Joint Object Detection and Depth Estimation

Vahid Shahiri- Student ID : 99301051 Seyed Reza Hosseini- Student ID : 99301016

Abstract

Given an image, the purpose of object detection is to obtain the location and category information of each object instance in it. As an important part of computer vision, object detection has a broad range of applications in many areas such as autonomous driving, robot vision and surveillance system. However, some applications (e.g. autonomous driving) not only need the positions of objects in the image but also require these detected objects actual depth. In this project we utilize the state of the art YOLO v5 object detection network to obtain the category information of the objects and then we pass the resulting bounding boxes to a hybrid CNN (convolutional neural network) to infer the depth value. Our code requires the user to enter a value for maximum target depth so that at the output any object closer than this target value will be labeled. The label of the bounding boxes at the output of our code, shows both the object type and the estimated depth.

I. Introduction

Autonomous cars and robotics are two main applications which can be named when talking about the joint object detection and depth estimation tasks. Since the introduction of AlexNet which exploits the convolutional neural network (CNN) techniques, the performance of image processing algorithms has steadily been increasing. The state-of-the-art methods like YOLO (You Only Look Once) and Fast R-CNN (Fast Region with CNN) are used extensively in various object detection tasks. Each of the aforementioned networks can reach outstanding results and as we are allowed to use the pre-trained models in this project we have chosen to use YOLO-v5 for our object detection task.

To achieve reliable and valid picking of the objects in both robotics and autonomous cars applications, people need to localize them after recognition and detection. Some techniques to achieve these goals use a mono camera, stereo cameras, or a depth camera. However, Using a mono camera is more challenging in the depth determination task[1]. As the NYU-v2 dataset

for depth estimation is based on mono camera RGB images, we face grave challenges in implementing our depth estimation network.

In this project, we first tried to train our own model for the depth estimation task. Due to the hardware limitations that we face in training such a challenging network the obtained results are not impressive. In the next step we used a pre-trained depth estimation model and explored the architecture and performance of it. Then we joined this model with the YOLO-v5 object detection network to obtain the final joint object detection and depth estimation task and qualitatively evaluated the overall performance. Finally, we focus on the various methods that can be deployed in joining two networks and the associated performance metrics.

II. Our Trained Depth Estimation Model

A. Network Structure

In this project we are required to train a depth estimation network which is able to tell the depth of the indoor scenes, through using the NYU-v2 dataset as training set. We started our work on this project by deigning such a deep network. Our architecture is inferred through deploying and combining the methods and techniques devised in [2] and [3]. The implemented structure consists of three main blocks namely, "Downscale", "Bottleneck" and "Upscale". Firstly, the input image is fed to the downscaling part in which the size of the feature maps are reduced with the kernels of the size (3×3). Each layer of this block has $f = [16, 32, 64, 128, 256]$ filters. The output of the convolutions is passed through the "leaky ReLu" function with factor "0.2" and then batch normalization layer is also inserted. Finally, maxpooling with the kernels of size (2×2) is applied.

Then we have the bottleneck block consisting of two convolutional layers with 256 filters which are applied on the output of the previous block or downscale block.

At the end we have the upscale block which does exactly the opposite of what downscale block does. First of all, we have an upsampling applied with a (2×2) kernel. This is followed by two layers of upconvolution with (3×3) kernel sizes. Again we have "leaky ReLu" function with factor "0.2" and the batch normalization layer. Here also the number of filters are 16, 32, 64, 128 and 256. Finally a (1×1) convolution is exerted to fit the output with input. At last, we have used the ReLu activation function as the output values which are images have positive value.

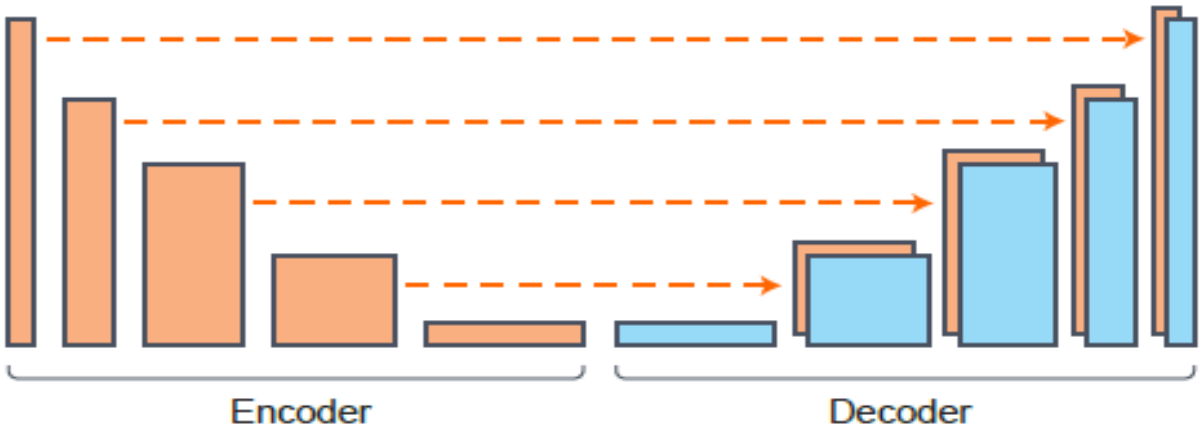

Fig. 1: The general encoder-decoder architecture used in most of the depth estimation methods (we have used the names downscale-upscale in our file).

We also note that the "U-Net" like skip connections are effectively used in the structure. These connections can play a major role in battling the notorious gradient vanishing problem which always occurs in such deep networks.

For the detailed description of what is implemented please refer to the "Our-Implemented-Depth-Estimation.ipynb" file. We have tried to make the code clear and easy to read. Fig. 1 shows the overall structure that we and most of the networks deployed in depth estimation tasks (We note that this is not the exact structure of our implemented network but the general concept is the same). As we need to have an output of the size of the input image, inevitably we have to use encoder-decoder architecture (just like the downscale-upscale blocks of ours...).

B. Loss Function

Defining a suitable loss function in the task of depth estimation is very crucial. A standard loss function for depth regression problems considers the difference between the ground-truth depth map y and the prediction of the depth regression network $\hat{y}$. Different considerations regarding the loss function can have a significant effect on the training speed and the overall depth estimation performance. Many variations on the loss function employed for optimizing the neural network can be found in the depth estimation literature. In our method we seek to define a loss function that balances between reconstructing depth images by minimizing the difference of the depth values while also penalizing distortions of high frequency details in the image domain

of the depth map. These details typically correspond to the boundaries of objects in the scene. For training our network, we define the loss L between y and $\hat{y}$ as the weighted sum of three loss functions:

$$L(y, \hat{y}) = \lambda_1 L_{\text{depth}}(y, \hat{y}) + \lambda_2 L_{\text{grad}}(y, \hat{y}) + \lambda_3 L_{\text{SSIM}}(y, \hat{y}) \tag{1}$$

The first term L_{depth} is the point-wise L-1 loss defined on the depth values:

$$L_{\text{depth}}(y, \hat{y}) = \frac{1}{n} \sum_{p}^{n} |y_p - \hat{y}_p| \tag{2}$$

The second loss term L_{grad} is the L-1 loss defined over the image gradient g of the depth image:

$$L_{\text{grad}}(y, \hat{y}) = \frac{1}{n} \sum_{p}^{n} |g_x(y_p, \hat{y}_p)| + |g_y(y_p, \hat{y}_p)| \tag{3}$$

where g_x and g_y, respectively, compute the differences in the x and y components for the depth image gradients of y and $\hat{y}$.

Lastly, L_{SSIM} uses the Structural Similarity (SSIM) term which is a commonly-used metric for image reconstruction tasks [4]. It has been recently shown to be a good loss term for depth estimating CNNs. Since SSIM has an upper bound of one, we define it as a loss L_{SSIM} as follows:

$$L_{\text{SSIM}} = \frac{1 - \text{SSIM}(y, \hat{y})}{2} \tag{4}$$

An inherit problem with such loss terms is that they tend to be larger when the ground-truth depth values are bigger. In order to compensate for this issue, we have normalized the values of depth matrix in our implemented code.

