École Polytechnique Fédérale de Lausanne

Context-Aware Object Detection for Zero-Shot Monocular Depth Estimation

by Ahmed Jellouli

Semester Project Report

Prof. Sabine Süsstrunk
Project Advisor

Bhattacharjee Deblina
Project Supervisor

January 8, 2022

Abstract

Recent advances in monocular depth estimation has shown considerable performance improvements in zero-shot settings. Zero-shot depth estimation is useful in many fields such as autonomous driving, robot navigation and others, but remains limited compared to the stereo setting. In this project, we explore the use of other pieces of information in the input image for estimating depth. Specifically, we train an object detection network and use it to extract semantically contextualized object features from the image. We then fuse these feature maps as extra channels in the decoder of a convolutional neural net. Our results show a strong dependency of performance on the quality of the object detection. We observe that object information may be confusing the decoder rather than helping it.

Contents

Chapter 1

Introduction

Monocular depth estimation is a field of Computer Vision that is concerned of estimating the depth dimension from a single 2D image. Depth is extremely useful and sometimes critical information in many engineering fields such as autonomous driving, robotics, drones and others.
Traditionally, depth is reconstructed from stereo information, i.e, 2 or more different images from the same scene. The ability to estimate depth from a single signal alleviates the need for expensive camera settings and tedious calibration, but is however more challenging than the former case.
Recent work [20] [18] has shown that in order to obtain good performance in different settings, one needs to train on diverse sources of data.
In this project, we follow on the footsteps of this idea. We train a modified version of MiDaS [20] that uses object information for estimating depth. We hypothesize that isolated object feature vectors would not give sufficient information to the model. Therefore, we train a second model based on HKRM [10] in order to obtain semantically contextualized object features to use in the downstream task.

Chapter 2

Background

We spent a fair portion of the semester researching existing literature on the topic. In this section, we present the necessary background to understand the final idea and the data/model exploration we performed.

2.1 MiDaS

In [20], Ranftl et al. train a ResNet-based architecture from Xian et al. [7] on multiple datasets using a special scale and shift-invariant loss function as well as multi-objective optimization [23]. The original midas model was trained on a meta-dataset of 5 datasets, named MIX5. In [18], the same authors introduce a bigger meta-dataset comprised of the original MIX5 and 5 other datasets, coined MIX6. They use this dataset to train a model based on vision transformers [2] and to re-train the original midas model.

2.1.1 MIX5 and MIX6

In order to train our final model, we use the original MIX6 dataset. This meta-dataset is comprised of 10 datasets and comprises a total of 1.4 million images. Since the source of each dataset is different, extra care has to be taken in order to train the model.
The authors in [18] and [20] do not make MIX6 publicly available. We therefore had to reconstruct the dataset from scratch. Out of the 10 datasets, we were able to download/reconstruct 6 of them, which were used for the final training. Table 2.1 summarizes the different MIX6 datasets. MegaDepth [13] is a set of outdoor images extracted from the internet along with depth maps obtained from structure-from-motion and multi-view stereo matching methods. ReDWeb [16] is a small dataset of outdoor dynamic scenes along with stereo based disparity maps. TartanAir [27] is comprised of synthetic images of several outdoor environments along with accurate depth

Table 2.1: MIX6 datasets, 6 out of the 10 datasets were used for training

Dataset	Size	Annotated	Annotation	Available	Reason
3D Movies [20]	75K	No		No	Proprietary 3D movies
DIML Indoor [1]	220K	Yes	Depth	No	Google Drive download limits
MegaDepth [13]	130K	Yes	Depth	Yes	
ReDWeb [16]	3600	Yes	Disparity	Yes	
WSVD [25]	1.5M	No		No	Requires Stereo matching of videos
TartanAir [27]	151K	Yes	Depth	Yes	
HRWSI [29]	20K	Yes	Disparity	Yes	
ApolloScape [9]	4K	Yes	Depth	Yes	
BlendedMVS [30]	17K	Yes	Depth	Yes	
IRS [26]	100K	Yes	Disparity	No	Google Drive download limits

maps. Like MegaDepth, HRWSI [29] is a dataset of web stereo photos and their associated depth maps. ApolloScape [9] is comprised of images taken for autonomous driving, and contains therefore scenes from a vehicle point of view. Finally, BlendedMVS [30] is a dataset of 3D reconstructed scenes, sculptures, buildings and small objects.

2.1.2 Testing and failure cases

We test the original MIX6-trained model on the 6 datasets that we were able to download and find the common failure cases. To do this we evaluate the Root Mean Squared Error (RMSE) on a random sample of 1000 images for each of the 6 datasets. Figure 2.1 shows a boxplot of the result of this evaluation. We can see that the performance is good and that the results are balanced between the different datasets. We notice however that there is a large number of outliers in each case. We investigate these images in order to find potential failure cases.

Failure cases

For each dataset, we investigate the top 10 highest loss images and comment about the model performance. Figure 2.2 presents some examples of the biases we find. We notice that the network has trouble with similar coloured and textured surfaces, as well as constant depth surfaces. For ApolloScape, we notice a bias towards labeling the right side of the image as lower in depth. Finally, we note the loss of granularity present for low disparity regions.

2.1.3 Training on multiple datasets

Loss function

Training on a set of diverse datasets presents many challenges. The ground truth may be in depth space or disparity space, and it may be arbitrarily scaled and shifted. The accuracy of

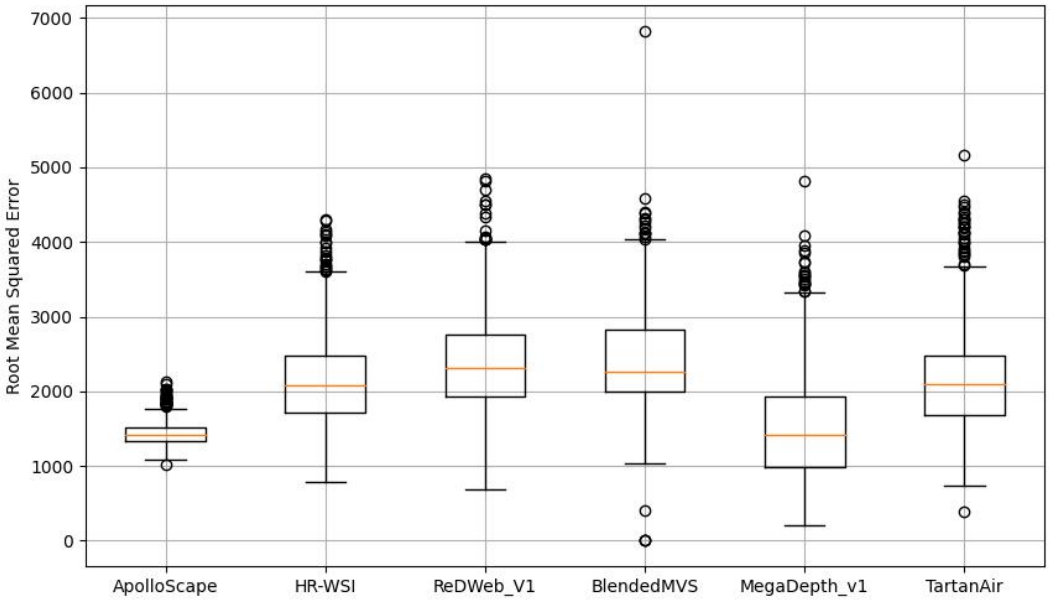


Figure 2.1: Box-plots of the result of the evaluation of the model on a batch of 1000 samples for each of the 6 datasets. Note that the 0 loss point for BlendedMVS is to be ignored as it corresponds to an invalid sample.

the ground truth varies between datasets. For these reasons, the loss function needs to be shift and scale-invariant, as well as robust to outliers. The authors experiment with many losses to deal with this problem. Here we present the one that was used for training our model. First, all ground truth labels are converted into disparity space. It is assumed that pixels with a value of 0 correspond to invalid ground truth labels. This disparity is then mapped into the $0-1$ range according to the formula:

$$D = \frac{D - \min\limits_{D}}{\max\limits_{D} - \min\limits_{D}} \quad \text{where D is the disparity image} \tag{2.1}$$

Given a prediction disparity map P and ground-truth disparity D we compute the Scale and Shift Invariant Trimmed Mean Absolute Error (SSIMAE). First the shift and scale of each image is estimated according to the following:

$$t(I) = \text{median}(I); \quad s(I) = \frac{1}{M}\sum_{i=1}^{M}|I_i - t(I)| \tag{2.2}$$

where M is the number of valid pixels and the image is flattened into a single dimension.
Then both both D and P are mapped into a scaled and shifted versions D^* and P^*:

$$D^* = \frac{D - t(D)}{s(D)} \quad P^* = \frac{P - t(P)}{s(P)} \tag{2.3}$$

The SSIMAE is computed by calculating the mean absolute difference of the images while trimming the largest t% of residuals:

$$SSIMAE(D^*, P^*) = \frac{1}{M} \sum_{i=1}^{\lfloor (1-t) \rfloor M} R_i; \tag{2.4}$$

where t is the trimming percentage (the authors use 0.2) and the R_i's are the sorted residuals:

$$R_i = |D^* - P^*|_i; \quad R_i \leq R_j \quad \text{for} \quad i \leq j \tag{2.5}$$

