TOWARDS UNCERTAINTY-AWARE DISENTANGLED REPRESENTATIONS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SEZAİ ARTUN ÖZYEĞİN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2021

Approval of the thesis:

TOWARDS UNCERTAINTY-AWARE DISENTANGLED REPRESENTATIONS

submitted by **SEZAİ ARTUN ÖZYEĞİN** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Assoc. Prof. Dr. Sinan Kalkan
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Assist. Prof. Dr. Emre Akbaş
Computer Engineering, METU _____

Assoc. Prof. Dr. Sinan Kalkan
Computer Engineering, METU _____

Assoc. Prof. Dr. Hacer Yalım Keleş
Computer Engineering, Hacettepe University _____

Assoc. Prof. Dr. Mehmet Erkut Erdem
Computer Engineering, Hacettepe University _____

Assist. Prof. Dr. Ramazan Gökberk Cinbiş
Computer Engineering, METU _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: SEZAİ ARTUN ÖZYEĞİN

Signature :

ABSTRACT

TOWARDS UNCERTAINTY-AWARE DISENTANGLED REPRESENTATIONS

ÖZYEĞİN, SEZAİ ARTUN
M.S., Department of Computer Engineering
Supervisor: Assoc. Prof. Dr. Sinan Kalkan

September 2021, 71 pages

In many computer vision tasks, not every part of an object of interest is always visible because of challenges like occlusion, viewpoint and pose variation. One approach to these kinds of challenges is separating the representation so that they would correspond to different regions. In this thesis, we tackle the problem of obtaining disentangled representations while estimating the uncertainty of each factor to assess its availability. Representations are disentangled using a factor-related supervised task and by using an adversarial loss, unrelated information is removed. Uncertainty of factors are estimated using loss attenuation over the same factor-related task. We try several methods to integrate uncertainty values into both the training procedure and the decision making process during test time to make the model more robust to unavailable parts. The experiments are conducted over a toy dataset and the person re-identification task (namely, the Market-1501 dataset) which can benefit from disentangled representations.

Keywords: Disentanglement, Uncertainty Estimation, Representation Learning, Deep

Learning

ÖZ

BELİRSİZLİĞE DUYARLI DOLAŞIKLIĞI AÇILMIŞ TEMSİLLERE DOĞRU

ÖZYEĞİN, SEZAİ ARTUN
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Sinan Kalkan

Eylül 2021 , 71 sayfa

Birçok bilgisayarlı görü probleminde ilgilendiğimiz bir objenin her kısmı her zaman görünür olmayabilir. Bunun birkaç nedeni başka bir obje tarafından kapatılma, kameranın bakış açısı veya objenin o sıradaki pozu olabilir. Bu gibi sorunları çözmenin bir yolu, temsili farklı bölgelere kaşılık gelecek şekilde birden çok parçaya ayırmaktır. Bu tezde bir objenin dolaşıklığı açılmış temsillerini elde ediyoruz ve bunu yaparken de faktörlerin ne kadar elde edilebilir olduğunu belirleyebilmek için belirsizlik tahmini yapıyoruz. Her bir temsilin dolaşıklığını açmak için o faktöre ait gözetimli bir problem kullanıyoruz ve çekişmeli yitim fonksiyonu kullanarak ilgisi olmayan bilgileri ortadan kaldırıyoruz. Belirsizliği ölçmek için yine önceden bahsetmiş olduğumuz gözetimli problemi kullanıyoruz ve bu problemin yitim fonksiyonu üzerine yitim sönümleme uyguluyoruz. Bu ölçülen belirsizlik değerlerini modeli eksik olan parçalara karşı daha dayanıklı yapmak için hem test sırasındaki karar verme hem de eğitim süreçlerine entegre eden farklı yöntemler deniyoruz. Deneyleri kendi olusturdugunuz basit bir veri üzerinde ve dolaşıklığı açılmış temsillerden fayda görebileceği için kişilerin yeniden saptanması problemine (Market-1501 veri kümesine) uyguluyoruz.

Anahtar Kelimeler: Dolaşıklık Açma, Belirsizlik Tahmini, Temsil Öğrenme, Derin Öğrenme

To my loved ones...

ACKNOWLEDGMENTS

Firstly, I would express my gratitude to Assoc. Prof. Sinan Kalkan for his invaluable guidance and support. I enjoyed every minute of doing research with him.

I would like to thank my family for their constant support before and during my Master's studies.

For the emotional and professional support I received, I cannot thank enough to my partner, Hazan.

I would like to thank my friends for their support and the good times we have spent together.

I want to thank Prof. Fatoş T. Yarman Vural for teaching me how to do research with love. I also would like to thank Asst. Prof. Emre Akbaş and Asst. Prof. Ramazan Gökberk Cinbiş for their help and guidance in the field.

The numerical calculations reported in this paper were partially performed at TUBITAK ULAKBIM, High Performance and Grid Computing Center (TRUBA resources).

This work was partially supported by the Scientific and Technological Research Council of Turkey (TÜBITAK) through the project titled "Object Detection in Videos with Deep Neural Networks" (project no 117E054).

I would like to thank the Scientific and Technological Research Council of Turkey (TÜBITAK) for the financial support offered in the 2210A - Domestic Graduate Scholarship Program during my education.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ALGORITHMS

ALGORITHMS

LIST OF ABBREVIATIONS

AP	Average Precision
BN	Batch Normalization
CL	Center Loss
CLA	Main Task Classification Loss Attenuation
CMC	Cumulative Matching Characteristics
Dis	Disentanglement
fc	fully connected
FLA	Factor Loss Attenuation
FS	Factor Supervision
ID	Identity
mAP	mean Average Precision
RBC	Recognition by Components
ReLU	Rectified Linear Unit
SOTA	State Of The Art
STN	Spatial Transformer Network

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

(a) (b) (c) (d)

Figure 1.1: Some challenging examples from the Market-1501 dataset [1]. (a) Occlusion. The person-of-interest can occluded by another object. (b) Viewpoint variation. The angle we are capturing the person may change. (c) Pose change. People can be in several poses. (d) Misalignment. Due to the errors of the detection algorithms, a person may not be aligned in the same way throughout their appearances.

In machine learning, how we represent data matters. For example, if we want to compare two images to understand whether the people in them are the same person, a pixel-wise comparison would not tell us much. Instead, we want to map images of people to another space with the property of closeness when the identities of the people are the same. This property may change with respect to the problem we work on. (Bengio et al. [4] discuss thoroughly what makes a representation good.)

Although we would want to obtain a complete representation (a representation without missing components) of an object, in general, this is not possible because of the lack of information in the data. A simple example for this is: When we observe a

person from a camera, we only see the side of them that is facing us. Furthermore, there are other challenges that result in information loss owing to e.g. occlusion – see Figure 1.1. In such cases, this information loss may not be uniform throughout all parts of the object. For example, a motorcycle may occlude the leg part of a person which severely affects the obtained leg representation while the representation of the torso may not be affected at all.

Many works in computer vision aim to obtain a single, holistic representation for an object which may not be robust to missing components, noises, and variations such as viewpoint, pose. Their workflow can be generalized as shown in Figure 2.3a.

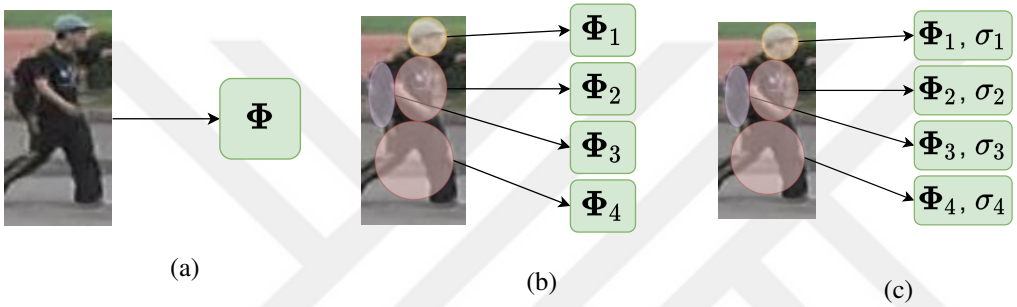


Figure 1.2: Different kinds of representations. (a) Holistic representation. The aim here is to obtain a single vector that represent whole of the object. (b) Part-based representation. Several representations belonging to different parts are obtained. (c) Disentangled representations with uncertainty values. Along with the factor-based representations, uncertainty values related to them are estimated. Note that the disentangled representations or their support in the image may or may not coincide with the parts. Image taken from the Market-1501 dataset [1].

Biederman [5] in his theory Recognition-by-Components (RBC), works on describing how humans recognize objects. RBC proposes that there is some underlying vocabulary of simple shapes that constitute the actual object and humans identify those simple shapes and their arrangements in the scene in their first encounters. This theory explains how humans can recognize objects even in challenging scenarios as the vocabulary of these simple shapes are distinct enough to be recognized from different viewpoints and partial matches are computable.

Similarly in computer vision, one solution to these kinds of challenges is obtaining a separate representation for each part instead of the whole object. This approach is illustrated at Figure 2.3b. In this way, even if the information related to a certain part is missing, we would still have the other parts to make the task-related decisions. As many tasks in computer vision suffer from these issues, they have the potential of gaining benefit from part-based representations. These tasks range from person re-identification to object detection.

There are some underlying factors that generate our data. These can be simple ones like the t-shirt color of a person or more complex ones such as the body pose of a person, the light source position, shadows. Interaction of these factors can be rather complicated; however, these factors can be separated into groups so that variation in one of the groups does not affect the factors in the other groups. Therefore, disentangling underlying factors is another way to separate representation. Zhao et al. [6] use attribute information to guide separate features to represent different parts. Similarly, in this thesis, we investigate the relevance of the disentangled representations with the actual parts. These two concepts may suggest to separate the data differently; for example, the color and length of a T-shirt belong to the same part while they might be represented independently. Whereas, having a handbag and a backpack may have a negative correlation in the data although they are attributes of different parts.

While obtaining separate representations, the knowledge of how much information is available at each one of them would be valuable as we can focus on the ones that we are certain of and disregard the others. Therefore, we also integrate uncertainty estimation to our system for each representation, which would give us a hint about how much that representation is corrupted. This approach is shown in Figure 2.3c. It should be mentioned that these representations or their support in the images may or may not coincide with object parts. By considering uncertainty value of a specific representation, we may be able to combine, compare, or do any task-related decision in a more reliable manner.

In this thesis, we investigate whether we can address the challenges that affect parts differently such as occlusion, viewpoint change, and body pose variation by obtaining disentangled representations, $\mathbf{\Phi}_j$'s, with also their uncertainty estimations, σ_j's.

1.2 Proposed Methods and Models

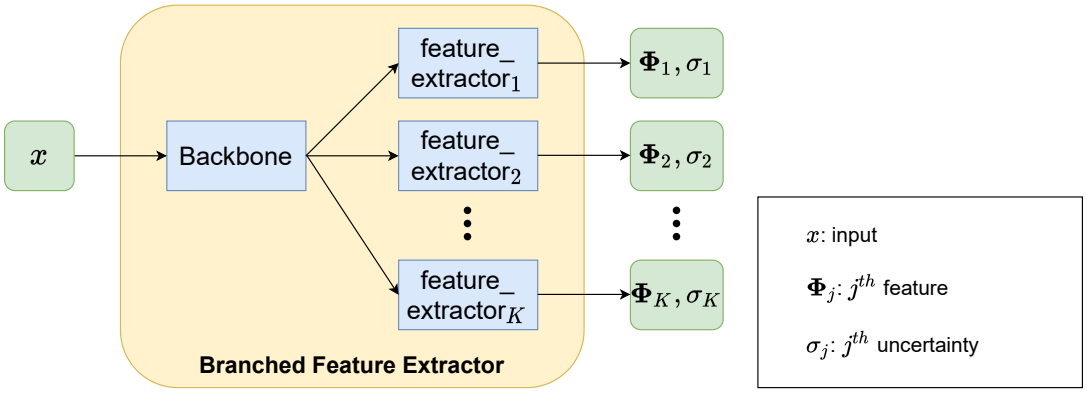


Figure 1.3: High-level architecture. x is the input to the model. It is passed through a backbone network first and processed through K separate branches where K is the number of factors. As a result, a representation $\mathbf{\Phi}_j$ and an uncertainty estimation σ_j are obtained for the j^{th} factor where $j \in \{1, ..., K\}$.

Since we want to consider every representation separately, we first work on extracting disentangled representations (features). Some works achieve this by using a pre-trained pose estimator; then, extracting local features [7], applying affine transformations [8, 9, 10], or masking the features [11, 12], accordingly. In this work, however, we experiment on extracting disentangled representations, $\mathbf{\Phi}_j$'s, by separating the model into branches and guiding each branch to its related factor while cleansing the information related to the other branches. A high-level depiction of this architecture can be seen in Figure 1.3.

To guide each branch to extract features related to its assigned factor, a prediction related to that specific factor is supervised during the training. On top of this guidance, the information belonging to the unrelated factors is eradicated from each branch. This is done by applying an adversarial loss (also called confusion loss) following Alvi et al. [2]. In the end, what we are trying to obtain is independent feature vectors for each factor.

There are different types of uncertainties [13]. One type originates from the lack of knowledge. This type of uncertainty can be reduced by simply having more data. The one we focus on in this work, however, tries to represent the uncertainty in the

prediction which is not related to the knowledge. This may stem from sources such as sensor noise or some randomness inherent in the process such as the output of rolling a die.

For uncertainty estimation, we try to estimate the variances belonging to each branch, σ_j^2's. To do that, the same factor-related label predictions we used are employed. The uncertainty we want to estimate, however, should be input-dependent since we want to address the issues like occlusion, e.g., the occlusion of the leg is specific to some examples, not all. Following Kendall et al. [14] we weigh the loss values of those guidances with the multiplicative inverse of the estimated variance, $\frac{1}{\sigma_j^2}$. We also add an additional penalty term in the loss function, $\log \sigma_j$, so that the loss cannot be made zero by simply increasing the variance. Then, the model tries to find an optimal place between the original loss and the penalty term. This minimizer is towards the low variance when the model is certain about its prediction and high variance otherwise to decrease the effect of a high valued loss function. In our work, we do not estimate the uncertainty related to the knowledge of the model since we specifically want to focus on the challenges we discussed. However, the performance may improve if done so.

One of the reasons for estimating uncertainty values is that it gives us the ability to make task-related decisions using the features available to us. However, it is not limited to this as accounting for the noisy data during training has been shown to make the model more robust on several occasions [13, 15].

1.3 Contributions and Novelties

Our contributions are as follows:

- We tackle the problems that affect parts differently by obtaining disentangled representations using adversarial loss. Adversarial loss has already been applied on disentangling representations into their related factors [2, 16, 17, 18]. We try that approach to obtain disentangled representations using attribute-related supervised tasks on person re-identification. The very same supervision

is applied for the adversarial loss to remove the information related to the other attributes.

- While estimating disentangled features, their uncertainties are also estimated with loss attenuation [14, 19]. Especially, we focus on estimating input-dependent uncertainty since we want to address the challenges such as occlusion, viewpoint change, and pose variation. We experiment on toy and person re-identification datasets and try different ways of incorporating uncertainties into testing.

1.4 The Outline of the Thesis

In Chapter 2, we go over the related work and provide sufficient background for the thesis. We mainly focus on adversarial-loss-based disentanglement, input-dependent uncertainty estimation techniques, part-based representation learning, and person re-identification.

In Chapter 3, we describe how we extracted uncertainty-aware disentangled representations in detail. Several components in the architecture and their relations are formally defined.

In Chapter 4, we explain the experimental settings and discuss the results. Mainly, we have two different datasets we work on: a toy dataset and Market-1501[1] dataset.

In Chapter 5, we summarize, evaluate this work and discuss limitations and possible future research directions.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, we discuss the related work in detail and present the background for our study.

2.1 Adversarial Learning of Disentangled Representations

One of the properties of a good representation is having the independent underlying factors in it disentangled [4]. One of the reasons is that it enhances the generalization capability of the model to unseen examples [20] since the model can easily pick the task-related factors and can remain oblivious to the unrelated ones.

There are many approaches for obtaining disentangled representations. One approach is to use supervision [2, 16, 17, 18] though it is possible to perform disentanglement in an unsupervised way [21, 22].

The disentanglement mechanism employed in this thesis is applying an adversarial loss (also known as confusion loss) to remove unwanted information in the representations. In our case, this unwanted information is the information related to the other factors; however, adversarial loss has been used in many different tasks. Before inspecting the related work on adversarial losses; let us describe what it is first. We are going to base this narration on the work of Alvi et al. [2] since it is the work that we mainly build on and their task expresses the adversarial loss well.

Suppose that we have a dataset where we have the labels for the task we want to work on and also labels related to the underlying factors that are unwanted for the task at hand. We are going to refer to the main task as the primary task and the unwanted

ones as the secondary tasks throughout this chapter. The primary task Alvi et al. [2] work on is age estimation and they have labels for the spurious information they want to get rid of, such as gender, which is a secondary task.

