PatchIt: Self-Supervised Network Weight Initialization for Fine-grained Recognition

Patrick Sudowe
sudowe@vision.rwth-aachen.de
Bastian Leibe
leibe@vision.rwth-aachen.de

Visual Computing Institute
RWTH Aachen University
Germany

Abstract

ConvNet training is highly sensitive to initialization of the weights. A widespread approach is to initialize the network with weights trained for a different task, an *auxiliary task*. The ImageNet-based ILSVRC classification task is a very popular choice for this, as it has shown to produce powerful feature representations applicable to a wide variety of tasks. However, this creates a significant entry barrier to exploring non-standard architectures. In this paper, we propose a self-supervised pretraining, the *PatchTask*, to obtain weight initializations for fine-grained recognition problems, such as person attribute recognition, pose estimation, or action recognition. Our pretraining allows us to leverage additional unlabeled data from the same source, which is often readily available, such as detection bounding boxes. We experimentally show that our method outperforms a standard random initialization by a considerable margin and closely matches the ImageNet-based initialization.

1 Introduction

ConvNets have been successfully applied to a large number of tasks in Computer Vision. A popular approach is to start training with a weight initialization obtained by training on the ImageNet-based ILSVRC classification task. This results in a two-phased procedure which has produced superior results on a large variety of tasks (c.f. [2, 5, 6, 14, 17, 21]). Here, the ImageNet classification task serves as an *auxiliary task*. Supervised learning on this auxiliary task (*pretraining* phase) yields the initialization for training the final model (*fine-tuning* phase).

This setup has grown popular, mostly because successful architectures in combination with publicly available weights promise good results quickly. Unfortunately, this creates a lock-in situation, so that the fine-tuning popularity causes considerable frustration: Architectures such as AlexNet [11] or VGG [15] have been designed for the ImageNet task and need not be optimal for other tasks with differing domains. However, when using pretrained weights, it is hardly possible to modify the network architecture. Even small changes, like switching non-linearities or adapting the input sizes, render the publicly available weights useless. Certainly, one could pretrain an adapted net on the ImageNet-based ILSVRC classification task, but this is extremely costly in terms of time and hardware resources. If one

Pages 75.1-75.12
DOI: https://dx.doi.org/10.5244/C.30.75

instead starts out with a random initialization of the weights, performance often drops significantly. As a consequence, such modifications are rarely explored. Therefore, it seems that much of the flexibility in deep nets is not leveraged currently.

This paper presents an alternative pretraining method – the *PatchTask* – which provides a viable alternative to ImageNet pretraining. The core idea is to leverage data from the *same domain* as the target task for pretraining. The pretraining is self-supervised, *i.e.*, it solely relies on automatically generated rather than human annotated labels. We target fine-grained recognition tasks that appear in person analysis applications (*e.g.*, pose estimation, re-identification, action and attribute recognition). Their common aspect is that they make predictions for an object that has been located before (*e.g.*, by a detector). So, we will assume such a specific input domain.

The *PatchTask* idea is inspired by the work of Doersch *et al.* [3], who propose an auxiliary task defined by the spatial layout of pairs of patches. In contrast to their work on general images, we focus on fine-grained recognition, where the input images come from a *restricted data domain* (*i.e.*, bounding boxes showing persons). In this restricted setting, it is feasible to directly predict the original location of single patches (Fig. 1). The application scenario in this work is to predict pedestrian attributes from the *PARSE-27k* dataset [17], such as gender or whether the person is carrying a bag. Here, the data domain is bounding boxes produced by a pedestrian detector. The representations learned here could directly be applied to other person analysis tasks, such as re-identification, action recognition, or pose estimation. However, the *patch task pretraining* approach is general and extends to any task defined over detection bounding boxes.

Our approach makes it possible to leverage additional unlabeled data in a very natural way. In practice, one can often easily obtain more unlabeled data from the same data source (i.e., domain) as the target task. For example, it is easy to collect additional detection bounding boxes. One can then define an auxiliary task on this data and perform a regular training. The assumption is that the network has to pick up some of the structure of the input data, in order to solve the auxiliary task. Accordingly, the obtained weights can serve as a good initialization for the target task.

