fractal

INTELLIGENCE FOR IMAGINATION

AI’s apparel eye

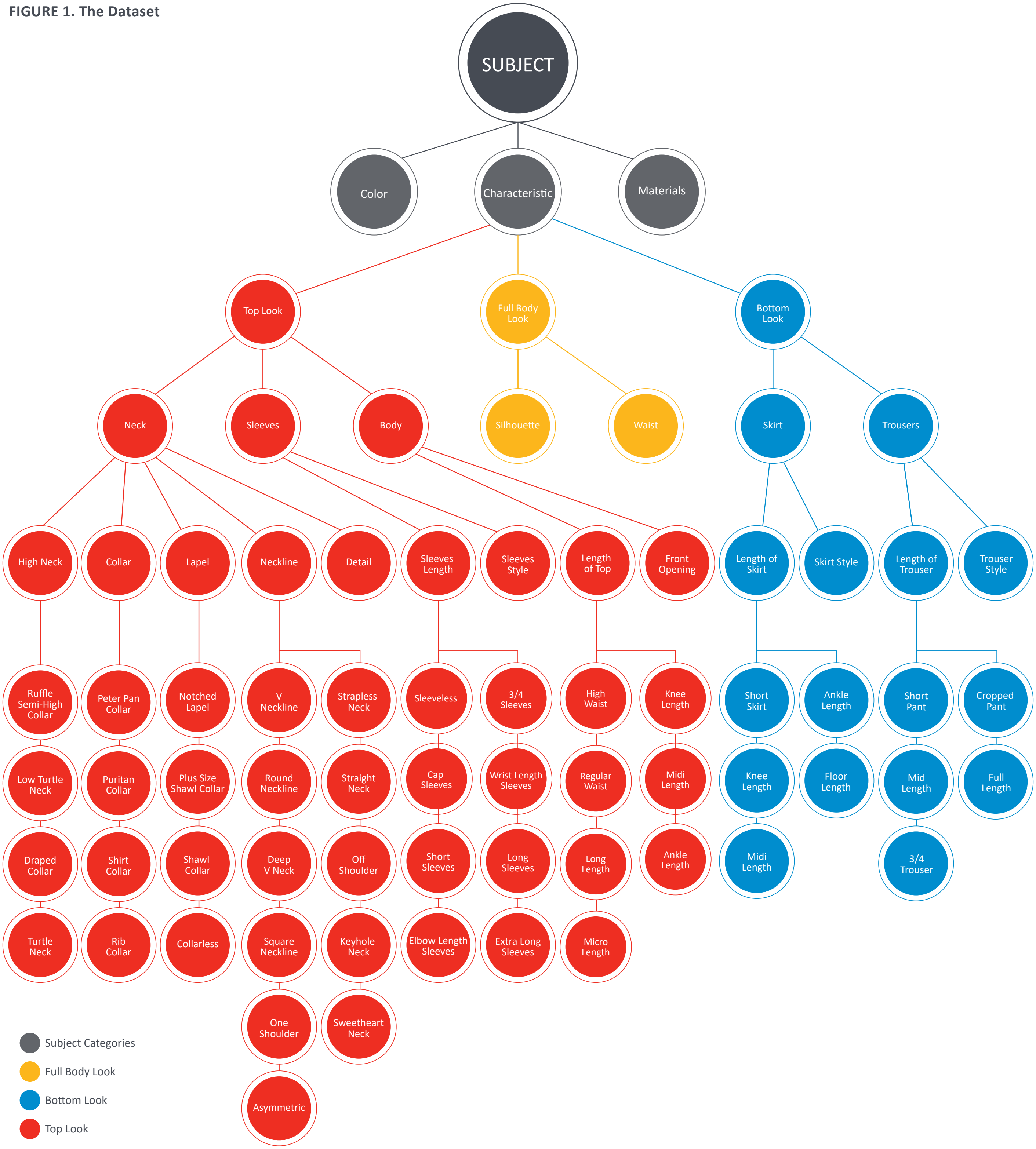
ABSTRACT

The “FashionAI Global Challenge 2018—Attributes Recognition of Apparel” is conducted to push the ability of AI to help the fashion industry in recognizing the attributes of clothing from a given image. This capability could be widely applied in applications such as apparel image searching, navigating tagging, mix-and-match recommendations, etc. The competition was hosted at the Alibaba cloud competitions site: Tianchi. The dataset released for the competition is the largest dataset available in the domain of attributes recognition of the apparel. We finished the competition at 30th position out of 2,950 contestants across the world. In the sections that follow, we will define the dataset, our approach, other experiments, results, conclusions and further ideas, and references.

THE DATASET

Apparel attributes are the basic knowledge of the fashion field, and they are large and complex. The competition provided us with a hierarchical attributes tree as a structured classification target to describe the cognitive process of apparel, which is shown below. The “subject” refers to an apparel. Our focus for the competition was in the characteristics of the apparel.

