Classification of runway images using Convolutional Neural Networks

Gabriela Gandler, Helder Martins, Xinyi Lin

KTH Royal Institute of Technology

Abstract. Our personal clothing style is heavily influenced by the fashion shows promoted by international brands. In this paper we present a method to classify runway images of these shows regarding the season when they were presented. For this task we modeled a Convolutional Neural Network reference architecture and trained it with images from the Runway dataset composed of several different outfit samples. We further experimented and assessed some different model variations as to understand how the classifier accuracy behaves. A final accuracy of 82,7% could be obtained. Finally, the paper proposes some suggestions based on the current work to improve the performance of the model.

Keywords: CNN, convolutional neural network, classification, fashion, computer vision

1 Introduction

Clothes play an important role on how people communicate with the world around and express who they are. The choice of what to wear is influenced by several aspects, including personal taste, cultural background and geographic location. Additionally fashion is significantly influenced by cyclical trends, which can be perceived in seasons. For instance the fashion industry usually offers different clothing styles, which have their characteristics influenced by the current season. A summer outfit, for instance, commonly displays completely different features if compared to a winter outfit.

Being fashion a domain with a high visual appeal, it represents a potentially powerful object for computer vision research. Furthermore the last decade has allowed computer science research to evolve significantly, since the storage and computational capacities of machines are suitable to deal with large amounts of data. Therefore supervised machine learning techniques on clothing data are able to provide insightful results for different applications. Such techniques enable researchers to explore the vast field of object detection algorithms, similarity measures between different outfits, and even detect complex patterns such as personal outfit styles. The following section - *Background* - will be devoted to mention some recent applications.

Convolutional Neural Networks (CNNs) have proven to perform very well within computer vision tasks [1] [2]. Experiments have been done [1] to compare the performance - in computer vision tasks - of CNNs and networks composed solely by fully connected layers. The conclusions show that CNNs are more suitable, since they are able to capture spatial topology, while standard neural networks ignore topological properties of the input. Compared to standard neural network with equally large layers, CNNs have much fewer connections and parameters, which makes them easier to train [3]. The following subsection addresses some characteristics of CNNs in more detail.

1.1 Convolutional Neural Networks

CNNs is a variant of Multi-layer Perceptrons where the idea of exclusively using fully connected layers is dropped in favor of using a smaller set of weights that is shared over the whole input. The idea was inspired by [4] early work on cat's visual cortex, where it was demonstrated how the complex arrangements of cells in the brain is sensitive to a smaller region of the visual field, which is called the *receptive field*. These cells act like filters that react to the spatial features, and are well suited to explore locally correlated features of the input. This characteristic tackles one of the main challenges related to training deep neural networks, which is the enormous amount of weights required to learn any reasonably sized input, and the tendency to overfit the parameters to the training data. By enforcing a local connectivity pattern between neurons of adjacent layers, CNNs have the ability to create a different representation of the input, which evolves along the layers, from locally correlated features to more complex characteristics. Another peculiarity of CNNs is that by using a shared set of weights the same feature is bound to be detected in each part of the input, disregarding where the feature is located.

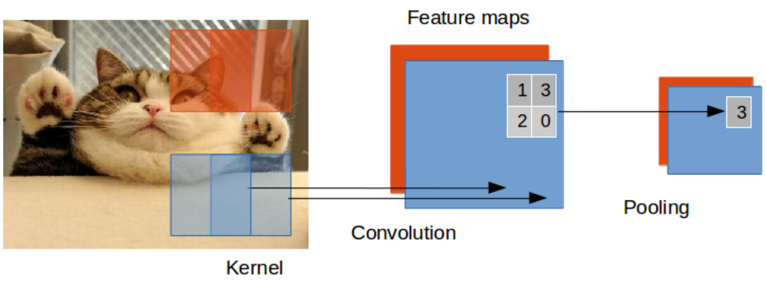


Fig. 1: Convolution and pooling operations

The main building block of a CNN model is the *convolutional* layer, which is responsible for learning specific local features of the data. This is accomplished by applying a set of *filters* (also called *kernels*) to a specific part of the input at each time through a convolution operation that generates a single scalar value. The kernel is then moved a set number of positions (also known as *strides*) over the input before applying the convolution again. The output of this operation over the whole input, which can be seen in Figure 1, is called *feature maps*, since it represents the features extracted by each one of the filters.

