AVI@PRS, ETH Zurich Computer Vision and Artificial Intelligence for Autonomous Cars

Project 1: Semantic Segmentation and Depth Estimation

25% of the course grade 10-10-2025 – 14-11-2025

Quick Links:
→ Course website
→ Piazza forum and announcements*
→ Codabench grader and leaderboard
→ Latest version of the solution template and training instructions
→ Latest version of this document
→ SLURM tutorial
* Announcements about exercise updates will be made on Piazza. Questions about the exercise are welcome; however, if you believe that your question may reveal the solution, please make sure to *post to instructors* when creating a post.

Introduction: In this exercise, we will investigate two computer vision problems: semantic segmentation and monocular depth estimation. In particular, semantic segmentation is the task of associating each pixel of an image with a class label, e.g., person, road, car, etc., whereas monocular depth estimation is the task of estimating the per-pixel depth of a scene from a single image. As with many other tasks nowadays, semantic segmentation and monocular depth estimation can be effectively tackled by using Convolutional Neural Networks (CNNs) [10]. To achieve state-of-the-art results, deep convolutional networks [9, 11] are typically trained on datasets that contain a large number of fully annotated images, that is, images with their corresponding ground-truth label. This allows the networks to encode feature representations that are discriminative for the task at hand. In what follows, we are going to train deep learning models to perform semantic segmentation as well as monocular depth estimation.

Training models on CVL's SLURM: Each team will be given an account for CVL's SLURM to conduct the experiments required to solve the problems in this assignment. Each team will have access to one TITAN X GPU. The main source of technical information not covered by this document is the solution template `README.md` (see Quick Links). Everyone should look at it and follow the explained steps for everything related to:

— Environment Setup
— Interactive Debugging
— Sbatch Training
— Monitoring
— Checkpoints and Submission
— Resume Training

The solution template comes ready for training on SLURM and implements a baseline for the problems below. After each successful training of the model, the code will generate predictions on the RGB images of the test split (see Dataset section below). The submission archive containing predictions is created automatically. Apart from test predictions, the archive also contains the experiment configuration, source code, and the training log. This submission archive will be saved to `/srv/beegfs-benderdata/scratch/$USER/data/ex1_submission` by default. The archive does not include the checkpoints (trained model weights) to keep the dimension of the archive low. However, you can find the checkpoints under the checkpoints folder. Each submission archive can be downloaded locally and then uploaded to the grader (see "Grader" section) to (1) participate in the leaderboard and (2) obtain the test split metrics for the final submission (see "Final submission" section). Important: the code will clean the saving directory

at the beginning of each training; therefore, if you want to preserve some, you should move it to another directory.

Dataset: For this exercise, we use a toy dataset of synthetic scenes in the autonomous driving context. The dataset is composed of predefined splits with 20000 training images, 2500 validation, and 2500 test images. The validation and test splits are quite similar, so observing the validation performance in W&B should give a good estimate of the expected score with the grader (see W&B, Grader). The dataset contains three modalities for each image sample: RGB, Semantic annotation, and the Depth map. The solution template and the bash scripts automatically handle dataset copying and decompression; nothing should be changed about the data loading pipeline.

Metrics: The following metrics are used to evaluate each experiment's outcome:

— IoU (intersection-over-union) is a metric of performance of the semantic segmentation task. Its values lie in the range [0,100]. Higher values are better. It is shown as `metrics_summary/semseg` in W&B.

— SI-logRMSE (scale-invariant log root mean squared error) is a metric of performance of the monocular depth prediction task. Its values are positive. Lower values are better. It is shown as `metrics_summary/depth` in W&B.

— The Multitask metric is a simple product of the aforementioned task-specific metrics, computed as $max(iou - 50, 0) + max(50 - silogrmse, 0)$. Its values lie in the range [0,100]. Higher values are better. It is shown as `metrics_summary/multitask` in W&B. For Problems 1. and 4., the multi-task metric will be zero, since the former focuses only on segmentation and the latter on depth estimation.

W&B: The code template uses Weights and Biases to monitor the training progress. As part of the setup, each team will need to register a free account with the service. Navigate to https://wandb.ai, register a new account (or login with email), then navigate to `Settings` → `API keys`. A default key will be available: hover over the key, click the plus button to copy this key, and add it to `wandb.key` file in the root folder of the project. The default visibility of W&B projects is "private", as indicated by the closed lock icon against the project name – make sure to keep it that way throughout the course. Feel free to erase "bad" runs to avoid cluttering; however, keep all the runs that you reference in your final report (see "W&B sheet" section) until the end of the course.