FIGURE 1. The Dataset



Data provided has eight categories, each representing a clothing type. Each category is further broken down into labels defining it in terms of design or length. If the design or length is not clearly visible in the image, an invisible label is assigned. Tables below show the categories, the labels in them and the total number of images inside each label:

Table 1.

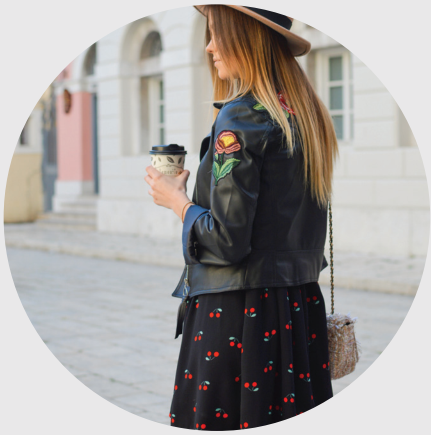
CATEGORY	NUMBER OF IMAGES	NUMBER OF LABELS
neckline_design_labels	17,148	10
sleeve_length_labels	13,299	9
coat_length_labels	11,320	8
skirt_length_labels	9,223	6
collar_design_labels	8,393	5
pant_length_labels	7,460	6
lapel_design_labels	7,034	5
neck_design_labels	5,696	5
TOTAL	**79,573**	

Table 2.

CATEGORY	LABELS
neckline_design_labels	Invisible, Strapless Neck, Deep V Neckline, Straight Neck Neckline, Square Neckline, Off Shoulder, Round Neckline, Sweat Heart Neck, One Shoulder Neckline
sleeve_length_labels	Invisible, Sleeveless, Cup Sleeves, Short Sleeves, Elbow Sleeves, 3/4 Sleeves, Wrist Length, Long Sleeves, Extra Long Sleeves
coat_length_labels	Invisible, High Waist Length, Regular Length, Long Length, Micro Length, Knee Length, Midi Length, Ankle & Floor Length
skirt_length_labels	Invisible, Short Length, Knee Length, Midi Length, Ankle Length, Floor Length
collar_design_labels	Invisible, Shirt Collar, Peter Pan, Puritan Collar, Rib Collar
pant_length_labels	Invisible, Short Pant, Mid Length, 3/4 Length, Cropped Pant, Full Length
lapel_design_labels	Invisible, Notched, Collarless, Shawl Collar, Plus Size Shawl
neck_design_labels	Invisible, Turtle Neck, Ruffle Semi-High Collar, Low Turtle Neck, Draped Collar

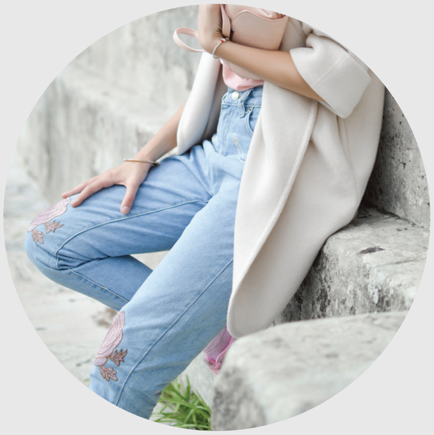
EXAMPLE IMAGES

CATEGORY SKIRT LENGTH LABELS



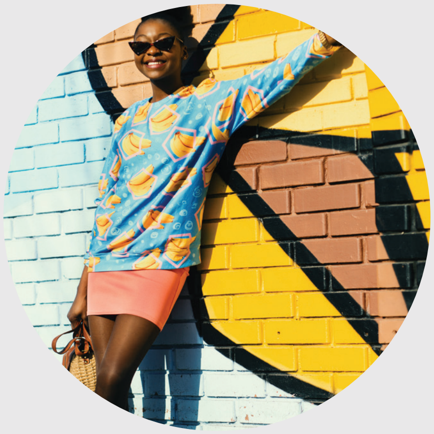
INVISIBLE

(Since the length of skirt is not clearly visible)