This paper makes the following contributions: (1) We describe a family of self-supervised *patch tasks* for fine-grained analysis. (2) We demonstrate and evaluate their use for human attribute recognition, where we achieve state-of-the-art performance without using external labels (in particular, without ImageNet). This facilitates further exploration of architectures. (3) We provide data for person analysis pretraining and supporting code that may be used to improve person representations in other ConvNet architectures.

2 Related Work

Weight initialization is important for solving the non-convex optimization problem inherent in training deep networks. Historically, there are three general approaches to initialization: sampling random weights, unsupervised pretraining (*e.g.* based on reconstruction error), and transferring weights from auxiliary tasks.

The fastest method is to start with a random weight initialization, which can be tricky in particular for very deep networks. For example, Simonyan *et al.* report a multi-stage procedure to iteratively train their popular VGG16 network, adding groups of layers each time [15]. The problems were caused by vanishing or exploding gradients due to bad scaling of the activations, which grow worse with network depth.

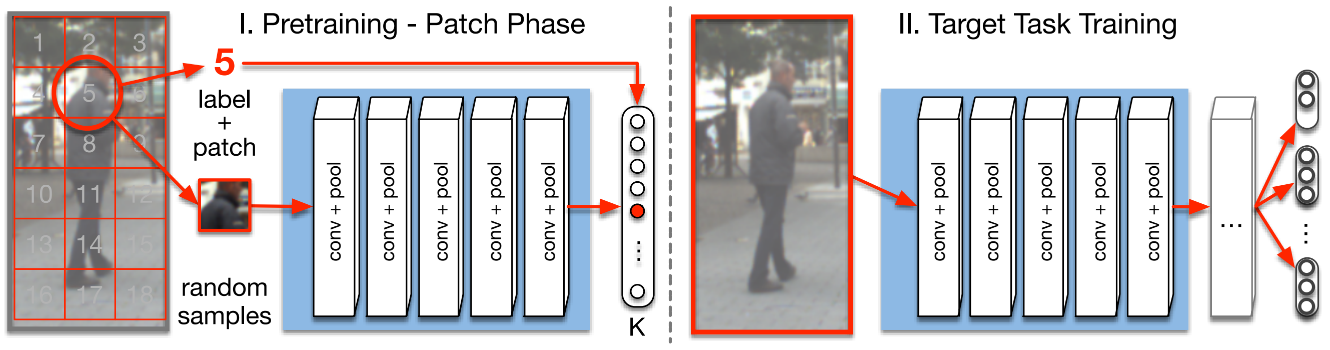

Figure 1: Patch Task: Classify the extraction position given one 32×32 pixel patch. During the pretraining phase, the model needs to encode local patch structure. The parameters are transferred to the target task net. Subsequent fine-tuning benefits from a better initialization.

A closer inspection of the activation distributions within the network gave rise to improved initialization strategies. Glorot *et al*. proposed a method that scaled weight variance according to a layer's fan in and fan out [7]. Their approach was refined by He *et al*. by adapting the variation for the popular *ReLU* non-linearity [9]. This enabled them to train extremely deep networks with more than one hundred layers.

Another, more complex, initialization option is pretraining. On a more general note, Erhan *et al*. discuss unsupervised pretraining as a means to improve deep learning methods [4]. They find that pretraining induces a regularizing effect on the models, although their work does not specifically address ConvNets. Unsupervised pretraining has been used to initialize hierarchical feature representations [13]. Another option is to use a reconstruction task in combination with variants of autoencoders [18]. Instead of using such unsupervised pretraining, one can transfer a representation learned for a different task. Yosinski *et al*. show the efficacy of transferring weights from auxiliary tasks [20]. The main motivation is that the training paradigm in pretraining is essentially the same as training for the target task, and as such is well understood. Several authors observe that the ImageNet weights, even without task specific fine-tuning, yield a good feature extractor for a variety of tasks [8, 12, 14]. In general, however, it seems to be more promising to adapt the weights by fine-tuning, *i.e*., by training the full network on the target task [1, 6, 15, 17, 20, 21]. Initializing with ImageNet weights is very popular, but it has limitations, which has led to a growing interest in alternatives. The main limitation is that the network architecture cannot be changed easily, if one wants to leverage the available weights.