The kernels represent the network parameters where the learning will happen, as they will specialize over time to extract different features at some spatial position of the input. As the convolutional layers are stacked one after the other, the receptive field - i.e. the covered area of the input -, will increase in size as features which are correlated are grouped together. This means that convolutions firstly learn a good representation over a small part of the input, becoming increasingly more global as more layers are added.

Another important component of a CNN is the *pooling* layer. In this operation, a function is applied to a small window of the input to return a value which is independent of the order of how the features in this window are organized. The reasoning behind this is that, once the kernels have learned a feature, the exact position of them is irrelevant and thus can be disregarded. Commonly *max pooling* is used, through which the feature with largest value is chosen to represent the entire window.

Multi-classification problems usually demand probability distributions over the classes, so a *softmax* layer is commonly added after fully connected layers in a CNN. This layer is simply a function which transforms the output so that each output node has values between 0 and 1 and all the values sum up to 1. Each class has a corresponding output node and the one with highest probability in an inference represents the predicted class assigned to the corresponding input.

1.2 Goal and contributions

The goal of our work is to offer a framework to classify images of clothing outfits into different seasons. In order to do so we use a dataset called *Runway* provided by [5], which offers plenty of labeled data for the supervised training. The model we propose is a CNN whose final architecture is determined by performing experiments and evaluating the effects of selected parameters. We then compare the performance of subtly different architectures and find the framework that best adjusts to the problem at hand.

By tackling this problem we intend to offer initial steps to further understand outfit styles in different seasons. We expect this work to be the starting point of more potential fashion applications, which may for instance be able to propose new unseen outfits given a certain season.

This paper is organized in different sections. Section 2 addresses recent works related to the given problem. Section 3 describes the method used to tackle the classification problem, by means of thoroughly describing the network's architecture, as well as discussing the model's theoretical computational complexity. Subsequently Section 4 presents the dataset, displays the results of different experiments and discusses the experiments. Finally Section 5 is devoted to concluding remarks.

2 Background

The field of computer vision has been displaying an increasing interest towards the analysis of clothing datasets. A variety of fashion applications exploring different image processing tasks have been published in the recent years. Such applications range from systems in the surveillance context, which are able to recognize different clothing categories in videos [6], to systems that can distinguish clothing styles [7], revealing clues about different personal styles, social status, occupation and so on.

A technique to measure the similarity between images of clothing outfits was developed by [5]. Starting from images of models on the runway, this application enables a mapping that captures the most similar corresponding outfits in a real-world scenario, i.e. outfits that people wear in real life. In order to offer the mapping from runway to street fashion outfits, they train multiple models of fashion similarity. This is done by making use of a feature representation that derives from pose estimation and clothing parse, as well as of human judgements of outfit similarity. Additionally the learned similarity measures are evaluated by means of predicting season, year and brand related to a runway image.

Large multilayer CNNs have been applied to tackle a variety of computer vision tasks since LeCun *et al.* [8] showed an impressive solution to a handwritten digit classification task. Another milestone in computer vision should be granted to [3], who presented a CNN originally used to classify images of objects in 1000 classes, within the *ImageNet* dataset. Given the impressive accuracy of this solution, this architecture has been used as an inspiration to other neural network architectures, as well as an enriching source for feature extraction [9].

Within the fashion domain, the last two years have been showing an increasing acceptance of CNNs to tackle computer vision problems. For instance a fashion item detector was developed by [10], which consists of a system that detects garments and accessories by surrounding them with bounding boxes. As opposed to segmentation-based methods, this work is the first detection-based fashion item spotting method. Their solution can be summarized according to the following recognition pipeline: firstly an object proposal algorithm generates a set of candidate bounding boxes; then a pretrained CNN is used to extract features from the image patches within the bounding boxes; subsequently these features are fed into an SVM to obtain an appearance-based posterior for each class; finally geometric priors based on pose estimation are also used to update the class probability.

Besides detecting objects, CNNs are also useful to detect different styles depicted in images, i.e. styles related to different outfits and social features. This means that CNNs have been used to detect patterns that are even more complex than the ones present in physical objects. An example of such a work is given by [9]. The authors used the pre-trained and fine-tuned CNN on [3] (in the *ImageNet* dataset) to generate features in order to distinguish different urban tribes. This method consists of a CNN feature-based architecture, which combines individual and global scene features and makes use of SVM classifier on the extracted features. This method could output 71% of correct classifications of urban tribes.

