# Multi-Task Learning Architectures and Applications

*Andy Yan*
*Xin Wang*
*Yanlai Yang*
*Roy Fox*
*Xiaolong Wang*
*Joseph Gonzalez*

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-54
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-54.html

May 22, 2020

Multi-Task Learning Architectures and Applications

by

Andy Yan

A project report submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

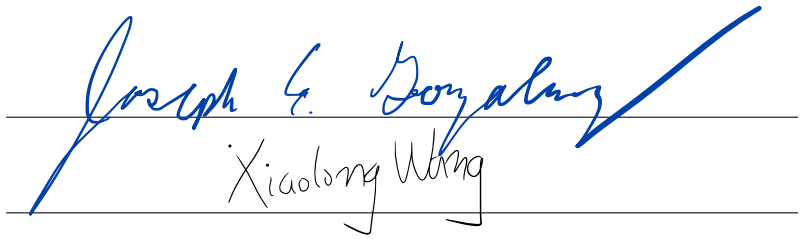
University of California, Berkeley

Committee in charge:

Professor Joseph Gonzalez,
Doctor Xiaolong Wang,

Spring 2020

The project report of Andy Yan, titled Multi-Task Learning Architectures and Applications, is approved:



Date 05/22/2020

Xiaolong Wang

Date 05/22/2020

University of California, Berkeley

Multi-Task Learning Architectures and Applications

Abstract

Multi-Task Learning Architectures and Applications

by

Andy Yan

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Joseph Gonzalez,

Doctor Xiaolong Wang,

Multi-task learning fundamentally involves utilizing multiple tasks to assist with generalization. In this report, we first investigate the motivation for research in multi-task learning, showing that when training multiple tasks together on the same neural network, performance may benefit if the tasks are related and suffer if the tasks are not. We then study multi-task architectures and evaluate in depth a mixture-of-experts model. We show that in experiments on the CIFAR-100 MTL domain, multi-clustering outperforms prior architectures in accuracy and computation time. Next, we apply multi-task learning to the object detection task using the BDD100K dataset. We explain our method of training the object detection task with the self-supervised tasks of angle and distance prediction and colorization, demonstrating performance benefits. Lastly, we demonstrate our work in few-shot learning, where we proposed a method to train a task to which we have little to no data by exploiting the task's compositionality.

To my mother, father, and sister

# Contents

# List of Figures

# List of Tables

## Acknowledgments

I would like to thank Prof. Joseph Gonzalez for the experiences I had learning and researching in his lab for the past two years. I would also like to thank Prof. Roy Fox, Dr. Xin Wang, and Dr. Xiaolong Wang for their guidance. Lastly, I want to thank my peer researchers including Yanlai Yang and my friends who have provided me with support throughout the process of obtaining my master's degree at UC Berkeley.

# Chapter 1

# Introduction

## 1.1 Background to Multi-Task Learning

Multi-task learning fundamentally involves utilizing multiple tasks to assist with generalization. In the biological context, humans utilize multi-task learning in day to day tasks. For instance, when learning to play a song on different instruments such as a guitar and a piano, aspects of the song such as the melody can be generalized between the instruments. Therefore, playing the song on the guitar can assist with playing the song on the piano. The same intuition may be applied to deep learning. Given two tasks that are similar to each other, co-training the two tasks on the same neural network can help the network to generalize learning of certain features.

Given a single task, generally reasonable performance can be obtained by optimizing for that task by collecting data and training a network to optimize for it. However, limitations may arise as data collection may be costly. In addition, collecting enough data for a certain task to cover edge cases may also be difficult if those edge cases are rare. Another problem that arises with training using a single task involves training time and resources, as training complicated tasks may require significant and expensive computational resources.

Multi-task learning can assist in these cases. In the context of generalization, although it may be difficult to collect diverse data for a single task, utilizing multiple tasks may help alleviate some of these issues by providing more data in general and more diversity in data. For instance, in the case of image digit classification, consider two datasets of digits from different domains such as MNIST and SVHN, where the former contains black and white digits while SVHN contains digits that appear on street house numbers. Digits found in MNIST are drawn in a different font style than SVHN, so multi-task learning could potentially aid in generalizing across different digit styles, resulting in a more robust classifier.

The digit classification domain is a relatively easy domain. However when tasks become more difficult, multi-task learning can help with minimizing computational resources. An example for this is ImageNet pretraining, which is a common strategy in vision where a neural network is initialized using the parameters obtained after training on ImageNet [1].

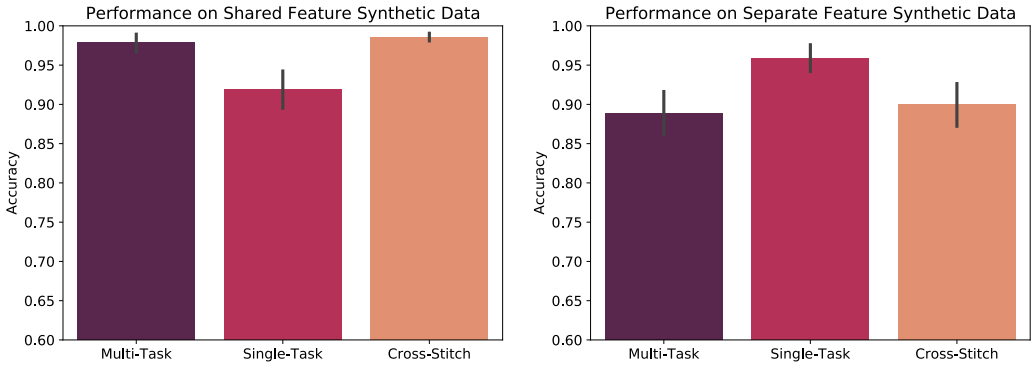



Figure 1.1: Testing accuracy of three architectures after training on two different synthetically generated datasets. The left depicts the dataset generated using shared features and the right depicts the dataset generated using separate features.

The network is then fine-tuned, requiring a smaller amount of training time. Thus, the use of multiple tasks can reduce the required amount of computational resources.

In this report, we explore architectures for multi-task learning and then some applications of multi-task learning including co-training for object detection and few-shot learning for image classification.

We first review previous architectures and then introduce a novel architecture that outperforms previous architectures. We present exploration of our novel architecture on the CIFAR-100 multi-task learning (MTL) dataset and evaluate performance benefits that our architecture presents, including generalization and compute. We then investigate the use of multi-task learning on a self-driving dataset to improve the object detection task using self-supervised tasks. These include tasks such as angle distance prediction and colorization. Lastly we explore the application to few-shot learning and discuss an approach of using multiple tasks to train a generator to generates synthetic data that assists with training of new tasks.

Prior to discussing these explorations, we first show a toy example that depicts the potential benefits of multi-task learning. Intuitively, using multiple tasks may be beneficial when the tasks are related but also harmful when tasks are not. Thus, we first explore artificially creating tasks that are related and unrelated.

We randomly generate a dataset, where each data point is a binary string of length 20 with a label determined by the task labeling functions below.

In the first case, we utilize two tasks that share features, task 0 (t0) and task 1 (t1).

```
def t0(x):
    return x[0] == 1 or sum(x[1..10]) % 2 == 0

def t1(x):
    return x[0] == 0 or sum(x[1..10]) % 2 == 0
```

These two tasks use the same features, as they are a function of the first 10 bits of 20. However, the task labels differ in that although they use the same features, task 0 outputs 1 when x[0] is 1 while task 1 outputs 1 when x[0] is 0.

We contrast this to a different setting, where tasks use separate features. Task 0 uses the first 10 bits while task 1 uses the last 10 bits.

```
def t0(x):
    return x[0] == 1 or sum(x[1..10]) % 2 == 0

def t1(x):
    return x[10] == 0 or sum(x[11..20]) % 2 == 0
```

The results of using different architectural choices to learn these tasks are in Figure 1.1, where we use three different architectures, multi-task, single-task, and cross-stitch.

The multi-task architecture involves the sharing of a single backbone of parameters, shown in Figure 2.1, whereas the single-task architecture uses two separate backbones. Cross-stitch involves a simple architectural adjustment that connects together two separate backbones, allowing tasks to learn whether they should use the same backbone or not.

We observe that when tasks use the same features, using the multi-task and cross stitch network outperform single-task whereas when tasks use different features, using single-task outperforms both multi-task and cross-stitch. The figure depicts testing accuracy, showing differences of around 10% when using the appropriate architecture.

As a result, we observe that the multi-task architecture can help when features are shared, but when features are not shared, the multi-task architecture could hurt performance. This motivates exploration in studying when and how to train tasks together in order to maximize performance.

# Chapter 2

# Related Work

## 2.1 Multi-Task Architectures

Extensive research has been done in multi-task learning, seeking to improve task performance [17, 22] and reduce the requirement of computational [27] and memory resources [15]. As a result, many creative models and algorithms have been proposed, including ones that cluster tasks in a top-down fashion [13], learn relationships between tasks using matrix priors [11], and more [26].

Two basic models involve fully sharing the same backbone between tasks and fully separating tasks with different backbones. These are depicted in Figure 2.1, where the left side depicts the fully shared architecture and the right side depicts the fully separated architecture. In the fully shared architecture, each task uses the same backbone but different task-specific layers afterwards, while in the fully separated architecture, each task uses different backbones and task specific parameters. We note here that as the number of tasks increases in the fully shared architecture, the number of backbones scales linearly, posing potential memory issues.

We proposed a model that uses the mixture-of-experts (MoE) principle for multi-task learning. Two related models are the cross-stitch network [17], where outputs from all experts are mixed together by a task-specific linear combination, and the routing network [25], where one expert per gating layer is selected, depending on the input and the task. Routing and cross-stitch networks can be viewed as two extremes of a spectrum, with the former making gating decisions that are hard (yes / no) and sparse (exactly one expert), and the latter performing soft and dense "gating". Our method enjoys the best of both approaches, by learning how many experts to select for each task, as well as task-specific linear combinations that mix the selected experts. Our exploration in architecture only studies networks with a single gating layer, but our method can be straightforwardly extended to multiple gating layers.

A similar approach is taken in [27] in the single-task setting, where the gating layer selects a predetermined number of experts. In contrast, we learn a task-specific number of

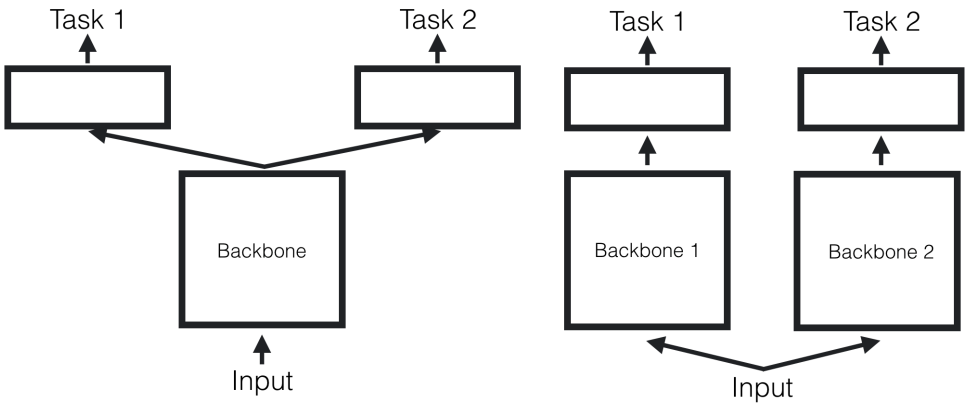



Figure 2.1: Architectures for fully sharing and fully separating. The input passes through the backbone, which is depicted by the first two layers in this figure. Fully sharing uses the same backbone whereas fully separating uses two backbones.

