École Polytechnique Fédérale de Lausanne

Conditionally Adaptive Multitask Learning

by Elif Sema Balcioglu

Master Project Report

Approved by the Examining Committee:

Prof. Dr. Sabine Süsstrunk
Thesis Advisor

Deblina Bhattacharjee
Thesis Supervisor

EPFL IC IVRL
CH-1015 Lausanne

June 12, 2023

Abstract

Multitask Learning (MTL) has emerged as a framework to deal with the problems single networks per task approach presents. These problems include ignoring useful information from training signals of different tasks and memory inefficiency. MTL helps these problems by optimizing for more than one objective and brings advantages such as better generalization and reduced memory footprint. In this work, we extend the traditional multitask mechanism with conditionally adaptive modules, which produce different sets of parameters for different tasks in a multitasking network to achieve more efficient parameter sharing. These modules include conditional attention, conditional adapter, and conditional layer normalization. We show the performance of these models with a series of experiments, which gradually explore the effectiveness of various architectural choices.

Contents

Abstract **2**

1 Introduction **5**

2 Related Work **7**

3 Method **12**
3.1 Multitask Learning Setup . . . 12
3.2 Conditionally Adaptive Modules . . . 14
3.2.1 Conditioning . . . 14

4 Experimentation Details **19**
4.1 Datasets . . . 19
4.1.1 The NYU Depth Dataset V2 . . . 19
4.1.2 Taskonomy Dataset . . . 20
4.2 Metrics . . . 20
4.3 Loss Functions . . . 21
4.4 Training Settings . . . 22
4.5 Results . . . 23
4.5.1 Baseline Results . . . 24
4.5.2 Adding Conditioned Modules . . . 25
4.5.3 Multitask Weighting . . . 28
4.5.4 Effect of Layer Normalization . . . 29
4.5.5 Effect of Adaptive Layer Size . . . 30
4.5.6 Same Batch vs Different Batch . . . 32
4.5.7 Evaluating Task Embeddings . . . 33
4.5.8 Taskonomy . . . 34

5 Conclusion **36**
5.1 Failure Cases and Limitations . . . 36
5.2 Future Directions . . . 36
5.3 Conclusion . . . 37

Bibliography **38**

Chapter 1

Introduction

In machine learning, the goal is to get the best possible performance on a particular task measured by a metric. This is typically done by training a single model dedicated to the task and iteratively finetuning the model. However, there are multiple disadvantages to this approach. While it often yields good results, it does not use valuable information that may be beneficial for improving the desired metric. Specifically, this approach ignores the information that may come from training clues of other related tasks. Shared representations between these tasks can help the model generalize better, leading to better performance [24].

Another disadvantage of this approach is its inefficiency in terms of memory requirements. It is expected that tasks in the same domain share some basic domain information, however, when using separate networks, this sharing mechanism is overlooked, leading to inefficient storage. [31] supports this claim by studying task affinities, which are the similarities of features used by different tasks, and discovers that tasks start to differentiate in later layers and most of the information is shared among the first layers.

Multitask Learning (MTL) has emerged as a framework to deal with these problems. In general, MTL refers to optimizing for more than one objective. It has been shown as an effective mechanism to improve performance in a variety of areas such as natural language processing, computer vision, and speech recognition.

Multitask learning has various advantages. First, it can be seen as an inductive transfer [24], by introducing an inductive bias provided by the auxiliary tasks, it causes the model to prefer hypotheses that explain more than one task, leading to better generalization. It enables attention focusing, by helping the model focus on features that matter by providing evidence for the relevance or irrelevance of those features using the training signal of other tasks. For example, [26] studies the interactions of various computer vision tasks and sees that semantic segmentation can gain up to 5% gain when trained with other tasks such as surface normal detection.

It is also advantageous in terms of efficiency. As some layers are shared between tasks, the resulting memory footprint is reduced. It reduces inference time, with a less repeated calculation of shared features.

Generally, MTL architectures consist of a shared encoder, and task-specific decoders, as can be seen in Figure 2.1. In these models, differentiation between the tasks starts at the decoders, and the information extracted from the encoder is shared exactly among all tasks. Although this is an established approach, it limits the model in the initial layers by preventing it from learning task-specific features. To overcome this issue, we use *conditioning* and enable the model to start differentiation at the encoder level. Conditioning refers to modulating the computation carried out by a model by information extracted from an auxiliary input, in our case, we propose to condition the encoder features for each task, enabling the model to learn task-specific features starting from the initial layers. As conditioning uses shared representations, we do not lose the advantages of multitask learning but increase the model's capacity to learn task information.

We use Swin Transformer [13] as the backbone architecture. The concept of transformers was originally introduced for natural language processing and was extended to handle image data with the Vision Transformer [5] (ViT) model. Images are represented by the ViT model as a series of patches that are fed into a transformer architecture. ViTs capture long-range interactions and global dependencies between patches by using self-attention processes. Swin Transformer is an improved version of ViTs, which solves the problems of increased complexity when handling dense predictions, with a hierarchical structure and shifted window mechanism, enabling the transformer architectures to be used in high-level vision tasks.

On top of the modules of the Swin Transformer, we propose various conditioned modules as in [20]. *Conditional attention*, which affects the original attention matrix, allows tasks to have different attention mechanisms, giving the tasks the chance to attend to the tokens within a window differently. *Conditional adapters*, which are small trainable modules inserted into specific layers of the network, enable the network to learn to use the pre-trained information differently for each task, with the intuition "most important layers for a given task appear at specific positions" [29]. *Conditional layer normalization* accounts for task-specific rescaling and allows more fine differentiation based on tasks. Details of these modules can be found in Section 3.2.1.

We conduct a series of experiments to evaluate these modules. Initially, we obtain baseline results by evaluating the model without any additional modules. Subsequently, we explore different architectures that varied in their configuration of conditioned models. By comparing the performance of these different architectures, we aim to assess the effectiveness of these modules in overall model performance. We show these results quantitatively and qualitatively, showing the advantages of using conditioning to improve multitask learning.

Chapter 2

Related Work

Vision Transformers

Vision Transformers (ViT) gained attention in recent years as a new approach for image classification and other computer vision tasks. The concept of transformers was originally introduced for natural language processing and extended to handle image data with the ViT model. The ViT architecture proposed by [5] shows impressive performance on image classification benchmarks.

Images are represented by the ViT model as a series of patches that are flattened and fed into a typical transformer architecture. ViTs capture long-range interactions between picture patches and global dependencies with attention processes.

Despite the success of ViTs, they face challenges in handling dense prediction tasks efficiently due to their nature and quadratic computational complexity coming from global dependencies. There exist many vision tasks such as semantic segmentation that require dense prediction at the pixel level. However, applying transformers directly to high-resolution images for this task becomes impractical due to this complexity.

A variety of different transformer approaches on top of ViTs are proposed to address their flaws and enhance performance. DeiT (Data-efficient Image Transformers) [30] was aimed to reduce computational and memory requirements. Another work, the Swin Transformer [13] model was also introduced to handle pixel-level predictions.

Swin Transformer utilizes a hierarchical structure that divides the image into non-overlapping patches at multiple scales, after starting from small-sized patches, they gradually merge neighboring patches in deeper transformer layers. With the concepts of hierarchical self-attention and shifted window mechanism the model captures both local and global dependencies efficiently and has linear computational complexity to image size.

Its model achieves state-of-the-art performance on various image benchmarks such as depth estimation and semantic segmentation which are dense prediction tasks. Its architecture allows it to handle such tasks in high-resolution images without excessive computational costs. More details on the architecture of Swin Transformer can be found in Section 3.1.

In this work, we also use pre-trained Swin Transformer architecture as the backbone of our models. In the original implementation to make dense predictions, authors use decoders with various architectures such as UperNet [34] on top of the Swin backbone. In our work, we use a U-Net [23] architecture, which copies and mirrors the number of transformer layers of the backbone. This approach can also be seen on [1] and [2].

Multitask Learning

Multitask learning (MTL) is a machine learning paradigm that aims to improve the performance of multiple related tasks by jointly learning them. Unlike traditional single-task learning approaches that train separate models for each task independently, MTL uses the shared information and dependencies among tasks to improve learning.

MTL has been studied across a wide range of domains, including natural language processing (NLP) and computer vision. The idea behind MTL is that by training on multiple tasks simultaneously, the model can learn robust and generalizable representations that capture the common patterns across tasks.

Historically, multitask learning is performed by learning shared representations from multitask signals. These attempts can be divided into two subcategories in terms of the way they share parameters of the networks: *hard and soft parameter sharing* [24]. Hard parameter sharing models generally consist of a shared encoder with task-specific heads, however in soft parameter sharing, tasks have their own set of parameters shared with a cross-talk mechanism. These ways of sharing parameters can be seen in Figure 2.1. While UberNet [10] was one of the first hard parameter sharing models to tackle a large number of low and high-level vision tasks, Cross-stitch networks [14] introduced soft-parameter sharing in deep multitask learning models.

A survey [32], improves this classification by introducing a new taxonomy based on where the task interactions take place. This criterion introduces two types of architectures: *encoder-focused* and *decoder-focused*. Encoder-focused architectures share information only in the encoder stage, with either hard or soft parameter sharing, and have differentiated task decoder heads. Decoder-focused models also exchange information during the decoding stage.

