Università degli Studi di Milano-Bicocca

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di Laurea Magistrale in Data Science

Computer Vision for Fashion: a Deep Learning Approach for Image Generation

Relatore: Prof. Simone Bianco

Co-relatore: Prof. Raimondo Schettini

Tesi di Laurea Magistrale di:

Giulia Chiaretti

Matricola 800928

Anno Accademico 2019-2020

Summary

Fashion industry, one of the world's largest industries, is facing a constantly increasing problem: with new technologies and social media, fashion trends from all over the world constantly influence customers taste, which is becoming always more ever-changing. Fashion brands, with their stylists and designers, need to continue to produce successful items by creating always new, trendy and up-to-date items to satisfy the taste of their customers. Computer vision, generative models and deep learning applied in fashion domain can become good allies to stylists and designers creativity giving them a new source of inspiration. Indeed, the aim of this work is to develop a system able to create new and realistic fashion items starting from existing clothes images. Thus, two different tasks are developed: a preliminary one of fashion item detection to crop images and make them more suitable for the second task of image generation. To carry out these two tasks, state-of-the art models are used: Faster R-CNN object detector model, trained on DeepFashion dataset, and a custom DCGAN that generates new items starting from images of fashion brands' runways collected ad hoc by scraping the social network Pinterest.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Fashion industry has become one of the world's largest industries, generating $ 2.5 trillion in global annual revenues before the pandemic[1]. Fashion has thus attracted much attention from computer vision and Artificial Intelligence researchers in recent years. Multiple studies have been done about different topics related to fashion: from detecting fashion items, analyzing and synthesizing them, and finally providing personalized recommendations.

Moreover, fashion industry is facing a constantly increasing issue: customer taste is becoming ever more erratic and ever-changing. Fashion consumers, especially the young generation's ones, are always affected by multiple fashion visual inputs and influenced by fashion trends from all over the world.

This leads stylists and fashion designers to create always new, trendy and up-to-date items to satisfy their customers. Designers are

[1] https://www.mckinsey.com/industries/retail/our-insights/state-of-fashion

thus always looking for new source of inspirations to produce nice-looking new patterns and unique and wonderful fashion items that, especially in haute couture, look ever more like piece-of-art than simple clothes. This is also crucial for fashion brands to continue to produce successful items in order to keep generating and increasing revenue.

In this scenario computer vision can be a powerful ally to stylists and designers. Current state-of-the-art deep learning models are able to help and improve their creativity providing them with a new source of inspiration.

This works aims to generate new fashion clothing images following current trends of luxury fashion industries. The best way to achieve this goal is to collect images of existing and current clothes from luxury fashion brand and merge their styles and patterns to create new items. This is the purpose of Generative Adversarial Networks (GAN) [16] developed in this project.

More specifically, this work consists in two parts. A preliminary task on images is developed: fashion item detection and image cropping. In this way just suitable images are used for the second task of image generation through GAN.

To carry out these two tasks, state-of-the art models are used, which are introduced in chapter 2: the Faster R-CNN [42] is trained to accurately detect clothes within images; after the model is constructed, it's used to make inference on other fashion images and

predict bounding boxes to crop these images.

The predicted cropped images containing only the fashion item are used for the second task: these images are used to feed a custom Deep Convolutional Generative Adversarial Network (DCGAN) which generates similar fashion items.

For the first task, the DeepFashion[2] [32] dataset has been used. It was developed by the Chinese University of Hong Kong and contains thousands of images of people wearing different categories of clothes with various backgrounds. For the second task, instead, an ad hoc dataset was used created by scraping the social network Pinterest with a particular focus on images of several luxury fashion brands. More about the data used in the experiments are given in the third chapter.

In the fourth chapter the methodology used is illustrated, more specifically the different architectures of the Faster R-CNN implemented for the object detection task, the hyperparameter configuration and the training settings. Likewise, there is a deep dive on the architecture of the custom DCGAN implemented with PyTorch[3] used for the second task, the architectural configuration and the training settings.

The results of the models' training are described in the fifth chapter. Performance metrics have been considered: Average Precision,

[2]http://mmlab.ie.cuhk.edu.hk/projects/DeepFashion.html

[3]https://pytorch.org/

Recall and Intersection over Union are used to evaluate the Faster R-CNN and the Loss metric is used to measure the performance of the DCGAN.

Eventually, the conclusions are exposed in the sixth chapter with a particular focus on further improvements.

Chapter 2

State of the art

Computer vision is a multidisciplinary area of study whose application domains ranges from autonomous driving [48] and human computer interaction [37] to deseases diagnosis [27], robotics and surveillance [50].

In all these fields object detection is a very popular and crucial computer vision part because it aids several visual tasks such as pose estimation [59] [9], vehicle perception [33] [17], human and object parsing [57] and so on. For this reason, many progresses have been done in improving techniques to distinguishing and classifying instances of a certain semantic class of object from images.

In the past two decades, it is widely accepted that the progress of object detection has generally gone through two historical periods: traditional object detection period (before 2014) and deep learning based detection period (after 2014) [64].

Due to the lack of effective image representation, most of the

early object detection algorithms were built based on handcrafted features.

The most famous detector dating back to 2000s are the Viola-Jones detectors [53] [54], built in 2001 for face detections, based on sliding window technique; the HOG detectors [6], developed in 2005, widely known for their use in pedestrian detection and the Deformable Part-based Model (DPM) [12], proposed in 2008, based on the decomposition of the object and the inference of different object parts with ensembling detections.

An epochal turning point in object detection was made in 2014 with a widespread use of convolutional neural networks: a valid approach was introduced by Ross Girshick et al. to overcome the limitation of standard convolutional neural network: Region Based Convolutional Neural Networks (R-CNN). This method uses a selective search algorithm to compute candidates region proposals which are fed into a CNN that, acting as a feature extractor, generates the output dense layer which is fed into a SVM to classify the presence of the object within that candidate region [15].

To surmount the drawbacks of R-CNN, such as the huge amount of training time required, Girshick developed in 2015 a similar but faster object detection algorithm called: Fast Region Based Convolutional Neural Networks (Fast R-CNN). This method is faster than R-CNN because, instead of feeding the region proposals to the CNN, the entire input image is forwarded to the CNN to generate a convolutional feature map. At the end of the network a RoI pooling

layer is used to slices out each Region of Interest from the network's output tensor, reshape it into a fixed size so that it can be fed into a fully connected layer, and, finally, classify it by predicting the class of the proposed region [14].

One year later, Shaoqing Ren et al. invented Faster Region Based Convolutional Neural Networks (Faster R-CNN) to reduce the inference time, mostly caused by the use of selective search algorithm to find out the region proposals, by letting the network learn the region proposals itself [42].

In this project the Faster R-CNN model is used to build the fashion item detector which is trained on the DeepFashion dataset; a deep dive on Faster R-CNN architecture is in chapter 4.2 – "Item detection – Faster R-CNN: architectures and training".

Actually, a step forward in the object detection field was made in 2018: Kaiming He and a team of researchers, including Girshick, developed an architecture known as Mask R-CNN. This technique performs image segmentation: by adding a branch to Faster R-CNN (a Fully Convolutional Network) that outputs a binary mask that says whether or not a given pixel is part of an object [20]. Since in this project the outputs of the object detector are used as input training images for the second part of the experiment, it is important for them to be consistent in shape and dimension.

As a matter of fact, using Mask R-CNN and cropping the predicted mask, would generate infinite image shape and size that

would cause unstable training for GAN. For this reason, even though Mask R-CNN provides best accuracy for object detection, Faster R-CNN was chosen for this project.

For the second task, the one focused on generating new fashion images starting from existing ones, a particular deep-learning-based approach to generative modeling has been used: Generative Adversarial Network (GAN).

The GAN architecture was first described in 2014 by Ian Goodfellow et al. which proposed a new framework for estimating generative models via an adversarial process. In this framework two models are simultaneously trained: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than from G [16].

Further considerations on GANs structure, more specifically on generator and discriminator, are in chapter 4.3.

Due to their good performances, GANs are widely used in huge variety of applications, including human faces [25] and human poses [34] generation, new cartoon characters [24] generation, super-resolution photo editing [29] [1] [52], 3D object generation [56], video frames prediction [55] and video generation [7], image-to-image translation [22], text-to-image translation [61] and synthesis [41] and so on.

GANs have also been applied in fashion. Some existing research

papers focus on generating new images from existing ones and text description [44]; generating new outfit items based on a base image and a attribute description [63]; estimate the attributes of an existing garment (color, texture and shape) and modifying them through the generator [58]; generating multiple-view images from single-view ones [62] and creating virtual try-on images from simple fashion articles images [5] [28] [23].

The aim of this project is rather to generate high-quality fashion images starting from an ad-hoc dataset of real fashion images, carefully collected and preprocessed.

So, the aim is to generate new fashion item without any textual help or attributes claim.

Since the final task is an unsupervised task, one of the most notable architectural design of GAN will has been used: Deep Convolutional Generative Adversarial Network (DCGAN) formalized in 2015 by Alec Radford et al. [39].

In contrast with multi-scale architectures such as LAPGAN [8] or Progressively-Growing GAN [25], or in contrast with the state-of-the-art, BigGAN [2], which uses many auxiliary techniques such as Self-Attention [60] or Spectral Normalization [36], DCGAN is an easier system that can achieve good results.

While designing this architecture, the authors cite three sources of inspiration: - the all Convolutional Net: replacing pooling operations with spatial downsampling convolutions; - eliminating fully

connected layers after convolutions; - Batch Normalization to normalize activations and help gradient flow. The specific architectural guidelines the authors landed on, will be investigated more in-depth in chapter 4.4 – "Image generation – DCGAN: architecture and training".

Chapter 3

Data

In this work two different datasets have been used: the DeepFashion Dataset and the Pinterest one.

3.1 DeepFashion Dataset

DeepFashion [1][32] dataset is developed by the Multimedia Laboratory of the Chinese University of Hong Kong. It is one of the most used dataset in fashion researches and projects because it has a lot of advantages and appealing properties.

