A. Supplementary Structure

In Section B of this supplementary material, we discuss more details about our proposed network's architecture and we provide a diagram with all it's details. In Section C we provide more details about some Task Palette rules. In Section D we provide additional visual results of our method. In Section E we provide more details on implementational details and hyperparameters used to conduct experiments. In Section F we provide further analysis, mostly in depth details about the hyperparameter choices. Finally, in Section G we discuss some limitations of our proposed method.

B. Architecture

The full CompositeTasking network diagram is presented in Figure 12. The encoder is the well known ResNet34 network. The choice for ResNet34 trades-off between performance and memory/computational demands. In the diagram, the encoder is divided into 5 blocks: B1,...,B5. Each block Bi is a part of the encoder that takes features of spatial size $(\frac{H}{2^{i-1}}, \frac{W}{2^{i-1}})$, and reduces the spatial size to $(\frac{H}{2^i}, \frac{W}{2^i})$. In the case of ResNet34, the spatial size is reduced with a strided convolution.

The Task Palette Embedding $\mathcal{E}(\mathcal{T})$ is calculated once for each forward pass, with the task representation block from Figure 12. Since all the task composition blocks use the same embedding $\mathcal{E}(\mathcal{T})$, a spatial pyramid of $\mathcal{E}(\mathcal{T})$ is created to be available in all spatial sizes $(\frac{H}{2^i}, \frac{W}{2^i})$, where $i = 0, 1, ..., 5$.

C. Task Palette Rules

Random mosaic rule $\mathcal{R}_{1r}$: The image is spatially divided into four rectangles by intersecting a vertical and horizontal line through a point $\mathbf{c} = (c_x, c_y)$. The point is chosen randomly as $c_x \sim U[\frac{W}{4}, 3\frac{W}{4}]$, $c_y \sim U[\frac{H}{4}, 3\frac{W}{4}]$. Each region $r \in \{a, b, c, d\}$ receives its task k_r. In other words:

$$t_{yx} = \begin{cases} k_a, \text{ for } x \leq c_x, y \leq c_y \\ k_b, \text{ for } x > c_x, y \leq c_y \\ k_c, \text{ for } x \leq, y > c_y \\ k_d, \text{ for } x > c_x, y > c_y \end{cases} \tag{6}$$

The specific tasks $k_a, ...k_d$, as well as $\mathbf{c}$, can be changed every time the Task Palette is requested from this rule.

The rule $\mathcal{R}_2$ requests:

- **Surface normals** on pixels which belong to the semantic classes: bottle, chair, dining table, potted plant, sofa and tv monitor. In other word, this rule requests surface normals on common *household objects* from the dataset.
- **Human body parts** on pixels which belong to *humans*.
- **Segmentation** on pixels which belong to birds, horses, cows, cats, dogs and sheep. In other words, this rule requests segmentation on *animals* from the dataset.
- **Saliency** on aeroplanes, bicycles, boats, buses, cars, motorbikes and trains. In other words, this rule requests saliency on *vehicles* from the dataset.
- **Edges** everywhere else.

The rule $\mathcal{R}_3$ requests:

- **Surface normals** on pixels which belong to chairs, dining tables, sofas, bicycles, buses, cars, motorbikes and trains. In other words, it requests surface normals on *some vehicles* and *some common household objects* from the dataset.
- **Human body parts** on pixels which belong to *humans*.
- **Saliency** on aeroplanes, boats, birds, horses, cows, cats, dogs, sheep, bottles, potted plants and tv monitors. In other words, it requests saliency on *other vehicles*, *other household objects* and *animals* from the dataset.
- **Edges** everywhere else.

The rule $\mathcal{R}_3$ is designed to be similar to rule $\mathcal{R}_2$. It uses all the same tasks as rule $\mathcal{R}_2$, except for semantic segmentation, which has been left out. The tasks of predicting edges and human body parts are requested at the exact same locations. The tasks of surface normals, and saliency are requested on a few classes where they are already requested in rule $\mathcal{R}_2$, but mostly on different classes.

D. Additional Qualitative Results

Here we provide more visual examples of the CompositeTasking network's capabilities

Random mosaics. Additional visual examples from the experiment presented in Figure 6 are presented in Figure 13. We can see that the network successfully predicts different tasks simultaneously at different pixel locations, all during the same forward pass. It shows the ability to sharply change the task which is being predicted in different spatial locations, with respect to the Task Palette $\mathcal{T}$.

Single task predictions. Additional visual examples from the experiments presented in Figure 7 are presented in Figure 14. Although the network is designed to be used in a CompositeTasking fashion, here we see that our network can make all task predictions successfully. In this case, images are encoded only once followed by multiple task-specific decodings.

