# VampNet : Vampirisation non supervisée de réseaux de convolution

Trong-Lanh Nguyen, Thierry Chateau, Guillaume Magniez

# VampNet : Unsupervised Vampirizing of Convolutional Networks

Trong-Lanh R. Nguyen[1,2] Thierry Chateau[2] Guillaume Magniez[1]

[1] Safran Electronics & Defence
[2] Université Clermont Auvergne,
CNRS, SIGMA Clermont, Institut Pascal

trong-lanh.nguyen@safrangroup.com

## Résumé

*Nombre d'applications demandent la résolution simultanée de plusieurs tâches. Nous proposons une méthode non supervisée qui permet à partir de deux réseaux convolutifs profonds A et B de créer un réseau B' approximant B en s'alimentant d'une partie des couches de A. Ce réseau, nommé Vampire Network, permet de fortement réduire le poids combiné des deux réseaux. Nous proposons ces contributions : (1) nous montrons que des réseaux de même architecture entraînés à des tâches distinctes ont des propriétés de linéarité assez fortes entre couches; (2) Un algorithme non supervisé, remplaçant des cartes de caractéristiques du réseau vampire par une projection linéaire des cartes du premier réseau; (3) Nous montrons que le réseau vampire généré réduit fortement le nombre de paramètres, donc les calculs du système global.*

## Mots Clef

Réseau de neurones, Compression, multitâche.

## Abstract

*Numerous applications need to concurrently solve multiple tasks. We present an unsupervised method enabling to create from two pre-trained neural networks A and B, a third network B' approximating B while feeding on a part of A's layers. This network, that we call Vampire Network, allows to significantly reduce the combined weight of the two networks. To these ends, we propose the following contributions: (1) we show that two networks of the same structure but trained on different tasks display quite strong linear properties between their layers; (2) an unsupervised algorithm replacing part of the vampire network's features by linear projections of features from the first network; (3) we show that the vampire network thereby created significantly reduces the number of additional parameters needed to accomplish the second task, and thus the computational load of the full system.*

## Keywords

Neural network, Compression, multitask.

# 1 Introduction

Deep Convolutional Neural Networks (DCNN) are widely used for different tasks such as detection, semantic segmentation or depth estimation. Since some applications like autonomous driving need to combine theses different outputs, Multi-task models seem to be a relevant solution. It consists in generating networks performing several related tasks at once on the same input, effectively sharing resources between tasks [2]. One major convenience of such methods is the inductive bias that arises during the training step, that allows one task to benefit from the training of others, both raising the convergence speed and the generalisation abilities of the trained network [2]. Another interesting property for embedded applications is the reduction of the model size due to some shared parts. However, most of proposed approaches assume that all tasks must be trained jointly with available annotated data-sets. Moreover, adding a new task without modifying the performances of existing ones is also a important feature from an industrial point of view.

We propose VampNet: 1) Given two DCNN networks $\mathcal{N}_A$ (that we will call Master network) and $\mathcal{N}_B$, already trained on two different but related task $A$ and $B$, we formulate a framework to build a new network $\mathcal{N}_{B'}$ that approximates $\mathcal{N}_B$ under the challenging hypothesis:

- no annotated learning base is available,
- the master network $\mathcal{N}_A$ must not be modified,
- the model size of the new network $\mathcal{N}_{B'}$ must be lower than $\mathcal{N}_B$

Since both networks estimate related tasks, they should be correlated. We study this assumption and show that a very simple linear relation can be applied to replace features in $\mathcal{N}_B$ by features from $\mathcal{N}_A$. The new generated network $\mathcal{N}_{B'}$ is then called VampNet (from Vampire): it saves some computation by using some simple linear projections of $\mathcal{N}_A$ features (so-called *vampirizing* thereafter) resulting in a strong reduction of the size of the two merged networks. Figure 1 presents an overview of VampNet.

When using a classical DCNN $\mathcal{N}_B$ (*i.e.*: without skip connections) replacing the full feature map of a layer has as consequence that the preceding layers of the network do not have to be computed anymore. This implies that the deeper a full layer is replaced, the lower the resulting global network size is. This is why we focus thereafter on vampirizing a layer (*i.e.*: the full feature map of the layer).

We demonstrate our method on several public data-sets for two related tasks: semantic segmentation and depth estimation. Moreover, we provide a thorough ablation study to analyse linear correlation between layers and the proposed model that select the vampirized layer.

The next section presents some relevant works linked to multitask learning, network merging and correlation based feature analysis. Section three describes the core of our model while section four shows and analyses the experiments provided in order to evaluate the VampNet framework.

In this paper we use the following nomenclature for convolutional neural networks:

- a **feature** extracted by a convolutional layer is a single channel of its output volume. Each element of a channel is then a **sample** of the feature, as it is the result of a dot product with the convolution kernel for a different patch of the input volume.

- a **feature-map** is the set of features computed by a convolutional layer.

# 2 Related work

In order to assess the proposed VampNet model with the wide literature, we consider three aspects: 1) correlation based feature analysis, 2) neural network merging and 3), multitask neural networks.