We note that all operation above (scaling, shifting, SSIMAE) is done in masked fashion i.e, it is only computed on valid ground truth pixels.
The authors use an adapted version of the multi-scale, scale-invariant gradient matching term [13] as a regularization for the gradients. This term has a bias for sharp discontinuities in the disparity maps [20]. Let R^k denote the difference between P^* and D^* at scale k:

$$R^k = |P_k^* - D_k^*| \tag{2.6}$$

where P_k^* and D_k^* denote the two disparity maps with resolution halved k times. The L_{reg} is:

$$L_{reg}(P^*, D^*) = \frac{1}{M} \sum_{k=1}^{K} \sum_{i=1}^{M} (\nabla_x R_i^k + \nabla_y R_i^k) \tag{2.7}$$

Where ∇_x and ∇_y denote the discrete gradient operator in the x and y direction respectively, M is the number of valid pixels, and K is the number of scales (the authors set $K = 4$). The inner sum is taken oven the valid pixels, like previously.
The final loss for one sample is then:

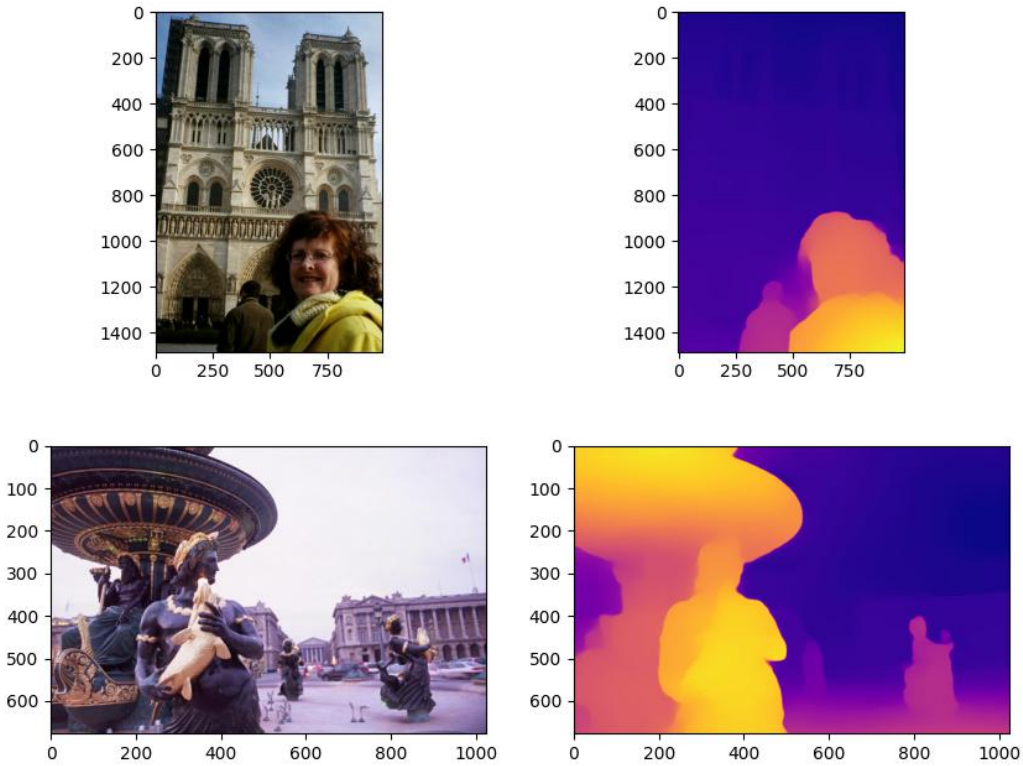
$$L(P^*, D^*) = SSIMAE(P^*, D^*) + \alpha L_{reg}(P^*, D^*) \tag{2.8}$$

Where α is set to 0.5

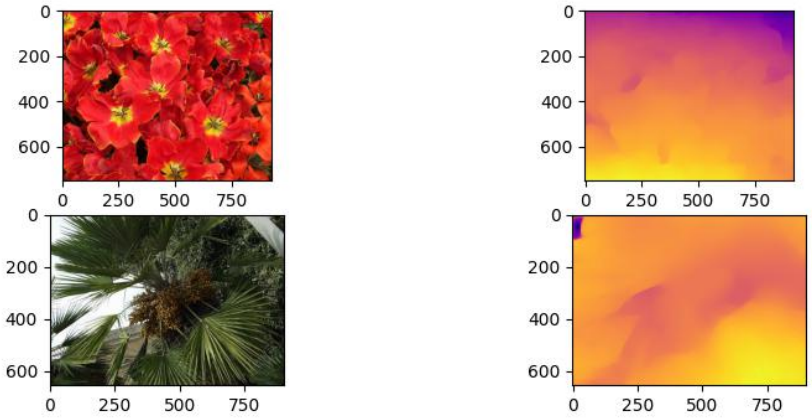
Dataset mixing strategies

The authors experiment with two different mixing strategies. The first uses multi-objective optimization [23]. In our case, we use the second strategy where we sample the same number of images from each dataset for each iteration and compute the sum of the losses, i.e, for a batch

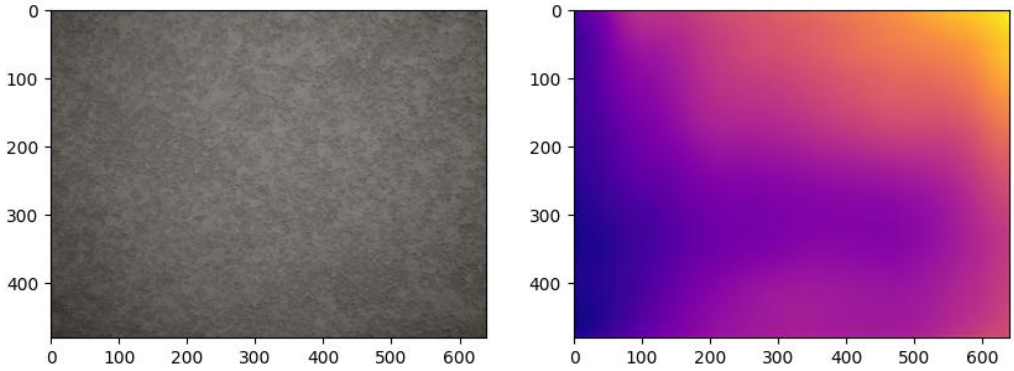
size of B and L datasets, each iteration samples $\frac{B}{L}$ samples from each dataset.



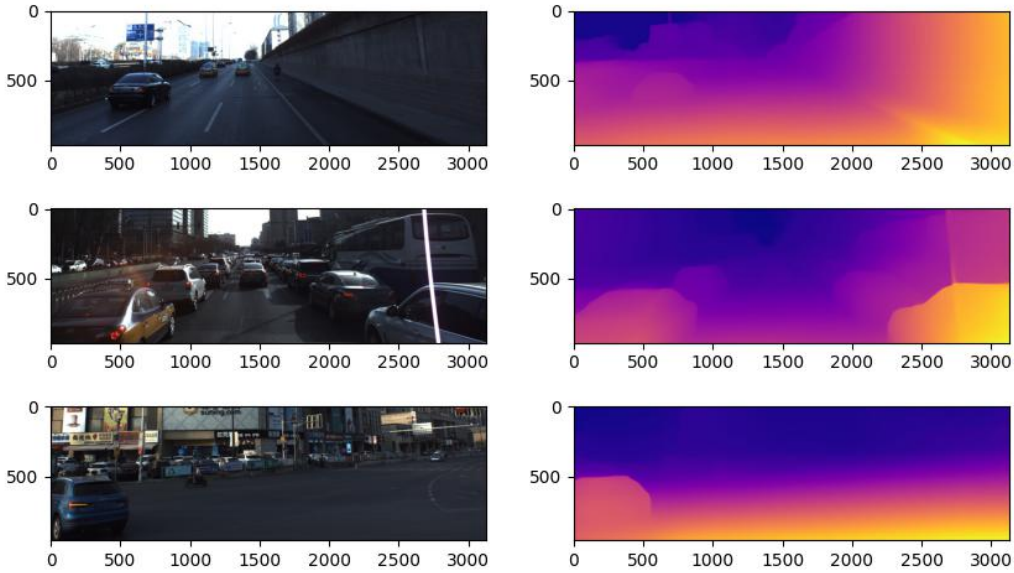
(a) MegaDepth failure cases. From top to bottom: Loss of granularity for low disparity, Similar color scheme is seen as similar depth



(b) HRWSI failure cases. Similar texture leads to equivalent depth



(c) Tartan Air failure case. Constant depth case not handled



(d) ApolloScape failure cases. First and second images: bias towards low disparity on the right side, Third: loss of granularity for low disparity

(e) BlendedMVS failure case. Multi-colored object

Figure 2.2: Some failure cases of MiDaS

2.2 R-CNN: Regions with CNN features

RCNNs [5], Fast-RCNNs [4] and Faster-RCNNs [21] are neural networks for object detection, each one a refinement of the previous architecture.

