Supplementary Material for "Feature Embedding by Template Matching as a ResNet Block"

Ada Görgün
ada.gorgun@metu.edu.tr
Yeti Z. Gürbüz
yeti@metu.edu.tr
A. Aydın Alatan
alatan@metu.edu.tr

Dept. of Electrical and Electronics Eng.
& Center for Image Analysis (OGAM)
Middle East Technical University
Ankara, Turkey

1 Architectural Details

Table 1: Summary of the architectural choices including the input (in) and the output (out) feature dimensions, and the spatial reduction (reduction) of each stage for Cifar.

	stage-1			stage-2			stage-3			Our Block			stage-4		
Architectures ↓	in	out	reduction	in	out	reduction	in	out	reduction	in	out	reduction	in	out	reduction
RN26	16	64	↓ 1	64	128	↓ 2	128	128	↓ 2	-	-	-	128	256	↓ 1
RN26-Ours	16	64	↓ 1	64	128	↓ 2	128	128	↓ 2	128	128	↓ 1	128	256	↓ 1
RN38	16	64	↓ 1	64	128	↓ 2	128	128	↓ 2	-	-	-	128	256	↓ 1
RN38-Ours	16	64	↓ 1	64	128	↓ 2	128	128	↓ 2	128	128	↓ 1	128	256	↓ 1
WRN16	16	32	↓ 1	32	64	↓ 2	64	128	↓ 2	-	-	-	128	128	-
WRN16-Ours	16	32	↓ 1	32	64	↓ 2	64	128	↓ 2	128	128	↓ 1	128	128	-
DN100	24	108	↓ 2	108	150	↓ 2	150	342	-	-	-	-	342	534	-
DN100-Ours	24	108	↓ 2	108	150	↓ 2	150	342	-	342	342	↓ 1	342	534	-
DN100-Ours-C	24	108	↓ 2	108	150	↓ 2	150	342	-	342	534	-	534	726	-

Table 2: Summary of the architectural choices including the input (in) and the output (out) feature dimensions, and the spatial reduction (reduction) of each stage for Mini-Imagenet.

	stage-1			stage-2			stage-3			Our Block			stage-4		
Architectures ↓	in	out	reduction	in	out	reduction	in	out	reduction	in	out	reduction	in	out	reduction
RN26	16	64	↓ 2	64	128	↓ 2	128	128	↓ 2	-	-	-	128	256	↓ 1
RN26-Ours	16	64	↓ 2	64	128	↓ 2	128	128	↓ 2	128	128	↓ 1	128	256	↓ 1
RN38	16	64	↓ 2	64	128	↓ 2	128	128	↓ 2	-	-	-	128	256	↓ 1
RN38-Ours	16	64	↓ 2	64	128	↓ 2	128	128	↓ 2	128	128	↓ 1	128	256	↓ 1
WRN16	16	32	↓ 2	32	64	↓ 2	64	128	↓ 2	-	-	-	128	128	-
WRN16-Ours	16	32	↓ 2	32	64	↓ 2	64	128	↓ 2	128	128	↓ 1	128	128	-
DN100	24	108	↓ 2	108	150	↓ 2	150	171	↓ 2	-	-	-	171	363	-
DN100-Ours	24	108	↓ 2	108	150	↓ 2	150	171	↓ 2	171	171	↓ 1	171	363	-
DN100-Ours-C	24	108	↓ 2	108	150	↓ 2	150	171	↓ 2	171	363	-	363	555	-

We provide details of the architectural choices for the baseline methods for the sake of reproducibility of our experimental work. We use ResNet (RN) [1], Wide-ResNet (WRN) [3], and DenseNet (DN) [2] as the baseline architectures. We use 4 *stages* for each architecture. Note that the implementation of the *stage* differs from method to method as we will disclose shortly.

ResNet (RN). We stick to the original implementation of ResNet v2 [1] including the combination of convolution, batch normalization (BN) and ReLU layers at the start of the first stage. In our notation, a typical RN v2 *stage* includes multiple residual blocks which are called *bottleneck residual units*. The first block of each stage perform 1×1 convolution in the shortcut connection. For the stages that perform spatial reduction, the stride of that convolution is 2. We use an additional stage (stage-4) to incorporate our method easily during implementation. We perform experiments with two RN architectures with the number of blocks for each stage being 2 (RN26) and 3 (RN38), respectively. We summarize the architecture details in Tabs. 1 and 2 for Cifar [3] and Mini-Imagenet [1], respectively. $\downarrow k$ in reduction means we have 1×1 convolution with stride k in the shortcut connection before addition, and $-$ means direct shortcut connection. Only for Cifar 10, we find that using an additional 2×2 average pooling in the shortcut before the 1×1 convolution layer (*i.e.*, linear transform) better generalizes the incoming features. Moreover, we use the output of the BN as the input to the *soft-max* operation to shape the softness of the *soft-max* predictions. With that being said, one can use temperature scaling to logits instead. Yet BN performs such a scaling inherently since it provides us with scaled and normalized activations. Hence we do not have to choose the temperature manually. Such tricks bring marginal improvements to the performance in Cifar 10.

