Training Neural Networks to Produce Compatible Features

Michael Gygli
Google Research
gyglim@google.com

Jasper Uijlings
Google Research
jrru@google.com

Vittorio Ferrari
Google Research
vittoferrari@google.com

1. Introduction

In computer vision we often train a different neural network for each task, while reuse of existing knowledge typically remains limited to ImageNet pre-training. However, human knowledge is composable and reusable (e.g. [16]). Therefore it seems prudent to give neural networks these properties too. For example, adding a new class should only require training parts of the network, as is done in incremental learning [6] and few-shot learning [14, 18]. Furthermore, when a network can recognize cars in daylight, this knowledge should help recognizing cars by night through domain adaptation [15].

We believe that a general way to achieve network reusability is to have a large set of *compatible* network components which are specialized for different tasks: Some would extract features from RGB images, depth images, or optical flow fields. Other components could use these features to classify animals, detect cars, or segment roads. The compatibility of the components makes it easy to mix and match them for the task at hand. Besides few-shot learning and domain adaptation, this would also enable training a single classifier which can be deployed on various devices, each with its own hardware-specific backbone network. To explore if it is feasible to obtain a set of compatible components, we ask: is it possible to train network components so that they are compatible directly after training?

This question is related to works which investigate whether different networks trained on the same data learn similar representations, even when they are trained independently [10, 11, 12, 17, 9, 7]. This would allow to recombine components of different networks by learning a simple mapping from the feature representation space of one network to the space of the other [9]. If this mapping is a simple permutation, it means that the networks learn exactly the same features but in different order. In practice, low-level features tend to be repeatedly learned [9], while high-level features learn feature spaces which cannot be mapped through a simple transformation [11, 17, 7, 9]. Instead of such a post-hoc analysis, we want to directly optimize networks to learn compatible features without the need for determining any mapping afterwards.

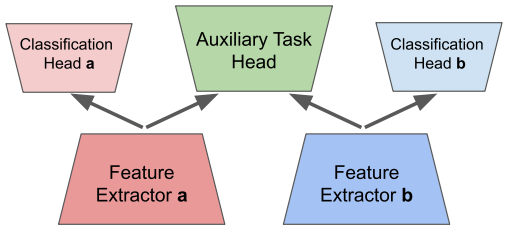


Figure 1: **Experimental setup with auxiliary task head.** We train two networks a and b. We regularize training by adding an auxiliary task head, which may discriminate common classes or predict rotation. This forces both feature extractors to be compatible with the auxiliary task head, effectively aligning their features. This makes feature extractor b compatible with classification head a, and vice versa.

This paper makes a first step towards exploring whether we can train network components to be compatible. We do this in a constrained setting (Fig. 1). We train two networks on two classification tasks, where in this paper both tasks are CIFAR-10 [8]. We define components by splitting the network into two parts: a feature extractor and a classification head. The components of the two networks are *compatible* when we can recombine the feature extractor of one network with the classifier of the other while still producing good predictions. We propose three ways which modify training to make components compatible: (i) We add a shared supervised auxiliary task which discriminates between the common classes (Fig. 1). (ii) We add a shared self-supervised auxiliary task: rotation prediction [1] (Fig. 1). (iii) We initialize the networks using the same random weights. We also compare to a known but more restricted way to align features: knowledge distillation [5].

In controlled experiments on CIFAR-10 [8] we show: (i) we can train networks to produce compatible features, without degrading task accuracy compared to training the networks independently. (ii) the degree of compatibility is highly dependant on where we split the network into a feature extractor and a classification head; (iii) random initialization has a large effect on compatibility; (iv) we can train *incrementally*: given previously trained components, we can train new ones which are also compatible with them.

2. Method

We aim to make network components compatible. We define components by splitting a network at a predefined intermediate layer k. This results in a *feature extractor* $f(\cdot)$ and a *target task head* $h(\cdot)$, parameterized respectively by $\mathbf{\Phi}$ and $\mathbf{\Theta}$. In standard supervised learning, one trains a neural network on task t by solving its task loss $\ell_t(h(f(\mathbf{x}_i;\mathbf{\Phi}_t);\mathbf{\Theta}_t),\mathbf{y}_i)$ over all examples $\mathbf{x}_i$ with label $\mathbf{y}_i$ in dataset $\mathcal{D}_t$. In this paper, we train two networks on tasks a and b and explore various methods that adapt the standard training process to encourage compatibility. In particular, we optimize compatibility between the features produced by their feature extractors $f(\mathbf{x}_i;\mathbf{\Phi}_a)$ and $f(\mathbf{x}_i;\mathbf{\Phi}_b)$.