An interesting point when applying the loss term of (1) in our code was the great role of hyper parameters λ_1, λ_2 and λ_3 in the overall performance of our network. A slight change in each of these parameters (even in the range of 0.01!) has a great impact in the outcome. After numerous experiments we have come to the values of $\lambda_1 = 0.1$, $\lambda_2 = 1$ and $\lambda_3 = 0.9$ for our project. We also note that considering a large value for the L_1 loss will lead to exact reconstruction of the input image which is not a favorable phenomenon.

```
Epoch 1/6
29/29 [==============================] - 32s 491ms/step - loss: 0.3717 - val_loss: 0.7832
Epoch 2/6
29/29 [==============================] - 10s 353ms/step - loss: 0.2108 - val_loss: 0.7341
Epoch 3/6
29/29 [==============================] - 10s 357ms/step - loss: 0.1694 - val_loss: 0.2016
Epoch 4/6
29/29 [==============================] - 10s 359ms/step - loss: 0.1495 - val_loss: 0.1523
Epoch 5/6
29/29 [==============================] - 10s 359ms/step - loss: 0.1358 - val_loss: 0.1101
Epoch 6/6
29/29 [==============================] - 11s 363ms/step - loss: 0.1349 - val_loss: 0.1045
Model: "depth_estimation_model"
```

Fig. 2: A snapshot of the training results for our implemented code for depth estimation

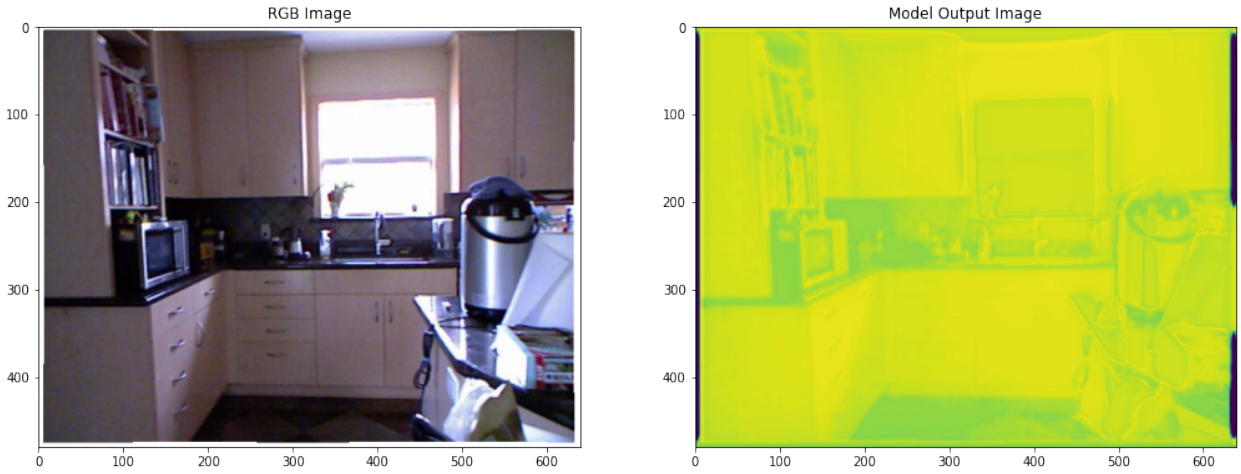

Fig. 3: The qualitative test result for our implemented code for depth estimation

C. Training and Outcome

Fig. 2 shows a snapshot of our training status. A batch size of 32 is used along with the 6 epoches. The values for training and validation losses are decreasing rationally. However, due to limitations in our hardware we could not feed more than 200 images in the training phase. Because of these limitations we also couldn't implement the usual augmentation mechanisms usually used in depth estimation tasks (Like the augmentations in [4]). Consequently, the results in the test phase were not outstanding. Fig. 3 shows a qualitative test result of our trained network. It shows that the overall structure of depth is largely rebuilt, however the values for depth are not perfect.

Finally, we can say that inferring depth from mono camera is a challenging task which requires

a delicate network and parameter design along with good hardware resources to get tangible results. For the joining phase of our project we plan to use a pre-trained network to be able to produce more concrete results at the end. We will describe this network architecture in the next section.

III. Pre-trained Depth Estimation Model: The Hybrid CNN Method

As discussed earlier, due to the hardware limitations that we faced in the process of acquiring a reliable depth estimation method, we could not achieve an acceptable outcome by using our own trained model. Consequently, to have tangible results when combining the two networks, we decided to use a pretrained model for the depth estimation part. In the following we will try to provide some insights into this pretrained depth estimation network (We name it "Hybrid CNN"hereafter).

A. Hybrid CNN Architecture

Actually, there are various methods for the depth estimation problem with promising results. In this project we implemented the network architecture proposed in [5]. This network is a fully convolutional neural net that combines the advantages of deep residual nets [6] and U-net architectures [7]. It is able to achieve state-of-the-art accuracy without significantly increasing the depth and the number of variables compared to the former models. This is achieved by augmenting the CNN from the work of Laina et al. [3] with U-net-like connections between the encoding and decoding parts.

The structure of the hybrid CNN model is described in Fig. 4. The model outlined in [3] can be divided into two parts: an encoding part based on the ResNet50 architecture, and a decoding part consisting of repeated up-projection blocks (see Fig. 5). The aim of the contributors in [5] was to boost the reliability of this model without significantly increasing its depth and the number of its variables.

Inspired by U-net-like architectures, instead of deepening the model by increasing the number of layers, its interconnectedness is enhanced by introducing side-to-side connections into the network flow. These lateral connections are realized by concatenating the corresponding feature maps between the encoding and decoding parts of the model. After each depth concatenation, an

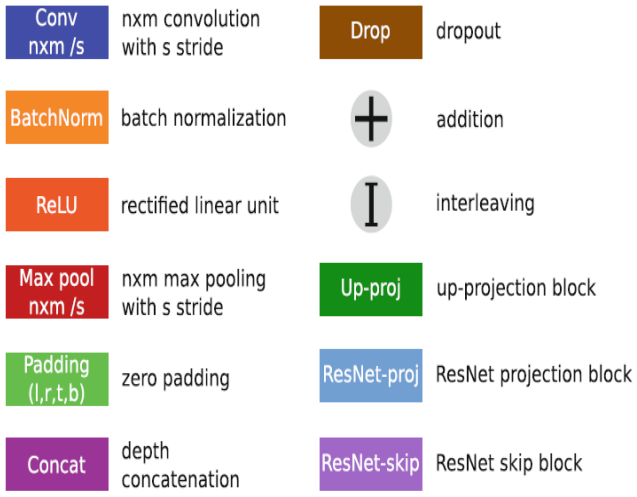

Fig. 4: The representations of the different layers used in our architecture [5]. The definition for the interleaving operation can be found in [3].

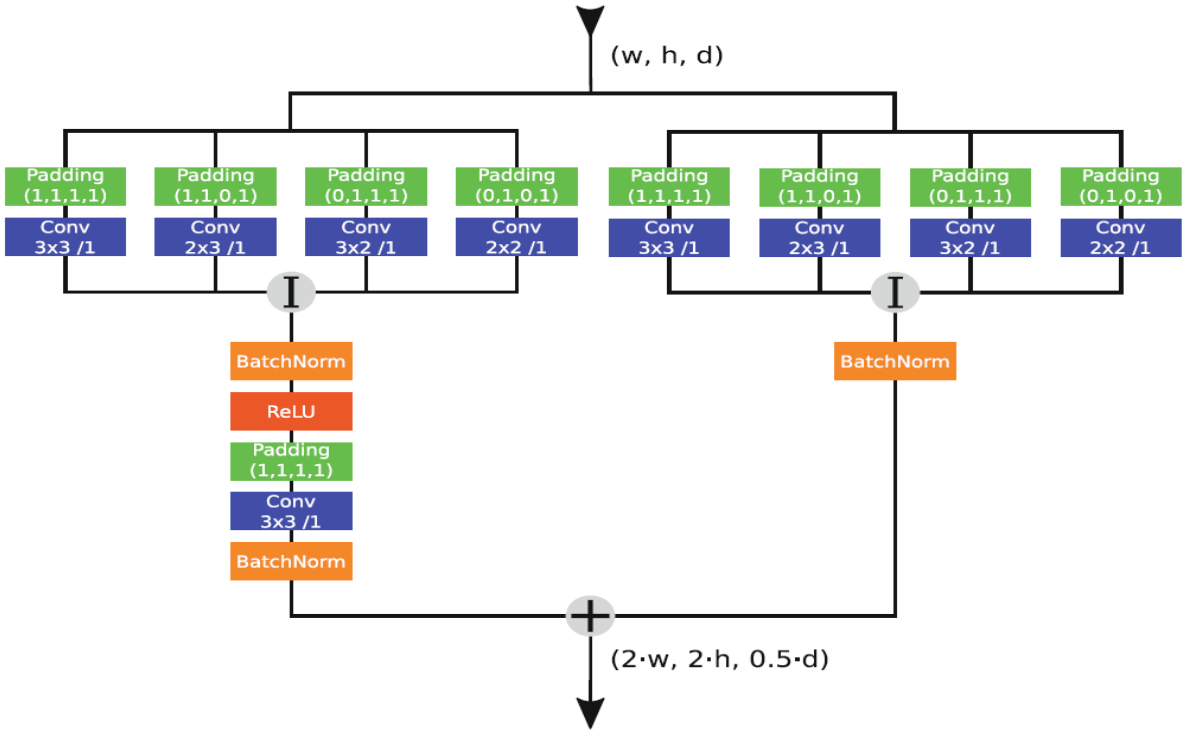

Fig. 5: An up-projection block [3]. This block is equivalent to a residual up-projection, but this version is more efficient and leads to reduced training time.

additional (1×1) convolution is inserted in order to keep the number of input channels of the up-projection blocks intact. Thanks to these connections the network is able to translate additional information from its previous feature maps into its expansive path. The complete architecture of the proposed network is depicted in Fig. 6.