Besides being available for non-commercial research purposes, it is the largest clothing dataset to date, with over 800.000 diverse fashion images ranging from well-posed shop images to street snapshot, from structured studio images to unconstrained consumer photos.

Moreover, it is annotated with rich information of clothing items:

[1] http://mmlab.ie.cuhk.edu.hk/projects/DeepFashion.html

each image in this dataset is labeled with 50 categories, 1.000 descriptive attributes, bounding box and clothing landmarks.

Some examples of DeepFashion Images are shown in Figure 3.1.

Figure 3.1: Examples of DeepFashion images

For the aim of the project the image categories and bounding boxes have been used for clothing detection and category prediction: this part of the dataset consists of 289.222 fashion images [2]. In Figure 3.2 the distribution of the images for the top 20 fashion categories is shown.

[2]https://drive.google.com/drive/folders/0B7EVK8r0v71pQ2FuZ0k0QnhBQnc

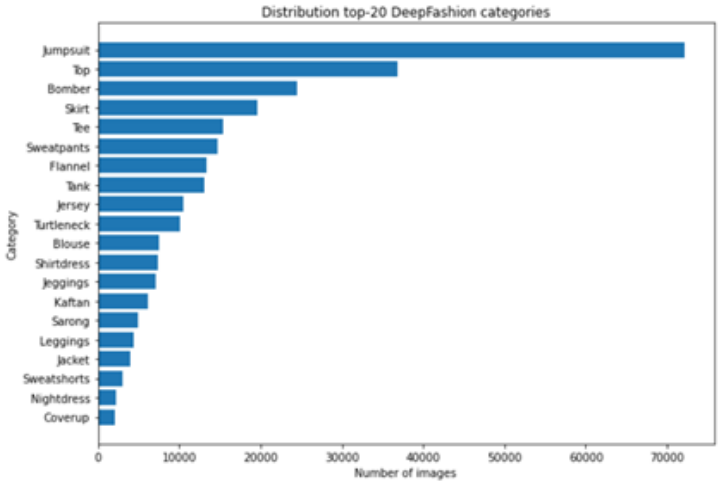

Figure 3.2: Distribution of DeepFashion images by top-20 categories

The 50 DeepFashion categories are divided in 3 category type: “upper”, “lower”, “full”.

The number of images for each category type is shown in Figure 3.3.

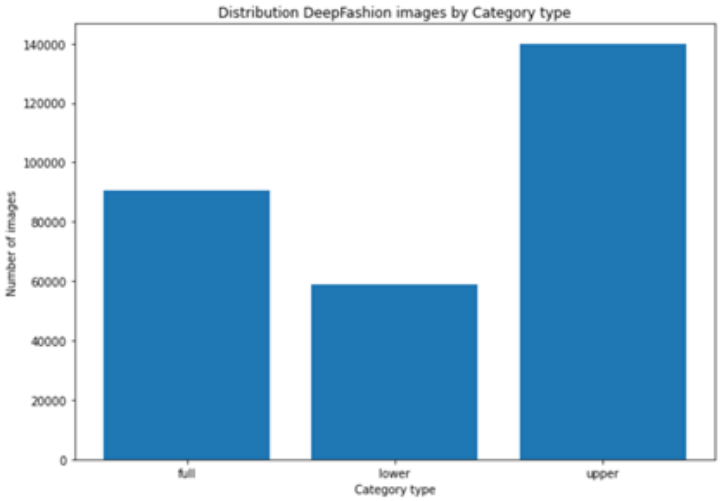

Figure 3.3: Distribution of DeepFashion images by category type

These 3 types identify if a fashion item is related to the upper part of the body, the lower one or if it is a full body item. This kind of information is the most important for the aim of the first task of this project: the item detection task is made with the purpose of generating output images with consistent shape and dimension because they are used as training data for the GAN in the second task of the project.

As a matter of fact, the detector implemented and described more specifically in chapter 4.1 and 4.2, was trained to predict the category type (upper, lower, full) of the item identified, not the specific category (top, shirt, blouse, etc.).

3.2 Pinterest Dataset

For the second part of the project a dataset made on purpose was used. More specifically, it is a fashion dataset made of images taken from the website Pinterest[3].

In this social network users can share images, videos and all kinds of visual content organizing them by hashtags.

For this project there was the need of a dataset to test the trained object detection model and, after cropping the fashion item area, use the output images to train the GAN and generate new realistic clothing images.

Thus comes the necessity to work with realistic, good quality and high definition images.

The photographs related to the fashion show of luxury brands fulfilled the requirements, as a results all the images related to the runways of 20 luxury fashion brand were collected: Balenciaga, Bluemarine, Burberry, Chanel, Chloe, Dior, Elisabetta Franchi, Fendi, GCDS, Givenchy, Gucci, Hermes, Louis Vuitton, Max Mara, Michael Kors, Missoni, Prada, Valentino, Versace and Yves Saint Laurent.

The scraping activity led to a dataset of 10.301 images. Some examples are shown in Figure 3.4: “Examples of Pinterest ‘Prada Runway’ images”.

[3]https://www.pinterest.com/

Figure 3.4: Examples of Pinterest 'Prada Runway' images

After analyzing the scraped images and noticing that some of them appear in multiple Pinterest board related to different brands, a phase of duplicates removing was implemented. The structure of the scraping tool made the images to be stored in different folders, one for each brand. All these 20 folders were merged in one to generate a single dataset for GANs. So, in order to store images once, even if they were scraped twice or more, before importing every single image in the final unique folder, the image name was previously searched in it to check if it was already present.

Eventually 9.940 distinct fashion images were collected.

3.2.1 Scraping tool

The scraper was implemented in Phyton using the package Selenium[4] (version 3.8.0) which is useful to automate web browser interaction. This package supports different browsers (Chrome, Edge, Firefox, Internet Explorer, Safari) and requires a specific driver to interface with the chosen browser. In this project the ChromeDriver[5] was used.

Basically, the scraper follows the procedure below:

1. The browser is automatically launched through the web driver;

2. The account is automatically logged into Pinterest by passing username and password;

3. Passing the html of the Pinterest board of interest (the one related to the runway of a luxury brand), the scraper automatically load the page and scroll it down till the end of the page, or rather till all the images are loaded from the Pinterest board. This is possible by setting a sufficient value of max iteration threshold and persistence: in this way we ensure the scraper to reach the end of the web page and not to stop in the middle if the request takes too long;

4. The scraper identifies all the pictures present in the web page and saves them in a list;

[4]https://www.selenium.dev/

[5]https://sites.google.com/a/chromium.org/chromedriver/downloads

5. It extracts the path to the original image;

6. Eventually, it downloads every single image and save it to the output local folder destination;

7. It repeats points from 3 to 6 for all the luxury brand of interest.

Concerning time performances, the scraper took about 20 minutes for each brand: it depended on how much images where associated to the hashtag, generally from 300 to 700 images.

Chapter 4

Metodology

The ultimate goal of this project is to generate new fashion images starting from real, good quality and high definition images of luxury brands' clothes.

In order to do it, two different parts were developed: a first task of object detection and the second task of image generation.

The aim of the object detection and cropping process is to provide clean data to GAN by removing unsuitable training images.

As shown in Figure 4.1 – "Metodology schema", DeepFashion dataset was used to train three different architectures of Faster R-CNN object detection model.

The one that achieved the best performance was then used to detect and crop fashion item on the test set, Pinterest dataset, generating thus the training set for the second part of the project.

The second task consists in developing a custom Deep Convolutional Generative Adversarial Network and train it on the aforementioned preprocessed Pinterest dataset.

Moreover, to evaluate the effectiveness of the first task of object detection, the same DCGAN was trained on Pinterest dataset without applying any cropping procedure.

Finally, new realistic fashion images were generated.

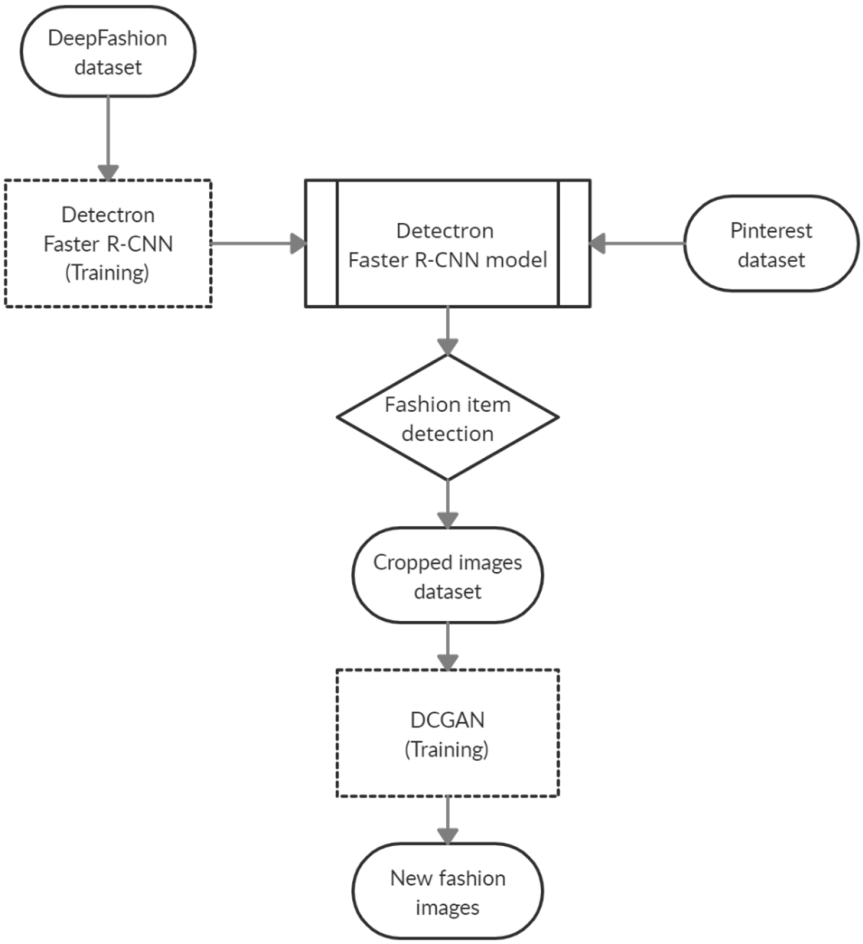

Figure 4.1: Metodology schema

4.1 Item detection – Facebook Detectron and COCO Style

Object detection is one of the areas of computer vision that is maturing very rapidly: new algorithms and models keep on outperforming year on year the previous ones.