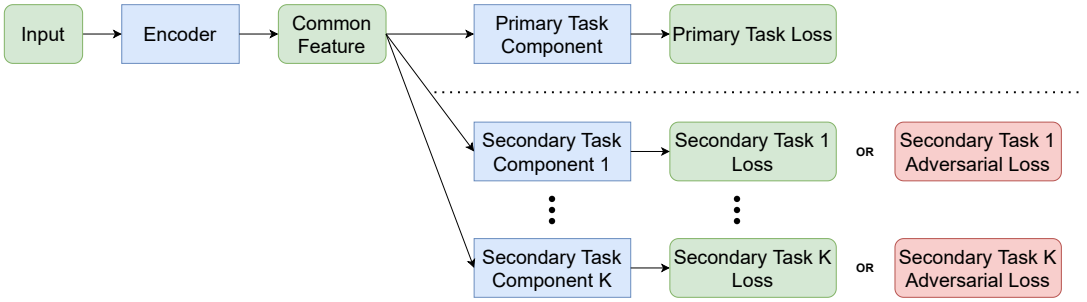


Figure 2.1: High-level architecture for disentanglement with an adversarial loss as it is used by Alvi et al. [2]. A common feature is obtained from an input using an encoder. From that common feature, every component tries to do the task assigned to it. There are one primary task and K secondary tasks in the figure. Blue boxes show the networks, green boxes show the values and the red boxes are the adversarial losses.

To solve the primary task, the encoder part of the model first produces a feature that will be commonly used by many components. This common feature can be seen in Figure 2.1. The most important component that uses this common feature is the one that solves the primary task. This component can be simply implemented as a fully-connected layer. The difference of this component from the others is that the task supervision is applied over it to train both the encoder and the component itself. The primary task loss is named as $\mathcal{L}^{pri}$. The other components can also be simply fully-connected layers. However, they try to do the secondary tasks and if the whole network is simply trained together, instead of getting rid of the unwanted information, it would force the network to represent them in the common feature. Still, we need to train the secondary task components since they will be used while applying the adversarial loss. To prevent the secondary task losses from affecting the whole model, the encoder is frozen while training the secondary task components. Therefore, the training procedure is divided into two: training the secondary task components and training other parts of the network.

Although the secondary task components are trained without affecting the encoder

part of the network now, the unwanted information from the common feature has not been dispelled yet. To do that, the parameters of the secondary task components will be frozen and an adversarial loss will be applied to train the encoder. This adversarial loss can just be the negatives of the original secondary task losses. In the case that a secondary task is classification, we can also force the distribution to be a uniform one as it is the most uninformative discrete distribution. Let the loss of the k^{th} secondary task be $\mathcal{L}_k^{sec}$. Then, if we choose the confusion loss (adversarial loss) coming from the k^{th} secondary task to be the negative of the original loss, the k^{th} confusion loss will be

$$\mathcal{L}_k^{conf} = -\mathcal{L}_k^{sec}. \tag{2.1}$$

The overall confusion loss can be written as

$$\mathcal{L}^{conf} = \sum_{k=1}^{K} \mathcal{L}_k^{conf}. \tag{2.2}$$

By applying such a loss, the encoder is trained in such a way that it produces a common feature from which the secondary tasks cannot be done by the secondary task components. After training the encoder with the adversarial loss, the secondary task components will be re-trained so that they can keep up with the changes in the common features. The training will be alternated in this fashion.

Notice that the training of the encoder with primary task loss is compatible with its training with the adversarial loss. Therefore, these two can be grouped together and we are going to only have two separate training procedures:

1. **Training of the main model.** The primary task component and the encoder are trained with the primary task loss, $\mathcal{L}^{pri}$ and the overall confusion loss, $\mathcal{L}^{conf}$. In this setting, the parameters of the secondary task components are frozen so that they do not get affected by the adversarial loss.

2. **Training of the secondary task components.** The secondary task components are trained with the secondary classification losses, $\mathcal{L}_k^{sec}$'s. In this setting, the encoder is frozen so that it does not get affected by the secondary task losses. This part can be done more than once before alternating to the main model training so that secondary task components can keep up with the changes done by the encoder on the common feature.

As it is discussed, Alvi et al. [2] propose such an architecture to remove the bias of spurious factors like gender in the age estimation task. Their secondary tasks were classification; therefore, they were able to employ uniform distribution as the true distribution in cross-entropy loss. Then, they use this as the adversarial loss to train the encoder so that it produces features that are invariant to the unwanted factors. Notice that their procedure chooses to produce invariant features by getting rid of the unwanted information. However, some works choose to preserve the unwanted information and just disentangle them as we discuss in the following paragraphs.

Liu et al. [16] work on the face identification task. They propose a two-branch network, one estimating identity (ID) related features while the other one estimating identity unrelated features. Different from Alvi et al. [2], they preserve the unwanted attribute information as well and they reconstruct the input face image which essentially makes the architecture an encoder-decoder. They apply the ID information to guide the primary task branch while using the adversarial version of it to dispel the ID-related information from the other branch. Therefore, the other branch can only contain ID-unrelated information.

Jaiswal et al. [17] also propose a two-branch network and similar to Liu et al. [16], they use the task supervision to guide one of the branches. To separate the information encoded in these two branches, however, they train 2 fully-connected layers. One of them predicts the feature of the first branch from the second branch and the other one does the inverse. Then, they use them and send the negative of their losses as the adversarial loss. Another important aspect of this work is that while reconstructing the original image, they corrupt the features coming from the task-related branch so that that branch becomes unreliable. This guides the network to encode all task-unrelated features in the secondary branch.

Robert et al. [18] continue to work on a two-branch encoder-decoder architecture with the addition of attribute supervision. They dedicate one branch to the main task, face identification, while dedicating the other one to attribute recognition. They claim that only with proper, two-sided supervision, can we ensure that the second branch encodes the ID-unrelated information. Then, using these two separate supervisions, they disentangle the two branches.

We mainly base this thesis on the works of Alvi et al. [2] and Robert et al. [18]. These works disentangle the extracted representations using some kind of supervision which both guides the network and eradicates the spurious information in it. They do this to enhance the generalization capabilities of the model or make it more unbiased towards the undesirable information whereas we use it to separate the information related to different factors (attributes in person re-identification case).

2.2 Example Primary Task: Person Re-identification

Figure 2.2: A retrieval example in Market-1501 dataset [1]. A query image on the left-hand side is searched through a gallery of images. The ones shown on the right-hand side are the top 10 retrieved images.

Person re-identification is the task of retrieving a specific person from a set of images obtained from non-overlapping cameras [23]. Basically, we have a query image of a person and we look for the same person in a set called gallery images. This is illustrated in Figure 2.2. Because of the scope of this thesis and the success of the deep models, we are going to limit our coverage to the closest deep-learning-based approaches in this section.

Following Ye et al. [23], we have certain assumptions on how the person re-identification task is performed. First of all, we do not work on raw camera images. Instead, we have the bounding box information for every example. As the number of examples increases throughout the years, annotating these bounding boxes has become a tedious task. Therefore, more recent datasets either use an object detector to extract the bounding boxes or follow a hybrid way [23]. Secondly, during test time, we assume that the person in the query image appears at least once in the gallery set and we only try to retrieve those without questioning its existence.

In general, the achievements in person re-identification can be attributed to 3 factors [23]: representation learning, metric learning, and ranking optimization.

2.2.1 Representation Learning

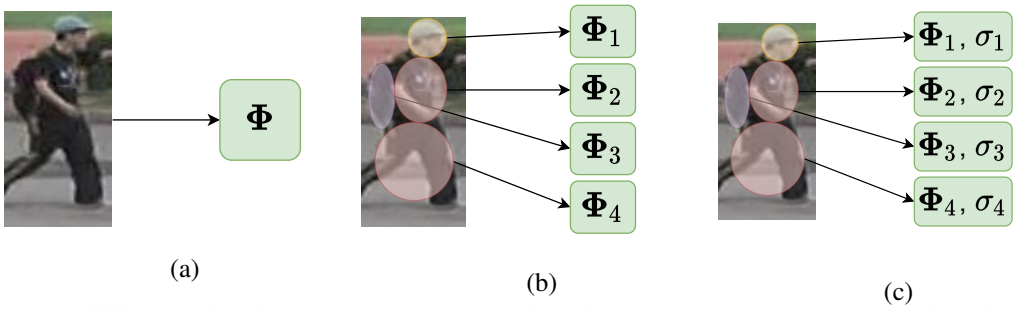


Figure 2.3: Different kinds of representations. This figure is repeated here as it is relevant. (a) Holistic representation. The aim here is to obtain a single vector that represent whole of the object. (b) Part-based representation. Several representations belonging to different parts are obtained. (c) Disentangled representations with uncertainty values. Along with the factor-based representations, uncertainty values related to them are estimated. Disentangled representations may or may not coincide with the parts. Image taken from the Market-1501 dataset [1].

Representation learning aims to obtain a good representation of the person image and this problem is attacked mainly in two ways. The first one aims to obtain a holistic representation, a single feature vector for a person. The second one, on the other hand, tries to extract part/region-based representations which may improve the robustness of the model to challenges such as occlusion, background clutter, pose changes, and viewpoint variations. These different approaches are depicted in Figure 2.3. Furthermore, aside from identity, to enhance the representation, additional information can be included in the procedure like attributes, viewpoint, pose, etc.

As this thesis focuses on obtaining separated representations, we give weight in our discussion to part-based representations. In general, to give the model the ability to focus on a specific section of the input, people employ mechanisms like attention, Spatial Transformer Network (STN) [24] or a pretrained model trained in a super-

vised fashion to specify the part boundaries. In the case of person re-identification, this pretrained model is generally a body pose estimator. After estimating the part locations using this pretrained pose estimator, some extract local features by cutting the input image accordingly [7], some apply affine transformation using an STN [24] guided by the pose [8, 9, 10], some mask the features using the pose estimation network output [11, 12].

Zhao et al. [25] use a fully-convolutional architecture to preserve spatial properties so that they can train several branches that can attend different spatial locations on the feature map and extract region features later. By extracting region-based features, they address the problem of misalignment in person re-identification. Different from ours, they do not apply attribute-based supervision or adversarial loss which could further help the separation of attended regions in the branches.

Zhao et al. [6] propose to use attribute supervision to disentangle parts in video person re-identification to merge part features along the time domain. They do that by splitting the features into parts and guiding them through the supervision of attribute classification. They do not apply adversarial loss to get rid of the information related to other parts. While guiding the part features, as they are predicting discrete distribution over attributes, they use the entropy of it to rate their confidence into the extracted part features and take a weighted average.

2.2.2 Metric Learning

Metric learning can be on the loss function design to guide the representation learning part. The three most-used losses for person re-identification are verification loss, identity loss, and triplet loss [23]:

1. **Verification loss.** In this type of loss, an image pair is processed in parallel and the network decides whether those images come from the same identity or not. In general, a cross-entropy loss is applied to train the network on a two-class classification task. This loss can be shown as

$$\mathcal{L}_{verification} = -\log(\hat{\mathbf{y}}_i), \tag{2.3}$$

where $\hat{\mathbf{y}} \in \mathbb{R}^2$ is the predicted discrete probability distribution and i is 1 if two inputs are from different identities and 2, otherwise.

2. **Identity loss.** This one pretends as if the only possible identities are the ones that occur in the training set. Then, the problem becomes M class classification where M is the number of different identities in the training split. Therefore, we can simply apply softmax and use cross-entropy loss to solve the classification task. Then, the loss becomes

$$\mathcal{L}_{identity} = -\log(\hat{\mathbf{y}}_i), \tag{2.4}$$

where $\hat{\mathbf{y}} \in \mathbb{R}^M$ is the predicted discrete probability distribution and i the index of the ground truth identity in the estimated distribution.

During test time, however, the model will encounter new identities; therefore, the classification part will not work. To solve that, the features produced by the penultimate layer are used to measure the distances between other examples in the gallery set. This distance can simply be euclidean or cosine distance which can be implemented simply by subtracting the cosine similarity from 1.

3. **Triplet loss.** It works directly on the feature space and tries to move the features that belong to the same ID closer and move the others far apart, simultaneously. To do that, we select 2 examples from the same ID and say that they are a positive pair. The first example of this positive pair is called the anchor. In addition, we select another example with a different ID and refer to the newly selected example and the anchor as the negative pair. Let d_p be the distance between the positive pair and d_n be the distance between the negative pair. Then, the triplet loss is defined as

$$\mathcal{L}_{triplet} = \max(0, d_p - d_n + \alpha), \tag{2.5}$$

where α is the margin of triplet loss. The margin permits the network to make the example belonging to the same identity close enough but not the exact same point.

If the pairs are selected randomly, most of the time, the model will be able to distinguish the trivial examples and these examples will not be very informative. To account for that, Hermans et al. propose to select images that are

similar but belonging to different identities, which is called hard negative mining, and also choose images that are different but belonging to the same identity, which is called hard positive mining.

Although these losses affect different spaces, they can be used together. For example, combining verification and identity losses can improve the performance [23]. Similarly, Luo et al. [3] combine identity loss and triplet loss alongside many tricks that exist in the literature to propose a strong baseline. In our experiments, for our strong baseline, we employ a very similar architecture and similar training methods.

2.2.3 Ranking Optimization

Ranking optimization, on the other hand, utilizes the similarities between the images within the gallery set to polish the retrieval results or tries to fuse multiple retrieval results in case we have more than one method. These techniques, in general, improve the retrieval further. However, in this thesis, we present the results and do the comparison without performing a ranking optimization.

2.2.4 Performance Evaluation

Although the specific evaluation process may depend on the dataset, in general, two evaluation measures are used [1, 23]: Cumulative Matching Characteristics (CMC) and mean Average Precision (mAP). Note that, for a specific query, our retrieval result is basically a ranking list of the gallery set excluding the examples coming from the same camera. The overall evaluation is performed using the whole query set.

2.2.4.1 Cumulative Matching Characteristics

CMC is also known as the rank-k (matching) accuracy as it checks whether the correct identity for a specific query is retrieved in the first k results in the ranking list. The overall accuracy is calculated by doing this check over the query set. One thing to notice in this measurement is that if there were more than one ground truth example

in the gallery set for a query image; then, only the one with the highest ranking would matter to CMC. However, many datasets contain multiple ground truths per query [23] and need a measurement that accounts for all related images.

2.2.4.2 Mean Average Precision (mAP)

To describe the calculation of mAP, we first need to define the precision for the first k elements in the ranking list, $p(k)$. Some of the examples in the ranking list are correct retrievals and some are not. $p(k)$ is the number of correct retrievals divided by k:

$$p(k) = \frac{\sum_{i=1}^{k} \text{corr}(i)}{k}, \tag{2.6}$$

where corr(k) is 1 if the k^{th} example in the ranked list is a correct retrieval and 0 otherwise.

Now, we can define average precision (AP). In our case, we can say that calculation of AP is done at the locations of correct retrievals as follows:

$$\begin{aligned} \text{AP}(i) &= \frac{\text{precision at relevant retrievals}}{\text{number of relevant retrievals}} \\ &= \frac{\sum_{k=1}^{M} p(k)\text{corr}(k)}{\sum_{k=1}^{M} \text{corr}(k)}, \end{aligned} \tag{2.7}$$

where AP(i) is the AP of the i^{th} example and M is the length of the ranking list belonging to the i^{th} example.

Mean average precision (mAP) is then the mean of AP values over the whole query set which can be written as

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}(i), \tag{2.8}$$

where N is the number of examples in the query set.

2.2.5 Market-1501 Dataset

Created by Zheng et al. [1, 26], the Market-1501 dataset has been one of the most commonly used benchmarks in person re-identification. It has 751 identities in train split and 750 in its test split. All of the images in this dataset are captured using

six different cameras. Test split splits further into two: query and gallery. Query bounding boxes are hand drawn whereas in the gallery set they are produced using an object detector. The gallery set contains two types of noisy images: one comes from failed object detections and the other one. During test time, examples from the query set are retrieved through the gallery set. The images have 128-pixel height and 64-pixel width.

As the task is retrieval and there are multiple ground truths, the most commonly used evaluation metrics used on Market-1501 are mean average precision (mAP) and additionally, CMC (rank-k accuracy).

2.3 Estimating Uncertainty in Deep Learning

Uncertainty is a valuable source of information that can be beneficial e.g. for increasing robustness of the model to noisy data during training [13, 15] or enhancing a decision making process in noisy situations. In this thesis, however, we work on estimating one specific kind of uncertainty; therefore, it is important to learn some major types of uncertainties and clarify the distinction.

2.3.1 Types of Uncertainty

Kendall and Gal [13] talk about two major types of uncertainties. The first one is related to our lack of knowledge and it is called *epistemic uncertainty*. Its major characteristic is that it can be reduced if the model is introduced with more examples that it has not seen before. Figure 2.4 illustrates this type of uncertainty with red curly braces. Notice that these places lack training data. Another example can be that if our dataset consists mainly of sedan and hatchback cars, then, if it encounters a pickup truck, we expect its epistemic uncertainty to be high since it has never encountered such a vehicle before. However, the model can become more certain of its prediction if we add pickup truck images to its training data.