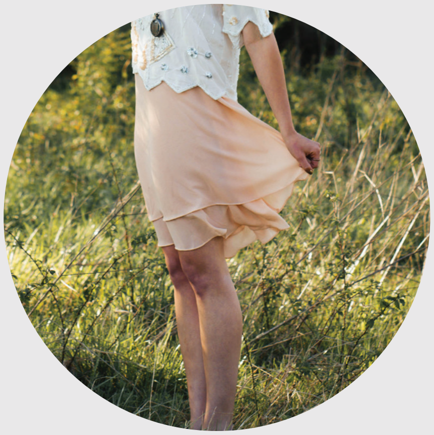


INVISIBLE

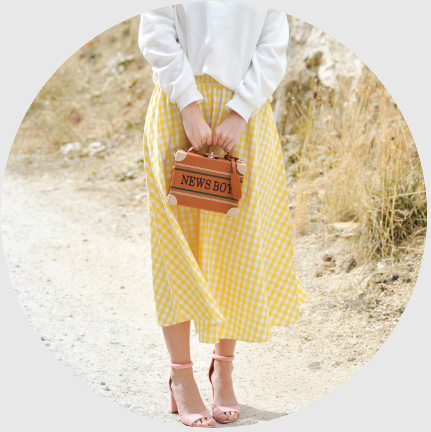
(Since there is no skirt in the image)



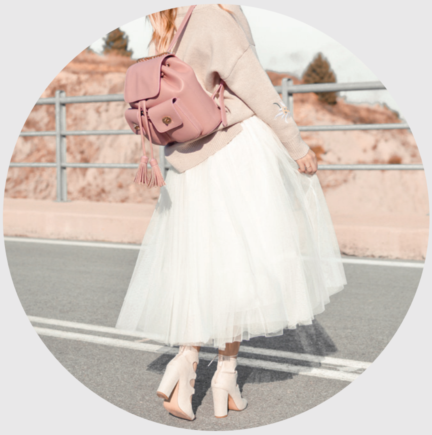
SHORT LENGTH



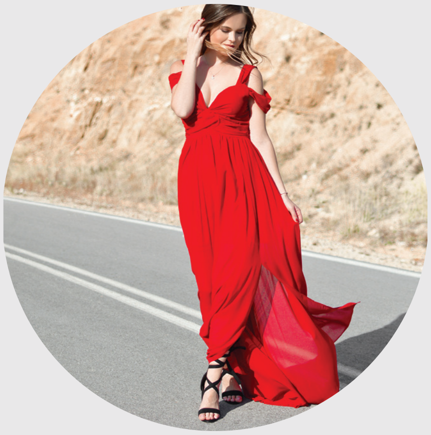
KNEE LENGTH



MIDI LENGTH



ANKLE LENGTH



FLOOR LENGTH

CHALLENGES IN THE DATA

- In some images, the way the person is posing might obscure the design or length of the clothes. For example, if the person is sitting, a floor length skirt might seem like an ankle length skirt
- The background in the image also added to the noise. For some images, it merged with the dress color, making it difficult for the model to distinguish between the dress boundary and the background
- Also, in some cases, the model couldn't differentiate between clothes of similar length (example, knee vs. midi length skirt)

Data is given to us in separate folders for each category. This eliminated the requirement to first predict the category and then the labels inside it. We need to predict the labels of each category. As obvious as it may sound, the test dataset also has a similar structure.

OUR APPROACH

Convolutional neural networks (CNNs) are used to solve the problems related to image classification. We have used the same technique and approach, which can be divided into four parts:

A. Preprocessing the images and data augmentation

B. Choosing network architecture of CNN

C. Optimizing the parameters of the network

D. Test time augmentation

A. PREPROCESSING THE IMAGES

We normalized (took difference from the mean) the pixel values of the images (0-255) to suit the network architecture used. We applied certain transformations like zooming (1-1.1X), adjusting the image contrast (randomly between 0-0.05), rotation (randomly between 0-10 degrees) and flipping the images. This helped in making the model more invariant to orientation and illumination in the image. In every epoch, a random transformation was chosen, so that in every epoch we are showing a different version of the same image, thus avoiding the network to overfit to a set of images.

B. CHOOSING NETWORK ARCHITECTURE OF CNN

We used transfer learning for solving this problem. Transfer learning means using a model that is trained for another task to assist us in solving the problem at hand. This helps in creating the initial base features and avoids training the model from scratch when you have limited data and computational resources. We took network trained on ImageNet data as a starting point. ImageNet is a large database of images, and every year many researchers try to improve upon the accuracy of the classification of objects in ImageNet and submit it to Large Scale Visual Recognition Challenge (ILSVRC). This challenge has 1,000 categories to predict.

To suit the problem at hand, the final output layer after the Fully Connected layers (FC layers) in the architecture were replaced with the number of labels of the given category.

We experimented with different types of Residual Networks: ResNet [1], ResNext [2] and the current state of the art architectures NasNets& SeNets. In our experiments, ResNext gave better results than other algorithms when looked in both accuracy and computational time.

C. OPTIMIZING THE PARAMETERS OF THE NETWORK

FINDING THE LEARNING RATE:

Choosing a starting value of the learning rate is highly important to ensure convergence of the network parameters to the optimal value.

