preprocessing of the data and the integration of GCP. This included setting up the corresponding code files and report for the rest of the MLiP class which allowed them to submit jobs to GCP. During the second competition Mick focused on developing the batch generator (e.g. probability sampling and class weights), EDA, integration of GCP code in ML pipeline, ensembling of models and finding class-specific thresholds. Furthermore, Mick worked on both the competitions' final reports.
- **Dennis Verheijden:** For both projects Dennis was involved in implementing networks, optimizing training routines and writing code for making submissions. The major focus of the first project was on doing feature extraction on the textual data and integrating these features such that they could be used as an input for the model. For this project, the networks that Dennis implemented were MobileNet, InceptionV3 and Xception. Another big focus was to optimize various aspects of the code, for example building an image downloader to resize and pad images using MultiProcessing. The creation of the production routines and pipeline, including maintaining them, were also produced by Dennis. As side-projects, Dennis worked on precomputation and the custom loss function. And finally, Dennis trained the aforementioned models and worked on the report.
- **Roel van der Burg:** As a team leader, Roel was in charge of planning weekly meetings and communication within the team and with the supervisors. For both the projects Roel was involved with the pre-processing of the data and the corresponding data analysis, furthermore he was involved with the training and development of the Bi-directional LSTM. For the second competition Roel developed the framework of the batch generator and parts of the later development. Moreover Roel implemented, trained and finetuned the resnet-50, creating a per-label thresholding function and working on the report.
- **Brian Westerweel:** During the first competition Brian mainly focused on developing a LSTM network and a one dimensional CNN. For the second competition Brian implemented a custom F1-measure for Keras so the models could work with them. He also worked on a object detection network to use the found objects in all images as new inputs. Besides that Brian trained networks and worked on the report.
- **Joost Besseling:** In the first competition Joost focused on developing the bi-directional LSTM and worked on the report. For the second competition he worked on the object detection network to create alternative inputs from the images and the ensemble algorithm. Besides that he also worked on the final report.

References

[1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[2] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 512–519. IEEE, 2014.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[4] Ivan Krasin, Tom Duerig, Neil Alldrin, Vittorio Ferrari, Sami Abu-El-Haija, Alina Kuznetsova, Hassan Rom, Jasper Uijlings, Stefan Popov, Shahab Kamali, Matteo Malloci, Jordi Pont-Tuset, Andreas Veit, Serge Belongie, Victor Gomes, Abhinav Gupta, Chen Sun, Gal Chechik, David Cai, Zheyun Feng, Dhyanesh Narayanan, and Kevin Murphy. Openimages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://storage.googleapis.com/openimages/web/index.html*, 2017.

[5] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[6] 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.

[7] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[8] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

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