The second type of uncertainty is *aleatoric uncertainty*. Unlike epistemic uncertainty, aleatoric uncertainty cannot be reduced by introducing new examples since it is not

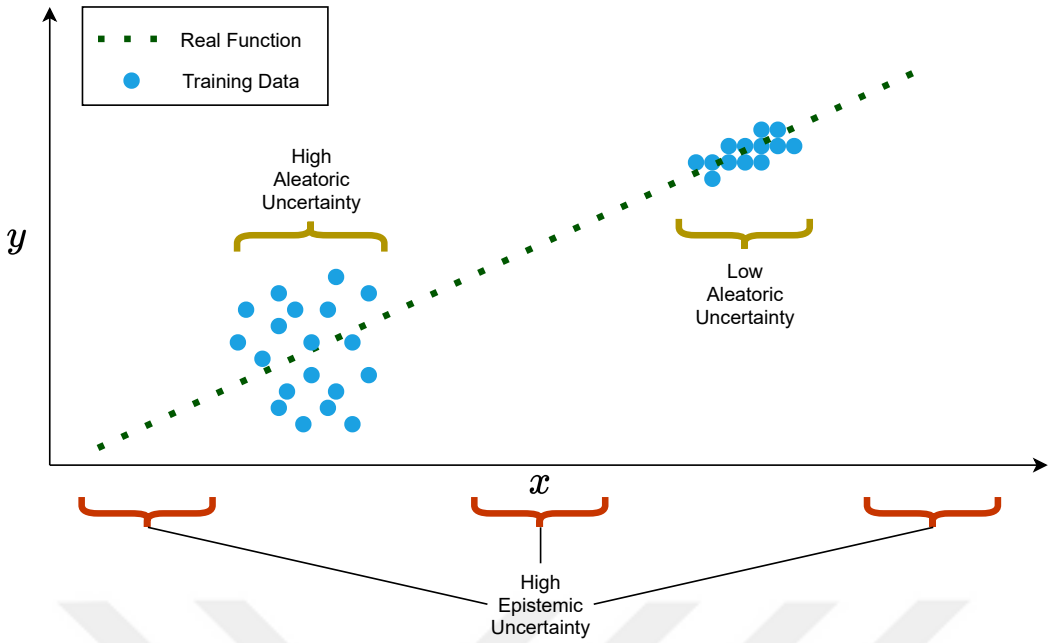


Figure 2.4: Uncertainty types. x is the explanatory variable and y is the response variable. The figure is adapted from the work of Michel Kana [27]. There is high epistemic uncertainty at the places where the data do not have such x values. On the other hand, aleatoric uncertainty is high where we are not sure about the value of the response variable y even we have a lot of data near that x.

related to our knowledge. Aleatoric uncertainty is inherent in a specific example or the task itself. In Figure 2.5, we see that the t-shirt of a person is occluded; therefore, we cannot reduce the uncertainty related to the t-shirt color by just knowing more about t-shirts. As it is stated before, aleatoric uncertainty may depend on a specific example (the case in Figure 2.5). In this case, we call it *heteroscedastic aleatoric* uncertainty. On the other hand, the aleatoric uncertainty related to the task (and independent from a specific example) is called *homoscedastic aleatoric uncertainty*.

This thesis solely focuses on representing the heteroscedastic aleatoric uncertainty since the problems we address (occlusion, pose change, etc.) are all input dependent. Although we could benefit from modeling the epistemic uncertainty, the main point of this thesis is to model the input-dependent uncertainty of disentangled features.

(a) (b)

Figure 2.5: T-shirt occlusion as an aleatoric uncertainty example. (a) Occluded example. As the T-shirt is occluded, having more knowledge will not reduce the uncertainty. (b) The original image which is taken from Market-1501 [1].

2.3.2 Estimating Heteroscedastic Aleatoric Uncertainty

In regression, many works estimate the heteroscedastic aleatoric uncertainty by estimating a distribution and use a technique called loss attenuation. We can assume that the model outputs a Gaussian distribution and let the model estimate both the mean and the covariance matrices, given a specific input. To simplify the model, we can also assume that the model outputs an isotropic Gaussian which means the covariance matrix will be $\sigma \mathbf{I}_k$ where $\mathbf{I}_k$ is k-by-k identity matrix. Let $\hat{\mathbf{y}} \in \mathbb{R}^k$ be the output of our model and $\mathbf{y} \in \mathbb{R}^k$ be the ground truth value. Then, heteroscedastic aleatoric uncertainty can be obtained with loss attenuation as follows:

$$\mathcal{L}_{reg} = \frac{1}{2\sigma^2}||\mathbf{y} - \hat{\mathbf{y}}||^2 + \log \sigma, \tag{2.9}$$

where $\mathcal{L}_{reg}$ is the modified version of the regression loss. Notice that the model has the ability to reduce the loss coming from the regression task, $||\mathbf{y} - \hat{\mathbf{y}}||^2$, by just increasing the σ value. However, there is a penalty for high σ values since the modified loss includes the term $\log \sigma$. Therefore, the model needs to find a balance. However, this balance does not have to be a constant one. For example, if the model is not sure about its output given a specific example, then, it can prefer to increase the σ for that specific example.

For classification, however, since the heteroscedastic aleatoric uncertainty is somehow captured in the output distribution, the solution here is not straightforward. Kendall and Gal [13] propose to put the distribution estimation on the logit level, just before the softmax. Then, they sample from the Gaussian distribution at logit level and estimate the loss using Monte Carlo samples.

Kendall et al. [14] employed homoscedastic aleatoric uncertainty estimation for classification and regression combined for estimating the task weights in losses for multi-task learning. Notice that this uncertainty estimation is not input-dependent. For the classification task, they weigh the logits before the softmax operation. Let $\mathbf{\Phi}$ be the logit level output of the model and σ be a positive scalar. Then, the estimated probabilities of classes, $\hat{\mathbf{y}}$, become:

$$p(\hat{\mathbf{y}}|\mathbf{\Phi}, \sigma) = \text{softmax}\left(\frac{1}{\sigma^2}\mathbf{\Phi}\right). \tag{2.10}$$

Then, with the assumption that $\frac{1}{\sigma}\sum_{c'}\exp(\frac{1}{\sigma^2}\mathbf{\Phi}_{c'}) \approx (\sum_{c'}\exp(\mathbf{\Phi}_{c'}))^{\frac{1}{\sigma^2}}$, we can write the negative log likelihood as follows:

$$\begin{aligned}
-\log\text{softmax}\left(\frac{1}{\sigma^2}\mathbf{\Phi}\right)_c &= -\log\frac{\exp(\frac{1}{\sigma^2}\mathbf{\Phi}_c)}{\sum_{c'}\exp(\frac{1}{\sigma^2}\mathbf{\Phi}_{c'})}, \\
&= -\log\exp(\frac{1}{\sigma^2}\mathbf{\Phi}_c) + \log\sum_{c'}\exp(\frac{1}{\sigma^2}\mathbf{\Phi}_{c'}), \\
&= -\frac{1}{\sigma^2}\mathbf{\Phi}_c + \log\left[\sigma(\sum_{c'}\exp(\mathbf{\Phi}_{c'}))^{\frac{1}{\sigma^2}}\right], \\
&= -\frac{1}{\sigma^2}\mathbf{\Phi}_c + \log\sigma + \frac{1}{\sigma^2}\log\sum_{c'}\exp(\mathbf{\Phi}_{c'})), \\
&= \frac{1}{\sigma^2}(-\mathbf{\Phi}_c + \log\sum_{c'}\exp(\mathbf{\Phi}_{c'}))) + \log\sigma, \\
&= \frac{1}{\sigma^2}(-\log\frac{\exp(\mathbf{\Phi}_c)}{\sum_{c'}\exp(\mathbf{\Phi}_{c'})}) + \log\sigma, \\
&= \frac{1}{\sigma^2}(-\log\text{softmax}(\mathbf{\Phi})_c) + \log\sigma
\end{aligned} \tag{2.11}$$

where $\mathbf{\Phi}_c$ is the c^{th} element of $\mathbf{\Phi}$. Notice that, at the last step of Equation 2.11, the term inside the parenthesis is the cross-entropy loss without the multiplication with $\frac{1}{\sigma^2}$ before the softmax. It means that we are going to weight the original loss with

$\frac{1}{\sigma^2}$ and add a penalty term $\log \sigma$, which is very similar to what had been done in heteroscedastic aleatoric uncertainty estimation in regression.

As mentioned above, Kendall et al. [14] apply this on homoscedastic aleatoric uncertainty estimation; however, in our case, we need the heteroscedastic one. Cai et al. [19] apply this in heteroscedastic aleatoric uncertainty estimation, where they work on data-dependent sample weighing in object detection. Similarly, in this thesis, we employ the homoscedastic aleatoric uncertainty estimation method proposed by Kendall et al. [14] for heteroscedastic aleatoric uncertainty estimation.

CHAPTER 3

UNCERTAINTY-AWARE DISENTANGLED REPRESENTATIONS

In this section, we describe our methodology in detail.

3.1 Overview

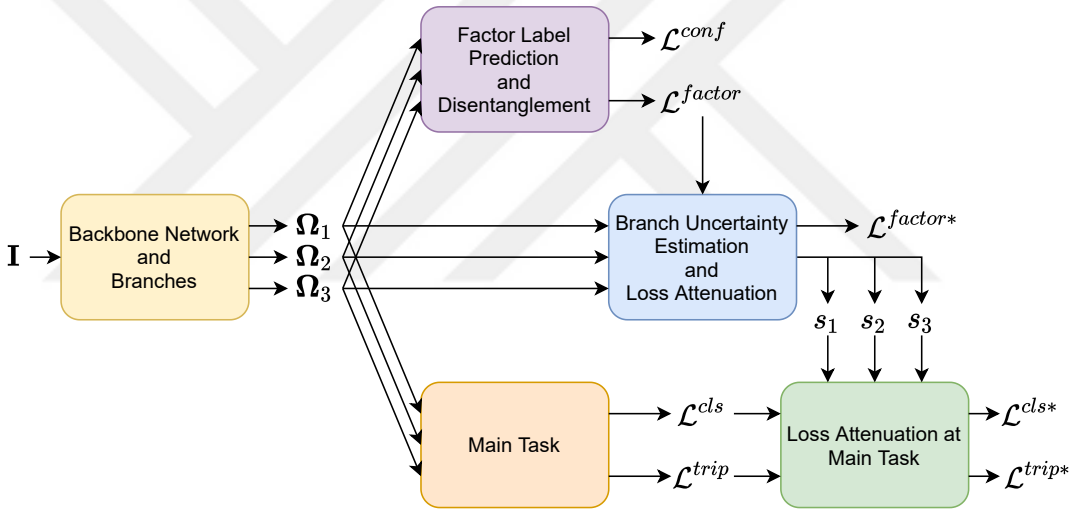


Figure 3.1: The illustration of the sections in this chapter. The block names match with the section names so that the reader can follow the chapter easily. $\mathbf{I}$ is the input image, $\mathbf{\Omega_j}$'s are the branch features, s_j's are the estimated uncertainty values, $\mathcal{L}^{factor}$ is the factor classification loss, $\mathcal{L}^{conf}$ is the confusion (adversarial) loss, $\mathcal{L}^{cls}$ is the main task classification loss and $\mathcal{L}^{trip}$ is the main task triplet loss. The stars $(*)$ marks the loss attenuation applied over a loss. The figure shows only 3 branches for illustration purposes.

We suppose that there are various underlying factors that affect the generation of the data. In our method, we extract disentangled features using supervision related

to those factors. Therefore, we work on a dataset with some factor-related labels alongside the main task. For example, this can be a person re-identification dataset with its attributes like t-shirt color labeled. In this section, we are going to assume that the main task and the factors have discrete labels and thus, we are mainly working on the classification problem. An additional triplet loss will also be incorporated into the design.

Our architecture has multiple branches, each dedicated to a different factor. At each branch, the factor-related class will be predicted and its loss will be attenuated with an estimated scalar uncertainty. These uncertainties will belong to the branches themselves. Additionally, these uncertainties will be used while blending the features with each other to do the main task-related prediction. As our method consists of many components, we divided this chapter into several sections, each discussing and adding an additional structure. The section names are given in Figure 3.1 with boxes and they are placed so that the discussion in that section is related to that specific part in the architecture.

3.2 The Dataset

We assume that our dataset has K different factors labeled for each example and that the dataset has M different classes for the main task and M_k many different class labels for the factor k. Let $\mathcal{S} = \{(\mathbf{I}^{(i)}, y^{(i)}, a_1^{(i)}, ..., a_K^{(i)})\}_{i=1}^N$ be the dataset where $\mathbf{I}^{(i)} \in \mathbb{R}^{H \times W \times 3}$ is the i^{th} image, $y^{(i)} \in \mathbb{R}^M$ is the one-hot vector of the main task label of the i^{th} object, H is the height, W is the width of the image, and $a_j^{(i)} \in \mathbb{R}^{M_j}$ is the one-hot vector of the label of the j^{th} factor for the i^{th} object. Throughout this chapter, we are going to omit the example index i for simplicity and provide definitions for a single example.

3.3 The Backbone Network and the Branches

The backbone network $\mathcal{B}$ is any network (e.g. VGGNet [28], ResNet [29]) with which we can extract common features that will be used by the branches. Branch networks

also do not have to obey a specific design other than processing the common features separately. Through our experiments, we adapted the backbone network to the task; however, we are going to do the discussion over the ResNet50 [29] based backbone and branch networks. For the ResNet50 backbone and branch networks, we use the first 13 residual blocks of ResNet50 [29] architecture with its pretrained weights.

Let $\mathbf{\Omega}_{common} \in \mathbb{R}^{\frac{H}{16} \times \frac{W}{16} \times 1024}$ be the common feature extracted using the backbone network $\mathcal{B}$ from image $\mathbf{I}$. After the backbone network, the features propagate through K separate branches. Initially, each branch is allowed to select its related channels in $\mathbf{\Omega}_{common}$ through a 1×1 convolutional layer. With this, the number of channels is changed from 1024 to $\frac{KC}{2}$ where C will be the channel size of each branch output feature. The remaining 3 residual blocks of the ResNet50 [29] are re-implemented with appropriate channel sizes and group convolutions with K groups so that each factor features propagate in their respective branches. As they are re-implemented with different channel sizes, the parameters are randomly initialized. The adaptive average pool layer is kept at the end of the network.

Let $\mathcal{E}_j : \mathbb{R}^{\frac{H}{16} \times \frac{W}{16} \times 1024} \rightarrow \mathbb{R}^C$ be the j^{th} branch network that includes the 1×1 convolution layer, 3 residual blocks, and the adaptive average pooling layer. Extraction of the branch related outputs are depicted in Figure 3.2. The output of the j^{th} branch from the i^{th} image is:

$$\mathbf{\Omega}_j = \mathcal{E}_j(\mathcal{B}(\mathbf{I})). \tag{3.1}$$

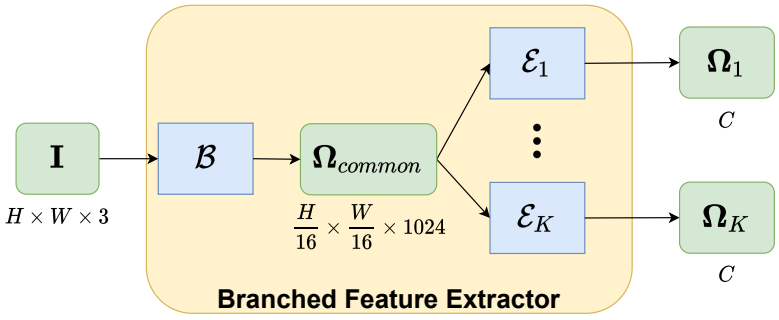


Figure 3.2: The backbone network and the branch networks. Image $\mathbf{I}$ is fed to the backbone network $\mathcal{B}$ to obtain a common feature $\mathbf{\Omega}_{common}$. From the common feature, each branch network extracts branch related outputs, $\mathbf{\Omega}_j$.

Immediately after the average pooling layer, we put a fully-connected layer for each

branch with an addition of Rectified Linear Unit (ReLU) activation function. Let $\mathbf{W}_j^{feat} \in \mathbb{R}^{C \times C}$ be the weight matrix and $\mathbf{b}_j^{feat}$ be the bias vector of the fully-connected layer of the j^{th} branch. The result of the fully connected layer is the representation of the j^{th} factor from the i^{th} image as follows:

$$\mathbf{\Phi}_j = \text{ReLU}(\mathbf{W}_j^{feat}\mathbf{\Omega}_j + \mathbf{b}_j^{feat}). \tag{3.2}$$

Following the work of Luo et al. [3], we use the Batch Normalization Neck (BNNeck) structure. This structure allows compatibility between the triplet loss and the classification loss by reorganizing the feature space accordingly. Therefore, immediately after the extracted features, we put batch normalization layers after every branch. Let $\mathcal{BN}_j$ be the batch normalization layer used for the j^{th} branch. Let $\mathbf{\Psi}_j = \mathcal{BN}_j(\mathbf{\Phi}_j)$ be the features after the batch normalization layer at j^{th} branch extracted from i^{th} image. The illustration of the branch feature extraction and BNNeck [3] can be seen in Figure 3.3.

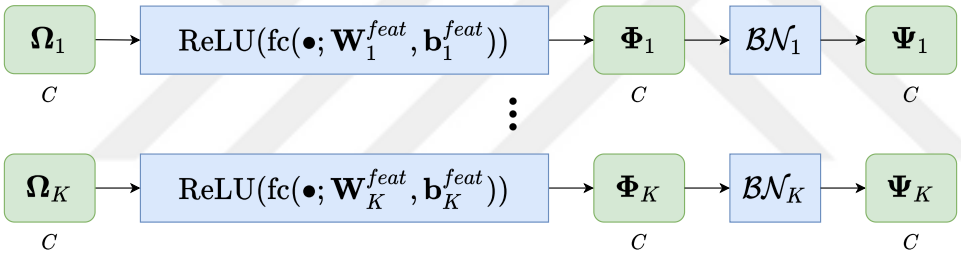


Figure 3.3: Extraction of branch features and BNNeck [3] structure. The output of each branch, $\mathbf{\Omega}_j$, is passed through a fully connected layer and a ReLU activation function to obtain the branch related feature, $\mathbf{\Phi}_j$. Batch normalization at each branch shown as $\mathcal{BN}_j$ implements the BNNeck structure [3]. The triplet loss is calculated from $\mathbf{\Phi}_j$'s and the cross-entropy loss is applied over features extracted from $\mathbf{\Psi}_j$'s.