2.2.1 RCNN

The original RCNN uses the selective search algorithm [24] in order to find a number of region proposals. Each region is then fed through a convolutional neural network. The output is a feature vector for each region proposal. These vectors are fed into N Support Vector Machines, where N is the number of object categories to classify the proposal.

2.2.2 Fast-RCNN

Fast-RCNN is an improvement on the original RCNN. The input image is fed through a series of convolutional and max-pooling layers to obtain a volume of feature maps. For each region proposal, computed through an algorithm like selective search, the corresponding region from the feature map is fed into a pooling layer (named RoI pooling layer) in order to extract region features. The resulting feature vector is used to classify the region, where instead of N SVMs, a Feed-Forward Neural Network with $N + 1$ output classes (extra background class) is used. The same feature vector is used as well as to perform a regression on the bounding box (BBox regression), which refines the coordinates of the region proposal. The network is trained using a multi-task loss.

2.2.3 Faster-RCNN

Faster-RCNN removes the need for a region proposal algorithm such as selective search, which is a computational bottleneck for Fast-RCNN. Instead, the input image is passed through a convolutional backbone, the feature maps are passed through a Region Proposal Network (RPN) to generate candidate proposals. The last phase follows the same approach as Fast-RCNN (RoI Pooling, Classification and BBox Regression).

2.2.4 Detectron2

Detectron2 [28] is a framework for object detection and segmentation by Meta. It offers many pre-trained models as well as easy training and evaluation of custom models. We build and train our object detection network using this library.

2.3 Hybrid Knowledge Routed Modules for Large-scale Object Detection

In [10], Jiang et al. present a new Faster-RCNN object detection network. The idea is to use semantic relationships between objects in order to improve detection on categories which are less present in the training set, or objects that obfuscated in the input image.
The network uses a relationship graph in order to transform object features extracted from a backbone network into contextualized object features, therefore containing information from other features present in the image. Given a set of R region proposals (obtained from an RPN branch) and a set $\{f_r\}_{r=1}^{R}$ of feature vectors for each region obtained from a backbone encoder (for e.g, ResNet [6]) followed by RoIPooling, a number of Explicit and Implicit Knowledge Routed Modules are defined. In our network, we do not use the implicit modules, therefore we explain the general idea but omit the details.

2.3.1 Explicit Relationship Module

Explicit relationship modules encode some form of prior semantic relationships between object classes. For N object classes, we have a graph $G =< V = |N|, E >$ where E is a symmetric matrix and $e_{ij} = (E)_{i,j}$ encodes the specific semantic similarity between class i and j. Note that the background (no-object) class is included in the N classes.
For the R region proposals, we have the ground truth class for each region, this is done is essence by a matching of each proposal to the closest ground truth region.
Having R regions proposals, the authors build a graph $G^* =< V^* = |R|, E^* >$ where each edge e_{ij}^* is:

$$e_{ij}^* = MLP(|f_i - f_j|) \quad \text{where MLP is a Multi-Layer Perceptron} \tag{2.9}$$

Therefore each module is parameterized with a Feed-Forward neural network. This network is trained on the Mean Squared Error (MSE) with the ground truth weights e_{ij}.

Feature Transformation

The matrix E^* is row-normalized (row-wise softmax) into $\hat{E}$. Let F be the matrix with $\{f_r\}_{r=1}^{R}$ in its rows and $F^{'}$ the transformed features. Then:

$$F^{'} = \hat{E}FW \tag{2.10}$$

Where $\hat{E} \in \mathbb{R}^{R\times R}$, $F \in \mathbb{R}^{R\times D}$, $W \in \mathbb{R}^{D\times S}$, D is the output dimension of the RoIPooling layer, and W is a trained linear transformation to the desired output dimension S. The interpretation is that each feature vector is transformed to a convex combination of all other feature vectors in the image, where the combination is weighted by the similarity graph, followed by a linear

transformation to a set output dimension.

2.3.2 Implicit Relationship Module

The Implicit Relationship Modules attempt to capture information about object relationships which isn't encoded through prior knowledge. This module is similar to the explicit module, except that there are no ground truth labels e_{ij}. Therefore these are trained with backpropagation on the final loss of the network.

2.3.3 Final Classification

The transformed features for each region from each Relationship Module are concatenated into one single vector. This vector then goes through usual classification and BBox regression branches, similar to Fast-RCNN,

2.3.4 Knowledge Matrix Extraction

The authors extract the explicit Knowledge matrices from co-occurrence statistics on the Visual Genome (VG) dataset [12]. Two types of knowledge graphs are extracted.

Attribute Knowledge Matrix

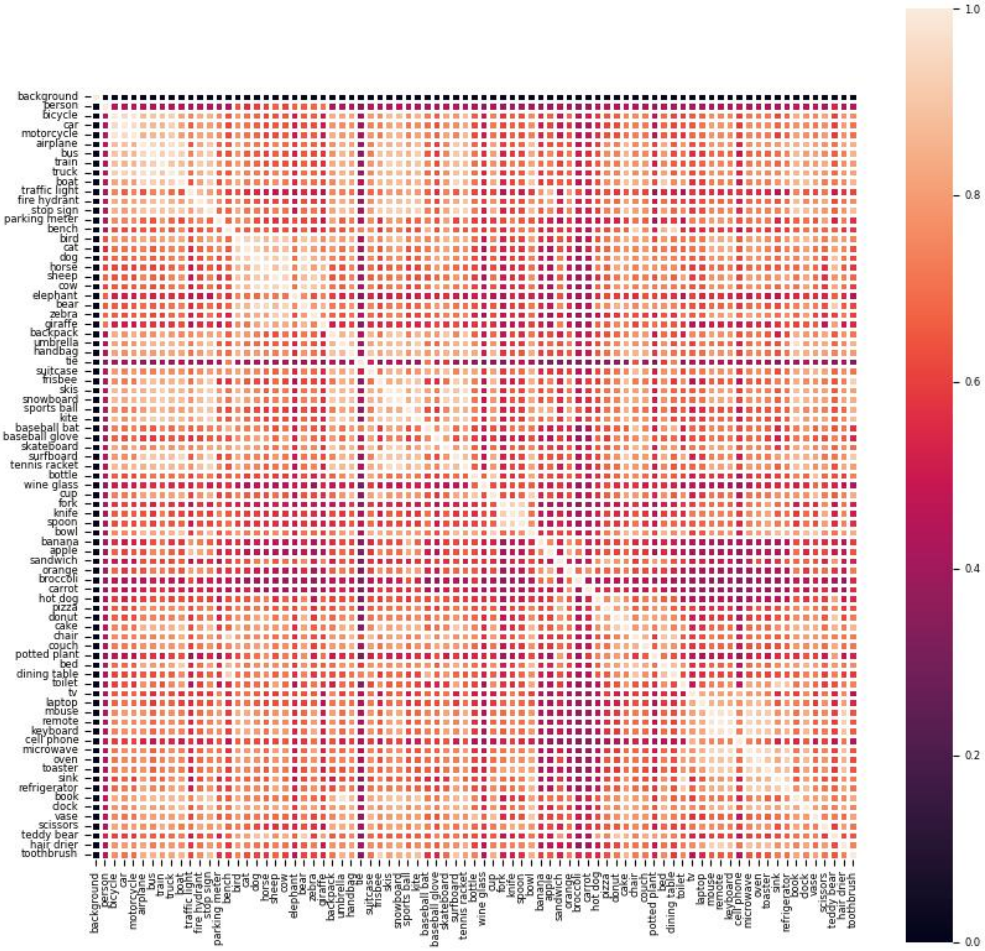
The attribute-based matrix is concerned with the co-occurrence of object attributes (such as color or material) between the different classes. Let K be the number of attributes considered and N the number of classes. Though simple counting statistics, an $C \in \mathbb{R}^{N \times K}$ co-occurrence matrix is obtained. This matrix is normalized and each C_i is regarded as a probability distribution over the K classes. Then:

$$e_{ij} = JS(C_i||C_j) = JS(C_j||C_i) = e_{ji} \quad 1 \leq i, j \leq N \tag{2.11}$$

Where JS denotes the Jensen-Shannon divergence.
Figure 2.3 shows a heatmap of the extracted matrix. We see that the background class has 0 similarity to all other classes, which is expected. Notice that the Person class is associated to almost all other classes. Figure 2.4 shows the class with maximum attribute similarity for each object class as a graph. Here we can see clearly how the knowledge matrix captures the semantic similarity between object classes. For example, the categories: [airplane, truck, bus], [motorcycle, bicycle, car], [suitcase, handbag], etc.

Relationship Knowledge Matrix

The relationship-based matrix is obtained in the same fashion as the previous attribute based matrix but uses the most frequent VG class relationships (such as location, subject-verb) instead.



,

Figure 2.3: Heatmap of the Relationship Attribute matrix. The diagonal elements are the maximum for each line. Note that the matrix is symmetric.