Learning what to do where. Additional visual examples from the experiments presented in Figure 8 are presented in

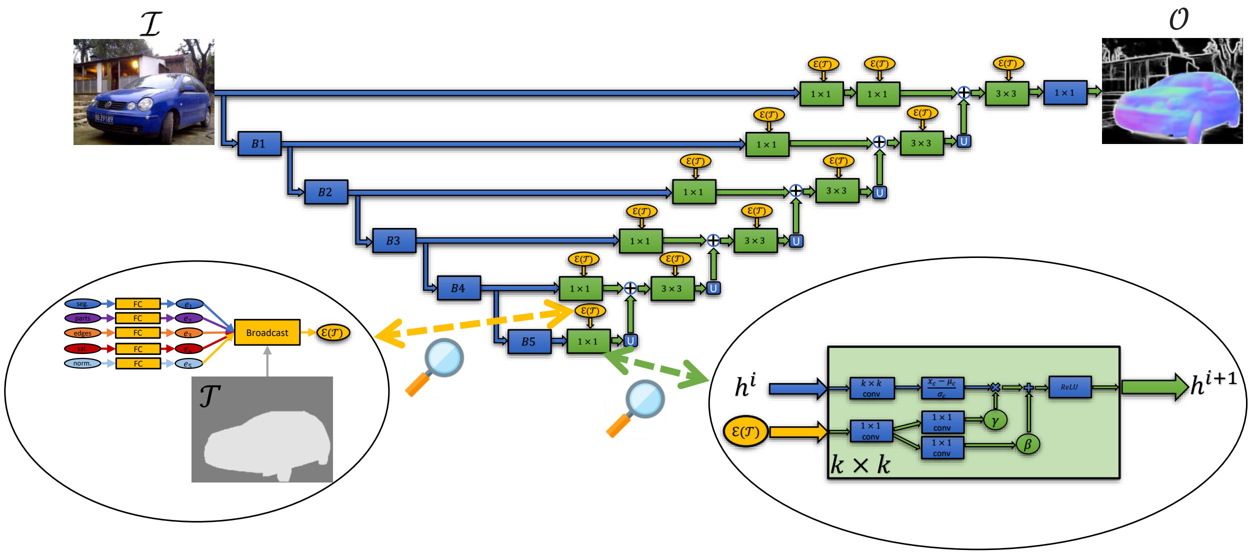

Figure 12: **CompositeTasking network architecture**. Blue blocks process features unconditionally, while the green blocks process features in a spatial task-conditioning manner. The network is composed out of an encoder and decoder. The decoder is composed only out of task composition blocks and upsampling operations. The task composition block is depicted in the lower-right corner of this diagram, while the upsampling operations are denoted with the blue rectangle containing the letter "U". The task composition block takes in a Task Palette embedding $\mathcal{E}(\mathcal{T})$, which is obtained using the task representation block, depicted in the lower-left part of this diagram.

Figure 15. We can see the networks ability to predict the Task Palette. Using the predicted palette, the model successfully makes CompositeTasking predictions.

Task Palette editing. Additional visual examples from the experiments presented in Figure 9 are presented in Figure 16. We can see that a user can make any modification to a given Task Palette, and obtain the respective CompositeTasking prediction.

Breaking the rule. Additional visual examples from the experiments presented in Figure 10 are presented in Figure 17. We see the model's ability to transfer the knowledge of the rule it trained on ($\mathcal{R}_2$), to a new rule ($\mathcal{R}_3$). The performance gets even better when fine-tuning is done on the new rule. In the 1st row we can see that the model was able to predict relatively good surface normals for the motorbike, even though it was never supervised to predict normals on motorbikes. After some fine-tuning on the new rule, it's predictions get even better. Notice how some almost flat surfaces on the vehicle (which have the same surface orientation) get more consistent normal predictions after fine-tuning. Since the new rule $\mathcal{R}_3$ does not contain the task of semantic segmentation anymore, that task is forgotten as can be seen in the 3rd row. In the 4th row, human body parts result, obtained using the rule $\mathcal{R}_3$, is shown.

E. Implementation Details

The networks are trained on the train partition of the data set and evaluated on the validation partition of the data set, in absence of a test partition. The images are resized to 256×256 both for training and validation. When using the rule $\mathcal{R}_{1r}$ for training, a new center $\mathbf{c}$ and task region-task assignments k_r are sampled for every batch. Training data was augmented using random horizontal flipping, shifting, scaling, cropping, Gaussian noise, adaptive histogram equalization, brightness change, sharpening, blurring, contrast change, hue change and saturation change. Flipping, scaling and shifting were not used when predicting surface normals.

Each distinct task k of the Task Palette $\mathcal{T}$ is embedded into a latent vector $\mathbf{w}$ using the embedding network depicted in figure 12. The embedding network is a fully connected neural network with 6 layers and leaky ReLU activations. To obtain an embedding for a specific task, its code z_k is given to the embedding network. The codes are vectors of length $20K$, where K is the number of tasks. The code of task 1 is constructed by putting ones in the first 20 elements of the vectors, and zeroes everywhere else. For task 2, we have ones in the vectors positions 21...40, etc. This way, the code vectors for different tasks will be orthogonal to each other. The code vectors z_k are also L_2 normalized to have unit length. Since the embedding $\mathcal{E}$ from Figure 3 is needed in the decoder in different spatial sizes, a pyramid of downsized versions is prepared, along with the original $\mathcal{E}$. We use downsampling with bilinear interpolation.

The losses for each task k are computed independently for every pixel belonging to that task in $\mathcal{T}$, and averaged to produce $\mathcal{L}_k(\mathcal{O}, \mathcal{Y}_k, \mathcal{T})$ from (4). For semantic segmentation and human body parts we use the focal loss [29] with