As mentioned in the previous section, we aim to offer a contribution in detecting different outfit styles in images as well, being these styles the different seasons. This is precisely the reason why deep CNNs are appealing for our problem: they are able to construct a complex feature hierarchy which highlights the *under-the-hood* characteristics of each season. Additionally their capacity is adaptable by adjusting their architecture to the learning nuances that are demanded by the particular problem.

3 Approach

The core mathematical algorithm behind our classifier consists of a supervised learning model. We then make use of labeled data to train the model. Our starting point is to define a baseline network's architecture. Subsequently we discuss the theoretical computational complexity of the proposed model.

3.1 The Architecture

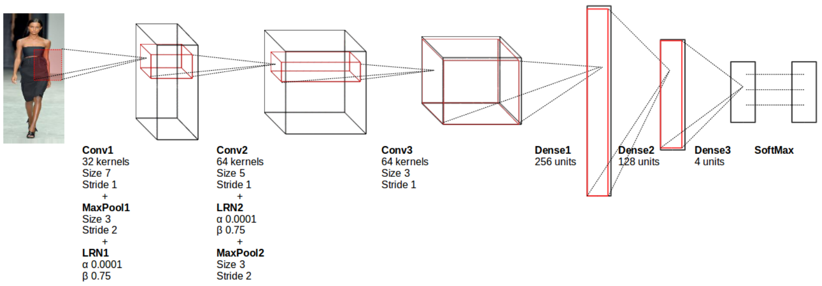


Fig. 2: *Baseline* model

To serve as a basis of comparison for our experiments, a reference network was created heavily inspired by [3], scaled down due to time and resources constraints. The model, which can be seen in Figure 2, will be used as a reference point to assess the performance - regarding the accuracy of seasons classification - of models that are subtle variations of it.

Our so called *Baseline* model is composed of 3 convolutional layers, followed by 3 fully-connected layers. A *Rectifier Linear Unit*(ReLU) activation function is used throughout the whole architecture. A *max pooling* operator is used, where the stride size is set to be smaller than the kernel as to make the operation overlap. *Local response normalization*(LRN), which is a technique used by [3] that consists in normalizing the convoluted tensors over the neighboring kernels, is also used after every convolutional layer. At the output of the first dense layer a *dropout* function is applied, which consists of "dropping out" the weights with a set probability as to ignore their influence during the forward and backward passes [11]. This has been reported to increase the model's generalization capacity [3]. Finally, a *softmax* function is utilized as to calculate the posterior probability distribution of classes (in our case, seasons) given the input image.

After assigning probabilities to every image belonging to each one of the classes, a cost function needs to be used to evaluate how well our network performed. A *cross entropy* function is suitable for a classification problem like the one we are addressing, so it was used in our network implementation. To improve our network at each iteration of the training algorithm, weights should be updated as to progressively reduce the value of the loss function. Gradient descent optimization method is a standard technique for minimizing loss which consists of calculating the gradients of the loss function w.r.t. the network's parameters and updating these parameters towards the direction where the loss is reduced. The *stochastic gradient descent* is an approximation of the gradient descent method which is frequently needed in practice since calculating the true gradients over the whole dataset is too costly to be computed [12]. In it, a subset of the input (a "mini-batch") is instead used to compute the derivatives, which ends up resulting in a smoother convergence towards the local minima, in comparison to one sample stochastic gradient descent. As for regularization, a *L2-norm* was applied to every weight parameter as to penalize large values which frequently means the model is overfit to the training data.

3.2 Computational Complexity Estimation

Typically the complexity of a neural network can be calculated by summing up all the multiplication and addition (mul-add) operations that need to be performed at each layer.

The number of mul-add operations $\#mul\_adds$ needed at each layer in a forward pass of a single image can be calculated as in Equation 1.

$$\#mul\_adds = output\_size \times \#mul\_adds\_per\_output \tag{1}$$

For convolutional layers, this is basically measured by multiplying the size of a single kernel by the size of the output tensor generated by this kernel, multiplied by 2, since both multiplications and additions need to be performed at each stride. For dense layers, a similar calculation is done. Based on that, we calculated the mul-adds for our *Baseline* model as to get an insight on the computational demand for training the network. The result can be seen on Table 1.