Wide-ResNet (WRN). We stick to the original implementation of WRN [3] including a single convolution layer at the start of the first stage. Similar to ResNet, a typical WRN *stage* includes multiple residual blocks which are called *basic residual architecture* in the original paper [3]. For the stages that perform spatial reduction, the first block includes 1×1 convolution with stride 2 in the shortcut connection. If the channel dimensions of the input and the output features are not the same for that stage, the first block again includes 1×1 convolution with stride 1 in the shortcut connection. We use an additional stage (stage-4) to incorporate our method easily during implementation. We use WRN of *depth* 16 and *widening factor* 2. Namely, the depth of the each stage is computed so that the total depth is 16. We do not use *dropout*. We summarize the architecture details in Tabs. 1 and 2 for Cifar and Mini-Imagenet, respectively. $\downarrow k$ in reduction means we have 1×1 convolution with stride k in the shortcut connection before addition, and $-$ means direct shortcut connection. In our block, we use an extra BN before 4×4 average pooling owing to the slight architectural differences of WRN from RN (In fact, we are doing that to make the internal classification stage of the patches more similar to the final classification stage of the original network). Only for Cifar 10, we use the same implementation tricks as in RN.

DenseNet (DN). We stick to the original implementation of DN-BC [2] including a single convolution layer at the start of the first stage. In the context of DN, a typical *stage* includes a multiple-layered *dense block* [2], and a *transition layer* [2] if reduction is specified. We use *bottleneck implementation* [2] in dense blocks with 0.5 *compression factor* [2] at the transition layers since we are using DN-BC. The *compression factor* reduces the channel dimension by the specified factor. We use an additional stage

(stage-4) to incorporate our method easily during implementation. We use DN of *depth* 100 and *growth rate* 12. Namely, the depth of the each stage is computed so that the total depth is 100. For our method, we additionally employ concatenation of the embedded feature and the input feature instead of addition through shortcut to align with the architectural design of DN, which is referred as DN100-Ours-C. For DN100-Ours-C, we find that using value vectors of the half dimension of the input gives good results. Thus, we use 192 dimensional *value* vectors (*i.e.*, 192-many 1×1 convolutions for embedding) and concatenate them with the corresponding input. Aligned with the baseline architecture, we do not use 1×1 convolution in the shortcut connection. We summarize the architecture details in Tabs. 1 and 2 for Cifar and Mini-Imagenet, respectively. $\downarrow 2$ in reduction means we have the *transition layer* in between DN stages, $\downarrow 1$ means we have 1×1 convolution in our shortcut connection, and $-$ means direct connection without any convolution. Similar to WRN, we use an extra BN before 4×4 average pooling in our block. Only for Cifar 10, we use the same implementation tricks as in RN and WRN except that we do not use 2×2 average pooling in the shortcut since it results in over smoothing considering the transition layers also inheriting 2×2 average pooling.

2 Magnified Figures and Discussion

We provide magnified visualizations of class predictions and embedding vectors of patches in Figs. 2 and 3, a summary of which is already included in the main paper. Specifically, we generate a sprite image of the patches, where each patch is embedded with respect to its class prediction vector (Fig. 2) or embedding vector as the convex combination of class embedding, *i.e.*, *value*, vectors (Fig. 3). We enhance patch images with further visual aids as illustrated in Fig. 1, in which the color in the frame represents the true class, the colored box in the left corner represents the final predicted class coming from the classifier and the grayish-filled box in the middle represents the entropy calculated from the *soft-max* predictions of our block for each patch image extracted. We especially use entropy calculation to understand how peaky or how uniform the *soft-max* predictions are to further interpret the results. The color of the entropy box goes darker as the entropy goes lower and vice versa.

Showcasing the patch image convention, we include two examples in Fig. 1 with their corresponding histograms obtained from the *soft-max* predictions of our block, which are also used in the entropy calculation. These examples consist of one patch image having a *dog face* and the other one having an *animal body*. Taking the *dog* image for instance, our *soft-max* predictions (histogram) have a peak at *dog* class and the final classifier (the box in the left corner) as well predicts the class *dog*, which can be seen from its color. This color also matches with the frame color, indicating that we make the correct assignment for the image from which the patch is extracted. Moreover, since the histogram is very peaky, we have a very low entropy. Hence, we have a darker colored box in the middle as expected. As an example of another case, we predict the wrong class in the final classifier for the image from which the patch with an *animal body* is extracted. Note that in that case the color of the frame and the box in the left corner do not match. We also have a relatively higher entropy which is indicated by the brightness of the middle box. We indeed expect such kind of results for the patches having semantic entities which are shared among the classes. That