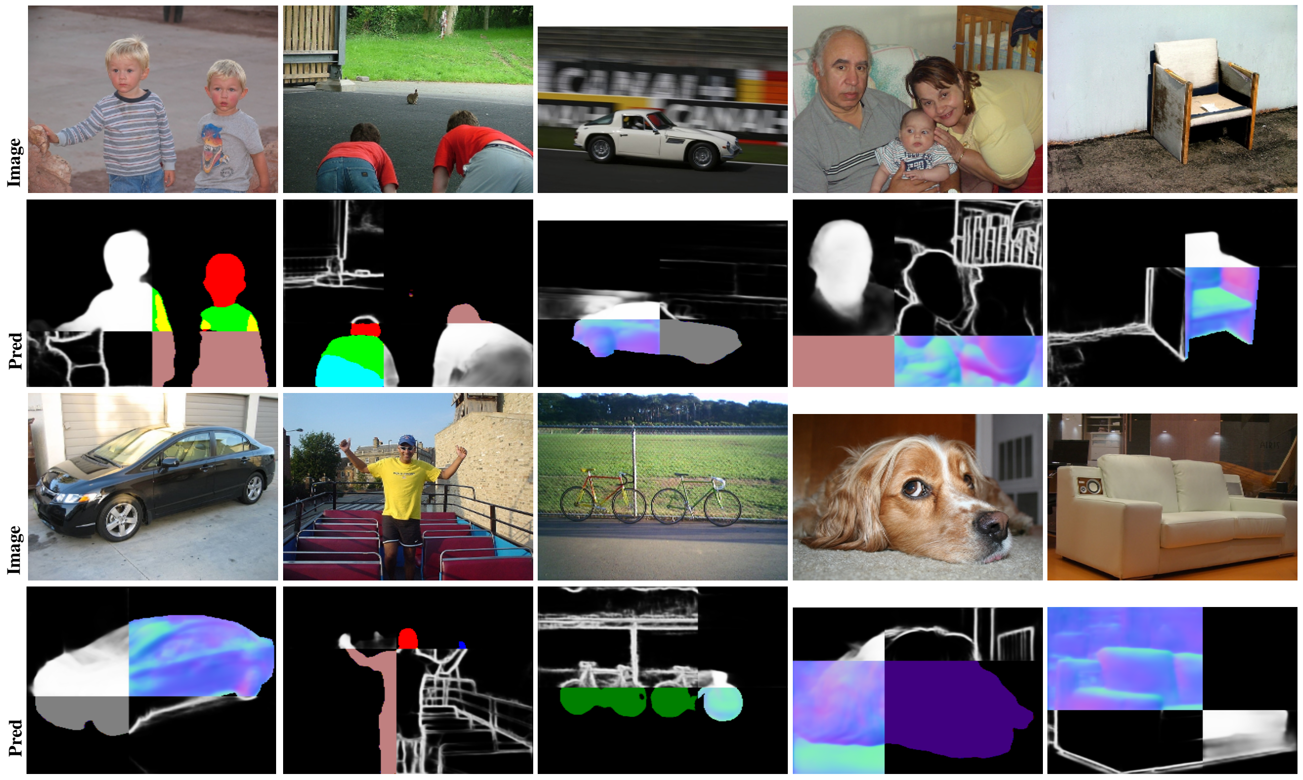

Figure 13: **Random mosaic compositions.** CompositeTasking network predictions on requests with the $\mathcal{R}_{1r}$ rule.

$\gamma = 2$, which was shown to be more effective in our setup than standard cross-entropy loss. We use weighted cross-entropy loss for edges. The weight for the positive class (edges) is 0.95 while the weight or the negative class is 0.05, because the edges are usually present in a substanially smaller portion of the image pixels. We use weighted cross-entropy loss for saliency. The weights are calculated in each batch as the inverse frequency of elements that belong to the positive and negative class, normalized to sum up to 1.0. For surface normals we use $1 - CS(o, y)$, where CS is the cosine similarity. The chosen loss weights λ_k are 3 for semantic segmentation, 4 for human parts, 50 for edges, 8 for saliency and 4 for surface normals. This provided the best trade-off in experiments.

The exact 3D class anchors $\mathbf{a}_i$ (see eq. 5) used for the task of semantic segmentation and human body parts can be found in Tables 5 and 6, respectively.

The model was optimized using the Adam optimization algorithm, using weight decay of magnitude 0.00001. The encoder is pre-trained on ImageNet , while the weights of the decoder are randomly initialized. Because of that the encoder is optimized with a smaller learning rate of 0.00001, while the decoder is optimized with the learning rate of 0.001. If the loss does not improve over the course of 12 epochs, the learning rates are multiplied by a factor of 0.3. We use leaky ReLU as the activation function. Since the PASCAL data set only provides the training and validation partitions, the networks are trained for 100 epochs which was shown to be enough for them to converge. They were trained on Nvidia GPUs with either 12Gb or 16Gb of RAM memory. A batch size of 10 was used.

F. Further Analysis

In this section, we present analysis that shed more light on our choice of hyper-parameters. The average per-task drop metric from Section 6.1 is also used here. Instead of computing the metric with respect to a single task baseline, now it is computed with respect to the performance of the model with the chosen hyper-parameters. All the following analysis are conducted by evaluating on the single task rule $\mathcal{S}$.

In the task composition blocks we use one shared convolution that processes the Task Palette embedding $\mathcal{E}(\mathcal{T})$, before it is converted to the affine transformation parameters γ and β. In Table 7 we see that one shared convolution was the best choice. If we use more shared convolutions, we achieve a slightly worse performance with an increased computational cost.