Leslie Smith's (researcher in field of deep learning) recent work on "Cyclical Learning Rates for Training Neural Networks" [3] contains a point on choosing an initial learning rate for the given problem. In summary, the idea is to start with a very small learning rate and gradually increase the learning rate in powers of 2 or 10 for every iteration in the epoch. Initially, when the learning rate is too small, error will decrease at a very slow rate. If you keep on increasing, at some point the learning rate becomes so high that the error skips the minimal value and starts shooting upwards. This indicates that beyond this learning rate shouldn't be chosen, as it has become too high for parameters to converge.

The image below shows the learning rate finder for the ResNext-101 architecture when trained on a category of clothing. Loss has been decreasing drastically between 10-4 to 10-3, and then from 10-2 loss has started increasing, and at 10-1 it has increased drastically. Ideally, we should choose a learning rate between 10-4 and 10-3. We have chosen 10-4 to accommodate another technique of adjusting the learning rate.

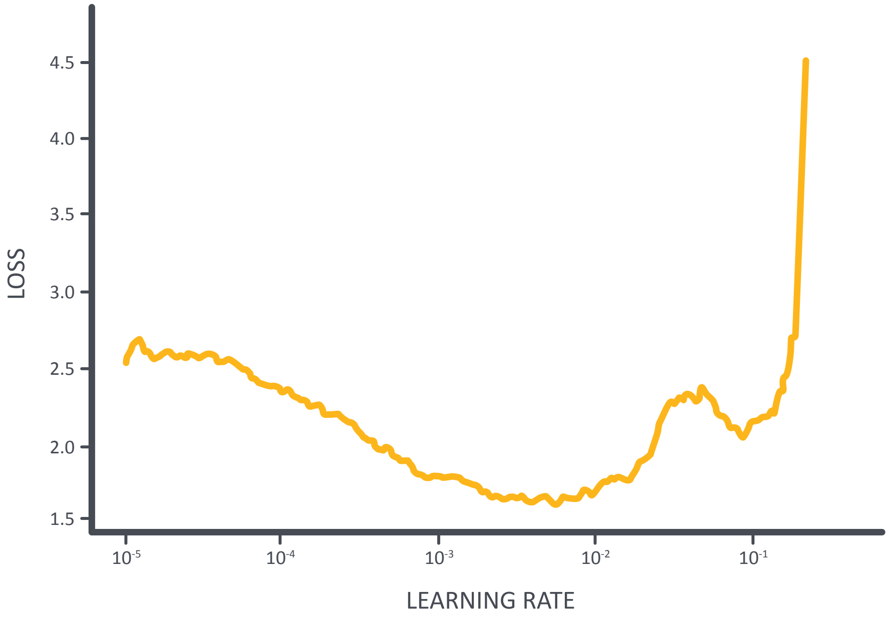


FIGURE 2. The Learning Rate

CYCLIC LEARNING RATE:

Leslie Smith's (a researcher in the field of deep learning) recent work on "Cyclical Learning Rates for Training Neural Networks" points out that instead of just having a constant learning rate across the epochs, the learning rate can be made cyclical across the epochs. In summary, the number of epochs could be equal to the number of cycles, and in each cycle the learning rate resets back to the original learning rate (learning rate chosen from learning rate finder above). Inside a cycle, the learning rate decreases gradually for each batch in cosine fashion. This process helps the network to escape the narrow regions (local minima) in error surface and favors a wider region.

The plots below show the cyclic learning rate plots. After running for the few cycles, we can change the length of the cycles so that the learning rate gradually decreases to help weights converge. When the cycle length is two, in that case, the learning rate of the next epoch is equal to the latest learning rate in the last epoch.