Training scheme. In order to make networks compatible we investigate two different schemes: In *joint training*, multiple networks are trained at once to be compatible with each other. Instead, in *incremental training*, an initial network is produced by training on task a only. Then, the second network is trained to perform well on task b and also be compatible with the network components trained for task a, *i.e.* it is aligned to them.

2.1. Alignment via Distillation

We explore knowledge distillation [5] to align the features on the example level. Specifically, we add an l_2 loss on the feature activations produced by different networks.

Trade-offs. This is a strong form of alignment, as it aligns the feature activation per example. However, it can only be used if we train on a single dataset (standard distillation), or if there is paired input data for each task we learn. For example, it was used in [2], which relied on pixel aligned RGB and depth images of the same scene.

2.2. Alignment using Common Classes

We propose to align features via an auxiliary task head c, which uses the classes which the two target tasks have in common. This auxiliary task has its own head, but operates on the intermediate features produced by the respective target tasks (Fig. 1), *i.e.* its prediction function is $c(f(\mathbf{x};\mathbf{\Phi}_t);\mathbf{\Theta}_c)$, where $\mathbf{\Theta}_c$ are the parameters of the auxiliary task head. During training, we minimize the target task losses and the auxiliary task loss over the set of training tasks $\mathcal{T}=\{a,b\}$:

$$\sum_{t\in\mathcal{T}}\sum_{\substack{(\mathbf{x}_i,\mathbf{y}_i)\\ \in\mathcal{D}_t}}\ell_t(\mathbf{x}_i,\mathbf{y}_i;\mathbf{\Phi}_t,\mathbf{\Theta}_t)+\lambda\ell_c(\mathbf{x}_i,\mathbf{y}_i;\mathbf{\Phi}_t,\mathbf{\Theta}_c)\mathbf{1}[\mathbf{y}_i\in\mathcal{S}] \tag{1}$$

where $\ell_c(\cdot)$ is the auxiliary task loss. It is computed only over examples that have a label coming from the set of common classes $\mathcal{S}$ ($\mathbf{1}$ is the indicator function). λ is the weight of the auxiliary task loss. While the target task parameters $\mathbf{\Phi}_t$ and $\mathbf{\Theta}_t$ are optimized per task, we tie the auxiliary task parameters $\mathbf{\Theta}_c$ across tasks. This forces the feature extractors $f(\mathbf{x}_i;\mathbf{\Phi}_t)$ of each task t to produce features that are compatible with the same auxiliary task head.

Trade-offs. This method requires that the used tasks share a set of common classes. In this work, we always train on the same dataset, hence the classes are the same. Instead, in a practical setup where networks trained on different tasks should be made compatible, there might be few or no shared classes. We investigate the effect of the number of shared classes on alignment quality in Sec. 3.

2.3. Alignment via Self-Supervision

We also propose an auxiliary task using self-supervision. Unlike the method above, this avoids requiring common class labels. Thus, we minimize the following loss:

$$\sum_{t\in\mathcal{T}}\sum_{\substack{(\mathbf{x}_i,\mathbf{y}_i)\\ \in\mathcal{D}_t}}\ell_t(\mathbf{x}_i,\mathbf{y}_i;\mathbf{\Phi}_t,\mathbf{\Theta}_t)+\lambda\ell_s(\mathbf{x}_i;\mathbf{\Phi}_t,\mathbf{\Theta}_s) \tag{2}$$

where the auxiliary task loss $\ell_s(\cdot)$ uses labels automatically created from the data itself. Typical self-supervision methods create labels by applying some transformation on the input $\mathbf{x}$ and the task is to predict which transformation was applied [13, 1].

Choice of self-supervision task. Ideally, the auxiliary task needs all features produced by $f(\mathbf{x}_i;\mathbf{\Theta}_t)$ to solve the original task. In this work we use rotation prediction [1], which works well for feature learning [1, 18, 15]. The input image is randomly rotated with an angle in $\{0°, 90°, 180°, 270°\}$ and the task is to classify which rotation angle was applied.

Trade-offs. This alignment method is the most general. Its only requirement is that the shared self-supervised task is both meaningful and non-trivial [15]. While such a task can be defined on almost any dataset, the quality of the induced alignment depends on how much the target task and the auxiliary task rely on the same features. In theory, the auxiliary task could negatively affect the performance of the network on the target task. In practice, it instead typically improves performance, when the two tasks are related [18, 4].

2.4. Alignment via Initialization

Zhang et al. [19] demonstrated that for many layers in a trained network, resetting the weights of that layer to their initial values leads to a limited loss in accuracy. This suggests that the initialization defines a set of random projections which highly shape the trained feature space. Hence we propose to produce more aligned features simply by initialising networks with the same random weights.