We used a regular convolution with kernel size 3×3 in the task composition blocks of the decoder. In Table 8 we see that a kernel size of 1×1 achieves worse performance

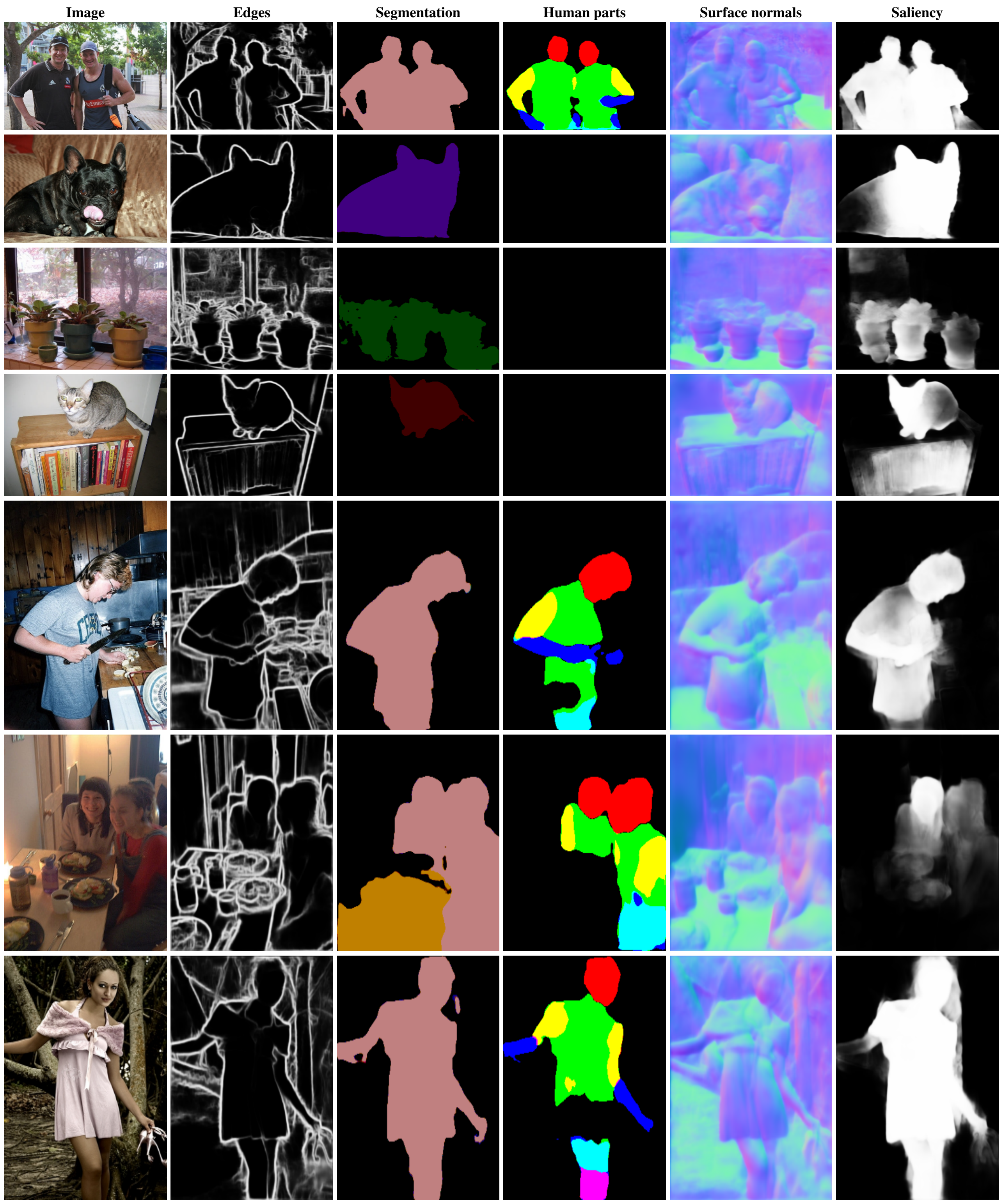

Figure 14: **Single task predictions.** Even though our model is made for CompositeTasking, it can also make predictions on requests with the $\mathcal{S}$ rule.