One of the latest state-of-the-art software system for object detection was released by Facebook AI Research[1] (FAIR) team.

This high-performance codebase for object detection, that covers bounding box and object instance segmentation outputs, is called Detectron[2] and it is powered by the Caffe2[3] deep learning framework, which starting from 2018 has been merged with PyTorch[4] framework [3].

Starting from July 2016, when the Detectron project started, the codebase has matured and supported a large number of projects, including Mask R-CNN [20] and Focal Loss for Dense Object Detection [30], which won the Marr Prize and Best Student Paper awards, respectively, at International Conference on Computer Vision in 2017.

[1] https://ai.facebook.com/

[2] https://ai.facebook.com/tools/detectron/

[3] https://caffe2.ai/

[4] https://pytorch.org/

Detectron was open sourced on January 22, 2018 and is available under the Apache 2.0 license[5] on Facebook Research's official Github repository[6]. Moreover, more than 70 pre-trained models for object detection (YOLO, RCNN, Fast RCNN, Mask RCNN, RetinaNet etc.) are available and extensive performance baselines are also released [13].

Detectron can be used out-of-the-box for general object detection or it can be modified to train and run inference on different datasets, as in this case. Detectron supports custom dataset with an only requirement: using COCO annotation format. COCO, Common Objects in Context[7], is a large scale object detection, segmentation, and captioning dataset released by Microsoft in 2015 [31].

The Microsoft COCO dataset is the gold standard benchmark for evaluating the performance of state-of-the-art computer vision models because it has several features:

- Object segmentation;
- Recognition in context;
- Superpixel stuff segmentation;
- 330.000 images, more than 200.000 are labeled;
- 1,5 million object instances;

[5]https://www.apache.org/licenses/LICENSE-2.0

[6]https://github.com/facebookresearch/Detectron

[7]https://cocodataset.org

- 80 object categories;
- 91 stuff categories;
- 5 captions per image;
- 250.000 people with keypoints.

There is an annual competition based on Microsoft COCO dataset that has been held since 2015; it has less number of object categories than ImageNet Large Scale Visual Recognition Challenge (ILSVRC)[8], but more object instances. Compared with Pascal Visual Object Classes (VOC)[9] [11] and ILSVRC, the biggest progress of MS-COCO is that, apart from the bounding box annotations, each object is further labeled using per-instance segmentation to aid in precise localization.

In addition, MS-COCO contains more small objects (whose area is smaller than 1% of the image) and more densely located objects than VOC and ILSVRC.

All these features make the objects distribution in MS-COCO closer to those of the real world.

For these reasons, just like ImageNet in its time, Microsoft COCO has become is the most challenging object detection dataset available today [64].

COCO style datasets require the annotations to be stored in a its JSON format: the JSON file should follow the basic following

[8]http://image-net.org/challenges/LSVRC/2016/index

[9]http://host.robots.ox.ac.uk/pascal/VOC/

structure for object detection: info, image, license, annotations and categories.

In conclusion, in order to implement Faster R-CNN using Detectron, DeepFashion dataset was converted to COCO style to meet the requirement.

4.2 Item detection - Faster R-CNN: architectures and training

After converting the dataset to COCO format, the selected object detector model, Faster R-CNN [42], with different architectures has been configured with the corresponding hyperparameter and set for the training phase.

As explained before, the outputs of the object detector are used in this project as input training images for the second part of the experiment.

Due to this Faster R-CNN was preferred to Mask R-CNN because, even if it provides best accuracy for object detection, cropping the predicted mask would have generated infinite image shape and size that would have caused unstable training for GAN.

Faster R-CNN is a very good algorithm that eliminates the use of selective search algorithm to find out the region proposals, typical of R-CNN and Fast R-CNN, and replaces the region proposal algorithm by a region proposal network that gives an estimate of regions with objects.

This is the main reason why Faster R-CNN is much faster than its predecessors as shown in Figure 4.2 - “Comparison of test-time speed of object detection algorithms”.

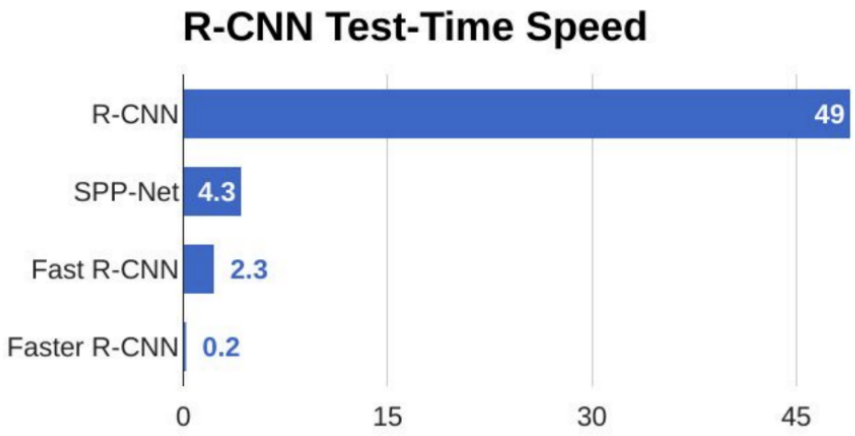

Figure 4.2: Comparison of test-time speed of object detection algorithms

So, basically Faster R-CNN consists of two stages.

The first stage, called a Region Proposal Network (RPN), proposes candidate object bounding boxes.

The second stage, which is in essence Fast R-CNN, extracts features using Region-of-Interest pooling (RoIPool) from each candidate box and performs classification and bounding-box regression.

Since both Fast R-CNN and RPN require a CNN based feature extractor to perform their task, they share a unique feature extractor instead of using two separate models with the almost same weight.

Figure 4.3 - “Faster R-CNN: a single unified network structure for object detection” shows the unified structure of Fast RCNN and RPN [10].

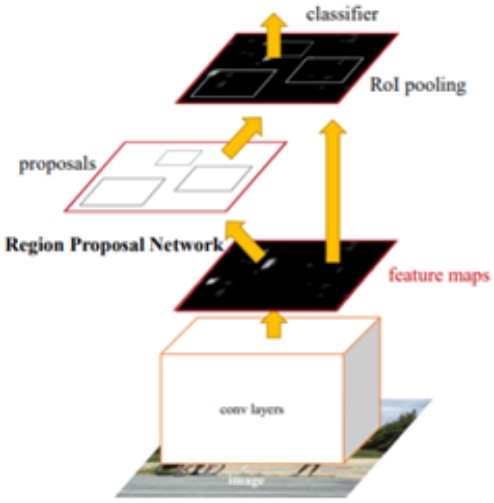

Figure 4.3: Faster R-CNN: a single unified network structure for object detection

CONVOLUTIONAL LAYERS AND FEATURE MAPS

The input images are represented as $Height \times Width \times Depth$ tensors (multidimensional arrays), which are passed through a pre-trained CNN up until an intermediate layer, ending up with a convolutional feature map which has spatial dimensions much smaller than the original image but greater depth as shown if Figure 4.4 - "From image to convolutional feature map".

The width and height of the feature map decrease because of the pooling applied between convolutional layers and the depth increases based on the number of filters the convolutional layer learns [40].

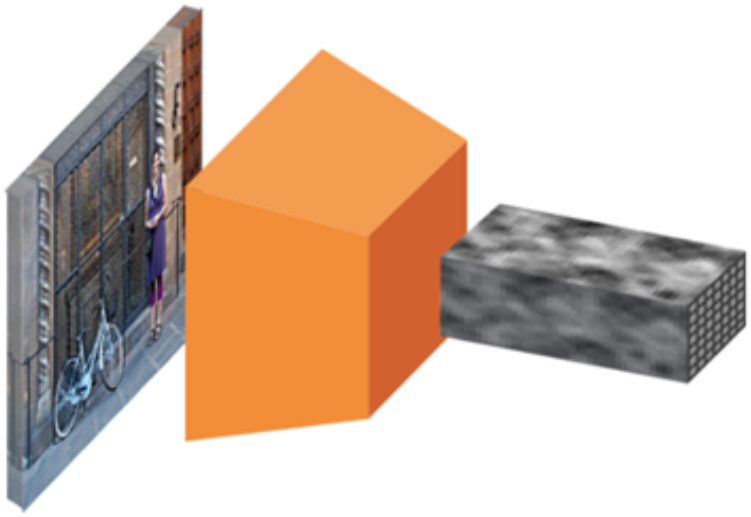

Figure 4.4: From image to convolutional feature map

Several architectures for the CNN can be used as feature extractor: one of the standard is VGG [47], but recent studies [19] demonstrated that residuals nets (ResNet), which are deeper and bigger than VGG, have more capacity to actually learn what is needed.

ResNets also ease the training of deep models by using residual connections and batch normalization.

REGION PROPOSAL NETWORK

The features that the CNN computed are passed into the Region Proposal Network which is used, as shown in Figure 4.5 - "From convolutional feature map to region proposal", to find the regions (bounding boxes), which may contain objects.