## 2.1 Correlation-based feature map analysis

Since the proposed method relies on the assumption that there are linear links between features of two correlated task networks, we first review relevant works dealing with linear analysis for neural networks.

[7] uses Canonical Cross Correlation (CCA) between feature maps to compare learned representations. By comparing in-training feature maps to their fully trained version, the authors are able to study the training dynamics of a network. They explore the application of CCA to model compression.

[13] notes that multiple trainings of the same network starting from different random initial states usually converge toward solutions with similar performances, and that learned feature maps of a same layer often correlate with each other across the solutions. The authors show that it is possible to find a one-to-one, then a few-to-one mapping between features of the same layer of two versions of the same network, using activations' correlation as distance metric.

## 2.2 Multitask neural networks

Multitask learning [2, 14] encompasses learning methods aiming to accomplish multiple tasks at the same time. The main interest of this kind of approach is inductive bias: by learning two different but related tasks, more meaningful features are trained. Each task can thus benefit from features that would not have appeared with its sole training gradient.

Multitask neural network are the deep learning pendant of multitask learning. Two families stand out fairly distinctively [9]: approaches that have a "hard sharing" of weights, *i.e.* using a common body of computations, and approaches that have a "soft sharing" of weights, *i.e.* giving each task its own trainable weights but putting constraints between them.

## 2.3 Networks merging

While in multitask learning specific networks are trained to estimates several tasks, networks merging considers two existing networks which are mixed to produce a lightweight one. [12] introduces a post-training merging and compression method based on the convolution kernels' weights' values. The approach consists in a separation of kernels into $1 \times 1$ convolutions, a K-means clustering of those new kernels, followed by an Huffman encoding of the found centroids. A codebook can then be used to get back the full kernels. The clustering step has the effect that retrieved kernels are not exactly equal to the original ones. The authors suggest to make up for the changes in performances by fine tuning the model on the original training data.

[10] proposes a cascaded architecture to speed up classifiers' ability to discard negatives, replacing a monolithic network by a sequence of smaller classifiers called *stages*. Stages are of increasing abstraction level and size, the later ones only being computed if the earlier did not return a negative result. Because each subsequent stage must be of higher abstraction than the preceding one, building such an abstraction at each stage would induce a substantial amount of computing for examples that are not early rejected. To avoid that, the author gives each stage access to all the features extracted by the previous one, a stage only adding layers and/or channels to the preceding stage. All the stages are trained at the same time under a composite loss. As the sharing is unidirectional, the later stages do influence the convergence of the earlier ones but not *vice versa*.

In this paper, we propose a model that starts from several assumptions: 1) we have two existing trained networks like in network merging, 2) no annotated data-set is available like in unsupervised learning and 3) the function of the master network should not be changed.

# 3 Method

This section describes the core of the proposed model, relying on that correlated tasks trained using two networks with the same structure generate correlated Feature Maps (FM) within the two networks. After defining how to compute linearities between FMs, we propose a simple way, using a convolutional operator, to replace a feature by a linear projection of one vampirized from another task network. Since replacing the full FM of a layer is very interesting to save both computation time and model size, we propose a layer selection relation to automatically choose where replacing a layer while keeping a good trade-off between performances and computation budget.

## 3.1 Linearity between feature maps

Given two networks: $\mathcal{N}_A$ and $\mathcal{N}_B$ with the same structure but trained on two different tasks ($A$ and $B$), we are interested in replacing some features of $\mathcal{N}_B$ by linear projections of features of $\mathcal{N}_A$, without using any annotated data. This strategy, called VampNet ($\mathcal{N}_B$ acts like a vampire when it gets some already computed features of $\mathcal{N}_A$) is motivated by:

- The network $\mathcal{N}_A$ won't be modified: it can be mandatory in some industrial contexts (i.e. such network has already be certificated for task $A$).
- The resulting new $\mathcal{N}_B$ network will save computation time.
- We argue that if task $A$ and $B$ are correlated, The new $\mathcal{N}_B$ network will keep good performances.

Let $\mathbf{F}^{A,l}(\mathbf{X})$ be a 3D-tensor function returning the feature map associated to layer $l \in \{1,..,N_l\}$ of network $\mathcal{N}_A$ for the input tensor $\mathbf{X}$. Moreover, we define $f^{A,l}_{w,h,c}(\mathbf{X})$ a function returning the feature sample value for layer $l$, channel $c \in \{1,..,N^l_c\}$, and position $w \in \{1,..,N^l_w\}$, $h \in \{1,..,N^l_h\}$. $\boldsymbol{F}^{A,l}_c(\mathbf{X}) \doteq \mathbf{F}^{A,l}_{:,:,c}(\mathbf{X})$ is a function computing the 2D-slice feature matrix from channel $c$ of tensor $\mathbf{F}^{A,l}(\mathbf{X})$ and $\mathbf{f}^{A,l}_c(\mathbf{X}) \doteq \mathrm{vec}(\boldsymbol{F}^{A,l}_c)(\mathbf{X})$ the vectorization of $\boldsymbol{F}^{A,l}_c(\mathbf{X})$. The linear relation between the feature computed by channel $c$ of layer $l$ of network $\mathcal{N}_A$ and the feature computed by the channel $c'$ of the same layer of network $\mathcal{N}_B$ can be expressed by:

$$\mathbf{f}^{B,l}_c(\mathbf{X}) = \begin{bmatrix}\mathbf{f}^{A,l}_{c'}(\mathbf{X}) & 1\end{bmatrix}\mathbf{w} \tag{1}$$

with $\mathbf{w}$ a parameter vector of size 2.
Given a set of input images $\mathcal{X} \doteq \{\mathbf{X}_1, ...\mathbf{X}_{n_x}\}$, estimating $\mathbf{w}$ is given by the resolution of the following linear system:

$$\begin{bmatrix}\mathbf{f}^{B,l}_c(\mathbf{X}_1)\\ \vdots \\ \mathbf{f}^{B,l}_c(\mathbf{X}_i)\\ \vdots \\ \mathbf{f}^{B,l}_c(\mathbf{X}_{N_x})\end{bmatrix} = \begin{bmatrix}\mathbf{f}^{A,l}_{c'}(\mathbf{X}_1) & 1\\ \vdots & \vdots \\ \mathbf{f}^{A,l}_{c'}(\mathbf{X}_i) & 1\\ \vdots & \vdots \\ \mathbf{f}^{A,l}_{c'}(\mathbf{X}_{N_x}) & 1\end{bmatrix}\mathbf{w} \tag{2}$$

However, the number of equations of this linear system is huge ($N_x \times N^l_w \times N^l_h$) and solving if becomes too complex. We propose a sub-sampling strategy to reduce the number of equations using only a subset of all possible pixels of the features.
Let $s^l_w(i)$ and $s^l_h(i)$ be two sub-sampling functions providing width and height indexes of a feature of layer $l$ for $i \in \{1,..,N^l_s\}$ and with $N^l_s << N^l_w \times N^l_h$. Sub-sampling vectors associated to the network $\mathcal{N}_.$ can be defined as:

$$\hat{\mathbf{f}}^{.,l}_c(\mathbf{X}) \doteq \|_{i=1}^{N^l_s} \mathbf{F}^{.,l}_{s^l_w(i),s^l_h(i),c}(\mathbf{X}) \tag{3}$$

with $\|$ the concatenation operator. The linear equation 2 can be approximated from a new one changing full vectors to sub-sampled ones. Several sub-sampling strategies can be defined.

## 3.2 Ranking linearity between features

When data are standardized (zero-mean and unit-std), the residue between the linear prediction and the set of target values is a simple way to estimate linearity. We define $\tilde{\mathbf{f}}^{.,l}_c(\mathbf{X})$ returning the standardized sub-sampled vector by applying $\tilde{\mathbf{f}}^{.,l}_c(\mathbf{X}) = \frac{1}{\sigma_{\hat{\mathbf{f}}^{.,l}_c}}.(\hat{\mathbf{f}}^{.,l}_c(\mathbf{X}) - \overline{\hat{\mathbf{f}}^{.,l}_c})$ with $\sigma_{\hat{\mathbf{f}}^{.,l}_c}$ the standard deviation ($std$) and $\overline{\hat{\mathbf{f}}^{.,l}_c}$ the mean of $\hat{\mathbf{f}}^{.,l}_c(\mathbf{X})$ over $\mathbf{X}$, and compute the residue by:

$$r^l_{c,c'} = \frac{1}{N_x}\sum_{\mathbf{X}\in\{\mathcal{X}\}} \|\tilde{\mathbf{f}}^{B,l}_c(\mathbf{X}) - [\tilde{\mathbf{f}}^{A,l}_{c'}(\mathbf{X}), 1]\tilde{\mathbf{w}}\|_2 \tag{4}$$

The residue provides a natural way to predict if two features are correlated. We propose to compute a feature-wise residue matrix between the same layer of networks $\mathcal{N}_A$ and $\mathcal{N}_B$.

## 3.3 Vampirizing a feature using a convolutional operator

The simple model we propose to replace a feature is two-steps:

**Selection of the closest feature in $\mathcal{N}_A$** given the feature $\mathbf{F}^{B,l}_c$ of channel $c$ of layer $l$ of network $\mathcal{N}_B$, we define the association function providing the closest feature's channel for the same layer of $\mathcal{N}_A$ by:

$$t^l_c(c) \doteq \underset{c'\in\{1,..,N^l_c\}}{\mathrm{argmin}}\ r^l_{c,c'} \tag{5}$$

**Replace by convolution** The vampire network $\mathcal{N}_B$ replaces one of its features by the selected one of $\mathcal{N}_A$ applying a linear projection. It can be done very simply using a biased convolutional 1x1 kernel. Given a linear relation estimated by $\mathbf{w} = [a, b]^T$ between $\mathbf{f}^{B,l}_c$ and $\mathbf{f}^{A,l}_{s^l_c(c)}$, the associated feature of network $\mathcal{N}_B$ can be replaced by:

$$\boldsymbol{F}^{B,l}_c = \boldsymbol{F}^{A,l}_{t^l_c(c)} * \boldsymbol{K} \tag{6}$$