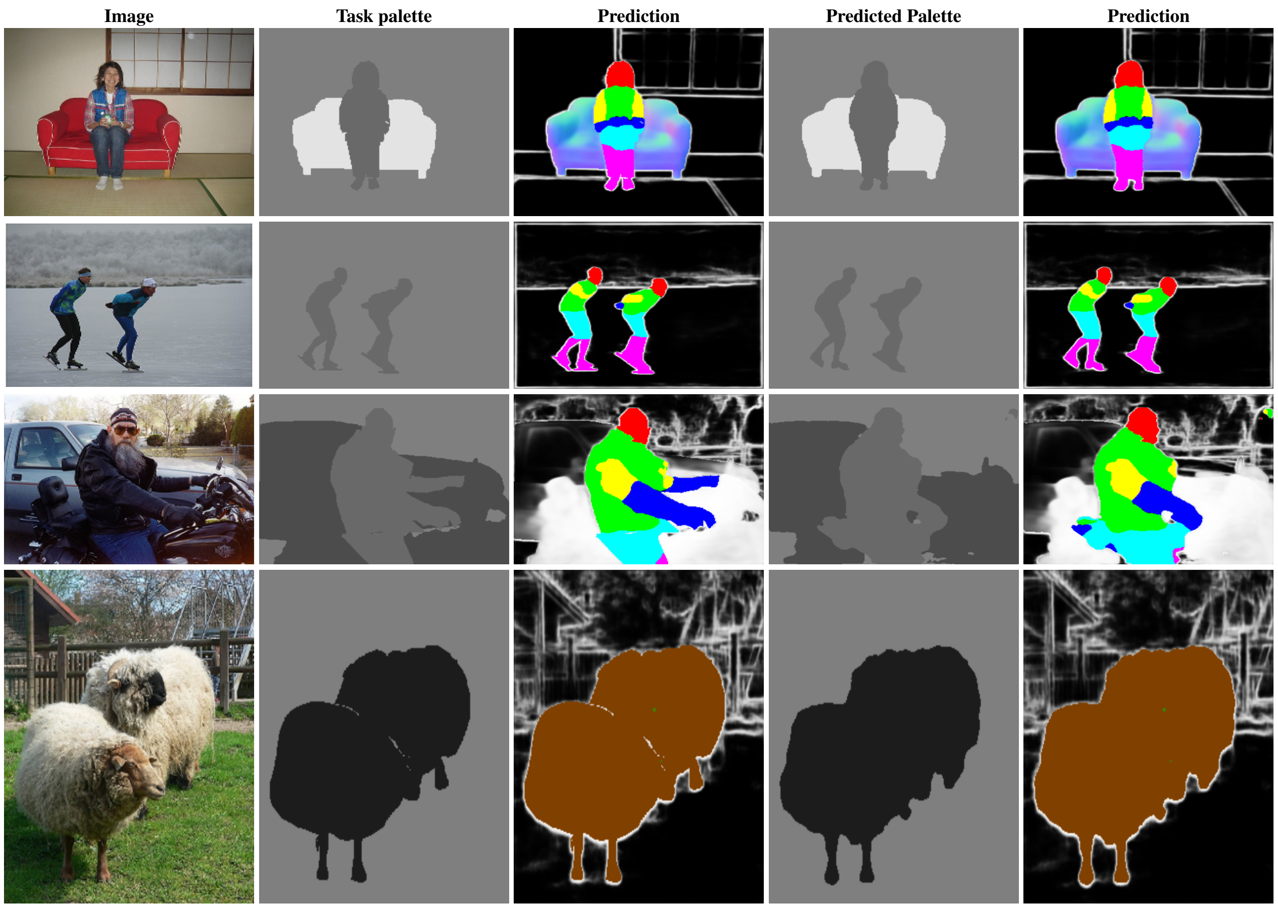

Figure 15: **Learning what to do where.** CompositeTasking network predictions with the learned Task Palette. A separate network is learned to predict the Task Palette with 75.04% mIoU.

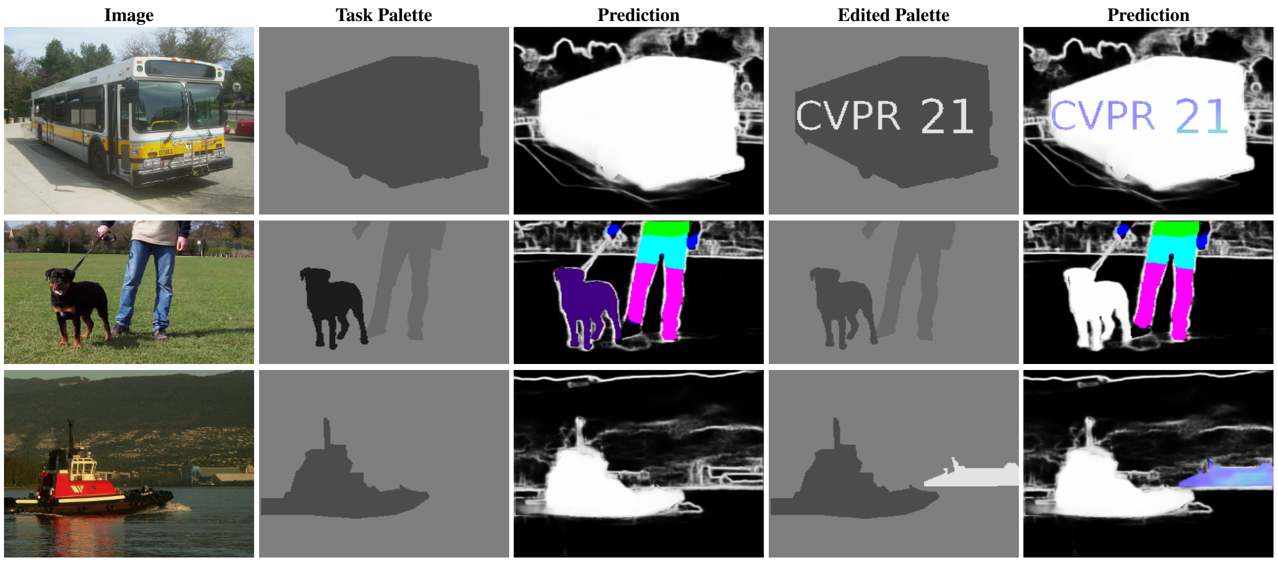

Figure 16: **Task Palette editing.** The Task Palette has been modified manually. A prediction of the CompositeTasking network is shown before and after modification.