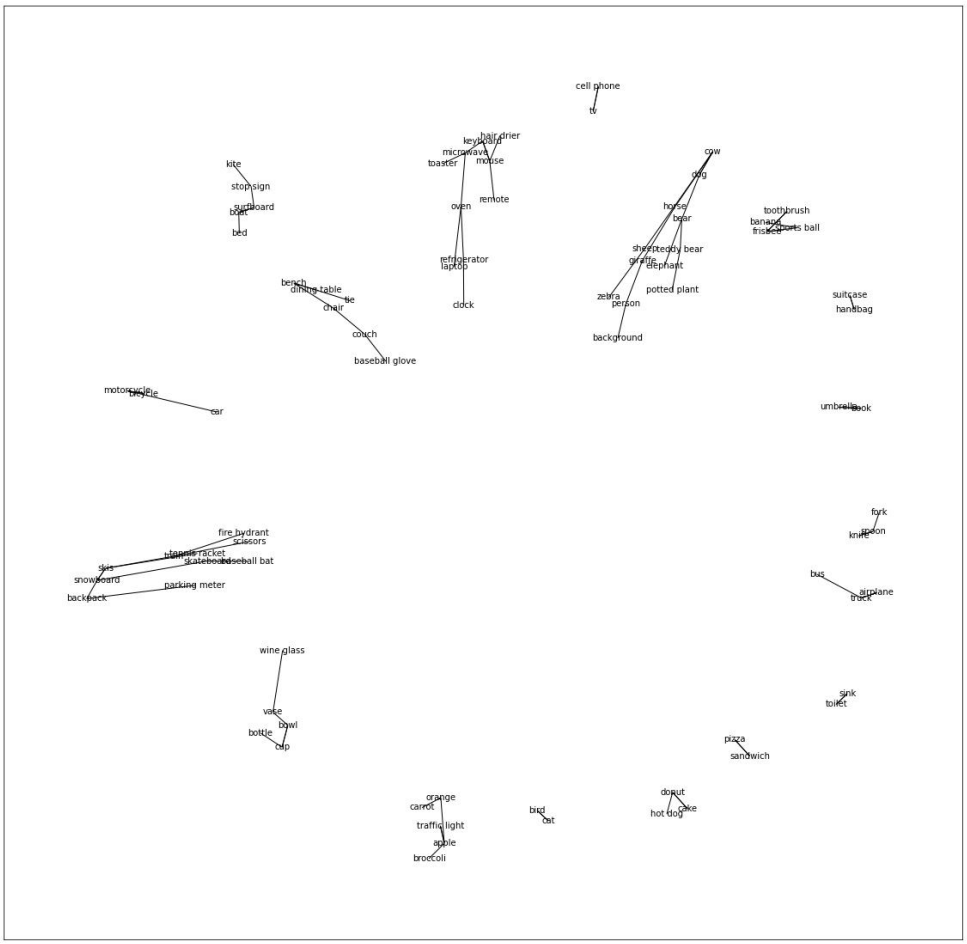


Figure 2.4: Maximum similarity for each object class (other than the same class itself).

Chapter 3

Models

3.1 HKRM modified Faster-RCNN

Because we were not able to use the HKRM code [8] as is, we retrain a model on the Microsoft COCO [15] dataset. Our goal is not to train a state-of-the-art object detection network but rather to obtain contextualized object features in the image. Therefore we retain only the explicit relationship knowledge modules from the original paper. The MSCOCO object classes are a strict subset of the VG classes. The authors extract sub-matrices for the COCO dataset and make them available publicly. We use those as ground truth edge weights in the model.

3.1.1 Model

The model is very similar to the original HKRM model. We use ResNet101 as the backbone feature extractor. The RPN is a Feature Pyramid Network pooler [14] (FPN). Instead of pooling at a single resolution as is done in the original Faster-RCNN paper, the FPN performs the pooling operations on feature maps at many different scales (in our case 4 scales). Therefore, the network uses the 4 ResNet101 feature maps to construct 4 RoI feature boxes for each region. The second difference with the original model is that instead of using the output of the RoIPooling layer as is, these features are passed through a FFN network in order to learn a lower dimensional representation before going through the feature transformation networks. The whole operation may be understood easier with equations:

$$f_1, f_2, f_3, f_4 = ResNet(I) \tag{3.1}$$

where I is the input image and f_i denotes the features maps at scale i.

$$r_{i=1}^{N} = RPN(f_1, f_2, f_3, f_4) \tag{3.2}$$

N proposals r are generated. Then for each proposal r:

$$s_i^r = FPN(f_i) \quad 1 \leq i \leq 4 \tag{3.3}$$

where s_i^r is feature for region r as scale i.

$$f_r = FFN(flatten(concat(s_1^r, s_2^r, s_3^r, s_4^r))) \tag{3.4}$$

f_r is feature vector for region r. It is passed through both the attribute and relationships knowledge modules:

$$f_r^{rel} = REL(f_r) \quad f_r^{attrib} = ATTRIB(f_r) \tag{3.5}$$

Finally:

$$f_r^* = concat(f_r^{rel}, f_r^{attrib}) \tag{3.6}$$

$$class = FFN(f_r^*) \tag{3.7}$$

$$bbox = FFN(f_r^*) \tag{3.8}$$

3.1.2 Implementation

As stated previously, we use Meta's detectron2 object-detection network in order to simply the model building code. Both the relationship and attribute knowledge modules have hidden layer sizes $256, 128, 64$ which are the sizes from the original paper . The activation function used is ReLU. $W \in \mathbb{R}^{64\times512}$ where W is the linear transformation matrix from Equation 2.10. Therefore the final feature vector f_r is in $\mathbb{R}^{1024}$. This is the same dimension as the concatenated multi-scale features output from the RoI Pooling layer.
Because the pairwise L1 difference operation can take a considerable amount of GPU memory, we follow the HKRM paper and set the number of region proposals to 128.

3.1.3 Training

We train the model on the COCO2017 Detection dataset. The set of losses that are backpropagated are:

- The MSE loss from the Relationship Knowledge network
- The MSE loss from the Attribute Knowledge network
- The Cross-Entropy loss for the classification
- the BBox regression loss

Table 3.1: Performance comparison with the baseline network

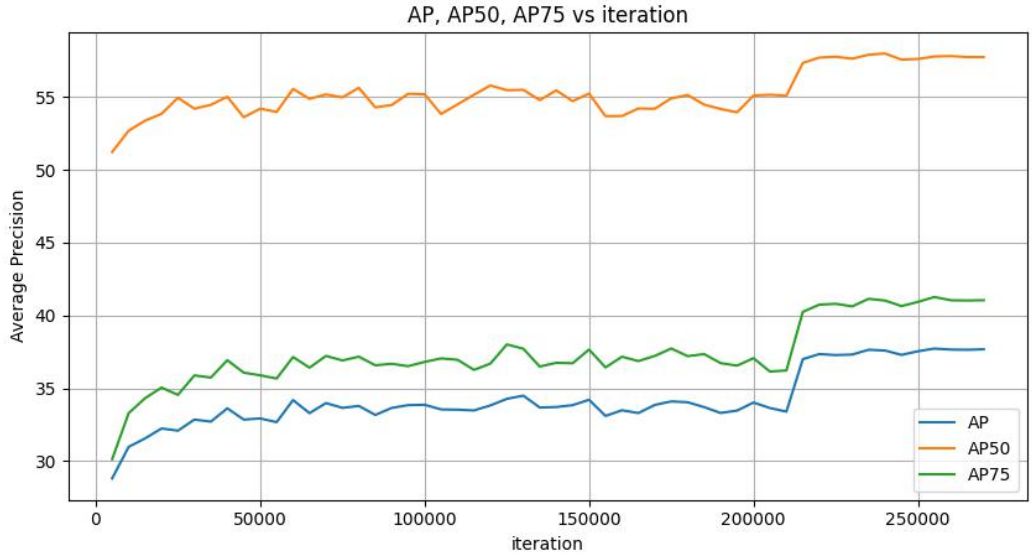
	AP	AP50	AP75
Baseline	42	62.4	45.8
Modified HKRM	37.7	57.7	41.2
Difference	-4.3	-4.7	-4.6

We initialize the weights from a pretrained detectron2 model which is exactly the same as the previously described model except that the two relationship knowledge modules steps are removed. We freeze the backbone ResNet weights due to GPU memory constraints.
We use SGD [22] with a learning a rate set to 0.01, momentum set to 0.9 and weight decay of 10^{-4}. We train for 270000 iterations with a batch size of 10, which corresponds to 23 epochs on the COCO 2017 training set. The training took 4 days on a single GPU. It is worth mentioning that the model diverges for a learning rate of 0.02, which is the detectron2 default.