While we can see convolution neural network-based architectures (CNN) in multitask learning setups, the use of transformers is limited. [26] uses a CNN-based architecture to propose a framework to evaluate task performances when learned together or separately. [3] focuses on low-level computer vision tasks such as denoising or super-resolution. [15] addresses two high-level tasks of

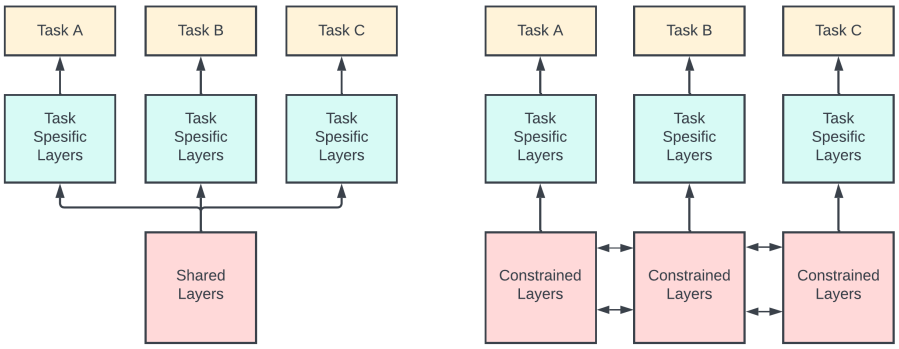

Figure 2.1: **Two parameter sharing architectures, hard and soft parameter sharing respectively**

object detection and semantic segmentation. Another work [25] focused on videos. [1] uses Swin Transformers in an encoder-decoder sense to predict multiple high-level vision tasks. It differs from other work by using a shared attention mechanism modeling the dependencies across the tasks, following a decoder-focused approach.

Adapters

Adapters are plug-in modules added in between layers of a pre-trained network. They have emerged as a technique for enhancing model flexibility, efficiency, and transferability. Two main advantages they offer are efficient training and easy domain adaptation. In this section, we discuss relevant works that have explored the concept of adapters and their applications in deep learning.

Most of the initial work was in the natural language processing field. [8] used adapters as a method for parameter-efficient customization. In order to enable task-specific modifications without significantly changing the parameters of the original model, they use adapters as tiny auxiliary modules put between the layers of a pre-trained model. This model can then be fine-tuned for certain downstream tasks while maintaining the knowledge acquired during the pre-training stage by incorporating task-specific adapter modules.

The concept of adapters has also been applied beyond natural language processing tasks. For instance, [22] extended the idea of adapters to computer vision tasks and introduced Vision Adapters. Vision Adapters enable task-specific modifications in convolutional neural networks, allowing for efficient customization and transfer learning in visual recognition tasks. [28] introduce adapter-based parameter-efficient transfer learning techniques for vision-and-language (VL) tasks to efficiently tune language models on diverse downstream VL tasks while achieving performance comparable to full fine-tuning.

In recent work, [4] proposes the Vision Transformer Adapter (ViT-Adapter) as a pre-training-

free additional network that can adapt the plain ViT to downstream dense prediction tasks without modifying its original architecture. Although some of these models explore adapters on transformers, our work focuses on Swin Transformer as a difference.

Conditioning

Conditioning refers to modulating the computation carried out by a model by information extracted from an auxiliary input [6]. It has been used in a wide range of areas such as visual question-answering and generative domains.

Conditional generative models use conditioning to generate data based on specified conditions. [21] uses concatenation-based conditioning to affect the class of generated image. [16, 17] make generative modeling of images and audio, respectively, and use conditional biasing. Conditional domain adaptation techniques have been used to adapt models to different domains. [12] updates the per-channel batch normalization statistics of a network trained on one domain with that network's statistics in a new, target domain.

Another conditioning mechanism, feature-wise transformations are commonly employed in deep neural networks to enhance network performance.

[19] proposed an approach for feature-wise image generation and manipulation using conditional models. They introduced Feature-wise Linear Modulation (FiLM), a technique that conditions the generator network by modulating the activations of individual feature maps. By enabling fine-grained control over individual feature maps, FiLM provides a framework for manipulating visual representations. By applying a feature-wise affine transformation based on conditioning parameters, FiLM allows explicit control over the style and appearance of generated images. More details on the FiLM model can be found in Section 3.2.1. FiLM has also been utilized in the context of visual question-answering (VQA) tasks. The authors showed its effectiveness in conditioning the VQA model to affect specific image regions based on question content. The FiLM technique has been widely adopted in subsequent research to enable fine-grained control over various image-generation tasks. In our work, we use FiLM layers to condition the multitask encoder based on different tasks.

[27] introduces a method that applies a conditional feature-wise transformation over the convolutional activations, with *task-routing layers*, which masks the convolutional layer output, allowing specialized subnetworks per task with lower dimensionality compared to the main model. This approach is similar to ours in the sense that it also uses task-specific conditioning, however, differs in the way this conditioning is used. While they allow only a conditional binary mask over convolution layers, we affect the whole output of the transformer outputs.

In a similar work, [20] introduces a novel approach for Natural Language Understanding (NLU) tasks using a FiLM-conditioned framework. The authors propose a Conditional Adaptive Multi-Task Learning (CAMTL) approach that adjusts the shared model's parameters based on task-specific

embeddings. The authors leverage a hypernetwork to generate FiLM parameters, which consists of FiLM layers that generate the parameters of the shared model conditioned on the specific task, enabling the model to effectively capture task-specific patterns and improve performance. This work uses the adaptive mechanisms we also use such as conditional adapter and attention but in the context of natural language processing.

Chapter 3

Method

Our model follows an encoder-decoder based multitask learning architecture. Alongside the classical architectures, it is further extended with conditionally adaptive modules. In this section, the details of these modules are explained.

3.1 Multitask Learning Setup

Most multitask learning approaches can be divided into two categories depending on the sharing of hidden layers: hard or soft parameter sharing. In our architecture, we follow the hard parameter-sharing scheme with an encoder-focused architecture as in the first figure of Figure 2.1. Given K different tasks, our architecture consists of an encoder and K task-specific decoders for each task. While the parameters of the encoder are shared, task-specific decoders are differentiated for each task. The decoders have the same architecture, and task-specific heads are appended to account for task-specific requirements. For example, while a N class segmentation requires a N channel output, depth prediction requires a single channel output.

As the baseline architecture, we use Swin Transformer [13]. For the encoder module, we use Swin-S configuration, which applies four stages of a variable number of transformer blocks to features of gradually decreasing resolution in a hierarchical manner.

The first stage starts by dividing the input image into patches of $4x4$ pixels, then mapping them to $C = 96$ dimensional tokens using the *patch embedding* layer. After this initial step, the tokens are divided into non-overlapping square windows of size $MxM = 7x7$, which are used to perform the intra-window self-attention independently. This design only allows each token to attend to tokens in its own window, which restricts the model to have global or long-range interactions. To overcome this problem, authors suggest a *shifted window approach*, which alternates between two

configurations in consecutive Swin Transformer blocks. After the initial regular window partitioning which starts from the top-left, windows are shifted by displacing the windows by $(\lfloor \frac{M}{2} \rfloor, \lfloor \frac{M}{2} \rfloor)$ pixels. To allow this alternating window attention, each stage consists of an even number of transformer blocks. In the configuration Swin-S, Stage-1 consists of 2 transformer blocks. Stage-2 starts with a *patch merging* step to produce a hierarchical representation. The number of tokens is reduced by a multiple of 4, by concatenating the features of 2 by 2 neighboring patches with dimension C. The resulting $4C$-dimensional features are mapped to $2C$-dimensional features using linear layers. This step therefore maps the $(\frac{H}{4}, \frac{W}{4}, C)$ dimensional features coming from the first stage to $(\frac{H}{8}, \frac{W}{8}, 2C)$ dimensional features. Then, the shifted attention mechanism explained above is performed in the 2 transformer blocks of Stage-2. Stage-3 starts with a patch merging layer to map the features to $(\frac{H}{16}, \frac{W}{16}, 4C)$ dimensional space, and apply the shifted attention with its 18 transformer blocks. The final stage, Stage-4, maps the features to $(\frac{H}{32}, \frac{W}{32}, 8C)$ dimensional space, and applies the shifted attention with its 2 transformer blocks. These gradually decreasing resolutions produce a hierarchical representation, with the same feature map resolutions as those of typical convolutional networks such as ResNet [7].

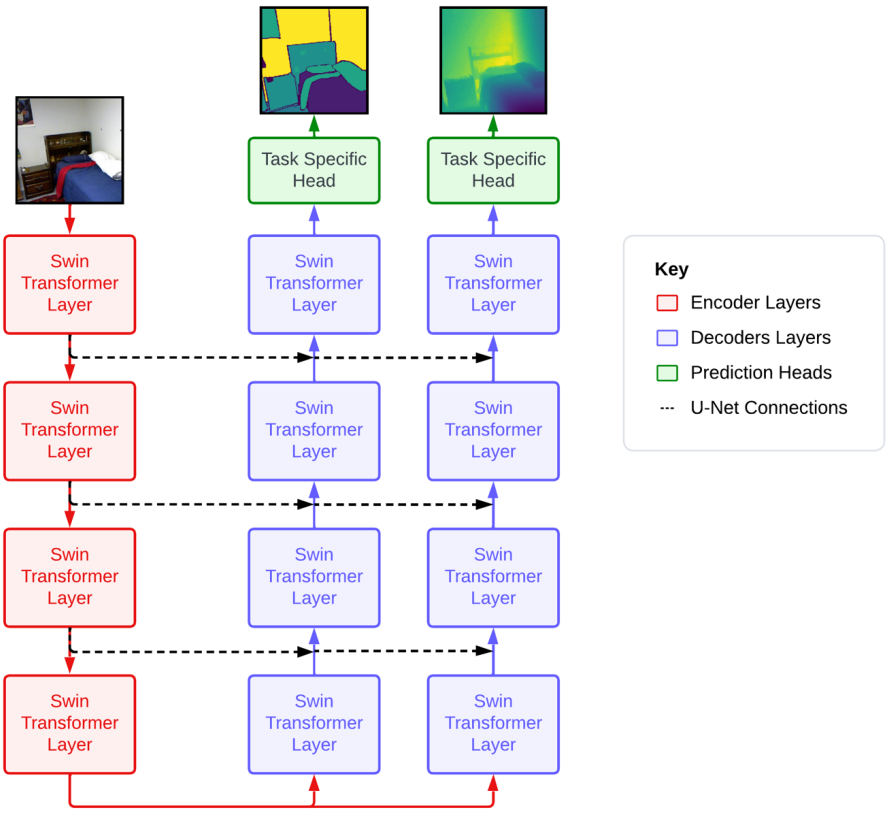

Figure 3.1: **Our multitask architecture** The architecture consists of a shared encoder and separate decoders, with task-specific heads attached to each decoder.