This gives rise to the more recent approaches of self-supervised initialization. Wang *et al*. learn a representation by tracking objects through videos [19]. Doersch *et al*. work on images and learn to predict relative spatial positioning of pairs of patches[3]. Both of those approaches make use of large sets of images or videos. Hence, they are learning representations for very general settings. In contrast, we aim at leveraging self-supervised learning for a specific domain for which there is often an abundance of unlabeled data from the same source is available. This is particularly attractive for small training sets, where deep networks tend to overfit. In practice, this is a scenario often encountered, as it is costly to annotate examples.

3 Method: Patch Task

In this section, we introduce the details of the *PatchTask* pretraining. The core idea is to make the network learn about the structure in the images without using ground truth labels from the target task. The auxiliary task is self-supervised, *i.e.*, trained on automatically generated labels. The immediate advantage is that some learning can occur already without any danger of overfitting on the target task labels. Furthermore, one can often easily obtain more unlabeled data from the same source. Such additional data can be incorporated directly in a self-supervised pretraining step. The challenge is to define a self-supervised auxiliary task which is hard enough to force the model to pick up relevant structure in the images.

To this end, we propose the *PatchTask*: Given a square patch of pixels from the input, the task is to predict its origin out of K possible locations. The locations are discrete positions within the input image (*c.f.* Fig. 1). This is a K-class classification problem. Similarly to placing pieces in a jigsaw puzzle, knowledge of the whole picture facilitates placing individual pieces. Hence, we expect that the network will pick up structure of the images while learning to solve the patch task.

We train train the *PatchTask* model by iterating over all training images and randomly extracting one patch from the K possible positions. The starting weights are randomly initialized following the method of He *et al.* [9]. Note that the resulting parameters can be transferred to the final net (*i.e.*, for the target task) without changes (blue box in Fig. 1), as the convolution parameters are independent from the input size.

We use patches of size 32×32. Observable structure is already present in a window as small as this. A larger patch size would decrease the number of non-overlapping patches that could be placed into the input images, thereby making the classification task easier. The 32×32 patch size allows for a maximum of five groups of convolution layers each followed by a 2×2 max-pooling layer. This matches the setup of the popular VGG16 network architecture. So, it is possible to use the *PatchTask* to obtain an initialization for the convolutional layers of VGG16. Following common interpretation, these layers form a hierarchical representation of the data. We believe that learning this representation from the target domain should be superior to a representation learned on unrelated images (*i.e.*, ImageNet images).

In this work, we focus on input images which are pedestrian detection bounding boxes. While our experiments will evaluate the final performance on the attribute recognition task defined by the *PARSE-27k* dataset [17], the patch task described here is equally applicable to other fine-grained recognition tasks. The only prerequisite is that the input images need to have some structure – *e.g.*, they all contain detections of a particular class of objects.

3.1 PatchTask Variants

There are many possible variants of the *PatchTask*. The individual patches should not overlap too much, which given a patch size limits the maximum number of patch locations. However, less locations yield a different, possibly easier, classification problem. In contrast, the larger the patch size, the more structure can potentially be discovered within. During training, the patches are sampled randomly. To further augment the training set one can introduce small random shifts of the patch locations (*i.e.*, *jitter*).