B. Hybrid CNN Training and Augmentation

The model is trained and evaluated on the NYU Depth v2 dataset. This dataset captures 464 indoor scenes with an RGB camera and a Microsoft Kinect. The dataset is split into 215 testing and 249 training scenes. The equallyspaced RGB-D image pairs are extracted from the training scenes of the raw dataset and ended up with roughly 48000 pairs as the training set.

Before training, every RGB-D pair should be resized to 352 264 pixels. This way the image sizes are closer to the input shape of the network and the aspect ratio stays roughly the same. Additionally, the authors used random online augmentation during training. Every RGB-D image pair is:

- scaled by $s \in R[1, 1.5]$, and the depths are divided by s
- rotated by $r \in R[5, 5]$ degree
- randomly cropped down to 320×256 pixels
- the RGB color values are multiplied by a random $c \in R[0.8, 1.2]^3$
- horizontally flipped with 0.5 probability.

The division by s in the scaling step is necessary to preserve the world-space geometry of the scene. In order to conserve the boundaries between valid and invalid pixels on the depth images, Nearest-neighbor interpolation is used for the scaling and the rotation. After the augmentation, the size of the RGB-D pairs is 320×256 pixels. The authors further downscale the depth image to 160×128 pixels to match the output size of network.

C. Hybrid CNN Loss Function

The loss function used here is Huber (BerHu) loss during training instead of the regularly used L2 loss. This concept was first introduced by [3]. The BerHu loss puts a higher weight on the pixels with higher residuals by applying L2 on them. Simultaneously, it allows smaller residuals to have a larger effect on the gradients during training due to the use of the L1 loss.

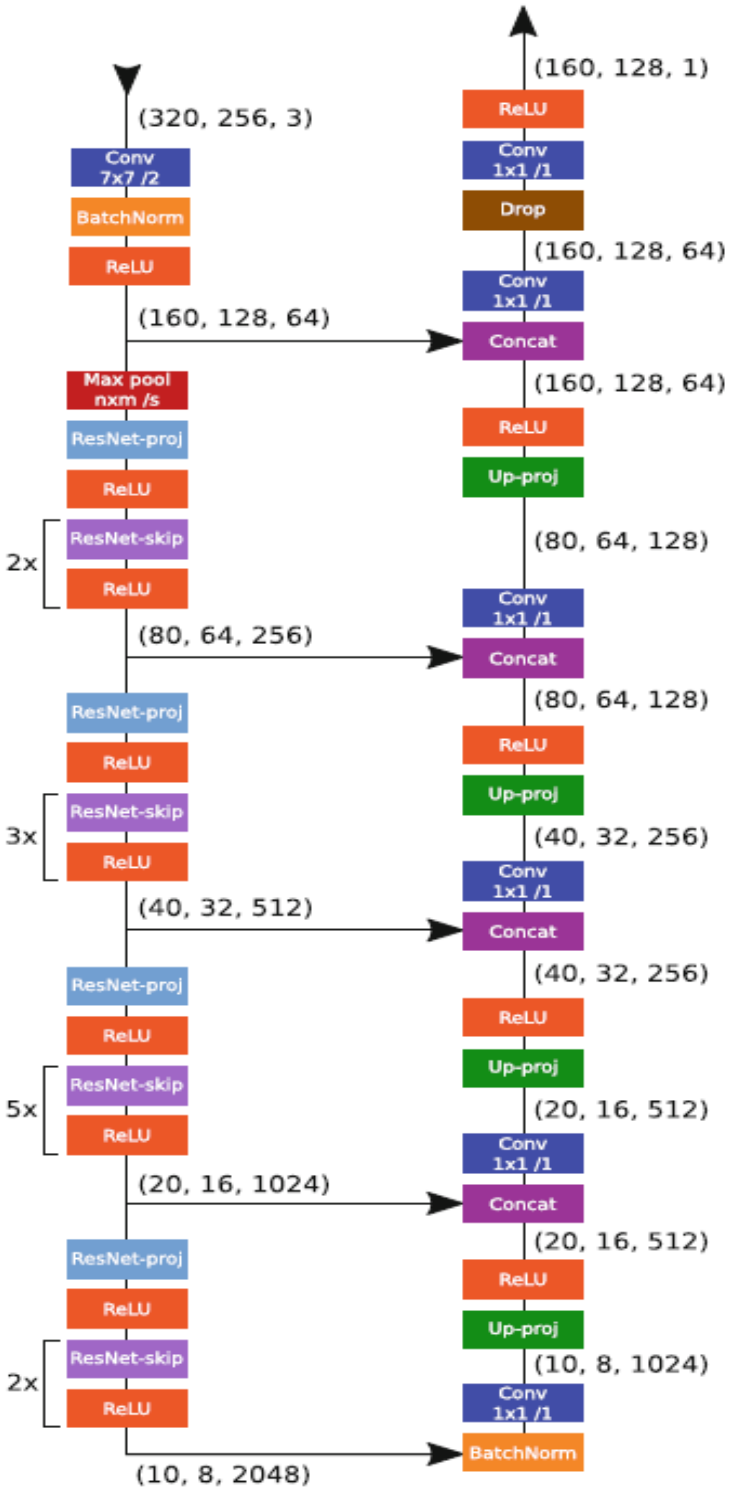

Fig. 6: The illustration of the implemented depth estimation network used in our final code [5].

In order to calculate the BerHu loss $B(y, \tilde{y})$ for a batch of predictions y and the ground truths $\tilde{y}$, we have to compute $c = \frac{1}{5}.\max_i |y_i - \tilde{y}_i|$, where i indexes are over every pixel of every image in the batch y. Once c is calculated $B(y, \tilde{y})$ is defined as:

$$\begin{cases} |y - \tilde{y}|, & where |y - \tilde{y}| \leq c \\ \frac{(y-\tilde{y})^2 + c^2}{2c}, & otherwise. \end{cases} . \tag{5}$$

Therefore, the BerHu loss is equal to the L1 norm on the pixel i where $|y_i - \tilde{y}_i| \in [c, c]$ and equal to L2 outside this interval. The version defined in Eq. (5) comes from [3]. This form is favorable because it is continuous and differentiable in the switch point c.

D. Selected Hyper Parameters for Hybrid CNN

The first half of the network is responsible for encoding the images. This part is identical to the ResNet-50 network architecture. Thus, we can initialize these layers with the ResNet-50 weights pre-trained on ImageNet. The variables in the second half are initialized by random normal distribution with 0 mean and 0.001 standard deviation.

The network is trained for 25 - 30 epochs, with a batch size of 16. The stochastic gradient descent optimizer with 0.9 momentum is used. The initial learning rate is 10^{-2}. After 10 epochs it is halved, and for the final 5- 10 epochs, it is reduced to 10^{-3}.

To prevent overfitting, a dropout layer with a dropout rate set to 0.5 is inserted into the network before the final convolution. Additionally, a weight decay of 0.00025 for every layer in the network is set.

The authors implemented the network architecture in PyTorch, and trained it on an NVIDIA GeForce GTX 1080 Ti Graphics Card. An entire training session took approximately 15 h (This again justifies the performance of our own trained network!). Once the model is loaded into the GPU memory, forwarding an arbitrary image through the network takes roughly 0.01 s (after resizing the picture to match the input size of the network). This means this model is fast enough to be utilized in applications that require real-time performance.

E. Evaluation of Hybrid CNN

For evaluation, the 654 RGB-D image pairs from the labeled test subsets of the NYU Depth v2 is used. The RGB images are resized to 352×264 pixels, and center-cropped , to match

TABLE I: Quantitative comparison with state-of-the-art CNN based methods on the NYU Depth v2 dataset. In the case of RMSE, REL, and Log_{10}, lower is better. For the δ_i accuracies, higher is better.