W&B allows inspecting the training dynamics (e.g., loss curves, validation metrics), collecting advanced statistics (histograms of weights or activations), as well as displaying images of predictions. The code template makes heavy use of all these features. The main tabs of interest within each individual run (select a certain run from the dashboard first) are "Overview" and "Charts", which can be found on the left side.

However, the main benefit of W&B for efficient development comes in the aggregated view of the experiments (select the project CVAIAC-Ex1 from the dashboard): the "Charts" and "Table" tabs allow one to compare different runs with different settings and identify configurations which cause improvement (not automatically though). Each experiment may be given a name (such as "feature1value1_feature2value2") by changing the value of the `--name` flag in `slurm_train.sh` script to allow easy identification of a run.

At the end of each epoch, the metrics (see "Metrics" section) are evaluated for the validation split of the dataset. This should serve as guidance to improving models when solving exercise problems. The test performance can only be evaluated by the grader.

Grader: The grader's purpose is to evaluate a submission archive corresponding to one experiment run on the test data. This is required to report scores of solutions to the exercise problems.

To register, navigate to https://www.codabench.org/ and create a shared team account with the username `cvaiac25_teamXX`, where XX is your team number (with a leading zero if ≤ 9). Please share the credentials with the team members. Next, follow the grader URL (see Quick Links), and click `"Participate" -> "Register"`.

To get a submission graded, navigate to `"Participate" -> "Submit / View Results"`, enter the W&B run name of the form `GXX_XX-XX_<run_name>_XXXXX` followed by some comment (e.g., "added ASPP") into the "description" field, then click on the large `"Submit"` button, and wait to get the submission archive uploaded. Codalab begins file upload immediately after it was selected in the system dialogue, and does not indicate the upload progress, so give it a minute to upload. After the upload has finished, the status will change to indicate grading has been scheduled. You may need to refresh the page with `F5` to see submission status updates after that. If you are satisfied with the scores, you can push a submission to the leaderboard by clicking the corresponding button. Each team can make a total of 40 submissions to the grader, at most five submissions per day.

W&B sheet: For traceability, we require you to annotate and export your W&B experiment table. To annotate a run, please go to the "Table" tab of the project on W&B and add an annotation in the column "Notes". The annotation should clearly indicate where the run was used in the final report (e.g. "Row 1 in Tab. 1: Adam with LR 1e-4"). If you did not use a run in the report, please indicate that by leaving the cell empty. Make sure that you show all columns by clicking on the "Columns" button in the upper right and select "Show all". You can export the W&B table using the download icon in the upper right and choose "CSV Export". Please include this CSV file in the final submission (see "Final submission" Section).

Grader score sheet: We require you to upload the run with the best validation metric (the exact metric will depend on the task) to the Grader (see "Grader" Section) in order to obtain the scores on the test set. You are not required to report the Grader scores in the report, but you have to enter them in the grader score sheet. You can find the grader score sheet in the solution template under `doc/cvaiac25_teamXX_ex1_scores.csv`. This file will be part of your final submission.

Report: The report should be prepared as a PDF document. We recommend using Overleaf for typesetting in LaTeX, but any text editor capable of exporting into PDF should do. There is no page limit, but please avoid lengthy and redundant descriptions.

Final submission: The final submission must be sent by each team in a file named `cvaiac25_teamXX_ex1_submission.zip`, where `XX` is your team number, to Danilo (danilo.dordevic@geod.baug.ethz.ch) by 23:59 CET 14-11-2025 with subject `"[CVAIAC25] EX1 final submission: teamXX"`. The zip archive should contain the PDF report `cvaiac25_teamXX_ex1_report.pdf`, the grader score sheet `cvaiac25_teamXX_ex1_scores.csv`, the W&B sheet `cvaiac25_teamXX_ex1_wandb.csv`, and the solution code `cvaiac25_teamXX_ex1_code.zip`.

Important: Sharing the code, dataset, configurations, and run artifacts (W&B) with anyone except your teammate is not allowed at any time (even after the end of the course). To use versioning (GitHub or GitLab), make sure to use a private repository. Do not change the visibility of W&B project to public.