From the design space described above, we have selected four variants for detailed analysis – *PatchTask18*, pairwise *PatchTask18*, *PatchTask8*, and *PatchTask8+margin* (Fig. 2). In our experiments, all input images are 100×200 pixels. A grid of 3×6 patches fills the

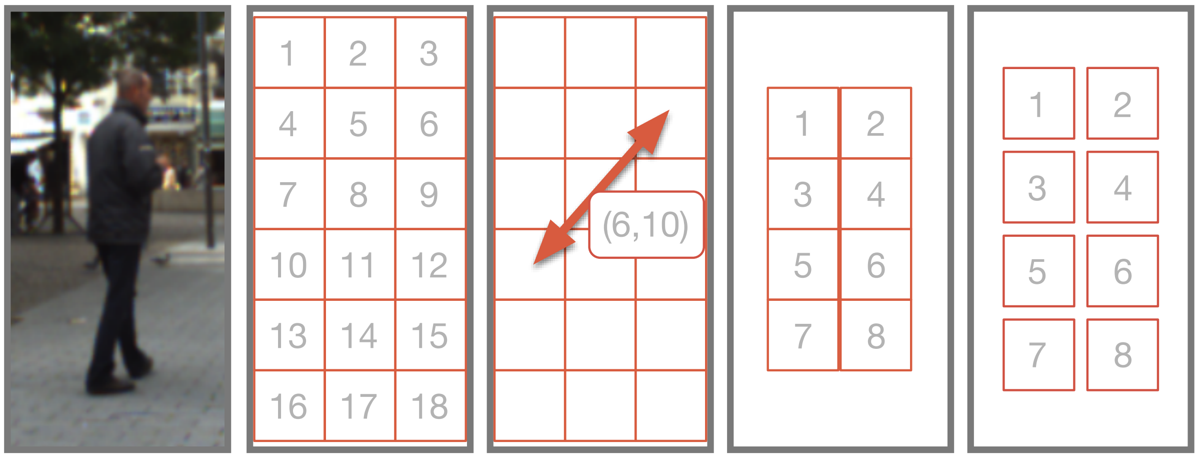

Figure 2: Variants explored in our experiments – from left to right *example input image*, *PatchTask18*, pairwise *PatchTask18*, *PatchTask8*, *PatchTask8+margin*. 18 patches fit tightly into the input image. 8 patches allow optionally for a margin, which enables a heavier data-augmentation by adding random noise to the positions (jittering). To predict the extraction location of a patch, a model needs to learn a representation of local structure within the patches. We leverage this for the target task.

space almost fully, and results in an 18-class classification problem – *PatchTask18*. Considering that the most relevant content, the person, is usually close to the bounding box center, it might make more sense to focus on this area. This idea leads to a 2×4 grid, centered both horizontally and vertically – *PatchTask8*. For these variants, we use a small amount of random noise (± 4 px in both directions) on the positions to augment the training dataset (*i.e.*, jitter). As the 8 patch configuration does not cover the image fully, it is possible to introduce a margin between the patch positions and add a stronger jitter (± 12 px). We name this variant *PatchTask8+margin*. This is a form of data augmentation, which leads to a larger number of training examples. In addition to those single-patch variants, we also evaluate a pairwise task – *pairwise PatchTask18* – which tries to estimate the relative location of pairs of patches, similar in spirit to the task proposed by Doersch *et al*. [3].

4 Evaluation

The ultimate objective of the *PatchTask* is to provide a good initialization for a target task. We adopt the attribute recognition task from the *PARSE-27k* dataset [17] for our evaluation, which forms an ideal testbed for several reasons. First, since person analysis is an interesting application area, an approach to learn a good representation may be leveraged directly for other similar tasks such as pose estimation. Moreover, the *PARSE-27k* dataset is based on video sequences, where only persons from a subset of the frames have been labeled. Only the detections of every 15th frame have been annotated with attribute labels. However, we can leverage additionally all unlabeled detection bounding boxes for the *PatchTask* training. Note that the *PARSE-27k* dataset assigns each video sequences to exactly one dataset split, *i.e.*, training, validation, and test. Hence, we use all detection boxes from the training set's video sequences, regardless of whether attribute labels are present or not. This amounts to a total of $244,584$ examples, with K possible extraction locations each. These bounding boxes are raw detector output, *i.e.*, there was no additional manual supervision. Thus, the overall procedure is self-supervised. The occasional false positive detections do not hurt the

PatchTask.