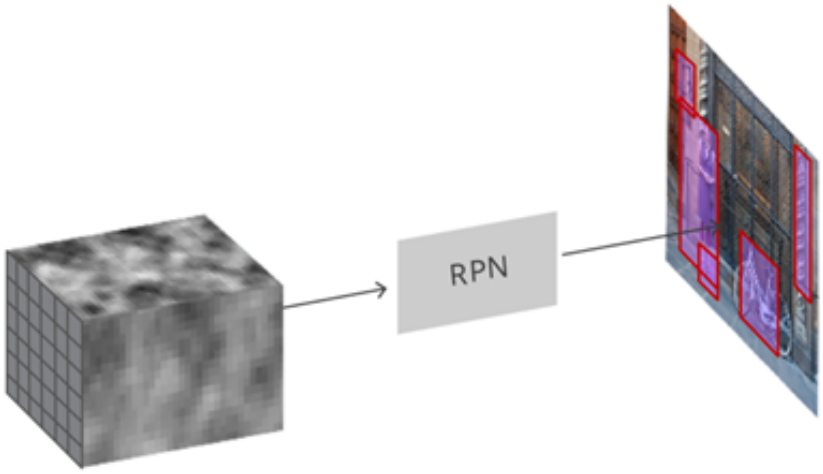

Figure 4.5: From convolutional feature map to region proposal

The RPN works with anchors: fixed bounding box placed uniformly on the image (actually on the features map) with different sizes (e.g. 64px, 128px, 256px) and ratios between width and height of boxes (e.g. 0.5, 1, 1.5).

For every anchor, the RPN looks for two aspects: how likely the anchor contains an object and how the coordinates of the bounding box can be adjusted to better fit the object.

To answer these two answer the RPN is efficiently implemented in a fully convolutional way: it has a fully convolutional layer with 512 channels and 3 x 3 kernel size and two parallel convolutional layers with 1 x 1 kernel size and respectively output of dimension 2 and 4.

The first parallel conv layer is the classification one and it outputs

two predictions per anchor: the score of it being background, not an object, and the score of it being foreground, an actual object.

The other parallel layer, instead, is the regression layer and predicts 4 values: $\{\Delta_x, \Delta_y, \Delta_{width}, \Delta_{height}\}$ which are the coordinates to apply to the anchor to better fit the object and get the final proposal.

Generally, anchors may overlap among them and it causes that there might be multiple proposals overlapping over the same object.

To solve this issue RPN uses a simple algorithmic approach called Non-Maximum Suppression [45] (NMS) and eventually find the top N, generally 2.000, proposals sorted by "objectness" classification score.

RoI POOLING

After the RPN step, Faster R-CNN extracts fixed-sized feature maps for each proposal using Region of Interest pooling.

The ROI pooling layer, in essence, crops the region corresponding to a proposal from the feature map, divides this region into a fixed number of sub-windows and performs max-pooling over these sub-windows to give a fixed size output: 7 x 7 x 512, where 512 is the depth of the conv feature map.

In Figure 4.6 - "Region of Interest Pooling", the structure of the RoI pooling layer is shown.

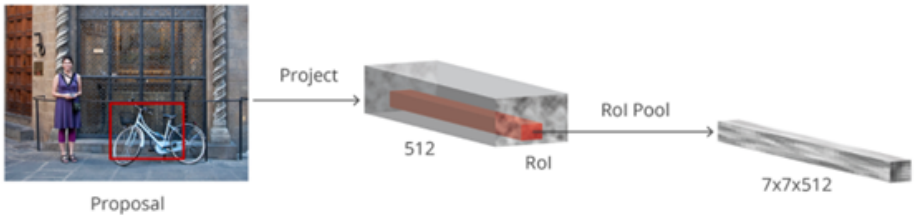

Figure 4.6: Region of Interest Pooling

The RoI pooling step is required by the next and final algorithm's step: fixed size feature maps are needed for the R-CNN in order to classify them into a fixed number of classes.

R-CNN

Region-based convolutional neural network (R-CNN) is the final step in Faster R-CNN's pipeline.

The fixed size feature map obtained from the Roi Pooling layer, is passed to the R-CNN which has two different goals similar to the RPN's ones: classify the proposal into one of the specific object classes or into the background class and better adjust the bounding box of the proposal according to the predicted class.

More specifically, R-CNN takes the feature map for each proposal, flattens it with a fully-connected layer of size 4096 with ReLU activation, and then pass it to two different fully-connected layers to achieve its two goals.

As shown in Figure 4.7 - "R-CNN: Region-based convolutional neural network", R-CNN has a first fully-connected layer with as

many units as the number of classes plus one, and a second fully-connected layer with 4N units where N is the total number of classes.

This is the regression layer one that predicts $\{\Delta_x, \Delta_y, \Delta_{width}, \Delta_{height}\}$ for each of the possible N classes.

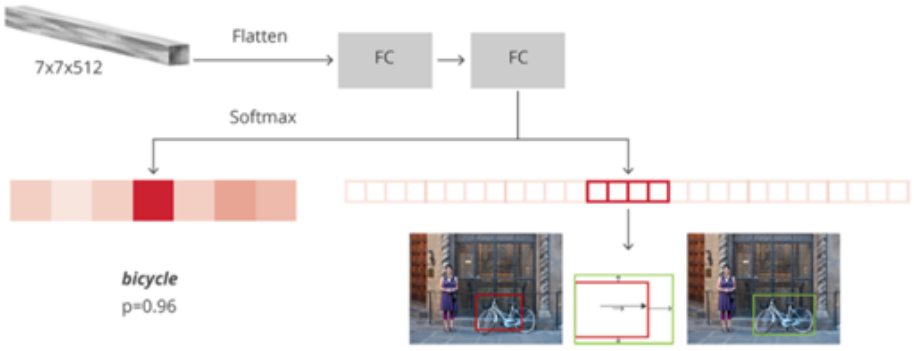

Figure 4.7: R-CNN: Region-based convolutional neural network

In this project three different model architectures were used with pretrained weights from ImageNet[10] dataset:

- ResNet 50, 23 million trainable parameters;
- ResNet 101, 45 million trainable parameters;
- ResNeXt-101, 88 million trainable parameters.

More specifically, the models were trained with the following hyperparameter settings:

- weight decay 0,0001;

[10]http://www.image-net.org/

- base learning rate 0,0025;
- steps [0, 20.000, 40.000];
- gamma rate 0,1;
- 50.000 epochs.

The models were trained on the training set made of around 70% of the entire DeepFashion dataset. As mentioned in chapter 3.1 – "DeepFashion Dataset", the Category and Attribute Prediction benchmark[11] from the DeepFashion dataset consists of 289.222 number of fashion images: 200.000 images randomly taken were used for training, and the rest 89.222 images were used for validation.

As mentioned from Detectron's official website[12], data augmentation was used just for training applying horizontal flipping to images; on the contrary no test-time augmentation was used on test set, Pinterest dataset in this case, for inference.

The training time for the first two models, using T4 or P100 NVIDIA Tesla GPU made available by Google Colaboratory Pro[13], was about 15 hours, for the ResNext architecture it took around 20 hours instead.

[11] https://drive.google.com/drive/folders/0B7EVK8r0v71pQ2FuZ0k0QnhBQnc

[12] https://github.com/facebookresearch/Detectron/blob/master/MODEL_ZOO.md

[13] https://colab.research.google.com/notebooks/intro.ipynb

4.3 Image generation – GAN

Generative models are unsupervised machine learning models that summarize the distribution of input variables, by automatically discovering and learning the regularities or patterns in them, and use it create or generate new examples that plausibly could have been drawn from the input distribution.

Two up-to-date examples of deep learning generative modeling algorithms are the Variational Autoencoder [26] (VAE) and the Generative Adversarial Network (GAN) [16].

The GAN model deep-learning architecture involves two submodels: a generator model that is used to generate new plausible examples from the problem domain, and a discriminator model for classifying whether generated examples are real, from the domain, or fake, generated by the generator model.

Thus GAN model consists of two networks: the generator (G) and the discriminator (D) network, which are multi-layer perceptrons [35] networks.

In the model, the two training procedures are separate, but simultaneous: during training, the purpose of the generator is to minimize the likelihood that the discriminator will classify its outputs as fake and the purpose of the discriminator is to maximize the probability that it correctly classifies whether images are real or fake.

More specifically, the model takes a noise input z which is defined as prior probability p_z, then it tries to learn the distribution of generator, p_g, by representing a mapping $G(Z;\theta_g)$ from z to data space.

The discriminator network D takes an input image, x then finds a mapping $D(x;\theta_d)$ from x to a single scalar, that is the probability of the image x being sampled from from p_{data}, where p_{data} defines where images are sampled from.

The output of network D returns a value close to 1 if the x is a real image that is from p_{data}. Otherwise, if the x is from p_g, the output will be very close to 0.

So the main goal of the network D is maximizing $D(x)$ for real images, generated from true data distribution p_{data} while minimizing $D(x) = D(G(z;\theta_g))$ for fake images, generated from p_z not p_{data}.

The aim of the generator G is, instead, to fool the network D, therefore G aims to maximizing $D(G(z;\theta_g))$. This is equivalent to minimize $1 - D(G(z;\theta_g))$ because D is a binary classifier [18].

There is thus a conflict among the goal of Generator and Discriminator and this odds is called as minimax game and it is given in equation (4.1).

$$minmax E_{X\sim p_{data}(x)}[logD(x)] + E_{Z\sim p_z(z)}[log(1 - D(G(z;\theta_g))] \quad (4.1)$$

The global optimum of this minimax game is the case of p_g= p_{data}, which means that the distribution of the real data is exactly the same distribution learned from the generator.

4.4 Image generation – DCGAN: architecture and training

Deep Convolutional Generative Adversarial Network (DCGAN) [39] is one of the most successful and notable architectural designs of GAN that uses deep convolutional networks in place of the fully-connected networks used in the original GAN.

Convolutional nets in general find areas of correlation within an image, which means that they look for spatial correlations. This makes DCGAN more fitting for image and video data, whereas the general idea of a GAN can be applied to wider domains.

DCGAN was firstly formalized by Alec Radford et al. in 2015 [39] and, as mentioned in chapter 2 - "State of the Art", while designing this architecture, the authors cite three sources of inspiration:

- the all convolutional net, replacing pooling operations with spatial downsampling convolutions;
- eliminating fully connected layers after convolutions;
- Batch Normalization to normalize activations and help gradient flow.