After decreasing the resolution of the image, to make predictions for dense, pixel-level tasks such as segmentation or depth, we use a *U-Net* [2, 23] architecture, which mirrors the encoder architecture to gradually increase the feature resolution. We also add skip-connections from encoder blocks to decoder blocks to help with training. We then add task-specific heads to each of the decoders, which maps the output to produce a correct shape. A figure of our architecture can be seen in Figure 3.1.

3.2 Conditionally Adaptive Modules

If we think about training for multiple tasks, on one end of the spectrum we have a separate network for each task, without any parameter sharing or interaction between each other. This approach focuses on a single task, but does not take advantage of the information that can come from the training signals of related tasks, which is shown to improve the generalization.

On the other end, we have the mentioned parameter-sharing network, which has an encoder whose outputs are used by all tasks, and a decoder that is used to differentiate between tasks. Although it uses information coming from other tasks, it has a strict parameter sharing, which does not allow tasks to use information coming from other tasks differently. To move to the middle of the spectrum and allow for task-specific usage of collective information, we use conditionally adaptive modules which modulate features of the encoder differently for each task, without losing the multitask learning advantages.

3.2.1 Conditioning

Conditioning refers to modulating the computation carried out by a model using information extracted from an auxiliary input. There are various ways to condition a computation such as concatenating or summing with the input. In our work, we use Feature-wise Linear Modulation (FiLM) [19] layers. Given the conditioning input z, and the feature to be conditioned x, FiLM learns functions f and h which output γ and β as a function of z.

$$\gamma = f(z) \qquad \beta = h(z)$$

These values are then used to modulate x:

$$FiLM(x) = \gamma \cdot x + \beta$$

This operation can be seen as a conditional affine transformation, where multiplication with γ is conditional scaling and adding with β is conditional biasing. Functions f and h can be learned in various ways where using linear layers can be an example.

In our setting, given we want to condition based on task definitions, we use *task embeddings* as conditioning input z. These embeddings are extracted with an embedding layer, which has a size of the dictionary equal to the number of tasks. This embedding layer is also trained to learn task representations in terms of these embeddings.

One point worth noting is the change in inference mechanism compared to traditional multitask learning approaches. As we condition the network based on the task, we do not have the batching mechanism of the regular multitask networks where you get the output for all tasks in one forward pass. Instead, for all tasks, separate forward passes are needed with corresponding task embeddings since the output of the encoder is not the same for tasks.

We use three modules that use conditioning mechanism to affect the encoder generation with task embeddings: conditional attention, conditional adapter, and conditional layer normalization.

Conditional Attention

The first conditioned module is conditional attention. The regular attention called "Scaled Dot-Product Attention" [33] is defined with the query Q, the key K, the value V matrices and calculated as follows, given the input dimension d:

$$Attention(Q,K,V) = softmax\left[\frac{QK^T}{\sqrt{d}}\right]V$$

In this calculation, QK^T matrix has shape MxM where M is the maximum sequence length. For example, if attention is calculated for a Swin Transformer with a window of size $7x7$, $M = 7*7$ and QK^T has shape $49x49$.

In our model, this mechanism is adapted to include the conditioning using the task embedding z:

$$Attention(Q,K,V) = softmax\left[M(z) + \frac{QK^T}{\sqrt{d}}\right]V$$

$M(z)$ which is added to the attention mechanism can be called *conditional attention matrix*. It depends on the task embedding z and affects the softmax function by changing the way tokens attend to each other differently for each task, as can be seen in Figure 3.2. $M(z)$ has the same shape as the QK^T. It is a block diagonal matrix, where only tokens can only attend to other tokens in its neighbor of length L. Therefore, $M(z)$ has $49/L$ blocks, placed along its diagonal, and outside of these blocks are zero and do not affect the original attention matrix QK^T. Each of these blocks A' is found by conditioning learnable matrices A of size LXL by the task embeddings. More formally given a task embedding z:

$$A(z)' = FiLM(A) = A\gamma(z) + \beta(z)$$

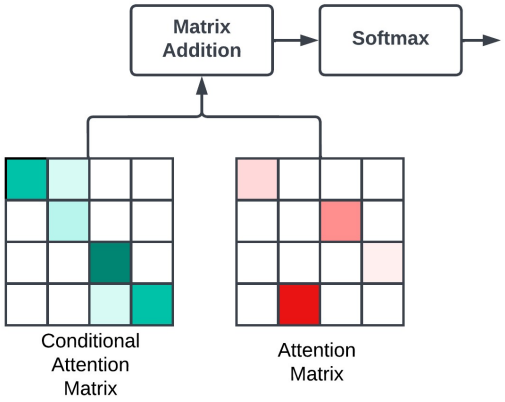

Figure 3.2: **Conditional attention mechanism** Learned task-dependent "Conditional Attention Matrix" is added to the original "Attention Matrix" before the softmax calculation.

This approach provides localized attention, [20] suggests that using localized attention instead of full attention where all elements of $M(z)$ are calculated provides better performance. We also follow this localized attention.

The intuition behind this module is to allow tasks to have different attention mechanisms, giving the tasks the chance to attend to the tokens within a window differently. For example, segmentation and depth have different losses, while one is a classification other is a regression task, these tasks may have different optimal ways they use information from the neighbor cells, and with conditioned attention, we allow the network to learn these differences.

Conditional Adapter

The second conditional module is adapters. They act as lightweight and modular components that can be inserted into specific layers of a deep neural network, allowing the network to perform additional tasks or learn new representations without requiring extensive training or modification of the existing parameters. In our case, we use adapters to take advantage of the pre-trained encoder and learn multitasking without training the whole network.

While the adapters can be fully connected layers, convolutional layers, or any other architecture suitable for the given task, we use two linear layers, in the bottleneck sense. These adapters can be seen as a separate flow alongside the original network where the outputs of the original layers are added to the information learned using the adapters. After each transformer layer, the output of the transformer is added to the flow of the adapter as seen in Figure 3.3. The weights of these bottleneck layers are conditioned based on the task embedding as in the conditional attention.

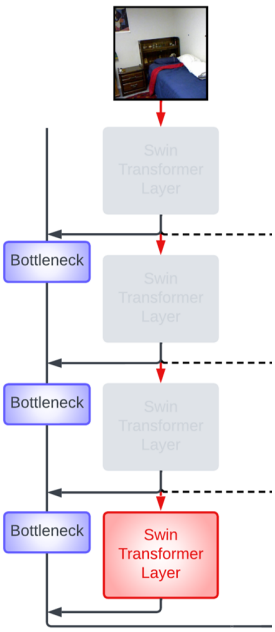

Figure 3.3: **Conditional adapter** "Conditional Adapter" is a separate flow alongside the original network where the outputs of the original layers are added to the information learned using the adapters.

When training using the adapters, only the adapters and a subset of the transformer layers whose attention matrices are conditioned are unfrozen for training. The remainder of the encoder layers are not trained. This way network has fewer parameters to train, but also does not use the frozen layers as is and has the chance to learn with the help of adapters.

This component follows observation from [29] that showed that “most important layers for a given task appear at specific positions”. With the trained and conditioned layers of the adapter, the network can learn to use the pre-trained information differently for each task.

Conditional Layer Norm

The third and last component conditioned is layer normalization as proposed in [20]. Given inputs x with length K, the original layer norm formulation is as follows:

$$\mu = \frac{1}{K}\sum_{k=1}^{K} x_k$$

$$\sigma^2 = \frac{1}{K}\sum_{k=1}^{K} (x_k - \mu)^2$$

$$\hat{x}_k = \frac{x_k - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$y_i = \gamma' \hat{x} + \beta' \equiv LN_{\gamma,\beta}(x)$$

First, the mean and the variance of each element of x is calculated. Then each sample is normalized such that the elements in the sample have zero mean and unit variance. In the last step, there is a scaling and shifting step where γ and β are learnable parameters. In our formulation, the final layer, which also resembles FiLM layers, is changed to a conditioning mechanism where learned γ and β are conditioned by the task embedding z:

$$y_i = \gamma(\hat{z})\hat{x} + \beta(z) \quad \gamma(\hat{z}) = \gamma' * \gamma(z) + \beta'$$

Where γ' and β' are learned parameters of layer normalization, $\gamma(z)$ and $\beta(z)$ are parameters of the FiLM layer as explained in Section 3.2.1.

This module was included to account for task-specific rescaling and to allow more fine differentiation based on tasks.

Chapter 4

Experimentation Details

4.1 Datasets

Although there are various possible datasets over which we can train and test our algorithms, To test our approach, we use two datasets: NYUv2 and Taskonomy. Details of these datasets can be found in this section.

4.1.1 The NYU Depth Dataset V2

NYU Depth Dataset V2 (NYUv2) is an established multitask learning dataset with labels for both semantic segmentation and depth captured using Microsoft Kinect devices. An example of labeling can be seen in Figure 4.1.

