Bruna Is All You Need!

1. Introduction

In this paper we are going to analyze two different tasks on a video-survelliance dataset containing images of multiple people, each of which is captured multiple times by different cameras along with a set of annotations that specify attributes of each person such as age, gender and clothing. The first task consists in building a multi-class classifier to predict such attributes for each image. In the second task, a solution to a person re-identification problem is provided, where a query image of a person is given and all the images of the same person should be retrieved from a collection of images.

2. Data Exploration

We start by exploring the images in the dataset (notebook here Open in Colab), to understand the distribution of the classes and the quality of the images. After having obtained some insights we thought a way to split the training set in a way that preserved the attribute distribution in both the training and validation sets. This split assigns all images of the same person to a specific set, in this way, the model is forced to generalize if we want to obtain satisfying results in the validation set. The assignment of a *person_id* to the training or validation set is partially randomized, but with some deterministic choices to assure that the split is equal (eg. if we have 200 people with a hat and a 80% training 20% validation split, we wish to have 160 people with a hat in the training set and the other 40 in the validation set). An implementation can be found here: Open in Colab.

3. Proposed Solution

3.1. Classification Task

Task2: (train & inference) Open in Colab
This task consists in training a neural network to perform different classifications, using the Market-1510 dataset. In particular, the predictions to be made are:

- gender
- hair length
- sleeve length
- length of lower-body clothing
- type of lower-body clothing
- wearing hat
- carrying backpack
- carrying bag
- carrying handbag
- age
- color of upper-body clothing
- color of lower-body clothing

For this task we exploit different approaches in order to understand the task, the behavior of each neural network and to find the final best approach. We start with simple handcrafted CNNs in order to understand the complexity of the task. Then we move to more complex models such as AlexNet and ResNets, we test ResNet-18, 32, 50, 101 and 152 (usually ResNets are a typical backbone for person re-identification NN, as stated in [6]). We notice that ResNet-18 is enough to solve the task (all the others tend to learn more slowly and are more prone to overfit). During the experiments we experiment with the techniques presented during the course, such as dropout, batch-normalization, different optimizers and different architectures. Below we explain the final approach we developed.

3.1.1 Architecture

We choose ResNet-18, which is illustrated in the figure 1, as backbone for our architecture. We decide to use the pre-trained model on ImageNet provided in PyTorch, and use the technique of fine-tuning. We re-trained the first convolutional layer and the last two layers with a high learning rate, while a lower is used for the other layers since they are already trained.

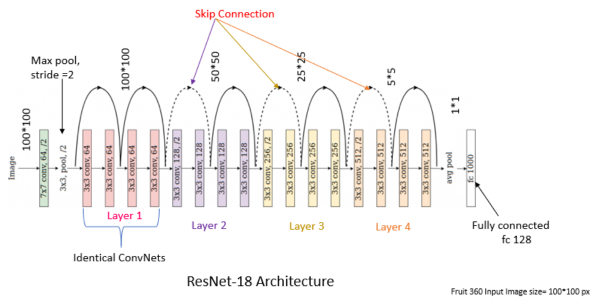

Figure 1. ResNet-18 Architecture.
Image source: www.pluralsight.com

3.1.2 Data Loader

Data Augmentation During the experiments we noticed that the models tend to overfit easily due to their complexity (simpler models underfit). In order to overcome this problem we decide to run some data augmentation in order to increase the number of images and the generalization of the models. We apply some transformations to the dataset:

- A *pad* of 4 is added to each image, then a crop of dimension 128x64 is done.
- A random `ColorJitter` is applied, with brightness=0.2, contrast=0.2, saturation=0.2, hue=0.01

- A `RandomRotation` is applied, with a maximum of 4 degrees.
- A `RandomEqualize(0.01)` is done
- A `RandomHorizontalFlip()` is applied.
- The image is converted into a tensor.
- A normalization is done, with the value provided by PyTorch[1] `mean=[0.485, 0.456, 0.406],std=[0.229, 0.224, 0.225]`.