Project evaluation:

The project is evaluated based on the following criteria:

- For each problem statement and question, the report contains an accurate and complete description of your solution (e.g. how the configuration was altered or how a certain module was implemented).
- For problems that require code changes on top of the template, the report contains code

snippets[1] of the relevant changes.
- The solution code is complete and correct.
- For each question (e.g. "How does SGD compare to Adam?"), the team has run one of a few experiments to base their answers on.
- The results of the experiments are presented clearly and concisely. For instance, you can use tables, diagrams, and example predictions. The means of presentation should be as detailed as necessary to discuss all relevant aspects but still as concise as possible. For example, if you only discuss the performance at the end of the training, you do not need the performance curve over the course of the training but can report the metrics in a table. Do not mention model performances only in the text without including another means of presentation (e.g. table or diagram).
- All relevant observations from the experiments are described and possible reasons are discussed.
- All questions within a task are answered completely and correctly.
- The grader scores submitted in the score sheet are on a par with the expected reference score.
- The experiments are traceable (see "W&B sheet" section).

The total number of exercise points will be communicated back before the exam. A review of graded hand-ins will happen at the exam review session upon request.

For your reference, we provide the indicative validation results after the first epoch with our implementation. If your solution performance is decisively worse than the provided reference values, that might be a sign to look deeper into the current problem before going to the next one.

Problem 1. Semantic Segmentation

(4+3+3=10 points)

Your starting point is a DeepLab model [3, 4, 5, 7] that consists of a ResNet-like encoder [9], an ASPP module, and a decoder with skip connection. The template code functions properly and can be trained straight away; however, the baseline performance will be poor: we intentionally chose sub-optimal default values for some of the hyperparameters and short-circuited some of the model parts, namely ASPP and the decoder.

0. Baseline (0 pts): Reproduce the baseline (no changes to the code are required) and submit it to the Grader (Codebench). The approximate graded performance of the `submission.zip` (after 1 epoch) should be as follows: `semseg`: 49.2, others will be zero. This is just a sanity check for you. There is no need to include this task in the report.

1. Hyper-parameter tuning (4 pts): As a first step, you need to familiarize yourself with the hyperparameters. The file `source/utils/config.py` describes hyperparameters, which can be changed using command line keys to the training script. We encourage you to try different settings and examine the effect that each parameter has on the final result in order to get a better understanding of the codebase. More specifically, you should investigate the following options and report informative conclusions about your findings.

   (a) *Optimizer* and *learning rate*: What is the best choice for the optimizer (SGD vs. Adam) and the learning rate for this task? You can use a base of 10 for the learning rates (e.g. 0.01, 0.001, ...). Comment on the optimal learning rate for both SGD and Adam. How does the learning rate affect the learning process (remember to include an appropriate visualization as basis for your answer)? How does using SGD compare to Adam? From now on, we recommend using Adam with a learning rate of 1×10^{-4}.

[1] Recommended: https://www.overleaf.com/learn/latex/Code_Highlighting_with_minted

   (b) *Batch size*: How does the batch size affect the performance metrics of the multitask network? When changing the batch size, the number of steps per epoch will decrease proportionally. To alleviate this effect, we recommend changing the number of epochs proportionally to the batch size. Which other practically relevant aspects besides the task metrics does the batch size affect and how? For the following tasks, we recommend keeping the batch size 16.

2. Hardcoded hyperparameters (3 pts): Now you need to study the main building blocks of the experiment, model, and loss modules (arranged in the respective subdirectories under `source` directory), and perform one-line changes of the code to improve the model.

   (a) Encoder initialization: How are the weights of the encoder network initialized in the solution template? What happens when you switch the `pretrained` flag of the encoder? Please answer these two previous questions precisely. How do both variants compare performance-wise and what might be the reason for that? Make sure to persist the option leading to the improvement before proceeding to the next questions.

   (b) Dilated convolutions: Look closely at the encoder code in `model_parts.py` and check whether dilated convolutions are enabled. This aspect is closely related to the term "output stride" used in [7]. You can use the commented `print` statement in `model_deeplab_v3_plus.py` to help you see the mapping of each scale of the feature pyramid to the respective number of channels. The largest scale factor in the pyramid corresponds to the "output stride". Set dilation flags to (False, False, True) and train the model. Does the performance improve? If so, why? Also, include a comparison of example predictions from W&B into your discussion. Use the model with dilation as a reference when reporting the effects of ASPP and Skip Connection.