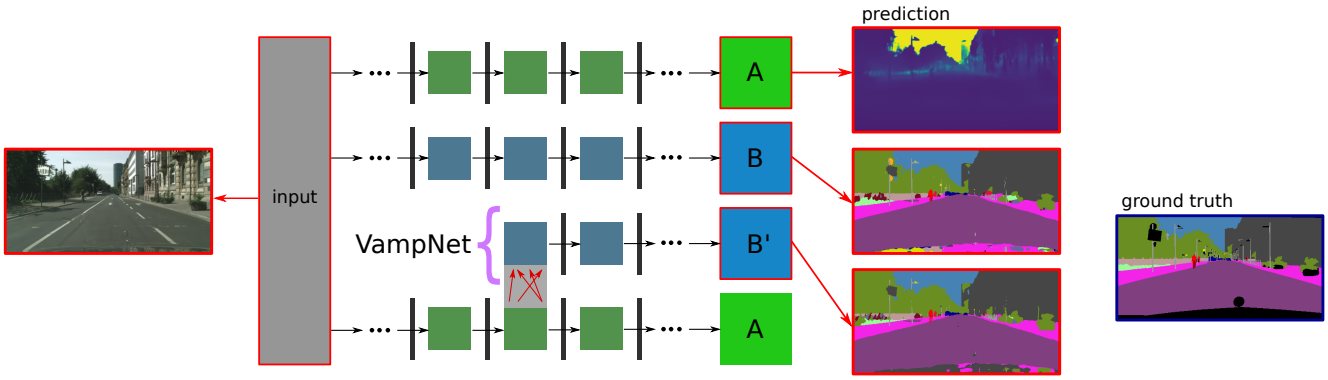


Figure 1: Overview of the method: computations are spared by reusing results from a related network. In a first time, features of two networks $\mathcal{N}_A$ and $\mathcal{N}_B$ are compared with each other by linear regression. Features from $\mathcal{N}_A$ with the smallest residuals can then be used to predict features from $\mathcal{N}_B$ by linear projection, thus creating the network $\mathcal{N}'_B$, approximating $\mathcal{N}_B$. In this example, the displayed results are from fully replacing UNet layer 11.

with $\boldsymbol{K}$ a kernel of size $1 \times 1 \times (N_c + 1)$ defined by:

$$\boldsymbol{K} \doteq \left[ \|_{i=1}^{N_c} a.\delta_i^{t_c^l(c)} \parallel b \right] \tag{7}$$

with $\delta$ the Kronecker function and $\|$ the concatenation operator. The bias $b$ is provided by a virtual last channel with unit values.

### 3.4 Vampirizing a layer

Replacing a layer is very important in order to save high computational cost. When VampNet replaces a full layer, it does not have to compute the layers before it anymore. Vampirizing a layer is achieved by replacing all of its features. The simple way to do that is by the strategy presented in the previous subsection.

### 3.5 Automatic selection of the layer to be replaced

VampNet should produce a new model that approximates the original $\mathcal{N}_B$ net with a lower inference computation cost. Choosing the layer to be replaced is very important. The Deeper this layer is, the higher the computation gain will be. However, we expect that the correlation decreases along the layers. Since the new network must approximate an existing one for a given task, we propose to define a layer-to-vampirize selection function with two terms:

**Computation budget loss** Let $C_l(\mathcal{N}_B)$ the computation cost of the new network when replacing layer $l$ of network $\mathcal{N}_B$ and $C_0(\mathcal{N}_B)$ the cost of $\mathcal{N}_B$ without any replacement. We propose to define a loss function by:

$$\mathcal{L}_C(C_l(\mathcal{N}_B)) \doteq \frac{C_0(\mathcal{N}_B) - C_l(\mathcal{N}_B)}{C_0(\mathcal{N}_B)} \tag{8}$$

**Accuracy loss** The new network should provide good performances while using a large number of features coming from a network trained for another task. Like in knowledge distillation, we consider the output of $\mathcal{N}_B$ as annotations that should be estimated by $\mathcal{N}'_B$. We then propose to estimate the accuracy between the two networks according to the layer to be replaced with a typical metric for the targeted task: $\mathcal{L}_A$. In the semantic segmentation case, we choose to use the *mean Intersection over Union* metric (mIoU).