	RMSE	REL	Log_{10}	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Eigen et al. [8]	0.907	0.215	-	0.611	0.887	0.971
Wang et al. [9]	0.745	0.220	-	0.605	0.890	0.970
Eigen and Fergus [10]	0.641	0.158	-	0.769	0.950	0.988
Laina et al. [3]	0.597	0.137	0.059	0.818	0.955	0.988
Our Hybrid CNN [5]	**0.593**	**0.130**	**0.057**	**0.833**	**0.960**	**0.989**

the input size of the model. The output size of the model is 160×128 pixels, while the size of the ground truth depth maps of the test set is 640×480. To compare the results without distortions, the authors resize the predictions to 582×466 pixels, and center-crop the ground truth depth maps to the same size. For a fair quantitative comparison, the authors apply the same conversions to the output of Laina et al.s model [3]. The rest of the data in Table I is based on the values reported by the authors. The following error metrics on the test set are computed, for quantitative evaluation [8]:

1) average relative error (REL): $\frac{1}{n}\sum_{p}^{n}\frac{|y_p - \hat{y}_p|}{y}$;
2) root mean squared error (RMSE): $\sqrt{\frac{1}{n}\sum_{p}^{n}(y_p - \hat{y}_p|)^2}$
3) average (log_{10}) error: $\frac{1}{n}\sum_{p}^{n}(log_{10}y_p - log_{10}\hat{y}_p|)^2$;
4) threshold accuracy (δ_i): $\%$ of y_p s.t. max($\frac{y_p}{\hat{y}_p}, \frac{\hat{y}_p}{y_p}$)$= \delta < thr$ for $thr = 1.25, 1.25^2, 1.25^3$;

Here y_p is a pixel in depth image y, $\hat{y}_p$ is a pixel in the predicted depth image $\hat{y}$, and n is the total number of pixels for each depth image.

The comparison between hybrid CNN network and other models can be seen in Table I. The test results for the evaluation of the hybrid CNN network run by us, is also shown in Fig. 7 (The images is directly extracted through running the code file "Depth-Estimation-Hybrid-CNN.ipynb" attached to this report.). For qualitative results, see Fig. 8. Both the qualitative and the quantitative evaluation shows a considerable improvement compared the original architecture in [3] as well as the other state-of-the-art approaches.

```
Use GPU: False
Loading model...

Running evaluation:
654 / 654
Mean Absolute Relative Error: 0.130346
Root Mean Squared Error: 0.592839
Mean Log10 Error: 0.056754
Delta1: 0.832582
Delta2: 0.960195
Delta3: 0.989193
```

Fig. 7: The test results of the hybrid CNN network implemented in the joint depth-object detection task

IV. Object Detection Model: YOLO-v5

A. *YOLO-v5*

Yolo is a state-of-the-art, real-time object detector, and Yolov5 is based on Yolov1- Yolov4. Continuous improvements have made it achieve top performances on two official object detection datasets: Pascal VOC (visual object classes) and Microsoft COCO (common objects in context).

Due to its outstanding performance, we decided to use YOLO-v5 network for our object detection part. Accordingly, we plan to provide some insights on its structure and performance in the following subsections.

1) Architecture: The network architecture of Yolov5 is shown in Fig. 9. There are three reasons why we choose Yolov5 as our first learner. Firstly, Yolov5 incorporated cross stage partial network (CSPNet) into Darknet, creating CSPDarknet as its backbone. CSPNet solves the problems of repeated gradient information in large-scale backbones, and integrates the gradient changes into the feature map, thereby decreasing the parameters and FLOPS (floating-point operations per second) of model, which not only ensures the inference speed and accuracy, but also reduces the model size. In some cases, detection speed and accuracy is imperative, and compact model size also determines its inference efficiency on resource-poor edge devices. Secondly, the Yolov5 applied path aggregation network (PANet) as its neck to boost information flow. PANet adopts a new feature pyramid network (FPN) structure with enhanced bottom-up path, which improves the propagation of low-level features. At the same time, adaptive feature pooling, which links feature

Fig. 8: (a) RGB input, (b) Laina et al. [3], (c) Hybrid CNN model [5], (d) Ground truth [5].

grid and all feature levels, is used to make useful information in each feature level propagate directly to following subnetwork. PANet improves the utilization of accurate localization signals in lower layers, which can obviously enhance the location accuracy of the object. Thirdly, the head of Yolov5, namely the Yolo layer, generates 3 different sizes (18 18, 36 36, 72 72) of feature maps to achieve multi-scale prediction, enabling the model to handle small, medium, and big objects. Multi-scale detection ensures that the model can follow size changes in the process detection [11].

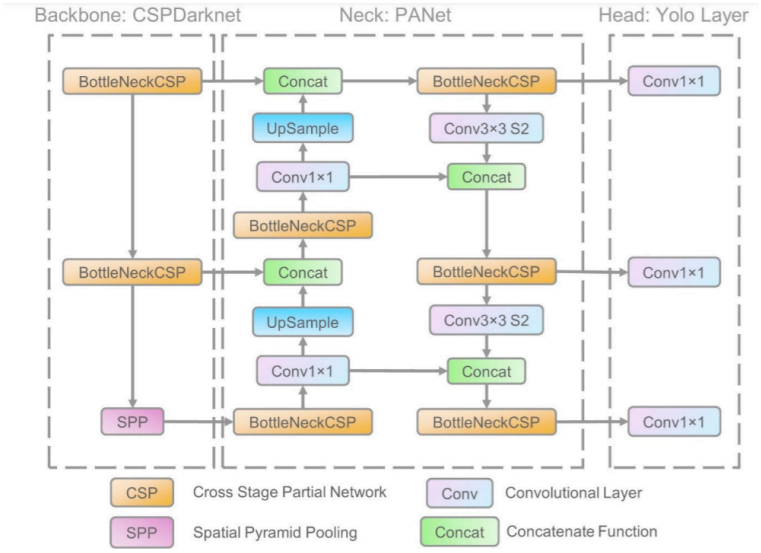

Fig. 9: The network architecture of Yolov5. It consists of three parts: (1) Backbone: CSPDarknet, (2) Neck: PANet, and (3) Head: Yolo Layer. The data are first input to CSPDarknet for feature extraction, and then fed to PANet for feature fusion. Finally, Yolo Layer outputs detection results (class, score, location, size) [11].

2) Data Augmentation: Once can see a picture of augmented training images in YOLOv5 in Fig. 10.

With each training batch, YOLO-v5 passes training data through a data loader, which augments data online. The data loader makes three kinds of augmentations: scaling, color space adjustments, and mosaic augmentation. The most novel of these being mosaic data augmentation, which combines four images into four tiles of random ratio.

The mosaic data loader is native to the YOLOv3 PyTorch and now YOLOv5 repo. Mosaic augmentation is especially useful for the popular COCO object detection benchmark, helping the model learn to address the well known ”small object problem” - where small objects are not as accurately detected as larger objects. It is worth noting that it worth experimenting with your own series of augmentations to maximize performance on your custom task.

B. Evaluation Metrics For Object Detection Models

This subsection explains mAP, an evaluation metric for object detection models. Mean Average Precision (mAP) is an evaluation metric used in object detection models such as YOLO 5. The calculation of mAP requires IOU, Precision, Recall, Precision Recall Curve, and AP.

