Udacity Machine Learning Engineer Nanodegree Capstone Report

Peter Bauer

October 28th 2020

Pedestrian Detection

Definition

Project Overview

As final project of the Udacity Machine Learning Engineer Nanodegree, a pedestrian detection system will be created. Therefore, the Penn-Fudan Database [3] is used.

Computer Vision plays a crucial role in state-of-the art autonomous vehicles. Especially in urban areas, an autonomous system must be able to perceive its environment and interact with it to prevent accidents. In general, Computer Vision is an interdisciplinary field of research, that is aiming to teach computers a high-level understanding from digital images or videos. Therefore, Computer Vision is concerned with the extraction, analysis and understanding of useful information from images [1].

The completion of this Capstone project is split into four parts:

1. Data Exploration and Generation
   - explore the Penn-Fudan Database and provide a function to properly load the data
   - create a sliding window video to showcase region generation for later classification
   - generate positive and negative samples out of the given images for classifier training
   - create training, testing and validation sets out of the generated image patches

2. The Classifier
   - define transforms, data loaders and necessary parameters (e.g. batch size, training epochs)
   - instantiate pretrained model and replace output layer
   - set “requires_grad” argument to false to update only last layer (feature extraction)
   - train the model and save best model weights
   - use the test set to perform an evaluation for unseen images

3. Pedestrian Detection Pipeline
   - get the input picture
   - use sliding window and image pyramids to extract regions of interest
   - classify regions of interest with the resnet50 architecture adapted for pedestrian detection
   - apply non maximum suppression to display best fitting bounding boxes

4. Benchmark Model and Evaluation
   - load and test the OpenCV people detector
   - load the detector created in this project
   - compare both detectors on 20 unseen test images and evaluate performance

Problem Statement

Pedestrian detection is an application area of Computer Vision and a non trivial task to solve. There are multiple difficulties that the final algorithm has to cope with. Some of them are changing light, weather and environmental conditions, others are the pedestrians size, its orientation and clothing as well as occlusion in the given image . Due to this varying parameters no hard coded algorithm for object detection can be utilized, either a system has to learn from data to obtain knowledge.

Although many modern systems use a sensor fusion including stereo vision and lidar technology, the object detection from a given input image is still the main component of most vision-based safety systems.

Metrics

Both, the person classifier and the detection system as a complete pipeline need to be evaluated. For the classifier an simple accuracy score can be used. This should be sufficient, since the non-pedestrian pictures can be created out of the given data.

$$\text{accuracy}(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N-1} 1(\hat{y}_i = y_i)$$

For the detection itself, a simple intersection over union (iou) approach is used. This means, that a detection is valid, if it overlaps with the ground-truth bounding box by a specified iou-value.

$$\text{iou} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Using this definition and a certain threshold value (e.g. 0.5) all proposed regions can be grouped into True-Positive, False-Positive, True-Negative and False-Negative. Most important are the correct detected pedestrians (True-Positive), patches that are marked as pedestrian without a person in it (False-Positive) and regions containing a person but not marked from the detector (False-Negative). Therefore, recall and precision are a good indicators for the model performance.

$$\text{recall} = \frac{tp}{tp + fp}$$

$$\text{precision} = \frac{tp}{tp + fn}$$

Where:

- $\hat{y}$ is the predicted value of the i-th sample
- y is the true value of the i-th sample
- tp (True Positive) is the number of correctly classified samples with category *positive*
- fp (False Positive) is the number of wrongly classified samples with category *positive*
- fn (False Negative) is the number of wrongly classified samples with category *negative*

Analysis

Data Exploration

As mentioned in the project overview, the data from the Penn-Fudan Database for Pedestrian Detection and Segmentation [3] is used. The complete dataset consists of 170 images with 423 marked pedestrians.

Folders in the zip-file of the dataset:

- Annotation Folder
  Text-files that contain useful information for each image
- PedMasks folder
  PNG images of masks for each picture (will not be used in this project)
- PNGImages folder
  Contains all pictures of the dataset in .PNG format
- Added-object-list.txt
  Text file that counts the number of detected persons in each picture
- Readme.txt
  General information

Exploratory Visualization

Figure 1 showcases an image from the dataset as well as the image with the corresponding bounding boxes in it. The goal of the detection tasks is to match these bounding boxes as close as possible.