The main questions we want to answer with the following experiments are the following: (1) Is pretraining with PatchTask beneficial in comparison to a random initialization? (2) How do variants of the PatchTask influence target task generalization? (3) What is the accuracy achieved on the *PatchTask*?

4.1 Experimental Setup

All of the experiments presented below are based on the same architecture: VGG16 from [15]. For our experiments, we use the groups of convolutional and max-pooling layers, but omit the fully connected last layers. This means the base net has 13 convolutional layers with weights. The base net is extended by corresponding layers for the *PatchTask* prediction or for the attribute task, accordingly. Unless stated otherwise, we use the following experiment parameters: We use standard batch-wise stochastic gradient descent (SGD), wth learning rate 0.001 and Nesterov updates with momentum 0.9. For simplicity, we use a constant learning rate throughout the process and also keep the same for both pretraining and the fine-tuning phase.

Patch Task. Patch Training is continued until the accuracy on the validation set plateaus. We prepare one fixed validation set of patch + label pairs for each of *PatchTask8* and *18*. All experiments use the same validation set. Sec. 4.2 reports accuracy on the validation set. We abort training after a plateau has been reached and use the model snapshot (taken after each training epoch) with best validation error for fine-tuning. Training runs within 12-24 hours on a standard GPU (GTX 980Ti). Note that this is significantly less than the training time required for the ImageNet task, or the procedure proposed by Doersch *et al.* [3], which ran for weeks on the ImageNet data. We find that the *PatchTask* accuracy sometimes does further improve with longer running training sessions. But one motivation of this work is to save time, and thus we abort *PatchTask* training after 150-200 epochs, unless indicated otherwise.

Fine-tuning. After the patch training phase, we perform the same fine-tuning procedure for the *PARSE-27k* attributes in all experiments. This means the network adaptation is identical across all experiments. On top of the first layers, which remain unchanged, we add one shared layer of 512 densely connected units and one loss layer for each attribute. This follows roughly the same procedure as Sudowe *et al.* [17], who published the dataset. We report the test set *mAP* score of the model with best *mAP* score on the validation set. For the score computation, we follow the evaluation protocol suggested by the dataset publishers.

Random Initialization. There are several popular variants to randomly initialize network weights. The most widely used ones are the methods proposed by Glorot *et al.* [7] (sometimes called Xavier) and more recently He *et al.* [9]. The latter proposes to scale standard deviation by a factor of $\sqrt{2}$ to adapt to the expected value of the *ReLU* non-linearity. As all of the networks used in our experiments use *ReLU*, we adopt this initialization. We find that, with these settings, training does converge with this random initialization even for our relatively deep models with 15 or more layers.

4.2 Patch Task Accuracy

We first report some details on the accuracy on the *PatchTask* itself. Naturally, it is not clear that a high patch-level classification accuracy will translate to an equally successful target

Task	Accuracy
Patch Task 8	81.16%
Patch Task 8 + margin	80.72%
Patch Task 18	67.75%

Table 1: Accuracy of the different *PatchTask* variants on their respective task – *PatchTask* 18 has more classes, so the accuracies are not comparable to *PatchTask8*. There is only a slight difference for the additional margin. These numbers show that the models successfully learn to predict patch locations, *i.e.*, they learn to represent the input domain.

Model	Initialization	mAP
AlexNet [17]	ImageNet	63.6%
VGG	Random He	67.70%
VGG	ImageNet	73.60%
VGG	*PatchTask18*	72.66%
VGG	*PatchTask8*	71.14%
VGG	*PatchTask8+margin*	72.76%

Table 2: *PARSE-27k* Results (mAP). Comparison of different initialization methods. VGG-based networks outperform earlier work. The *PatchTask* initializations improve clearly over random initialization (method of He *et al*. [9]), showing the merit of self-supervised pretraining. Our proposed methods *PatchTask18* and *PatchTask8+margin* get very close to the more costly ImageNet initialization, but use no external labels.

task model. However, it is a first indication of whether *PatchTask* training has managed to produce a meaningful representation of some inherent structure within the data. Tab. 1 shows the accuracies on the patch classification task. We report the accuracy on a set of $100,000$ patches extracted from detections on the *PARSE-27k* validation set. This ensures that the validation set does not overlap with the training set, as the *PARSE-27k* training and validation sets are comprised of separately recorded video sequences. For both tasks, the final models reach accuracies well beyond the prior probabilities. We use the same validation set across experiments for a given *PatchTask*.