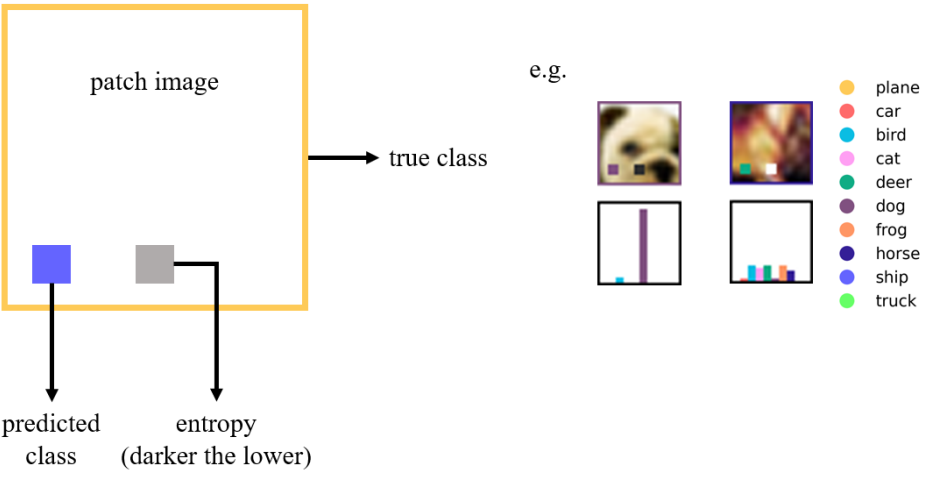

Figure 1: The convention used in the visualization of 2D t-SNE projections of the features.

being said, we see from the corresponding histogram that the non-zero histogram bins only come from *animal* classes as expected. Due to the structure of the body and the combination of the other corresponding patch images at the final classification stage, the final prediction becomes the class *deer* instead of *horse*, which are close species in nature.

For Fig. 3, we additionally embed the class value vectors as the images filled with solid colors corresponding to classes. We also embed 0-*vector* as a black filled image. We observe that value vectors can be considered as the vertices of the convex hull of the embedded features. Hence, their convex combination creates the corresponding embedding vectors.

Once we look at the origin (black box) by zooming in the image, we see that the patch images nearby have larger entropy compared to the ones away from the 0-*vector*. In other words, the patches of high entropy are assigned to 0-*vector*. Such behavior is not surprising since we believe the patches of high entropy (*i.e.*, shared among many classes) should not carry too much information. These patches generally include shared nuisance information than discriminative patterns such as *beak*, *ear* and *wing*. Similarly, we observe relatively higher entropy of the predictions in the transition between classes such as *bird* and *plane*, *car* and *truck* or *plane* and *ship*. These passage points represent mutual semantic entities for those classes, such as *wing* for *bird* and *plane*, *tire* for *car* and *truck* or *blue background* for *plane* and *ship*, yet another supporting result for our claims on combining class labels to generate novel labels corresponding to different semantic entities. For the discriminative entities (*i.e.*, the ones nearby the class value vectors), the distinction between the classes are more clear, resulting in smaller entropy.