3.4 Factor Label Prediction and Disentanglement

At each branch j, every factor label is predicted from $\mathbf{\Psi}_j$ using a fully-connected layer as the j^{th} estimated label is used as the guidance and the others are used for the adversarial loss. Let $\mathbf{W}_{j,k}^{factor} \in \mathbb{R}^{M_k \times C}$ be the weight matrix and $\mathbf{b}_{j,k}^{factor} \in \mathbb{R}^{M_k}$ be

the bias vector of this fully-connected layer where k is the index of the predicted factor. Then, we apply softmax and use cross-entropy loss to train these factor classifiers as follows (for $j = 1...K$, $k = 1...K$):

$$\mathcal{L}_{j,k}^{factor} = \text{CrossEntropy}(\hat{\mathbf{a}}_{j,k}, \mathbf{a}_k) \tag{3.3}$$

with $\hat{\mathbf{a}}_{j,k}$ defined as follows:

$$\hat{\mathbf{a}}_{j,k} = \text{softmax}(\mathbf{W}_{j,k}^{factor} \mathbf{\Psi}_j + \mathbf{b}_{j,k}^{factor}), \tag{3.4}$$

Following the work of Alvi et al. [2], we consider the prediction of the j^{th} factor label at branch j as its primary task and others as its secondary tasks. To get rid of the information unrelated to the branch, we apply confusion loss in an alternating training fashion which is discussed in detail in later sections. Similar to Robert et al. [18], we apply the additive inverse of the secondary task losses as the confusion losses as follows:

$$\mathcal{L}_{j,k}^{conf} = -\mathcal{L}_{j,k}^{factor}, \tag{3.5}$$

for all $j = 1...K$, $k = 1...K$ where $j \neq k$. The factor label prediction is illustrated at Figure 3.4.

3.5 Branch Uncertainty Estimation and Loss Attenuation

Let $\mathbf{W}_j^{unc-1} \in \mathbb{R}^{C \times C}$ and $\mathbf{b}_j^{unc-1} \in \mathbb{R}^C$ be the weight matrix and the bias vector of the first fully connected layer and let $\mathbf{W}_j^{unc-2} \in \mathbb{R}^{1 \times C}$ and $\mathbf{b}_j^{unc-2} \in \mathbb{R}$ be the weight matrix and the bias vector of the second fully connected layer to estimate the uncertainty values at branch j from $\mathbf{\Omega}_j$. Let $\sigma_j \in \mathbb{R}$ be the estimated uncertainty value for the branch j. In fact, the network actually estimates a value $s_j = \log \sigma_j^2$ as it is numerically more stable [13]. Inspired from Kendall et al. [14] and Cai et al. [19], we weight the factor primary classification loss of the j^{th} branch with $\frac{1}{\sigma_j^2}$ and we add a regularization term $\log \sigma$ to the loss function. Therefore, the loss functions for the primary factor classification tasks assigned to the branches become (see also Figure 3.5):

$$\begin{aligned} \mathcal{L}_{j,j}^{factor*} &= \exp(-s_j)\mathcal{L}_{j,j}^{factor} + \frac{s_j}{2}, \\ &= \frac{1}{\sigma_j^2}\mathcal{L}_{j,j}^{factor} + \log \sigma_j, \end{aligned} \tag{3.6}$$

for all $j = 1...K$. The loss attenuation process is illustrated in Figure 3.6.

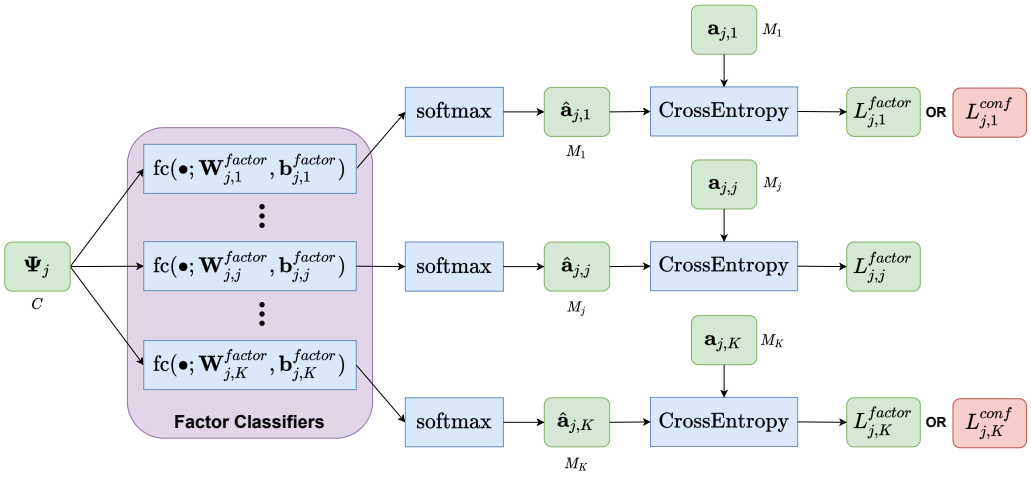


Figure 3.4: Factor label prediction process. Feature related to branch j, $\mathbf{\Psi}_j$, is fed into fully-connected layers for every factor to predict their labels. These networks are all trained using cross-entropy loss. Aside from the primarily assigned factor to this branch, which is the j^{th} one, confusion losses, $\mathcal{L}_{j,k}^{conf}$'s, are applied to the feature extraction networks to remove the information related to the other factors. Factor prediction loss is obtained.

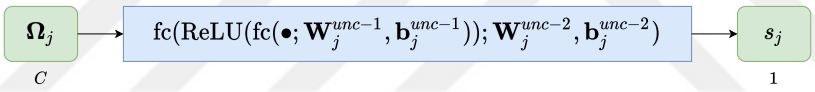


Figure 3.5: Uncertainty estimation network. Feature related to the branch j, $\mathbf{\Omega}_j$, is mapped to the uncertainty values with a 2-layer fully-connected network. Instead of directly estimating σ_j, we estimate $s_j = \log \sigma_j^2$ following Kendall et al. [14].

3.6 Main Task

As we have K branches for the K different factors, to address the main task, different kinds of approaches can be taken while creating a representation of the example. Before the triplet loss, we basically try two different approaches to obtain $\mathbf{\Phi}^{trip}$ to be used in the triplet loss calculation:

1. **TripCat**. In this approach, we directly concatenate the $\mathbf{\Phi}_j$'s from every branch to obtain a representation of size KC. This approach is illustrated at Figure 3.7a

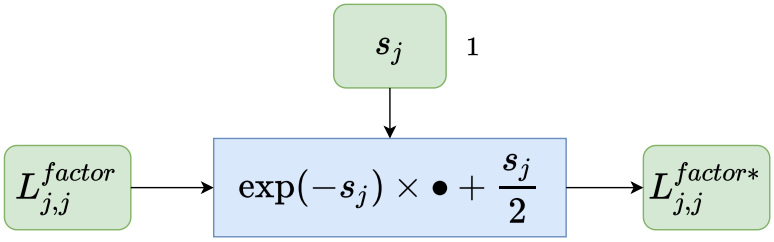


Figure 3.6: The loss attenuation applied over the branch related factor prediction loss. The estimated uncertainty values, $s_j = \log \sigma_j^2$, are applied to the loss itself to attenuate it with an additional penalty of $\frac{s_j}{2}$. The uncertainty estimated for the branch is only applied to the primary factor classification loss.

2. **TripMerge**. This approach integrates the estimated uncertainty values to blend features together. Specifically, the way we do that is similar to inverse-variance weighting in which we combine the features of each branch by inversely weighting them with their predicted variances. The variance of the combined result is obtained as:

$$\sigma^2 = (\sum_{j=1}^{K} \frac{1}{\sigma_j^2})^{-1}. \tag{3.7}$$

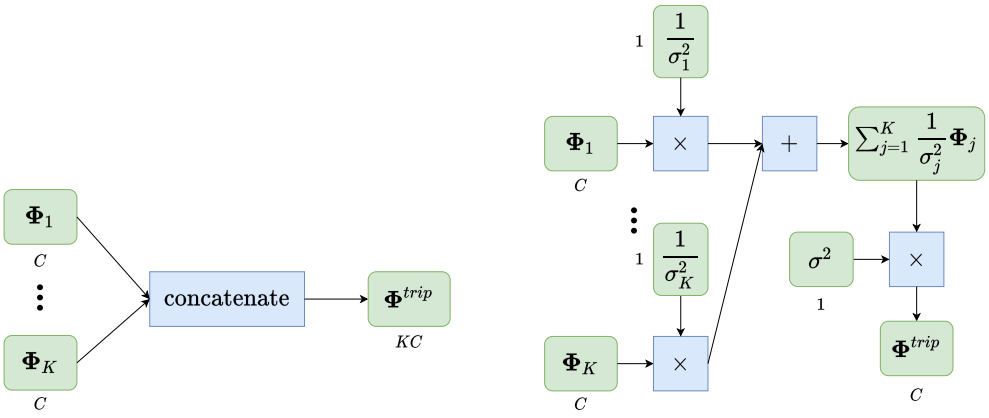
Then, after we sum these values up, we reweight them with the calculated overall variance to complete the weighted average calculation (see also Figure 3.7b):

$$\mathbf{\Phi}^{trip} = \sigma^2 \sum_{j=1}^{K} \frac{1}{\sigma_j^2} \mathbf{\Phi}_j. \tag{3.8}$$

After the calculation of features, $\mathbf{\Phi}^{trip}$, we use them in the triplet loss with hard positive/negative mining [30].

Similarly, there are 3 different approaches we take for applying the classification loss for the main task. The decision also depends on the approach taken for triplet loss since if the features, $\mathbf{\Phi}_j$'s, are already merged (the **TripMerge** approach), then, $\mathbf{\Phi}^{trip}$ can be used as input for the classification.

1. **ClsTripMerge**. This approach is the direct continuation of **TripMerge**. In this case, we need to re-define the BNNeck [3] since we only have one feature to consider. To do that, we define an additional batch normalization layer $\mathcal{BN}$.



(a) TripCat (b) TripMerge

Figure 3.7: Obtaining the triplet loss feature, $\mathbf{\Phi}^{trip}$. (a) All branch features before the BNNeck [3] are concatenated. (b) The branch features before the BNNeck [3] component are merged by taking weighted average using the inverse variances of the branches as weights.

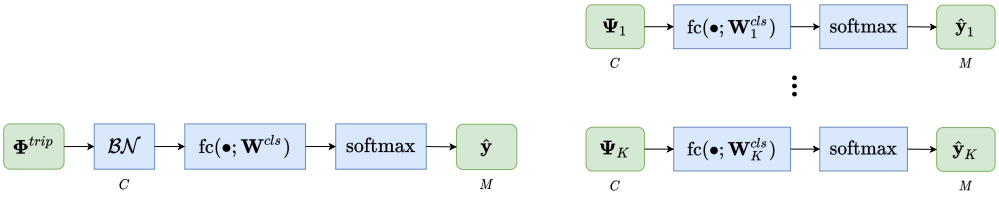
Then, the result of the merged features for the triplet loss, $\mathbf{\Phi}^{trip}$, is first passed through the batch normalization layer $\mathcal{BN}$ and a fully connected layer with weight matrix $\mathbf{W}^{cls} \in \mathbb{R}^{M \times C}$ and without bias afterward. This process is illustrated at Figure 3.8a. As a result, the class probability vector is obtained by

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{W}^{cls}\mathcal{BN}(\mathbf{\Phi}^{trip})). \tag{3.9}$$

2. **ClsSep**. In this approach, all branches try to do the classification task individually. To do that, we define K fully-connected layers without biases and with weight matrices $\mathbf{W}_j^{cls} \in \mathbb{R}^{M \times C}$. The architecture can be seen in Figure 3.8b. Then, the class probability vector for the j^{th} branch is calculated as a result of the matrix multiplication and a softmax operation:

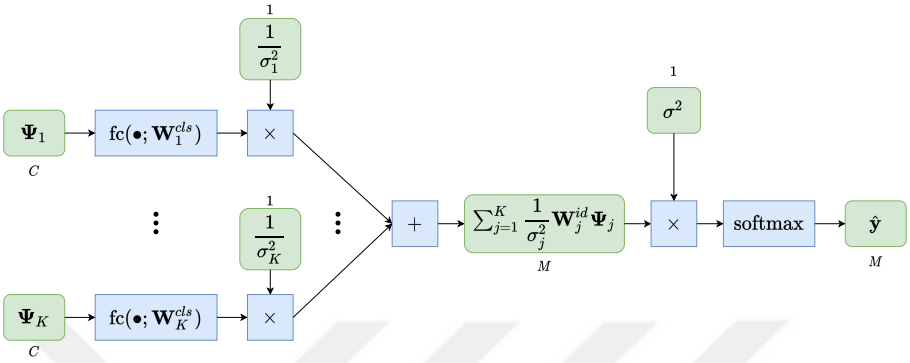
$$\hat{\mathbf{y}}_j = \text{softmax}(\mathbf{W}_j^{cls}\mathbf{\Psi}_j). \tag{3.10}$$

3. **ClsMerge**. This is a very similar approach to TripMerge; however, this one merges the branches at the logit level. To do that, we again define a fully-connected layer with weight matrices $\mathbf{W}_j^{cls} \in \mathbb{R}^{M \times C}$ and without biases. After obtaining the logit vectors by $\mathbf{W}_j^{cls}\mathbf{\Psi}_j$ for $j = 1...K$, we merge them by taking



(a) ClsTripMerge

(b) ClsSep



(c) ClsMerge

Figure 3.8: Approaches for the class probability vector calculation. (a) The merged features at triplet level, Φ^{trip}, is passed through the batch normalization layer, $\mathcal{BN}$, and a fully connected layer. (b) The branch features after the BNNeck [3], Ψ_j's, are passed through fully connected layers, individually. (c) The branch features after BNNeck [3] are merged by taking weighted average with multiplicative inverses of the branch variances.

the weighted average using inverse variances. The architecture is illustrated at Figure 3.8c. The resultant class probability vector is:

$$\hat{\mathbf{y}} = \text{softmax}(\sigma^2 \sum_{j=1}^{K} \frac{1}{\sigma_j^2} \mathbf{W}_j^{id} \mathbf{\Psi}_j). \tag{3.11}$$

For the case we obtain a single class probability vector, the classification loss is:

$$\mathcal{L}^{cls} = \text{CrossEntropy}(\hat{\mathbf{y}}, \mathbf{y}), \tag{3.12}$$

and for the case we obtain multiple class probability vectors, $\hat{\mathbf{y}}_j$'s, the classification loss is:

$$\mathcal{L}^{cls} = \sum_{j=1}^{K} \text{CrossEntropy}(\hat{\mathbf{y}}_j, \mathbf{y}). \tag{3.13}$$

3.7 Loss Attenuation at Main Task

As the branch features are used to solve the main task, their estimated uncertainties can also be used for the main task loss attenuation. Without this modification, the estimated uncertainties are related to the uncertainty of the factor-related classification tasks whose relations to the availability of the factors are investigated in this thesis. However, if we use the same uncertainty values for the main task, then, the estimated uncertainties should also account for the uncertainty coming from the main task. This effectively may change the estimated values; however, as the main task is one of the main objectives, this kind of attenuation can be favorable.

According to the approach taken while calculating the main-task-related losses, the application of loss attenuation changes. However, there are two losses that we can apply attenuation to: triplet and cross-entropy. If the cross-entropy loss of the main task is calculated separately at each branch; then, the attenuation can be applied to each branch using its own uncertainty estimation. In other approaches, since we obtain only one triplet and one cross-entropy loss, the overall variance σ^2, calculated from the branch variances can be used for both.

The attenuated triplet loss can be defined as follows:

$$\begin{aligned} \mathcal{L}^{trip*} &= \frac{1}{\sigma^2}\mathcal{L}^{trip}, \\ &= (\sum_{j=1}^{K} \frac{1}{\sigma_j^2})\mathcal{L}^{trip}, \end{aligned} \tag{3.14}$$

and the attenuated cross-entropy loss, if applied over classification loss directly, can be defined as:

$$\begin{aligned} \mathcal{L}^{cls*} &= \frac{1}{\sigma^2}\mathcal{L}^{cls}, \\ &= (\sum_{j=1}^{K} \frac{1}{\sigma_j^2})\mathcal{L}^{cls}, \end{aligned} \tag{3.15}$$

and the attenuated cross-entropy loss, if applied over individual branch cross-entropy losses for **ClsSep** can be:

$$\mathcal{L}^{cls*} = \sum_{j=1}^{K} \frac{1}{\sigma_j^2}\text{CrossEntropy}(\hat{\mathbf{y}}_j, \mathbf{y}). \tag{3.16}$$

3.8 Overall Losses and Alternating Training

For disentanglement not to affect the parameters of the secondary task components, $\mathbf{W}_{j,k}^{factor}$'s and $\mathbf{b}_{j,k}^{factor}$'s, while applying adversarial loss, they should be frozen. However, they should be trained to keep up with the changes in $\mathbf{\Psi}_j$'s. Therefore, we do our training in an alternating fashion.

1. The parameters of the secondary task fully-connected layers, $\mathbf{W}_{j,k}^{factor}$ and $\mathbf{b}_{j,k}^{factor}$ for $j = 1...K$, $k = 1...K$ where $j \neq k$, are frozen. The backbone $\mathcal{B}$, branch networks $\mathcal{E}_j$'s, parameters of the first fully-connected layer in the main task feature extraction $\mathbf{W}_j^{feat}$'s and $\mathbf{b}_j^{feat}$'s, batch normalization layers $\mathcal{BN}_j$, if available the batch normalization layer for **TripMerge** $\mathcal{BN}$, uncertainty estimation network parameters $\mathbf{W}_j^{unc-1}$, $\mathbf{b}_j^{unc-1}$ $\mathbf{W}_j^{unc-2}$ and $\mathbf{b}_j^{unc-2}$, parameters of the fully-connected layer in the main task classification $\mathbf{W}^{cls}$ or $\mathbf{W}_j^{cls}$'s, and parameters of the primary task factor classifiers $\mathbf{W}_{j,j}^{factor}$ and $\mathbf{b}_{j,j}^{factor}$ for $j = 1, ..K$ are trained. The applied loss is

$$\mathcal{L}^{total-1} = \sum_{j=1}^{K} \mathcal{L}_{j,j}^{factor*} + \mathcal{L}^{trip*} + \mathcal{L}^{cls*} - \alpha \frac{1}{K} \sum_{j=1}^{K} \sum_{\substack{k=1 \\ k \neq j}}^{K} \mathcal{L}_{j,k}^{factor} \tag{3.17}$$

   where $\alpha \in \mathbb{R}$ controls the strength of the disentanglement loss.