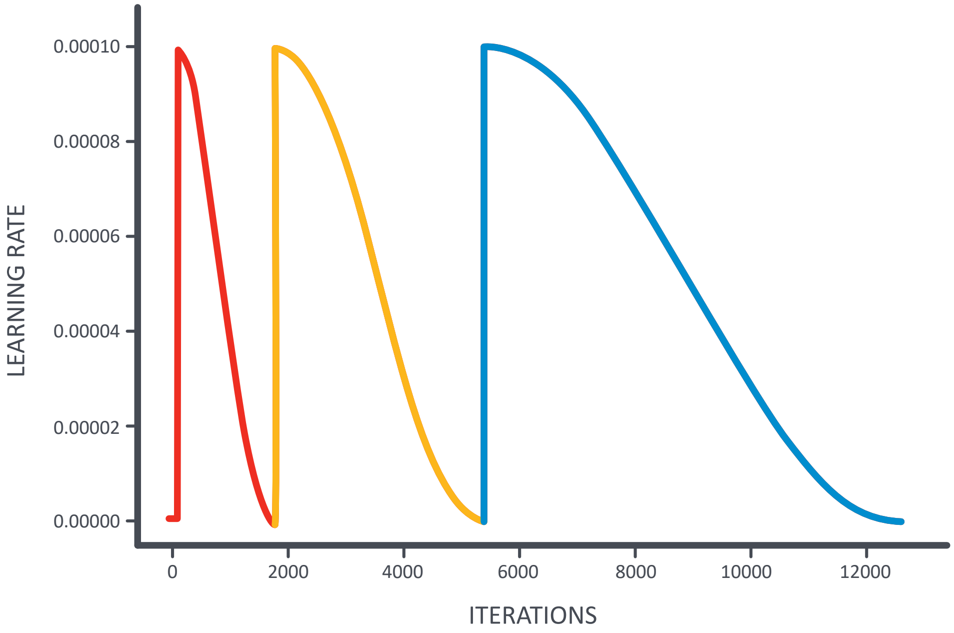


FIGURE 2. Iterations

A total of seven epochs were run in the case shown in the image. Here, the total cycles are three, and the cycle length is multiplied by two times the length of the previous cycle. We can see that the last cycle was run for four epochs, the second cycle was run for two epochs, and the first one was run for one epoch.

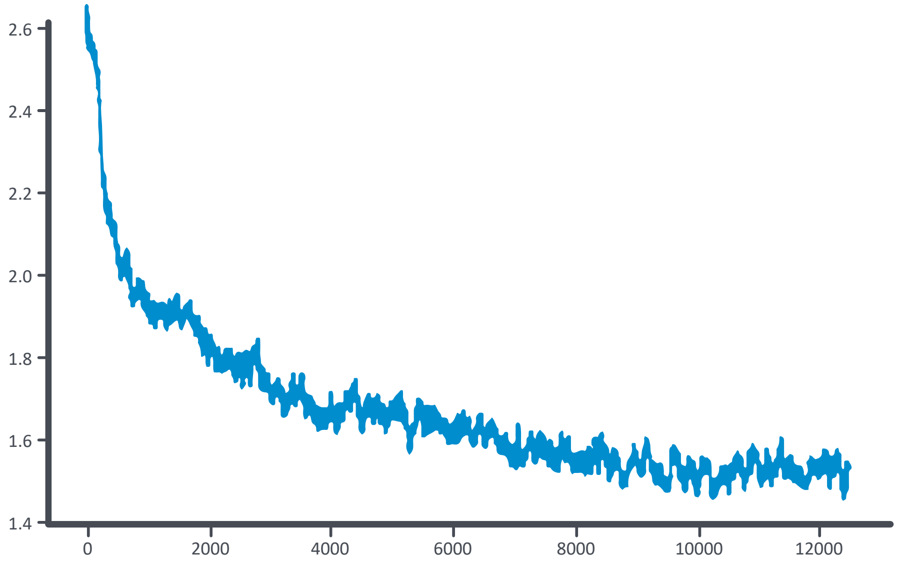


FIGURE 3. Loss vs. Iterations

LOSS VERSUS THE ITERATIONS:

We can see that error surface is not smooth, and the concept of cyclic learning rate can help us jump past the narrow regions of error surface. We have increased the number of cycles and have seen that loss has been constant, indicating that it is not a narrow region of error surface.

UNFREEZING THE LAYERS AND DIFFERENTIAL LEARNING RATES:

As we are using a pretrained architecture and performing transfer learning, not all layers require additional training. As the architectures are state of the art on ImageNet, they are already good in identifying the low-level abstract features like boundaries and edges. Those are captured in the few initial layers of the architecture. Hence, they don’t require much re-training.

We chose different learning rates for different parts of the network, and the layers are grouped into three parts. The first part corresponds to the initial set of layers, the second part corresponds to the layers in the middle, and the third part corresponds to the last set of layers plus FC layers (Fully Connected layers).

Two steps that are used to train the network are listed below:

- Initially, the network is frozen for all layers except for the last Fully Connected layer. We mean those layers are not trained; we are just predicting the values till the layers before the FC layers and trying to tune the weights between Fully Connected layers (two layers of size 512) and the output layers (size is dependent on the category we are predicting).
- Next, the network is unfrozen, i.e. all the layers are made trainable. Now, the learning rate for the three groups of layers is set by the rule of thumb of [lr/100, lr/10, lr] in the order for the images like ImageNet, but in our case [lr/100, lr, lr] has proven to work well. Information getting captured in the middle layers is having equal importance to the information that is getting captured in the layers near to the FC layers. (Here, “lr” refers to learning rate).

This concept of using different learning rates across different layer groups is termed as the differential learning rates [4].

INCREASING INPUT SIZE OF IMAGE GRADUALLY:

The images we received in the data are mostly at 512 pixels, and we resized the images to 224 (since most of the ImageNet images are of this size) for the initial tuning of weights. And then we resized the images to 299, and we ran the same number of epochs using the final weights generated at 224 as initial weights. Then we resized all the images to 512 pixels and used the weights generated, using 299 pixels as the initial weights.