We propose a layer-to-vampirize selection function that combines both the accuracy and the computation budget terms:

$$\hat{l_v} = \underset{l \in \{1,..,N_l\}}{\operatorname{argmin}} \ \lambda \mathcal{L}_C(C_l(\mathcal{N}_B)) + (1-\lambda)\mathcal{L}_A(\mathcal{N}'^{l}_{B}) \tag{9}$$

## 4 Experiments

### 4.1 Setup

**Networks and Tasks.** Since the proposed model applies on convolution layers, we study it on fully convolutional networks with encoders and decoders like the ones used for segmentation or depth estimation tasks. We use the UNet network as it is a fairly simple network with such an encoding/decoding structure. However, the presence of skip-connections (transmission of features from early layers to later non-adjacent ones) breaks the assumption that when replacing a layer, the previous ones don't have to be computed anymore. We will compare the impact of such connections into the network structure by comparing UNet with a degraded version of it without skip-connections that we will call Encode/Decoder-like (or ED-like). Figure 3 illustrates The evolution of VampNet model size related to the vampirized layer with (UNet) and without (ED-like) skip-connections. On the left figure, the model size (y-axis) is computed as a ratio related to the original network size. The middle and right figures illustrate the layers that do not need to be computed (within the overlay areas) if we choose to replace the layer 14 (red line in the left figure). In this case, the new model size would be about $1\%$ of the original for both networks (that value can also be read in tables 1 and 2). When using skip connections, some layers before the replaced one still have to be computed while

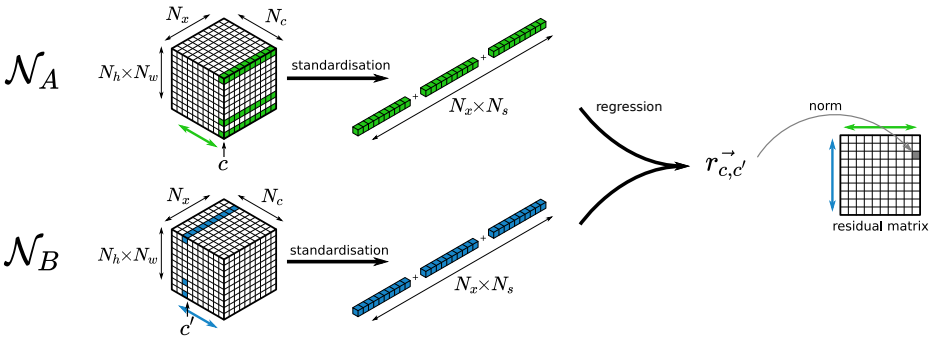


Figure 2: Sampling Method : in a given layer, for a pair of features $c$ from network $\mathcal{N}_A$ and $c'$ from network $\mathcal{N}_B$, we take a same random set of $N_s$ pixels in the volumes extracted by both networks. $N_h$ and $N_w$ are the spacial dimensions, $N_x$ is the data-set's size, and $N_c$ is the number of features.

all preceding layers can be forgotten in the case of ED-like networks (no skip connections). In the case of UNet and similar architectures however, the first and last layers do not contain much parameters in comparison to the central ones which have a lot more channels, and computing them does not cost a lot, it is visible as the low variation rate on both sides of the left figure. The difference between the two networks' model sizes is drawn in green on the left figure.

Two tasks have been selected for experiments: 1) Depth estimation that consists in estimating a dense depth map from a monocular image [1, 4] and 2) semantic segmentation that associates a semantic class to each pixel of an input image [6, 8]. These two tasks are known to be quite related [15]. Depth estimation is selected as the task to be vampirized ($A$) and semantic segmentation as the task to be approximated ($B$). Evaluation of the performance of the semantics segmentation task is achieved using the classical Mean Intersection Over Union criteria (mIoU) on a testing data-set.

**Data-sets and Implementation details.** Different data-sets have been used to train depth estimation network (called $\mathcal{N}_D$) and the semantic segmentation network (called $\mathcal{N}_S$). $\mathcal{N}_D$ was trained using ApolloScape [11] (sequences from road n°3 for training and sequences from road n°2 for validation) while $\mathcal{N}_S$ was trained using Cityscapes [3] (2975 images). Related to Semantic segmentation, we use the 19 default Cityscapes training classes.

The implementation we use was made in the PyTorch framework using a single GPU. For sub-sampling during the linear analysis, we chose a selection function that get a constant spatial coverage such as each sample covers $1/16^{th}$ of the features For example, for input images of resolution $256 \times 256$, the first layer's output is also $256 \times 256$ that is $65536$ pixels, we sample $4096$ of them. For the middle layers, the spatial resolution drops to $16 \times 16$ that is $256$ pixels and we sample $16$ of them. This is one of the possible strategies that provides a computational solution. Moreover, the analysis is achieved on as set of $1000$ images.

## 4.2 Linearity

**Feature-Based Linearity** Since the main hypothesis of VampNet is that a task B feature can be replaced by a linear projection of a task A feature, we first study the loss of accuracy according to the number of replaced filters in a layer. Figure 4 shows such evolution for several layers (3, 7, 11 and 15) for both ED-like and UNet networks. We define the loss of accuracy as the mIoU degradation: a degradation of $0\%$means that the VampNet version of the segmentation network outputs the exact same segmentation maps as the original network. This figure shows that the more features are approximated, the more the network loses in accuracy, which is not surprising. However it also appears that it does not increase according to the depth. It means that two networks with correlated tasks share linear information into deep layers: for example, replacing features in layer 11 yields better results than replacing features in layer 15 (for both tested networks). It is counter-intuitive because we expected that features grow in abstraction levels and should become more specific to the task [5].