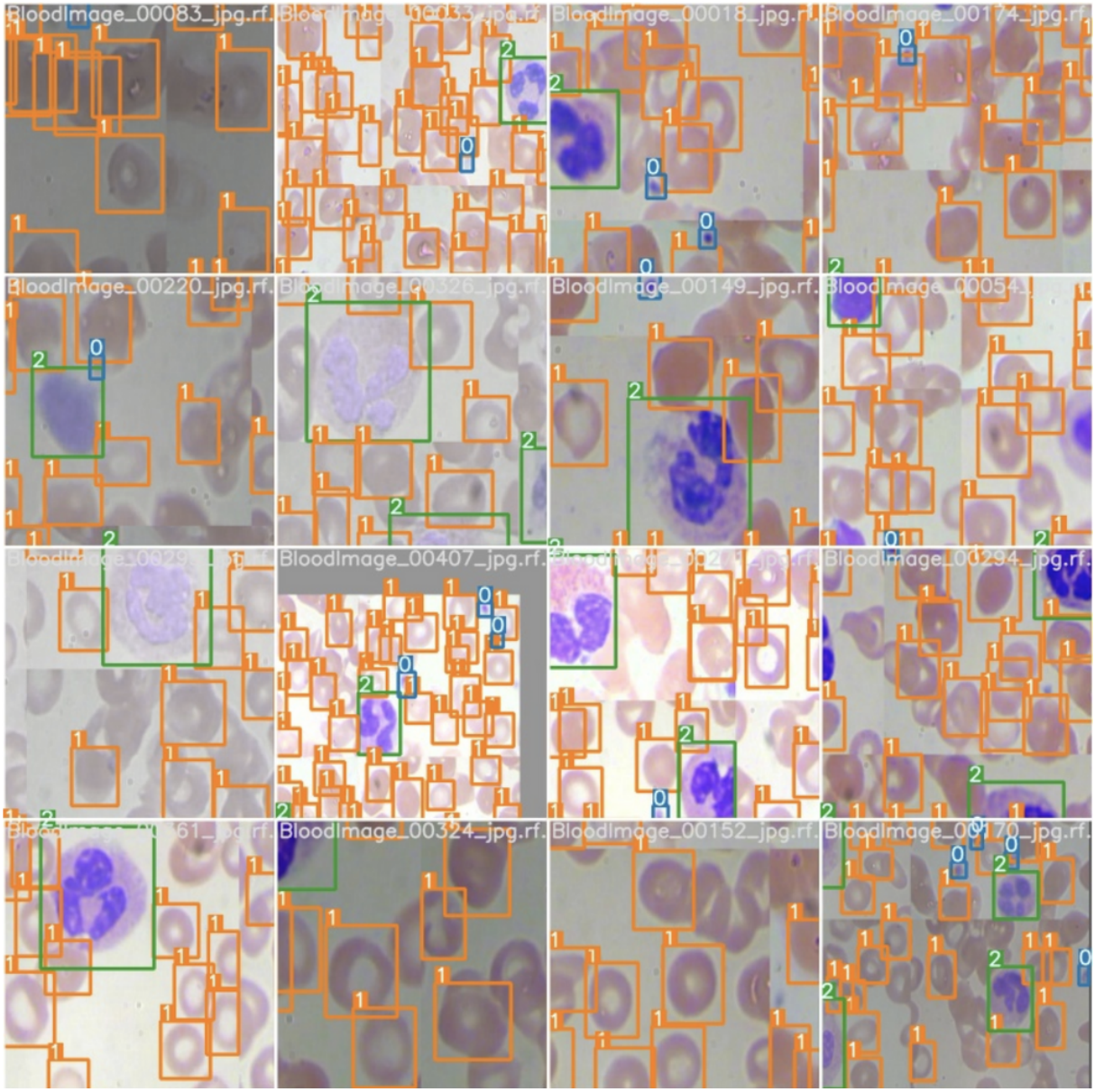

Fig. 10: Augmentation in YOLOv5

1) Intersection Over Union: Object detection models predict the bounding box and category of objects in an image. Intersection Over Union (IOU) is used to determine if the bounding box was correctly predicted. The IOU indicates how much bounding boxes overlap. This ratio of overlap between the regions of two bounding boxes becomes 1.0 in the case of an exact match and 0.0 if there is no overlap.

In the evaluation of object detection models, it is necessary to define how much overlap

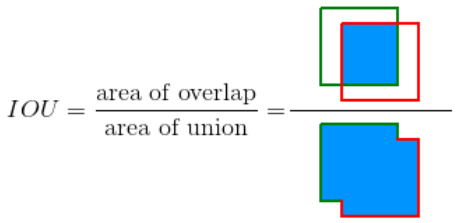

Fig. 11: Graphically expresion of IOU

of bounding boxes with respect to the ground truth data should be considered as successful recognition. For this purpose, IOUs are used, and mAP50 is the accuracy when IOU=50, i.e., if there is more than 50% overlap, the detection is considered successful. The larger the IOU, the more accurate the bounding box needs to be detected and the more difficult it becomes. For example, the value of mAP75 is lower than the value of mAP50.

2) Precision and Recall: Precision is the ability of a model to identify only the relevant objects. It answers the question "What proportion of positive identifications was actually correct?". A model that produces no false positives has a precision of 1.0. However, the value will be 1.0 even if there are undetected or not detected bounding boxes that should be detected.

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} = \frac{\text{TP}}{\text{all detections}} \tag{6}$$

Recall is the ability of a model to find all ground truth bounding boxes. It answers the question "What proportion of actual positives was identified correctly?". A model that produces no false negatives (i.e. there are no undetected bounding boxes that should be detected) has a recall of 1.0. However, even if there is an *overdetection* and wrong bounding box are detected, the recall will still be 1.0.

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} = \frac{\text{TP}}{\text{all ground truths}} \tag{7}$$

3) Precision Recall Curve: The Precision Recall Curve is a plot of Precision on the vertical axis and Recall on the horizontal axis.

There is a threshold for object detection. Increasing the threshold reduces the risk of overdetecting objects, but increases the risk of missed detections. For example, if threshold=1.0, no

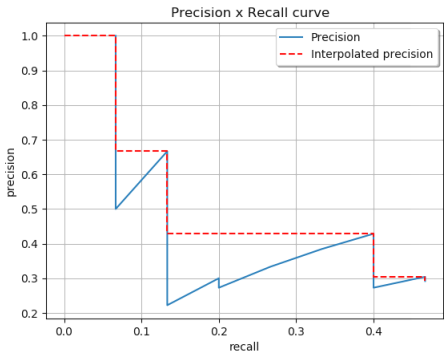

Fig. 12: Precision Recall Curve

object will be detected, Precision will be 1.0, and Recall will be 0.0. Conversely, if threshold=0.0, an infinite number of objects will be detected, Precision will be 0.0, and Recall will be 1.0. In the case of a good machine learning model, over-detection will not occur even if threshold is reduced (Recall is increased), and Precision will remain high. Therefore, the higher up the curve to the right in the graph, the better the machine learning model is.

4) AP: When comparing the performance of two machine learning models, the higher the Precision Recall Curve, the better the performance. It is time-consuming to actually plot this curve, and as the Precision Recall Curve is often zigzagging, it is subjective judgment whether the model is good or not. A more intuitive way to evaluate models is the AP (Average Precision), which represents the area under the curve (AUC) Precision Recall Curve. The higher the curve is in the upper right corner, the larger the area, so the higher the AP, and the better the machine learning model.

5) mAP: The mAP is an average of the AP values, which is a further average of the APs for all classes. AP is averaged over all categories. Traditionally, this is called mean average precision (mAP). We make no distinction between AP and mAP (and likewise AR and mAR) and assume the difference is clear from context.

The mAP is calculated by fixing the confidence threshold. COCO2017 TestSet can be used to measure mAP on various confidence thresholds to check the effect of this threshold. As a result, we confirmed that the smaller the confidence threshold is, the higher the mAP becomes.

This result suggests that the more over-detection occurs, the higher the mAP. A higher Recall will result in a larger area than a higher Precision, and we believe this is due to the small