Table 1: Mul-add operations per layer for the *Baseline* model

	Conv1	**Conv2**	**Conv3**	**Dense1**	**Dense2**	**Dense3**
Output Size	151×71×32	76×36×64	38×18×64	384	192	4
Kernel Size	7×7×3	5×5×32	3×3×64	43776×384	384×192	192×4
Mul-adds	100.863.168	280.166.400	50.429.952	33.619.968	147.456	1.536

4 Experiments

In this section, we present results for the *Baseline* model, as well as for variations of its architecture, by changing some parameters. Table 2 highlights the differences across the investigated models.

Table 2: Comparison of different models

Name	Meaning	1st conv. layer	2nd conv. layer	3rd conv. layer	Dropout	Overlapping max pooling
Baseline	Baseline architecture	✓	✓	✓	✓	✓
NoDrop	No dropout	✓	✓	✓		✓
NoOverlap	No overlapping in max pooling	✓	✓	✓	✓	
NoConv3	No third conv. layer	✓	✓		✓	✓

Each model keeps the same architecture used in the *Baseline* model, with however a specific change. *NoDrop* eliminates the dropout, while *NoOverlap* does not use overlapping max pooling kernels, which means that the kernel size and the stride in the max pooling layers are the same and equal to 3. Finally the *NoConv3* eliminates the third convolutional layer. Furthermore additional experiments based on the findings from these models are undergone. These experiments explore some few other settings and refine the proposed solution.

4.1 Dataset

The dataset used in this paper comes from the *Runway* dataset [5], which is composed of several fashion show images from different brands, most of them are of models walking on the runway. Some samples of this dataset can be seen in Figure 3. The main characteristic of these images are the similar position that most of them exhibit: models are normally centered in the frame and in an upright position, unlike other more general datasets where the position of the objects may vary significantly between samples.

The *Runway* dataset contains 348,598 images of 9,328 fashion shows from 2000 to 2014, together with their season(e.g., Spring 2010), category(e.g., Ready-to-Wear), brand(e.g., Chanel), date and city of the fashion show [5]. Due to time and resources constraints, only a part of the original *Runway* dataset will be used in the experiments. This subset, henceforth also called *Runway* Dataset, contains 131,072 images of four different fashion shows seasonal events (spring, fall, pre-fall and resort). As is standard in machine learning, the data was split into 3 different sets: 75% of it was assigned to the training set which will be used to train all the different models, 10% belongs to the validation set used to compare the accuracies in different architectural settings, and 15% to test set used to check the accuracy of the best performing architectures. Figure 4 shows the distribution of seasons across the dataset. All fashion events happening in spring are here grouped into the season Spring, which includes Spring Ready-to-Wear, Spring Menswear and Spring Couture, and the same approach is also implemented for season Fall.

Fig. 3: Dataset samples

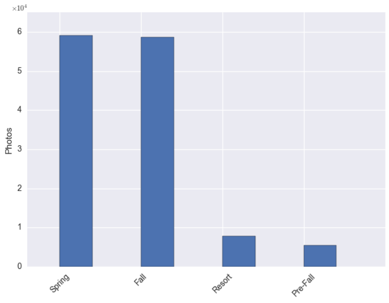


Fig. 4: Dataset statistics

4.2 Experiment on Baseline model

Firstly the *Baseline* model was trained using the training set. The model was trained for 8 epochs and a batch size of 32 input images. All the parameters related to the training procedure, as well as the architecture's parameters are thoroughly described in the Appendix. Results of the evaluation on the validation set can be seen on the left graph of Figure 5. For every epoch two evaluation points were used. It can be noticed that the validation error rate always decreases across the 8 training epochs, which means that the *Baseline* model does not seem to overfit to the training set. The graph on the right side of Figure 5 shows the behavior of the batch accuracy and the batch loss during training. The lines represent moving averages with windows 64 and 16, respectively. The horizontal axis displays the training steps, which are in total around 24576 steps (given 98305 training samples, batch size of 32 and 8 epochs). This graph shows that the loss presents a decreasing tendency during training, while the proportions of correctly classified images within batches increases.