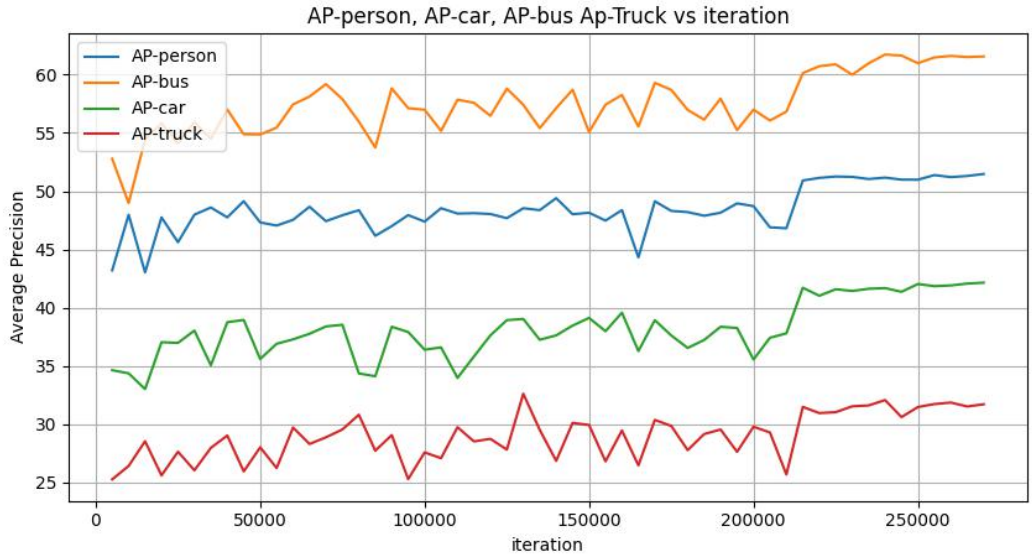
3.1.4 Evaluation

Comparison with a baseline network

We evaluate our model on the COCO2017 test set every 10000 iterations. Figure 3.1 shows the result of this evaluation. It is interesting that The AP curves seems to follow the same trend, but shifted along the y-axis. The AP seems to have stabilized at iteration 270000, where the training stopped.
We compare our model performance to the baseline model from which the weights have been initialized (same model without the relationship modules). Table 3.1 summarizes the result of this comparison on three metrics: AP, AP50 and AP75. It seems that the modified HKRM model is outperformed by the baseline network.We hypothesize some reasons why this may have happened.
One reason may be the freezing of the encoder weights. Another approach would have been to set a very small learning rate (for e.g, 10^{-4}) on its parameters instead.
The second reason might be the number of region proposals output by the RPN. The baseline detectron2 model uses 2000 region proposals at train time, which is considerably greater than the 128 we set for our model.
Finally, we note that it may be that we haven't trained the model enough. Figure 3.1 shows a strong stagnating tendency for the first 210000 iterations followed by a sudden jump in performance. This is further justified by the fact that we use a batch-size of 10 vs the default of detectron2 which is 16 (due to GPU memory constraints).



(a) AP, AP50, AP75 vs iteration evaluation of the COCO 2017 test set



(b) AP for the some of the most common classes vs iteration

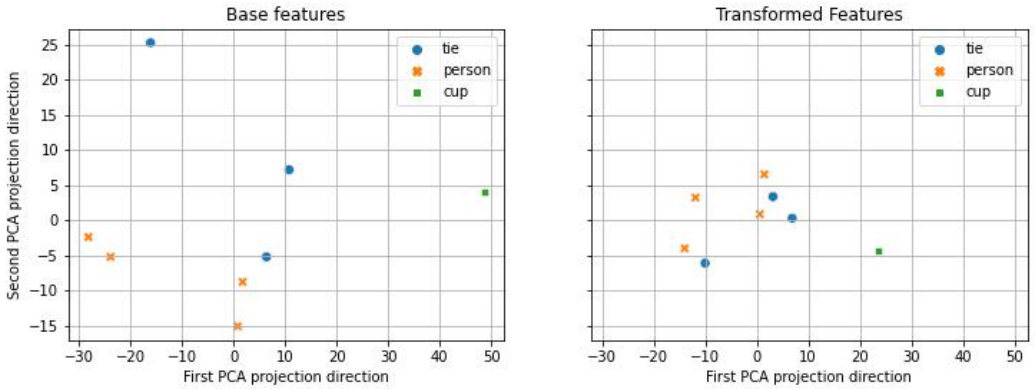
Figure 3.1: COCO 2017 evaluation

Comparison with the original HKRM model

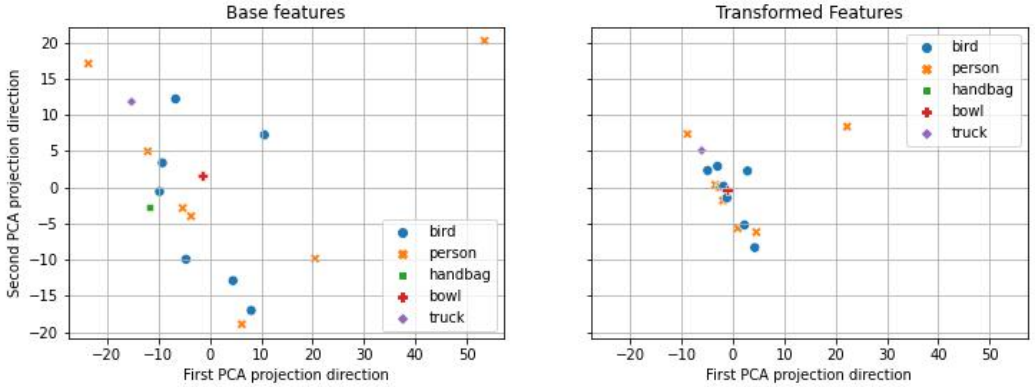
The modified HKRM reaches almost the same performance compared to the original HKRM model. The original model reaches an AP of 37.8 vs our 37.7. This is interesting since it shows that the implicit knowledge modules may not greatly influence the performance. Therefore, it may be that using the Feature Pyramid Network as a pooling layer to obtain multi-scale base features compensates for the lack of implicit relationship modules.

3.1.5 Visualizing the feature transformations

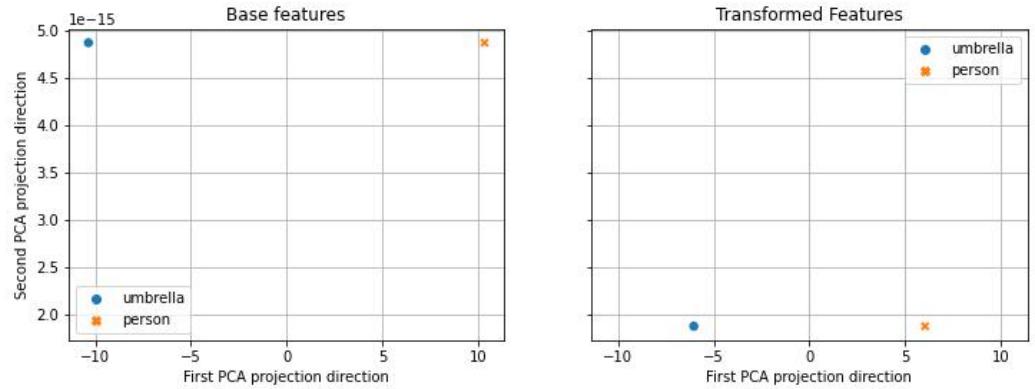
In order to get some intuition on how the feature vectors are transformed by the knowledge routed modules, we use Principal Component Analysis (PCA) on the features before and after the knowledge routed transformation (f_r from Equation 3.4 and f_r^* from Equation 3.6 respectively). We sample 10 images at random from the COCO 2017 training set, and set the detection threshold to 0.3 in order to obtain more detections per image. This is acceptable even if the precision is affected, since are interested in knowing how the features are transformed with respect to the class predicted by the model, not necessarily the ground truth class.
Figure 3.2 shows the result of this evaluation. We notice that in almost all cases the vectors are closer to each other on the whole, meaning that the pairwise distances are reduced. This is to be expected, since in essence the model transforms each vector into a convex combination of all vectors (itself included). We see as well how the relationships between the object classes affects the feature transformation. Points which are semantically related seem to be made closer overall.



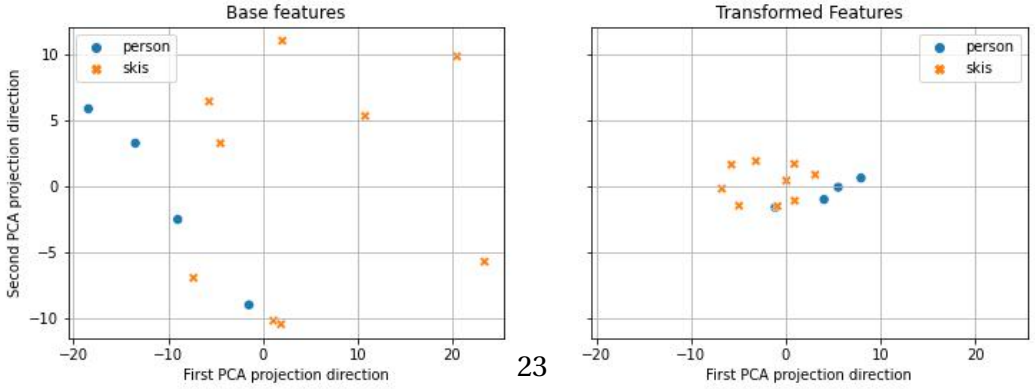
(a) We see that Person is closer to Tie than Cup. Tie and Cup are objects so we have some kind of Person-Tie-Cup ordering.