Trade-offs. This method only works for identical network architectures and only enforces compatibility *before* any training is done. Nevertheless, we expect positive effects when used in combination with the methods above.

3. Experiments

We analyze how the different methods from Sec. 2 perform in producing compatible features. The experimental setup when using an auxiliary task head, our main method, is visualized in Fig. 1. We always train two networks on CIFAR-10 [8], where each network gets examples in a different random order. As network architecture we adopt ResNet-56 [3], which consists of 3 stages with 9 ResNet blocks of two layers. We split this into a feature extractor and target task head after the second stage, unless stated otherwise. By default we train networks jointly, but there is one experiment in which we use incremental training (Sec. 2). When we discriminate between common task labels, we use the first 5 classes in the shared auxiliary task head (Sec. 2.2). We report two metrics on the CIFAR-10 test set: (i) classification accuracy, when swapping task heads, *i.e.* using the target task head of one network with the feature extractor of the other; (ii) the correlation between the features produced by different networks, as in [10, 11].

Comparison of alignment methods. We compare the different methods to produce compatible features in Fig. 2. While the independently trained networks perform at chance level as expected (10%), the proposed methods enforce good levels of compatibility. Notably, we find that using self-supervised rotation prediction leads to strong performance, while also being simple and general. It even outperforms using a shared auxiliary task head that discriminates between 5 of the 10 common class labels (85.8% *vs.* 72.0%). Interestingly, even independently trained networks starting from the same initalization are reasonably compatible. Combining multiple methods offers performance close to the upper bound of using the feature extractor and task head from the same network. For example, combining rotation and same initialization yields 93.4%, which even slightly outperforms distillation. We note that this is partially because adding rotation prediction as an auxiliary task improves the target task accuracy itself from 93.5% to 94%, when not swapping task heads. To summarize, we have shown that strong compatibility can be achieved even when no paired data is available (as assumed in [2]).

Feature similarity. Fig. 3 shows the correlation of features for independently trained networks and when made compatible with self-supervised rotation prediction. Independently trained networks produce misaligned features. After using bi-partite graph matching to align features [10], we measure 0.32 average correlation, in line with [10]. Instead, our method directly optimizes for compatible features, hence making such a post-hoc alignment unnecessary. Our method not only aligns features, it also makes them more similar, yielding 0.61 correlation. Interestingly, even though the correlation is still imperfect, there is only a limited effect on classification accuracy (Fig. 2). This demonstrates that perfect alignment is not a requirement for compatibility.

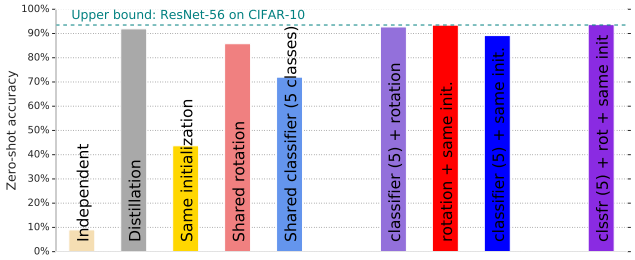


Figure 2: **Comparison of different alignment methods.** All proposed alignment schemes improve compatibility compared to independent training.

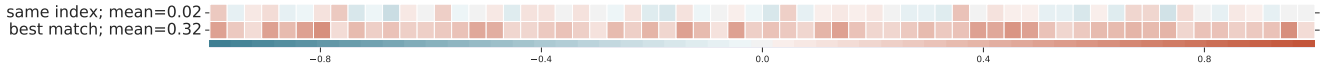
Joint *vs.* incremental training. All experiments so far were in a joint training regime. We also run some experiments incrementally: we first train one task, and freeze the feature extractor (for distillation) or the auxiliary task head (for discriminating common classes (Sec. 2.2) and self-supervised rotation prediction (Sec. 2.3)) Afterwards we train the second task. We found no significant difference between the two training regimes. This shows that we can train new components to be compatible with existing ones.

Where to align. We split the network into the feature extractor and the target task head at every possible layer. Depending on where we split, the correlation of the features of the two networks differs (Fig. 4). In particular, we find that (i) aligning on the last few layers before the classification layer performs poorly; (ii) aligning after the first block of a stage leads to the the strongest alignment. This makes intuitive sense, as the first block of a stage not only learns the residual, but also linearly projects the features to match the dimensionality [3]. Thus, this block has more parameters and has the largest effect on the features, while the other blocks are truly residual and not as crucial [19].

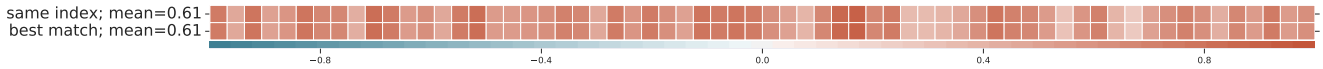
Number of shared classes (Sec. 2.2). We tried sharing 3, 5, and 10 classes. Accuracy of swapped classifier heads was respectively 24.1%, 72.0%, and 92.3%. This suggests that this strategy only works when tasks share many classes.