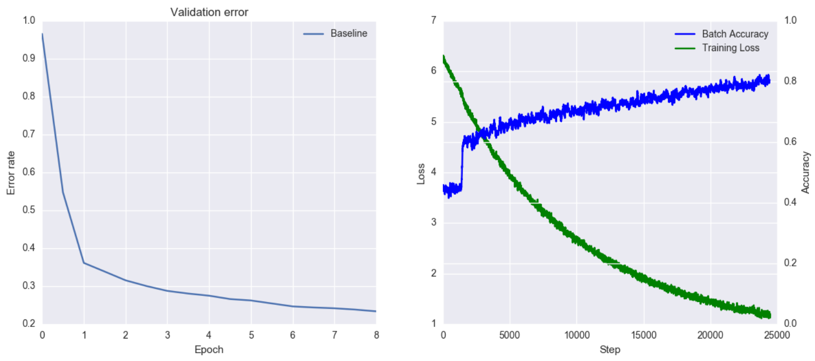


Fig. 5: Performance of the *Baseline* model

4.3 Experiments on variations of the Baseline model

In this subsection, we focus on analyzing the effect of singular architectural structures, namely dropout, overlapping max pooling and the third convolutional layer. By evaluating the different models in the *Runway* dataset, we expect to extract insights about the effects of the mentioned structures on the model performance.

Dropout In the *Baseline* model, about 17 million parameters need to be learned. Due to the complexity of this model, it is highly prone to overfit. As mentioned in Section 3, the dropout was used in the first fully connected layer, which is a method to prevent overfitting [11]. The idea of dropout comes from ensemble learning, which shows that combining many different models can significantly reduce the test error rate and make better predictions. However for complex neural network models it is too time-consuming to learn several different architectures. Hence, the dropout was proposed [13] to play a similar role as ensemble learning. The function of dropout is to randomly "drop out" some units, which means setting the output of these units to be zero and these units would not contribute to the forward pass and the backpropagation. Since in each iteration there are different units to be dropped out, this is similar to using different models to predict. However, using the dropout means more time to converge since the units need more examples to learn.

In Figure 6, the green line represents the error rate of our so called *NoDrop* model, while the blue line represents our reference model. As expected, the error rate of *NoDrop* model drops faster in the beginning because it is easy to converge. At the end of 8 epochs, the two lines have almost achieved same error rates, namely 23.4% for the *Baseline* model and 22.6% for the *NoDrop* model. In theoretical analysis, the *Baseline* model should perform better because it uses the dropout. However, both models were only trained for 8 epochs, which might not be enough for them to converge, especially the *Baseline* model, which

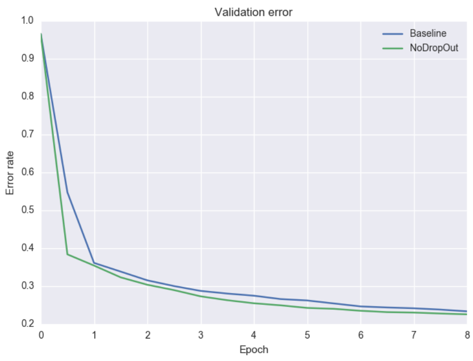


Fig. 6: *Baseline* and *NoDrop* models

demands more epochs. The limitation of training epochs may be the reason why the accuracy of *NoDrop* model is slightly better in this experiment.

Overlapping pooling Pooling layers summarize the neighborhood information and compresses the output size of convolutional layers. It can be noticed from Figure 2 that the *Baseline* model used pooling size 3 and stride 2, which means the model carried out an overlapping pooling operation because the stride was smaller than the pooling kernel size. In this experiment, the *NoOverlap* model presents pooling size 3 and stride 3. The comparison with the reference model can be seen in the Figure 7. The *NoOverlap* model, which is represented by the green line, performs much worse. From this experiment, it can be concluded that using overlapping max pooling strategy is considerably practical to reduce the error rate, as already experienced in [3].