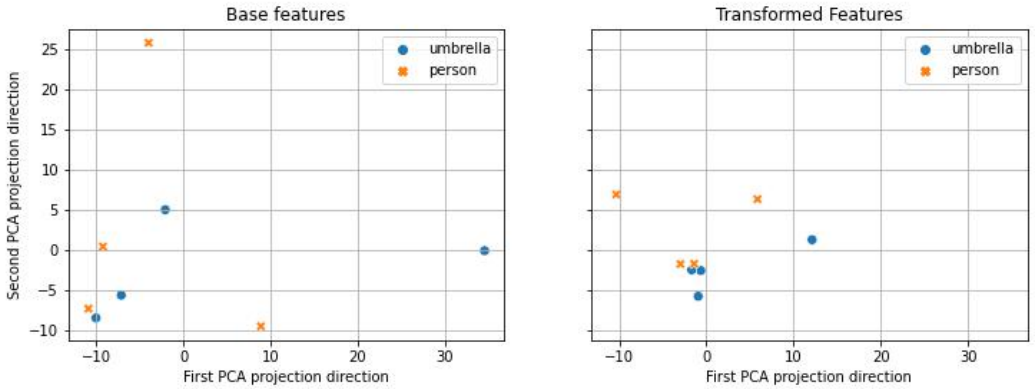
(b) All points are squeezed into a tight layout. The top-right Person vector is made closer to this center.



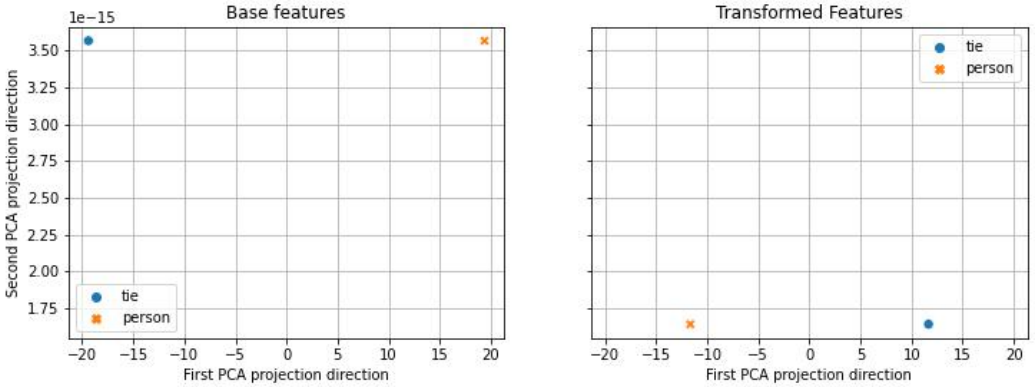
(c) The two points are closer. Semantically this makes sense.



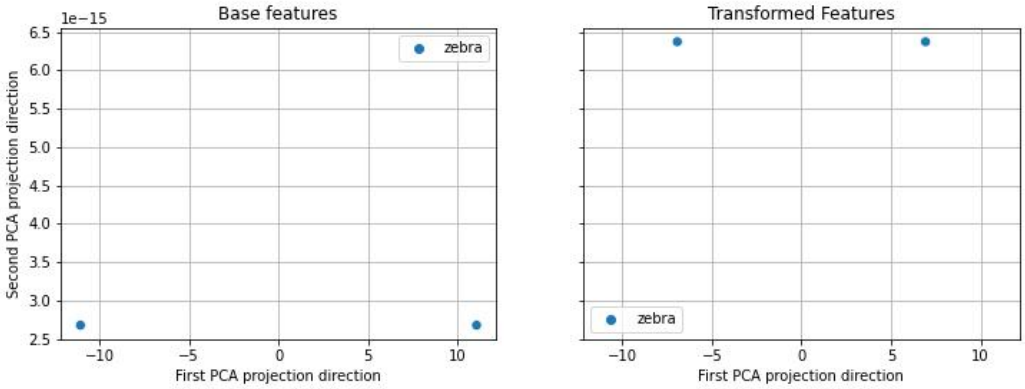
(d) Ski and Person points are grouped into distinct groups and made closer



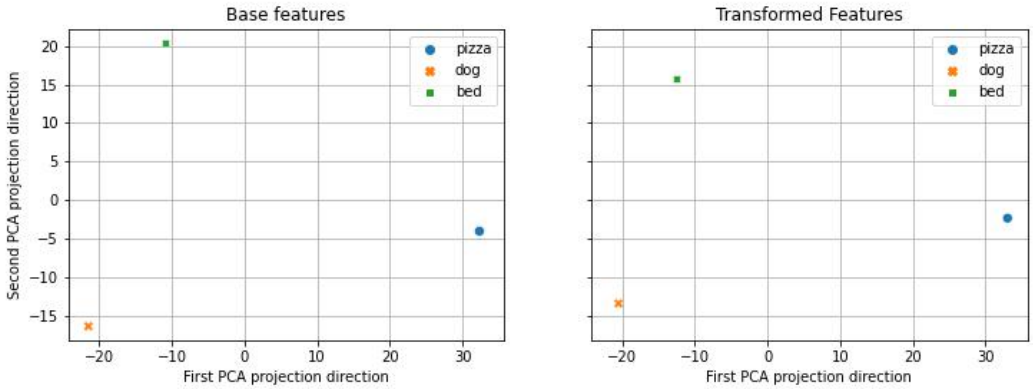
(e) Points are closer to each other. Umbrella is often associated to a Person in location.



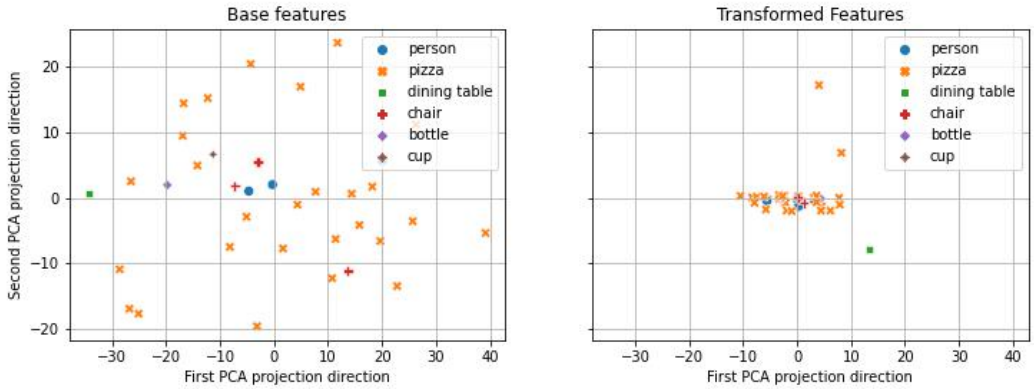
(f) Similar to case (c). Tie is made closer to Person



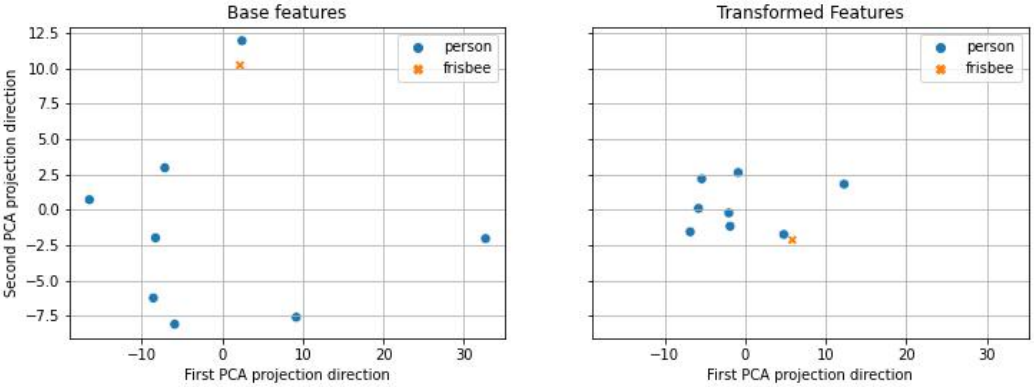
(g) Same class points are made closer.



(h) Interesting case. It seems there is not much contextualization happening



(i) All points are grouped at the center. We can understand this semantically.



(j) Person points grouped together

Figure 3.2: PCA on 10 samples from COCO2017 training set. Left: Base features, Right: Contextualized features

3.2 MidasHKRM

The second model we train is a monocular depth estimation model based both on Midas and the HKRM neural network. We call this model **MidasHKRM**. MidasHKRM uses object features and masks as extra feature maps for the decoder of the original Midas model.