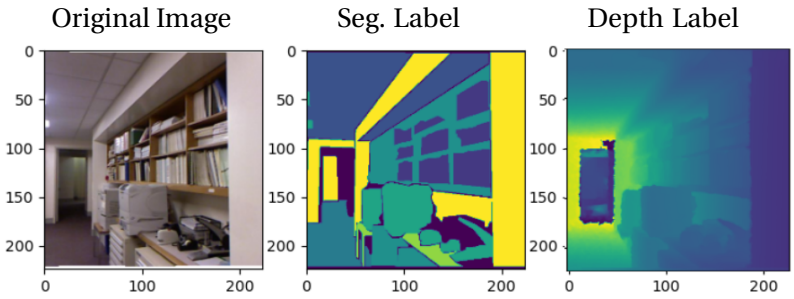

Figure 4.1: **Example labeling from NYUv2 Dataset** Shows the original image, semantic segmentation task label, and depth estimation task label respectively.

It consists of 1449 densely labeled pairs of aligned RGB and depth images. These 1449 pairs

are further split into 795 training and 654 test data points. Among possible two tasks, semantic segmentation has 14 possible classes. These classes include background, bed, books, ceiling, chair, floor, furniture, objects, painting, sofa, table, tv, wall, and window.

4.1.2 Taskonomy Dataset

Taskonomy [35] is a multitask dataset that has labels for 18 tasks. The dataset includes over 4.5 million images from over 500 buildings. Given the large size of the full dataset, they provide 4 partitions (Tiny, Medium, Full, and Full+) with increasing sizes.

In this work, we use the Tiny version which consists of 272678 images in the train and 35631 images in the validation splits.

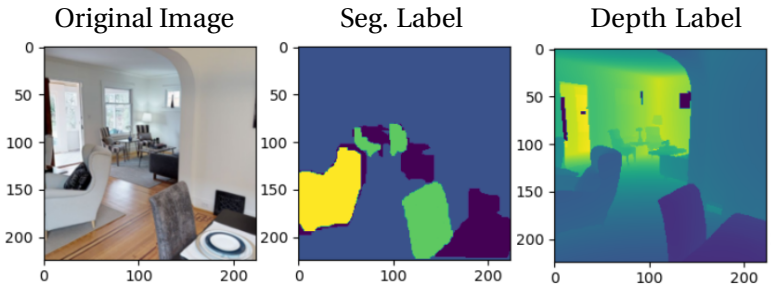

Figure 4.2: **Example labeling from Taskonomy Dataset** Shows the original image, semantic segmentation task label, and depth estimation task label respectively.

The semantic segmentation task has 18 classes, which include 16 object classes, a "background" class, and an "uncertain" class. These are a subset of MS COCO dataset classes. "Background" means that pixels belong to none of the 16 object classes. "Uncertain" means the classifiers had too low confidence for those pixels. Depth labels have units of 1/512m with a max range of 128m.

4.2 Metrics

While there are many tasks to test our models, as we started with the NYUv2 dataset, in our experiments we used semantic segmentation and depth estimation as two main tasks. Evaluation metrics are essential tools to quantitatively measure the performance of models or algorithms. While semantic segmentation is a multi-class classification problem, depth prediction is a regression problem, therefore they differ in evaluation metrics. To test our models we take advantage of the following metrics:

Semantic Segmentation

Mean Intersection Over Union

Intersection over Union (IoU) is a metric widely used in tasks such as object detection, semantic segmentation, and instance segmentation, which are mainly classification tasks. It measures the overlap between the predicted bounding box or segmentation mask and the ground truth annotation. In the case of binary or multi-class segmentation, the mean IoU of an image is computed by calculating the IoU for each class and then taking the average.

Depth Prediction

Delta Threshold

Delta threshold metrics measure the percentage of predicted depth values that fall within a certain threshold of the ground truth values. Common delta thresholds include Delta1 (d1) (percentage within 1.25), Delta2 (d2) (percentage within 1.25^2), and Delta3 (d3) (percentage within 1.25^3). We report our results using Delta1 (d1) metric.

4.3 Loss Functions

To train our networks for the two tasks we have we use two different loss functions. While semantic segmentation is a multi-class classification problem, depth prediction is a regression problem, therefore they differ in their loss functions. For semantic segmentation, as it is a multi-class classification problem, we use cross-entropy loss. For the depth prediction, we use reversed Huber (berHu) loss.

Semantic Segmentation

Cross Entropy Loss

Cross-entropy loss is a commonly used loss function in classification tasks that measures the similarity between predicted probabilities and true class labels. It aims to minimize the difference between predicted and true probabilities. The formula for cross-entropy loss involves taking the negative sum of the element-wise multiplication of the true class labels and the logarithm of the predicted probabilities. It provides a way to quantify the dissimilarity between predicted and true probabilities, guiding the model to better classification performance.

Depth Prediction

berHu Loss

The Huber loss is a regression loss function that combines the properties of the mean squared error (MSE) with L_2 loss and the mean absolute error (MAE) with L_1 loss. It is less sensitive to outliers

than the squared error loss. It is defined as follows where δ is the threshold value:

$$L_\delta(a) = \begin{cases} \frac{a^2}{2} & \text{for} |a| \leq \delta, \\ \delta \cdot \left(|a| - \frac{1}{2}\delta\right), & \text{otherwise.} \end{cases}$$

berHu [11, 18, 36] loss which is also called *reverse Huber* is a modification made to the Huber loss. Compared to the Huber loss, the condition for L_1 and L_2 reverses.

$$L_\delta(a) = \begin{cases} |a| & \text{for} |a| \leq \delta, \\ \frac{a^2+\delta^2}{2\delta} & \text{otherwise.} \end{cases}$$

Empirically, BerHu shows a good balance between the two norms; it puts a high weight on pixels with a high residual. In practice, a is given as the difference between the pixel labels and its predictions reciprocal to compensate for the problem of large errors. To make the prediction, the output of the network is passed from the sigmoid function and multiplied by the biggest possible reciprocal depth. When visualizing, these values are again changed back to regular depth values.

4.4 Training Settings

For all of the experiments, we used a single Tesla A100 GPU with 40GB of memory. Models are trained with AdamW optimizer with (0.9, 0.98) betas. We used the OneCycleLR scheduler of PyTorch with a maximum learning rate of 4e-5, with 10% percent of the cycle spent increasing the learning rate (pct_start). The behavior of the scheduler can be seen in Figure 4.3. We observed the best performance on around 1000 epochs.

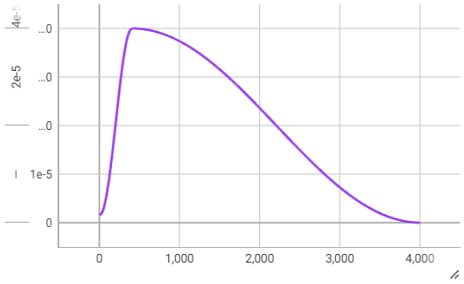

Figure 4.3: **OneCycleLR Scheduler**

For the training, several data augmentation techniques are used. As the images were variable sized and pre-trained Swin Transformer architecture takes images of size 224, *RandomResizedCrop* of PyTorch with size 224 is used. This augmentation crops a random portion of the image and resizes it to a given size of 224, with a scale between (0.5, 1.0). Images are then horizontally flipped randomly

with *RandomHorizontalFlip* using the default settings. These augmentations are applied to both the input image and the labels. We also applied *ColorJitter* to further augment the input RGB image. It randomly changes the brightness, contrast, saturation and hue of an image. Ranges are selected as $(0.8, 1.2)$, $(0.8, 1.2)$, $(0.8, 1.2)$ and $(-0.1, 0.1)$ for brightness, contrast, saturation and hue respectively. Given the size of the NYUv2 dataset, these augmentations are used to reduce overfitting.

4.5 Results

We conducted a series of experiments to evaluate the impact of using various modules in the multitask architecture. Initially, we obtained baseline results by evaluating the model without any additional modules. Later, we explored different architectures that varied in their configuration of conditioned models. By comparing the performance of these different architectures, we aimed to assess the effectiveness of incorporating these modules in improving overall model performance.

For each of the experiments, you can see an experiment card showing the differences in architectures with the template of Table 4.1.

Architectural Choice	Model 1	Model 2
Multitask	All	Some
Conditioned Modules	All	Some
Layer Normalization	Regular	Conditioned
Frozen Encoder	False	True
Adapter	False	True
Task Weighting	5-1	5-1

Table 4.1: **Example experiment card**

The details of the choices are as follows:

- **Multitask**: Whether the model is a multitask model trained jointly for multiple tasks or a separate network handling only one task. Possible choices: *True*, *False*
- **Conditioned Modules**: Swin Transformer consists of multiple transformer blocks divided into stages. In the *small* configuration which we use, there are 24 of these blocks, 2 in the first, 2 in the second, 18 in the third, and again 2 in the fourth stage. We experiment with various configurations of conditioning these modules. This configuration parameter represents these choices. All, None, or some of these blocks can be conditioned where the last one is shown with the number of conditioned transformer blocks for each stage, for example: "0-0-18-2" conditions only the last two stages. Possible choices: *All*, *None*, *0-0-12-0*
- **Layer Normalization**: Whether a regular or a conditioned layer normalization is used, as explained in Section 3.2.1. Possible choices: *Regular*, *Conditioned*

- **Frozen Encoder**: Whether the encoder is open for training or not. Possible choices: *True, False*
- **Adapter**: Whether there is adapters added to the model. Possible choices: *True, False*
- **Task Weighting**: The hyperparameter used to balance the task weights. It has the form semantic segmentation weight - depth estimation weight. Possible choices: *5-1*

The results of the experiments are recorded with the metrics explained in Section 4.2 section. The best-performing setting is written in bold. For these settings, unless otherwise specified, we use The NYU Depth Dataset V2, explained in Section 4.1.1, with a Swin Transformer base. The details of the training that are common in all settings can be found in Section 4.4.