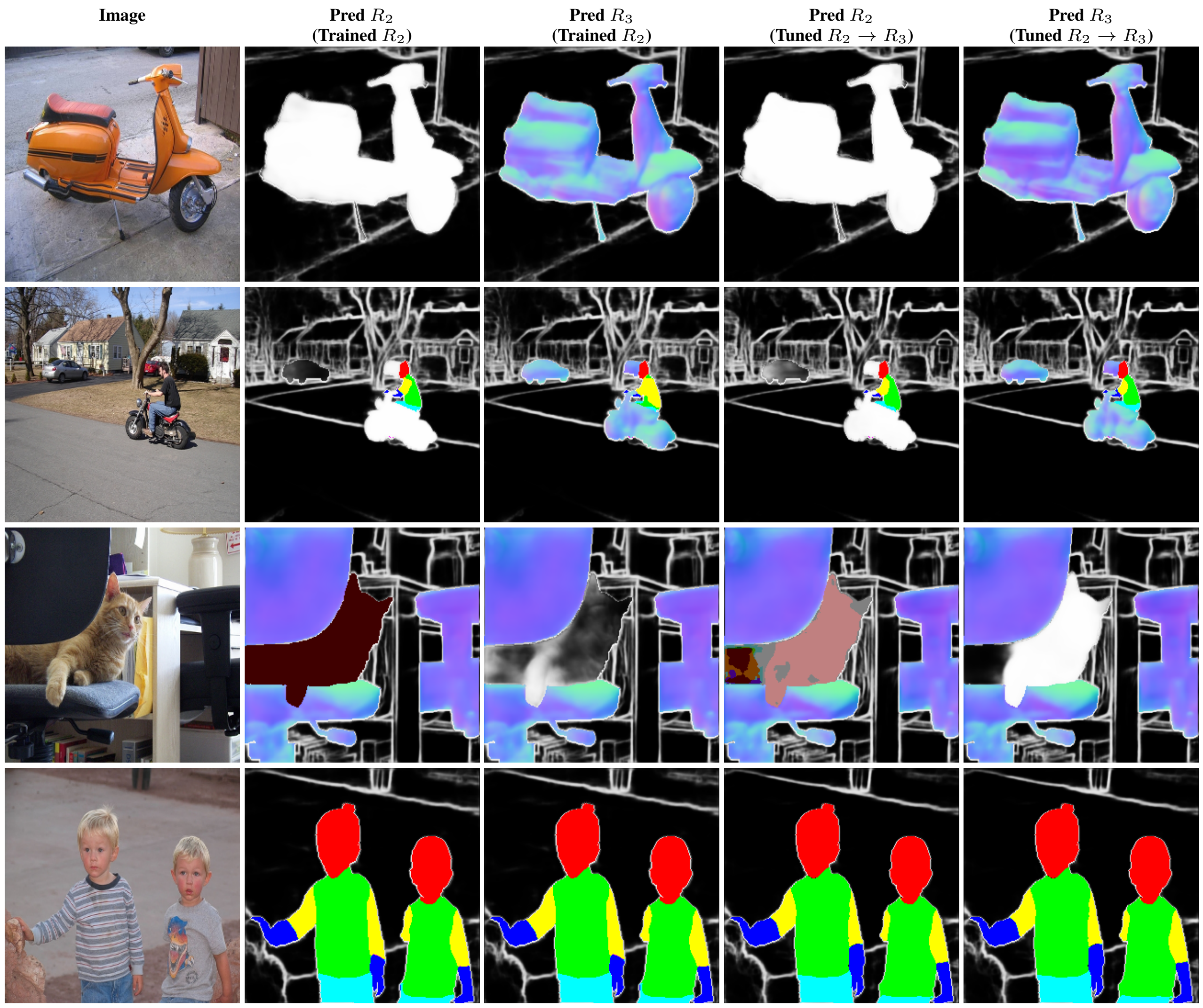
Image
Pred R_2 (Trained R_2)
Pred R_3 (Trained R_2)
Pred R_2 (Tuned $R_2 \rightarrow R_3$)
Pred R_3 (Tuned $R_2 \rightarrow R_3$)

Figure 17: **Breaking the rule.** Predictions of the CompositeTasking network are show before and after fine-tuning from the old to the new semantic rule. Because the rules are similar in a way, the model can already extrapolate even before fine-tuning.

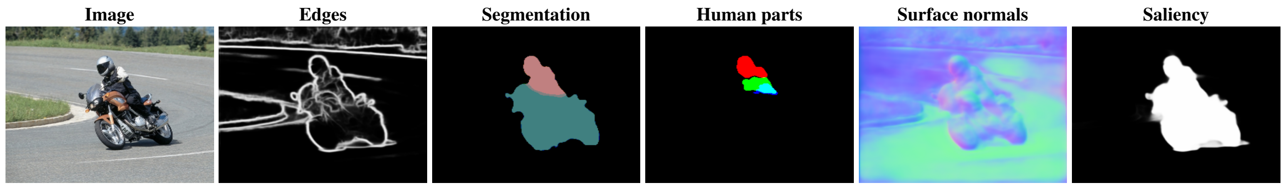

Figure 18: **Unsuccessful single task predictions.**

then our choice.

We choose to do image-to-image translation by having 3 output channels. In Table 9 we observe that if we use more output channels, there is no gain in predictive performance.