The advantages were twofold:

- We would get computational time advantage, since larger images increase the time it takes to tune the weights of the network. Hence, if we provide the weights obtained from the smaller size of images for the same problem, we are providing optimal weights for the problem and network converges in less time than it takes.
- We get accuracy gains from this. We are providing the data at different sizes; hence, those categories that are very far way in the classification (i.e., sleeveless vs. wrist length of the sleeve) will already be taken care of in the smaller sizes. But the nearby classes will be classified more accurately in the case of higher resolution input image.

D. TEST TIME AUGMENTATION

During the prediction, we applied the same transformation parameters that we used during the training and generated eight images. We choose four of them randomly and predicted on these sets of images and also on the original image. We averaged out the prediction probabilities, and this has increased the accuracy of the predictions obtained. We believe that one possible reason for this is some center cropping could happen while resizing the image, which could result in loss of information from the sides. When we do transformations, that information is captured in one or more of the images, and averaging the probabilities is increasing the accuracy of the model.

OTHER EXPERIMENTS

As discussed earlier, a gradual increase in size is bringing the accuracy improvements, but most of the original images are of size 512 pixels; hence, we applied a concept of super resolution (a deep learning based method to resize the images to a higher resolution). We resized the images to 1024 pixels and performed the similar experiments, but we didn't get the accuracy improvements, and computational time grew exponentially from 720 pixels.

During the semi-finals of the competition, we were provided with the dataset that contains images of apparel that were just hanging on a hanger or on the wall (i.e., not worn by humans). We used a similar approach but used yolo to separate out images of hanger and humans and built separate models. But the combined model of human and hanger always gave better results in comparison with separate models.

RESULTS

We have results for all the categories following similar trends of results across the experiments. Hence, we will present the results of one category: skirt length (as we have provided example images for the same category).

COMPARISON OF RESULTS ACROSS ARCHITECTURES:

We have started with ResNet architecture and moved on to ReNext-50 and ReNext-101. ResNext-101 outperformed all the other architectures, as shown in the results below. Notations: Epoch: Number indicates the cycle-indexing starting with zero; trn_loss- log loss on training data; val_loss: log loss on validation data; accuracy: classification accuracy of validation data.

It can also be observed that the TTA has always provided an improvement in the prediction accuracy when compared to the last epoch's accuracy.

TABLE 2.

RESNET 50				RESNEXT-50				RESNEXT-101			
epoch	trn_loss	val_loss	accuracy	epoch	trn_loss	val_loss	accuracy	epoch	trn_loss	val_loss	accuracy
0	1.182948	0.941432	0.629453	0	1.168192	0.970368	0.621448	0	1.218546	0.996771	0.614763
1	1.093747	0.923126	0.641472	1	1.06272	0.912483	0.640185	1	1.074181	0.914764	0.640086
2	0.918479	0.868569	0.65581	2	0.923527	0.885018	0.649675	2	0.994793	0.896094	0.648168
3	0.991089	0.929273	0.633599	3	0.986922	0.941786	0.637282	3	0.990731	0.874594	0.651401
4	0.900364	0.85171	0.664717	4	0.925634	0.871556	0.649554	4	0.988717	0.860471	0.66056
5	0.797379	0.826631	0.664596	5	0.843087	0.85589	0.663144	5	0.923964	0.847489	0.670259
6	0.768733	0.821365	0.672887	6	0.781276	0.844633	0.666168	6	0.918755	0.851443	0.670259
After unfreezing layers				**After unfreezing layers**				**After unfreezing layers**			
100% 3/3 [02:52<00:00, 57.51s/it]				100% 3/3 [03:31<00:00, 70.66s/it]				100% 7/7 [24:32<00:00, 210.40s/it]			
epoch	trn_loss	val_loss	accuracy	epoch	trn_loss	val_loss	accuracy	epoch	trn_loss	val_loss	accuracy
0	0.740729	0.613043	0.763272	0	0.724171	0.588369	0.785362	0	0.55786	0.502819	0.821121
1	0.66588	0.56469	0.777863	1	0.691434	0.510525	0.803472	1	0.566776	0.498866	0.821121
2	0.474322	0.533103	0.798832	2	0.470145	0.484568	0.816942	2	0.403399	0.411643	0.850754
After TTA accuracy— valset0.8028245518739815				**After TTA accuracy— valset0.8234655078761542**				**After TTA accuracy— valset0.8559432198402341**			

SEQUENTIALLY INCREASING THE SIZE OF INPUT IMAGE:

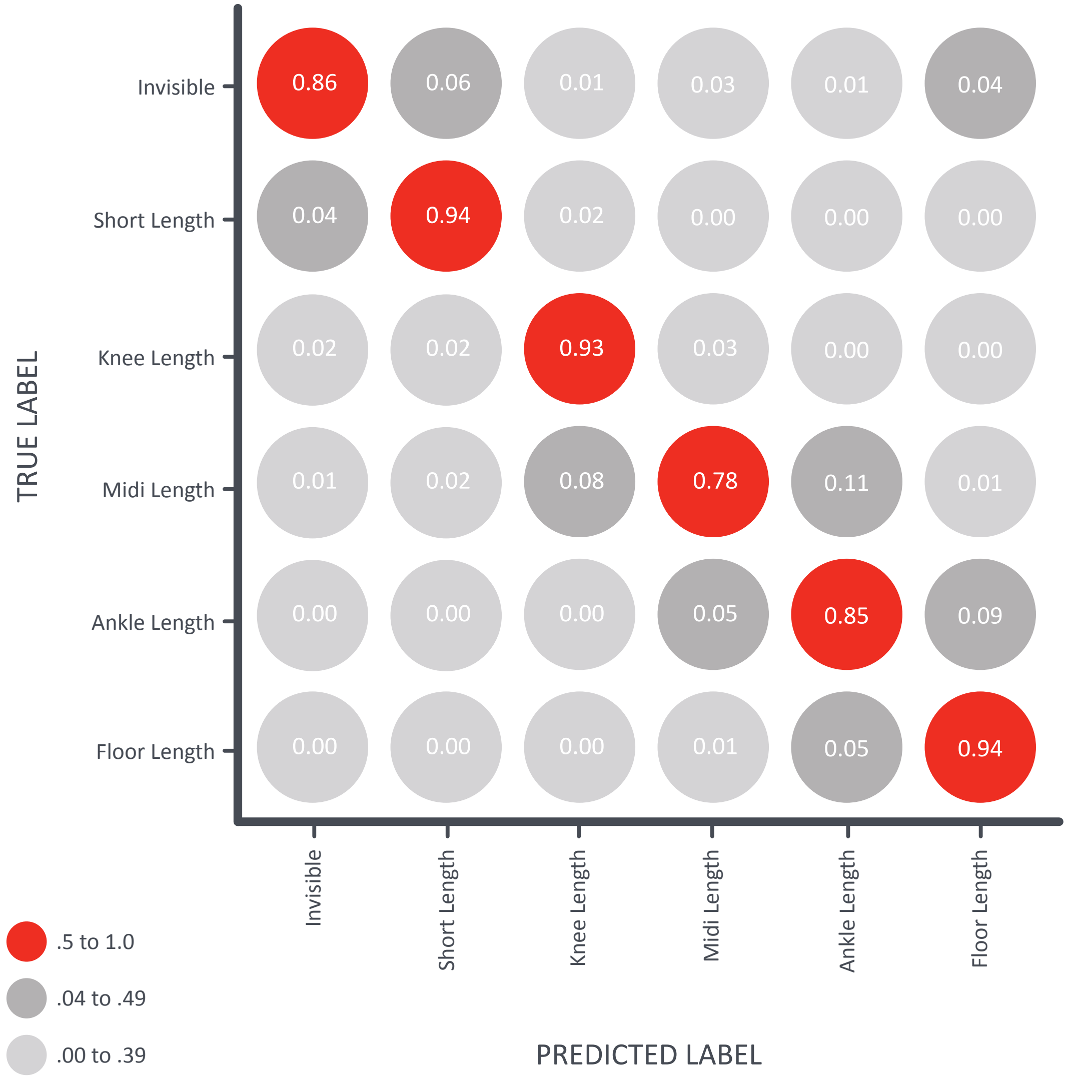
Choosing the best performing architecture, ResNext-101, we have sequentially increased the size of the input image. Accuracy has increased from 85.39% at size 224 pixel to 88.25% at size 512 pixel.

TABLE 3.

INPUT SIZE OF 224				INPUT SIZE OF 299				INPUT SIZE OF 512			
epoch	trn_loss	val_loss	accuracy	epoch	trn_loss	val_loss	accuracy	epoch	trn_loss	val_loss	accuracy
0	1.446606	1.058099	0.600216	0	0.376161	0.407957	0.852909	0	0.405602	0.395749	0.852909
1	1.296488	0.992362	0.635237	1	0.353691	0.410197	0.855603	1	0.43674	0.388715	0.857759
2	1.228191	0.986237	0.642241	2	0.40821	0.403435	0.855065	2	0.367794	0.391069	0.857759
3	1.218322	0.935863	0.646013	3	0.380771	0.409982	0.850216	3	0.383081	0.388278	0.858836
4	1.207356	0.936442	0.644397	4	0.374635	0.404102	0.858836	4	0.387282	0.392801	0.854526
5	1.146785	0.956587	0.644935	5	0.381706	0.398791	0.860453	5	0.375392	0.382729	0.860453
6	1.173328	0.942111	0.64278	6	0.367866	0.398856	0.860453	6	0.341563	0.38279	0.856142
After unfreezing layers				After unfreezing layers				After unfreezing layers			
epoch	trn_loss	val_loss	accuracy	epoch	trn_loss	val_loss	accuracy	epoch	trn_loss	val_loss	accuracy
0	0.571965	0.495939	0.816272	0	0.38667	0.36833	0.858297	0	0.325804	0.344223	0.877694
1	0.632768	0.559584	0.79472	1	0.402689	0.48716	0.835129	1	0.288727	0.362832	0.866918
2	0.354606	0.396713	0.853987	2	0.246952	0.343845	0.872306	2	0.236958	0.327507	0.882543