3. *ASPP and skip connections* (3 pts): The next task is to implement the ASPP module [5, 7] along with skip connections to the decoder, whose design details are provided in the referenced papers. You are already given the skeleton of the *ASPP* class, and you are asked to replace the current trivial functionality with the proper one. The details of the ASPP module can also be found in Figure 1. If desired, you can use the *ASPPpart* class as well. The last missing part is the decoding stage with skip connection as done in [7]. You are given the `DecoderDeeplabV3p` class that already contains the appropriate inputs and outputs. You have to replace the current functionality with the intended one, essentially processing the features that come from the encoder and the ASPP module and outputting the final channels that contain the predictions. The detailed diagram of this part can be found in Figure 1. Further important information about the configuration of the layers can be found in [7]. The code parts which require work are marked with `TODO` annotations. What is the intention of adding ASPP and skip connections? How does the model performance change with ASPP and skip connections functioning? Also, include a comparison of example predictions from W&B into you discussion. Do not forget to describe your solution and to provide the code snippets in your report as mentioned in the section "Project evaluation".

Roughly expected validation performance: `mIoU`: ≥ 70

Problem 2. Introducing a Second Task: Depth Estimation (5 points)

In this problem, we would like to solve both dense tasks, namely semantic segmentation and depth estimation, at the same time. The main reasoning is that semantic and geometric should help each other. Since we need to solve both tasks (semantics and depth) under a single model, a naive Multi-Task Learning (MTL) solution is to share all operations (i.e., encoder, ASPP, decoder) between tasks except for the last convolution that maps the features of the preceding layer to $n_{\text{classes}} + 1$ channels. The former n_{classes} channels correspond to pixel-wise class probability distribution before softmax (logits) for the semantic segmentation task, while the latter

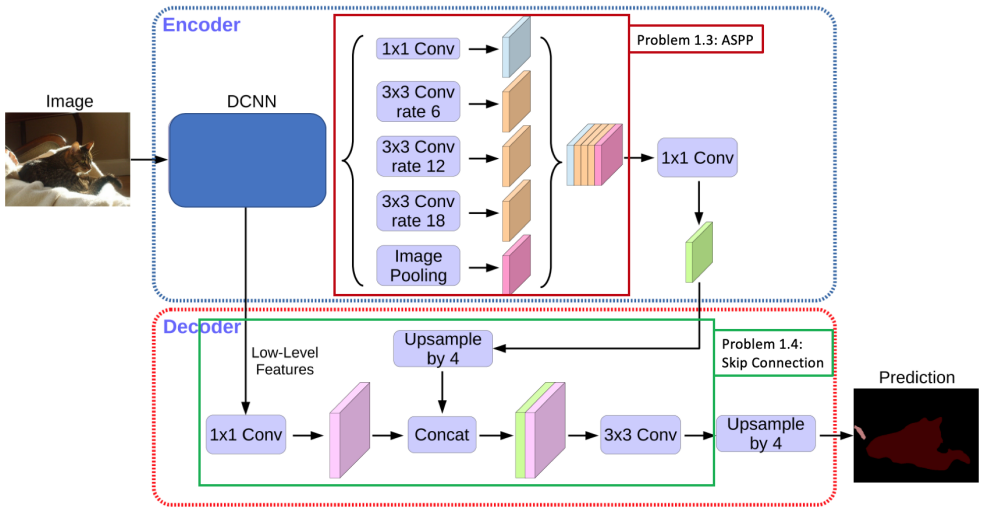

Figure 1: ASPP module and Skip Connection

1 channel corresponds to the regressed depth values for the monocular depth estimation task. This joint architecture is depicted in Figure 2.

The loss used for Monocular Depth Estimation is the *Scale Invariant Log* (SILog) loss. The proper mathematical formulation of the optimization induced is:

$$\begin{aligned} &\text{minimize } \mathbb{V}(\varepsilon), \quad \varepsilon = \log(y^*) - \log(\hat{y}) \\ &\text{s.t. } \mathbb{E}[\varepsilon]^2 \leq \mu \end{aligned} \tag{0.1}$$

where $y^*, \hat{y}$ are ground truth and predicted value, respectively. The program can be rewritten via its *Lagrangian* and it boils down to: minimize $\mathbb{V}(\log(y^*) - \log(\hat{y})) + \lambda \mathbb{E}[(\log(y^*) - \log(\hat{y}))]^2$. As always, the loss is rewritten in terms of empirical loss, hence variance and expected value are the empirical ones, e.g., $\mathbb{E}[\varepsilon] = \frac{1}{N}\sum_{i=1}^{N} \varepsilon_i$. It is worth noticing that the complete solution of the Lagrangian would relate in closed-form λ and the μ in the original formulation. However, we set directly λ which can be interpreted as a trade-off parameter between focusing on respecting the constraint and focusing on the unconstrained minimization problem. This is the standard practice since the Lagrangian is not solvable in closed form (we have access to empirical distributions only). You can set $\lambda = 0.15$ as per standard practice. Moreover, the final loss is the square root of the one defined above.