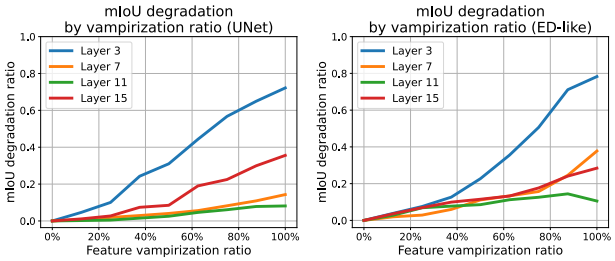


Figure 4: mIoU degradation depending on feature replacement ratio at varying depths, on the left for UNet and on the right for ED-like.

**Layer-Based Linearity** Since the best strategy to reduce the computation budget is to replace all the features of a layer, the next experiment evaluates the loss of accuracy (mIoU) according to the replaced layer (See figure 5). Like in the previous experiment, we observe that mIoU degradation does not increase monotonically according to the

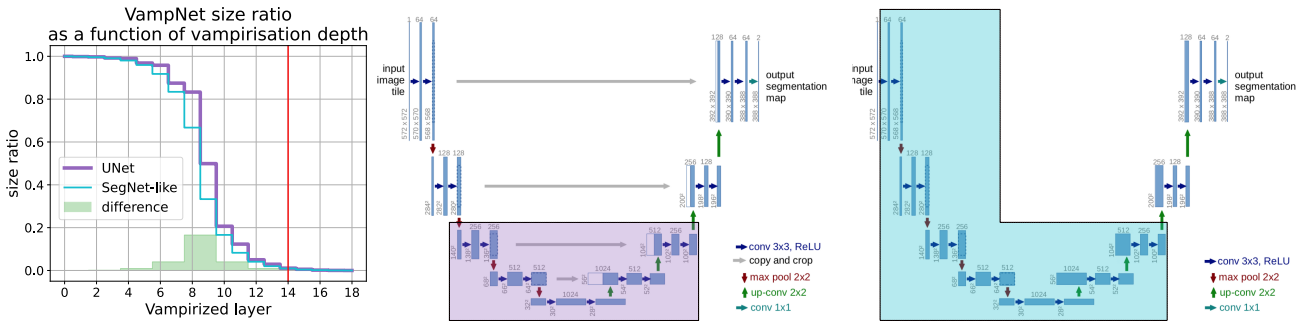


Figure 3: Visualisation of saved computation: on the left, the evolution of the saved network volume ; On the right the two studied architectures : UNet in the middle and ED-like on the right. The overlays are an example of what can be removed if we choose to vampirize layer 14, shown as a red line on the left graph. The x-axis units of the left graph correspond to blue arrows in the networks.

depth. The general shape of the curve outlines that layer-wise linearity seems to be better in the decoder (from layer 10).

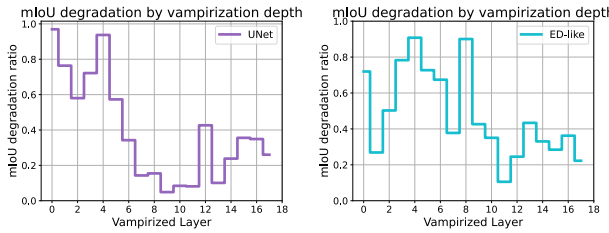


Figure 5: mIoU degradation depending on vampirization depth for full feature map replacement, on the left for UNet and on the right for ED-like.

### 4.3 Trade-off selection between the accuracy and computational budget

This section studies the couple accuracy, computation budget and the influence of the function that selects the layer to be replaced.

Tables 1 and 2 present the couple mIoU degradation and size ratio related to the replaced layer for the two networks. It confirms that replacing encoding layers does not provide good performances for both accuracy and computational budget. Regarding UNet, mIoU degradation is below $10\%$ for layers 9 to 11: the first decoding layers. The model size decreases along layers but we observe a high reduction between layers 9 and 11. This is directly linked to the network auto-encoder-like structure with many parameters near the embedded middle representation. The UNet variant without skip connection does not seem to follow the same variations and mostly presents a degradation of about $10\%$ at layer 11.

The layer to be replaced must be selected according to the desired trade-off between the accuracy and the computation budget. Adjusting this trade-off is achieved by a selection function that uses a hyper-parameter $\lambda$ described in section 3.5. Figure 6 studies the evolution of the accuracy

Table 1: mIoU degradation and size ratio for a given vampirization layer (UNet)

| layer | mIoU degradation | size ratio |
|---|---|---|
| 0 | 96.94% | 99.99% |
| 1 | 76.38% | 99.86% |
| 2 | 58.02% | 99.74% |
| 3 | 72.18% | 99.22% |
| 4 | 93.73% | 98.96% |
| 5 | 57.30% | 96.87% |
| 6 | 34.25% | 95.82% |
| 7 | 14.31% | 87.47% |
| 8 | 15.46% | 83.30% |
| 9 | 4.88% | 49.90% |
| 10 | 8.44% | 20.67% |
| 11 | 8.14% | 12.31% |
| 12 | 42.66% | 5.01% |
| 13 | 10.08% | 2.92% |
| 14 | 23.82% | 1.09% |
| 15 | 35.57% | 0.57% |
| 16 | 34.89% | 0.17% |
| 17 | 26.02% | 0.04% |