For various model architectures tested below, the number of trainable parameters and the number of floating point operations per forward pass can be found in Table 4.2. We observe that while layer normalization and adapters affect the number of parameters much, the number of transformer blocks with conditioned attention has a smaller effect. Floating point operations (FLOPs) have similar behavior for all cases since separate forward passes are required for different tasks as explained in Section 3.2.1.

	Conditioned Modules	Number of Trainable Parameters	FLOPs for average Forward Pass
Single Task	-	57M	23.7B
Baseline Multitask	-	63.9M	23.7B
Frozen Encoder	Ad, LN, At (0-0-6-2)	49.8M	25B
Conditioned Multitask-1	Ad, LN, At (All)	88.1M	25B
Conditioned Multitask-1	LN, At (All)	76.4M	23.7B
Conditioned Multitask-2	At (All)	64M	23.7B
Conditioned Multitask-3	At (0-0-6-2)	63.9M	23.7B

Table 4.2: **Model size - inference statistics** Each trained network has a different number of parameters and floating point operations(FLOPs) due to architecture changes. These statistics can be found in this table. Conditional modules included are *Ad*: Conditioned Adapter, *LN*: Conditioned Layer Normalization, *At*: Conditioned Attention.

4.5.1 Baseline Results

For the purpose of establishing a baseline for comparison, we trained three models. Two of these models (Only Depth, Only Segmentation) were trained without any multitask approach and consisted of a single decoder. The goal behind these models was to see the individual performance of each task on its own, without any multitasking or information sharing. The third model (Baseline Multitask) was a multitask model, following the architecture described in Section 3.1, but without

any conditional mechanisms. In this model, two tasks shared an encoder without any conditioning, and they were predicted with separate decoders.

All parameters in these models were trainable, without any frozen parameters in both the encoder and decoders. By evaluating these models, we were able to observe the full performance potential of the model for the given tasks.

Architectural Choice	Only Depth	Only Segmentation	Baseline Multitask
Multitask	False	False	True
Conditioned Modules	None	None	None
Layer Normalization	Regular	Regular	Regular
Frozen Encoder	False	False	False
Adapter	False	False	False
Task Weighting	-	-	5-1

Table 4.3: **Baseline models experiment card**

	d1	Mean IOU
Only Depth	0.7586	-
Only Segmentation	-	0.4904
Baseline Multitask	**0.7793**	**0.4977**

Table 4.4: **Results for baseline models**

Based on the results presented in Table 4.4, it is seen that both models show improved performance when trained using a multitask learning approach compared to a single-task learning approach, although improvement in segmentation is minor. This was expected as multitask learning enables them to share representations and improve generalization. Example predictions can be seen in Figure 4.4.

4.5.2 Adding Conditioned Modules

The baseline results provide an evaluation of the base architecture without any conditioned modules added. We then proceed to examine the performance of conditioning by considering two possible architectures for evaluating the conditioning modules.

In our experimentation, we trained two different networks to explore the impact of conditioned adapters and frozen encoders. The first network (Frozen Encoder) used conditioned adapters, with a frozen encoder that included a limited number of trainable conditioned transformer blocks, as outlined in Section 3.2.1. The objective of this network was to assess the effectiveness of adapters on the frozen encoder and to determine if satisfactory performance could be achieved with frozen parameters.

On the other hand, the second network (Conditioned Multitask) utilized a trainable encoder

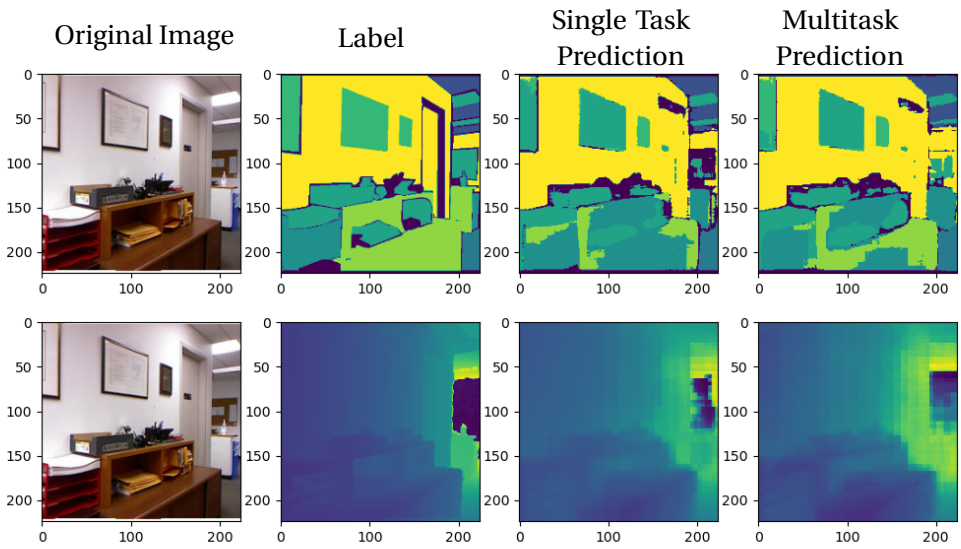

Figure 4.4: **Example prediction using baseline models** First column shows the original image, the second column shows the labels for both tasks, the third column contains the predictions of the single-task networks (Only Depth and Only Segmentation) and the last column shows the prediction of the Baseline Multitask network.

Architectural Choice	Baseline Multitask	Frozen Encoder	Conditioned Multitask	Swin Transformer Backbone Trained for Segmentation
Multitask	True	True	True	True
Conditioned Modules	None	0-0-6-2	All	0-0-6-2
Layer Normalization	Regular	Conditioned	Conditioned	Conditioned
Frozen Encoder	False	True	False	True
Adapter	False	True	False	True
Task Weighting	5-1	5-1	5-1	5-1

Table 4.5: **Conditional models experiment card**

without any frozen modules. As the main purpose of adapters is to introduce small trainable modules, they were not incorporated into this network. Aside from these modifications, the two networks shared the same underlying architecture. Both networks use other conditioned modules, namely conditional attention and conditional layer normalization.

By comparing the performance of these two networks, we aimed to investigate the effect of conditioned adapters and frozen encoders on the model performance and assess the trade-offs between using frozen parameters versus fully trainable encoders.

Upon analyzing the results in Table 4.6, it became clear that even with the incorporation of adapters, comparable performance to open training without frozen modules could not be achieved.

This experiment was the main motivation of this project and took most of the time to evaluate. As training large transformer models require careful fine-tuning, our initial approach was to try different hyperparameters, training strategies such as learning rate schedulers, or different data augmentation techniques to make frozen network work. But even with extensive experimentation on these settings, we were not able to get comparable or qualitatively pleasing results. We then trained a fully trainable network without adapters and this adjustment enabled us to achieve results that were comparable to the baseline outcomes. The decision to pursue open training proved crucial in obtaining the desired performance and improving the overall quality of the model's outputs. This outcome can be attributed to the fact that the Swin Transformer was not originally trained for dense prediction tasks. Therefore, the latent space mapped by the encoder fails to capture pixel-level information. Which makes the decoder's task of generating accurate outputs challenging or even impossible. Even the adapters are not enough to help with this issue, as they can only use the information from the frozen encoder modules and are limited by their expressiveness. This showed us the limitation of adapting computer vision models trained for low-level tasks on new high-level domains, which were not so evident in NLP works as in [20].

	d1	Mean IOU
Baseline Multitask	0.7793	0.4977
Frozen Encoder	0.5524	0.2666
Conditioned Multitask	**0.7826**	**0.4978**
Swin Transformer Backbone Trained for Segmentation	0.6947	0.3982

Table 4.6: **Results for conditional models**

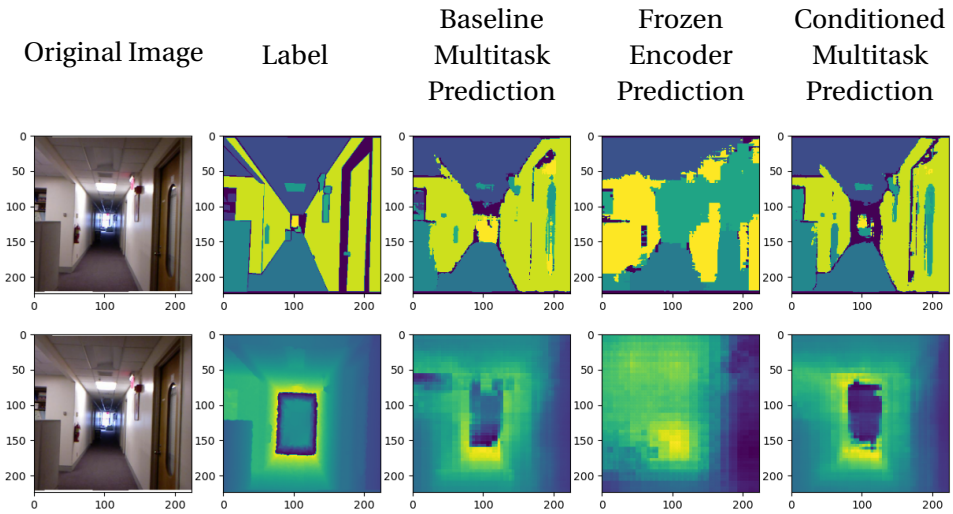

Figure 4.5: **Example predictions** Predictions are generated using multitask models in Section 4.5.2.