With these advancements in mind, the authors searched for a stable DC-GAN architecture and landed on the following architectural guidelines:

- remove fully connected hidden layers for deeper architectures;

- use transposed convolution for upsampling;
- replace any pooling layers with strided convolutions in the discriminator and fractional-strided convolutions in the generator for downsampling;
- use batch normalization layers in the generator and discriminator, except for the output layer of the generator and the input layer of the discriminator;
- use ReLU activation in generator for al layers except for Tanh in output;
- use LeakyReLU activation in the discriminator.

In Figure 4.8 - "DCGAN – Generator Original Architecture" [39] it is possible to see all the steps used by the generator network in order to transform a random noise and map it into images such that the discriminator is not able to distinguish from real to fake images.

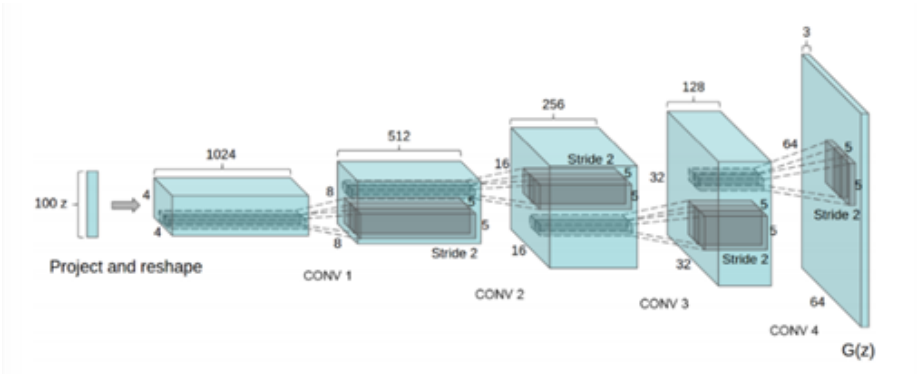

Figure 4.8: DCGAN – Generator Original Architecture

The network takes a 100 x 1 noise vector z and maps it into the G(Z) output which is 64 x 64 x 3, because the network generates 3 channel RGB 64 x 64 images.

This first layer of the architecture, denoted as 'Project and reshape', is particularly interesting because it expands the random noise from 100 x 1 to 1024 x 4 x 4.

This layer is followed by four classical convolutional layers where the parameter height/width goes from 4 to 8, to 16, to 32, till 64, dimension of final output [46].

The kernel filter parameter is 5 x 5 and a stride of 2 for downsample: the original paper [39] mentions it is a good practice to use strided convolution [49] rather than pooling to downsample because it lets the network learn its own pooling function.

The discriminator network, instead, is a binary classification network: it takes the 64 x 64 x 3 image from the generator and uses it as input.

It processes it through a series of convolutional layers, batch normalization layers and LeakyReLU layers, and output the final probability through a Sigmoid activation function.

Due to computational limitation, in this project the focus was not on the optimization of the hyperparameter but on the general performance that GANs can reach in fashion domain.

As a matter of fact, the aim is to see how DCGANs works having as input only high-quality fashion images without any textual

information and to investigate the effectiveness of cropping fashion images before passing to GAN.

Thus, a classical DCGAN architecture was used, generating 64 x 64 x 3 channel RGB fashion images.

As can be seen from Figure 4.9 - "DCGAN – Generator Architecture" and Figure 4.10 - "DCGAN – Discriminator Architecture" the convolutional architecture developed using PyTorch[14] implementation is similar to the original one described above but with lower tensor dimensions, in order to reduce the computational training effort.

```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
```

Figure 4.9: DCGAN – Generator Architecture

[14]https://pytorch.org/

```
Discriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): Sigmoid()
  )
)
```

Figure 4.10: DCGAN – Discriminator Architecture

The model was trained on the Pinterest dataset which was previously used as test set of the Faster R-CNN object detector. From the initial 9.940 scraped images, only the ones classified as full body item were cropped based on their predicted bounding boxes and considered for training the GAN: a dataset of 5.328 cropped images was finally used as GAN's training dataset.

Moreover, in order to test the effectiveness of object detection task, the model was trained with other two different training dataset:

- 'Pinterest original dataset': the whole Pinterest dataset was used to train the GAN. 9.940 images of full, lower and upper fashion item, but also a lot of noise: among the scraped images there were also some unsuitable training ones such as prevalence of background, head shot, multiple human figures, shoes or accessories close-up and so on.
- 'Pinterest subset dataset': a random subset of 5.328 images from original Pinterest dataset was used to train the GAN. This experiment is particularly important because the same

number of images of the 'cropped dataset' is considered.

In this way it is possible to really test if detecting full fashion item and cropping it before training GAN is a crucial activity or if the noise is not a real negative factor for GAN in fashion domain.

All the three models were trained for 200 epochs with a batch size of 128.

Adam optimizer was setup both for Generator and Discriminator with beta hyperparameter of 0,5 and learning rate of 0,0002.

With these settings the training time for the first and the third experiment was around 7 hours, instead the second experiment took around 9 hours due to the greatest amount of training images.

Chapter 5

Results and evaluation

5.1 Item detection – Average Precision, Recall and Intersection over Union

In order to evaluate quantitatively the performances of the 3 architectures of Faster R-CNN developed for the first task of item detection, the metrics proposed by COCO in its official website[1] and shown in Figure 5.1 - "COCO evaluation metrics" were used:

[1] https://cocodataset.org/#detection-eval

```
Average Precision (AP):
  AP                  % AP at IoU=.50:.05:.95 (primary challenge metric)
  AP^IoU=.50          % AP at IoU=.50 (PASCAL VOC metric)
  AP^IoU=.75          % AP at IoU=.75 (strict metric)
AP Across Scales:
  AP^small            % AP for small objects: area < 32^2
  AP^medium           % AP for medium objects: 32^2 < area < 96^2
  AP^large            % AP for large objects: area > 96^2
Average Recall (AR):
  AR^max=1            % AR given 1 detection per image
  AR^max=10           % AR given 10 detections per image
  AR^max=100          % AR given 100 detections per image
AR Across Scales:
  AR^small            % AR for small objects: area < 32^2
  AR^medium           % AR for medium objects: 32^2 < area < 96^2
  AR^large            % AR for large objects: area > 96^2
```

Figure 5.1: COCO evaluation metrics

Precision and Recall [51] [38] are often used in classification tasks but also in information retrieval because they allow to compute respectively the proportion of positive identifications that were actually correct and the proportion of actual positives that were identified correctly.

To fully evaluate the effectiveness of a model, both precision and recall must be examined jointly because they are analytically opposed: improving precision typically reduces recall and vice versa, as is visible looking at their mathematical formulas:

$$Precision = \frac{TP}{TP + FP} \tag{5.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

In the equations (5.1) and (5.2) TP represents *true positives*, number of instances predicted correctly as positive; FP represents *false positive*s, number of instances predicted as positive even if they were actually negative; and FN represents false negatives, number of object predicted as negative even if they were actually positive.

The general definition for the Average Precision (AP) is finding the area under the precision-recall curve above and it's generally calculated for each class; Mean Average Precision (mAP) is therefore the average of AP.

mAP is a metric that comes from information retrieval and is commonly used for calculating the error in ranking problems and for evaluating object detection problems.

Basically, mAP penalizes the model when it misses a box that it should have detected, as well as when it detects something that does not exist or detects the same thing multiple times.

Even if in some context AP is calculated for each class and averaged to get the mAP, in others, such as in COCO challenge evaluation[2], there is no difference between AP and mAP.

So, as previously shown in Figure 5.1 - "COCO evaluation metrics", the evaluation is done using the standard Average Precision (AP) and Recall at some specific IoU threshold.

[2]https://cocodataset.org/#detection-eval

Intersection over Union (IoU) is another evaluation metric that can be used to assess the performance of any kind of algorithm that has predicted bounding boxes as its output [43].

As shown in Figure 5.2 - "Intersection over Union ratio", Intersection over Union is simply a ratio: the numerator is given by the area of overlap between the predicted bounding box and the ground-truth bounding box, which is the correct one; the denominator, instead, is the area of union, or more simply, the area encompassed by both the predicted bounding box and the ground-truth bounding box.

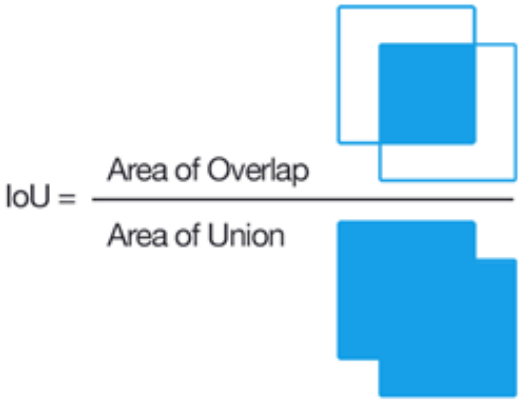

Figure 5.2: Intersection over Union ratio

This evaluation metrics rewards predicted bounding boxes that heavily overlaps the ground-truth bounding boxes.

In this case IoU metric is used as threshold of acceptance of error between the ground truth and the prediction of the bounding boxes.

For example, a value of 0.75 IoU means that, if the area of the region of intersection over union between the ground truth and the

prediction is greater than 0.75, the prediction is counted as correct.

A further distinction in computing the evaluation metrics given by COCO is the differentiation by object dimension: they compute AP and Recall across small, medium, and large objects.

More specifically, according to COCO, the area of the object is measured as the number of pixels in the segmentation mask and is divided into 3 classes: small size objects, that have area less than 32^2 pixels, medium size objects, that have area between 32^2 and 96^2 pixels, and large size objects, whose area is larger than 96^2 pixels .

In Table 5.1 – "Object detection models evaluation metrics", the values of AP and Recall achieved by the models developed are shown.