Figure 1: Image from the dataset with and without bounding boxes for pedestrians

Figure 2 illustrates examples of image patches thar are feed into to classifier for training. The first two images are positive examples that should be classified as pedestrian. The images three and four are negative examples and should be rejected by the classifier.

Figure 2: Positive and negative samples created for classifier training

Algorithms and Techniques

The techniques and algorithms used in the project are already mentioned in the Project Overview. This section explains them in detail and discusses advantages and shortcomings of the chosen approaches.

Sliding Window and Image Pyramid

To extract regions of interest out of a given image a sliding window approach in combination with an image pyramid is used. This results in image patches of different scale and position. *Figure 3* shows some example image crops for a given input image. To save time in the processing pipeline, the window has roughly the shape of a pedestrian (height ≈ 2.75*width). The main advantage of this approach is its simple and intuitive implementation in code. Drawbacks are the parameters that have to be selected in advance (i.e. window size, stride size, pyramid scale factor, minimum image size) as well as the fixed window proportions (of course these could also vary, but this would lead to even longer detection times). A video for the described approach can be seen in the notebook "Data Exploration and Generation".

Figure 3: Regions of interest at different scale and location

Classification using a pretrained Neural Network

For the classification of the extracted image patches a retrained resent50 network is used. For the given task, the last layer is removed (originally classification into 1000 classes) and replaced with a fully connected layer containing only two classes (pedestrian or not). While training, only the weights of the last layer are updated. This transfer learning approach is called feature extraction and offers the possibility to get good results for a complex problem with a relatively small dataset. The model is implemented using the PyTorch framework and trained on the data generated in the *Data Exploration* notebook.

Non maximum suppression

To avoid multiple rectangles that denote only on person, a non maximum suppression is applied to the detections.

Benchmark

As a benchmark for the detection, the inbuild person detector of OpenCV can be used. The detector utilizes a histogram of oriented gradients in combination with a linear SVM. The model can easily be loaded and used. Therefore this approach provides a good ground truth for the detection task.

Methodology

Data Preprocessing

To access the all relevant parts of the data directly the DataClass.py file was created. The defined *PennFudanDataset* class inside the python file is able to load the images and their corresponding bounding boxes. An instance of this class is initialized in every notebook, which made it much easier to access the data and all of its components properly. Most of the code in this class is adapted from the PyTorch Object Detection Finetuning Tutorial [2], and only the outputs are changed to the specific demands of this project. The image patches fed into the classifier for training and testing are of course rescaled to a fixed input size before classification.

As already mentioned the dataset contains 170 images with 423 marked pedestrians. For the final evaluation of these images, 20 pictures are only used for detection testing. The remaining 150 images are prepared to train, validate and test the created classifier. To create the positive samples, all images are loaded and the bounding boxes containing a pedestrian are cropped out and rescaled to the classifier input size. The negative images are quiet harder to produce. First of all it was decided to create three times more negative samples then positives, to enlarge the number of training samples and train the classifier already on some given heuristics (there are more image patches without a pedestrian). To produce a negative sample, a random crop of the image is taken. If this crop does not lie inside one of the bounding boxes of the image (that denote a person) the image piece is stored as negative sample. Although this approach seems straightforward, the results for the detection had been quite moderate. Image pieces containing only a head or a leg of a pedestrian have already been classified as positive. Therefore, the detector was to sensitive. To counteract this issue, negative samples are created if the iou-value of the image crop and a bounding box in an image is below 0.45. This leads to a worse classification accuracy (drops from 96% to about 90%) but also makes the detector less sensitive to single features of a person.

Implementation

I began the implantation with the data exploration and the DataClass for effective handling of the dataset. As another step that could be helpful in future a function that can display the bounding boxes in the images was created. After this, the sliding window as well as the image pyramid have been implemented. To gain a better understanding of the approach a video was created inside the notebook.

As a following step, the data for the classifier had to be created. The positive and negative samples have been produced as explained in the *Data Preprocessing* section. For training, validation and testing the images got shuffled into the three corresponding folders with a split of 60% training data and respectively 20% for validation and testing containing altogether 1565 images.

In the second notebook the classifier is created and trained. As a first step the data transforms, as well as the important parameters of a neural network, such as batch size and number of epochs, are defined. The resnet50 model is loaded and the output shape of the network is changed to the desired two class problem. Furthermore, the “requires_grad” parameter is set to false to only re-train the last layer of the network. After those steps the network is ready to train again. As mentioned above the first models were trained on pictures containing either a complete pedestrian or none. This lead to a very sensitive model that recognized pedestrians even if there was only one a part of a person inside the window. The models trained later on tried to resolve this issue with pictures containing parts of pedestrians, but not a complete person.