When a Task Palette embedding $\mathcal{E}(\mathcal{T})$ is produced, it is down-scaled in all spatial sizes that the decoder needs. In Table 10 we compare our choice of bi-linear interpolation to nearest neighbours. We choose the bi-linear interpolation because it achieves slightly better results. This discrepancy in performance may be caused due to the negligence of the sampling theorem, i.e. ignoring that downsampled pixels on the task boundary represent more than just one distinct task [63]. Also, the nearest neighbours interpolation can be used as a trade-off between slightly worse performance and slightly faster computations. When applying nearest neighbours interpolation there is no need to take into account all

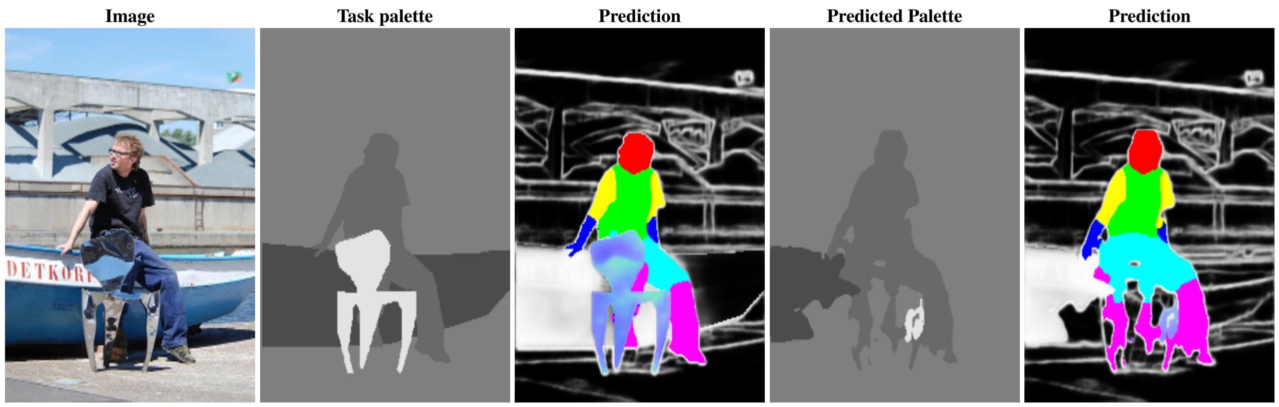

Figure 19: **Unsuccessfully predicting what to do where.**

Table 5: **Class anchors $\mathbf{a}_i$ for the task of semantic segmentation**. These anchors define which part of the 3D output space corresponds to which class, based on the nearest neighbor principle. They also define the RGB class colors used for all the presented results.

Class id	Class name	x_1	x_2	x_3
0	Background	0	0	0
1	Cat	0	0	64
2	Aeroplane	0	0	128
3	Chair	0	0	192
4	Potted plant	0	64	0
5	Sheep	0	64	128
6	Bicycle	0	128	0
7	Cow	0	128	64
8	Bird	0	128	128
9	Dining table	0	128	192
10	Sofa	0	192	0
11	Train	0	192	128
12	Boat	128	0	0
13	Dog	128	0	64
14	Bottle	128	0	128
15	Horse	128	0	192
16	TV monitor	128	64	0
17	Bus	128	128	0
18	Motorbike	128	128	64
19	Car	128	128	128
20	Person	128	128	192

Table 6: **Class anchors $\mathbf{a}_i$ for the task of human body parts**. These anchors define which part of the 3D output space corresponds to which class, based on the nearest neighbor principle. They also define the RGB class colors used for all the presented results.

Class id	Class name	x_1	x_2	x_3
0	Background	0	0	0
1	Head	0	0	255
2	Neck & torso	0	255	0
3	Upper arms	255	0	0
4	Lower arms	0	255	255
5	Upper legs	255	0	255
6	Lower legs	255	255	0

Table 7: **Number of shared convolutions that process the Task Palette embedding $\mathcal{E}(\mathcal{T})$ in the task composition block**.

# of convolutions	Edge↑	SemSeg↑	Parts↑	Normals↓	Sal↑	Δ_m%↓
1 (**chosen**)	68.60	62.45	52.59	16.93	67.81	0.00%
2	68.60	61.93	51.49	17.15	67.56	-0.92%
3	68.30	62.01	52.28	17.16	67.60	-0.68%

Table 8: **Kernel size of regular convolutions inside the decoders task composition blocks**.

Kernel size	Edge↑	SemSeg↑	Parts↑	Normals↓	Sal↑	Δ_m%↓
3×3 (**chosen**)	68.60	62.45	52.59	16.93	67.81	0.00%
1×1	66.20	61.50	51.05	17.87	67.16	-2.89%

Table 9: **Number of output channels of the model**.

# of output channels	Edge↑	SemSeg↑	Parts↑	Normals↓	Sal↑	Δ_m%↓
3 (**chosen**)	68.60	62.45	52.59	16.93	67.81	0.00%
6	68.50	62.35	52.99	17.08	67.65	-0.13%
9	68.50	62.42	52.50	17.39	67.89	-0.59%

Table 10: **Type of interpolation used during the Task Palette embedding $\mathcal{E}(\mathcal{T})$ downsampling**.

Interpolation	Edge↑	SemSeg↑	Parts↑	Normals↓	Sal↑	Δ_m%↓
Bi-linear (**chosen**)	68.60	62.45	52.59	16.93	67.81	0.00%
Nearest neighbour	67.60	60.92	51.37	17.20	67.51	-1.65%

the values of $\mathcal{E}(\mathcal{T})$, but only the closest elements to the new pixel centers.

In Table 11 we analyze what is the good number of fully connected layers in the task representation block. We can see that if we pick the number to be too small or too big, there appears to be problems during learning. Especially for a very big number of layers like 12, the network is having a hard time converging. Having 6 and 9 layers gave very similar results, and 6 was chosen simply because it is computationally less demanding.

From Table 12 we can see that the model is pretty robust to choosing the number of channels of the Task Palette em-

Table 11: **Number of fully connected layers in the task representation block**.