4. Conclusion

In this paper we demonstrated that we can train networks to produce compatible features. We observed that the degree of compatibility highly depends on where we split the network. Furthermore, self-supervised rotation is the most general method to achieve compatibility and works extremely well. Additionally, we found that initializing components with the same random weights has a large positive effect on compatibility. Finally, through the incremental training regime we demonstrated that we can train new components to be compatible with existing ones.



(a) Independent training.



(b) Alignment via rotation task head.

Figure 3: **Correlation of features with and without our self-supervised alignment method.** We compare correlation of features with the same index, *i.e.* assuming the features are aligned, and after finding the best alignment via bi-parte graph matching.

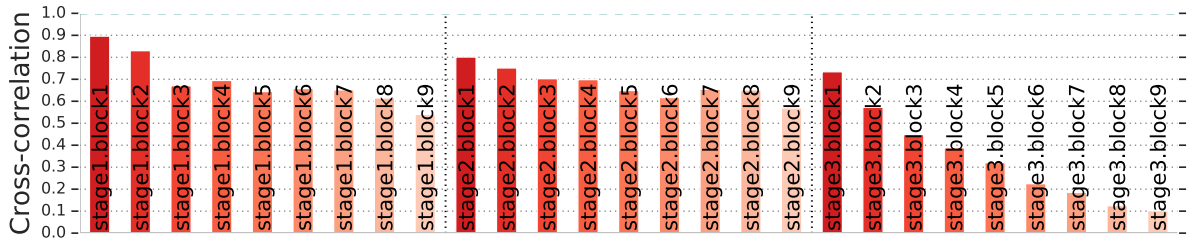


Figure 4: **Effects of alignment layer.** We plot cross-correlation when splitting the network directly after the layer named in this barplot. Depending on what layer is used for alignment, the correlation of the features of the two networks differs. In particular, aligning after critical layers [19] leads to the the strongest alignment (see text).

References

[1] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. In *ICLR*, 2018. 1, 2

[2] Saurabh Gupta, Judy Hoffman, and Jitendra Malik. Cross modal distillation for supervision transfer. In *CVPR*, 2016. 2, 3

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 3

[4] Olivier J Hénaff, Ali Razavi, Carl Doersch, SM Eslami, and Aaron van den Oord. Data-efficient image recognition with contrastive predictive coding. *arXiv preprint arXiv:1905.09272*, 2019. 2

[5] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 1, 2

[6] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proc. Nat. Acad. Sci. USA*, 2017. 1

[7] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *ICML*, 2019. 1

[8] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 1, 3

[9] Karel Lenc and Andrea Vedaldi. Understanding Image Representations by Measuring Their Equivariance and Equivalence. *IJCV*, 2015. 1

[10] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John E Hopcroft. Convergent learning: Do different neural networks learn the same representations? In *ICLR*, 2016. 1, 3

[11] Qihong Lu, Po-Hsuan Chen, Jonathan W Pillow, Peter J Ramadge, Kenneth A Norman, and Uri Hasson. Shared representational geometry across neural networks. In *NeurIPS*, 2018. 1, 3

[12] Ari Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *NeurIPS*, 2018. 1

[13] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *ECCV*, 2016. 2

[14] Jong-Chyi Su, Subhransu Maji, and Bharath Hariharan. Boosting supervision with self-supervision for few-shot learning. *arXiv preprint arXiv:1906.07079*, 2019. 1

[15] Yu Sun, Eric Tzeng, Trevor Darrell, and Alexei A Efros. Unsupervised domain adaptation through self-supervision. *arXiv preprint arXiv:1909.11825*, 2019. 1, 2

[16] Joshua B. Tenenbaum, Charles Kemp, Thomas L. Griffiths, and Noah D. Goodman. How to grow a mind: Statistics, structure, and abstraction. *science*, 2011. 1

[17] Liwei Wang, Lunjia Hu, Jiayuan Gu, Zhiqiang Hu, Yue Wu, Kun He, and John Hopcroft. Towards understanding learning representations: To what extent do different neural networks learn the same representation. In *NeurIPS*, 2018. 1

[18] Xiaohua Zhai, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer. S4L: Self-Supervised Semi-Supervised Learning. In *ICCV*, 2019. 1, 2

[19] Chiyuan Zhang, Samy Bengio, and Yoram Singer. Are all layers created equal? In *ICML Workshop Deep Phenomena*, 2019. 2, 3, 4