Doersch *et al*. mention problems of overfitting due to chromatic abberations (CA), *i.e.*, shortcuts found by the pretraining, which do not help the final task. We do not observe similar overfitting behavior. Initially, we randomly dropped color channels as suggested by [3]. However, it turned out that this was not necessary on the data used here, so we abandoned this strategy. The reason could be that the detection boxes used in our work may originate from anywhere in the original camera frame, so the patch location does not directly correlate with lense effects such as CA.

4.3 Target Task Results

The ultimate objective is to create the best performing model on the target task. So the results presented next evaluate the *PatchTask* variants with respect to the final performance in terms of target task *mAP* (we adhere to the datasets evaluation procedure [17]).

Single Patch Task. Tab. 2 shows the evaluation results on the target task (*i.e.*, binary attribute recognition). First, it is obvious that the network architecture is key to good perfor-

Model	Initialization	mAP	#epochs
VGG	Random He	67.70%	40
VGG BN	Random He	67.32%	28
VGG	*PatchTask18*	72.67%	43
VGG BN	*PatchTask18*	70.48%	21

Table 3: Batch Normalization Effect: We observe much faster convergence, but the final performance is slightly better for the models without *BN*.

mance. The previously published result of 63.6% mAP is based on the *AlexNet* architecture [17]. This is easily outperformed by the *VGG* architecture, even trained with randomly initialized weights, for which we observe 67.7% mAP. Note that this result is for a random initialization following He *et al*. [9]. We found that the scaling proposed by He *et al*. to account for the *ReLU* non-linearity yields better results. This large difference among the architectures underlines our goal of facilitating quick turnaround time in network exploration.

All three *PatchTask* variants improve over the random initializations. The best result, obtained with *PatchTask8+margin*, clearly improves results by 4 percentage points to 72.76% mAP. The results of *PatchTask18* and *PatchTask8+margin* are very similar, yet the subsequent fine-tuning procedure on the *PatchTask18* initialization converges considerably faster (21 vs. 42 epochs). Although based on a much faster training procedure, both *PatchTask8* and *PatchTask18* manage to come within 1% mAP of the performance obtained by costly ImageNet pretraining. Thus, both tasks present viable alternatives.

Pairwise Patch Task. In order to connect our work to Doersch *et al*. [3], who work on pairs of patches and their relative positions, we adapt our *PatchTask18* to predict a pairwise patch location configuration (Fig. 2). The setup is not identical, but similar to the setup originally proposed by Doersch *et al*. Effectively, this uses two independent patches from the same training example. Hence, a model might benefit from modeling more complex dependencies, where prediction of single patch locations is limited to the local structure within one patch. Here, we sample a pair of patches per input image, which leads to a $K \times (K-1)$ classification problem. Initial experiments showed no benefit of the pairwise patch prediction compared to the single patch setup in our scenario. An experiment with Batch Normalization and without DropOut resulted in 63.98% and 64.06% mAP for single and pairwise *PatchTask*, respectively. The pairwise setup needs more iterations for convergence and each takes longer for each iteration as more data has to be processed. We therefore suggest to use the more direct single variant in scenarios where it is applicable, *i.e*., whenever the input domain is as controlled as in fine-grained recognition tasks.

4.4 Detailed Analysis

In this section, we present experiments addressing particular interesting aspects of the method.

Effect of Batch Normalization. Batch Normalization (*BN*) is an established technique to improve the training process by normalizing intermediate activations [10]. *BN* introduces additional parameters, a shift and scale, for each activation. Thus, training becomes slightly slower, which is countered by a better convergence behavior.