# of layers	Edge↑	SemSeg↑	Parts↑	Normals↓	Sal↑	Δ_m%↓
3	68.00	7.61	42.62	16.37	71.53	-19.77%
6 (**chosen**)	68.60	62.45	52.59	16.93	67.81	0.00%
9	68.70	62.29	52.18	16.95	67.43	-0.31%
12	52.80	36.46	13.69	67.60	44.93	-94.33%

Table 12: **Number of channels of the Task Palette embedding** $\mathcal{E}(\mathcal{T})$.

# of channels	Edge↑	SemSeg↑	Parts↑	Normals↓	Sal↑	Δ_m%↓
32	68.70	62.14	51.86	16.95	67.46	-0.47%
64	68.60	62.06	52.67	17.06	67.86	-0.23%
128 (**chosen**)	68.60	62.45	52.59	16.93	67.81	0.00%
256	68.40	60.36	51.53	17.11	67.35	-1.48%
512	1.40	11.22	4.53	30.87	18.64	-85.26%

Table 13: **Encoder backbone**.

Backbone	Edge↑	SemSeg↑	Parts↑	Normals↓	Sal↑	Δ_m%↓
ResNet18	67.40	60.40	49.25	17.25	66.87	-2.93%
ResNet34 (**chosen**)	68.60	62.45	52.59	16.93	67.81	0.00%
ResNet50	69.40	62.34	53.81	16.64	68.19	+1.12%
ResNet101	69.60	63.75	55.78	16.71	69.55	+2.69%

Table 14: **Loss choice for the task of estimating surface normals**.

Loss function	Edge↑	SemSeg↑	Parts↑	Normals↓	Sal↑	Δ_m%↓
cosine similarity (**chosen**)	68.60	62.45	52.59	16.93	67.81	0.00%
l_1 distance	67.60	58.91	49.20	19.77	67.53	-6.15%

bedding $\mathcal{E}(\mathcal{T})$. We chosen the number of channels which gave the best performance.

In Table 13 we analyze what happens if we use encoder backbones of different capacity. We can see a clear trend from the table — as the backbone has more capacity (more learnable parameters and computations) the predictive performance of the model gets better. This was expected, and it shows one possible way to improve the performance. The choice for ResNet34 trades-off between performance and memory/computational demands.

In Table 14 we analyze the choice of the loss function for surface normals. As we can see, using cosine similarity achieves much better results that the popular L1 loss. Also, when using L1 loss, the values of the loss function over epochs looked much more unstable than compared to using cosine similarity.

In Table 15 we analyze the choice of γ in the focal loss used in segmentation and human parts. We choose $\gamma = 2$ because that it achieved the best performance. Using $\gamma = 0$ is the same as using regular cross-entropy. Even though regular cross-entropy was close in performance to using the focal loss, when supervising with it the models often failed to converge.

Table 15: **Choice of parameter** γ **from focal loss used in segmentation and human parts**.

γ	Edge↑	SemSeg↑	Parts↑	Normals↓	Sal↑	Δ_m%↓
0	68.40	61.39	51.93	17.29	67.40	-1.19%
1	68.10	61.99	51.61	17.46	67.05	-1.52%
2 (**chosen**)	68.60	62.45	52.59	16.93	67.81	0.00%
3	68.70	61.60	51.39	17.06	67.88	-0.83%

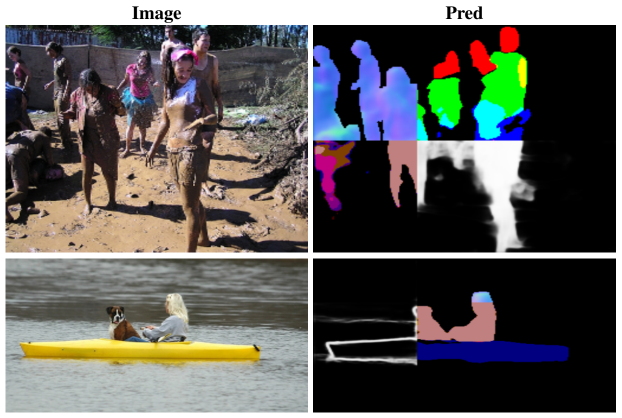

Figure 20: **Unsuccessful random mosaic composition predictions.**

G. Limitations

The proposed network has the ability to only predict one task for each spatial location, during one forward pass. In some cases, this could be a limitation. When making predictions on high-level tasks, modern approaches predict many different low-level tasks and fuse them in order to make the final prediction. Although it is unnecessary to predict every task everywhere – as laid out in our introduction –, sometimes it could certainly be necessary to perform more than one task per each spatial location. As one example, lets think about predicting where to turn the steering wheel in a self-driving application. When a person is in the frame we might want to detect them as a person (semantic segmentation), but we would also like to know the body part locations and the depth at its location so we can reason about the person's pose, intended motion, and how close the person is to the car.

Even in the case of our one-task-per-location model however, predictions are not always perfect. Now we will look at some examples, where the model does not make good predictions. In the 1st row of Figure 20, we can see that the model fails to predict the body parts, and the semantic segmentation correctly. This happens often when there are many people in the scene, especially in strange circumstances like in this example. In the 2nd row of Figure 20, we can see that the model segmented the dog as a human, probably because it was unlikely that a dog is riding on a boat in the dataset. In Figure 18, we can see that the model is not able to predict the human parts successfully, because the model recognises most of the human body as the motorcycle as can be notices in the semantic segmentation prediction. Finally, on Figure 19 we can see what happens if the Task Palette is not predicted correctly. Here the model is able to make a very good prediction given the appropriate palette. But when the palette is predicted wrongly, probably because of a lot of objects overlapping, the model makes a bad prediction.