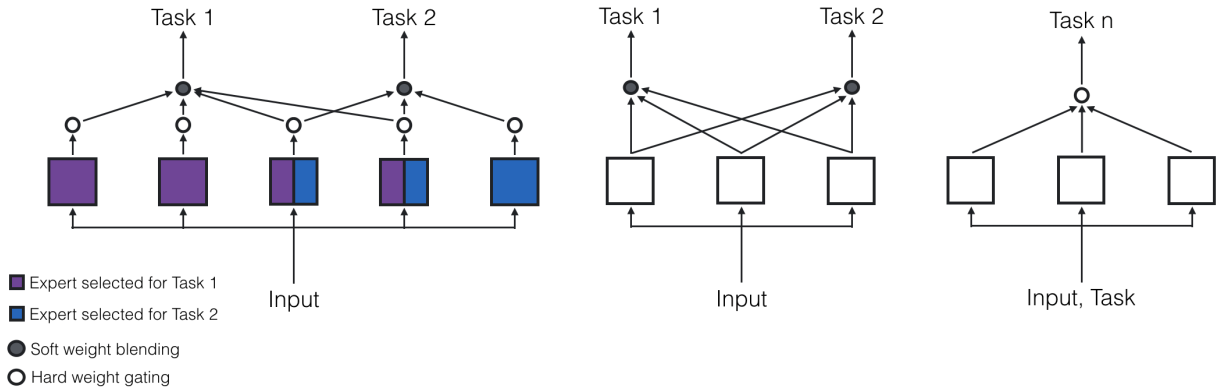



Figure 2.2: Three architectures for multitask learning. The left is our proposed model, the middle is the cross-stitch model [17], and the right is routing networks [25]. Cross stitch involves using soft blending of backbone outputs. Routing networks takes the task as input and selects selects which backbone to use for that task through hard gating. We note that these architectures each can be viewed as multi-task units and stacked together.

experts in the multi-task setting, which requires a different mechanism for expert selection (Section 3.1). During training time, our gating is a variant of dropout [28] in which entire experts are dropped out rather than single neurons, and the dropout probabilities are learned and task-specific rather than fixed.

Another MoE model for the single-task setting is DeepMoE [29], which uses ReLU activation to encourage sparsity in input-dependent gating weights. In contrast, we make discrete gating choices using a probabilistic model that is conditioned only on the task. After training, we fix the set of experts that contribute to each task, facilitating efficient model deployment, and improving the model's interpretability (Section 3.5).

## 2.2 Applications to Vision

As we investigate applications to the object detection task of self-driving, we use prior object detection architectures such as Faster R-CNN [24].

We use the BDD100K dataset [35], a self driving dataset. This self driving dataset consists of videos taken by drivers from multiple cities on their phone cameras. Besides the video data, the phone camera in addition contains data such as gps data, which we use to create self-supervised tasks. This data can be considered self-supervised, as it comes for free because phones can automatically record this data while recording video.

Multi-task learning has been applied to the task of object detection both through pre-training and co-training. As for pre-training, networks are commonly pretrained using the ImageNet [1] classification task. Other tasks are summarized in [8], which groups these tasks into four categories, generation-based, context-based, free semantic label-based, and cross modal-based. We investigate a task from the generation-based category, colorization [10]. We note that in contrast to our studies, these studies use these tasks for pre-training whereas we train these tasks simultaneously with the detection task. As for co-training, tasks such as Mask R-CNN [5] have shown success in improving generalization. However, we also note that Mask R-CNN uses supervised data, whereas in our studies we use self-supervised data.

## 2.3 Applications to Few Shot Learning

We additionally explore multi-task learning applications to few-shot learning, where we attempt to learn a new task with little to no data samples using the tasks related to the new task. We investigated a generative approach in [31], where we learn to generate features to assist with training a classifier on a new task.

Constructing task-conditional feature representations has been largely adopted in the recent work [16, 30, 18, 23]. The investigated tasks are tasks to classify some entity that composes of a variety of properties. One such composition is object and attribute. Wang *et al.* [30] and Purushwalkam *et al.* [23] creates task-aware feature representations by re-

configuring the network to be conditioned on attribute-object compositions. Nagarajan and Grauman [18] proposes to use attributes as operators to modify the object features.

In contrast to the existing works, we take a generative perspective, focusing on feature synthesis for compositional learning. We hypothesize that if a generator is capable of synthesizing feature distributions of the seen compositions (compositions to which we have data), it may be possible to transfer to novel compositions by producing synthetic features that are informative enough to train a classifier without needing real images.

# Chapter 3

# Multi-Task Learning Architectures

## 3.1 Task Multi-Clustering

A task is a tuple $\langle \mathcal{X}, \mathcal{Y}, P, \mathcal{L} \rangle$, with $\mathcal{X}$ and $\mathcal{Y}$, respectively, input and output domains, $P$ a distribution over $\mathcal{X} \times \mathcal{Y}$, and $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ a loss function. We say that a model $m : \mathcal{X} \to \mathcal{Y}$ achieves good performance if it has low expected loss $\mathbb{E}_{(x,y)\sim P}[\mathcal{L}(m(x), y)]$. In multi-task learning (MTL), we assume some distribution $Q$ over a set of tasks $\mathcal{T}$, and a dataset of tuples $(t, x, y)$, such that $t \sim Q$ and $(x, y) \,|\, t \sim P_t$. In our experiments, $Q$ is the uniform distribution over $N$ tasks.

We propose a mixture-of-experts (MoE) model for MTL in which the tasks in $\mathcal{T}$ are clustered into $M$ experts. The binary gating variables $b_i(t) \in \{0, 1\}$ are functions of the task representation $t$ indicating whether task $t$ is clustered into expert $i$. Intuitively, we would like a subset of the tasks $\mathcal{T}_i = \{t | b_i(t)=1\}$ to be clustered into expert $i$ if these tasks, and only these tasks, share useful features $f_i(x)$ of the input $x$. The expert should learn to extract such features. A visual representation is presented in Figure 2.2. Here, the input is fed into each expert / backbone. Task 1 selects the first 4 experts, while task 2 selects the last 3. After the features are passed through the experts, only the outputs of the first 4 experts are combined through soft gating for task 1 and only the outputs of the last 3 experts are combined through soft gating for task 2.

Given the expert featurizations $f_i(x)$ and the multi-clustering $b_i(t)$, we compute for each task $t$ a task-specific representation $z(t, x)$ as a mixture of the experts into which it is clustered

$$z(t, x) = \sum_i b_i(t) w_i(t) f_i(x), \tag{3.1}$$

where $w_i(t)$ are the mixture weights. We learn a *task head* $g$ mapping from the latent representation $z(t, x)$ to the output

$$m(t, x) = g(t, z(t, x)).$$

Our training objective is to minimize the expected loss between the predicted output and the true output

$$\min_{b,w,f,g} \mathbb{E}_{t\sim Q}\, \mathbb{E}_{(x,y)\sim P_t}[\mathcal{L}_t(m(t,x),y)]. \tag{3.2}$$

We note that, during both training and evaluation time, only the experts of the mixture (3.1) for which $b_i(t) = 1$ need to be called. Sparsity of the multi-clustering can therefore yield significant computational efficiency.

To perform gradient-based optimization of the objective (3.2), we can choose differentiable parametrizations of all $w_i$'s, $f_i$'s, and $g$ as neural networks. Unfortunately, the multi-clustering indicators $b_i$ are inherently discrete. To relax this modeling assumption, we introduce the probability $p_i(t)$ that expert $i$ will be used for task $t$, and have $b_i(t) \sim \text{Bernoulli}(p_i(t))$. This makes $z(t,x;b)$ and $m(t,x;b)$ random variables, requiring special care in the optimization.

Because the discrete gating variables are not differentiable, in order to optimize the gating distribution, we can either relax it into a reparametrizable distribution using the Gumbel–softmax trick [7], or use the score-function trick to sample from the same gating distribution that we optimize. Sampling from a relaxed gating distribution generates soft gating values $b_i(t) \in (0,1)$, and does not enjoy the computational benefits of sparsity during training time. We therefore choose the score-function trick

$$\nabla\, \mathbb{E}_{b\sim p(t)}[\mathcal{L}_t(m(t,x;b),y)] = \mathbb{E}_{b\sim p(t)}\left[\nabla\mathcal{L}_t + \mathcal{L}_t \sum_i \nabla \log p_i(b_i;t)\right], \tag{3.3}$$

where $p_i(b_i;t)$ is $p_i(t)$ if $b_i$=1, else $1-p_i(t)$. To estimate the gradient (3.3) in each optimizer step, we sample a single gating combination $b$ for each element of the mini-batch.

We define the density of each task as the expected number of experts used in its computation, $k(t) = \sum_i p_i(t)$, and refer to its complement $M - k(t)$ as the sparsity. We desire sparser representations for faster training and evaluation, as long as task performance does not degrade much.

To our optimization objective, we add the expected task density, in our case the mean $\frac{1}{N}\sum_{t=1}^{N} k(t)$, as a regularization term, namely $L_1$ regularization on $p$. The coefficient of this term, $\lambda$, can be viewed as the relative marginal cost of computing one expert and incurring a unit of loss. By tuning $\lambda$, we can control the trade-off between computation and loss.

During evaluation time, given an input $x$ for task $t$, we wish to keep the same task density $k(t)$ as in training time. Moreover, efficient model deployment and computation may require a fixed, deterministic multi-clustering $b$. We therefore round $k(t)$ to the nearest integer, and pick the set of $k(t)$ most likely experts

$$\underset{b:\ \sum_i b_i(t)=k(t)}{\arg\max} \sum_i b_i(t)p_i(t). \tag{3.4}$$

Other approaches to choosing the fixed multi-clustering $b$ from the learned distribution $p$ could be rounding the probabilities to 1 or 0, or sampling from them. However, rounding

would induce a different expert density at evaluation time than in training time, which may create a covariate shift for the downstream network, and sampling from the gating distribution may not select the most likely experts, thus degrading performance.

As a result of our gating discretization technique (3.4), the model computation time is $O(k)$, where regularization induces selection of only $k$ experts. This is generally much more efficient than mixing all $M$ experts in $O(M)$ computation time. As the diversity of the set of tasks $\mathcal{T}$ increases, we must also increase the total number of experts $M$, but not the task density $k$, contributing to the advantage of the multi-clustering method over naive MoE.