Target transformation The age is encoded in 4 different categories: 1, 2, 3, 4; for this reason we decide to convert this encoding into a one-hot encoding, the idea is to have one output neuron for each age value, this, in our opinion, should make easier the learning for the network.
In addition, we add 3 new classes:

- Camera-4
- Camera-6
- Junk

The first two indicate the images taken from camera number 4 and 6. We added the camera classes as an experiment to see if the network was capable of learning the background of the images (this may help generalization by limiting the simplicity bias). The idea behind the junk class is to have a network capable of identifying junk images, in this way we would be able to output that the image is not containing people. We experiment with this idea for the second task, to not output junk images as match of one person. In order to implement this, during training, we randomly zoom in on some part of the training images assigning the new zoomed image to the junk class. We notice that this technique is quite robust and, in the end, the network is able to identify junk images. We use this class in the second task, during inference each retrieved image is filtered and discarded if predicted as junk.

Class Balance Since we saw that some classes were particularly unbalanced, we performed some adjustments. The general rule to get an item is:

```
def __getitem__(self, idx):
    rnd = torch.rand(1)
    if  rnd > 0.02:
        return self.get_person(idx)
    elif rnd > 0.01:
        return self.get_noise(idx)
    else:
        return self.get_weak(idx)
```

The function `get_weak()` takes items from low probability classes. More specifically, if a class contains less than 30 people, it is added to the ”weak” dataset.

[1] https://pytorch.org/hub/pytorch_vision_resnet/

3.1.3 Losses

Since we use a single NN to predict different targets, which have both binary and multi-class format, we decide to split the losses for those two types.

Binary class loss We implment a custom version of sigmoid loss which was proposed Goodfellow et al. [2] and presented by Ines Pedro in Understanding the Motivation of Sigmoid Output Units [1] . The aim of this particular loss is to avoid the problem of the vanishing gradient when using the sigmoid activation.

Multi-class loss For the multi-class loss, we decide to use the `CrossEntropyLoss`, in two different ways. One without weights, and one with weights in order to manage possible problems due to unbalanced classes. For the weighted loss, we create a tensor of 15 elements, that indicates the weights each class has then we have initialized it to $\frac{1}{15}$. At each epoch, we analyze the training confusion matrix and compute the performances. The classes that have the lowest results will receive a bigger weight at the next iteration. More specifically, $W[i] \rightarrow 1.1 \cdot W[i]$.

3.1.4 Optimizer

We test the main optimizers seen during the course, finally we choose `Adam`. Since we use a pre-trained network, we do not want to retrain all the network at the same manner, so we divide the learning rate of the network. For the layer to fine-tune (i.e. the first convolutional layer and the last 2 layers), the learning rate is $lr = 0.01$, while for the others is equal to $lr^* = \frac{lr}{10} = 0.001$. We set also a weight decay $= 10^{-7}$.

3.2. Re-identification task

Task2: (train) Open in Colab (inference) Open in Colab
The objective of this task is to identify a specific queried person inside a dataset, we use the standard techniques in the literature, where we compare the features extracted using a pretrained CNN and, if the features of two images are close between each other, we can say that they belong to the same person.

3.2.1 Architecture

Our architecture is composed by two networks: Bruna (Based on Residual hUge Nn Architecture), which is the main NN (similar to the discriminator), and an adversarial network which is used to select samples that can trick Bruna, this competition can improve the quality of the training samples feeded to Bruna, thus, leading to better results. The Bruna’s architecture is built by two principal modules:

the feature extractor, which takes as input an image and outputs a vector containing 1600 features and the task1 classifier, which takes as input the features and outputs the attributes for the first task. The features extractor is built upon a pretrained mobilenet_v3 [3], while the task1 classifier is a MLP network. Once we have trained the network (explained below), we can compute the distance between *image1* and *image2* as follows:

$$distance(i_1, i_2) =$$

$$= d(ft(i_1), ft(1_2)) \cdot (0.1 + d(att(i_1), att(1_2)))$$

where d is the euclidean distance, $ft(i)$ is a function that returns the output of the feature extractor and att are the attributes of a person (eg. gender, age, ...), which are computed as $att(i) = task1(ft(i))$