Metric	**IoU**	**Area**	**ResNet-50**	**ResNet-101**	**ResNeXt-101**
Average Precision	0.50:0.95	All	0.647	0.663	0.635
Average Precision	0.50	All	0.890	0.896	0.881
Average Precision	0.75	All	0.770	0.787	0.758
Average Precision	0.50:0.95	Small	0.031	0.071	0.230
Average Precision	0.50:0.95	Medium	0.374	0.393	0.374
Average Precision	0.50:0.95	Large	0.672	0.687	0.660
Recall	0.50:0.95	All	0.724	0.737	0.720
Recall	0.50	All	0.752	0.764	0.751
Recall	0.75	All	0.752	0.764	0.751
Recall	0.50:0.95	Small	0.097	0.192	0.286
Recall	0.50:0.95	Medium	0.592	0.611	0.604
Recall	0.50:0.95	Large	0.768	0.780	0.767

Table 5.1: Object detection models evaluation metrics

From Detectron Model Zoo and Baselines[3], the benchmarks for the three architectures ResNet-50, ResNet-101 and ResNeXt-101 for Faster R-CNN model are available. These three models achieve respectively an Average Precision value predicting bounding box of 0,367, 0,394 and 0,413.

As mentioned on Detectron's official Github repository[4], all these baselines provided were run on Big Basin servers with 8 NVIDIA Tesla P100 GPU accelerators (with 16GB GPU memory, CUDA 8.0, and cuDNN 6.0.21) and trained using 8 GPU data parallel sync SGD with a minibatch size of either 8 or 16 images. Besides this computational power, an additional point needs to be taken into account: these models were trained on COCO dataset.

As shown in Table 5.1 - "Object detection models evaluation metrics", the performances achieved in this project are higher than Detectron baselines and also higher than the ones listed in COCO dataset leaderboard , where it is possible to check out the state-of-the-art algorithms suitable for various tasks, including object detection.

This significant difference can be explained by the dataset used model training and validation: COCO dataset is harder for object detection and usually detectors achieve much lower AP values [21].

[3]https://github.com/facebookresearch/Detectron/blob/master/MODEL_ZOO.md

[4]https://github.com/facebookresearch/Detectron/blob/master/MODEL_ZOO.md

Using DeepFashion dataset, instead, it's possible to achieve better results as in [4], where detectors reach around 96% of Average Precision.

However, these excellent results were achieved using consisting computational power: p3.2xlarge instance and a training time of 24 hours. These performances are not possible on Google Colab Pro due to both GPU and runtime availability.

So, all things considering, the performances achieved by the models implemented in this experiment and shown in Table 1, are still good ones.

In the following lines there is a further analysis of the results and comparison between different detection models.

Among the three architectures developed, ResNeXt-101 has the most complex architecture with more weights than the two other models, followed by ResNet-101 and lastly, ResNet-50.

If trained properly, the more the model is complex, the better performance it achieves, also evident in Detectron's baselines .

In this project, instead, as shown in Table 5.1 - "Object detection models evaluation metrics", ResNet-101 actually achieves higher AP and Recall than the other two, followed by ResNet-50, and finally ResNetXt-101.

This surprisingly differs from the results provided by the baselines. However, as mentioned before, it can be explained by the fact that more complex architectures with more parameters usually take longer to train and longer to converge.

Since all three models were trained with a comparatively lower-end GPU, for the same number of epochs which took almost the same number of hours, ResNetXt-101 might not have converged yet, while ResNet-50 and ResNet-101 might have.

Due to runtime availability, timeouts and time constraints, it is difficult to run very long-running computations on Google Colab so it is hard to train models longer enough to see significant improvements in performance.

As shown in Table 5.1, for a IoU value stepping from 0,5 to 0,95, ResNet-101 achieved an AP value of 66,3%, ResNet-50 an AP value of 64,7% and ResNeXt-101 an AP value of 63,5%.

Better results are on Recall value: with the same IoU stepping from 0,5 to 0,95 ResNet-101, ResNet-50 and ResNeXt-101 achieved a Recall value of respectively 73,7%, 72,4% and 72,0%.

Even though these results are incomparable with the baseline results because the dataset used to train is different, these results are relatively good considering all the problems and limitation of the scenario.

An exception is for Average Precision and Recall related to object of small size: for these images ResNext-101 performs significantly better than the other two models.

The performance on small areas are relatively low; it achieves

an AP value of 23% and a Recall of 28,6%, significantly higher than ResNet-50 and ResNet-101 which have AP lower than 10% and Recall lower than 20%.

This general low performance on small objects might be related to the use of DeepFashion dataset: being composed of fashion images, all the images are likely focused on the item and not on the background, so objects with a medium or large area are prevalent and objects that are classified as small size may not be found in a lot of the images. Moreover, since the objects are small, the item may not be easy to classify, thus leading to low detection accuracy. However, for the aim of this first task of the experiment these poor results on small objects are not so crucial because the images used as test set, Pinterest dataset, will likely have similar characteristics to DeepFashion.

In conclusion, focusing on the performance related to all sizes, such as medium and large ones, the model with the best performance among the other was ResNet-101 and it was used on Pinterest dataset to output cropped images for the second task of the project.

5.2 Image generation – Generator and Discriminator Lossn

As mentioned in chapter 4.3 – "Image generation – GAN", the global optimum of Generator and Discriminator minimax game is reached when the distribution of the real data is exactly the same distribution learned from the generator network.

In the DCGANs developed in this project, both Generator and Discriminator are trained together to achieve and maintain an equilibrium. In order to train, and then to compare and evaluate the models, Binary Cross Entropy Loss was used.

BCELoss[5] is defined as follow:

$$l(x, y) = L = \{l_1, ..., l_N\}^T, l_n = -[y_n \cdot logx_n + (1 - y_n) \cdot log(1 - x_n)] \tag{5.3}$$

In equation (5.3) y defines the label, in this case the same convention used in the original paper [16] was used: 1 is the label for real images and 0 for fake ones.

$logx_n$ and $log(1 - x_n)$ are the two main log components of the objective function: in this scenario they are $log(D(x))$ and $log(1 - D(G(z)))$.

As explained in chapter 4.3 – "Image generation – GAN", $D(x)$

[5]https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html#torch.nn.BCELoss

is the output of the discriminator network for real images (y=1) and $D(G(z))$ is the output for fake images (y=0) which are created by generator network through the learned mapping G applied to the noise z.

So, during Generator and Discriminator training some statistic reporting was done in order to visually track the progress of G and D. At the end of each epoch the following training statistics are computed:

- $Loss\_D$ which is the loss of the Discriminator, calculated as the sum of losses for both real and fake batches

  $(log(D(x)) + log(D(G(z)))$

- $Loss\_G$ which is the loss of Generator, calculated as $log(D(G(z))$

- $D(x)$ that is the average output of Discriminator calculated across all batches.

  As mentioned in chapter 4.3 – "Image generation – GAN", $D(x)$ is the probability of the image x being sampled from from p_{data}, where p_{data} defines where images are sampled.

  This value can thus range between 0, for fake images, and 1, for real images. During the first epochs of training, it is close to 1 and then converge to 0,5 when generator gets better. This happens because at the beginning of training the Generator is not able to produce good fake images so the Discriminator manages to distinguish them and outputs 1 for real batches, 0 for fake ones and the average output $D(x)$ is thus close to 1. Once the training is almost over, G produces good fake images so D

outputs are always close to 0,5 at least theoretically, because D is not able to state with certainty if the image is fake or real.

As mentioned in chapter 4.4 – "Image generation – DCGAN: architecture and training", three different experiments were done.

The DCGANs were trained on three different datasets:

- '*Pinterest cropped dataset*': 5.328 cropped images of full body fashion items. Original Pinterest dataset was previously used as test set of the Faster R-CNN object detector; so, from the initial 9.940 scraped images, only the ones classified as full body item were cropped based on their predicted bounding boxes and considered in this training set;

- '*Pinterest original dataset*': 9.940 images of full, lower and upper fashion items, but also a lot of noise like background scenes, head shot, multiple human figures, shoes or accessories close-up and so on;

- '*Pinterest subset dataset*': 5328 images randomly taken from Pinterest original dataset.

Insights into the results of all of the three models are given below. More specifically for each experiment the following information are reported:

- Generator and Discriminator Loss ($Loss\_G$, $Loss\_D$) and Discriminator average output ($D(x)$) achieved in the best epoch;

- A chart showing the progress of Generator Loss and Discriminator Loss during training;
- A sample of 64 generated images compared with 64 real images randomly taken from the corresponding training set.

Pinterest Cropped dataset

The DCGAN trained on Pinterest cropped dataset achieves:

$Loss\_G$	$Loss\_D$	$D(x)$
2,0062	0,5132	0,6736

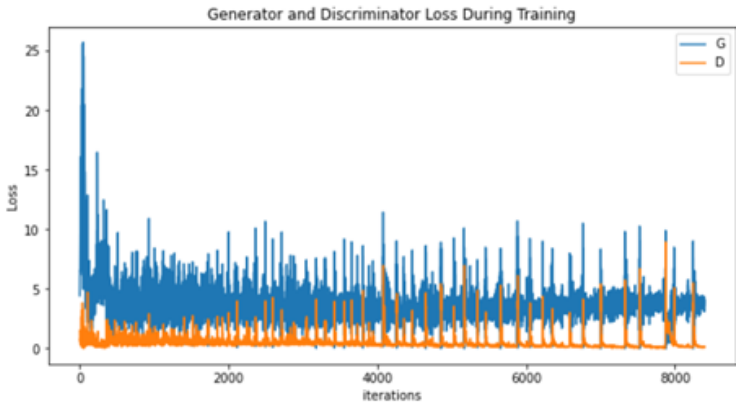

Figure 5.3: DCGAN Loss on 'Pinterest cropped dataset'

Figure 5.4: DCGAN real vs generated images - cropped dataset

Pinterest Original dataset

The DCGAN trained on Pinterest original dataset achieves:

$Loss\_G$	$Loss\_D$	$D(x)$
2,2091	0,5668	0,6509

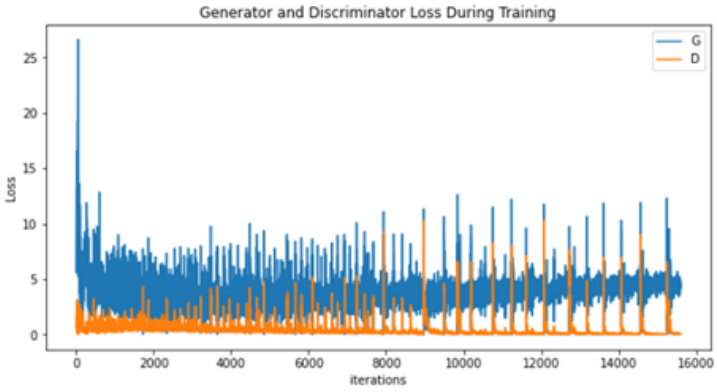

Figure 5.5: DCGAN Loss on 'Pinterest original dataset'

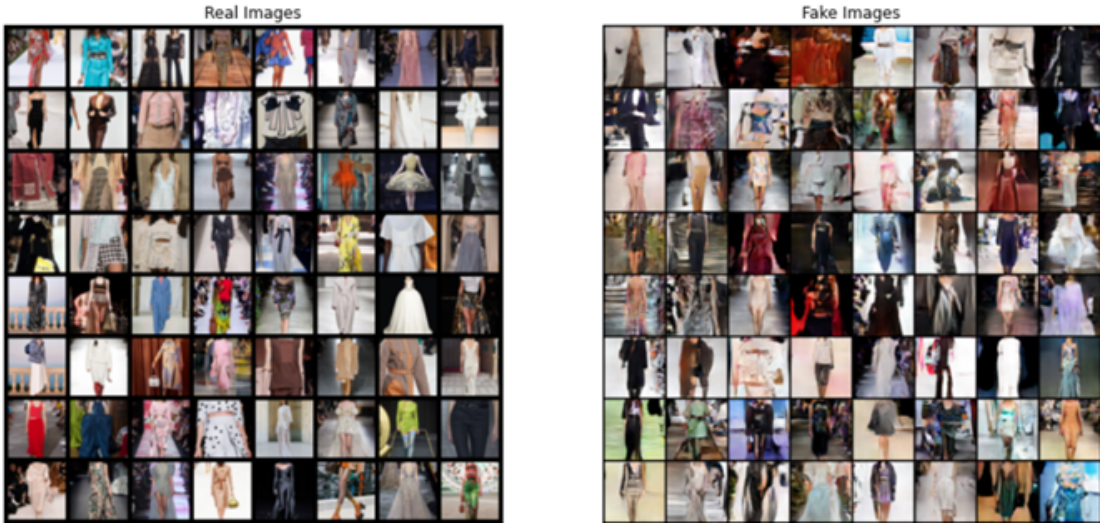

Figure 5.6: DCGAN real vs generated images - original dataset

Pinterest Subset dataset

The DCGAN trained on a sample of Pinterest original dataset achieves:

$Loss\_G$	$Loss\_D$	$D(x)$
2,7983	0,6572	0,6469

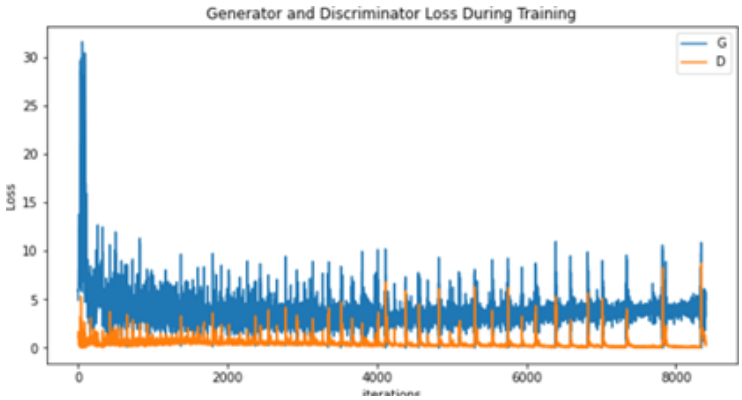

Figure 5.7: DCGAN Loss on 'Pinterest subset dataset'

Figure 5.8: DCGAN real vs generated images - subset dataset

Overall, DCGANs have good performances, according to the outputs shown above. Models' good achievements are evident both in Generator and Discriminator Loss values, but also in the charts showing their progresses during training: Generator and Discriminator models converge in all the three experiments.

Further confirmation of DCGANs' general good performance is given by Figure 5.4, Figure 5.6 and Figure 5.8 where is possible to see the comparison between real images, taken from the corresponding training dataset, and fake ones, generated by the model. These latter are very eye-appealing and likely.

Concerning comparison between the three experiments, no particular differences emerge from the output.

Generator and Discriminator Loss ($Loss\_G$, $Loss\_D$) ranges respectively between 2,0062 and 2,7983 and between 0,5132 and 0,6572; Discriminator average output ($D(x)$) achieved in the best epoch ranges between 0,6469 and 0,6736.

These extremely slight differences between the results suggest that the prior task of object detection and image cropping does not affect DCGAN performances.

So, in other words, the noise introduced in the training set using original scraped images instead of cropped fashion items, doesn't have an impact on the GAN ability of learning how to generate new realistic fashion images.

Regarding the second experiment, where GAN were trained on 'Pinterest original dataset', the noise introduced by using not cropped images might have been offset by an increased dimension of training set: 9.940 images were used, instead of 5.328.

Using twice images to train DCGAN might have balanced the introduction of noise and unsuitable images in training dataset.

For these reasons, the third experiment, the one related to 'Pinterest subset dataset', is essential to really test if detecting full fashion item and cropping it before training GAN is a crucial activity or if the noise is not a real negative factor for GAN in fashion domain.

However, by comparing first and third experiments, almost the same results were obtained.

In Table 5.2 Generator and Discriminator Loss and Discriminator average output of first and third experiment are given and in Figure 5.9 comparison between images generate by the two different networks is shown.

Metric	**Pinterest cropped dataset**	**Pinterest subset dataset**
Loss_G	2,0062	2,7983
Loss_D	0,5139	0,6572
$D(x)$	0,6736	0,6469

Table 5.2: DCGAN performances on different training dataset

Figure 5.9: DCGAN generated images comparison between 'cropped' and 'subset' training dataset

Concerning performance metrics values, which we remember to be in conflict with one another, Generator and Discriminator achieve a value of BCELoss slightly better when training on cropped images. Instead, the Discriminator average output results thinly better when training with unpreprocessed images.

However these differences are not significant enough to be defined as real differences. As a matter of fact is very difficult in Figure 23 to distinguis which images are generated from training with cropped images and which not.

The reasons that explain the ineffectiveness of cropping images are listed below:

- Pinterest original images did not contain too much noise.

The images related to the hashtag composed of 'brand's name' + 'runway' were scraped. In this way, the majority of the images were official photos of fashion show, whit a model placed in foreground and the fashion items that occupy most of the image like the ones showed as an example in chapter 3.2.1 "Pinterest Dataset" in Figure 3.4: "Examples of Pinterest 'Prada Runway' images".

Probably, scraping all the images related just to the hashtag composed of the brand's name, without the keyword 'runway', would have led to collect more noise images: pictures of accessories, Pinterest users' street photograph, covers of newspapers, and so on.

- Fashion domain coupled with the scenario of this project came up particularly favourable for GAN.

  The object of interest for the network is the fashion item which is in most of the images exactly in the same position.

  Moreover, the models in Pinterest images are always in a frontal pose because they were walking when the photo was taken. This definitely helps GAN training.

- Fashion domain also help GAN to "hide" some little errors in output images. Some little imperfections, such as a spot on a skirt or an asymmetry on a dress, might likely be considered respectively as a particular color choice of the stylist or as the shape assumed by the dress due to the moves of the model.

So, in conclusion, the task of image generation has been overall

wholly fulfilled: new realistic fashion clothing images were generated following current fashion trends and mixing styles and pattern from different luxury brands.

Regarding the task of fashion item detection and image cropping, instead, it has proven to be not a crucial preliminary activity to help DCGAN training.

Chapter 6

Conclusion and further improvements

The aim of the project was to generate new fashion clothing images to be used as a reference for stylist and designers in creating future works.

This goal was altogether achieved by using DCGAN, a notable architectural design for Generative Adversarial Network, deeply described in chapter 4.2 – 'Image generation – GAN' and 4.3 – 'Image generation – DCGAN: architecture and training'.

This kind of models generally work better during training if images used as input are consistent in shape and dimensions.

Therefore, an object detection system was implemented as preliminary task to detect fashion item and crop images in order to make them cleaner and more suitable before passing them to DCGANs.

As thoroughly outlined in chapter 4.2 – 'Item detection - Faster R-CNN: architectures and training', Faster R-CNN model was implemented with several architectures trained on DeepFashion dataset.

Then, different experiments were carried out and described in chapter 4.4 – 'Image generation – DCGAN: architecture and training'.

These experiments showed that preprocessing images through an object detector to make them more suitable for GANs, was not such a crucial preliminary activity to help DCGANs training.

In fact, in this particular fashion scenario the combined network of Faster R-CNN and DCGAN successfully generated a set of sample images just like DCGAN used standalone did.

The reasons why this happened are meticulously listed in chapter 5.2 – 'Image generation – Generator and Discriminator Loss'.

The main point, however, is that fashion domain is particularly advantageous for GANs.

Due to fashion domain some little errors or imperfections in output images can be in some ways "hidden" because it might likely be considered as intentional style choices.

But above all, DCGANs managed to generate good fashion images even without object detection and cropping task, thanks to the appropriate Pinterest dataset used for training.

Using a dataset created ad hoc with official photos of fashion

shows, has been essential for the success of GAN's training: scraping Pinterest website has made it possible to collect the most suitable photos for this task, with the smallest possible portion of noise.

In conclusion, by using DCGANs, both with and without object detection, new realistic and aesthetically appealing fashion clothing images were generated following current fashion trends and mixing styles from different luxury brands.