## 3.2 Experiments

We evaluate our method on the CIFAR-100 MTL dataset, in which each task is to classify an image into one of 5 classes, with cross-entropy loss between the true class and the predicted class distribution. There are $N = 20$ tasks, corresponding to 20 non-overlapping super-classes (fish, flowers, etc.). Each image in the dataset is involved in exactly one task and has exactly one true label for that task.

The deep network architecture of each of the experts $f_i(x)$ is four sequential convolutional layers, each with 32 filters, ReLU activation, and $2 \times 2$ max pooling. Each task head $g(t, z)$ is a fully connected layer that takes the mixture of experts $z(t, x)$ and outputs classification logits. Two $N \times M$ matrices represent $w_i(t)$ and the logits of $p_i(t)$, with each element of $w$ initialized by a standard Gaussian, and $p$ initialized uniformly as 0 logits. We optimize the mean task loss (3.2) using Adam with an initial learning rate of $10^{-3}$.

We present three experiments: (1) a comparison between different gating techniques in terms of accuracy, (2) a comparison between different ways to select experts during evaluation time; and (3) a qualitative evaluation of the multi-clustering similarity between tasks.

## 3.3 Comparison of Gating Techniques

To compare with cross-stitch networks [17], we generalize their method to have more than 2 tasks and 2 experts by representing the mixture weights as an $N \times M$ matrix. This is equivalent to setting $b \equiv 1$ in our model, and compares sparse gating to dense, *ungated* MoE.

The results are summarized in Figure 3.1. With sufficient data, multi-clustering outperforms ungated MoE with the same number of parameters. While MoE is largely insensitive to the number of experts in a wide range (20–100), multi-clustering benefits from sparser gating as the amount of data decreases, presumably due to a regularization effect. Our results consistently outperform those reported in rosenbaum2017routing: routing networks achieve 60% accuracy, whereas our model with $\lambda = 0.1$, using the same architecture, achieves 62.7% accuracy.

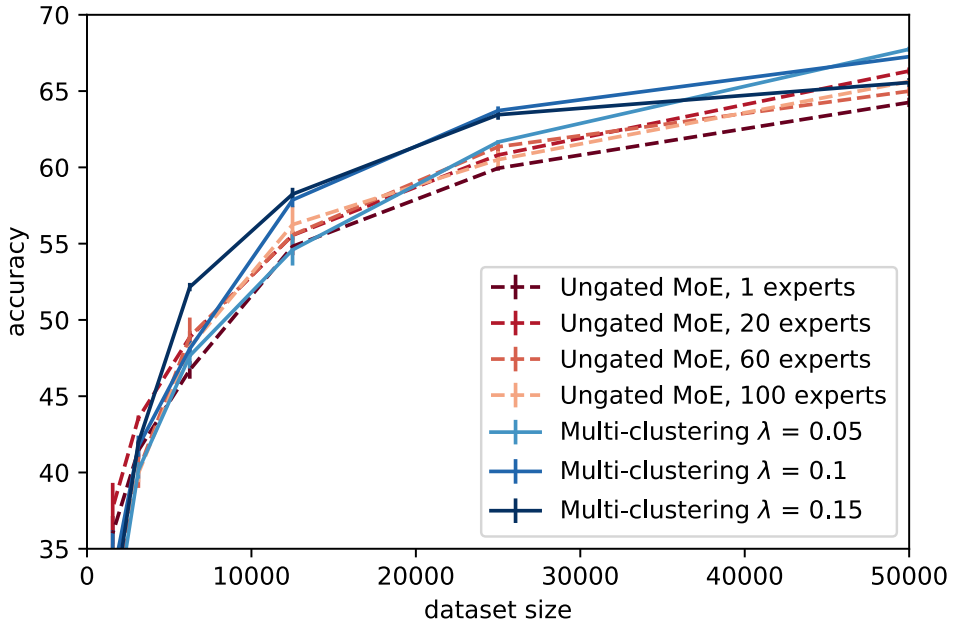



Figure 3.1: Accuracy of models trained with different amounts of data. Task multi-clustering with sufficiently large sparsity coefficient $\lambda$ outperforms ungated mixture-of-experts, when sufficient data is available. Some 1-std error bars are too small to be visible.

## 3.4 Expert Selection during Evaluation Time

We turn to the question of how task-specific subsets of experts should be selected during evaluation time, based on the learned distribution $p$. We wish to fix the multi-clustering after training to deploy a fixed model for each task with only the selected experts. This has the potential to make the model more computationally efficient and interpretable.

Prior works predefined the number of experts per task as a hyper-parameter $k$. In shazeer2017outrageously, the $k$ experts with the largest (noisy) weights are selected. In contrast, we choose the task-specific $k(t)$ to be the expected number of experts selected during training time for task $t$. We speculate that this choice of $k(t)$ reduces the covariate shift in the distribution of $z(t, x)$ experienced by the task head $g(t, z)$, thus improving its performance. Given $k(t)$, we select the maximum-posterior subset of experts for the task, which is the $k(t)$ most likely experts.

We compare this method of selecting the final multi-clustering to choosing $k$ to be mean expected number of experts over all tasks. The results, summarized in Figure 3.2, suggest that choosing task-specific $k(t)$ outperforms choosing the mean $k$. We speculate that the performance degradation is more pronounced in sparser multi-clustering because this increases the risk of having too few features for tasks that need more features than the average.

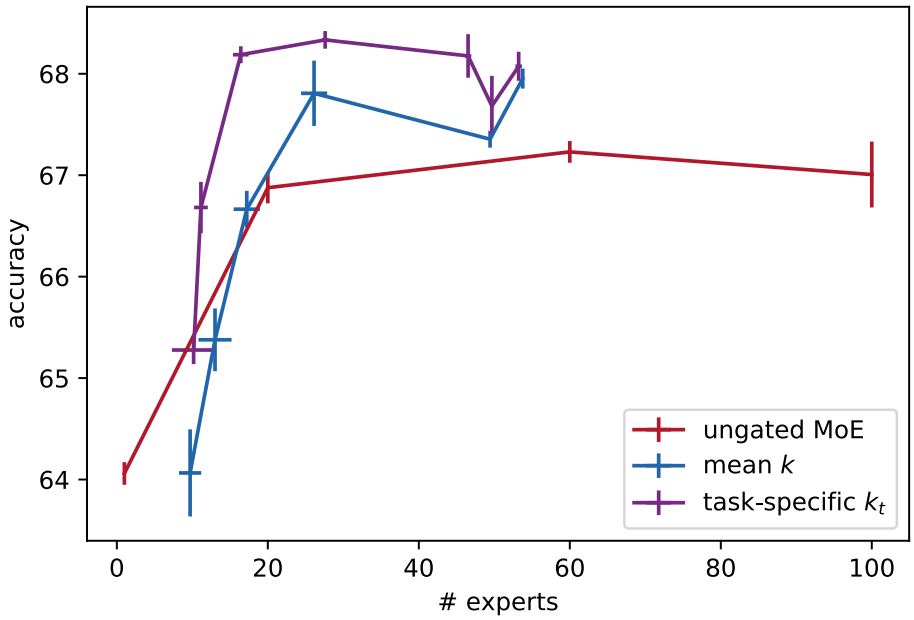



Figure 3.2: Performance over number of experts used during evaluation time on the full dataset of 50K images. Using task-specific selection of $k(t)$, the number of experts per task, outperforms using mean $k$ over all tasks and ungated mixture of all experts.

Figure 3.2 also compares the performance of ungated MoE as a function of the total number of experts $M$. Performance is largely insensitive to the number of experts, if there are sufficiently many, but forcing all $M$ experts to be mixed does degrade performance. This suggests that tasks do benefit from our discrete multi-clustering formulation.

## 3.5 Task Similarity

Using hard gating enables interpretable task relatedness involving how similarly tasks share experts. In our evaluation of this property, after adding sparsity regularization with coefficient $\lambda = 0.1$, we find that some tasks have a strong preference for some experts, as depicted in Figure 3.3. The left maps show the probabilities of the $M$ experts being used for the $N$ tasks. The right maps display the normalized correlation in gating probabilities between pairs of tasks (rows in the left maps). With the larger dataset (bottom), there is a clear correlation in gating probabilities between tasks, such as tasks 3 and 6, which are "food containers" and "household furniture", respectively. With the smaller dataset (top), such correlations between tasks are unclear.

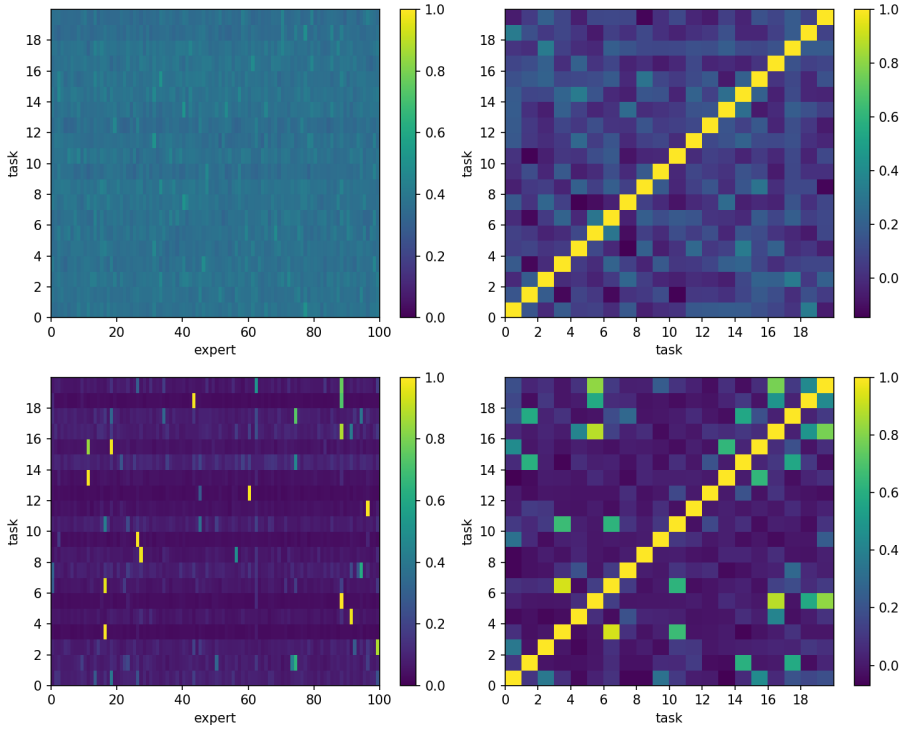



Figure 3.3: Gating representations for multi-clustering with sparsity regularization coefficient $\lambda = 0.1$, trained on 12.5K (top) and 50K (bottom) data points. Left: probabilities of selecting each expert for each task. Right: normalized correlation between the tasks.