To test this claim further, we trained a new model (Swin Transformer Backbone Trained for Segmentation) where we used the backbone of a Swin Transformer, trained for the segmentation

task, but with a UperNet decoder architecture. Given the difference in the decoders, we only used the weights of the encoder from this model. Results can be seen in Table 4.6. We see that we get better results with this set of weights, but again not comparable to the fully trainable encoder. This result also supports our claim that the frozen encoder was limited by what it learned during pretraining, as the Swin Transformer trained for segmentation captures more pixel-level information.

A noticeable improvement in the depth task was observed in the conditioned network training compared to the baseline results, while comparable performance was obtained for the segmentation task. Based on these findings, for the rest of the experiments, we adopt the approach used in the second network, which does not have adapters or frozen modules. This decision is motivated by the understanding that the limitations of adapters and the issues related to the frozen encoder prevent achieving optimal performance in tasks requiring pixel-level information.

4.5.3 Multitask Weighting

Multitask learning poses a significant challenge in achieving a proper balance of loss and weighting across different tasks. The multitask models are extremely sensitive to the weights given to different tasks [9] and incorrect balancing can severely impact performance. This is mainly due to the variation in the type of losses and their scales, therefore without proper tuning of weights loss of one task can interfere with others leading to negative task transfer.

The critical aspect of addressing this challenge lies in finding the optimal hyperparameters between tasks, enabling effective learning, and preventing negative interference. An adequate weighing is crucial to ensure that each task receives useful signals during training, accounting for the distinct characteristics and importance of each task. Therefore, without further experiments, we tried to find the optimal set of weights our architecture needs. To find the optimal weight distribution of our network, we tested multiple weighting schemes, with the architectures in Table 4.7.

Architectural Choice	All Models
Multitask	True
Conditioned Modules	All
Layer Normalization	Conditioned
Frozen Encoder	False
Adapter	False
Task Weighting	see below

Table 4.7: **Weighting experiment card**

Indeed, the results of our experiments in Table 4.8 show the critical role of task weighting in achieving performance in multitask learning. The varying weights assigned to different tasks have an impact on the overall results, highlighting the significance of careful weight selection. Qualitative examples can be seen in Figure 4.6.

	d1	Mean IOU
Depth: 1 - Segmentation: 1	0.7649	0.478
Depth: 1 - Segmentation: 5	**0.7826**	**0.4978**
Depth: 1 - Segmentation: 10	0.78	0.4918
Depth: 1 - Segmentation: 20	0.7765	0.4909
Depth: 1 - Segmentation: 40	0.7714	0.495

Table 4.8: **Results for different weighting settings**

In our case, we find that assigning a weight of 5 to the segmentation task and a weight of 1 to the depth task yields the best performance. We also observe that the depth estimation task is more sensitive to the weight changes showing different characteristics of the tasks.

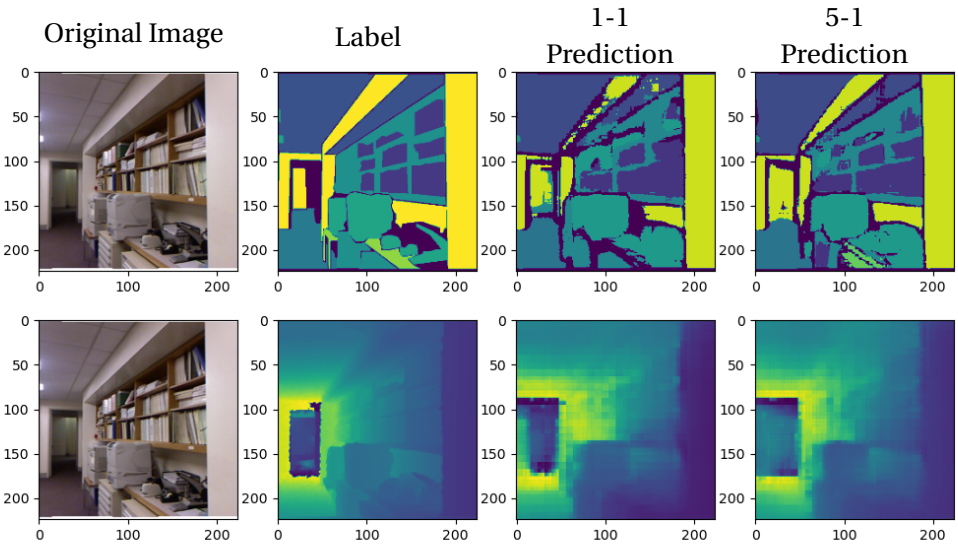

Figure 4.6: **Example prediction for two weighting options** Predictions for the worst (1-1) and best (5-1) performing models are shown.

4.5.4 Effect of Layer Normalization

In order to evaluate the impact of conditioned layer normalization layers in our model, we conducted an experiment with two identical models, differing only in the type of layer normalization layers employed. The purpose of this experiment was to assess the effect of this module on the overall performance.

The experiment comparing regular layer normalization layers (Regular Layer Norm) with conditioned layer normalization layers (Conditioned Layer Norm) revealed that regular layer normalization layers show better performance over the conditioned one as can be seen in Table 4.10.

Architectural Choice	Conditioned Layer Norm	Regular Layer Norm
Multitask	True	True
Conditioned Modules	All	All
Layer Normalization	Conditioned	Regular
Frozen Encoder	False	False
Adapter	False	False
Task Weighting	5-1	5-1

Table 4.9: **Layer normalization experiment card**

	d1	Mean IOU
Conditioned Layer Norm	0.7826	0.4978
Regular Layer Norm	**0.788**	**0.5032**

Table 4.10: **Results for two layer normalization settings**

I suspect this is due to the difference in weights when training. When training with conditioned layer normalization layers, the weights are initialized from scratch without incorporating information from the pre-trained network. This initialization process causes the model to lose the valuable information acquired during pretraining. On the contrary, we train on top of the pre-trained layer normalization weights in the regular case.

We did not observe similar behavior in conditional attention. This is because conditional attention does not directly impact the original attention matrix but rather introduces a new matrix on top of it. As a result, the conditioned attention module can leverage the existing knowledge obtained during pretraining without the risk of losing valuable information.

4.5.5 Effect of Adaptive Layer Size

The Swin Transformer architecture is composed of multiple transformer blocks organized into stages. In the specific configuration we utilized, there are a total of 24 transformer blocks distributed across the different stages. Specifically, there are 2 transformer blocks in the first stage, 2 in the second stage, 18 in the third stage, and another 2 in the fourth stage.

In our previous experiments, we conditioned all of the transformer attention matrices. However, in this particular experiment, we aim to investigate the impact of conditioning a varying number of transformer attention matrices.

As shown in Table 4.12, we observe a reduction in performance as the number of conditioned layers is decreased in the model with fewer transformers conditioned (Less Conditioned). This outcome aligns with our expectations, as conditioning has previously proven beneficial in the baseline multitask architecture. Moreover, adding more layers increases the model's capacity. Considering the relatively small number of parameters added with attention conditioning as can be