3.2.1 Model

Given an input image I, the model uses an object-detection network (such as the modified HKRM) in order to extract feature vectors and masks for the detected objects in the image:

$$F, M, C = OBJ(I) \tag{3.9}$$

where F denotes a matrix with feature vectors as rows i.e, $F \in \mathbb{R}^{K \times D}$, K is the number of detected objects and D is the object feature dimension, M denotes a tensor of K masks, one for each detected object. C is an $\mathbb{R}^K$ vector of the corresponding predicted classes . OBJ denotes any object detection network that can provide feature vectors as well as object masks. We denote by n the maximum number of object features that the model uses. This is a hyper-parameter of the model, and needs to be fixed. Features and masks are clipped and padded to first dimension equal to n. This means that we discard items after the n-th item, and if OBJ does not detect enough objects we pad the masks and features with zeros.

$$F = pad(clip(F, n), n) \tag{3.10}$$

$$M = pad(clip(M, n), n) \tag{3.11}$$

Then, given a backbone encoder (we use ResNet101), multi-scale features are extracted from the feature maps:

$$f_1, f_2, f_3, f_4 = ResNet(I) \tag{3.12}$$

These feature maps are lower and lower in resolution but with more and more feature maps. A convolutional sub-network is used in order to map these features into the same number of channels; the number of features maps of f_1 which in the case of ResNet101 is 256:

$$f_1^*, f_2^*, f_3^*, f_4^* = CONV1(f_1), CONV1(f_2), CONV1(f_3), CONV1(f_4) \tag{3.13}$$

We reshape each vector into a square image (D is assumed to be a perfect square) and bi-linearly interpolate it into the 4 different resolutions. Each mask in M is similarly interpolated into the 4 resolutions output by the decoder. We concatenate the masks and the features as extra channels:

$$F^{res_i} = interpolate(reshape(F, (\sqrt{D} \times \sqrt{D}), res(f_i))) \quad 1 \leq i \leq 4 \tag{3.14}$$

$$M^{res_i} = interpolate(M \times C, res(f_i))) \quad 1 \leq i \leq 4 \tag{3.15}$$

We multiply each mask with the corresponding object class in order to give the model information about about the type of detected object.

$$r_i = concat(f_i^*, F^{res_i}, M^{res_i}) \quad 1 \leq i \leq 4 \tag{3.16}$$

Finally a RefineNet [3] based decoder (followed by an output convolutional layer) is used in order to produce the final disparity map:

$$p_4 = RefineNet(r_4) \tag{3.17}$$

$$p_i = RefineNet(r_i, p_{i+1}) \quad 1 \leq i \leq 3 \tag{3.18}$$

$$O = OUTPUTCONV(p_1) \tag{3.19}$$

where O denotes the final disparity map.

3.2.2 Implementation

We use PyTorch in order to implement the model. Many of the building blocks use the Midas paper code [19] for e.g, the RefineNet decoder. The training code of Midas was not made publicly available. Therefore we re-implement it from scratch.

3.2.3 Training

We use the loss function from Equation 2.8 like the original Midas paper. We train on the 6 available datasets from MIX6. For each dataset we hold out 100 images as a test set.
The weights for the object detection network as well as the backbone ResNet are frozen.
We use Adam [11] with two different learning rates lr_{old} for the $CONV1$ sub-network (Equation 3.13), and lr_{new} for the decoder. The betas are set to 0.9 and 0.999.
We call the probability threshold for object detection c, and recall that n denotes the maximum number of objects the model uses. Weights are initialized from the best performing Midas (Midas V2.1) model, wherever possible and from a normal distribution otherwise.
The batch size is 6 due to GPU memory constraints. This means that we only sample 1 image from each dataset for each iteration.

3.2.4 Hyperparameters and the Object Detection Backbone

Due to an implementation bug, the first model we train only updated the weights from the $CONV1$ sub-network. The other weights come either from the Midas pretrained model or are random. Although this model was due to a mistake, it turns out to be interesting for comparison

Table 3.2: MIX6 datasets: 6 out of the 10 datasets were used for training

	lr_{old}	lr_{new}	n	c	backbone
MidasRandom	1e-5	N/A	20	0.4	HKRM
MidasHkrmV2	1e-5	1e-4	20	0.4	HKRM
MidasHkrmV3	1e-4	1e-3	16	0.3	HKRM
MidasHkrmV4	1e-6	1e-5	15	0.5	HKRM
MidasBase	1e-6	1e-5	15	0.5	Baseline

Table 3.3: Different trained models

with the others. We call this model **MidasRandom**.
Table 3.2 summarizes the different models trained.
MidasHkrmV2, **MidasHkrmV4** and **MidasBase**'s hyperparameters were set manually. MidasHkrmV4 and MidasBase use a detection threshold of 0.5 that corresponds to the best AP for the object detection network.
MidasBase uses the baseline detection network instead of the modified HKRM model. It was trained in order to test the usefulness of contextualized features versus independent ones.
Finally, **MidasHkrmV3**'s hyperparameters were found by evaluation on a held out validation set of 360 images from each dataset after training for 1000 iterations.

3.2.5 Data Augmentation

We follow the guidelines of Ranftl et al. for data transformation and augmentation.
At train and test time, the input image is resized so that its shortest edge is equal to 384 and its longest edge is a multiple of 32, this is a requirement of the ResNet encoder and means that images are not necessarily resized to the same shape.
Finally, the image is normalized (mean and standard deviation) using the same values as the original Midas code. At train time, images are randomly flipped horizontally (with a probability of 0.5), and randomly resized and cropped into a square patch of side 384 (with a probability of 0.3). This is done in order to augment the training dataset, and give the model a way to learn useful inductive biases.

3.2.6 Training and test losses

Due to time and GPU availability constraints, and more importantly the implementation bug mentioned earlier, we were only able to train each one of the previous models for 70000 iterations. Figure 3.3 plots the train and test losses for all 5 variants of the model.
We can see that train losses start low. This is to be expected since the majority of weights were initialized from the pretrained model. The training losses stay low but we notice that for most models the training is very noisy. This may be due to the very small batch size, which leads to

noisy gradient estimates. The training loss for MidasRandom is almost constant since a very small amount of parameters is updated at each iteration. We note that The training of MidasBase seems to be less noisy than the other models. We hypothesize that this may be due to better object detection on the input images.
For the test loss, we notice that it is low but almost constant. For all models, we notice a decreasing tendency which may indicate that the models were not trained long enough to reach their potential.

3.2.7 Zero-shot Evaluation on the NYU dataset

In order to test the zero-shot performance of the models, we use the NYUv2 [17] test split (654 images).

Criterion

In depth space let z_i^* be the ground truth depth for a pixel and z_i the predicted depth. We evaluate the percentage of pixels where $\delta = max(\frac{z_i}{z_i^*}, \frac{z_i^*}{z_i}) > 1.25$. Following [20], we cap the predicted depth at 10, since this is the max depth for the NYU dataset.

Recovering depth from disparity

Since the output of out model is in disparity space and the ground truth in depth space, we follow the same procedure as the original authors, where we use a least scales criterion to find the best scaling and shift constants. Specifically let G^* be the ground truth depth map and P the predicted disparity.

$$D^* = 1/G^* \tag{3.20}$$

$$L(s,t) = \frac{1}{M}\sum_{i=1}^{M}(sd_i + t - d_i^*)^2 \tag{3.21}$$

Where M is the number of valid ground truth pixels. This can solved in the least squares sense in order to find s^* and t^*. Finally:

$$D_i = \frac{1}{s^* D_i + t^*} \tag{3.22}$$

Where D is the final predicted depth map.

Table 3.4: Zero-shot evaluation results of NYU

	MidasV2.1	MidasHkrmRandom	MidasHkrmV2	MidasHkrmV3	MidasHkrmV4	MidasHkrmBase
$\delta > 1.25$	8.73	23.86	99	99	46.5	46.4

Evaluation Results

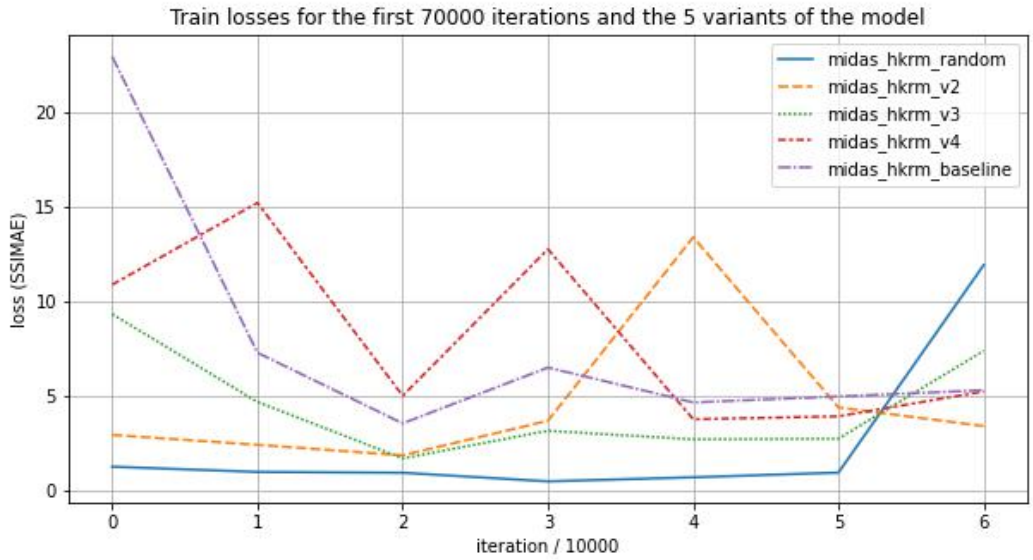
Table 3.4 summarizes the results of this evaluation.
First, we note that the performance is **very bad** compared to the original Midas model. Surprisingly, MidasRandom achieves the best performance out of the 5 models. This is interesting since a good portion of the weights in the decoder are randomly initialized. MidasHkrmV2 and MidasHkrmV3 have almost error=1 performance. MidasHkrmV4 and MidasBase have reasonable performance. We explore ideas and hypothesizes on why we observe this behavior.