## 3.6 Conclusion

In multi-task domains, tasks are often partially related, in the sense that they benefit from sharing some but not all of their latent representation with other tasks. We presented a method for discovering a multi-clustering of tasks into feature-extracting experts, such that each task uses a mixture of the experts it selected. Through these experts, the task shares each portion of the parameters in its model with a different cluster of partially related tasks.

The discovered structure of the multi-clustering facilitates more efficient learning, in terms of both task performance and computational efficiency. The model's sparsity allows evaluating only a subset of the experts per task, both during training and evaluation time. By fixing the multi-clustering after training, we enable deployment of smaller models per task, which only involve the most useful experts. Finally, with sufficient data, the model's performance is robust to significantly increasing the sparsity through regularization.

The single multi-clustering gating layer presented in this work can be extended to models with multiple such layers. This extension may require techniques that reduce the considerable variance of the score-function gradient estimator (3.3). Our method should further be evaluated on a variety of domains and architectures.

# Chapter 4

# Applications to Vision

## 4.1 Introduction

In this section, we survey some potential methods of improving the object detection task on the BDD100K dataset. In addition, we provide some details in the ways we configured self-supervised tasks to train with the object detection task.

To begin with, we discuss the task of object detection and the architecture we use as well as the BDD100K dataset.

In object detection our goal is to be able to, given an image, predict the bounding boxes and classifications of every object in the scene. This process is done by using a region proposal network to obtain the region proposals, and then using these proposals to predict bounding boxes and classifications.

In this process, a backbone extracts features that are used in this pipeline. We use

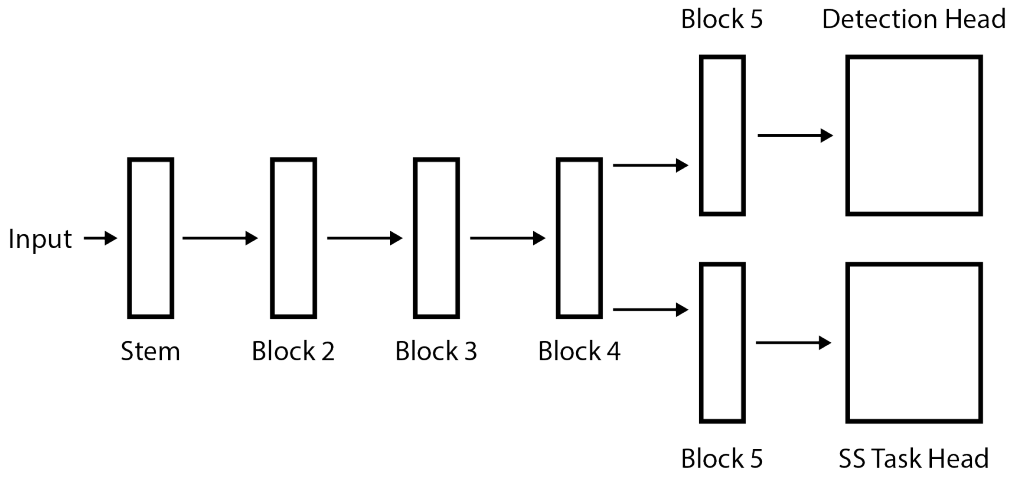



Figure 4.1: Architecture used in the co-training of object detection and self supervised tasks.

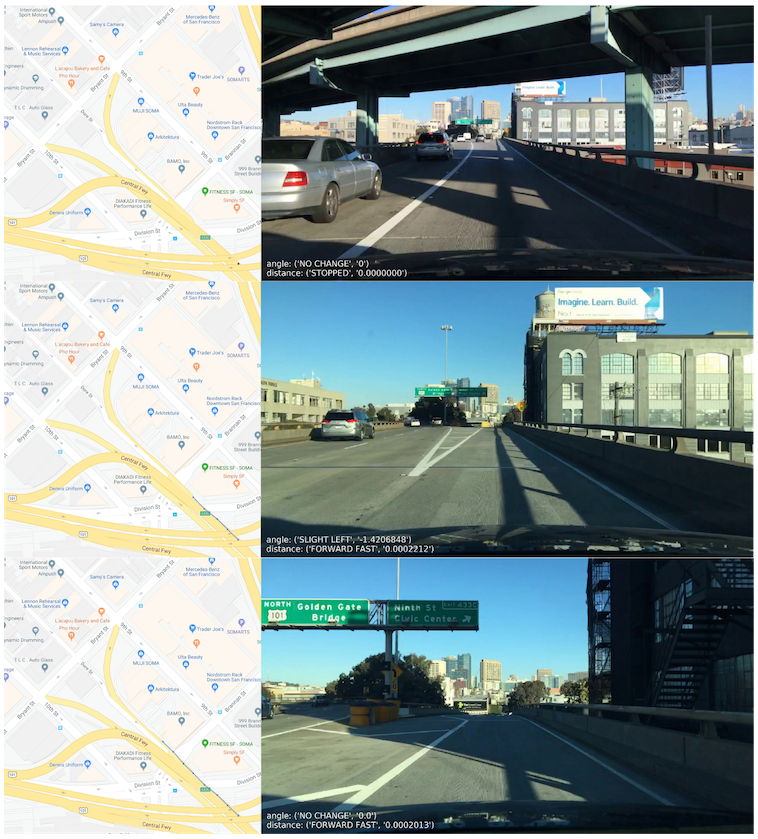



Figure 4.2: Sample images from a driving video in the BDD100K dataset, to which we have GPS labels visualized on the left side. The actual angle and distance labels are depicted in white text on the bottom left of the video images.

ResNet-50 as the backbone, but other architectures can be substituted similarly. We share the features of the backbone between tasks, using the intuition that if the features extracted by the backbone are useful for self-supervised tasks, they may be useful for the object detection task. Specifically, we choose to share the stem and first 3 blocks of the ResNet-50 architecture. Beyond the first 3 blocks of the ResNet-50 architecture, the central object detection task replicates the last block of the ResNet-50 architecture and thus shares different task-specific parameters. This is depicted visually in Figure 4.1.

We use the BDD100K dataset, where some samples are depicted in Figure 4.2. The BDD100K dataset contains driving videos from many diverse situations such as different cities, different times of day, and different weather conditions. These driving videos are labeled with bounding boxes and classes for the object detection task. The videos themselves provide enough information for our exploration on the colorization task. Because these videos are recorded on a phone, the phone also provides self-supervised data. This includes course,

which is a global indication of angle of travel, and longitude and latitude. These are the fields we use for the angle distance prediction task.

## 4.2 Task Exploration

We explore two different tasks, including angle distance prediction and colorization.

### Angle Distance Prediction

The angle or distance prediction task involves predicting the change in trajectory that a car has undergone given a start and end point from a driving video.

The intuition is that in order to predict the angle change and distance change between frames in a video, one must observe how objects in the frames have moved. If a still object has increased in size between the first frame and the second frame, it suggests that our car has moved forward. Thus, a network that can extract features important for this classification task may also extract features relevant for the object detection task.

First, we discuss how we extract angle and distance labels from the provided data. Given an image sampled from a video in the dataset, we sample another image nearby. With the pair of images, we should have information, knowing the direction of time, to predict the change in angle and distance between those two images. Distance is computed using the gps signal, where given the latitude and longitude of two points, we can compute the euclidean distance between them. Angle is computed using the course data. Course is a number between 0 and 360, indicating global direction. However, given an image pair, we desire the local change in course. We compute the local change by using the start and end course. However, given two course numbers, such as 30 and 180, it is ambiguous whether the car has turned 150 degrees clockwise or -210 degrees counter-clockwise. Given the small difference in time between the two frames, however, we assume that the car has turned in the less extreme manner. Thus in the case given, we decide that the car has turned 150 degrees clockwise. As a result, angles are bounded between -180 and 180.

Examples from each label are shown in Figure 4.3, though we note that instead of using an image pair, we use 4 images between the two sampled time-steps.

In terms of the implementation, we feed the images from the frames in the video through the shared backbone, through the self-supervised block 5, and then concatenate the features to feed through a final fully connected layer in order to output logits for classification. The self-supervised loss, computed using cross entropy loss, is scaled by 0.1.

We draw images from the videos with a delta of 10 at 5 fps, meaning that if we sample frame n from a video, we will also sample frame n+10 in order to perform classification. The difference in time between these two frames is also 2 seconds.

However, we note that an image pair may not provide information. For instance, if the car has turned too much, the objects in the scene between the two frames may completely change. Thus we include more than just 2 images between the start and end frame. For

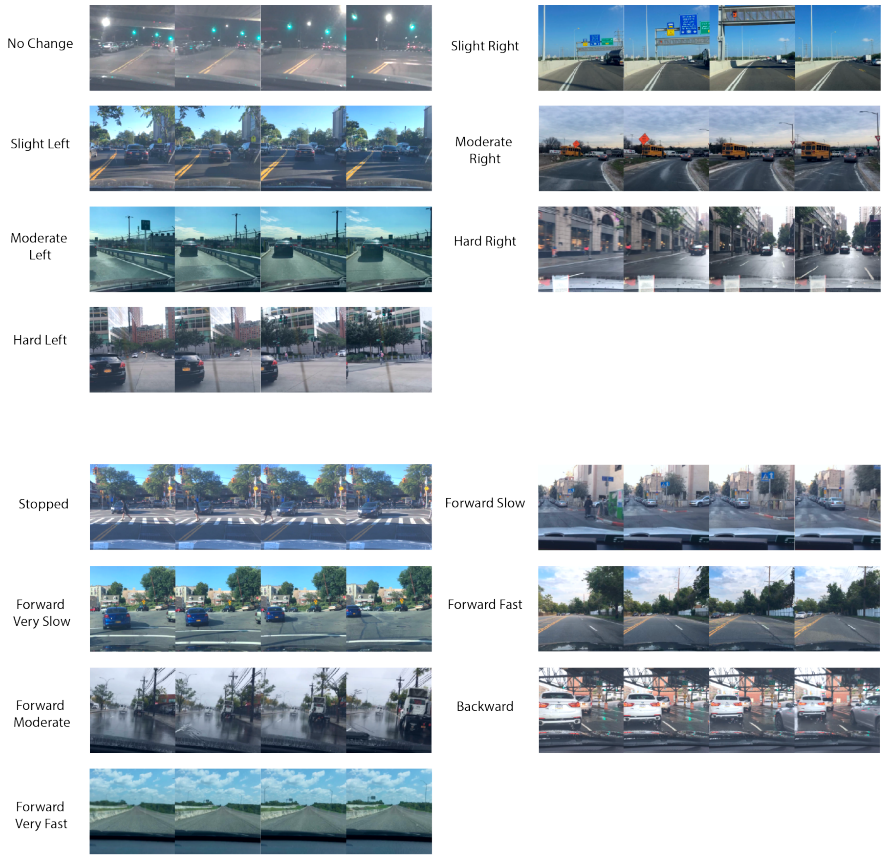



Figure 4.3: Examples of data points for each angle and distance change category.

example, between image n and n+10 in a sequence, we use images n, n+3, n+6, n+9 instead of n and n+10.