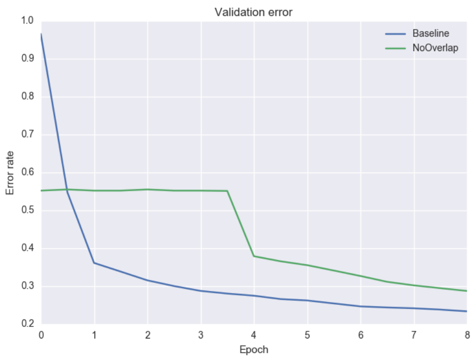


Fig. 7: *Baseline* and *NoOverlap* models

Table 3: Validation accuracy of the four models

	Baseline	NoDrop	NoOverlap	NoConv3
Validation Accuracy	76.6%	77.4%	71.3%	80.0%

Third convolutional layer Inspired by Krizhevsky *et al.*[3] work, we seek to replicate a similar architecture as much as resources constraints allow us to. However, it is still an interesting topic to discover the performance of the architecture with different number of convolutional layers. *NoConv3* model was established for this purpose, and only had two convolutional layers. In Figure 8, blue line still represents the *Baseline* model, while the green line is the *NoConv3* model. It is surprising to see that *NoConv3* achieves a lower error rate through all 8 epochs. One reasonable explanation is that unlike the ImageNet which has about 15 million images and more than 1000 labels, the *Runway* dataset used in this paper is quite small, the targets are four classes only and there are not many pose variations in the pictures. Hence, a complex model might not be needed.

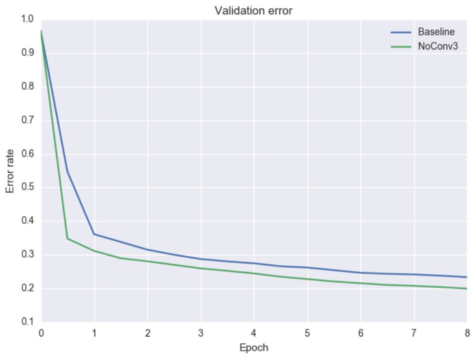


Fig. 8: *Baseline* and *NoConv3* models

Table 3 displays the validation accuracies for the four different investigated models. It can be seen that *NoConv3* model got the highest accuracy, namely 80%. Then, we performed another experiment to observe how the *NoConv3* model performs if given the chance to train for more epochs. We merged the training and validation sets and re-trained the model, using in total 15 epochs. The performance of the new model can be seen in Figure 9. The error rate on the test set is 17,7% and it almost achieves 100% accuracy on the training set.

Since models *NoConv3* and *NoDrop* were the ones that performed the best, we trained a new model which incorporated both characteristics. This means that this model has only two convolutional layers and no dropout. Figure 10 shows how the model performed in the training and test sets across the 15 training

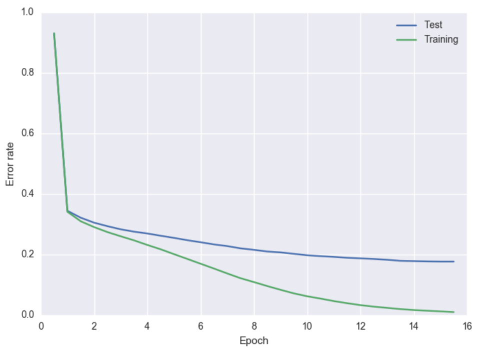


Fig. 9: Performance of the re-trained *NoConv3* model

epochs. It can be seen that the model starts to overfit in the last epochs, as the error rate doesn't improve in the test set - and even gets slightly higher at the end. The best error rate achieved for this setting is 17,3%, which is the best performance across all the experimented models.

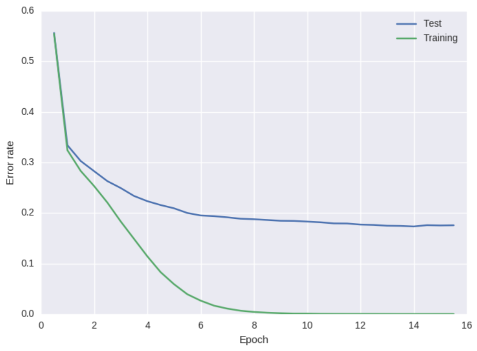


Fig. 10: Performance of the model without the third conv. layer and without dropout

Finally we have also trained a sightly different version of the last mentioned model. Instead of using Gaussian with zero mean and a constant standard deviation for the initialization of weights, we used Xavier initialization [14] to set the standard deviation of the weights in each layer. The evaluation of this model on the test set can be seen in Figure 11. This model performs slightly worse than the previous one - it reached a minimum error rate of 18,7%. A possible reason is that the Xavier initialization was not originally done to tackle ReLU activation functions, but other kinds of activation, such as the hyperbolic tangent function.