We ran extensive experiments with *BN* for the activations of each convolutional layer. Tab. 3 presents the *BN* results. We consider two weight initializations, random and *Patch-*

Task18. In both cases, *BN* reduces the required number of training iterations significantly (43 vs. 21 epochs). But also in both cases, the final performance drops, unfortunately (72.67% vs 70.48% and 67.70% vs 67.32%). At least in our experiments, we could not observe a clear advantage of applying *BN*, despite the time savings. This may be an artefact from our small batch size of 8 to 16 for the fine-tuning step. Due to memory limitations in the available hardware, we could not use a larger batch size in our experiments. This is a direct consequence of using the VGG16 network, which we chose in order to compare our results to ImageNet pretraining (*i.e.*, publicly available weights). Our *PatchTask* makes it much easier to explore the potential of smaller networks by reducing the turnaround time. We expect applications in fine-grained recognition to benefit greatly from more specialized network architectures, and it is likely that *BN* can again provide an advantage for those.

Effect of DropOut. In the same series of experiments with batch normalization in training as before, we also evaluated the effect of DropOut (as described in[16]). We find that generalization benefits from this, which is unsurprising and in line with results in the literature. All of the results shown in Tab. 2 are obtained with DropOut (Bernoulli noise with p=0.5) applied after each max pooling step both during patch training and fine-tuning.

We report an experiment with *BN* and *PatchTask18* initialization, but disabled DropOut in both pretraining and fine-tuning, resulting in a performance drop from 70.48% to 63.98% mAP. This is a significant drop, even below that of a randomly initialized network trained with DropOut. While DropOut slows down the training progress (*i.e.*, more epochs are required to reach optimal validation score), it leads to much improved generalization of the models. To answer the question which part of the pipeline benefits more from DropOut, we disabled DropOut for a fine-tuning run starting from the ImageNet weights. This resulted in a less pronounced performance drop from 73.60% to 72.12% mAP. We draw the conclusion that the patch pretraining crucially depends on the regularization enforced by DropOut.

Effect of Jitter on PatchTask8. As mentioned before, the *PatchTask8* allows to add more jitter, leading to *PatchTask8+margin*. The idea is that more training examples might lead to a better representation. We found that in fact it does improve the results from 71.14% to 72.76% mAP (Tab. 2). However, this comes at the cost of a significantly longer training time in the *PatchTask* (almost 200 epochs). In contrast, *PatchTask18* reaches the same target task performance with only half the training (95 epochs). As both tasks run equally fast, this translates into a significant time difference.

Effect of Pretraining Duration. We observe that during patch training the weight variance grows, an effect particularly pronounced in the first convolutional layers. One could interpret this as an indication of highly specialized nodes, although such claims are hard to verify. In the context of this work, we wonder whether there is an effect on fine-tuning. To investigate this, we compare the performance of *PatchTask18* initializations taken from epoch 95 (the previously reported result) and from epoch 400. It turns out that the performance drops from 72.66% mAP (Tab. 2) to 70.29% mAP. Overall, it appears to hurt performance slightly if the weight magnitudes increase.

Variation of Results due to Randomness. The experimental results presented in this paper are not deterministic. Several aspects of the setup incorporate random effects: the stochastic gradient descent used for optimization, random shuffling of the order of training examples, random changes made to each example (jitter and DropOut), and initialization of weights before the start of gradient descent. Obviously, these stochastic components lead to different results in every run, even for identical hyper-parameters. This is a general issue in this area

Run	1	2	3	4	5	mean	std dev
mAP	66.47%	66.27%	66.06%	68.04%	69.51%	67.27%	1.74%

Table 4: Repeat Experiments: We repeat the experiment multiple times and report the distribution of the results in order to assess the significance of the *PatchTask* improvement over random initialization.

of research and not limited to the work presented in this paper. Naturally, it is interesting to quantify the effect of this randomness on the results. In particular, it is important to rule out that observations are not the result of a random outlier. The literature on applications of ConvNets rarely discusses this aspect. Instead, it is common to report results of one experiment. Ideally, one would repeat all experiments very often with identical hyper-parameters and report the distribution of the results. Considering the time and resource cost for the experiments, this is hard to accomplish.