Despite the task of image generation has been overall wholly fulfilled, future improvements can be made.

The most important one is definitely increasing the computational power to train longer both the DCGANs but also the object detector models. This might have led ResNetXt-101 to perform better and it could have help reduce noise for the images fed to GAN even more than ResNet-101 did, making object detection task more significant and effective.

With greater computational power more also about DCGANs could have been explored.

The hyperparameters of both generator and discriminator might have been tuned, such as adding more convolutional and activation layers, changing image sizes and the kernel size, modifying the length of latent vector z, increasing the depth of feature maps carried through the generator and propagated through the discriminator.

Moreover, dropout should be introduced for generator, since existing research and tutorials suggest that it might improve the performance of GANs [22] because it prevents overfitting leading to clearer and more defined output images.

Lastly, a possible further development could be introduced by using as input images fashion show photos of just one brand, scraping spring/summer and fall/winter collections of different years. This would allow fashion industries to analyze the trend of their competitors' style and forecast fashion items that will be launched in the next season.

Bibliography

[1] Huang Bin et al. “High-Quality Face Image SR Using Conditional Generative Adversarial Networks”. In: (July 2017).

[2] Andrew Brock, Jeff Donahue, and Karen Simonyan. *Large Scale GAN Training for High Fidelity Natural Image Synthesis*. Sept. 2018.

[3] “Caffe2 and PyTorch join forces to create a Research + Production platform PyTorch 1.0”. In: *Caffe2 AI Blog* (May 2018).

[4] D. Carnino. *Clothing Detection for Fashion Recommendation*. June 2018.

[5] C. Cokbert. “GAN Fashion Photo Shoot: Garment to Model Images Using Conditional GANs”. In: *GPU Technology Conference Silicon Valle*. 2018.

[6] N. Dalal and B. Triggs. “Histograms of oriented gradients for human detection”. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 1 (2005), pp. 886–893. DOI: 10.1109/CVPR.2005.177.

[7] Emily Denton and Rob Fergus. “Stochastic Video Generation with a Learned Prior”. In: (Feb. 2018).

[8] Emily Denton et al. “Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks”. In: (June 2015).

[9] Jian Dong et al. “Towards Unified Human Parsing and Pose Estimation”. In: June 2014, pp. 843–850. DOI: 10.1109/CVPR.2014.113.

[10] P. Dwivedi. “Semantic Segmentation - Popular Architectures”. In: (Mar. 2019).

[11] M. Everingham et al. “The Pascal Visual Object Classes Challenge: A Retrospective”. In: *International Journal of Computer Vision* 111 (2014), pp. 98–136.

[12] Pedro Felzenszwalb, David Mcallester, and Deva Ramanan. “A Discriminatively Trained, Multiscale, Deformable Part Model”. In: vol. 8: June 2008. DOI: 10.1109/CVPR.2008.4587597.

[13] R. Girshick. “Facebook open sources Detectron”. In: *Facebook Research Blog* (Jan. 2018).

[14] Ross Girshick. “Fast r-cnn”. In: (Apr. 2015). DOI: 10.1109/ICCV.2015.169.

[15] Ross Girshick et al. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Nov. 2013). DOI: 10.1109/CVPR.2014.81.

[16] Ian Goodfellow et al. “Generative Adversarial Networks”. In: *Advances in Neural Information Processing Systems* 3 (June 2014). DOI: 10.1145/3422622.

[17] Yuliang Guo et al. *PoP-Net: Pose over Parts Network for Multi-Person 3D Pose Estimation from a Depth Image*. Dec. 2020.

[18] Ceren Guzel Turhan and H. Bilge. "Recent Trends in Deep Generative Models: a Review". In: Sept. 2018. DOI: 10.1109/UBMK.2018.8566353.

[19] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: June 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[20] Kaiming He et al. "Mask R-CNN". In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 2980–2988.

[21] J. Hui. *Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3)*. Mar. 2018.

[22] Phillip Isola et al. "Image-to-Image Translation with Conditional Adversarial Networks". In: July 2017, pp. 5967–5976. DOI: 10.1109/CVPR.2017.632.

[23] Nikolay Jetchev and Urs Bergmann. "The Conditional Analogy GAN: Swapping Fashion Articles on People Images". In: Oct. 2017, pp. 2287–2292. DOI: 10.1109/ICCVW.2017.269.

[24] Yanghua Jin et al. "Towards the Automatic Anime Characters Creation with Generative Adversarial Networks". In: (Aug. 2017).

[25] Tero Karras et al. "Progressive Growing of GANs for Improved Quality, Stability, and Variation". In: *ArXiv* abs/1710.10196 (2018).

[26] Diederik Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: Dec. 2014.

[27] Igor Kononenko. "Machine learning for medical diagnosis: History, state of the art and perspective". In: *Artificial intelligence in medicine* 23 (Sept. 2001), pp. 89–109. DOI: 10.1016/S0933-3657(01)00077-X.

[28] Shizuma Kubo, Yusuke Iwasawa, and Y. Matsuo. "Generative Adversarial Network-Based Virtual Try-On with Clothing Region". In: 2018.

[29] Christian Ledig et al. "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network". In: July 2017, pp. 105–114. DOI: 10.1109/CVPR.2017.19.

[30] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PP (July 2018), pp. 1–1. DOI: 10.1109/TPAMI.2018.2858826.

[31] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *ECCV*. 2014.

[32] Z. Liu et al. "DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 1096–1104.

[33] Feixiang Lu et al. *Fine-Grained Vehicle Perception via 3D Part-Guided Visual Data Augmentation*. Dec. 2020.

[34] Liqian Ma et al. "Pose Guided Person Image Generation". In: *NIPS*. 2017.

[35] Popescu Marius et al. "Multilayer perceptron and neural networks". In: *WSEAS Transactions on Circuits and Systems* 8 (July 2009).

[36] Takeru Miyato et al. "Spectral Normalization for Generative Adversarial Networks". In: Feb. 2018.

[37] G.R.S. Murthy and R.s Jadon. "Computer Vision Based Human Computer Interaction". In: *Journal of Artificial Intelligence* 4 (Apr. 2011), pp. 245–256. DOI: 10.3923/jai.2011.245.256.

[38] David Powers. "Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness Correlation". In: *Mach. Learn. Technol.* 2 (Jan. 2008).

[39] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: (Nov. 2015).

[40] J. Ray. "Faster R-CNN: Down the rabbit hole of modern object detections". In: (Jan. 2018).

[41] S. Reed et al. "Generative Adversarial Text to Image Synthesis". In: *ICML*. 2016.

[42] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (June 2015). DOI: 10.1109/TPAMI.2016.2577031.

[43] A. Rosebrock. *Intersection over Union (IoU) for object detection*. Nov. 2016.

[44] Negar Rostamzadeh et al. *Fashion-Gen: The Generative Fashion Dataset and Challenge*. June 2018.

[45] R. Rothe, M. Guillaumin, and L. Gool. "Non-maximum Suppression for Object Detection by Passing Messages Between Windows". In: *ACCV*. 2014.

[46] C. Shorten. *DCGANs (Deep Convolutional Generative Adversarial Networks*. Sept. 2018.

[47] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv 1409.1556* (Sept. 2014).

[48] Xibin Song et al. "ApolloCar3D: A Large 3D Car Instance Understanding Benchmark for Autonomous Driving". In: June 2019, pp. 5447–5457. DOI: 10.1109/CVPR.2019.00560.

[49] Jost Springenberg et al. "Striving for Simplicity: The All Convolutional Net". In: (Dec. 2014).

[50] G. Sreenu and M A Durai. "Intelligent video surveillance: a review through deep learning techniques for crowd analysis". In: *Journal of Big Data* 6 (June 2019), p. 48. DOI: 10.1186/s40537-019-0212-5.

[51] Kai Ming Ting. "Precision and Recall". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 781–781. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_652. URL: https://doi.org/10.1007/978-0-387-30164-8_652.

[52] Subeesh Vasu, Nimisha Madam, and A.N. Rajagopalan. *Analyzing Perception-Distortion Tradeoff using Enhanced Perceptual Super-resolution Network*. Nov. 2018.

[53] Paul Viola and Michael Jones. "Rapid Object Detection using a Boosted Cascade of Simple Features". In: vol. 1. Feb. 2001, pp. I–511. ISBN: 0-7695-1272-0. DOI: 10.1109/CVPR.2001.990517.

[54] Paul Viola and Michael Jones. "Robust Real-Time Face Detection". In: *International Journal of Computer Vision* 57 (May 2004), pp. 137–154. DOI: 10.1023/B:VISI.0000013087.49260.fb.

[55] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. "Generating Videos with Scene Dynamics". In: (Sept. 2016).

[56] Jiajun Wu et al. "Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling". In: (Oct. 2016).

[57] Fangting Xia et al. "Zoom Better to See Clearer: Human and Object Parsing with Hierarchical Auto-Zoom Net". In: vol. 9909. Oct. 2016, pp. 648–663. ISBN: 978-3-319-46453-4. DOI: 10.1007/978-3-319-46454-1_39.

[58] Seward Yildirim and Bergmann. *Disentangling Multiple Conditional Inputs in GANs*. June 2018.

[59] Y. Yin, J. P. Robinson, and Y. Fu. "Multimodal In-bed Pose and Shape Estimation under the Blankets". In: *ArXiv* abs/2012.06735 (2020).

[60] Han Zhang et al. *Self-Attention Generative Adversarial Networks*. May 2018.

[61] Han Zhang et al. "StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks". In: (Dec. 2016).

[62] B. Zhao et al. "Multi-View Image Generation from a Single-View". In: *Proceedings of the 26th ACM international conference on Multimedia* (2018).

[63] Shizhan Zhu et al. "Be Your Own Prada: Fashion Synthesis with Structural Coherence". In: Oct. 2017, pp. 1689–1697. DOI: 10.1109/ICCV.2017.186.

[64] Zhengxia Zou et al. "Object Detection in 20 Years: A Survey". In: *ArXiv* abs/1905.05055 (2019).