1. *Depth Loss* (4pt):

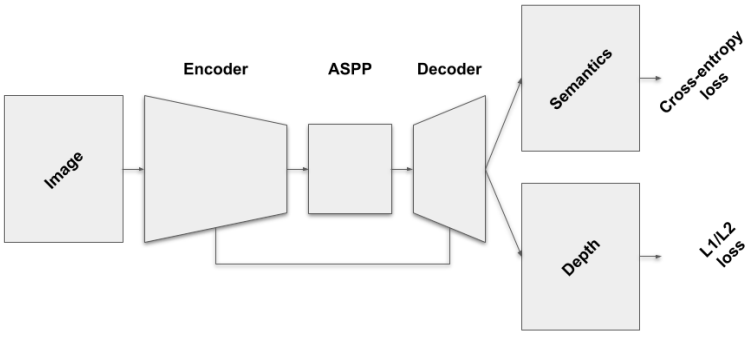

Figure 2: Joint architecture

(a) Implement the SILog forward method in `source/losses/si_log.py`.

(b) Explain why the log error variance is invariant to the predictions' scale. Remember what scale-invariance is: $f(\cdot)$ is defined as scale-invariant if $f(a \cdot x) = f(x)$.

(c) Explain the role of the term $\mathbb{E}[\varepsilon]^2 \leq \mu$ in 0.1 and why it is needed to have proper *metric* depth output.

2. *Task weighting* (1pt): When multiple tasks are learned together, the individual losses of each task are usually part of a weighted sum for the total loss, which is used for updating the network parameters. What is the effect of increasing the loss weight for one task while decreasing the weight for the other task? How should the task weights be chosen and why?

To run the MTL you can change the `--tasks` argument to `--tasks depth semseg`, in `shell_train.sh`, or `slurm_train.sh`.

Roughly expected validation performance: `mIoU`: ≥ 70 , `SILog-RMSE`: ≤ 30

Problem 3. MultiTask Learning (5 points)

In the previous problem, we used a joint architecture, which shared all network components except the last convolutional layer – to learn both tasks. Another MTL solution is the adopt a branched architecture [12, 13], where a common encoder is used for both tasks, but task-specific ASPP modules and decoders are implemented for semantic segmentation and monocular depth estimation, respectively. Figure 3 gives an overview of this architecture.

As part of this problem, you are asked to implement this branched architecture using the same building blocks: encoder, ASPP, and decoder modules. To narrow down the scope of effort and prevent unintentional breaking of the pipeline, all code changes should be restricted to `source/models` path. How does it compare to the joint architecture both in terms of performance but also w.r.t. the GPU memory consumption, the number of network parameters, and the training time? Please also explain possible reasons for your observations.

Instead of modifying `ModelDeepLabV3Plus` class in `source/models/model_deeplab_v3_plus.py`, you should modify `ModelDeepLabV3PlusMultitask` class in `source/models/model_deeplab_v3_plus_multitask.py`. To call the model you need to change the argument `--model_name` to `deeplabv3p_multitask`. You can look into the choices in line 96 at `source/utils/config.py`.

Roughly expected validation performance: `mIoU`: ≥ 72, `SILog-RMSE`: ≤ 25

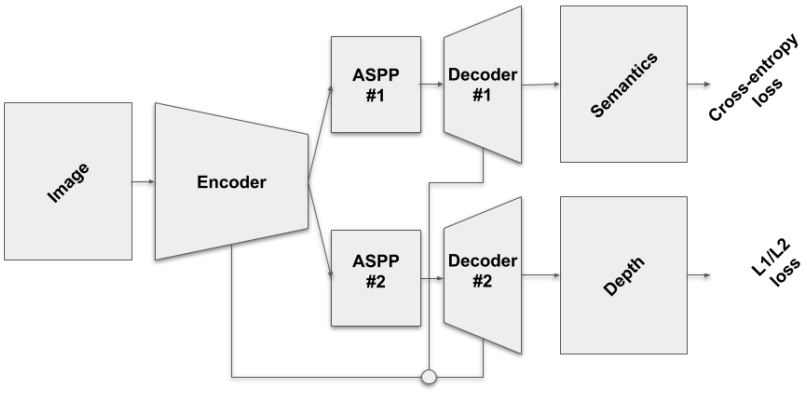

Figure 3: Branched architecture

Problem 4. Adaptive Depth Estimation (10+5=15 points)