The third notebook is about combining the created approaches of roi-generation and image classification to get the desired detector. The complete pipeline from end to end is executed in this notebook for the first time. At the end of the notebook some images and the detected bounding boxes are displayed.

The last notebook is about the benchmark model and the evaluation of the created detector. Both the benchmark model and the model created during this project get loaded and applied to an image. Since both models seem to work properly they get applied to the test dataset, to produce the desired output statistics.

Refinement

The first detection system that was built had a 0% iou-value for every training image of the classifier, and at the end of the pipeline there was not a non maximum suppression implemented (since the classifier did only output classes without corresponding probability values). Although the detections had pedestrians inside of the predicted regions, they have been to rough as *Figure 4* illustrates.

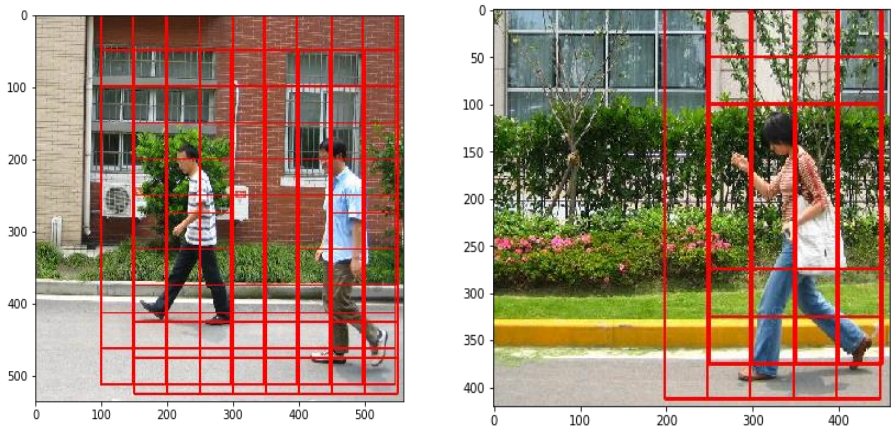

Figure 4: Initial detection results

To resolve this issues the iou-value for the negative training images was increased to 45%. Furthermore, a softmax function was implemented to make a "non maximum suppression of overlapping patches" possible . *Figure 5* demonstrates the improved results that could be obtained.

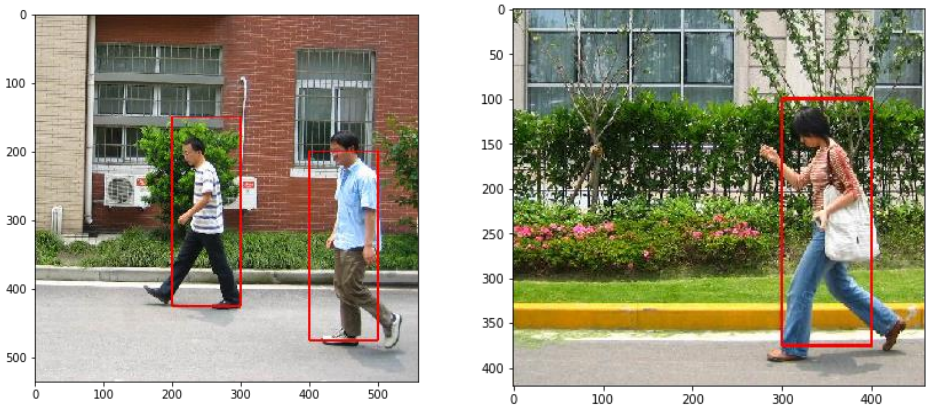

Figure 5: Final detection results

Results

Model Evaluation and Validation

Roi Generation

Having solved the issues discussed in the *refinement* section, there are still hyperparameters to determine for the final model. For the non maximum suppression, the overlap threshold has to be determined, the stride and window size for the sliding window has to be selected and the scale factor for the image pyramid must be chosen. After trying different combinations for the parameters, it became obvious that some kind of "trade off" is necessary for the choice of these values. A rather small stride size, in combination with a small window, as well as a fine scaling factor, results in a fairly long computation for the detection (since all the created images had to be classified by the neural network). Reducing the stride size from 50 to 25 pixels gave a 2.8 times slower detection, while a complete evaluation of the 20 pictures with a step size of 50 pixels in horizontal and vertical direction already took about three to four minutes.