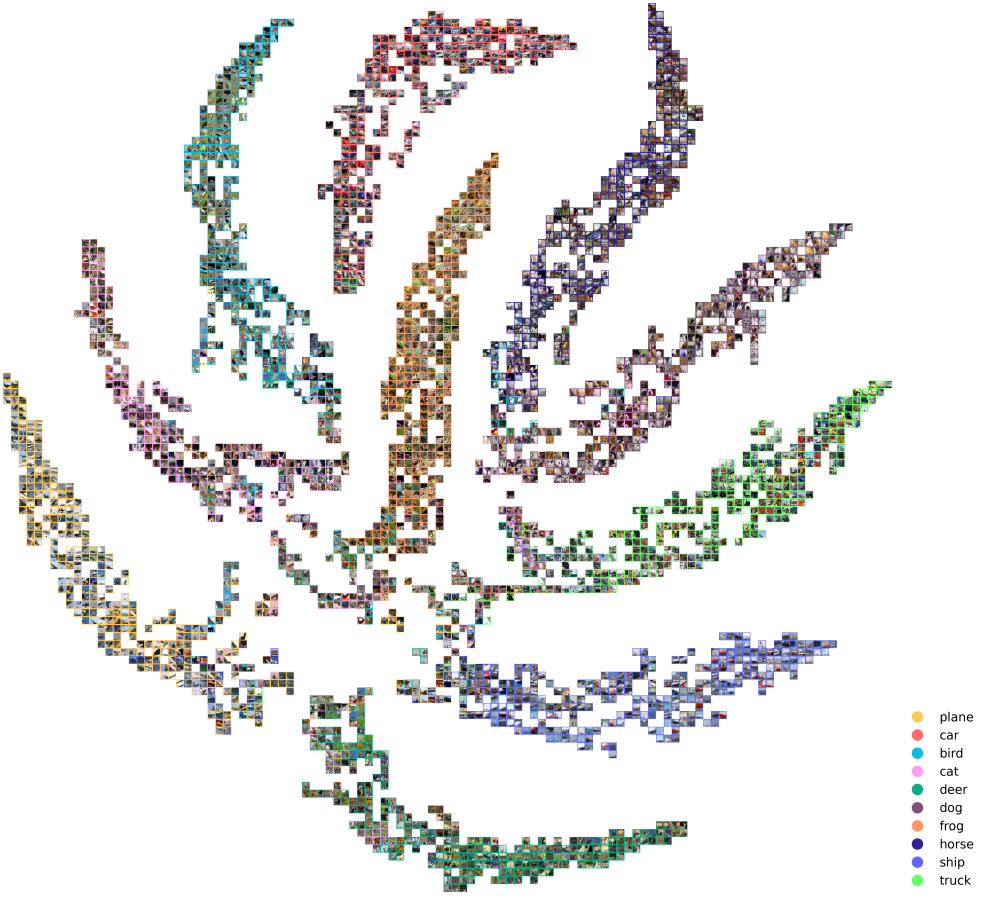

Figure 2: Patches embedded by 2D t-SNE with respect to their class predictions.

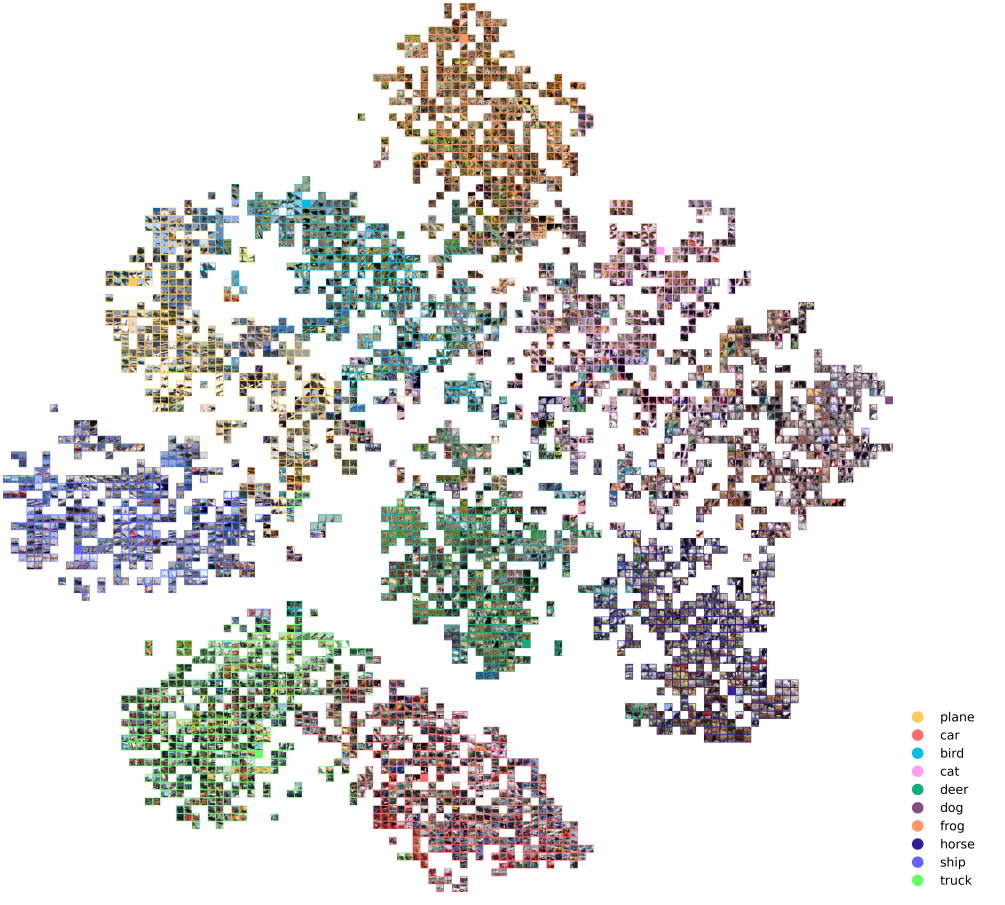

Figure 3: Patches embedded by 2D t-SNE with respect to convex combination of class embedding vectors.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, 2016.

[2] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *the IEEE conference on computer vision and pattern recognition*, 2017.

[3] Alex Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.

[4] Sachin Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017.

[5] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference 2016*. British Machine Vision Association, 2016.