Another consideration we took into account is that the batch size for object detection is 2 per GPU due to memory constraints. However, a batch size of 2 may not be large enough for a stable signal for training angle and distance classification. Thus, we use a batch size of 16 per GPU due to time and memory constraints, which we observe to be enough to reduce training loss to near 0.

However, we only observed minor performance gain co-training with the angle and distance tasks, as shown in Table 4.1. The baseline number is obtained by running the object detection pipeline with the angle and distance loss scales to 0. We show the AP of the object detection task after training with just the angle task, just the distance task, and both angle and distance tasks. In the future, we plan to investigate modifying the training procedure further in order to amplify the potential gains.

Table 4.1: Effect of co-training with angle and distance tasks on object detection AP.

| Configuration | AP |
|---|---|
| Baseline | 30.72 |
| Angle | 30.796 |
| Distance | 31.025 |
| Angle + Distance | 30.88 |

## Colorization

Colorization in previous works have attempted to predict the HSL values of an image as a auxillary task. [10] in particular uses hypercolumns, each of which consists of the features corresponding to a certain pixel in the original image stacked together. This means that the output features of each convolutional layer are upsampled to the size of the original image, stacked together, and fed through a fully connected layer to output color values. Instead of predicting HSL values, we choose to predict RGB values and only use the MSE loss on predicted and true color values with a scale of 0.1. Qualitatively, this has worked well in terms of learning to predict colors, as shown in Figure 4.4. We observe that we are able to accurately predict colors such as black/white car colors, building colors, road and sky colors, and light brightness. However, some colors are missing in our predictions that differ from the truth, such as the red in the stoplight. Instead, a white light is predicted, likely because the network is confusing the stoplight for a street light. We also observe that the true videos are tinted slightly different colors due to the glass window of the car. For example, the first row shows a blue tint while the third image shows a green tint. The predicted colors do not reflect these colors, likely because the appearance of these colors are vehicle specific and the network predicts the colors in the general case.

An important training detail we note here is that we did not have enough GPU memory to predict the colors of images of the original size. As a result, we instead downsampled the images to 1/4th of the original width and height for the self-supervised task.

We again observe a small benefit in co-training with colorization, shown in Table 4.2. However, the difference in AP is small and could be due to noise, so in the future we would like to investigate methods to further attain the benefits of co-training with colorization. We note that we do not use the same baseline number as angle distance, as the baseline numbers are computed by setting the scale of the self-supervised task loss to 0. In addition, the training configurations are slightly different for colorization and angle distance, as not all images have angle and distance labels. These images are filtered out for the angle distance prediction task but not for the colorization task, which may have an effect on the final results.

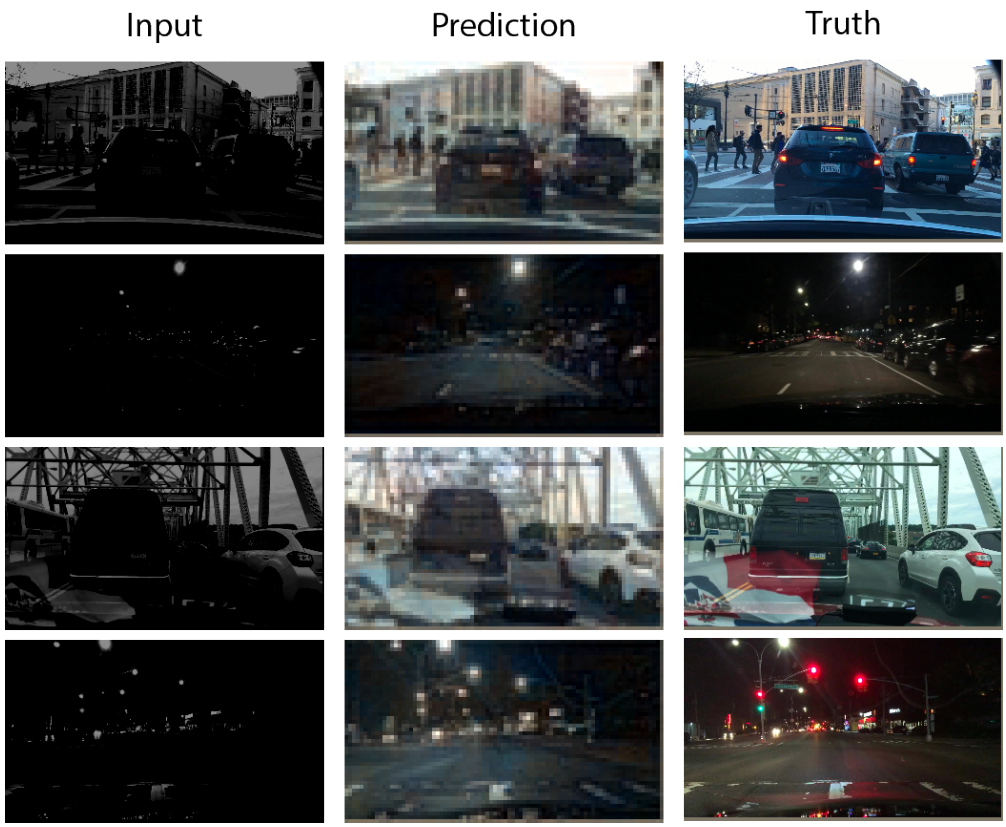



Figure 4.4: Qualitative results for colorization, where we have the black and white input images which are colorized by our network. We also have the original colored image.

Table 4.2: Effect of co-training with the colorization task on object detection AP.

| Configuration | AP |
|---|---|
| Baseline | 30.3045 |
| Colorization | 30.775 |

## 4.3 Conclusion

In this section, we show important training details to take into account when co-training with the angle distance prediction and colorization tasks as well as some preliminary results in improving performance. For the angle distance task, we noted the significance of batch size and the number of images between two sampled frames. For colorization, we noted a necessary workaround to memory constraints by downsampling the images. For both tasks, we showed minor benefits in AP for the object detection task when training with the self-supervised

tasks, and in the future we would like to investigate ways to enlarge these benefits.

# Chapter 5

# Applications to Few Shot Learning

## 5.1 Introduction

In this section, we explore compositional learning of tasks. In attempting to learn a new task to which we have limited data, we use tasks to which we have data to generate data for the new task. We then train a classifier on the generated data for the new task.

At its root, this breaks down to exploiting task compositionality [16]. For example, we may not have data of any examples of a *red elephant* (novel composition) but instead we have data of *red apples* or *red tomatoes*, as well as *big elephants*. We study the zero-shot compositional learning task, where the model needs to recognize novel attribute-object compositions of which no training images are available by transferring knowledge from seen compositions.

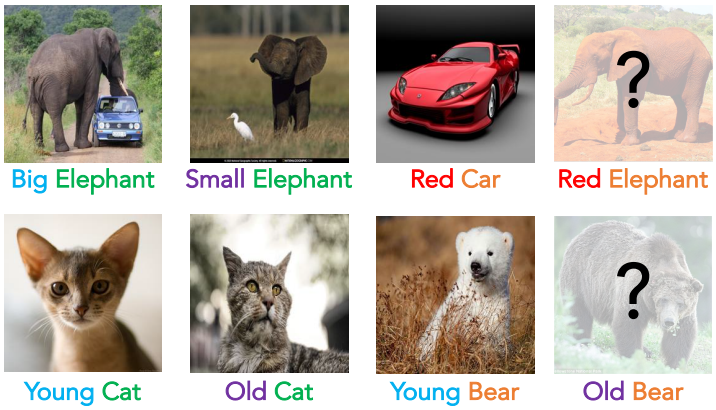



Figure 5.1: The task of zero-shot compositional learning is to build a classifier for recognizing visual concepts represented by an attribute-object pair (e.g., `old bear`) where no training images of the composition are available. Our model generates synthetic features for novel compositions, transferring knowledge from the observed compositions (e.g., `old cat`, `young bear`). The synthetic features are used for training the classifier directly.

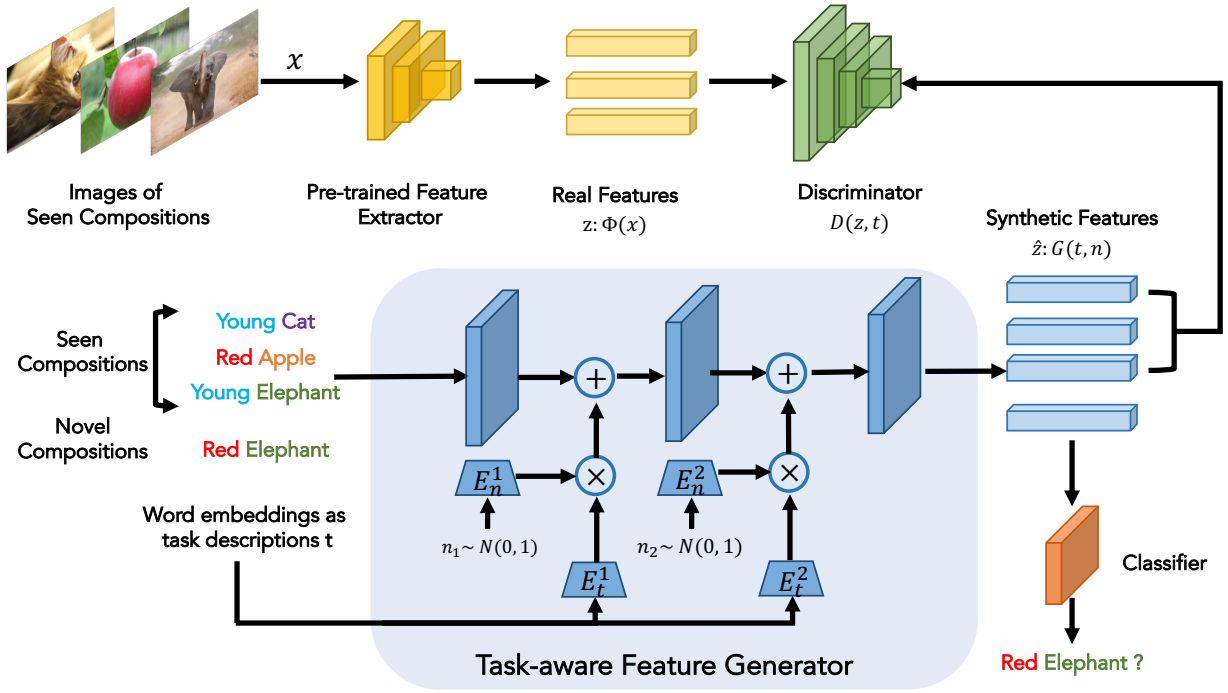



Figure 5.2: The task-aware deep sampling architecture studied here, during training, synthesizes features conditioned on the word embeddings of the composition, which is used to train a classifier for recognizing the seen and novel compositions. The discriminator is introduced to distinguish the real and synthetic features of the seen compositions. The classifier is used after training on the generated data.