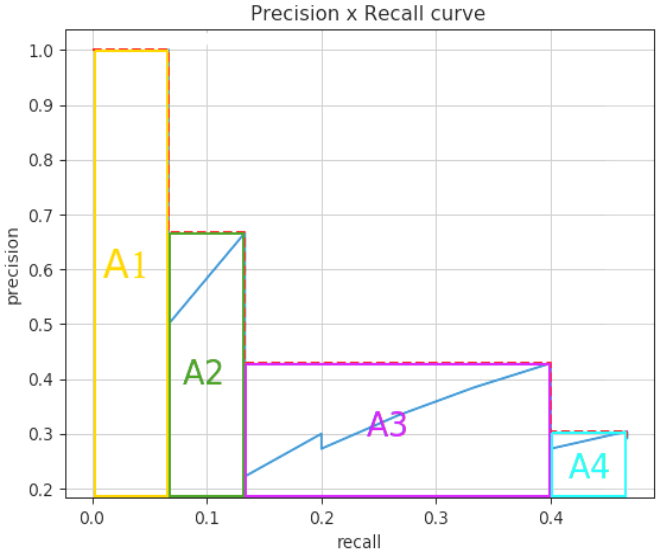

Fig. 13: Average Precision

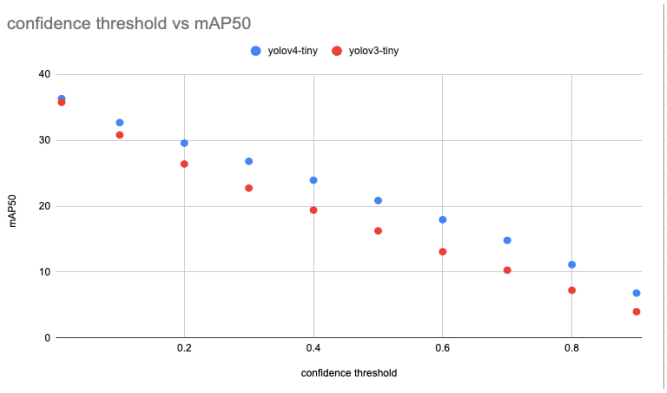

Fig. 14: mAP50 for various thresholds measured on yolo

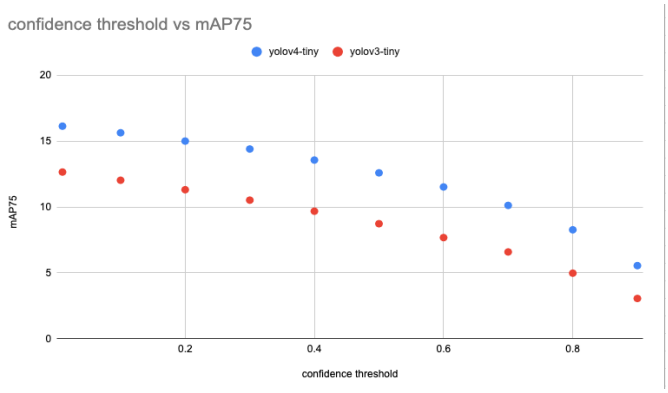

Fig. 15: mAP75 for various thresholds measured on yolo

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.507
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.689
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.552
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.345
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.559
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.652
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.381
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.630
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.682
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.526
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.732
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.829
Results saved to runs/val/exp
```

Fig. 16: The COCO2017 test results for our object detection model YOLO-v5

number images (40 670) in COCO2017 TestSet. Furthermore, in the file "YOLO-v5.ipynb", the COCO2017 test set is used to test the outcome of YOLO-v5 network. The results are presented in Fig. 16. All of the metrics used are now well-defined according to what we presented in this section.

V. Trends in Joining Two Deep Neural Networks and Their Evaluation Metrics

To handle various tasks such as speech processing, natural language processing, etc, we usually design different network models and train them with particular datasets separately, so they can behave well for specific purposes. However, in practical AI applications, it is common to handle multiple tasks simultaneously, leading to a high demand for the computation resource in both training and inference stages. Therefore, how to effectively integrate multiple network models in a system is a crucial problem towards successful AI applications. In the following, we are going to introduce some of state-of-the-art methods efficiently merging and joining two well-trained deep neural networks. We will also introduce some lemmas to evaluate the joint network performance.

A. Joining two networks in series

In this project we seek to join the two deep neural networks which detect objects and estimate depth of a given image. One of the well-known works in this task is the paper by Zhou et al. [12] which successfully joins the two networks. We note that the focus of this work is on the multiplexed images which is conceptually different than estimating depth using a single RGB

image like the NYU-v2 task. However, the authors have provided useful insights in joining the two networks. The authors utilize the bounding boxes from SSD object detection network [13] to locate objects and get the disparity[1] of each object from the disparity map predicted by depth estimation methods. The disparity of an object can be estimated by the average disparity of its central pixels or the median disparity of pixels inside its bounding box:

$$disp_{mean}^{object} = \frac{1}{0.4w \times 0.4h} \times \sum_{cx-0.2w}^{cx+0.2w} \sum_{cy-0.2h}^{cy+0.2h} disp(i,j) \tag{8}$$

and

$$disp_{median}^{object} = median\{disp(i,j) | (i,j) \in BB\}, \tag{9}$$

where $w, h, (cx, cy)$ are the width, height and center of an objects bounding box (BB), respectively. $disp(i,j)$ is the disparity map predicted by a depth estimation method such as MC-CNN [15], PSMNet [16] and Monodepth [17].

For the evaluation of the proposed disparity detectors performance of object-level depth estimation in [12], the end-point-error (EPE) is used. EPE is calculated by the average Euclidean distance between the estimated disparity and the ground-truth. The authors also use the percentage of disparities with EPE larger than t pixels ($> t$px). Here an object is considered as correct if its disparity EPE is less than t pixels. And the disparity of an object is estimated by the horizontal pixel distance between its centers of the left box and the right boxes. They have also considered three levels of occlusion for the objects in the image. The objects with occlusion = 0 are fully visible, occlusion = 1 means the objects are partly occluded, and the objects with occlusion = 2 are largely occluded.

Based on the above explanations, the performance of object level depth estimation is provided in Table. II. This table shows the evaluation results on the KITTI detection evaluation set. The best method is marked in bold-red. It can be observed that this combination scheme (with the state-of-the-art depth estimation method PSMNet [16]) achieves a comparable performance when the objects are not occluded (occlusion=0). However, their performance drops severely when the

[1]Disparity refers to the difference in horizontal location of an object in the left and right image- an object at position (x, y) in the left image appears at position $(x - d, y)$ in the right image. If we know the disparity of an object we can compute its depth z using the relation $z = \frac{fB}{d}$, where f is the focal length of the camera and B is the distance between the camera centers [14]

TABLE II: The performance of object-level depth estimation by combining object detection and depth estimation [12].

Method	Setting	Occlusion=0			Occlusion=1			Occlusion=2			Runtime(s)
		EPE	>3px	>5px	EPE	>3px	>5px	EPE	>3px	>5px	
Disparity Detector	*anchor pair*	1.11	6.58	2.16	1.26	8.00	2.67	1.38	9.54	3.28	0.027
SSD +MC-CNN	*median*	2.11	22.11	11.72	3.44	34.83	19.42	7.62	77.54	55.46	0.704
	mean	1.62	14.75	7.91	3.55	33.88	20.67	7.78	80.44	57.08	
SSD +Monodepth	*median*	1.69	16.44	5.45	2.87	32.71	14.80	6.21	69.46	44.91	0.072
	mean	1.70	17.34	5.61	3.33	36.67	18.55	6.80	74.11	49.94	
SSD +PSMNet	*median*	0.98	7.53	3.27	1.72	15.12	7.79	5.98	62.74	43.74	0.612
	mean	0.93	7.10	3.39	2.34	23.03	12.61	6.61	73.19	47.07	

object is occluded (occlusion=1 and occlusion=2) because most pixels inside the bounding box do not belong to the object. By contrast, the proposed Disparity Detector based on (8) and (9), performs well and steadily in all occlusion levels. Whats more, this method consumes much less time ($3\times$ to $20\times$ faster).

We note that our own strategy to combine object detection and depth estimation networks is the same as the joining method used in [12]. In other words, the input image is passed through the object detection network (in our case YOLO v5) and after acquiring the bounding boxes of respecting objects, these bounding boxes are fed to the depth estimation network (in our case hybrid CNN). Then according to ”mean” or ”median” strategy described in Eq. (8) and Eq. (9) respectively, we can decide whether the specific object is closer than the range user has passed to the code or not. As shown in Table. II, such joining strategy can yield shorter run-times and better EPE metric outcomes.

B. Learn Them All!