CONFUSION MATRIX (FOR THE SAME CATEGORY-SKIRT LENGTH):

There is no confusion between the short length and floor length (i.e., those categories that we as humans can also do very accurately). But the model suffers to correctly classify the nearby classes. We have tried other approaches like modeling separately for the nearby categories and making changes in loss function, but none of them are able to solve the problem of confusion of nearby classes.



TRUE LABEL \ PREDICTED LABEL	Invisible	Short Length	Knee Length	Midi Length	Ankle Length	Floor Length
Invisible	0.86	0.06	0.01	0.03	0.01	0.04
Short Length	0.04	0.94	0.02	0.00	0.00	0.00
Knee Length	0.02	0.02	0.93	0.03	0.00	0.00
Midi Length	0.01	0.02	0.08	0.78	0.11	0.01
Ankle Length	0.00	0.00	0.00	0.05	0.85	0.09
Floor Length	0.00	0.00	0.00	0.01	0.05	0.94

FIGURE 4. Confusion Matrix

CONCLUSIONS & FURTHER IDEAS

The way the learning rate is chosen is very important for neural nets convergence, and the way the network is optimized is also another important step in the process of model building. But we feel the current state of the art architectures are throwing away a lot of information when it reaches the FC layers. But taking all the activations will make the parameter space exponentially bigger, thereby causing overfitting and increasing time complexity to tune the network. Current methods are taking the average value of all the channels before the FC layers. By doing so, we are losing the detailed information captured till that layer. We propose the following ideas to improve on this:

- We tried XGBoost at the end of the competition by taking all activation values from all the filters in the layer before the FC layers or final conv layer. We observed that we can get better results when compared to just taking average values of those filters. But due to the time constraint in competition, we haven’t been able to complete the experiment, and we will publish those results soon. In summary, using XGBoost on the activations obtained on filters just before FC layers could help boost the accuracy of nearby classes by capturing some detailed information.

- The approach of bagging could help, where we selectively expose all the activations of the few important filters based on their weights importance and repeat it multiple times and take an average prediction. This might help us capture the detailed information from some important filters.

AUTHORS

ABHEET AGGARWAL
Data Scientist

Mathematics and Scientific Computing graduate from IIT Kanpur. Fascinated by the mathematics behind machine learning. Interested in applications of deep learning, especially in the field of computer vision.

EKTA SINGH
Senior Data Scientist

Material Science graduate from IIT Roorkee. Enjoys problem solving and has worked on creating data driven solutions for various domains like Insurance, Supply Chain, CPG.

GEETHASAIKRISHNA ANUMUKONDA
Data Scientist

Material Science graduate from NIT Warangal. Passionate about solving problems that create a meaningful impact to the society/business using machine learning and deep learning.

REFERENCES

1. ResNet. Deep Residual Learning for Image Recognition. https://arxiv.org/pdf/1512.03385.pdf
2. ReNext. Aggregated Residual Transformations for Deep Neural Networks. https://arxiv.org/pdf/1611.05431.pdf
3. Cyclical Learning Rates for Training Neural Networks. https://arxiv.org/pdf/1506.01186.pdf
4. Lectures by fast.ai. http://www.fast.ai/

ACKNOWLEDGEMENTS

We would like to thank Jeremy Howard and Rachel Thomas for generously helping everyone learn the state of the art deep learning techniques through their lectures at fast.ai [4]. Many of our ideas are inspired from the lectures there.

Fractal Analytics is a strategic analytics partner to the most admired Fortune 500 companies globally and helps them power every human decision in the enterprise by bringing analytics & AI.

Fractal Analytics has presence across 12 global locations including the United States, UK and India and has been recently featured as a leader on Forrester Wave™: Customer Analytics Service Providers, 2017. Fractal has also been recognized as a “Hot Artificial Intelligence (AI)” company by Forbes and a “Cool Vendor” and a “Vendor to watch” by Gartner.

Learn more at www.fractalanalytics.com

fractal

INTELLIGENCE FOR IMAGINATION