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 and hyperparameters used to conduct experiments. 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 divided into 5 blocks: B1,...,B5. Each block Bi is a part of the encoder that takes features of spatial size $(\frac{H}{2^i}, \frac{W}{2^i})$, 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 $e = (cx, cy)$. The point is chosen randomly as $cx \sim U[\frac{W}{4}, \frac{3W}{4}], cy \sim U[\frac{H}{4}, \frac{3H}{4}]$. Each region $r \in \{a, b, c, d\}$ receives its task $k_r$. In other words:

$$t_{yx} = \begin{cases} 
k_a, & \text{for } x \leq cx, y \leq cy \\
k_b, & \text{for } x > cx, y \leq cy \\
k_c, & \text{for } x \leq y < cy \\
k_d, & \text{for } x > cy, y > cy 
\end{cases} \quad (6)$$

The specific tasks $k_a, ..., k_d$, as well as $e$, can be changed every time the Task Palette is requested from this rule.

The rule $\mathcal{R}_{2r}$ 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.

- **Saliency** on aeroplanes, bicycles, boats, buses, cars, motorbikes and trains. In other words, it requests saliency on other vehicles, other household objects and animals from the dataset.

- **Edges** everywhere else.

The rule $\mathcal{R}_{3r}$ 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, cows, 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 $\mathcal{R}_{3r}$ is designed to be similar to rule $\mathcal{R}_{2r}$. It uses all the same tasks as rule $\mathcal{R}_{2r}$, 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 $\mathcal{R}_{2r}$, 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
Additional visual examples from the experiments presented in Figure 10 are presented in Figure 17. 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 ( \( R_2 \)), to a new rule ( \( R_3 \)). 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 motorcycles. 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 \( R_3 \) 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 \( R_3 \), 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 x 256 both for training and validation. When using the rule \( R_1 \), for training, a new center \( c \) 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 \( T \) is embedded into a latent vector \( w \) 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 \( E \) from Figure 3 is needed in the decoder in different spatial sizes, a pyramid of downsized versions is prepared, along with the original \( E \). We use downsampling with bilinear interpolation.

The losses for each task \( k \) are computed independently for every pixel belonging to that task in \( T \), and averaged to produce \( L_k(O, y_k; T) \) from (4). For semantic segmentation and human body parts we use the focal loss [29] with
\( \gamma = 2 \), 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 of the negative class is 0.05, because the edges are usually present in a substantially smaller portion of the image pixels. We use weighted cross-entropy loss for edges. The weight for the positive class (edges) is 0.95 while the weight of the negative class is 0.05, because the edges are usually present in a substantially 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 \( \lambda_k \) 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 \( a_i \) (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 \( S \).

In the task composition blocks we use one shared convolution that processes the Task Palette embedding \( E(T) \), before it is converted to the affine transformation parameters \( \gamma \) and \( \beta \). 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 \( 3 \times 3 \) in the task composition blocks of the decoder. In Table 8 we see that a kernel size of \( 1 \times 1 \) achieves worse performance.
<table>
<thead>
<tr>
<th>Image</th>
<th>Edges</th>
<th>Segmentation</th>
<th>Human parts</th>
<th>Surface normals</th>
<th>Saliency</th>
</tr>
</thead>
</table>

Figure 14: **Single task predictions.** Even though our model is made for CompositeTasking, it can also make predictions on requests with the $S$ 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.
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 $\mathcal{E}(T)$ 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 downsamped 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
the values of $E(T)$, 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.

<table>
<thead>
<tr>
<th># of layers</th>
<th>Edge↑</th>
<th>SemSeg↑</th>
<th>Parts↑</th>
<th>Normals↓</th>
<th>Sal↑</th>
<th>∆m,%↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>68.00</td>
<td>7.61</td>
<td>42.62</td>
<td>16.37</td>
<td>71.53</td>
<td>-19.73%</td>
</tr>
<tr>
<td>6 (chosen)</td>
<td>68.60</td>
<td>62.45</td>
<td>52.59</td>
<td>16.93</td>
<td>67.81</td>
<td>0.00%</td>
</tr>
<tr>
<td>9</td>
<td>68.70</td>
<td>62.29</td>
<td>52.18</td>
<td>16.95</td>
<td>67.43</td>
<td>-0.31%</td>
</tr>
<tr>
<td>12</td>
<td>52.80</td>
<td>36.46</td>
<td>13.69</td>
<td>67.60</td>
<td>44.93</td>
<td>-94.33%</td>
</tr>
</tbody>
</table>

Table 12: Number of channels of the Task Palette embedding $E(T)$.  

<table>
<thead>
<tr>
<th># of channels</th>
<th>Edge↑</th>
<th>SemSeg↑</th>
<th>Parts↑</th>
<th>Normals↓</th>
<th>Sal↑</th>
<th>∆m,%↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>68.70</td>
<td>62.14</td>
<td>51.86</td>
<td>16.95</td>
<td>67.46</td>
<td>-0.47%</td>
</tr>
<tr>
<td>64</td>
<td>68.60</td>
<td>62.06</td>
<td>52.67</td>
<td>17.06</td>
<td>67.36</td>
<td>-0.23%</td>
</tr>
<tr>
<td>128 (chosen)</td>
<td>68.60</td>
<td>62.43</td>
<td>52.59</td>
<td>16.93</td>
<td>67.81</td>
<td>0.00%</td>
</tr>
<tr>
<td>256</td>
<td>68.40</td>
<td>60.36</td>
<td>51.53</td>
<td>17.11</td>
<td>67.35</td>
<td>-1.48%</td>
</tr>
<tr>
<td>512</td>
<td>1.40</td>
<td>11.22</td>
<td>4.53</td>
<td>30.87</td>
<td>18.64</td>
<td>-85.26%</td>
</tr>
</tbody>
</table>

Table 13: Encoder backbone.

<table>
<thead>
<tr>
<th>Backbone</th>
<th>Edge↑</th>
<th>SemSeg↑</th>
<th>Parts↑</th>
<th>Normals↓</th>
<th>Sal↑</th>
<th>∆m,%↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet18</td>
<td>67.40</td>
<td>60.40</td>
<td>49.25</td>
<td>17.25</td>
<td>66.87</td>
<td>-2.93%</td>
</tr>
<tr>
<td>ResNet34 (chosen)</td>
<td>68.60</td>
<td>62.45</td>
<td>52.59</td>
<td>16.93</td>
<td>67.81</td>
<td>0.00%</td>
</tr>
<tr>
<td>ResNet50</td>
<td>69.80</td>
<td>62.34</td>
<td>53.81</td>
<td>16.66</td>
<td>68.19</td>
<td>-1.12%</td>
</tr>
<tr>
<td>ResNet101</td>
<td>69.60</td>
<td>63.75</td>
<td>55.18</td>
<td>16.71</td>
<td>69.55</td>
<td>-2.69%</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Loss function</th>
<th>Edge↑</th>
<th>SemSeg↑</th>
<th>Parts↑</th>
<th>Normals↓</th>
<th>Sal↑</th>
<th>∆m,%↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>cosine similarity (chosen)</td>
<td>68.60</td>
<td>62.45</td>
<td>52.59</td>
<td>16.93</td>
<td>67.81</td>
<td>0.00%</td>
</tr>
<tr>
<td>$\gamma$-distance</td>
<td>67.60</td>
<td>58.91</td>
<td>49.20</td>
<td>19.77</td>
<td>67.31</td>
<td>-6.33%</td>
</tr>
</tbody>
</table>

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 noticed 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.