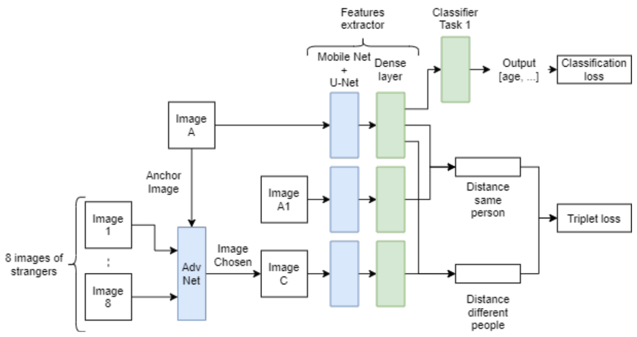

Figure 2. Architecture task 2. There are some simplifications to facilitate the reading on both task 1 and task 2, for example the output from the adversarial network is one image, instead in the final implementation the output is the two most similar images wrt. the anchor image, also Bruna's architecture is simplified, below the final training procedure is explained.

3.2.2 Training

We now explain one training iteration using a batch size of 1. Our custom dataset returns 12 images where the first 4 represent the same person (that we will call Luigi) and the other 8 represent other people. We feed the first photo of Luigi and the other 8 people to the adversarial network which will return the predicted distances between the photo of Luigi and each of the eight photos. Now we select the two images with the smallest predicted distance and we feed them with the 4 images of Luigi to Bruna. Bruna will now compute the centroids of each combination of three images of Luigi and the distance from the forth (referred to as distance_same_person or distance12 in the code), and the distances between each Luigi's image to the other 2 stranger images (referred to as distance_diff_person or distance13 in the code). Finally we take the mean of the various measurements and return them. We also compute the task1 by passing all the 12 images through the network and then use the loss described in the task1.

3.2.3 Losses

Bruna is trained in a multi-objective setting, where its loss function is a weighted sum of other two losses: the first one is the classification loss (given an image find the attributes as in the 1st task) and the second one is a slight variation of the triplet loss.

$$L_{triplet} = d\_same\_person + [P - d\_differ\_people]_+$$

where *d_same_person* is the distance from the a person from its centroid (the centroid is computed as the average of the features extracted from other photo of that person) and *d_differ_people* is the distance between the image of the person and another person (the other person is selected by the adversarial network), the idea of the distance from the centroids was taken by Wieczorek et al. [5]. *d_same_person* is outside the ReLU because we want to minimize the distance between the image and its centroid in every case. Figure 3 shows the idea behind the centroids.

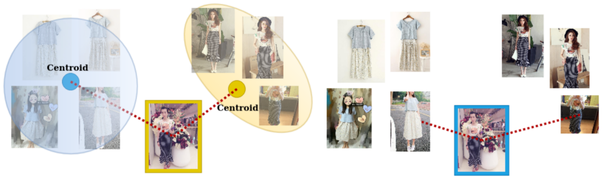

Figure 3. Instead of computing the distance between images, we compute the distance between an image and the centroid of its class. (Image taken from Wieczorek et al.)

3.2.4 U-Net

During our experiments we think about segmenting the people, this, based on our opinion, should lead the network to focus on the person in foreground increasing the overall accuracy. This idea has some assumptions which have to be made, first the segmentation network has to be able to segment well the person and second the person extracted has to be the interested one. We train U-Net [4] using a dataset found on www.kaggle.com. This network has been used on the second task in two settings. We try to use directly the segmentation of the image as input for Bruna. The second setting is the usage of the U-Net mask output as an additional channel for each image (increasing the number of features) and concatenating some features extracted from the bottleneck layer to the final embeddings. Both these techniques improved the performance of Bruna leading the mAP score to 0.59 using the second setting.