2. The calculated branch features after the BNNeck [3] $\mathbf{\Psi}_j$'s are detached so that their connection with the previous parts of the network are cut. Then, the secondary task full-connected layers, $\mathbf{W}_{j,k}^{factor}$ and $\mathbf{b}_{j,k}^{factor}$ for $j = 1...K$, $k = 1...K$ where $j \neq k$, are trained. The applied loss is

$$\mathcal{L}^{total-2} = \frac{1}{K} \sum_{j=1}^{K} \sum_{\substack{k=1 \\ k \neq j}}^{K} \mathcal{L}_{j,k}^{factor}. \tag{3.18}$$

CHAPTER 4

EXPERIMENTS

In this chapter, we evaluate our method on two datasets: A toy dataset and a person re-identification dataset.

4.1 Experiments with the Toy Dataset

The aim of this experiment is to create an optimal environment for our method. In order to do that we first generate a toy dataset and train a model to inspect the uncertainty values and disentanglement rates.

4.1.1 Three Pixels Dataset

An optimal setting for our method should have some independent generating factors which should also be reflected in the data itself. For simplicity, this dataset contains only three-pixel images. The first pixel can have a non-zero value only on its red channel. Similarly, the second pixel and the third pixel can have only green and blue values, respectively. In total, there are 3 pixels and 3 underlying independent stochastic procedures to give the pixels their values. The labels are determined according to the intensity values of each pixel. Additionally, we add noises to pixel values to introduce uncertainty near the decision boundary. The dataset generation procedure can be seen in Algorithm 1. From the algorithm, we obtain the labels as scalars; however, in the text, we also use the one hot version of those. $\mathbf{y}$ is the one hot version of y and $\mathbf{y}_k$ is the one hot version of y_k. Some samples from the dataset can be seen in Figure

4.1.

Algorithm 1: Toy dataset generation. $\mathcal{U}$ is the discrete uniform distribution, $\mathcal{N}$ is the normal distribution. $\mathbf{1}$ is the indicator function, y_k is the label for the k^{th} factor and y is the label of the overall example.

for *each image* $\mathbf{I} \in \mathbb{R}^{1\times 3}$ **do**
  $x_1, x_2, x_3 \sim \mathcal{U}(0, 255)$
  for *k in* $\{1, 2, 3\}$ **do**
    $y_k \leftarrow \mathbf{1}(x_k > 127)$
    $z \sim \mathcal{N}(0, 15)$
    $x_k \leftarrow \text{clip}(x_k + z, 0, 255)$
    $x_k \leftarrow \text{int}(x_k)$
  end
  $\mathbf{I}_{1,red} \leftarrow x_1$
  $\mathbf{I}_{2,green} \leftarrow x_2$
  $\mathbf{I}_{3,blue} \leftarrow x_3$
  $y \leftarrow 4y_1 + 2y_2 + y_3$
end

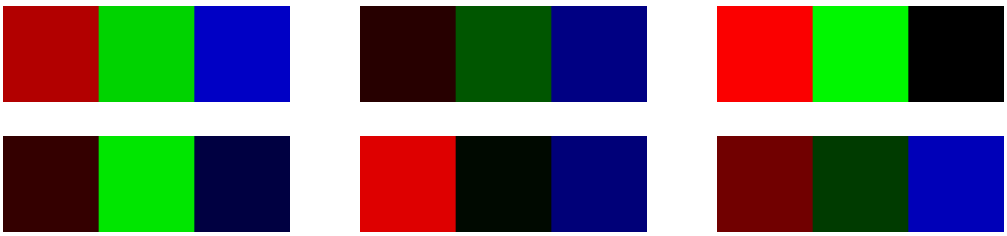


Figure 4.1: Some examples from the Three Pixel dataset.

This dataset aims to test the model on whether it can capture the introduced uncertainty near the decision boundary and whether uncertainty estimation at every branch responds only to its assigned factor. Notice that the place where high uncertainty is expected is introduced by adding noise on the color values of the pixels. This additional noise makes the task of each branch unreliable as the color values get closer to 127.

4.1.1.1 Data splits

There are 60000 images generated for training and 10000 images generated for testing. These 60000 images are further separated into training and hold-out validation set by sampling from Bernoulli distribution $\mathcal{B}(0.8)$ for every example and if the result is a success the example is assigned to the training set and otherwise, to the validation set.

4.1.2 The model

Since the data is not complicated and the underlying factors do not affect the resulting image in a complicated way, we simply used a 1-layer architecture for the common feature extraction and additional 3 layers for branch representations and uncertainty estimations. The detailed architecture can be seen in Figure 4.2.

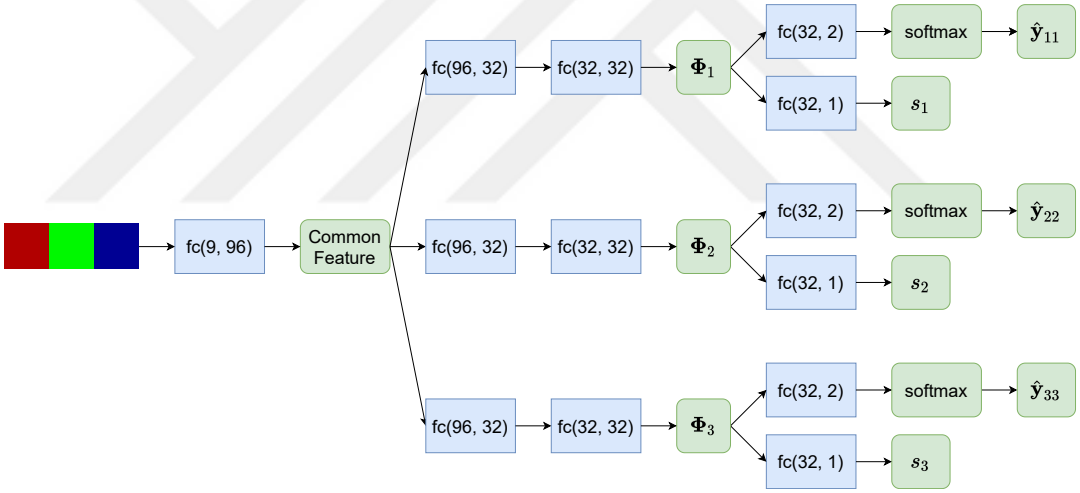


Figure 4.2: The architecture used on the Three Pixels dataset. Blue boxes are functions and green boxes are the values produced. Fully-connected layers that map $\mathbb{R}^a$ to $\mathbb{R}^b$ are shown with fc(a, b). Before every fully connected layer except for the first one, ReLU is used as an activation function. The branch features are shown with $\mathbf{\Phi}_j$'s. At every branch, model outputs $\hat{\mathbf{y}}_{jj} \in \mathbb{R}^2$ which is the estimated probability distribution for the j^{th} factor and $s_j \in \mathbb{R}$, the uncertainty estimation of that branch. Following Kendall and Gal [13], the model estimates $s_j = \log \sigma_j^2$ instead of the variance directly.

After the probability vectors at each branch are estimated, we apply cross-entropy

loss. The primary classification loss at each branch can be written as

$$\mathcal{L}_j = \text{CrossEntropy}(\hat{\mathbf{y}}_{jj}, \mathbf{y}_j). \tag{4.1}$$

After the integration of the loss attenuation, the primary classification losses become

$$\mathcal{L}_j^* = \exp(-s_j)\mathcal{L}_j + \frac{s_j}{2}. \tag{4.2}$$

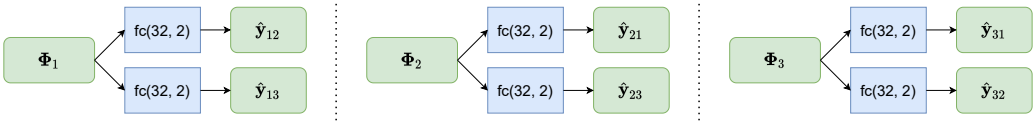


Figure 4.3: The secondary task components. Blue boxes are functions and green boxes are the values produced. Fully-connected layers that map $\mathbb{R}^a$ to $\mathbb{R}^b$ are shown with $\text{fc}(a, b)$.

For the disentanglement, we introduce additional linear layers that predict other branch factor labels from $\mathbf{\Phi}_j$'s. The secondary task components are illustrated in Figure 4.3. The losses to train the secondary task components are

$$\mathcal{L}_{jk} = \text{CrossEntropy}(\hat{\mathbf{y}}_{jk}, \mathbf{y}_k), \tag{4.3}$$

for $j \neq k$. The confusion losses to remove the unwanted information from the feature extractors can be written as are the additive inverses of the secondary task losses, $\mathcal{L}_{jk}^{conf} = -\mathcal{L}_{jk}$. As it is discussed in Chapter 3, the confusion losses and the primary task losses are applied together while the secondary task components are frozen and the secondary task components are trained separately.

To do the main task, branch features $\mathbf{\Phi}_j$'s are merged through weighted average to obtain the main task feature

$$\mathbf{\Phi}_{MT} = \frac{\sum_{j=1}^{3} \exp(-s)\mathbf{\Phi}_j}{\sum_{j=1}^{3} \exp(-s)}. \tag{4.4}$$

After obtaining the main task feature, it is passed through a fully-connected layer to map it to the logit space. Then, The obtained class scores are passed through softmax and cross-entropy loss is applied.

4.1.3 Training Details

The model is trained for 20 epochs using a batch size of 256. The training is done in an alternating fashion. Both of the optimizers use the learning rate of 10^{-4} with Adam with weight decay 0.01. The training of the secondary task components is done 3 times per iteration. After each epoch, the model is saved if the total validation loss is the least it has ever been.

4.1.4 Uncertainty Estimation Results

In this setting, there is no adversarial loss for disentanglement and no main task loss. Let's call this setting **BASE**.

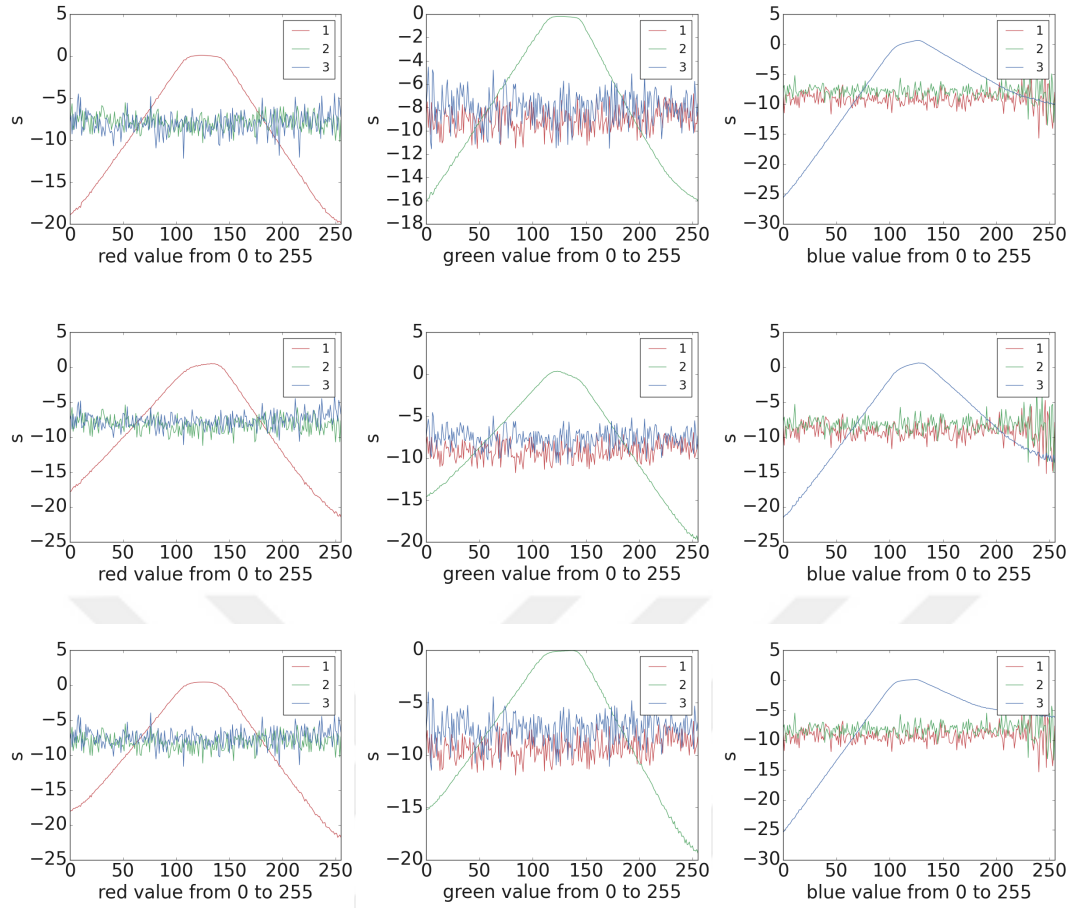


Figure 4.4: Uncertainty estimations for every color at every branch. Every row is the result of a different experimental setting. From top row to bottom row, the settings are **BASE**, **BASE + MT**, and **BASE + MT + DISENT**. At each row, every plot is specific to a color channel. The left-most plots are related to the red channel value in the first pixel, middle ones are related to the green channel value in the second pixel and the right-most ones are related to the blue value in the third pixel. At each plot, the horizontal axis shows the value of that specific color channel in the example. For example, horizontal axis having the value 7 at one of the left-most plots means that all of the examples included in the calculations have their red channel 7 at the first pixel. The vertical axis at each plot shows the average s_j values estimated by each branch. Branches are color coded to their related factors as also shown in the legends.

The aim of this experiment is to see whether heteroscedastic uncertainty estimations with loss attenuation correlate with the decision boundary. To do this, we plot the

uncertainty values. We first select a color for a specific plot. Then, for every value the color takes from 0 to 255, we average the estimated s_j values for each branch separately. The resulting plots for **BASE** setting are at the top row of Figure 4.4. As can be seen, at each plot, the estimated uncertainty values are higher as the values get closer to the decision boundary at their related branch. Furthermore, notice that the uncertainty estimations at unrelated branches remain oblivious to the value change even without the disentanglement since the estimated uncertainty values control the loss attenuation only at their respective branches.

4.1.5 Disentanglement experiments

To see whether each branch encodes features that are only related to its assigned factor, we train a linear model to estimate every factor class from each branch on the training dataset. Then, we measure the accuracy values on the test set for each model to obtain a table. We are going to call this table the disentanglement table in the rest of the discussion.

First, we obtain the disentanglement table for the **BASE** setting. The resulting table can be seen at Figure 4.5.

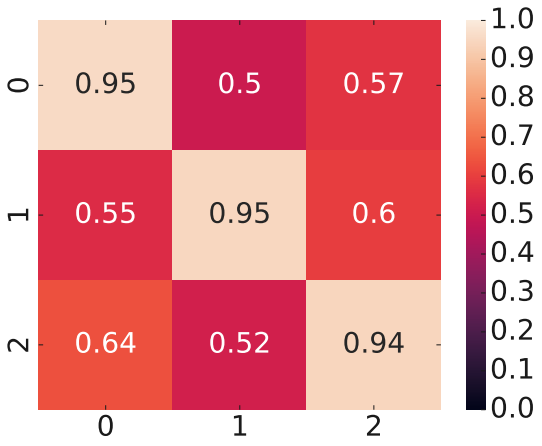


Figure 4.5: The disentanglement table for the **BASE** setting. The rows correspond to branches and columns correspond to factors. Scores are the accuracies calculated on the test set.

Interestingly, even without adversarial loss, since the factors are really independent in the data itself, the branches only encode information that is only related to their assigned factor.

Now, we are going to add the main task, which is the estimation of labels that can be derived from knowing the label of each factor. Let's call this setting **BASE + MT**.

The disentanglement table of the setting **BASE + MT** can be seen at Figure 4.6. Notice that different from the **BASE**, each branch tries to encode information related to every factor. Although we are collectively deciding the main task label, this result is expected since the losses that propagate through every branch have them encode the information related to the main task, which can be done by knowing information related to every factor. Nevertheless, the uncertainty estimations still respond only to the related factor of the branch as can be seen in the middle row of Figure 4.4.

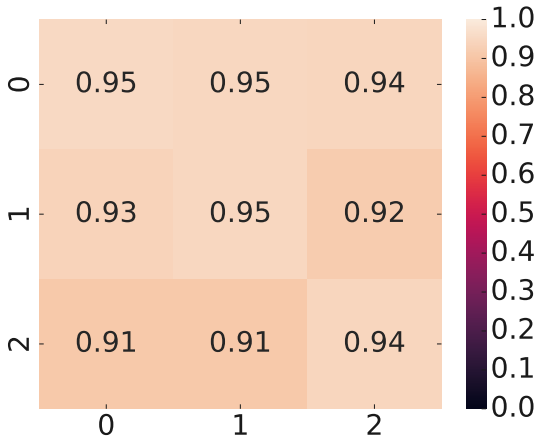


Figure 4.6: The disentanglement table for the **BASE + MT**. The rows correspond to branches and columns correspond to factors. Scores are the accuracies calculated on the test set.