In [31], we proposed a task-aware feature generation approach, consisting of a task-aware feature generator, a discriminator, and a classifier (Figure 5.2). During training, the feature generator synthesizes features conditioned on the word embeddings of the compositions (namely, task descriptions) and the discriminator is trained to distinguish the synthetic and real features of the seen compositions. The classifier is jointly trained to recognize the novel compositions using only the synthetic features. During inference, we just use the trained classifier to directly recognize features of novel compositions as if it were trained on the real features.

This is *task-aware deep sampling*, where the generator adopts the task description as input and task conditional randomness is incorporated incrementally at each level.

## 5.2 Zero-Shot Compositional Feature Synthesis

More formally, we are given a vocabulary of attributes $a \in \mathcal{A}$ and objects $o \in \mathcal{O}$ as well as a set of image features $\Phi(\mathcal{X})$ extracted by some pre-trained feature extractors (e.g., ResNet [4]). A visual concept (a.k.a. category) is represented as an attribute-object pair $c = (a, o) \in \mathcal{C}$ and each image is associated with one composition $c$. Moreover, $\mathcal{C} = \mathcal{S} \cup \mathcal{U}$, where the images in the training set associate with the compositions in $\mathcal{S}$ and not with the compositions in $\mathcal{U}$. We refer compositions in $S$ as the seen compositions and the compositions in $\mathcal{U}$ as novel compositions. Following the tradition of classic zero-shot learning [32], the goal is to build a classifier $f$ which classifies an image feature $z : \Phi(x) \in \mathcal{Z}$ using the labels in set of the novel compositions $c \in \mathcal{U}$ (close world setting) or using the labels in the set of all compositions $c \in \mathcal{C}$ (open world setting). We use the concatenation of word embeddings ($t \in \mathcal{T}$) of each attribute-object pair $c$ as the task description for recognizing the composition $c$ and $\mathcal{T}$ is available during training.

## 5.3 Task-Aware Feature Generation

We view the zero-shot compositional learning task from the generative modeling perspective. The key insight is to learn a projection from the semantic space $\mathcal{T}$ to the image feature space $\mathcal{Z}$ via feature synthesis, rather than projecting the two sources of inputs ($z$ and $t$) independently into one common embedding space and building a model to leverage the compatibility between the two modalities [30, 18, 16].

We now introduce our task-aware feature generator design $G : \mathcal{T} \rightarrow \mathcal{Z}$ for image feature synthesis. As illustrated in Figure 5.2, the task description $t \in \mathbb{R}^d$ ($d = 600$ using the GloVe [20] to obtain the word embedding of the compositon) is used as the input to $G$ (instantiated as a stack of fully-connected (FC) layers). At the $i$-th layer of $G$, random Gaussian noise $n_i \sim \mathcal{N}(0, 1)$ is sampled and then transformed by a sub-network of 2 FC layers, $E^i_n$, obtaining transformed noise $E^i_n(n_i)$. The task description $t$ transformed by $E^i_t$ (also a single FC layer) is multiplied with the transformed $E^i_n(n_i)$ to obtain the task-conditioned noise, which is then added to the immediate output of the $i$-th layer of $G$. Specifically, $\hat{z}^{i+1}$, the input of the $(i + 1)$-th layer of $G$, is obtained by

$$\hat{z}^{i+1} = \hat{z}^i + E^i_t(t) * E^i_n(n_i). \tag{5.1}$$

The feature synthesis procedure can be viewed as using *task-aware deep sampling*; different sets of task-conditioned noise are sampled at different levels of the generator, which progressively injects task-driven variation to the immediate features of the generator. Intuitively, the noise injected to the generator is sampled from a task constrained space, which reduces the number of samples necessary to learn the projection from the task space $\mathcal{T}$ to the image feature space $\mathcal{Z}$. We empirically show that our generator design has better sample efficiency than the alternative in the experiment section.

## 5.4 Overall Objective

The overall model pipeline is composed of a generator $G$, a discriminator $D$ and a classifier $f$ as illustrated in Figure 5.2. The discriminator is used during training to distinguish whether the input feature of seen compositions is real or fake. We use a simple logistic regression model as our classifier. All three components are jointly trained in an end-to-end manner and only the trained classifier is used during testing. The overall objective is described below.

### Classification Loss

The synthetic features are tailored to help the classifier generalization. We include a typical multi-class cross-entropy loss part of the objective function. Specially, the classifier $f$ takes the synthetic features $\hat{z} = G(t, n)$ as input ($n = \{n_i \sim \mathcal{N}(0, 1)|i = 1 \dots K\}$, $K + 1$ is the number of layers of the generator) and output the class prediction $\hat{y} = f(\hat{z})$. The classification loss is defined as

$$\mathcal{L}_{\text{cls}} = -\mathbb{E}_{\hat{z}\sim p_{\hat{z}}}[\log P(y|\hat{z}; \theta)], \tag{5.2}$$

where $y$ is ground truth composition that associated with the task description $t$. $P(y|\hat{z}; \theta)$ is the conditional probability predicted by the classifier $f$ parameterized by $\theta$.

### Adversarial Training

We include a GAN loss to help train the generator. We extend the WGAN [3] by integrating the task descriptions $t$ to both the generator and the discriminator. The extended WGAN loss can be defined as

$$\mathcal{L}_{\text{wgan}} = \mathbb{E}_{z\sim p_r}[D(z, t)] - \mathbb{E}_{\hat{z}\sim p_g}[D(G(t, n), t)], \tag{5.3}$$

which approximates the Wasserstein distance commonly used in the GAN literature to improve training stability compared to the original GAN loss [2]. $p_r$ and $p_g$ denote the real feature distribution and the generated feature distribution respectively. We add gradient penalties to the discriminator to enforce the discriminator to be a 1-Lipschitz function following [3, 33]. The overall adversarial loss is defined as

$$\mathcal{L}_{\text{adv}} = \mathcal{L}_{\text{wgan}} - \lambda_{\text{gp}}\mathbb{E}(\| \bigtriangledown_{\tilde{z}} D(\tilde{z}, t)\|_2 - 1)^2], \tag{5.4}$$

where $\tilde{z} = \alpha z + (1 - \alpha)\hat{z}$ with $\alpha \sim \text{Uniform}(0, 1)$. Following [3, 33], we set $\lambda_{\text{gp}} = 10$ in our experiments.

Under the zero-shot learning context, only image features of the seen compositions $S$ are available during training; therefore, the adversarial loss $\mathcal{L}_{\text{adv}}$ is only applied to the seen compositions.

### Clustering Loss

To circumvent the challenges of estimating the image feature distribution, we add a regularization term to make the synthetic features of the seen compositions closer to the cluster center of the true feature distribution. Intuitively, in the extreme case where no randomness is introduced to the generator, $G$ learns a mapping from $t$ to a "prototypical" image feature $\bar{z} \in \mathcal{Z}$. We find this regularization term reduces the complexity of modeling the target image feature distribution.

We realize the mapping by introducing a soft-clustering term with $L_2$ regression loss. Specifically, we randomly sample a real image feature $z$ of the composition $c \in \mathcal{S}$, and regularize the generated feature $\hat{z}$ to be close to $z$. By sampling multiple image features, we regularize the generated feature closer to the cluster center of the real image feature distribution. Similar to $\mathcal{L}_{\text{adv}}$, the prototypical loss term, defined as

$$\mathcal{L}_{\text{cluster}} = \sum_{k}^{K} \|\hat{z}_k - z_k\|^2, \tag{5.5}$$

where $K$ features from the seen composition $c$ are sampled.

### Overall Objective

The overall objective is a weighted sum of the three components shown as

$$\min_{G,C} \max_{D} \mathcal{L}_{\text{wgan}} + \lambda \mathcal{L}_{\text{cls}} + \mu \mathcal{L}_{\text{cluster}}, \tag{5.6}$$

and we adopt $\lambda = 0.01$ and $\mu = 10$ if not specified. Ablations of $\lambda$ and $\mu$ are provided in the supplementary material.

## 5.5 Training and Testing

Figure 5.2 shows the different components of the our model: the generator $G$, the discriminator $D$ and the classifier $f$. The generator $G$ and the discriminator $D$ are only used to assist the training of the downstream classifier. If $G$ can generate samples that capture the data distribution of the novel composition by transferring the knowledge from the seen compositions, a classifier trained with the synthetic features should generalize to the real features of the novel compositions during testing. To this end, we train all three components ($G$, $D$ and $f$) jointly and during testing, we directly feed the real features of the novel compositions extracted by the pretrained feature extractor to the trained classifier.

## 5.6 Experiments

We present the experimental evaluation of TFGon three zero-shot compositional learning (ZSCL) benchmark in Section 5.7. Our method outperforms the previous discriminative


Figure 5.3: Data samples of the MIT-States and UT-Zap50K datasets. An attribute-object composition is associated to each images. Only a subset of the composition is seen during training. Both MIT-States and UT-Zap50K are fine-grained recognition datasets where images in MIT-States come from natural scenes while images in UT-Zap50K are mostly with white background, depicting shoes with different materials.

models by a large margin. In Section 5.8, we evaluate our model on the new data splits introduced by the recent work [23] in the generalized ZSCL setting. We find our model is able to improve the previous methods by over 2×, establishing a new state of the art.

## 5.7 Zero-Shot Compositional Learning

We conduct experiments on three datasets: MIT-States [6], UT-Zap50k [34] and Stanford-VRD [12]. For MIT-States, samples of which are shown in Figure 5.3 left, each image is associated with an attribute-object pair, e.g., *modern city*, *sunny valley*, as the label. The model is trained on 34K images with 1,292 labeled seen pairs and tested on 34K images with 700 unseen pairs. The UT-Zap50k dataset (samples shown in Figure 5.3 right) is a fine-grained dataset where each image is associated with a material attribute and shoe type pair (e.g., `leather slippers`, `cotton sandals`). Following [18], 25k images of 83 pairs are used for training and 4k images of 33 pairs for testing. We also consider compositions that go beyond attribute-object pairs. For StanfordVRD, the visual concept is represented with a SPO (subject, predicate, object) triplet, e.g., `person wears jeans`, `elephant on grass`. The dataset has 7,701 SPO triplets, of which 1,029 are seen only in the test set. Similarly to [16], we crop the images with the ground-truth bounding boxes and treat the problem as classification of SPO tuples rather than detection. We obtained 37k bounding box images for

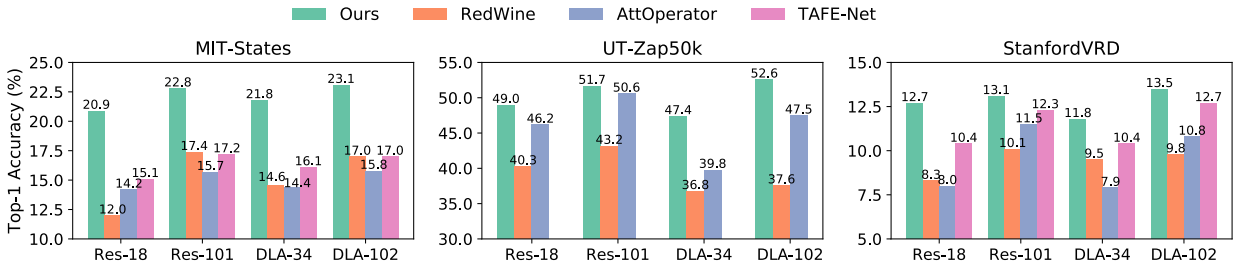



Figure 5.4: Top-1 accuracy of unseen compositions in compositional zero-shot learning on MIT-States (700 unseen pairs), UT-Zap50K (33 unseen pairs) and StanfordVRD (1029 unseen triplets). TFG(the first bar in each group) achieves state-of-the-art results on all three datasets with four different feature extractors (ResNet-18, ResNet-101, DLA-34 and DLA-102).

training and 1k for testing.