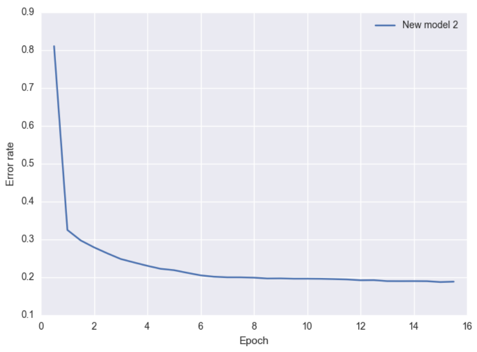


Fig. 11: Performance of the model without the third conv. layer, without dropout and with xavier initialization of weights

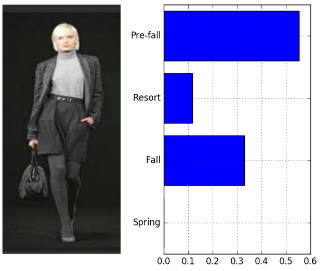
4.4 Discussion

In Figure 12 a qualitative assessment of the classification scores of some images in the *Runway* dataset was made. In the Fall and Spring samples, they were scored similarly possibly due to the similar outfit (short skirts and long sleeves) that both display. Even though it was classified correctly, the outfit of the Pre-fall image was surprising since we expected it to be more closely related to a summer garb, but since the body is almost fully covered it justifies the high score of the Fall class. The sample from the Resort collection, which is composed of outfits more closely related to summer and spring garbs, was classified with the highest score, with some score also being attributed to similar Pre-fall and Spring seasons.

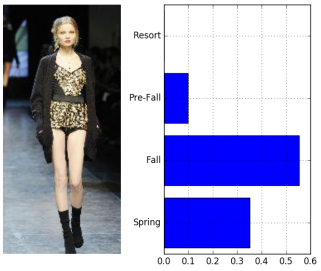
Our *Baseline* model was built based on what was reported to give positive classification results in the literature. A careful look in the network architectural structures described in the Appendix reveals that there is a number of structures that blend our model from overfitting, such as the use of dropout and weight decay. Our results show, as expected, that there is a relatively good generalization capacity due to these structures.

The model that performed the best was the one that didn't use both dropout and the third convolutional layer. This was a surprising result, since it shows that less regularization provides a better accuracy, for the given number of epochs. It is interesting to notice, however, that the decrement in regularization (no dropout) happened along with the decrement in model complexity (no third conv. layer). The achieved bias-variance *trade-off* seems to be the most beneficial one. Additionally we noticed that this model could converge faster, which is a consequence of not using dropout. We also spoted a starting tendency to overfit on the last training epochs, which was also expected. With this experiment we could notice how far the training phase could be streched until it starts to overfit.

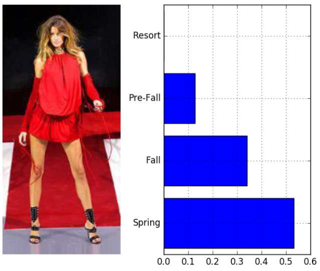
The proposed solution can be improved in different ways. Firstly using more training data may be beneficial to the problem. Only 1/3 of the *Runway* dataset



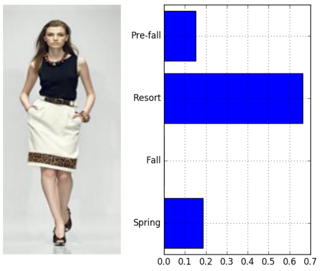
(a) Pre-fall example



(b) Fall example



(c) Spring example



(d) Resort example

Fig. 12: Class distribution examples

was used in the mentioned experients. The classification accuracy may improve if the model can be trained with the whole dataset. The data is originally highly unbalanced. Therefore techniques to balance the data - such as to augment data from certain classes - may be interesing. Furthermore, a pre-processing phase of the original images may offer a better starting point for the training procedure and therefore a potentially higher accuracy. Normalization techniques such as whitening can be explored. Similarly, focusing on the human on the image might help to reduce the noise and error rate, and this could be accomplished by cropping the input images. Regarding the initialization of weights, a new technique [15] to set the standard deviation of Gaussian distributed weights, which was developed for ReLU activation functions, may offer better performance. Finally further investigations involving other architectural structures, such as convolutional kernel size and dense layers' size, may offer revealing improvements as well.