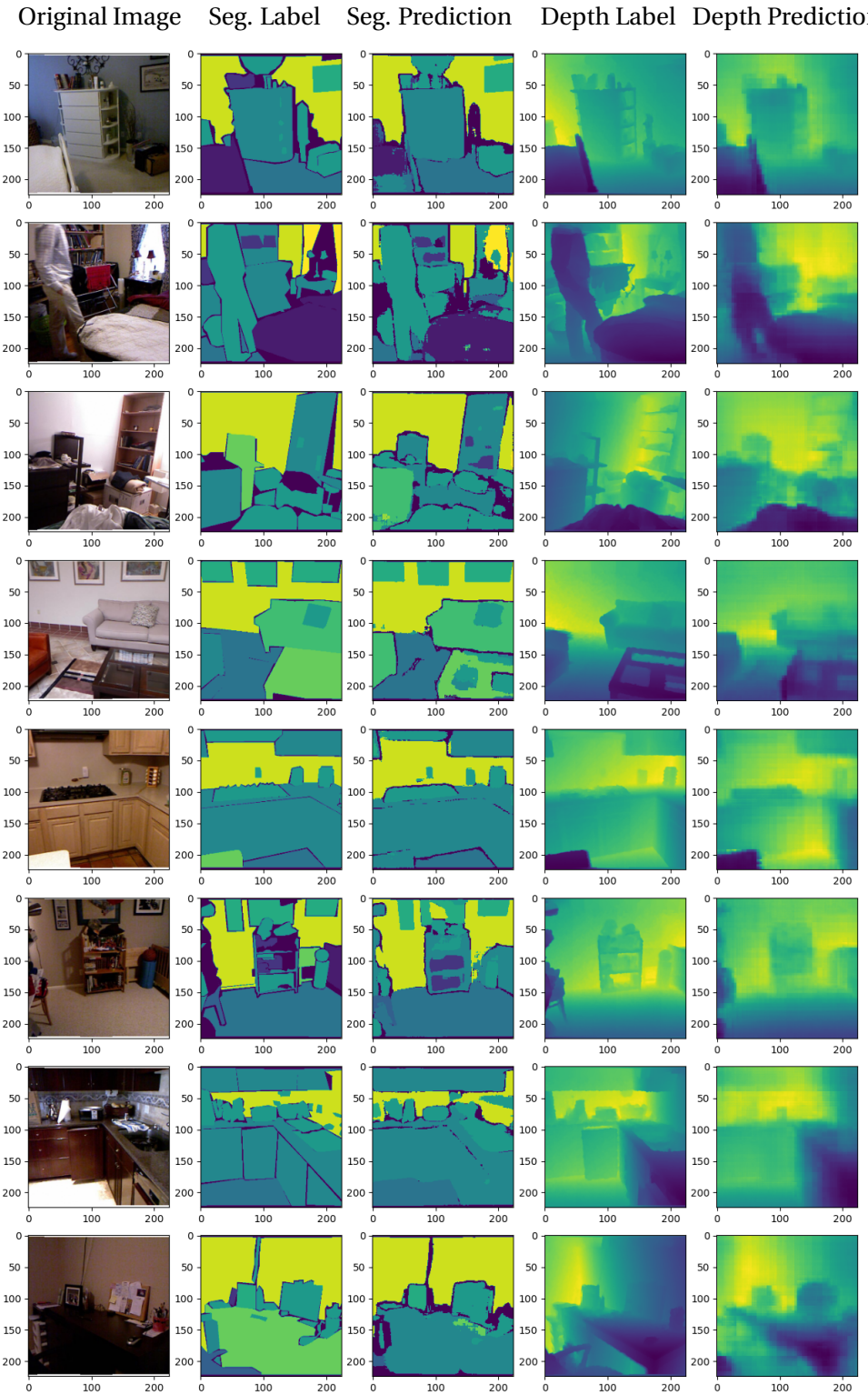

Figure 4.7: **Example predictions using best performing model** The best performing model has conditional attention on all its layers, regular layer normalization, no adapter, and 5-1 weighting.

Architectural Choice	All Conditioned	Less Conditioned
Multitask	True	True
Conditioned Modules	All	0-0-18-0
Layer Normalization	Regular	Regular
Frozen Encoder	False	False
Adapter	False	False
Task Weighting	5-1	5-1

Table 4.11: **Adaptive layer size experiment card**

	d1	Mean IOU
All Conditioned	**0.788**	**0.5032**
Less Conditioned	0.7809	0.4944

Table 4.12: **Effect of number of layers conditioned**

seen in Table 4.2, it is useful to use conditioning across all layers for optimal results.

4.5.6 Same Batch vs Different Batch

As we condition the network based on the task, we do not have the batching mechanism of the regular multitask networks, where you get the output for all tasks in one forward pass. However, in our conditioned network, separate forward passes are required for each task, conditioned by the corresponding task embeddings. This design introduces two potential options for feeding the network with different tasks.

The first approach (Different Batch) involves treating the different tasks associated with an image as distinct inputs, which are randomly assigned to different batches. Alternatively, the second approach (Same Batch) places the various tasks of an image within a single batch. To investigate the impact of these two batching mechanisms, we trained two identical networks, differing only in the chosen approach. The results of these experiments can be seen in Table 4.14.

Architectural Choice	Same Batch	Different Batch
Multitask	True	True
Conditioned Modules	All	All
Layer Normalization	Regular	Regular
Frozen Encoder	False	False
Adapter	False	False
Task Weighting	5-1	5-1

Table 4.13: **Batching mechanism experiment card**

Our findings indicate that incorporating the tasks of an image into the same batch yields slightly improved performance compared to randomly assigning them to different batches. This approach

	d1	Mean IOU
Same Batch	**0.788**	**0.5032**
Different Batch	0.7783	0.5

Table 4.14: **Effect of batching mechanism**

has similarities to the mechanism employed in traditional multitask learning, enabling model to learn to treat an image differently for different tasks. By putting the tasks within a single batch, the network can use the shared information across tasks more effectively, resulting in better performance.

4.5.7 Evaluating Task Embeddings

Interesting insights can be extracted from the task embeddings learned from the conditioned networks. We now evaluate how performance is affected by mixing the task embeddings. We see that the task embeddings for segmentation and depth estimation have 0.0313 cosine similarity, which shows us that these embeddings are differentiated. We then check how performance changes when we take a weighted combination of the embeddings. For example for a weight of 5 for segmentation and 1 for depth, the combined embedding is found by $\frac{5}{6}\cdot <$ segmentation embedding $> + \frac{1}{6}\cdot <$ depth embedding $>$. The result of the experiment can be seen in Table 4.15.

Weighting Segmentation - Depth	d1	Mean IOU
No Weighting	0.777	**0.492**
1-1 (Mean)	0.773	0.477
5-1	0.7484	0.482
10-1	0.7401	0.478
1-5	**0.780**	0.438
1-10	0.779	0.430

Table 4.15: **Effect of combined task embeddings**

We observe that the optimal weight for the segmentation task is its own embedding (No Weighting) and it is negatively affected by a combination with depth embedding. However, interestingly, the performance of the depth task is improved when combined with the segmentation embedding especially in the $1-5$ case. We see that when the majority is the segmentation, the performance of the depth estimation reduces, however when the majority is depth, it benefits from the combined use of information. This observation also coincides with the multitask learning paradigm, and we see that getting signals from the segmentation task helps the performance of the depth task.

4.5.8 Taskonomy

We also tried to see the results in the Taskonomy dataset, which is a much bigger dataset than NYUv2. However, due to technical issues, and the time it requires to train the network, we were not able to perform all the experiments but some experiments performed gave some insights into the dataset.

For example, segmentation labels are challenging in this task, because they mostly consist of background class, therefore even with a mean IOU of 53%, we were not able to see pleasing results as in Figure 4.8. Therefore, in this case, we require fairly higher metric values to have similar visual results in the NYUv2 dataset.

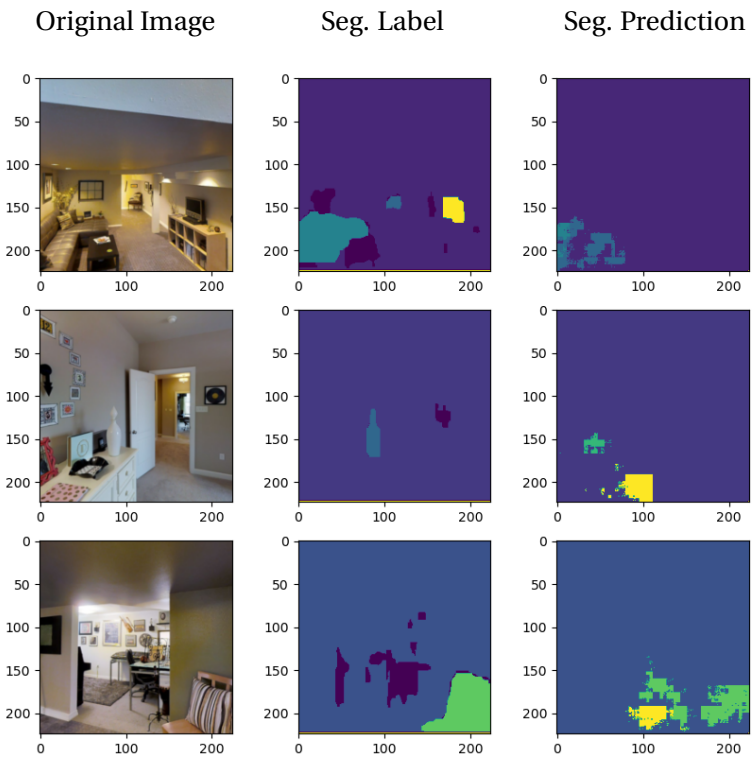

Figure 4.8: **Example segmentation predictions** The predictions are generated from the model trained with Taskonomy dataset.

When we ignore the background class when calculating the loss, although the depth model was able to learn fairly well compared to the previous setting, the mean IOU for segmentation remained very low. Some predictions can be seen in Figure 4.9. I believe this suggests that optimal task weighting may be different for this dataset in this model, which is not easy to find, as even with 4 GPUs, training takes days. Handling a big dataset like Taskonomy, therefore, requires more computing power, to achieve pleasing results.

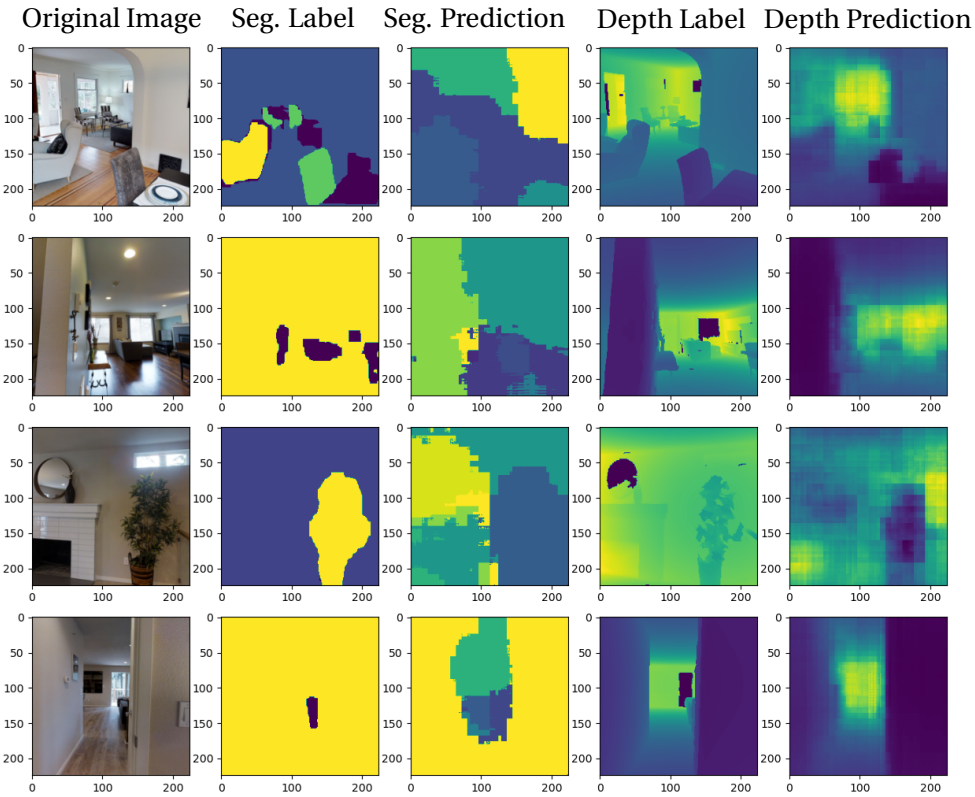

Figure 4.9: **Example segmentation and depth predictions** The predictions are generated from the model trained with Taskonomy dataset by ignoring the background class in the loss calculation.