In this context, one might pose the question how significant is the *PatchTask* improvement over random initialization? To address this concern, Table 4 reports results of 5 identical runs of random initializations identical to the setup reported in our earlier experiments (Table 2). This results in a mean over all identical runs of 67.27 and a standard deviation of 1.74. While one may object that a small sample size results in a noisy estimate, it is hardly possible to increase the number of experiments by an order of magnitude.

If we compare this to the 72.66 mAP that we obtained with *PatchTask18* initialization, then it is unlikely that a random effect would boost the performance of our baseline to this level. Strictly computationally, one could say that the *PatchTask* result is more than 3 standard deviations away from the mean. This leads us to the conclusion that indeed the *PatchTask* pretraining does yield an improvement over random initialization.

5 Conclusion

In this paper, we have proposed a novel patch task initialization for pretraining ConvNets. We have shown that our *PatchTask* initialization is superior to standard random initialization, while it comes at relatively minor cost. Compared to an initialization based on ImageNet, our method greatly reduces the turnaround time in exploring network architecture changes, facilitating further research. Our self-supervised method gets very close to the performance level of ImageNet pretraining without using any external labels. We have verified through an additional series of experiments that the results cannot be attributed to random variations. For future work, we plan to use the *PatchTask* initialization to explore other architectures for fine-grained recognition problems, such as pose estimation. Moreover, the *PatchTask* approach could deliver a specialized representation for a verification stage in object detection.

Acknowledgments. This work was funded, in parts, by ERC Starting Grant project CV-SUPER (ERC-2012-StG-307432).

References

[1] Steve Branson, Grant Van Horn, Serge Belongie, and Pietro Perona. Bird Species Categorization Using Pose Normalized Deep Convolutional Nets. In *BMVC*, 2014.

[2] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the Devil in the Details: Delving Deep into Convolutional Nets. In *BMVC*, 2014.

[3] Carl Doersch, Abhinav Gupta, and Alexei A. Efros. Unsupervised Visual Representation Learning by Context Prediction. In *ICCV*, 2015.

[4] Dumitru Erhan, Aaron Courville, Yoshua Bengio, and Pascal Vincent. Why Does Unsupervised Pre-training Help Deep Learning? In *AISTATS*, 2010.

[5] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[6] G. Gkioxari, R. Girshick, and J. Malik. Actions and Attributes from Wholes and Parts. In *ICCV*, 2015.

[7] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[8] David Hall and Pietro Perona. Fine-Grained Classification of Pedestrians in Video: Benchmark and State of the Art. In *CVPR*, 2015.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *ICCV*, 2015.

[10] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167*, 2015.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, 2012.

[12] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks. In *CVPR*, 2014.

[13] Marc'Aurelio Ranzato, Fu-Jie Huang, Y-Lan Boureau, and Yann LeCun. Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition. In *CVPR*, 2007.

[14] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN Features off-the-shelf: an Astounding Baseline for Recognition. In *CVPR Workshop*, 2014.

[15] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556*, 2015.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. In *JMLR*, 2014.

[17] Patrick Sudowe, Hannah Spitzer, and Bastian Leibe. Person Attribute Recognition with a Jointly-trained Holistic CNN Model. In *ICCV'15 ChaLearn Looking at People Workshop*, 2015.

[18] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and Composing Robust Features with Denoising Autoencoders. In *ICML*, 2008.

[19] Xialong Wang and Abhinav Gupta. Unsupervised Learning of Visual Representations using Videos. In *ICCV*, 2015.

[20] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How Transferable are Features in Deep Neural Networks. In *NIPS*, 2014.

[21] Ning Zhang, Manohar Paluri, Marc'Aurelio Ranzato, Trevor Darrell, and Lubomir Bourdev. PANDA: Pose Aligned Networks for Deep Attribute Modeling. In *CVPR*, 2014.