3.2.5 Optimizer

We use a default Adam optimizer for the adversarial network; for Bruna we use an Adam optimizer initialized with learning rate = 1e-4 for the final layers, while for the pre-trained MobileNetv2 backbone we use a learning rate of 1e-5. Moreover every 30 epochs, we halved the learning rate of every parameter, after some testing we found out that halving the LR reduces the performances.

4. Weights and Biases

Nowadays the understanding of neural networks has become a challenge, in order to overcome this issue some software can be used to study the behavior of the models. In our case we use weight and biases which allows us to track the learning and testing of the networks. We upload information about losses, accuracies, confusion matrices and images. These information are divided in training and validation in order to study the behavior of the models on these datasets. The images uploaded are based on the confidence of the model, we upload images with high and low confidence to gain insights about the behavior of the networks, to understand if the learning is proceeding as expected. This method helped us in finding bugs and improving the system.

5. Results and conclusions

5.1. Task1

In the first task we tried different architectures and approaches. We start with simple CNNs (3-4 layers) to understand the behavior of simple models and their capabilities and also to practice, then we move to more complex architectures such as LeNet5 and ResNet. LeNet5 is able to learn quickly until it asymptotes to an average accuracy of 0.76 (after 20 epochs), then we test all the versions of ResNet, increasing complexity showed a slow learning and a overfit pronunciation, for these reasons we decide to use ResNet-18 pre-trained on ImageNet as described above. The network was trained for only 3 epochs, since we saw that the validation loss was beginning to grow after them. The reason could be that for fine-tune a part of the network, only few epochs are needed. This setting leads the following results:

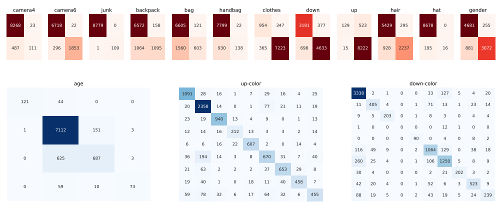

Figure 4. Training confusion matrix

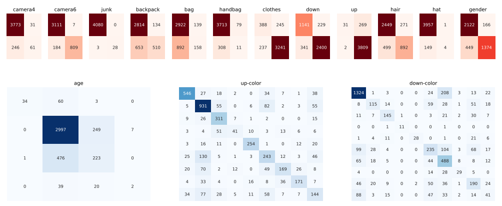

Figure 5. Validation confusion matrix

	TRAIN			VALIDATION		
	Acc	**Prec**	**Rec**	**Acc**	**Prec**	**Rec**
Camera4	94.1%	88.5%	57.2%	93.1%	76.7%	64.1%
Camera6	96.4%	97.2%	92.9%	96.6%	97.4%	93.4%
Junk	100.0%	100.0%	98.3%	100.0%	100.0%	100.0%
Backpack	85.7%	85.9%	74.2%	80.9%	78.3%	69.1%
Bag	81.0%	81.8%	62.8%	74.4%	65.8%	54.2%
Handbag	89.1%	87.8%	55.2%	91.7%	53.9%	50.3%
Clothes	91.4%	82.2%	82.7%	85.1%	71.9%	72.5%
Down	88.8%	88.1%	89.1%	84.7%	82.7%	84.3%
Up	94.2%	92.5%	59.9%	93.3%	95.1%	55.3%
Hair	86.0%	87.0%	82.2%	81.6%	80.3%	78.0%
Hat	98.0%	99.0%	55.6%	96.3%	98.1%	50.3%
Gender	86.6%	87.9%	85.5%	84.0%	84.2%	83.4%
Up-color	84.5%	84.0%	81.4%	68.5%	66.8%	62.2%
Down-color	82.6%	84.8%	74.3%	65.7%	67.8%	53.8%
Age	90.6%	92.3%	68.0%	80.0%	66.7%	40.0%
Overall	**89.9%**	**89.3%**	**74.6%**	**85.1%**	**79.0%**	**67.4%**