Another popular way of representing depth is via a discrete representation. More specifically, the continuous depth range is discretized in a set of possible values, *i.e.*, in a finite amount of bins. As a consequence, the depth estimation problem is turned into a classification task. For every pixel, the outputs correspond to the probabilities that the pixel falls in a specific depth bin. From a practical point of view, one needs to define the total depth interval and then the boundaries of the bins. The boundaries are defined via: the number of bins and the type of spacing (e.g., log or linearly spaced). [8]
However, the predicted depth is discretized and is inherently totally discontinuous, this can negatively affect downstream tasks (e.g., grasping). A simple way is to predict the final depth as the average of bin values weighted by the output probabilities, instead of the argmax. In this way, the final depth is differentiable (argmax is not) and we can optimize via regression losses on top of this continuous depth.
Even though the discontinuity problem can be solved via weighted sum, the depth range, and the bin values are still pre-determined and fixed. However, the model should be able to "allocate" denser bins to ranges that are needed the most based on the content of the image itself and avoid "wasting" bins on unused depth range. Hence, if we can define the range and the bins based on the input itself, better granularity can result in a more refined depth [2].

To tackle the problem we need to first define what is the attention mechanism and what a transformer is.

The attention mechanism is an operation where each output is computed as a convex combination of the inputs, and the combination weights are a function of the input itself. In other words, the weights of the attention layer are the softmax of the similarity between the (possibly projected) inputs. More specifically:

1. Given an input $\mathbf{X} \in \mathbb{R}^{N \times D}$, it is first normalized through LayerNorm operation [1].
2. After the normalization we obtain the "query", "key", and "value" tensor as the projection (a.k.a. linear layer) of the input, i.e., $\mathbf{Q} = \mathbf{W}^Q\mathbf{X}$, $\mathbf{K} = \mathbf{W}^K\mathbf{X}$, $\mathbf{V} = \mathbf{W}^V\mathbf{X}$, where $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^K \in \mathbb{R}^{D \times C}$ and D is the input channel dimension and C is a common channel dimension of the attention layer.
3. The similarity between the query and key tensor is computed as the dot-product: $\mathbf{S} = \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{C}}$. Where $\mathbf{S} \in \mathbb{R}^{N \times N}$, where N is the number of inputs(also called "tokens" in the attention and transformer context).
4. The output is computed as the convex combination of the value tensor, where the combination weights are obtained as softmax($\mathbf{S}$), where softmax is applied on the last dimension.
5. Usually the final output is projected back to dimensionality D with a linear layer (projection) $\mathbf{W}^O \in \mathbb{R}^{C \times D}$ and it is added in a residual fashion to the input itself, namely to $\mathbf{X}$.

To sum up, the attention layer computes each output as: (LayerNorm is not shown to avoid an even more cluttered notation):

$$\mathbf{X}^{l+1} = \mathbf{X}^l + \mathbf{W}^O[\text{softmax}(\mathbf{W}^Q\mathbf{X}^l(\mathbf{W}^K\mathbf{X}^l)^T)\mathbf{W}^V\mathbf{X}^l] \tag{0.2}$$

In its essence, the Attention mechanism is a computational paradigm that enhances the capacity of a network to *selectively* emphasize, prioritize, and propagate certain elements within a given input, mimicking the human cognitive process of focusing attention on the most useful information during the processing.
A natural improvement to vanilla Attention is to have multiple heads processing the input in H independent chunks along the channel dimension, also called MultiHead Attention. The main

change is in the implementation, namely channel dimension should be unpacked and the "head" dimension will be treated as if we had a bigger batch size: $\mathbb{R}^{B\times N\times(H\times\frac{D}{H})} \rightarrow \mathbb{R}^{(B\times H)\times N\times\frac{D}{H}}$, where B is batch size, D the original channel dimension, H the number of heads.