- First, we think that all models except MidasRandom may not have been trained nearly enough. The weight updates for MidasRandom are extremely small so the performance will most likely stay the same if we train further. However, for the other models, given that the test set loss is decreasing, it may be that MidasHkrmV2 and MidasHkrmV3 are stuck in a local maximum of the loss function, whereas MidasHkrmV4 and MidasBase are slowly reaching a better performance.
- This behavior may also be due to the learning rates. The learning rates for MidasHkrmV2 and MidasHkrmV3 are one order of magnitude higher than the other models. If we add to this the very small batch size, it may be that each iteration is updating "too much" is a very noisy direction.
- We note however that setting a very small learning rate makes the weight updates very slow, and so training may require more iterations.
- We notice also that the quality (AP) of the object detection backbone may influence the learning. Models for which the detection backbone has a high threshold seem to train better overall.
- Another potential reason for this behavior is the freezing of the encoder weights. The model may benefit from updates to the ResNet backbone parameters. Perhaps also to the object detection model parameters.

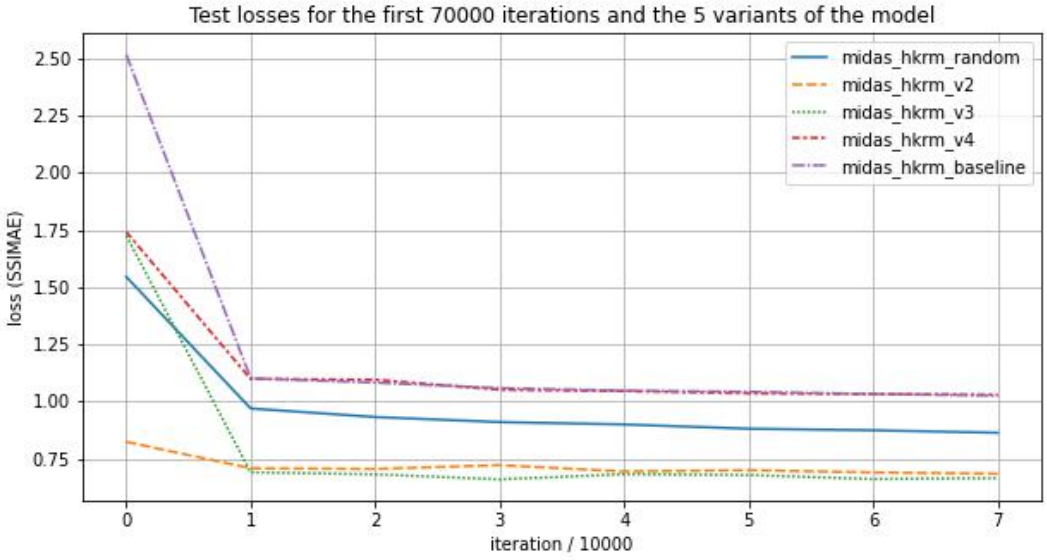
3.2.8 Comments on the overall model design

We cannot conclusively reject the hypothesis that object detection may be helpful to monocular depth estimation. However, we give some ideas for improvement on the model design.
First, we note that in order to have good zero-shot performance, and even good training and test set performance, it is needed that the object detection backbone performs well on samples

from unseen datasets. This may not be true in our case, as the model was trained only on the 80 object classes from the COCO dataset. An idea would be to train an object-detection model on multiple datasets each with their own biases, or to train on a bigger class space like the Visual Genome classes.
Second, the way that the object-features fusion is done may be improved. For example, instead of padding with zeros when there are not enough detections, an idea would be to lower the threshold until a set number of objects is output by the model.
Finally, another idea would be to train both the object detection model and the depth estimation model together. Although this requires a dataset annotated for both cases.



(a) Train losses



(b) Test losses

Figure 3.3: Train and test losses for the 5 different models

Chapter 4

Conclusion

Using high-level image features, such as object detection or segmentation may be the next step for improving the performance of state-of-the-art zero-shot monocular depth estimation.
In this work, we explore the idea of using semantically contextualized object features for this purpose.
Our primary findings show that the used technique hinders the performance of current SOTA models rather than improves it.
However, we think the idea is still worth exploring, either in different architectures or through further training and experimenting.

Bibliography

[1] Jaehoon Cho, Dongbo Min, Youngjung Kim, and Kwanghoon Sohn. "DIML/CVL RGB-D Dataset: 2M RGB-D Images of Natural Indoor and Outdoor Scenes". In: *CoRR* abs/2110.11590 (2021). arXiv: 2110.11590. URL: https://arxiv.org/abs/2110.11590.

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *CoRR* abs/2010.11929 (2020). arXiv: 2010.11929. URL: https://arxiv.org/abs/2010.11929.

[3] Keren Fu, Qijun Zhao, and Irene Yu-Hua Gu. "Refinet: A Deep Segmentation Assisted Refinement Network for Salient Object Detection". In: *IEEE Transactions on Multimedia* 21.2 (2019), pp. 457–469. DOI: 10.1109/TMM.2018.2859746.

[4] Ross B. Girshick. "Fast R-CNN". In: *CoRR* abs/1504.08083 (2015). arXiv: 1504.08083. URL: http://arxiv.org/abs/1504.08083.

[5] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *CoRR* abs/1311.2524 (2013). arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[8] *HKRM code repository*. URL: https://github.com/chanyn/HKRM.

[9] Xinyu Huang, Xinjing Cheng, Qichuan Geng, Binbin Cao, Dingfu Zhou, Peng Wang, Yuanqing Lin, and Ruigang Yang. "The ApolloScape Dataset for Autonomous Driving". In: *CoRR* abs/1803.06184 (2018). arXiv: 1803.06184. URL: http://arxiv.org/abs/1803.06184.

[10] Chenhan Jiang, Hang Xu, Xiaodan Liang, and Liang Lin. "Hybrid Knowledge Routed Modules for Large-scale Object Detection". In: *CoRR* abs/1810.12681 (2018). arXiv: 1810.12681. URL: http://arxiv.org/abs/1810.12681.

[11] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[12] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, Michael Bernstein, and Li Fei-Fei. "Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations". In: 2016. URL: https://arxiv.org/abs/1602.07332.

[13] Zhengqi Li and Noah Snavely. "MegaDepth: Learning Single-View Depth Prediction from Internet Photos". In: *Computer Vision and Pattern Recognition (CVPR)*. 2018.

[14] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. *Feature Pyramid Networks for Object Detection*. 2017. arXiv: 1612.03144 [cs.CV].

[15] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. "Microsoft COCO: Common Objects in Context". In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[16] Nian Liu, Ni Zhang, Ling Shao, and Junwei Han. *Learning Selective Mutual Attention and Contrast for RGB-D Saliency Detection*. 2020. arXiv: 2010.05537 [cs.CV].

[17] Pushmeet Kohli Nathan Silberman Derek Hoiem and Rob Fergus. "Indoor Segmentation and Support Inference from RGBD Images". In: *ECCV*. 2012.

[18] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. "Vision Transformers for Dense Prediction". In: *ArXiv preprint* (2021).

[19] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. *Midas github repository*. https://github.com/isl-org/MiDaS. 2020.

[20] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. "Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2020).

[21] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

[22] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[23] Ozan Sener and Vladlen Koltun. "Multi-Task Learning as Multi-Objective Optimization". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., 2018, pp. 525–536. URL: http://papers.nips.cc/paper/7334-multi-task-learning-as-multi-objective-optimization.pdf.

[24] Jasper Uijlings, K. Sande, T. Gevers, and A.W.M. Smeulders. "Selective Search for Object Recognition". In: *International Journal of Computer Vision* 104 (Sept. 2013), pp. 154–171. DOI: 10.1007/s11263-013-0620-5.

[25] Chaoyang Wang, Simon Lucey, Federico Perazzi, and Oliver Wang. *Web Stereo Video Supervision for Depth Prediction from Dynamic Scenes*. 2019. arXiv: 1904.11112 [cs.CV].

[26] Qiang Wang, Shizhen Zheng, Qingsong Yan, Fei Deng, Kaiyong Zhao, and Xiaowen Chu. "IRS: A Large Synthetic Indoor Robotics Stereo Dataset for Disparity and Surface Normal Estimation". In: *CoRR* abs/1912.09678 (2019). arXiv: 1912.09678. URL: http://arxiv.org/abs/1912.09678.

[27] Wenshan Wang, Delong Zhu, Xiangwei Wang, Yaoyu Hu, Yuheng Qiu, Chen Wang, Yafei Hu, Ashish Kapoor, and Sebastian A. Scherer. "TartanAir: A Dataset to Push the Limits of Visual SLAM". In: *CoRR* abs/2003.14338 (2020). arXiv: 2003.14338. URL: https://arxiv.org/abs/2003.14338.

[28] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[29] Ke Xian, Jianming Zhang, Oliver Wang, Long Mai, Zhe Lin, and Zhiguo Cao. "Structure-Guided Ranking Loss for Single Image Depth Prediction". In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[30] Yao Yao, Zixin Luo, Shiwei Li, Jingyang Zhang, Yufan Ren, Lei Zhou, Tian Fang, and Long Quan. "BlendedMVS: A Large-scale Dataset for Generalized Multi-view Stereo Networks". In: *CoRR* abs/1911.10127 (2019). arXiv: 1911.10127. URL: http://arxiv.org/abs/1911.10127.