Table 2: mIoU degradation and size ratio for a given vampirization layer (ED-like)

| layer | mIoU degradation | size ratio |
|---|---|---|
| 0 | 71.97% | 99.99% |
| 1 | 26.87% | 99.86% |
| 2 | 50.27% | 99.60% |
| 3 | 78.27% | 99.08% |
| 4 | 90.77% | 98.03% |
| 5 | 72.70% | 95.95% |
| 6 | 67.40% | 91.77% |
| 7 | 37.75% | 83.42% |
| 8 | 90.04% | 66.72% |
| 9 | 42.63% | 33.31% |
| 10 | 35.03% | 16.61% |
| 11 | 10.55% | 8.26% |
| 12 | 24.53% | 4.09% |
| 13 | 43.36% | 2.00% |
| 14 | 33.04% | 0.95% |
| 15 | 28.43% | 0.43% |
| 16 | 36.23% | 0.17% |
| 17 | 22.26% | 0.04% |

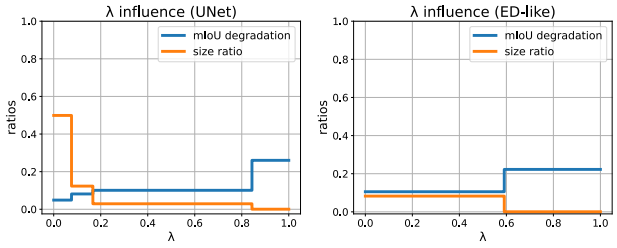


Figure 6: Evolution of the size ratio and mIoU degradation when varying $\lambda$ from 0 to 1, on the left for UNet and on the right for ED-like.

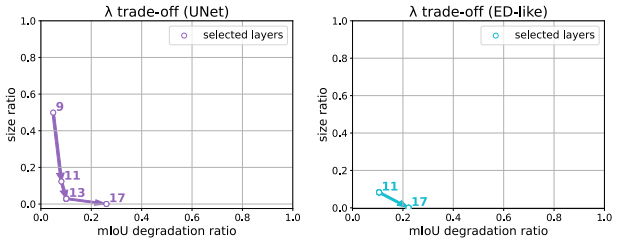


Figure 7: 2D parametric representation of the accuracy loss and model compression rate according to the selected layer to be replaced selected for $\lambda \in [0, 1]$, on the left for UNet and on the right for ED-like.

loss and model compression for $\lambda \in [0, 1]$. For small values of $\lambda$, the mIoU degradation is low while the model size ration is about $50\%$ for UNet and $8\%$ for its variant without skip connection. As $\lambda$ increases, the mIoU degradation also increases while the model size ratio decreases.

Figure 7 shows, according to selected layer to be replaced when $\lambda \in [0, 1]$, a 2D parametric representation of the accuracy loss and model compression rate. Since we want to minimize both the degradation and model size, the best layer to be replaced is the one that provide an accuracy loss / size ratio, near the origin. This graph confirms that layers 11 is a good candidate for both networks. It will be selected for $\lambda = 0.5$.

Figure 8 Shows some output examples from using VampNet on layer 11 of UNet for a semantic segmentation task with depth estimation as master network. The UNet original segmentation network has $28M$ parameters while it VampNet approximation reduces the model size to $3M$ ($12.31\%$ of the size, that is $87.69\%$ compression). Differences mainly occur on class boundaries and for small objects

# 5 Conclusion

We introduced vampire networks, an approach to reduce the cumulative size of two networks performing related tasks by replacing features of one of them by a linear projection of the features of the other, while leaving that last one's performances untouched. We explained our method of selecting which features to replace by analysing the linearity between them, and of computing the projection parameters, all of this in an unsupervised fashion. We showed that while replacing some features in a layer can somewhat reduce the needed resources, the true potential appears if we are able to replace a whole layer, in which case big portions of the vampire network can be discarded at once. We also showed how skip-connection can impede that alleviation. Our approach is oriented toward reducing the size of the vampire network; by approximating its features we overall reduce its accuracy, which is a problem that we do not solve here.

In future works we plan to explore several ideas: taking inspiration of what is done in [13], actually training the projection instead of computing it, potentially replacing it with a shallow neural network could prove interesting. We could also use a multiple regression instead of a singular one to predict features. The goal of our approach being to add new tasks in a cascading manner, experiments with more than two task, meaning multiple source networks, should also be done. A last idea would be to see how well this method works with networks of different architectures.

# References

[1] Ibraheem Alhashim, Peter Wonka, *High Quality Monocular Depth Estimation via Transfer Learning*, ArXiv, 2019.

[2] Rich Caruana, *Multitask Learning*, Machine Learning 28, 1997.

[3] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, *The Cityscapes Dataset for Semantic Urban Scene Understanding*, Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[4] Eigen, David and Puhrsch, Christian and Fergus, Rob, *Depth map prediction from a single image using a multi-scale deep network*, Advances in neural information processing systems, 2014.