## Experimental Details

In the experiments, we extract the image features with ResNet-18 and ResNet-101 [4] pretrained on ImageNet following [16, 18, 30] and also include the more recent DLA-34 and DLA-102 [36] for benchmarking. We report the top-1 accuracy of the unseen compositions following [16, 30]. We use Glove [20] to convert the attributes and objects into 300-dimensional word embeddings. In practice, the raw word embeddings of attributes and objects are transformed by two 2-layer FC networks $\phi_a$ and $\phi_o$ with the hidden unit size of 1024. $\phi(t)$ is the concatenation of $\phi_a(a)$ and $\phi_o(o)$ used as input to both $G$ and $D$.

The discriminator $D$ is a 3-layer FC networks with hidden unit size of 1024. For the generator $G$, we use a 4-layer FC network where the hidden unit size of the first three layers is 2048 and the size of the last layer matches the dimension of the target feature dimension. $E_t$ is a single layer FC network with no bias and hidden unit size matching the corresponding feature layer size of the generator. $E_n$ is a 2-layer FC network where the hidden unit size is 1024 in the first layer, matching the corresponding feature layer size of the generator in the second layer. The classifier $f$ is a simple soft-max classifier with one FC layer. We adopt the Adam [9] optimizer with an initial learning rate of $10^{-5}$ for the embedding network $\phi$ and $10^{-4}$ for the other parameters. We divide the learning rate by 10 at epoch 30 and train the network for 40 epochs in total, reporting the accuracy of the last epoch. The batch size is 128.

### Quantitative Results

We present the top-1 accuracy of the unseen attribute-object pairs in Figure 5.4 following [16, 18, 19, 30]. We consider three top performing models as our baselines. Redwine [16] leverages the compatibility of extracted generic image features $\mathcal{Z}$ and the task descriptions $\mathcal{T}$ with a simple binary cross entropy (BCE) loss. AttOperator [18] proposes to use attributes features to modify the object features building on top of the extracted features. It also adopts a metric-learning approach to score the compatibility of transformed image feature as well as the task embeddings. In our experiments, we report the results of these two methods on different backbone feature extractor using the open-sourced code from Nagarajan and Grauman [18][1]. TAFE-Net [30] is a recent method that learns a task-aware feature embeddings for a shared binary classifier to classify the compatibility of task-aware image feature embeddings and the task embeddings. We obtain the benchmark results using DLA as the feature extractor through the released official code[2]. Our classifier is directly trained on the synthetic image features of the unseen compositions and at testing time, only the real image features of the unseen compositions are fed into the classifier, not combined with the task descriptions as the existing approaches do.

We present the qualitative results in Figure 5.4. As we can observe from the bar charts, our model (denoted as the green bar, the first bar in each group) outperforms the other baseline methods by a large margin on both MIT-States and UT-Zap50k. Extending from attribute-object pairs to (subject, predicate, object) triplets, our model also outperforms all the considered baselines. This indicates that TFGeffectively synthesizes the real image feature distributions of the novel compositions and helps the classifier generalize to novel concepts without using real image features.

## 5.8 Generalized Zero-Shot Compositional Learning

In this section, we provide evaluation on the *generalized* zero-shot compositional learning recently introduced by Purushawakam *et al.* [23]. As pointed by Purushawakam *et al.*, the previous zero-shot compositional learning benchmark does not carefully evaluate the overall system performance when balancing both the seen and unseen compositions. Therefore, they introduce new data splits of the MIT-States and UT-Zap50k datasets and adopt the AUC value as the evaluation metric to examine the calibrated model performance. Our model is able to outperform the previous methods by a large margin with an over 2× accuracy on the MIT-States dataset.

[1] `https://github.com/Tushar-N/attributes-as-operators`

[2] `https://github.com/ucbdrive/tafe-net`

## Data Splits

In this generalized ZSCL task, the two datasets (MIT-States and UT-Zap50k) have the same images as used in the ZSCL task. In the new data split, the training set of MIT-States has about 30K images of 1262 compositions (the *seen* set), the validation set has about 10K images from 300 seen and 300 unseen compositions. The testing set has about 13K images from 400 seen and 400 unseen compositions. On the UT-Zap50K dataset, which has 12 object classes and 15 attribute classes, with a total of 33K images. The dataset is split into a training set containing about 23K images of 83 seen compositions. The validation set has about 3K images from 15 seen and 15 unseen compositions. The testing set has about 3K images from 18 seen and 18 unseen pairs.

## Metric

Instead of using the top-1 accuracy of the unseen compositions, Purushawakam *et al.* [23] introduce a set of calibration biases (single scales added to the scores of all unseen pairs) to calibrate the implicit bias imposed to the seen compositions during training. For a given value of the calibration bias, accuracies of both the seen and unseen compositions are computed. Because the values of the calibration bias have a large variation, we draw a curve of the accuracies of seen/unseen compositions and the area blow the curve (AUC) can describe the overall performance of the system more reliably.

## Quantitative Results

Table 5.1 provides comparisons between our model and the previous methods on both the validation and testing sets. The network structures of our model is the same as those used in the ZSCL task and the best training epochs are decided by the validation set. As Table 5.1 shows, our model outperforms the previous methods by a large margin. On the challenging MIT-States dataset which has about 2000 attribute-object pairs and is inherently ambiguous, all the baseline methods have a relatively low AUC score while our model is able to double the performance of the previous methods, indicating the effectiveness of our model.

## Alternative Generator Designs

We analyze two central differences of our task-aware deep sampling (TDS) strategy: deep sampling and multi-step task conditioning. We considered four generator designs using other sampling strategies as depicted in Figure 5.5. The leftmost, shallow sampling (SS) takes noise and the task as input once at the beginning whereas our TFGrepeatedly injects task-conditioned noise at each layer. Unconditional Deep Sampling (UDS) in contrast injects noise at each layer but does not use the task information at each layer. Two other variants, shallow sampling with multi-step task conditioning (SS-MTC) include SS-MTC+, which adds task information at every layer to the generator, and SS-MTC$*$, which adopts an affine

Table 5.1: AUC in percentage on MIT-States and UT-Zap50K. Our model outperforms the previous methods by a large margin, doubling the performance of the prior art on MIT-States.

| | **MIT-States** | | | | | | **UT-Zap50K** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Val AUC | | | Test AUC | | | Val AUC | | | Test AUC | | |
| Model Top $k\rightarrow$ | 1 | 2 | 3 | 1 | 2 | 3 | 1 | 2 | 3 | 1 | 2 | 3 |
| AttOperator [18] | 2.5 | 6.2 | 10.1 | 1.6 | 4.7 | 7.6 | 21.5 | 44.2 | 61.6 | 25.9 | 51.3 | 67.6 |
| RedWine [16] | 2.9 | 7.3 | 11.8 | 2.4 | 5.7 | 9.3 | 30.4 | 52.2 | 63.5 | 27.1 | 54.6 | 68.8 |
| LabelEmbed+ [18] | 3.0 | 7.6 | 12.2 | 2.0 | 5.6 | 9.4 | 26.4 | 49.0 | 66.1 | 25.7 | 52.1 | 67.8 |
| TMN [23] | 3.5 | 8.1 | 12.4 | 2.9 | 7.1 | 11.5 | 36.8 | 57.1 | 69.2 | 29.3 | 55.3 | 69.8 |
| TFGTDS [31] | **8.9** | **18.0** | **25.5** | **6.5** | **14.0** | **20.0** | **41.1** | **65.3** | **78.1** | **32.4** | **58.1** | **70.9** |

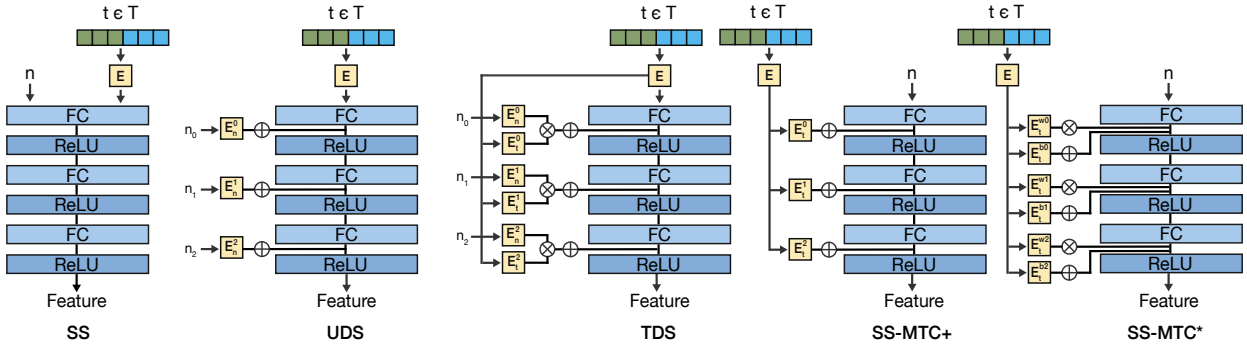



Figure 5.5: Depictions of various architectures, SS, UDS, TDS, SS-MTC+, SS-MTC*. SS, shallow sampling, does not inject noise at each layer while the rest do. The right three, the task-aware deep sampling (TDS), SS-MTC+, and SS-MTC* inject the task embedding at each layer but only TDS injects task and noise at each layer. We find that our chosen generator design using TDS in the middle of the figure obtains the highest classification accuracy compared to other designs.

transformation of the features conditioned on the task at each level inspired by FiLM [21] and TAFE-Net [30].