To preserve the property of branches encoding features only related to their factor, we add adversarial loss over the branch features to remove unrelated factor information in them. Let's call this setting **BASE + MT + DISENT**. Notice that, as the disentanglement losses are added, the features forcefully encode the information only related to its assigned factor as can be seen in Figure 4.7.

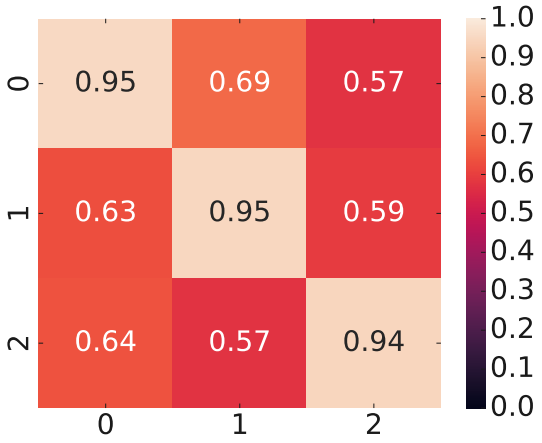


Figure 4.7: The disentanglement table for the **BASE + MT + DISENT**. The rows correspond to branches and columns correspond to factors. Scores are the accuracies calculated on the test set.

4.2 Experiments in Person Re-identification

4.2.1 Preprocessing of the images

Following the open-reid library [1], the images are first resized from 128x64 to 256x128. As mentioned in Luo et al. [3], the training images are flipped horizontally with probability 0.5, and padded 10 pixel with 0 and cropped into size 256x128 again. After these data augmentation techniques, the images are normalized using the mean and standard deviation values given in the PyTorch [31] ($mean = [0.485, 0.456, 0.406]$, $std = [0.229, 0.224, 0.225]$) so that the inputs are mapped to a similar distribution as their pretraining.

Additionally, following Zhong et al. [32] and Luo et al. [3], we applied random erasing to the training set which randomly erases a rectangle region in an image with a probability of 0.5. Since we apply this operation after the normalization, erasing means assigning the mean values inside the selected rectangle. The default parameters in PyTorch [31] implementation are used.

[1] `https://github.com/Cysu/open-reid`

4.2.2 The Model

The backbone model $\mathcal{B}$ is the first 13 residual blocks of ResNet50 [29] based architecture mention in Chapter 3. We selected $C = 328$ and $K = 12$ since there are 12 attributes in the Market-1501 Attribute dataset [33] and C should be divisible by K. $\mathcal{B}$ outputs common feature $\mathbf{\Omega}_{common}$ of the size (16, 8, 1024), where 1024 is the channel size, 16 is height and 8 is the width. After the common feature, the model branches and outputs features $\mathbf{\Omega}_j$'s of size $C = 328$.

4.2.3 Training details

Similar to Luo et al., we train the model for 200 epochs using the whole training dataset without separating a validation dataset. The final model after the last epoch is saved and used throughout the experiments.

Adam optimizer with learning rate 3.5×10^{-4} and weight decay 0.0005 is used for both parts of the alternating training. Following Luo et al. [3] warm up learning rate strategy is employed. For the first 10 epochs learning rate increases from 3.5×10^{-5} to 3.5×10^{-4} and at epoch 40 the learning rate is dropped to 3.5×10^{-5} and at epoch 60, it is further dropped to 3.5^{-6}.

Label smoothing [34] with $\epsilon = 0.1$ is applied for both factor and main classification tasks.

4.2.4 Results

4.2.4.1 The Effects of Loss Attenuation and Disentanglement

Before applying loss attenuation or disentanglement, we first fix the architecture to be **TripCat + ClsSep** since it is the most trivial one and perform an ablation study. The baseline, therefore, is the one in which we do not use any loss attenuation or disentanglement but still have branches. We also remove the Factor Supervision from it to prevent any attribute-related guidance. The test features that we use to compare two examples with cosine distance are obtained by concatenating the branch features,

Ψ_j's. We call this approach **TestCat**. The performance is measured directly on the main task, person re-identification.

The results can be seen in Table 4.1. In the first row, the results of the work of Luo et al. [3] without the Center Loss can be seen. After that, we introduce our branched baseline and add different techniques one by one. In general, we see that usage of loss attenuation only on Main Task Loss (Equation 3.15 or 3.16) decreases the performance. However, using loss attenuation on both factors (3.6) and the main task obtains the best performance in terms of mAP. For the disentanglement, we see both performance increases and decreases however, it affects the **TripCat + ClsSet Complete** positively in terms of mAP, and rank-1 accuracy. Notice that in terms of rank-5 and rank-10 accuracy, there is no significant increase or decrease in general. Therefore, from here on, we are going to use Factor Supervision, Factor Loss Attenuation, Main Task Loss Attenuation, and Disentanglement, which we call **Complete**, for other architectures as well.

Table 4.1: The results in person re-identification with several approaches using the architecture **TripCat + ClsSep**. The query and the gallery sets are not corrupted. During test time, **TestCat** is used. (FS: Factor Supervision, FLA: Factor Loss Attenuation, CLA: Main Task Classification Loss Attenuation, Dis: Disentanglement, CL: Center Loss).

Approach	FS	FLA	CLA	Dis	mAP	r1	r5	r10
BOT (w/o CL) [3]					0.857	0.941	-	-
TripCat+ClsSep Baseline					0.853	0.941	0.979	0.988
	✓				0.858	0.941	**0.982**	0.988
	✓			✓	0.862	0.941	**0.982**	0.988
	✓	✓			0.860	0.941	**0.982**	**0.989**
	✓	✓		✓	0.858	0.941	0.981	0.988
			✓		0.858	0.939	0.979	0.986
	✓		✓		0.852	0.939	0.979	0.988
	✓		✓	✓	0.852	0.938	0.976	0.986
	✓	✓	✓		0.862	0.944	0.981	0.987
TripCat+ClsSep Complete	✓	✓	✓	✓	**0.866**	**0.947**	0.980	0.988

4.2.4.2 Comparison with the state of the art

The performance of the current state of the art on the Market-1501 dataset [1] reported by Ye et al. [23] is presented in Table 4.2 along with the baseline we used. As can be seen, we obtained a small increase of the baseline; however, there is still a big gap with the state-of-the-art model. The scores can further increase when ranking optimization techniques are employed. However, throughout this thesis, we report the scores without ranking optimization.

Table 4.2: The comparison with the state of the art on Market-1501 dataset [1] as reported in Ye et al. [23]. The given result are calculated **without** re-ranking. (CL: Center Loss).

Approach	mAP	r1	r5	r10
BOT Strong Baseline (w/o CL) [3]	0.857	0.941	-	-
BOT Strong Baseline (w/o CL) [3]	0.859	0.945	-	-
TripCat+ClsSep Complete	0.866	0.947	0.980	0.988
VAL (Zhu et al. [35], SOTA)	**0.917**	**0.962**	**0.987**	-

4.2.4.3 The effects of normalization

By default, the experiments are done using the batch normalization layers in their evaluation mode. We also tried to use them in their training mode with several batch sizes. However, the performance degraded significantly. The mAP of **TripCat+ClsSep Complete** was 0.393 with batch size 32 and 0.793 with batch size 256.

We also tried layer normalization instead of batch normalization in BNNeck structure [3]. However, the mAP of **TripCat+ClsSep Complete** was 0.833 which means the results did not improve.

4.2.4.4 The effects of different architectures

In this section, we inspect the experimental results of several architectural designs in their **Complete** settings. Here we remind briefly the suggested architectures dis-

cussed in Section 3.6. Remember that these architectures can be obtained by changing how we calculate features for triplet and classification losses of the main task. If the triplet loss is obtained by the concatenation of Φ_j's, we obtain **TripCat** whereas if we take weighted average using the multiplicative inverse of the branch variances, then we have **TripMerge**. For the classification part, on the other hand, we proposed 3 ways. In the first one, **ClsSep**, every branch does the ID prediction on its own, separately. The second one, **ClsMerge**, is similar to the **TripMerge** as we take the weighted average of the logit level features before feeding it to the softmax. The last architecture of the classification part is a continuation of the **TripMerge** approach where we directly use the merged features during triplet and obtain logits directly from it. Therefore, this approach is called **ClsTripMerge** and it can be used only if **TripMerge** is employed.

In this section, we are going to directly concatenate the branch features as we did before (**TestCat**). Later, we are going to present and discuss other ways for testing to incorporate uncertainty estimations during test time. The results on person re-identification can be seen in Table 4.3. The best result is achieved by **TripCat + ClsSep Complete**. However, **TripMerge + ClsSep Complete** also obtain a close one. We can say that **ClsSep** performs better than **ClsMerge** when we concatenate the test features as in **TestCat**. Notice the low mAP score of the **TripMerge + ClsTripMerge Complete** architecture here. As this architecture merges the features for the main task during training time at Φ_j level by incorporating the uncertainty values and never use it separately afterward (Equations 3.8 and 3.9), different scales of the concatenated features may affect the overall decision more than or less than they should. Interestingly, this is not observed in other **TripMerge** architectures which may be because the ID-related classification features are calculated separately after merging the features for triplet loss.

4.2.4.5 Testing by Merging Features

Although some of the architectures are compatible with directly concatenating the test features, **TripMerge + ClsTripMerge Complete** could not obtain high scores like the others. To account for this problem and to incorporate the uncertainty estimations

Table 4.3: The results in person re-identification using different architectures. Factor Supervision, Factor Loss Attenuation, Main Task Classification Loss Attenuation, and Disentanglement are employed. The test features are calculated by simply concatenating the branch features (**TestCat**). The query and the gallery sets are not corrupted. (Trip: Triplet Loss Approach. Cls: Classification Loss Approach

Approach	Trip	Cls	mAP	r1	r5	r10
Complete	Cat	Sep	**0.866**	**0.947**	0.980	0.988
Complete	Cat	Merge	0.858	0.939	**0.982**	0.988
Complete	Merge	Sep	0.864	0.942	0.981	**0.989**
Complete	Merge	Merge	0.852	0.940	0.981	**0.989**
Complete	Merge	TripMerge	0.776	0.914	0.970	0.980

into the testing process, we propose to take the weighted average as we did during training time in some architectures. In more detail, the test feature in **TestMerge** can be calculated as follows:

$$\mathbf{\Psi}^{test} = \sum_{j=1}^{K} \frac{1}{\sigma_j^2} \mathbf{\Psi}_j. \tag{4.5}$$

The results can be seen in Table 4.4. A performance increase for **TripMerge + ClsTripMerge Complete** can be seen compared to the **TestCat** strategy. As we discussed in Section 4.2.4.4, this is expected since weighting is done at lower layers in **TripMerge + ClsTripMerge Complete**. On the other hand, this testing method resulted in a performance decrease in all other architectures. This kind of decrease may stem from the fact that merging branch features $\mathbf{\Phi}_j$'s has not been fully introduced during training which results in incompatibility issues between feature spaces. This compatibility problem may cause some important discriminative features to interfere with each other.

4.2.4.6 Testing by Weighting the Distances

As we hypothesize that features may be interfering with each other in **TestMerge**, we here propose another way to use uncertainty values without blending the separate feature spaces together. We are going to do this by first calculating the cosine distances

Table 4.4: The results in person re-identification using different architectures. Factor Supervision, Factor Loss Attenuation, Main Task Classification Loss Attenuation, and Disentanglement are employed. **TestMerge** is used. The query and the gallery sets are not corrupted. (Trip: Triplet Loss Approach. Cls: Classification Loss Approach)

Approach	Trip	Cls	mAP	r1	r5	r10
Complete	Cat	Sep	0.848	0.94	0.977	0.987
Complete	Cat	Merge	0.836	0.932	0.977	0.986
Complete	Merge	Sep	**0.855**	0.938	0.98	0.988
Complete	Merge	Merge	0.838	0.933	0.975	0.984
Complete	Merge	TripMerge	0.838	**0.942**	**0.982**	**0.989**

branch-wise between two examples and take the weighted average of these distances using the multiplicative inverses of the branch variances. As we have two weights per branch coming from two examples, we multiply them to obtain a final weight for a branch distance. Therefore, the distance between two examples a and b with features $\mathbf{\Psi}_j^a$'s and $\mathbf{\Psi}_j^b$'s are calculated as follows:

$$\mathcal{D}^{weight}(\mathbf{\Psi}^a, \mathbf{\Psi}^b) = \frac{\sum_{j=1}^{K} \frac{1}{(\sigma_j^a \sigma_j^b)^2} \mathcal{D}^{cos}(\mathbf{\Psi}_j^a, \mathbf{\Psi}_j^b)}{\sum_{j=1}^{K} \frac{1}{(\sigma_j^a \sigma_j^b)^2}}, \tag{4.6}$$

where $\mathcal{D}^{cos}$ is the cosine distance, σ_j^a and σ_j^b are the uncertainties of the j^{th} branches of examples a and b respectively.

We name this approach **TestDistWeight**. We see from the results in Table 4.5 that there is no significant difference in the kind of testing. We can observe a slight increase in **TripMerge + ClsTripMerge Complete** approach comparing it to the **TestCat** (0.784 mAP vs. 0.776 mAP); however, it is still way below the **TestMerge** version of it (0.784 mAP vs. 0.838 mAP).

4.2.4.7 Testing by Thresholding Uncertainties

As we could not obtain improvement by taking a weighted average of the distances, here, we apply a stronger constraint by putting a threshold on inverse branch vari-

Table 4.5: The results in person re-identification using different architectures. Factor Supervision, Factor Loss Attenuation, Main Task Classification Loss Attenuation, and Disentanglement are employed. **TestDistWeight** is used. The query and the gallery sets are not corrupted. (Trip: Triplet Loss Approach. Cls: Classification Loss Approach)

Approach	Trip	Cls	mAP	r1	r5	r10
Complete	Cat	Sep	**0.866**	**0.947**	0.98	0.989
Complete	Cat	Merge	0.856	0.937	**0.982**	0.988
Complete	Merge	Sep	0.864	0.943	0.981	**0.99**
Complete	Merge	Merge	0.851	0.942	0.98	0.989
Complete	Merge	TripMerge	0.784	0.92	0.973	0.982

ances, $\frac{1}{\sigma_j^s}$'s. To do that, we first calculate the $\frac{1}{\sigma_j^s}$'s for all examples in the training set. Then, we determine the threshold value for each branch that drops $\rho\%$ of the examples. For a branch feature with variance $\frac{1}{\sigma_j^2}$ to be involved in the overall feature calculation, $\frac{1}{\sigma_j^s} \geq \frac{1}{(\sigma_j^t)^2}$ where $(\sigma_j^t)^2$ is the variance of the $\frac{\rho N}{100}th$ training example in the sorted list of inverse branch variances.

While comparing two examples, if a branch's feature is dropped in one of them, then the distance between those branch features is not included in the distance calculation. The reasoning here is that if a part is missing in an image, then we do not compare that part. After dropping the branch features, we take the weighted average of the distances of the remaining ones, similar to **TestDistWeight**. Notice that during the comparison, we may be left with no remaining features. For those cases, we directly assign a very high distance to push them to the end of the ranking list as we cannot get a reliable result when we don't have any part to compare. We name this testing approach **TestThreshold**.

Setting ρ to higher values would make our filtering process stronger. We try $0.1, 0.3, 1, 3, 10$ for the ρ values and results can be seen in Table 4.6. We see that as we increase the threshold, more branches are dropped and this affects the performance severely. However, we could not obtain a performance gain in any threshold percentage, ρ.

Table 4.6: The results in person re-identification using different architectures with several ρ values. Factor Supervision, Factor Loss Attenuation, Main Task Classification Loss Attenuation, and Disentanglement are employed. **TestThreshold** is used. The query and the gallery sets are not corrupted. (Trip: Triplet Loss Approach. Cls: Classification Loss Approach)

Approach	Trip	Cls	ρ	mAP	r1	r5	r10
	Cat	Sep	0.1	**0.866**	**0.947**	0.980	0.989
	Cat	Merge	0.1	0.856	0.937	**0.981**	0.988
	Merge	Sep	0.1	0.864	0.943	**0.981**	**0.990**
	Merge	Merge	0.1	0.851	0.942	0.980	0.989
	Merge	TripMerge	0.1	0.784	0.920	0.973	0.982
	Cat	Sep	0.3	**0.866**	**0.947**	0.980	0.989
	Cat	Merge	0.3	0.856	0.937	**0.981**	0.988
	Merge	Sep	0.3	0.864	0.943	**0.981**	**0.990**
	Merge	Merge	0.3	0.851	0.942	0.979	0.989
	Merge	TripMerge	0.3	0.784	0.920	0.973	0.982
	Cat	Sep	1	**0.866**	**0.947**	0.980	0.989
	Cat	Merge	1	0.855	0.936	**0.981**	0.988
	Merge	Sep	1	0.864	0.943	**0.981**	0.989
	Merge	Merge	1	0.851	0.941	0.980	0.989
	Merge	TripMerge	1	0.784	0.919	0.973	0.982
	Cat	Sep	3	**0.866**	**0.947**	0.980	0.989
	Cat	Merge	3	0.850	0.928	0.979	0.986
	Merge	Sep	3	0.864	0.943	0.980	0.989
	Merge	Merge	3	0.837	0.917	0.976	0.984
	Merge	TripMerge	3	0.783	0.919	0.972	0.982
	Cat	Sep	10	0.856	0.940	0.972	0.980
	Cat	Merge	10	0.806	0.865	0.962	0.980
	Merge	Sep	10	0.852	0.934	0.969	0.979
	Merge	Merge	10	0.763	0.796	0.944	0.969
	Merge	TripMerge	10	0.717	0.772	0.954	0.974

4.2.4.8 Robustness to the Noise

Until now, we have used the original query and gallery images. To test the robustness the models may have gained during training, we corrupt the query and gallery sets with the same noise introduced during training as discussed in Section 4.2.1 such as random erasing. We have to use the same type of noise as we only estimate aleatoric uncertainty in our work. The main task performances with different testing options on the corrupted query and gallery sets can be seen in Table 4.7. Some of the architectures perform better than the baseline; however, none of them widens the gap between the mAP score of itself and the baseline. This means that aside from protecting the performance gain from the Loss Attenuation and Disentanglement, we could not obtain robustness to the noise by estimating and integrating uncertainty. This is discussed in more details in Section 4.3.