[5] Kozma, Robert and Ilin, Roman and Siegelmann, Hava T, *Evolution of abstraction across layers in deep learning neural networks* Procedia computer science, 2018

[6] Long, Jonathan and Shelhamer, Evan and Darrell, Trevor, *Fully convolutional networks for semantic segmentation*, Proceedings of the IEEE conference on computer vision and pattern recognition, 2015.

[7] Maithra Raghu *et al.*, *SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability*, NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017.

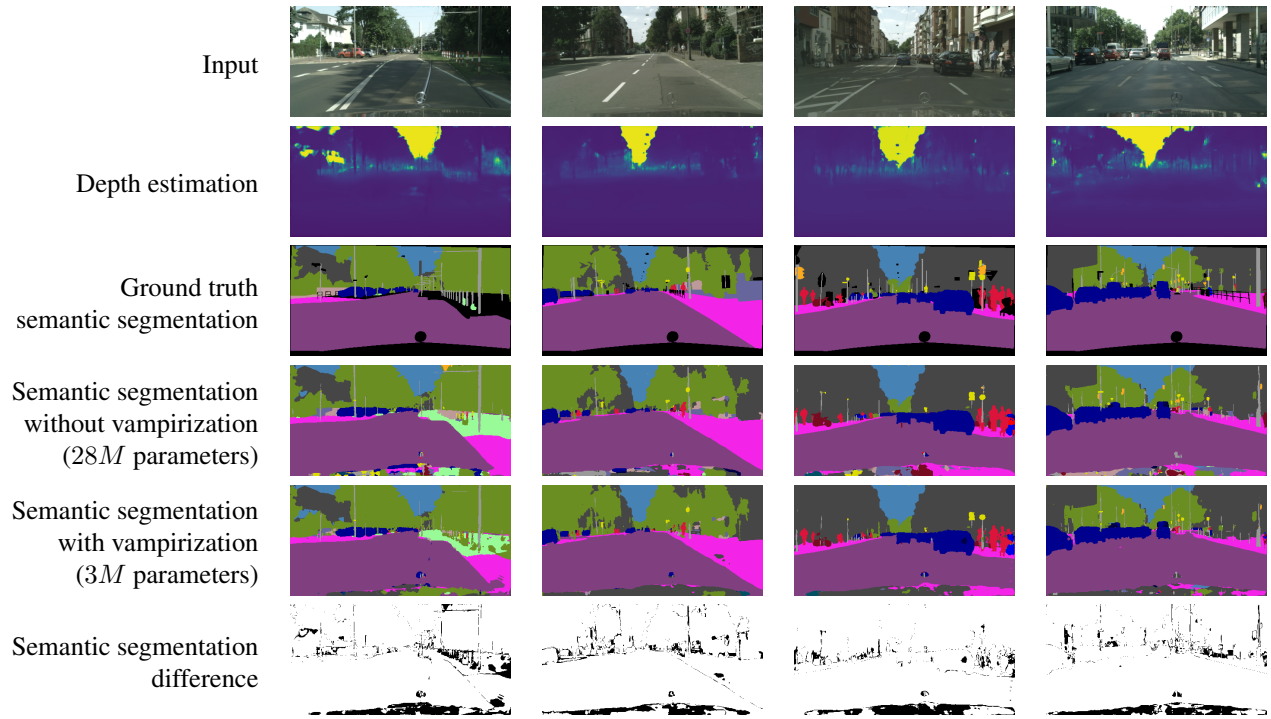


Figure 8: Examples of results for a vampirization of layer 11 of the UNet network, with from top to bottom: the input image, the output of $\mathcal{N}_A$, the ground-truth of task $B$, the output of $\mathcal{N}_B$, the output of $\mathcal{N}'_B$, and the error mask between $\mathcal{N}_B$ and $\mathcal{N}'_B$.


[8] Olaf Ronneberger *et al.*, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, MICCAI: Medical Image Computing and Computer-Assisted Intervention, 2015.

[9] Sebastian Ruder, *An Overview of Multi-Task Learning in Deep Neural Networks*, ArXiv, 2017.

[10] Martin Simonovsky *et al.*, *OnionNet: Sharing Features in Cascaded Deep Classifiers*, ArXiv, 2016.

[11] Xinyu Huang *et al.*, *The ApolloScape Open Dataset for Autonomous Driving and its Application*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2019.

[12] Yi-Min Chou *et al.*, *Unifying and Merging Well-trained Deep Neural Networks for Inference Stage*, IJCAI: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, 2018.

[13] Yixuan Li *et al.*, *Convergent Learning : Do different neural networks learn the same representations ?*, ICLR, 2016.

[14] Yu Lin Zhang *et al.*, *An overview of multi-task learning*, National Science Review, 2018.

[15] Amir R. Zamir *et al.*, *Taskonomy: Disentangling Task Transfer Learning*, CVPR: The IEEE Conference on Computer Vision and Pattern Recognition, 2018.