In Table 5.2, we present the top-1 accuracy of the unseen compositions on the three datasets using ResNet-18 as the feature extractor. We observe that both SS-MTC+ and SS-MTC∗ have better performance than the vanilla shallow sampling (SS) with single step task conditioning and that SS-MTC∗ has better performance than SS-MTC+ due to the more complex transformation. In addition, we find the unconditioned deep sampling (UDS) is better than SS, though both of them use single step task conditioning. In all cases, the proposed TDS, which utilizes both deep sampling and multi-step conditioning, achieves the

Table 5.2: Top-1 Accuracy of unseen compositions. SS-MTC+ and SS-MTC∗ utilizing multi-step conditioning have better performance than SS. UDS with deep sampling achieves higher accuracies than SS. Overall, Task-aware deep sampling (TDS) achieves better performance than all the alternatives.

| Sampling Strategy | MIT-States Top-1 Acc. (%) | UT-Zap50K Top-1 Acc. (%) | StanfordVRD Top-1 Acc. (%) |
|---|---|---|---|
| SS | 12.4 | 40.0 | 8.3 |
| UDS | 14.8 | 41.4 | 8.7 |
| SS-MTC+ | 18.3 | 43.4 | 9.3 |
| SS-MTC∗ | 19.2 | 44.3 | 10.1 |
| TDS | **20.9** | **49.0** | **12.7** |

best results among all the considered variants.

## 5.9 Qualitative Results

### Visualization of Synthetic Features

Observing Figure 5.6 which depicts the real and generated features of the novel compositions on MIT-States, we can see that the synthetic features (in blue) overlap with the real features (in red). The synthetic features form rough clusters compared to the real features, which may make training of the classifier easier. Zooming in to check different regions of the feature

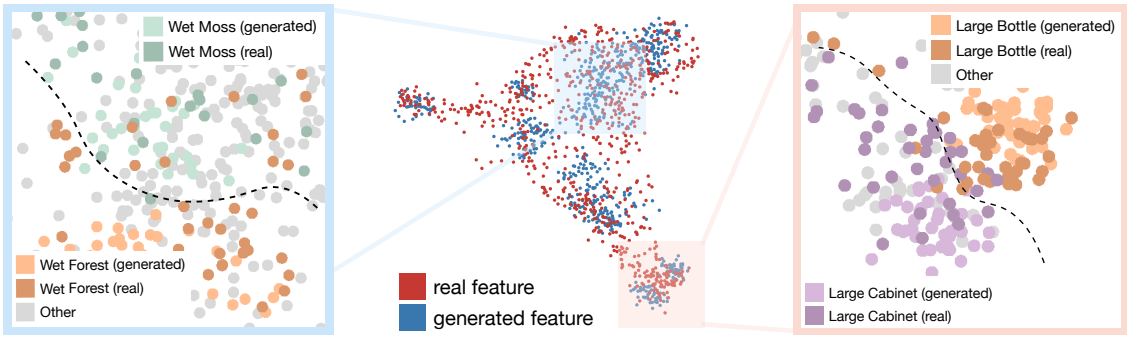



Figure 5.6: Feature visualization of real and generated features of images in the testing set. The center depicts real features, represented as red points, and generated features, represented by generated features visualized using UMAP. Within different regions, we observe in the left and right, that the generated feature distribution closely matches the real feature distribution, and that distributions of different classes are separated.

distributions (in the windows on both sides of Figure 5.6), we find that though semantically closer compositions are also closer in the image feature space, e.g., *wet moss* and *wet forest* in the window on the left, the synthetic features still closely cover the real feature distribution and form cleaner cluster boundaries between different compositions than the real features.

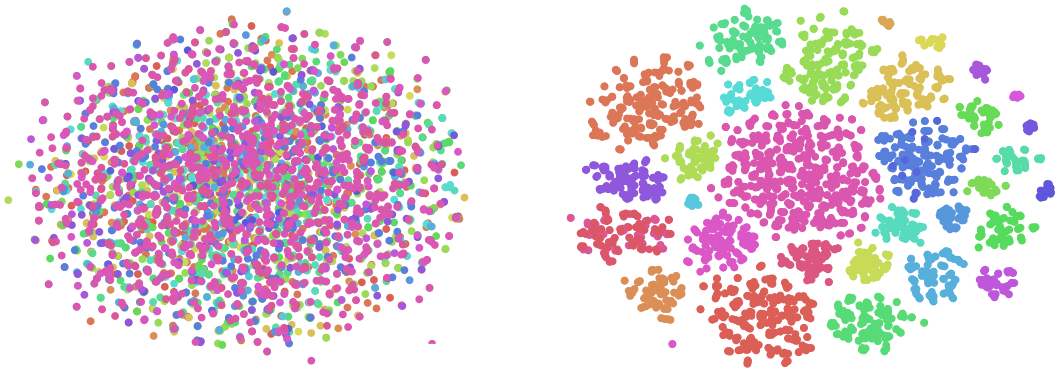


Figure 5.7: T-SNE visualization of the unconditioned noise used in UDS (left) and task-aware noise injected in the last layer of TFG(right) of 33 unseen attribute-object compositions on UT-Zap50K. The task-aware noise is clustered based on the task while the unconditioned noise is mixed in one cluster.

### Visualization of Task-Conditioned Noise

As discussed in the previous section, the task-aware deep sampling (TDS) used in our generator design is one of the key components that allow the model to achieve better sampling efficiency. TDS is different from the unconditional deep sampling (UDS) mainly because of the injection of the task-conditioned noise, which allows for sampling from a task-adaptive distribution.

In Figure 5.7, we visualize the noise injected to the last layer of the generator in UDS and TDS with t-SNE [14] of the 33 unseen compositions on UT-Zappos. We can observe from the figure that the task-aware noise is clustered based on the task while the unconditioned noise is mixed in one cluster. We hypothesize that the task-relevant samples injected to the generator help the generator to estimate the target image feature distribution.

## 5.10 Conclusion

In this section, we reviewed our zero-shot compositional learning task with a compositional feature synthesis approach in [31]. We proposed a task-aware feature generation framework, improving model generalization from the generative perspective. We designed a task-aware

deep sampling strategy to construct the feature generator, which produces synthetic features to train classifiers for novel concepts in a zero-shot manner. The proposed TFGachieved state-of-the-art results on three benchmark datasets (MIT-States, UT-Zap50K and StanfordVRD) of the zero-shot compositional learning task. In the generalized ZSCL task recently introduced by Purushawakam *et al.* [23], our model is able to improve the previous baselines by over $2\times$, establishing a new state of the art. In addition, a visualization of the feature distributions showed the generated features closely model the real image feature distributions with clearer separation between different compositions.

# Chapter 6

# Conclusion and Future Direction

In this report, we studied multi-task learning architectures, applications to object detection in self-driving, and applications to few-shot learning. We were able to propose and study a new architecture of multi-task learning, investigate co-training of different tasks with the object detection task and discuss some important training details, and study a generative method for synthesizing features to help train new tasks.

Given time limitations, we were not able to fully explore applications to object detection. For instance, the method of sharing we used simply involved naively sharing portions of the ResNet backbone. In the future, an investigation of using different multi-task learning architectures such as the one we proposed may assist with the co-training of object detection and self-supervised tasks. Further, other multi-task learning methods could have been investigated such as distillation, which has been shown to be useful in place of hard sharing of network parameters. In terms of our study on architecture, we note that our study is limited to using a single layer of gating. In the future, we would like to investigate our results on test accuracy and interpretability when stacking together multiple gating layers.

# Bibliography


[1] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[2] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[3] Ishaan Gulrajani et al. "Improved training of wasserstein gans". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5767–5777.

[4] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[5] Kaiming He et al. "Mask R-CNN". In: *The IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017.

[6] Phillip Isola, Joseph J Lim, and Edward H Adelson. "Discovering states and transformations in image collections". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1383–1391.

[7] Eric Jang, Shixiang Gu, and Ben Poole. "Categorical reparameterization with gumbel-softmax". In: *arXiv preprint arXiv:1611.01144* (2016).

[8] Longlong Jing and Yingli Tian. "Self-supervised Visual Feature Learning with Deep Neural Networks: A Survey". In: *CoRR* abs/1902.06162 (2019). arXiv: 1902.06162. URL: http://arxiv.org/abs/1902.06162.

[9] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[10] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. "Colorization as a Proxy Task for Visual Understanding". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[11] Mingsheng Long et al. "Learning multiple tasks with multilinear relationship networks". In: *Advances in Neural Information Processing Systems*. 2017, pp. 1594–1603.

[12] Cewu Lu et al. "Visual relationship detection with language priors". In: *European Conference on Computer Vision*. Springer. 2016, pp. 852–869.

[13] Yongxi Lu et al. "Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5334–5343.

[14] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[15] Arun Mallya and Svetlana Lazebnik. "Packnet: Adding multiple tasks to a single network by iterative pruning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7765–7773.

[16] Ishan Misra, Abhinav Gupta, and Martial Hebert. "From red wine to red tomato: Composition with context". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1792–1801.

[17] Ishan Misra et al. "Cross-stitch networks for multi-task learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 3994–4003.

[18] Tushar Nagarajan and Kristen Grauman. "Attributes as operators: factorizing unseen attribute-object compositions". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 169–185.

[19] Zhixiong Nan et al. "Recognizing Unseen Attribute-Object Pair with Generative Model". In: *AAAI 2019*. 2019.

[20] Jeffrey Pennington, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[21] Ethan Perez et al. "Film: Visual reasoning with a general conditioning layer". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[22] Lerrel Pinto and Abhinav Gupta. "Learning to push by grasping: Using multiple tasks for effective learning". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 2161–2168.

[23] Senthil Purushwalkam et al. "Task-Driven Modular Networks for Zero-Shot Compositional Learning". In: *arXiv preprint arXiv:1905.05908* (2019).

[24] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 91–99. URL: `http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf`.

[25] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. "Routing networks: Adaptive selection of non-linear functions for multi-task learning". In: *arXiv preprint arXiv:1711.01239* (2017).

[26] Sebastian Ruder. "An overview of multi-task learning in deep neural networks". In: *arXiv preprint arXiv:1706.05098* (2017).

[27] Noam Shazeer et al. "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer". In: *arXiv preprint arXiv:1701.06538* (2017).

[28] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[29] Xin Wang et al. "Deep mixture of experts via shallow embedding". In: *arXiv preprint arXiv:1806.01531* (2018).

[30] Xin Wang et al. "TAFE-Net: Task-Aware Feature Embeddings for Low Shot Learning". In: *arXiv preprint arXiv:1904.05967* (2019).

[31] Xin Wang et al. "Task-Aware Deep Sampling for Feature Generation". In: *CoRR* abs/1906.04854 (2019). arXiv: 1906.04854. URL: http://arxiv.org/abs/1906.04854.

[32] Yongqin Xian, Bernt Schiele, and Zeynep Akata. "Zero-shot learning-the good, the bad and the ugly". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4582–4591.

[33] Yongqin Xian et al. "Feature generating networks for zero-shot learning". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 5542–5551.

[34] Aron Yu and Kristen Grauman. "Semantic jitter: Dense supervision for visual comparisons via synthetic images". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 5570–5579.

[35] Fisher Yu et al. "BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling". In: *CoRR* abs/1805.04687 (2018). arXiv: 1805.04687. URL: http://arxiv.org/abs/1805.04687.

[36] Fisher Yu et al. "Deep layer aggregation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2403–2412.