Another well-known method in joining multiple networks is ”learn-them-all” approach. To tackle multiple recognition tasks in a single system based on either unique or various signal sources, a typical approach is to design a new model and train the model on the union datasets of these tasks. The authors in [18] have named this approach as ”learn-them-all”. Such ”learn-them-all” approaches train a single complex model to handle multiple tasks simultaneously. In this work a MultiModel is trained simultaneously on the following 8 corpora:

- WSJ speech corpus
- ImageNet dataset

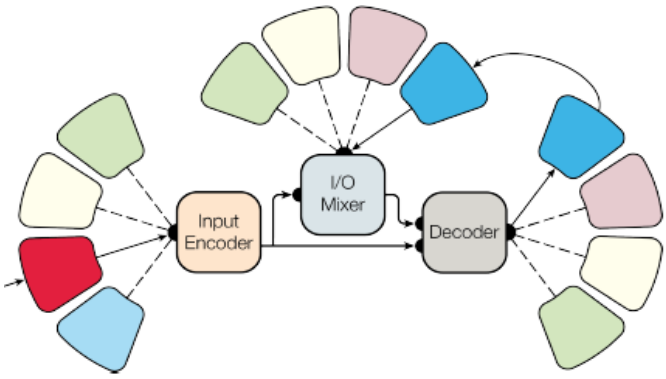

Fig. 17: The MultiModel, with modality-nets, an encoder, and an autoregressive decoder.

- COCO image captioning dataset
- WSJ parsing dataset
- WMT English-German translation corpus
- The reverse of the above: German-English translation.
- WMT English-French translation corpus
- The reverse of the above: German-French translation.

The proposed MultiModel architecture, is a single deep-learning model that can simultaneously learn multiple tasks from various domains. The model learns all of the above tasks and achieves good performance: not state-of-the-art at present, but above many task-specific models studied in recent past!

Here we do not intend to go deep into the details of the structure of "learn-them-all" approach but to put in a nutshell the MultiModel consists of a few small modality-nets, an encoder, I/O mixer, and an autoregressive decoder, as depicted in Fig. 17.

The authors ask three questions in the evaluation part one of which is what exactly we are seeking for in our project:

How does training on 8 tasks simultaneously compare to training on each task separately?

To answer the above question, the authors compare the MultiModel trained jointly with MultiModel trained separately just on a single task. When training jointly on 8 tasks, they had a separate worker training on each task with shared parameters of the model. When training on a single task, they used only a single worker training on this task for a similar number of steps.

TABLE III: Comparison of the MultiModel trained jointly on 8 tasks and separately on each task.

Problem	Joint 8-problem		Single problem	
	log(perpexity)	accuracy	log(perplexity)	accuracy
ImageNet	1.7	66%	1.6	67%
WMT EN→DE	1.4	72%	1.4	71%
WSJ speech	4.4	41%	5.7	23%
Parsing	0.15	98%	0.2	97%

Since they are comparing different instantiations of the same model, they report two internal metrics: the "negative log-perplexity"[2] and "per-token accuracy" (measured on the development set). As can be seen from the results in Table III, the joint 8-problem model performs similarly to single-model on large tasks, and better, sometimes significantly, on tasks where less data is available, such as parsing. So the joint "learn-them-all" network surprisingly performs better than the single task networks.

However, two issues may arise regarding "learn-them-all" joining approach. First, it is hard to choose a suitable neural-net architecture for learning all the tasks well in advance; hence, a trial-and-error process is required to conduct suitable architectures. Second, learning from a random initial with large training data of different types could be demanding. These issues motivated the authors in [20] to propose another solution which more effectively deals with the challenge of merging the networks.

C. Joining Two Networks Through Merging

Another method of joining the two well-trained networks which handle different tasks, is aligning the layers of the original networks and merge them into a unified model by sharing the representative codes of weights [20]. The shared weights are further re-trained to fine-tune the performance of the merged model[3]. As this method preserves the general architectures and

[2]The standard evaluation metric for Language Models is perplexity (However not limited to language models.). Lower perplexity is better! It means that the network is making "stronger predictions" in a sense. For further info please refer to [19].

[3]as we do not have any training data for joint object and depth detection task in this project, we are not able to implement the merging method and we will only explain the trends used.

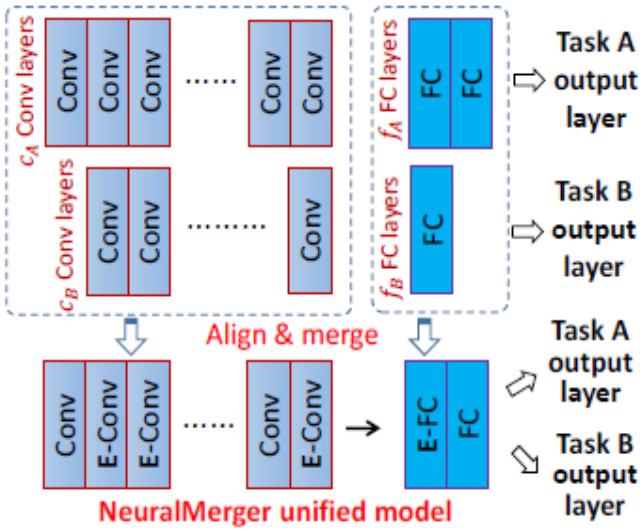

Fig. 18: Two tasks accomplished by feed-forward networks, where model A (or B) consists of c_A (or c_B) convolution and f_A (or f_B) fully-connected layers, respectively. Neural merger in [20] unifies the two models into a single one consisting of $\max(c_A, c_B)$ convolution and $\max(f_A; f_B)$ fully-connected layers for the model inference; E-Conv and E-FC are referred to as the Jointly-Encoded convolution and fully-connected layers, respectively.

leverages the co-used weights of well-trained networks, it will lead to a multitask network which can be run on resource-limited devices.

The method described in [20], assumes two distinct CNN models A and B. After merging the two, the output is a CNN consisting of jointly encoded convolution (E-Conv) and fully-connected (E-FC) layers. An overview of this approach is illustrated in Fig. 18. Assume that model A (or B) consists of c_A (or c_B) convolution (Conv) followed by f_A (or f_B) fully connected (FC) layers. Let $c_{min} = \min(c_A, c_B)$. In this approach, a correspondence $(c_{A(i)}, c_{B(i)})$ is established between the Conv layers for the alignment of the two models, $i \in \{1 : c_{min}\}$; $A(.)$ is a strictly increasing mapping from $\{1 : c_{min}\}$ to $\{1 : c_A\}$, and $B(.)$ is a strictly increasing mappings from $\{1 : c_{min}\}$ to $\{1 : c_B\}$. Likewise, a correspondence $(f_{A(i)}; f_{B(i)})$ is also established between the FC layers for $i \in \{1 : f_{min}\}$. Since in our project both the object detection and depth estimation networks have a CNN structure, this method can come in handy for joining them. Fig. 19 shows an example of merging three models by the mentioned approach.

After testing the proposed approach, the authors enumerate several benefits obtained through deploying their approach. It is stated that overall architectures of the original models are pre-

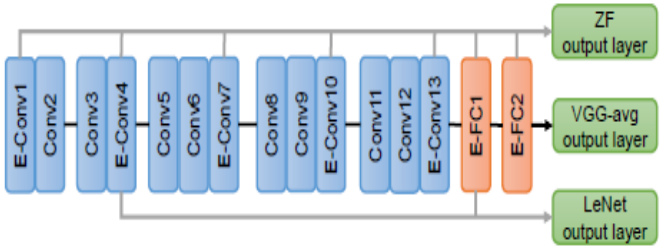

Fig. 19: Example of merging three models, ZF, VGG-avg, and LeNet into a single one for the inference stage via NeuralMerger in [20].

served. The unified model is still differentiable and can be fine-tuned to restore or enhance the performance. Experimental results show that the merged model can be extensively compressed under very limited accuracy drops, which makes this method potentially useful for deep model inference on low-end devices.

VI. Our Joint Object Detection and Depth Estimation Method

Finally, we reach to the endpoint of joining our networks! The code file for this purpose is entitled as "JOINT-DEPTH-OBJECT-DETECTION.ipynb" uploaded along with this report. In this code, we first clone the YOLO-v5 network into our notebook. Then we do the same thing for the hybrid CNN network to use it as our depth estimator. Until now, the user has nothing to do with the program.

But after this stage, as code runs through, a box appears for the user asking him/her to enter the URL address of where his/her image is saved. Then the image is fed to the object detector. We do not need the raw image output of the detector but we need the coordinates of bounding boxes. For further processing we also store the object types.