Chapter 5

Conclusion

5.1 Failure Cases and Limitations

The initial intuition and goal were to reduce the number of parameters with the use of adapters and therefore achieve a more efficient multitask learning paradigm. As can be seen in Section 4.5.2, we did not achieve comparable results using adapters and frozen encoders. This is a major limitation as it requires the whole model to be trained which is much more time-consuming than the use of adapters.

Another downside of our project was focusing on the NYUv2 dataset for a long time, without considering its size. It consists of around 1500 training points, which can be considered very small for training transformers as they are known to be data-hungry. With the limited dataset, although we saw some improvements after using the conditional modules, they can be considered minor. Therefore the limitation of the dataset also affected our ability to judge the power of the architecture fairly. We tried to focus on the Taskonomy dataset in the final days, however, due to its size, it is an even harder dataset to train. Because of this, we could not perform many experiments.

5.2 Future Directions

As expressed in Section 5.1, the use of the NYUv2 dataset was limiting for this architecture. The initial future work could be to experiment with various datasets with more appropriate sizes. We also trained for only two tasks at a time. This number can be increased to see how more tasks affect the overall performance.

Due to the limitation that we need the whole network to be trained, instead of looking at this work as a model proposal, it can be considered as an initial work that focuses on how tasks behave

together. Task embeddings of different tasks can be investigated to see how tasks interact. Some experiments can include seeing which tasks have the most similar or most different embeddings, and if this similarity overlaps with the performance gain or loss when trained together. It can even be checked if these similarities are universal or model dependent by testing the tasks in other similar multitask architectures such as [1], or by checking works such as [26], which also works on the problem: "Which tasks should and should not be learned together in one network when employing multi-task learning?".

Another experiment can be to find a global task embedding, which can be used as optimal starting weights of FiLM layers for new tasks. It can facilitate the training of these tasks, rather than starting from scratch. One approach to finding this embedding can be to train the network with multiple tasks, but with only one task embedding.

We only explored two supervised tasks. Adding an unsupervised task such as rotation prediction, or a self-supervised task such as autoregression can also enable a more generalizable performance. Even if the goal is to get the best performance from one task, it can be explored if adding a cheap unsupervised task that does not require labeling helps with the performance.

5.3 Conclusion

In this work, we proposed using a multitask learning approach for computer vision tasks with transformer architecture. Compared to the usual multitask learning frameworks which had a shared set of parameters, we included conditioning to allow for these shared set of parameters to be further optimized for different tasks. Conditioning is added to enable such behavior and it is added to the attention, layer normalization and adapter modules.

We tested our dataset using the NYUv2 dataset on semantic segmentation and depth prediction tasks, with various experiments. We observed that the pre-trained transformer was not capable enough for a dense prediction task without further training. It lacked the pixel-level information needed for dense prediction and saw that we could not solve our problem with frozen encoders. We also observed that adding conditional modules improved the performance of both tasks.

This work can be seen as an initial work to include conditioning in multitask computer vision networks and extended to experiment with interesting questions which explore how tasks behave when trained together and separately.

Bibliography

[1] Deblina Bhattacharjee, Tong Zhang, Sabine Süsstrunk, and Mathieu Salzmann. *MulT: An End-to-End Multitask Learning Transformer*. 2022. arXiv: 2205.08303 [cs.CV].

[2] Hu Cao, Yueyue Wang, Joy Chen, Dongsheng Jiang, Xiaopeng Zhang, Qi Tian, and Manning Wang. *Swin-Unet: Unet-like Pure Transformer for Medical Image Segmentation*. 2021. arXiv: 2105.05537 [eess.IV].

[3] Hanting Chen, Yunhe Wang, Tianyu Guo, Chang Xu, Yiping Deng, Zhenhua Liu, Siwei Ma, Chunjing Xu, Chao Xu, and Wen Gao. *Pre-Trained Image Processing Transformer*. 2021. arXiv: 2012.00364 [cs.CV].

[4] Zhe Chen, Yuchen Duan, Wenhai Wang, Junjun He, Tong Lu, Jifeng Dai, and Yu Qiao. *Vision Transformer Adapter for Dense Predictions*. 2023. arXiv: 2205.08534 [cs.CV].

[5] 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*. 2021. arXiv: 2010.11929 [cs.CV].

[6] Vincent Dumoulin, Ethan Perez, Nathan Schucher, Florian Strub, Harm de Vries, Aaron Courville, and Yoshua Bengio. "Feature-wise transformations". In: *Distill* (2018). DOI: 10.23915/distill.00011.

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

[8] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. *Parameter-Efficient Transfer Learning for NLP*. 2019. arXiv: 1902.00751 [cs.LG].

[9] Alex Kendall, Yarin Gal, and Roberto Cipolla. *Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics*. 2018. arXiv: 1705.07115 [cs.CV].

[10] Iasonas Kokkinos. *UberNet: Training a 'Universal' Convolutional Neural Network for Low-, Mid-, and High-Level Vision using Diverse Datasets and Limited Memory*. 2016. arXiv: 1609.02132 [cs.CV].

[11] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. *Deeper Depth Prediction with Fully Convolutional Residual Networks*. 2016. arXiv: 1606.00373 [cs.CV].

[12] Yanghao Li, Naiyan Wang, Jianping Shi, Xiaodi Hou, and Jiaying Liu. "Adaptive batch normalization for practical domain adaptation". In: *Pattern Recognition* 80 (2018), pp. 109–117.

[13] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. 2021. arXiv: 2103.14030 [cs.CV].

[14] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. *Cross-stitch Networks for Multi-task Learning*. 2016. arXiv: 1604.03539 [cs.CV].

[15] Eslam Mohamed and Ahmed El-Sallab. *Spatio-Temporal Multi-Task Learning Transformer for Joint Moving Object Detection and Segmentation*. 2021. arXiv: 2106.11401 [cs.CV].

[16] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. *WaveNet: A Generative Model for Raw Audio*. 2016. arXiv: 1609.03499 [cs.SD].

[17] Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. *Conditional Image Generation with PixelCNN Decoders*. 2016. arXiv: 1606.05328 [cs.CV].

[18] Art B Owen. "A robust hybrid of lasso and ridge regression". In: *Contemporary Mathematics* 443.7 (2007), pp. 59–72.

[19] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. *FiLM: Visual Reasoning with a General Conditioning Layer*. 2017. arXiv: 1709.07871 [cs.CV].

[20] Jonathan Pilault, Amine Elhattami, and Christopher Pal. "Conditionally adaptive multi-task learning: Improving transfer learning in nlp using fewer parameters & less data". In: *arXiv preprint arXiv:2009.09139* (2020).

[21] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].

[22] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. *Learning multiple visual domains with residual adapters*. 2017. arXiv: 1705.08045 [cs.CV].

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[24] Sebastian Ruder. *An Overview of Multi-Task Learning in Deep Neural Networks*. 2017. arXiv: 1706.05098 [cs.LG].

[25] Hongje Seong, Junhyuk Hyun, and Euntai Kim. "Video multitask transformer network". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*. 2019.

[26] Trevor Standley, Amir R. Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. *Which Tasks Should Be Learned Together in Multi-task Learning?* 2020. arXiv: 1905.07553 [cs.CV].

[27] Gjorgji Strezoski, Nanne van Noord, and Marcel Worring. *Many Task Learning with Task Routing*. 2019. arXiv: 1903.12117 [cs.CV].

[28] Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. *VL-Adapter: Parameter-Efficient Transfer Learning for Vision-and-Language Tasks*. 2022. arXiv: 2112.06825 [cs.CV].

[29] Ian Tenney, Dipanjan Das, and Ellie Pavlick. *BERT Rediscovers the Classical NLP Pipeline*. 2019. arXiv: 1905.05950 [cs.CL].

[30] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. *Training data-efficient image transformers distillation through attention*. 2021. arXiv: 2012.12877 [cs.CV].

[31] Simon Vandenhende, Stamatios Georgoulis, Bert De Brabandere, and Luc Van Gool. *Branched Multi-Task Networks: Deciding What Layers To Share*. 2020. arXiv: 1904.02920 [cs.CV].

[32] Simon Vandenhende, Stamatios Georgoulis, Wouter Van Gansbeke, Marc Proesmans, Dengxin Dai, and Luc Van Gool. "Multi-Task Learning for Dense Prediction Tasks: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), pp. 1–1. DOI: 10.1109/tpami.2021.3054719. URL: https://doi.org/10.1109%2Ftpami.2021.3054719.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[34] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. *Unified Perceptual Parsing for Scene Understanding*. 2018. arXiv: 1807.10221 [cs.CV].

[35] Amir R Zamir, Alexander Sax, William B Shen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. "Taskonomy: Disentangling Task Transfer Learning". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2018.

[36] Laurent Zwald and Sophie Lambert-Lacroix. *The BerHu penalty and the grouped effect*. 2012. arXiv: 1207.6868 [math.ST].