5 Conclusions

In this report, we presented a classifier of seasons in runway images using convolutional neural networks. We proposed a reference model to use as a basis of comparison for experiments and investigations about the effect of network's architectural structures. These different models were then compared to the reference one, and their accuracy in the classification task was measured. Our experiments have shown that for this problem a simpler model with one less convolutional layer, as well as less regularization, performs better than the other models.

The classification of runway images represents to us a promising application of computer vision and CNNs. We believe this work can be the starting point of a more complex fashion application. Further work may involve the development of an application which is able to propose new outfits according to the current season. This might require new features, such as the determination of similarity measures between outfit images.

References

1. Patrice Y Simard, Dave Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. *Proceedings. Seventh International Conference on Document Analysis and Recognition*, pages 958–963, 2003.
2. Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
3. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, pages 1097–1105, 2012.
4. David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.
5. Sirion Vittayakorn, Kota Yamaguchi, Alexander C Berg, and Tamara L Berg. Runway to realway: Visual analysis of fashion. *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 951–958, 2015.
6. Ming Yang and Kai Yu. Real-time clothing recognition in surveillance videos. *18th IEEE International Conference on Image Processing (ICIP)*, pages 2937–2940, 2011.
7. M. Hadi Kiapour, Kota Yamaguchi, Alexander C. Berg, and Tamara L. Berg. Hipster wars: Discovering elements of fashion styles. *European Conference on Computer Vision*, 2014.
8. Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.
9. Yufei Wang and Garrison W. Cottrell. Bikers are like tobacco shops, formal dressers are like suits: Recognizing urban tribes with caffe. *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 876–883, 2015.
10. Kota Hara, Vignesh Jagadeesh, and Robinson Piramuthu. Fashion apparel detection: The role of deep convolutional neural network and pose-dependent priors. *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2016.
11. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
12. Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.
13. Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.
14. Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *International conference on artificial intelligence and statistics*, pages 249–256, 2010.
15. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

6 Appendix

Table 4: Parameters of Baseline Model

	Description		**Parameters**
Training parameters	Epochs		8
	Batch size		32
	Training examples/Epoch		98305
	Learning rate - Initial value		0.1
	Learning rate - Decay factor		0.95
	Learning rate - Epochs/Decay		2
Architecture's parameters	*Conv1*	Weights - Shape	[7, 7, 3, 32]
		Weights - Std dev	0.0001
		Weights - Decay factor	0.0005
		Stride	1
		Padding	Yes
		Biases - Initial value	0
	MaxPool1	Weights - Shape	[1, 3, 3, 1]
		Stride	2
		Padding	Yes
	LRN1	Depth radius	4
		Bias	1
		Alpha	0.001/9
		Beta	0,75
	Conv2	Weights - Shape	[5, 5, 32, 64]
		Weights - Std dev	0.0001
		Weights - Decay factor	0.0005
		Stride	1
		Padding	Yes
		Biases - Initial value	0.1
	LRN2	Depth radius	4
		Bias	1
		Alpha	0.001/9
		Beta	0.75
	MaxPool2	Weights - Shape	[1, 3, 3, 1]
		Stride	2
		Padding	Yes
	Conv3	Weights - Shape	[3, 3, 64, 64]
		Weights - Std dev	0.0001
		Weights - Decay factor	0.0005
		Stride	1
		Padding	Yes
		Biases - Initial value	0.1
	Dense1	Weights - Shape	[43776, 384]
		Weights - Std dev	0.04
		Weights - Decay factor	0.0005
		Biases - Initial value	0.1
		Dropout - Probability	0.5
	Dense2	Weights - Shape	[384, 192]
		Weights - Std dev	0.04
		Weights - Decay factor	0.0005
		Biases - Initial value	0.1
	Dense3	Weights - Shape	[192, 4]
		Weights - Std dev	1/192
		Weights - Decay factor	0.0005
		Biases - Initial value	0