Table 1. Performance on the first task

5.2. Task2

In the second task our system reached a mAP of 0.59 in the validation set. During training both Bruna and the adversarial network improve, in Figure 7 we can notice a validation output during trainig. The network is capable of retrieving good matchings, however we have noticed that, in the current setting, the colors of clothing are able to trick the network, so this aspect should be taken in account for further work and improvement.
As we can see in 8 the mAP of the models seems to converge after 40 epochs. The value of different_person_distance is tied to the hyper-parameter P of the triplet loss. We finally can see that the adversarial loss decreases, this can be explained by the loss we adopt:

$$L_{adv} =$$

$$= (1 - correct) \cdot pred\_distance - correct \cdot pred\_distance$$

in practice: if Bruna correctly classified the samples we want to increase the predicted distance (punishment), while if bruna was wrong we want to decrease the predicted distance (reward). Bruna is almost always correct, so, the adversarial network predicts very big distances to minimize the loss (thus, generating a very negative loss). We surely need to improve this aspect, but the adversarial network seem to find decent hard examples even now 6.

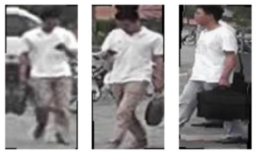
Figure 6. On the left: 2 images of Luigi. On the right: the example selected by the adversarial network

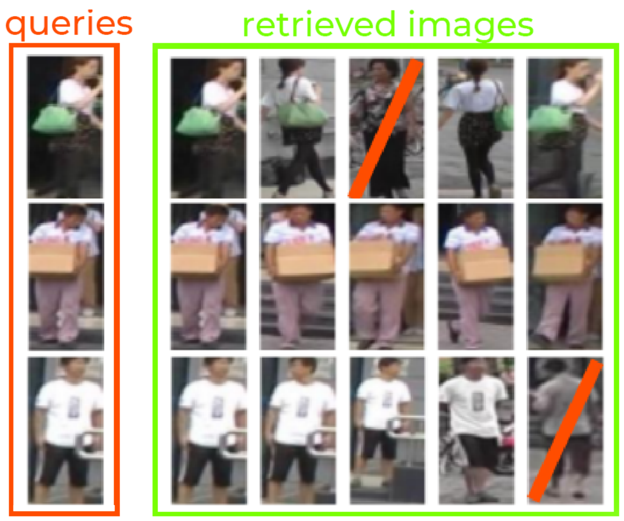

Figure 7. Some queries on the left and the retrieved images on the right. The red bar identify the wrongly returned images.

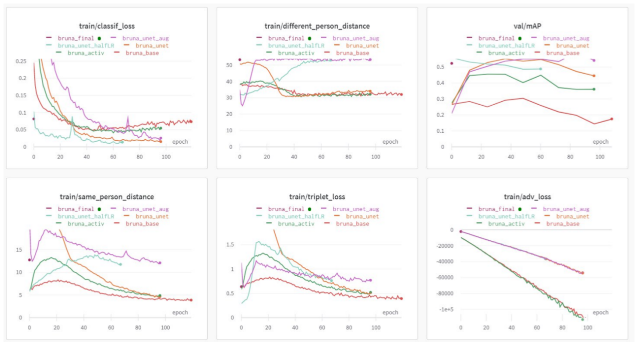

Figure 8. Losses of different tested models.

References

[1] Understanding the motivation of sigmoid output units. https://towardsdatascience.com/understanding-the-motivation-of-sigmoid-output-units-e2c560d4b2c4. 2

[2] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. http://www.deeplearningbook.org. 2

[3] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019. 3

[4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 3

[5] Mikolaj Wieczorek, Barbara Rychalska, and Jacek Dabrowski. On the unreasonable effectiveness of centroids in image retrieval. *arXiv preprint arXiv:2104.13643*, 2021. 3

[6] Mang Ye, Jianbing Shen, Gaojie Lin, Tao Xiang, Ling Shao, and Steven CH Hoi. Deep learning for person re-identification: A survey and outlook. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. 1