A Transformer is a sequence of alternating Attention Layers and MultiLayer Perceptron (MLP) Layers, the latter consists of two projections with a non-linearity (σ), e.g., ReLU, in the middle, and optionally a LayerNorm applied to the input:

$$\mathbf{X}^{l+1} = \mathbf{X}^{l} + \mathbf{W}^{2}\sigma(\mathbf{W}^{1}\text{LayerNorm}(\mathbf{X}^{l})) \tag{0.3}$$

Usually, the internal dimensionality is larger (inverted bottleneck) by a predefined expansion factor, typically 4. This results in, e.g., $\mathbf{W}^{1} \in \mathbb{R}^{C\times 4C}$ and $\mathbf{W}^{2} \in \mathbb{R}^{4C\times C}$.

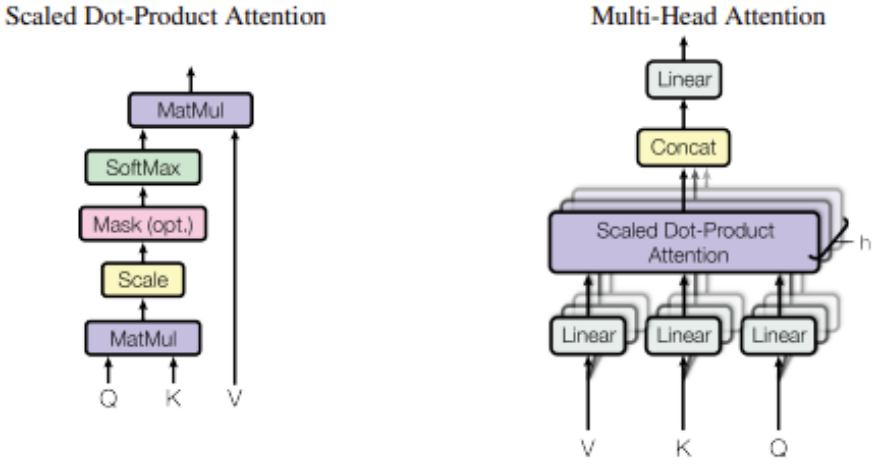

Figure 4: Attention mechanism based on dot-product similarity and whole Attention Layer. [14]

We will use Attention to build a model able to produce input-adaptive depth prediction. In the following algorithm, "bins" are interchangeable with latents, since the actual depth value bins are produced after the projection to scalar values.

1. You will "extract" the initialization of the bins from the bottleneck feature maps with a pooling layer and flattening (see *LatentsExtractor* `source/models/model_parts.py`). To control the number of depth bins you can check the argument `--num_bins` in `source/utils/config.py`.
2. Process the bins as tokens with, e.g., 3, Transformer Layers. You can decide the number of heads and the MLP's expansion with the arguments `--expansion` and `--num_heads`.
3. Compute the similarity between the bins and the projected feature map and apply softmax to it to obtain the probabilities that each pixel falls in each bin.
4. Obtain the bins' depth values by projecting them to scalar dimensionality, i.e., $D = 1$.
5. Compute the final depth as the weighted sum of the bins' scalar values, where the weights are the aforementioned probabilities.

The problem is divided into the following sub-tasks:

1. *Attention* (10pt).
   (a) *Implementation* (6pt): Implement Self-Attention in `source/models/models_parts.py`

(b) *Theoretical Questions* (4pt):

i. What is the scaling, i.e., "big O", of attention mechanism in B, N, D, H.

ii. What is a possible solution for the high memory requirements induced by the Attention mechanism?

iii. Why the attention mechanism, especially the combination "weights" calculation, cannot be factorized in more efficient and cheaper operations?

2. *Adaptive Depth* (5pt): Implement in `source/models/model_adaptive_depth.py` the adaptive depth network as described step-by-step above.

Roughly expected validation performance: `SILog-RMSE`: ≤ 28

Problem 5. Open Challenge (5 points)

In this task, you can try your own ideas to improve MTL for semantic segmentation and depth estimation.