Using the coordinates obtained, we extract the object parts of the original image and feed the resulting images(s) to the depth detector. In order to obtain the depth of the these objects we exploit the method used in [12]. As discussed in Section V, we deployed the equations (8) and (9) in our code which are the basis of our strategy in joining our depth and object detection networks. Here the user is required to enter a value for maximum depth into the code. The code uses the equations (8) and (9) to obtain the depth and compares the resulting value with the maximum depth value entered by the user. The objects with the depth less than the maximum

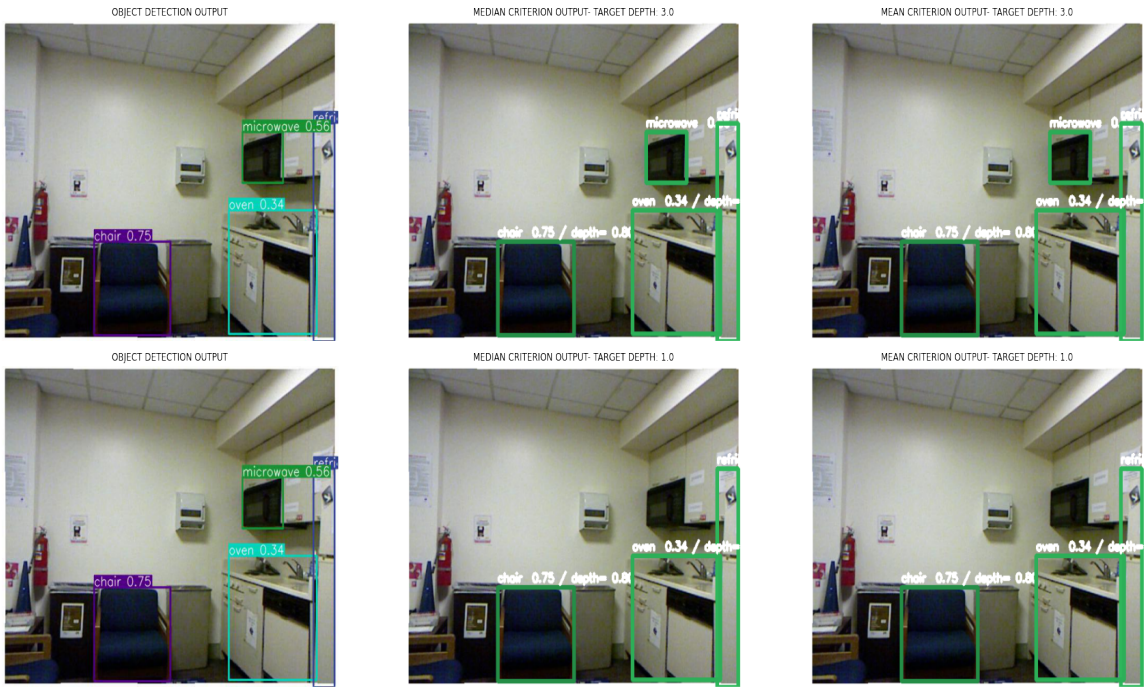

Fig. 20: Testing our joint object detection and depth estimation method by images from NYU-v2 dataset

depth will finally appear in the output image. We provided the visual results for both the mean and median methods in the code.

Figures 20 through 22, show the results after feeding the images from NYU-v2 dataset into our joint network. The tests are conducted for various values of maximum depth that the user gives to the code and the outcomes for both the median and mean criterion are presented. Fig. 20 shows that when we enter a large enough value for the maximum depth, as expected, all the objects are labeled. But when a smaller number is fed to the code, the farther object gets unlabeled (in this case the microwave oven). Both the mean and median criterion perform well in joint task. So far so good! But things can change ...!

Fig. 21 shows the results for a bedroom scene. First, we note that the results with two criterion can be different as it is evident from this image. Secondly, the question that arises here is that why the bottle is detected closer than the bed in the mean criteria with max depth of 1, whereas the bed is closer? We do believe this is because of the curvature effect inherent in the RGB image of bed that challenges our network. This observation shows that we should be more careful

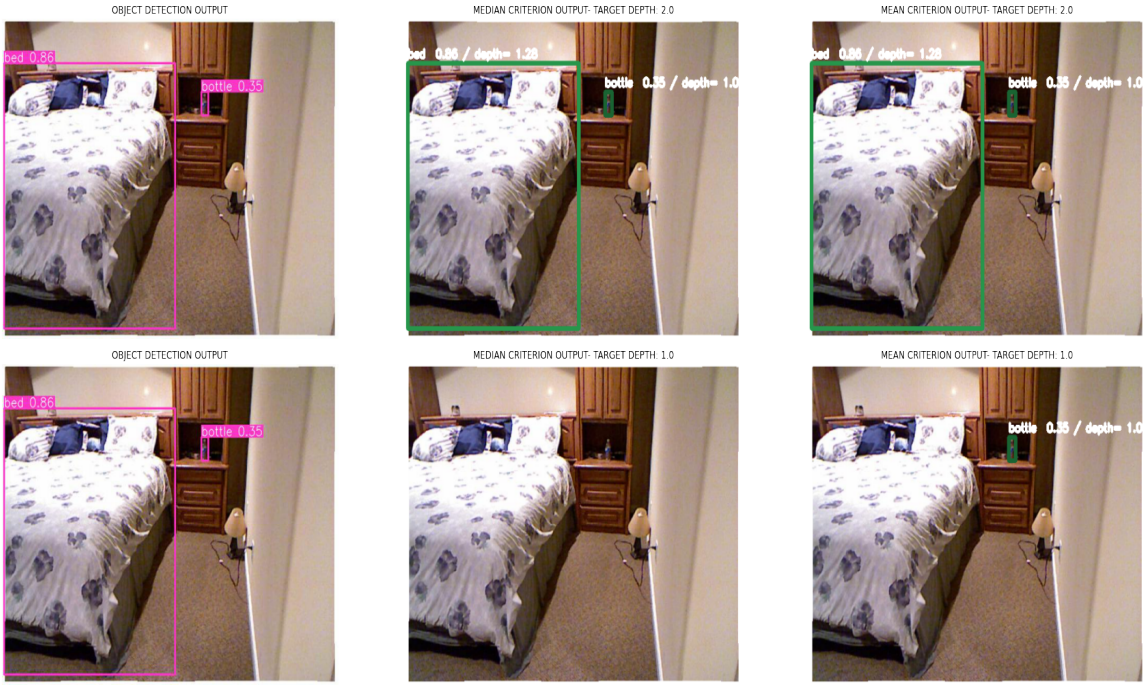

Fig. 21: Testing our joint object detection and depth estimation method by images from NYU-v2 dataset. The effect of curvature of bed is interesting!

Fig. 22: Testing our joint object detection and depth estimation method by images from NYU-v2 dataset. The nearest chair is not labeled! But why?! we explain this in the text!

about the object curvatures in joint task.

Fig. 22 shows another challenge that we face. Lets focus on the detection of the farther chair instead of the nearest when max depth is equal to 1. We remind that we used two metrics for object depth detection that rely on the mean and median of the values of depth inside the bounding box. For an object like chair which has a hollow form inside its bounding box, we can expect that our network will face grave challenge. So through this observation we can say that we need to be more careful when setting a criteria and pay more attention to not getting into the trap of hollow shaped objects.

VII. Conclusions

In this project we investigated the various aspects of joining two neural networks together. Numerous strategies for joining the networks along with the associated performance metrics were presented. We also implemented a code which performs the joint object detection and depth estimation task. Two strategies were used in our code to join the single networks together. We examined our network with the images from NYU-v2 dataset and presented the qualitative results. We also observed that the curvature of the objects along with the hollow-shaped objects can challenge our joint network.

References

[1] Muslikhin, J. -R. Horng, S. -Y. Yang and M. -S. Wang, "Object Localization and Depth Estimation for Eye-in-Hand Manipulator Using Mono Camera," in IEEE Access, vol. 8, pp. 121765-121779, 2020, doi: 10.1109/ACCESS.2020.3006843.

[2] Casser, V., Pirk, S., Mahjourian, R., Angelova, A. (2019). Depth Prediction Without the Sensors: Leveraging Structure for Unsupervised Learning from Monocular Videos. AAAI.

[3] Laina, I., Rupprecht, C., Belagiannis, V., Tombari, F., Navab, N.: Deeper depth prediction with fully convolutional residual networks. In: 2016 Fourth International Conference on 3D Vision (3DV), pp. 239248 (2016)

[4] Alhashim, I., Wonka, P. (2018). High Quality Monocular Depth Estimation via Transfer Learning. ArXiv, abs/1812.11941.

[5] Harsnyi K., Kiss A., Majdik A., Sziranyi T. (2019) A Hybrid CNN Approach for Single Image Depth Estimation: A Case Study. In: Choro K., Kopel M., Kukla E., Siemiski A. (eds) Multimedia and Network Information Systems. MISSI 2018. Advances in Intelligent Systems and Computing, vol 833. Springer, Cham.

[6] He, K., Zhang, X., Shaoqing, R., Jian, S.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770778 (2016)

[7] Ronneberger, O., Fischer, P., Brox, T.: U-net: convolutional networks for biomedical image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, pp. 234241. Springer, Heidelberg (2015)

[8] Eigen, D., Puhrsch, C., Fergus, R.: Depth map prediction from a single image using a multi-scale deep network. In: Advances in Neural Information Processing Systems, pp. 23662374 (2014)

[9] Wang, P., Shen, X., Lin, Z., Cohen, S., Price, B., Yuille, A.: Towards unified depth and semantic prediction from a single image. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 28002809 (2015)

[10] Eigen, D., Fergus, R.: Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 26502658 (2015)

[11] Xu, Renjie Lin, Haifeng Lu, Kangjie Cao, Lin Liu, Yunfei. (2021). A Forest Fire Detection System Based on Ensemble Learning. Forests. 12. 217. 10.3390/f12020217.

[12] C. Zhou, Y. Liu, Q. Sun and P. Lasang, "Joint Object Detection and Depth Estimation in Multiplexed Image," in IEEE Access, vol. 7, pp. 123107-123115, 2019, doi: 10.1109/ACCESS.2019.2936126.

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, Ssd: Single shot multibox detector, in European conference on computer vision. Springer, 2016, pp. 2137.

[14] J. Zbontar and Y. LeCun, Stereo matching by training a convolutional neural network to compare image patches, Journal of Machine Learning Research, vol. 17, no. 1-32, p. 2, 2016.

[15] J. Zbontar and Y. LeCun, Stereo matching by training a convolutional neural network to compare image patches, Journal of Machine Learning Research, vol. 17, no. 1-32, p. 2, 2016.

[16] J.-R. Chang and Y.-S. Chen, Pyramid stereo matching network, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 54105418.

[17] C. Godard, O. Mac Aodha, and G. J. Brostow, Unsupervised monocular depth estimation with left-right consistency, in Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on. IEEE, 2017, pp. 66026611.

[18] Kaiser, L., Gomez, A.N., Shazeer, N.M., Vaswani, A., Parmar, N., Jones, L., Uszkoreit, J. (2017). One Model To Learn Them All. ArXiv, abs/1706.05137.

[19] https://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture06-rnnlm.pdf

[20] Chou, Y., Chan, Y., Lee, J., Chiu, C., Chen, C. (2018). Unifying and Merging Well-trained Deep Neural Networks for Inference Stage. ArXiv, abs/1805.04980.