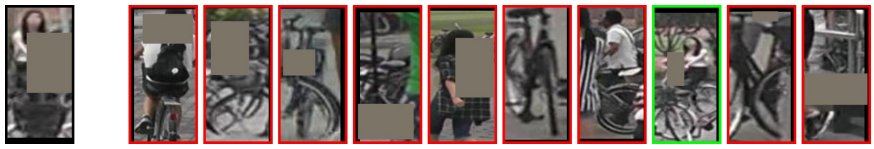
Table 4.7: The results in person re-identification using different architectures with corruption. Factor Supervision, Factor Loss Attenuation, Main Task Classification Loss Attenuation and Disentanglement are employed. For **TestThreshold**, $\rho = 10$ is selected.. **The query and the gallery sets are corrupted.** (Trip: Triplet Loss Approach. Cls: Classification Loss Approach)

Approach	Trip	Cls	Test	mAP	r1	r5	r10
Baseline	Cat	Sep	Cat	0.801	0.918	0.974	0.983
Complete	Cat	Sep	Cat	0.810	0.921	0.972	0.983
Complete	Cat	Merge	Cat	0.802	0.913	0.971	0.983
Complete	Merge	Sep	Cat	0.810	0.918	0.971	0.981
Complete	Merge	Merge	Cat	0.795	0.907	0.968	0.981
Complete	Merge	TripMerge	Cat	0.694	0.873	0.959	0.974
Complete	Cat	Sep	Merge	0.786	0.912	0.967	0.980
Complete	Cat	Merge	Merge	0.774	0.903	0.969	0.981
Complete	Merge	Sep	Merge	0.794	0.912	0.963	0.977
Complete	Merge	Merge	Merge	0.775	0.898	0.960	0.975
Complete	Merge	TripMerge	Merge	0.779	0.912	**0.977**	**0.988**
Complete	Cat	Sep	DistWeight	**0.811**	0.917	0.972	0.983
Complete	Cat	Merge	DistWeight	0.800	0.914	0.971	0.982
Complete	Merge	Sep	DistWeight	**0.811**	**0.922**	0.969	0.983
Complete	Merge	Merge	DistWeight	0.793	0.918	0.966	0.980
Complete	Merge	TripMerge	DistWeight	0.710	0.884	0.961	0.975
Complete	Cat	Sep	Threshold	0.711	0.849	0.890	0.896
Complete	Cat	Merge	Threshold	0.627	0.624	0.894	0.937
Complete	Merge	Sep	Threshold	0.703	0.837	0.880	0.889
Complete	Merge	Merge	Threshold	0.473	0.299	0.754	0.879
Complete	Merge	TripMerge	Threshold	0.261	0.163	0.515	0.696

We also visualize retrieval results of the baseline and **Complete** models in corrupted query and gallery set in Figures 4.8 and 4.9. In general we cannot observe signifi-

cant differences as discussed. One thing to note is that in Figure 4.8, **TripMerge + ClsTripMerge Complete** was able to retrieve the correct ID in its top two retrievals. On the other hand, it performed poorly in Figure 4.9 although the others' top ten results are all correct.

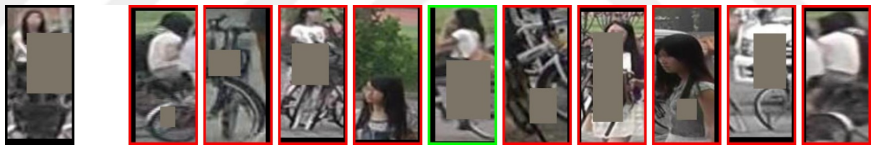
(a) **Baseline**



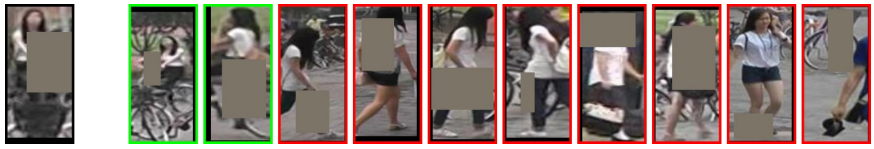
(b) **TripCat + ClsSep Complete**

(c) **TripCat + ClsMerge Complete**

(d) **TripMerge + ClsSep Complete**



(e) **TripMerge + ClsMerge Complete**



(f) **TripMerge + ClsTripMerge Complete**

Figure 4.8: Retrieval results on corrupted Market-1501 Dataset [1]. The leftmost image is the query and the other images are the most similar gallery images. The images with green frame are correct retrievals and the ones with red are incorrect.

(a) **Baseline**

(b) **TripCat + ClsSep Complete**

(c) **TripCat + ClsMerge Complete**

(d) **TripMerge + ClsSep Complete**

(e) **TripMerge + ClsMerge Complete**

(f) **TripMerge + ClsTripMerge Complete**

Figure 4.9: Retrieval results on corrupted Market-1501 Dataset [1]. The leftmost image is the query and the other images are the most similar gallery images. The images with green frame are correct retrievals and the ones with red are incorrect.

4.3 Discussion

Although we obtain a small performance increase by using Loss Attenuation and Disentanglement in person re-identification, we could not create a testing strategy robust to noise by incorporating the estimated uncertainty values. We think that this is because the obtained uncertainties do not correlate with the availability of their respective parts. To illustrate this, we conduct an experiment to understand where branch uncertainties are the most responsive.

We slide a 64×64 patch with the same color used in random erasing on resized (256×128) query images. At each step, the patch is shifted by 32 pixels. At every shift, we calculate the uncertainty at each branch and create a heatmap to understand where each branch is responsive to. Figure 4.10 shows the uncertainty heatmap for **TripCat + ClsSep Complete**. Notice that the results do not change as we go through the branches. Although some of the values change slightly, all of the branches respond to the same part. This is also similar for **TripMerge + ClsSep Complete** as can be seen in Figure 4.11. This may stem from the **ClsSep** architecture where each branch tries to estimate the ID only by itself. Therefore, it is reasonable for them to focus on the most distinctive ID-related parts instead of limiting themselves to their assigned factors. This also explains the unresponsiveness of these models to **TestDistWeight**.

If we inspect the architectures with **ClsMerge** or **ClsTripMerge**, on the other hand, we see small variations. Figure 4.12 and Figure 4.13 show the heatmaps for **TripCat + ClsMerge Complete** and **TripMerge + ClsMerge Complete**, respectively. For example, the responsiveness for the handbag location in both figures can be seen. This is also the case for **TripMerge + ClsTripMerge Complete** as can be seen in Figure 4.14. One of the reasons for the architectures where the features are merged before classification to have some variety unlike the others can be the fact that not every branch has to encode every ID-related feature since some other branch may cover for that. The low variety of the uncertainty responses of separate branch architectures and the large variety of the merged branch architectures can also be seen in Figure 4.15.

Not every example gives these kinds of sharp results. Sometimes we see that branch

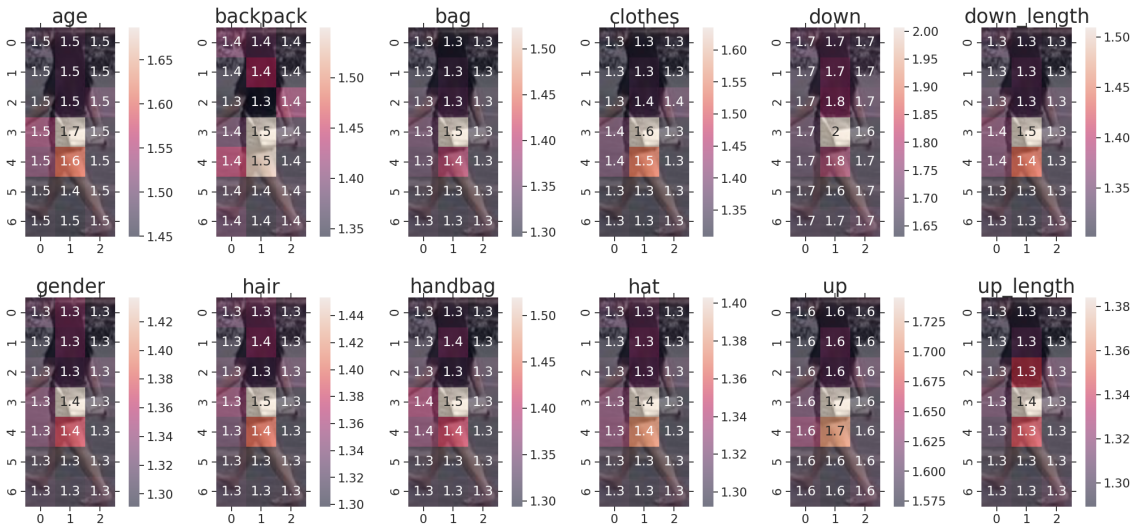


Figure 4.10: Uncertainty heatmaps of different branches of **TripCat + ClsSep Complete** obtained by shifting a patch over the image. The values are σ_j^2's estimated at different branches j.

uncertainties respond to unrelated parts as can be seen in Figure 4.17. We think that branches tend to focus on the ID-discriminative parts as they can guess their assigned attribute label from the ID itself. For example, suppose that we are guessing the hair length of a person. Since the hair of that person does not change throughout the images if the model can detect the ID of that person, it can predict all of the attributes even if the attribute-related part itself is occluded. Therefore, parts that are wide and discriminative like t-shirt and pants attract all branches. This damages the property of branches being responsive only to their attribute-related parts. For example, the branch related to the pants may increase its uncertainty value when the bottom part is occluded; however, it may give even higher uncertainty when the t-shirt of that person is occluded.

To investigate what branches encode, we created the disentanglement tables of each architecture as we did for the Toy Dataset. The tables can be seen in Figure 4.16. The results of these tables indicate how much we were able to disentangle the branch features and we see that in **ClsSep** architectures, the branches are still entangled. This also supports our hypothesis that the **ClsSep** architecture forces the branches to encode similar features. Additionally, we observe the best disentanglement in the **TripMerge+ClsTripMerge** architecture in which we merge the branch information

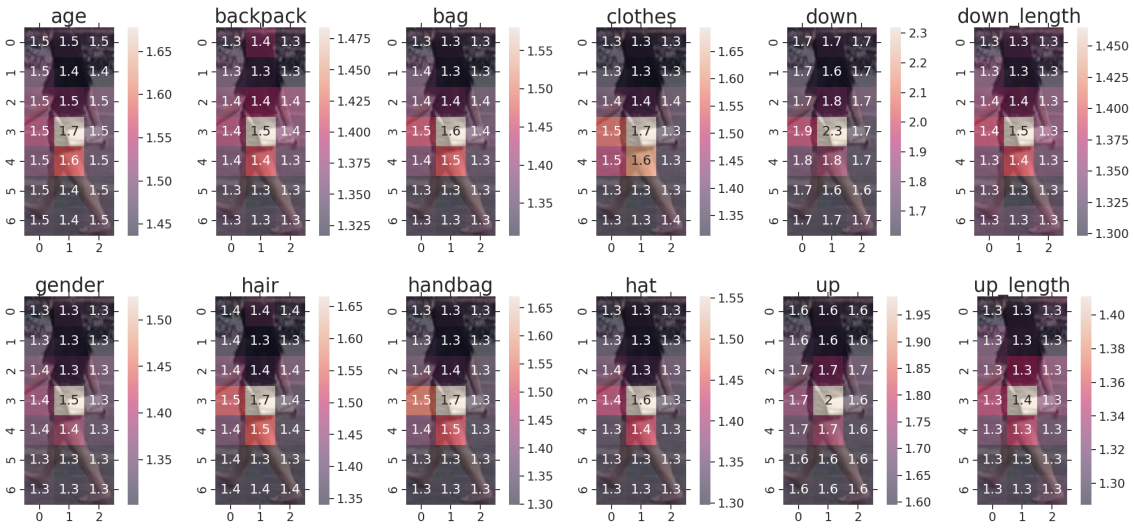


Figure 4.11: Uncertainty heatmaps of different branches of **TripMerge + ClsSep Complete** obtained by shifting a patch over the image. The values are σ_j^2's estimated at different branches j.

in the Triplet Space and directly calculate the feature for the classification from that.

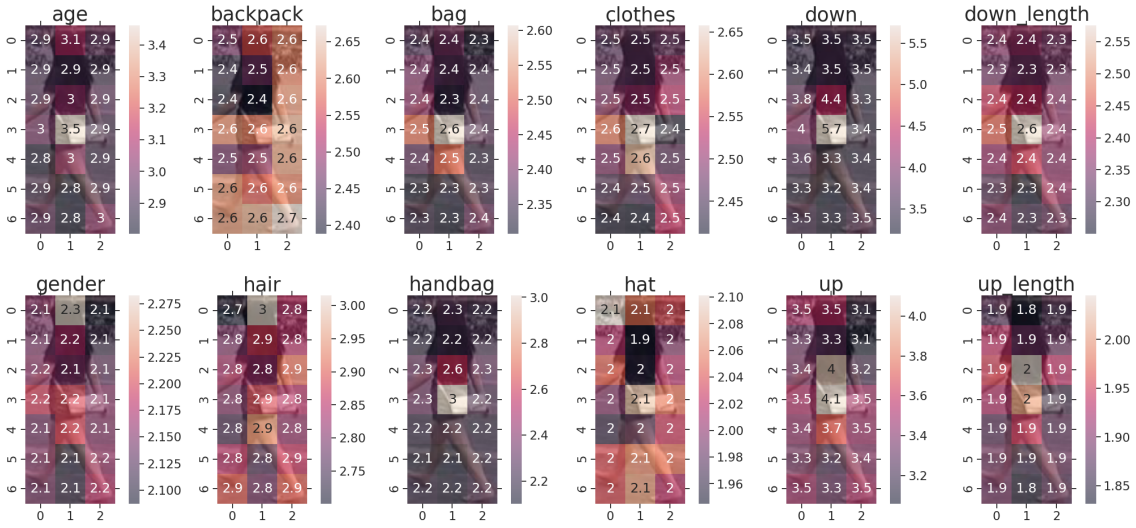


Figure 4.12: Uncertainty heatmaps of different branches of **TripCat + ClsMerge Complete** obtained by shifting a patch over the image. The values are σ_j^2's estimated at different branches j.

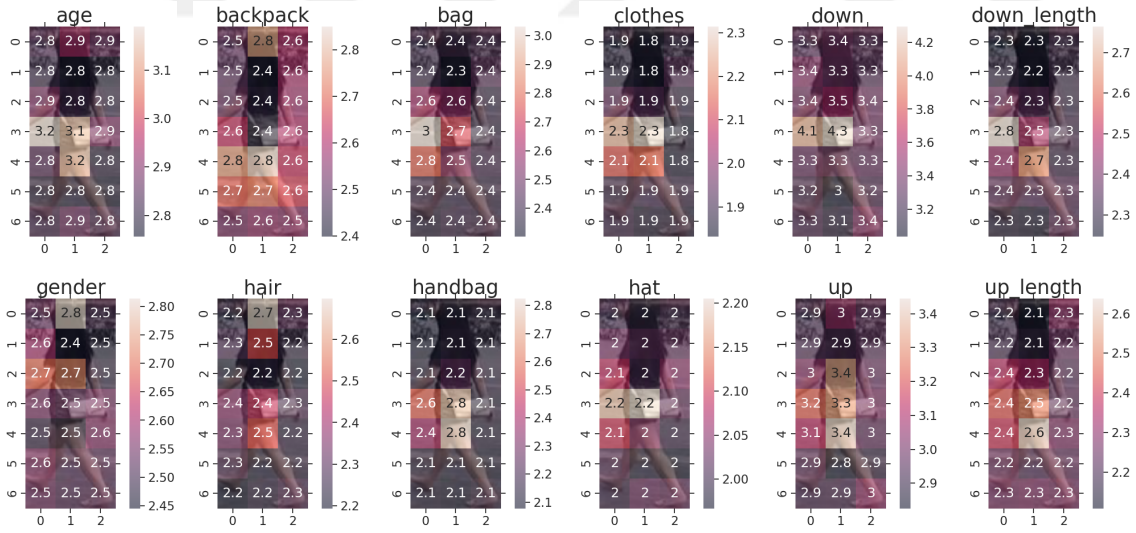


Figure 4.13: Uncertainty heatmaps of different branches of **TripMerge + ClsMerge Complete** obtained by shifting a patch over the image. The values are σ_j^2's estimated at different branches j.