This task will be graded based on the following criteria:

- Your idea significantly improves the MTL performance over a baseline network without that change. The comparison of your improved model and the baseline has to be fair in the sense that no confounders are introduced. For example, if you train your new model with a higher batch size or more iterations, you have to do the same for the baseline. If your idea fails to improve the MTL performance and you do not have time to try another one, please report it nevertheless. You can still earn points based on the other grading aspects.
- Your idea requires to modify the source code of the framework. Mere changes of configuration flags will not count as a sufficient contribution. Please include the relevant code in the report. If you have many small changes across different files, you can also provide the color-coded (git) diff[2].
- An in-depth analysis of the chosen approach is provided. If your idea consists of multiple components, please ablate them to show the effect of each. If your idea introduces hyperparameters, please study their influence. You should discuss possible reasons why your idea works. These should be backed up with an experimental analysis. It should go beyond just discussing the multitask metric.
- Related works of your idea are discussed. What are the similarities and differences?

Below are a few ideas that you can use as a starting point. You can choose which to try, pick an idea from another paper, or come up with your own idea. Please note that we do not guarantee that all ideas below give a significant performance improvement.

- Test-time augmentation methods aim to improve the prediction quality from a trained model by aggregating the predictions across transformed versions of a test input. One of the most widely used techniques in test-time augmentation is multi-scale testing, which is adopted in Deeplab models [4], and can date back to [6]. The basic implementation can be summarized as follows: multi-scale versions of the same test image are fed to the trained model. The predictions are then aggregated (average or maximum response for each position) across scales to make the final predictions. The choice of scales can be different for different vision tasks. Can this popular method also boost performance for MTL? Are there better aggregation approaches? Will additional augmentation methods further improve test-time augmentation performance? Note that it is not necessary to train a model for this task. You can use one of the previously trained checkpoints.

- The DeepLab architecture utilizes just one skip connection from the encoder at 4x scale. In the code provided, the encoder output is a feature pyramid with every scale present. Would it make sense to add further skip connections?

- Transformers have gained increasing attention in computer vision in the last few years. More specifically, can you come up with a symmetrical (with appropriate changes, i.e., classes cannot be "adaptive" in Segmentation but are predefined by design) structure of the one implemented in Problem 4 for Segmentation and apply MTL?

[2] https://tex.stackexchange.com/questions/105995/is-there-a-ready-solution-to-typeset-a-diff-file

References

[1] Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv preprint arXiv:1607.06450 (2016)

[2] Bhat, S.F., Alhashim, I., Wonka, P.: Adabins: Depth estimation using adaptive bins. In: IEEE Conf. Comput. Vis. Pattern Recog. pp. 4009–4018 (2021)

[3] Chen, L.C., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.L.: Semantic image segmentation with deep convolutional nets and fully connected crfs. arXiv preprint arXiv:1412.7062 (2014)

[4] Chen, L.C., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.L.: Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. IEEE Trans. Pattern Anal. Mach. Intell. **40**(4), 834–848 (2017)

[5] Chen, L.C., Papandreou, G., Schroff, F., Adam, H.: Rethinking atrous convolution for semantic image segmentation. arXiv preprint arXiv:1706.05587 (2017)

[6] Chen, L.C., Yang, Y., Wang, J., Xu, W., Yuille, A.L.: Attention to scale: Scale-aware semantic image segmentation. In: IEEE Conf. Comput. Vis. Pattern Recog. pp. 3640–3649 (2016)

[7] Chen, L.C., Zhu, Y., Papandreou, G., Schroff, F., Adam, H.: Encoder-decoder with atrous separable convolution for semantic image segmentation. In: Eur. Conf. Comput. Vis. pp. 801–818 (2018)

[8] Fu, H., Gong, M., Wang, C., Batmanghelich, K., Tao, D.: Deep ordinal regression network for monocular depth estimation. In: IEEE Conf. Comput. Vis. Pattern Recog. pp. 2002–2011 (2018)

[9] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conf. Comput. Vis. Pattern Recog. pp. 770–778 (2016)

[10] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Adv. Neural Inform. Process. Syst. pp. 1097–1105 (2012)

[11] Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: IEEE Conf. Comput. Vis. Pattern Recog. pp. 3431–3440 (2015)

[12] Neven, D., De Brabandere, B., Georgoulis, S., Proesmans, M., Van Gool, L.: Fast scene understanding for autonomous driving. arXiv preprint arXiv:1708.02550 (2017)

[13] Vandenhende, S., Georgoulis, S., De Brabandere, B., Van Gool, L.: Branched multi-task networks: deciding what layers to share. arXiv preprint arXiv:1904.02920 (2019)

[14] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need (2017), https://arxiv.org/pdf/1706.03762.pdf