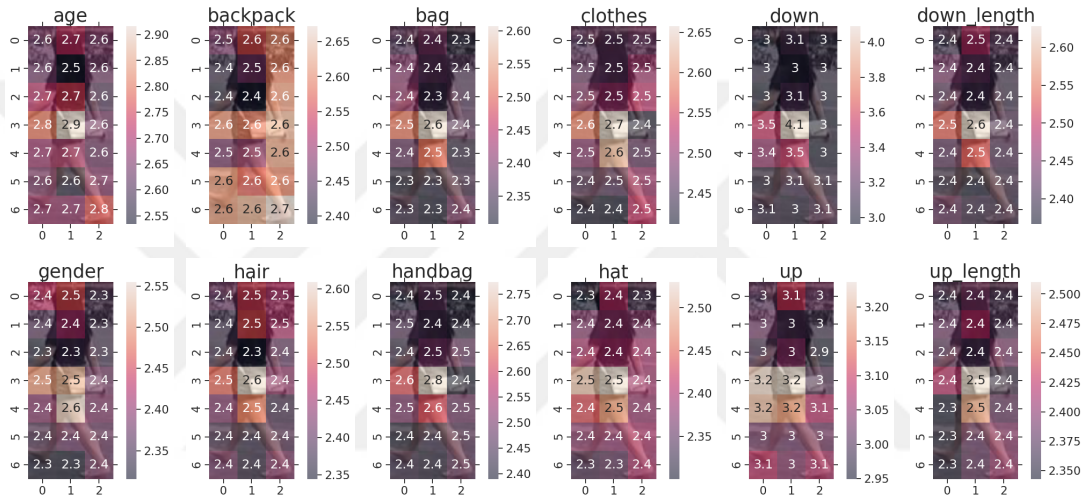
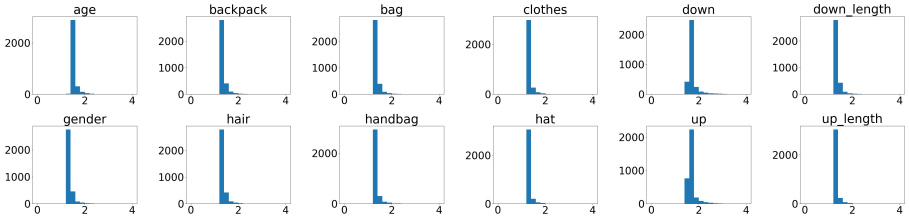
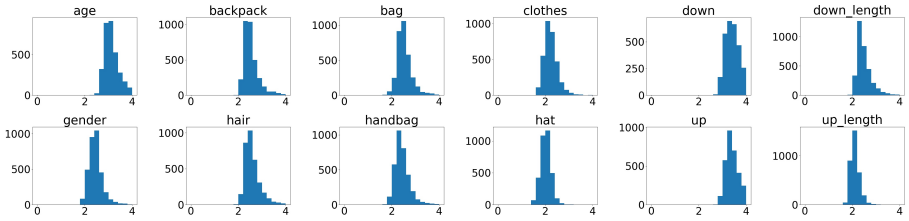


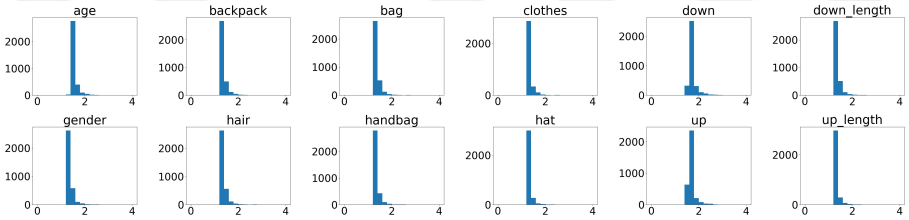
Figure 4.14: Uncertainty heatmaps of different branches of **TripMerge + ClsTripMerge Complete** obtained by shifting a patch over the image. The values are σ_j^2's estimated at different branches j.



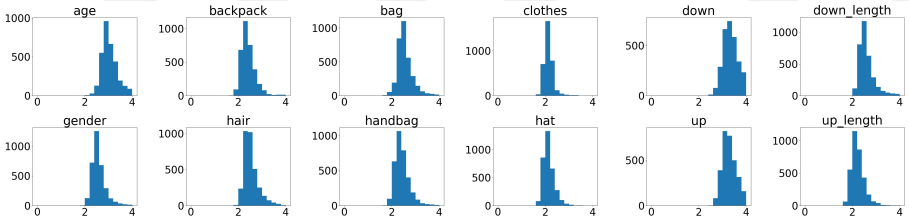
(a) **TripCat + ClsSep Complete**



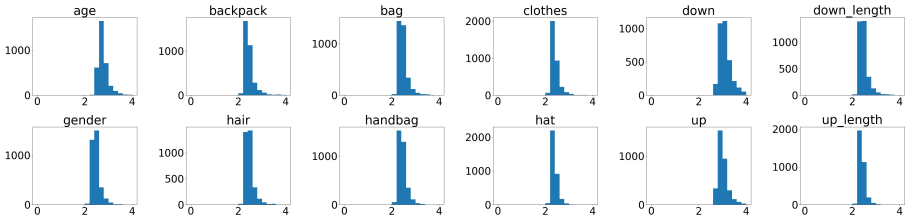
(b) **TripCat + ClsMerge Complete**



(c) **TripMerge + ClsSep Complete**

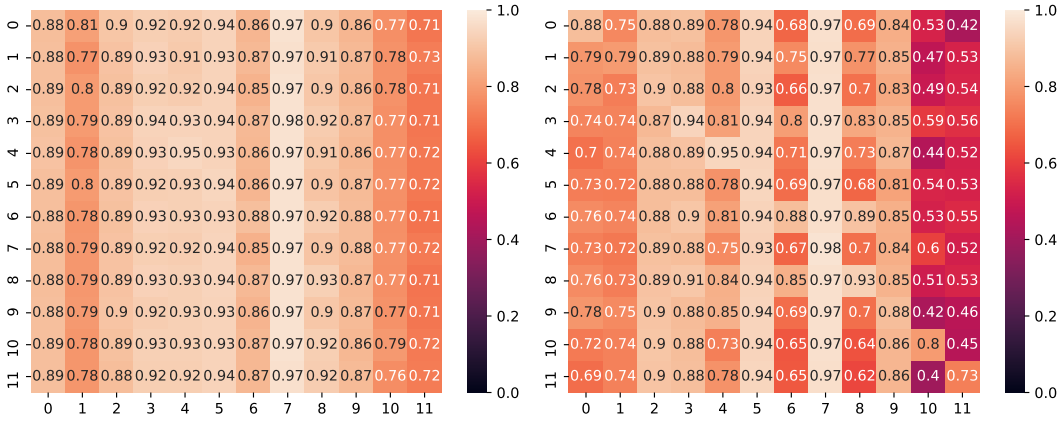


(d) **TripMerge + ClsMerge Complete**



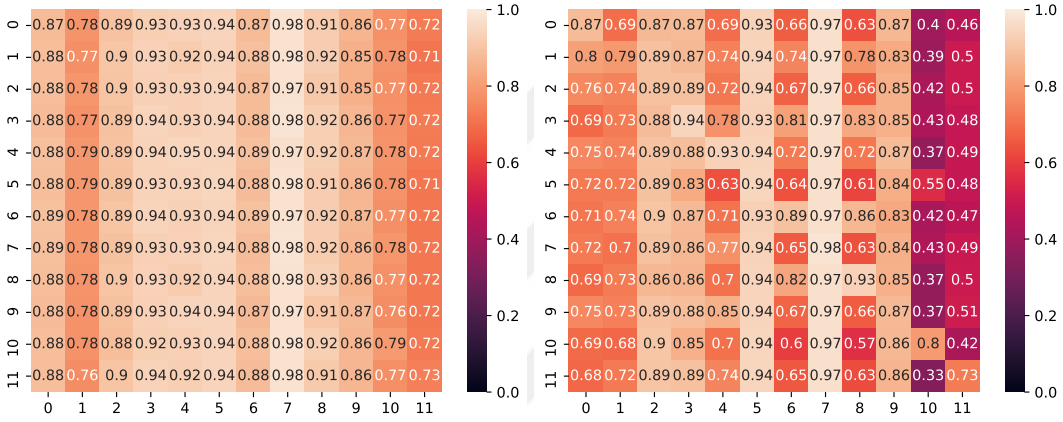
(e) **TripMerge + ClsTripMerge Complete**

Figure 4.15: Uncertainty histograms of different branches of different architectures. The horizontal axes show the σ_j^2's and vertical axes are the number of examples in that bin. The query set examples are used and they are corrupted using the data augmentation techniques during training. The range of the horizontal axis is the same for all figures.



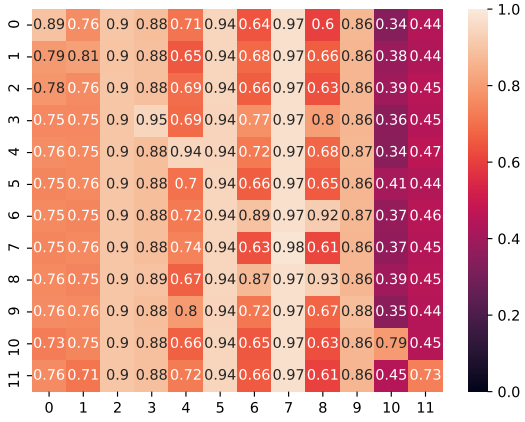
(a) **TripCat + ClsSep Complete**

(b) **TripCat + ClsMerge Complete**



(c) **TripMerge + ClsSep Complete**

(d) **TripMerge + ClsMerge Complete**



(e) **TripMerge + ClsTripMerge Complete**

Figure 4.16: Disentanglement tables for different architectures using both loss attenuation and disentanglement (Complete). The rows correspond to branches and columns correspond to attributes. Scores are the accuracies calculated on the query set.

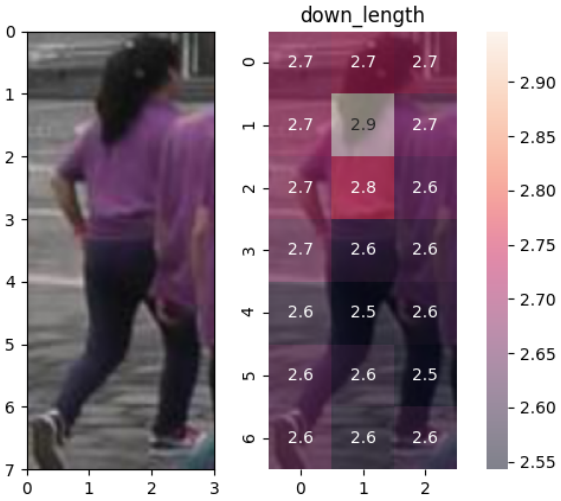


Figure 4.17: A failure example where the branch responsible for estimating the length of the bottom wear. Instead, the branch uncertainty responds to the upper parts.

CHAPTER 5

CONCLUSION

In this thesis, we worked on obtaining disentangled representations with also their uncertainty estimations. This way, we aimed to propose a solution to the problems like occlusion, viewpoint, and pose variations where different parts of an object are visible at different levels. To obtain separate representations, we used a branched architecture and applied adversarial loss to disentangle them. Meanwhile, we also experimented on how we could estimate the uncertainty for each branch representation, separately.

For the toy dataset experiments, we were able to estimate uncertainty separately for each factor successfully. One of the reasons was that the created dataset was an optimal setting for our method and the generative strategy was very simple.

In our real-life dataset experiments on person re-identification dataset, Market-1501 [1], with the attributes as the factor supervision, we obtained a small performance gain by using Loss Attenuation and Disentanglement. We employed different testing strategies to incorporate the estimated uncertainty values; however, we could not obtain a performance gain over the initial strategy which simply concatenates the branch features.

5.1 Limitations and Future Work

The proposed method is adaptable to different kinds of problems with different kinds of supervisions. In this thesis, we selected attribute supervision for person re-identification to guide and disentangle branches. However, as discussed in Section 4.3, this approach can be one of the main limitations of our experiments as we cannot guarantee

that branch features will only encode their attribute-related part information. Other than the attribute, different kinds of supervisions can be used. An example of that is using a pretrained pose estimator to estimate head, torso and leg locations and occluding those parts randomly. In that case, the supervised task for the branches can be the prediction of whether an occlusion has occurred for that specific part or not.

Another thing that may have increased the difficulty of training is that we have not filtered or grouped attributes according to their correlations or part correspondences. For example, carrying a handbag and a backpack can be negatively correlated. As we try to apply confusion loss to disentangle every attribute regardless of their relation, this may have cluttered the overall loss unnecessarily and created loss fluctuations as the task is not possible.

Employing an attention-based model for branching as in Zhao et al. [25] can help the networks to separate the representations spatially as the disentanglement can be applied only over the attention module. This property may address the problems such as occlusion in a more direct way as the representations would belong to separate regions.

We have only tried our method on person re-identification; however, it can be applied to different kinds of problems as well since disentangled representations can help in many computer vision problems where different parts of the objects are available to different degrees.

Our work only estimates aleatoric uncertainty which would only capture the noise introduced during training. It is possible to extend our architecture to work on different types of noises encountered during test time by estimating epistemic uncertainty. By capturing both uncertainty types, we believe that the architecture would be more robust to unexpected scenarios in real life.

REFERENCES

[1] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian, "Scalable person re-identification: A benchmark," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1116–1124, 2015.

[2] M. Alvi, A. Zisserman, and C. Nellåker, "Turning a blind eye: Explicit removal of biases and variation from deep neural network embeddings," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pp. 556–572, 2018.

[3] H. Luo, Y. Gu, X. Liao, S. Lai, and W. Jiang, "Bag of tricks and a strong baseline for deep person re-identification," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 1487–1495, 2019.

[4] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 35, no. 8, pp. 1798–1828, 2013.

[5] I. Biederman, "Recognition-by-components: a theory of human image understanding.," *Psychological review*, vol. 94, no. 2, pp. 115–147, 1987.

[6] Y. Zhao, X. Shen, Z. Jin, H. Lu, and X.-s. Hua, "Attribute-driven feature disentangling and temporal aggregation for video person re-identification," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4913–4922, 2019.

[7] S. Baabou, B. Mirmahboub, F. Bremond, M. A. Farah, and A. Kachouri, "Person re-identification using pose-driven body parts," in *International Conference on the Sciences of Electronics (SETIT), Technologies of Information and Telecommunications*, pp. 303–310, Springer, 2018.

[8] W. Li, X. Zhu, and S. Gong, "Harmonious attention network for person re-identification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2285–2294, 2018.

[9] D. Li, X. Chen, Z. Zhang, and K. Huang, "Learning deep context-aware features over body and latent parts for person re-identification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 384–393, 2017.

[10] J. Li, S. Zhang, Q. Tian, M. Wang, and W. Gao, "Pose-guided representation learning for person re-identification," *IEEE transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2019.

[11] J. Guo, Y. Yuan, L. Huang, C. Zhang, J.-G. Yao, and K. Han, "Beyond human parts: Dual part-aligned representations for person re-identification," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3642–3651, 2019.

[12] J. Miao, Y. Wu, P. Liu, Y. Ding, and Y. Yang, "Pose-guided feature alignment for occluded person re-identification," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 542–551, 2019.

[13] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?," in *Advances in Neural Information Processing Systems (NeurIPS)* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[14] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7482–7491, 2018.

[15] T. Yu, D. Li, Y. Yang, T. M. Hospedales, and T. Xiang, "Robust person re-identification by modelling feature uncertainty," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 552–561, 2019.

[16] Y. Liu, F. Wei, J. Shao, L. Sheng, J. Yan, and X. Wang, "Exploring disentangled feature representation beyond face identification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2080–2089, 2018.

[17] A. Jaiswal, R. Y. Wu, W. Abd-Almageed, and P. Natarajan, "Unsupervised adversarial invariance," in *Advances in Neural Information Processing Systems (NeurIPS)* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

[18] T. Robert, N. Thome, and M. Cord, "Dualdis: Dual-branch disentangling with adversarial learning," *arXiv preprint arXiv:1906.00804*, 2019.

[19] Q. Cai, Y. Pan, Y. Wang, J. Liu, T. Yao, and T. Mei, "Learning a unified sample weighting network for object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14173–14182, 2020.

[20] I. Higgins, D. Amos, D. Pfau, S. Racanière, L. Matthey, D. J. Rezende, and A. Lerchner, "Towards a definition of disentangled representations," *arXiv preprint arXiv:1812.02230*, 2018.

[21] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," in *5th International Conference on Learning Representations (ICLR), Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.

[22] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," in *Advances in Neural Information Processing Systems (NeurIPS)* (D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, eds.), vol. 29, pp. 2180–2188, Curran Associates, Inc., 2016.

[23] M. Ye, J. Shen, G. Lin, T. Xiang, L. Shao, and S. C. Hoi, "Deep learning for person re-identification: A survey and outlook," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2021.

[24] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," in *Advances in Neural Information Processing Systems (NeurIPS)* (C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds.), vol. 28, pp. 2017–2025, Curran Associates, Inc., 2015.

[25] L. Zhao, X. Li, Y. Zhuang, and J. Wang, "Deeply-learned part-aligned representations for person re-identification," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 3219–3228, 2017.

[26] L. Zheng, L. Shen, L. Tian, S. Wang, J. Bu, and Q. Tian, "Person re-identification meets image search," *arXiv preprint arXiv:1502.02171*, 2015.

[27] M. Kana, "Uncertainty in deep learning. how to measure?." https://towardsdatascience.com/my-deep-learning-model-says-sorry-i-dont-know-the-answer-that-s-absolutely-ok-50ffa562cb0b, May 2020.

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations (ICLR), San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[30] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," *arXiv preprint arXiv:1703.07737*, 2017.

[31] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NeurIPS 2017 Autodiff Workshop*, 2017.

[32] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 13001–13008, 2020.

[33] Y. Lin, L. Zheng, Z. Zheng, Y. Wu, Z. Hu, C. Yan, and Y. Yang, "Improving person re-identification by attribute and identity learning," *Pattern Recognition*, 2019.

[34] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, 2016.

[35] Z. Zhu, X. Jiang, F. Zheng, X. Guo, F. Huang, X. Sun, and W. Zheng, "Viewpoint-aware loss with angular regularization for person re-identification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 13114–13121, 2020.