As a final compromise between time consuming computation and model accuracy, a stride size of 50 pixels in width and height has been chosen (by an sliding window width of 100 pixels). This seemed to be small enough to detect most pedestrians quite accurately, and still had tolerable computation times (although far from real time).

For the overlap threshold for the non maximum suppression a quite small value of 0.2 was chosen. This somehow limits the capabilities of the detector to persons that have non overlapping bounding boxes, but yields back the best results in practice. In addition, pedestrians that occupy a fairly small region of the image (are in the background) are unlikely to be detected by the system or get detected as one person (although there are multiple per next to each other).

Neural Network

The resnet50 network was trained for 15 epochs with the training and validation loss shown in *Figure 6.* The final model uses the model trained for 12 epochs since that performed best on the training set (90% accuracy). After 12 epochs the validation loss increased again, what can be seen as an indicator of overfitting the training data. Furthermore, training and validation accuracy peak after 12 epochs of training. Since the training already took about 330min (5h 30min) it was decided to not train the model any further.

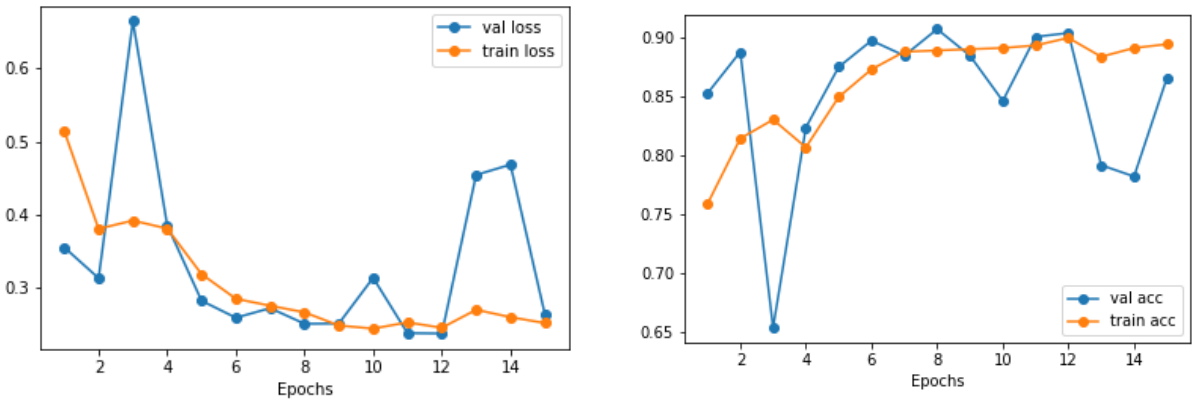

Figure 6: Training and validation accuracy/loss of the re-trained resnet50 architecture

Justification

In this project a simple pedestrian detector was created using traditional computer vision techniques such as the sliding window and the image pyramid in combination with nowadays state-of-the-art technologies like neural networks. At the beginning of the project 20 images have been put aside for later detection evaluation. Both models (the created and the OpenCV benchmark model) are used to predict pedestrians in the given images.

For a valid detection, the iou-value of the predicted and ground truth region have to overlap with a defined threshold. This threshold was chosen to be 0.5 for the benchmark and the created model. To compare both models properly the precision and recall values mentioned in the *Metrics* section had to be produced for the models.

To get the desired True-Positive, True-Negative and False-Negative numbers, an evaluation pipeline was developed. This pipeline detects in a first step, if there are predicted bounding boxes that do not overlap with any ground truth box. If this is the case, these boxes are denoted as False-Positive. In a second step, all boxes with a threshold below the chosen one (0.5 in this case) are marked as False-Positive. In addition, valid boxes that have the same corresponding ground truth are examined. The predicted box with the larger iou-value is retained, whilst the other box is rejected. Since this is not truly a False-Positive alarm, this special case is denoted as Double-Detection. The remaining predicted bounding boxes can be marked as True-Positive. Now that the True-Positive values are calculated, the False-Negatives can be easily gained by subtracting the number of predicted boxes from the actual number of ground truth boxes.

Figure 7 and *Figure 8* illustrate example results from both detectors.

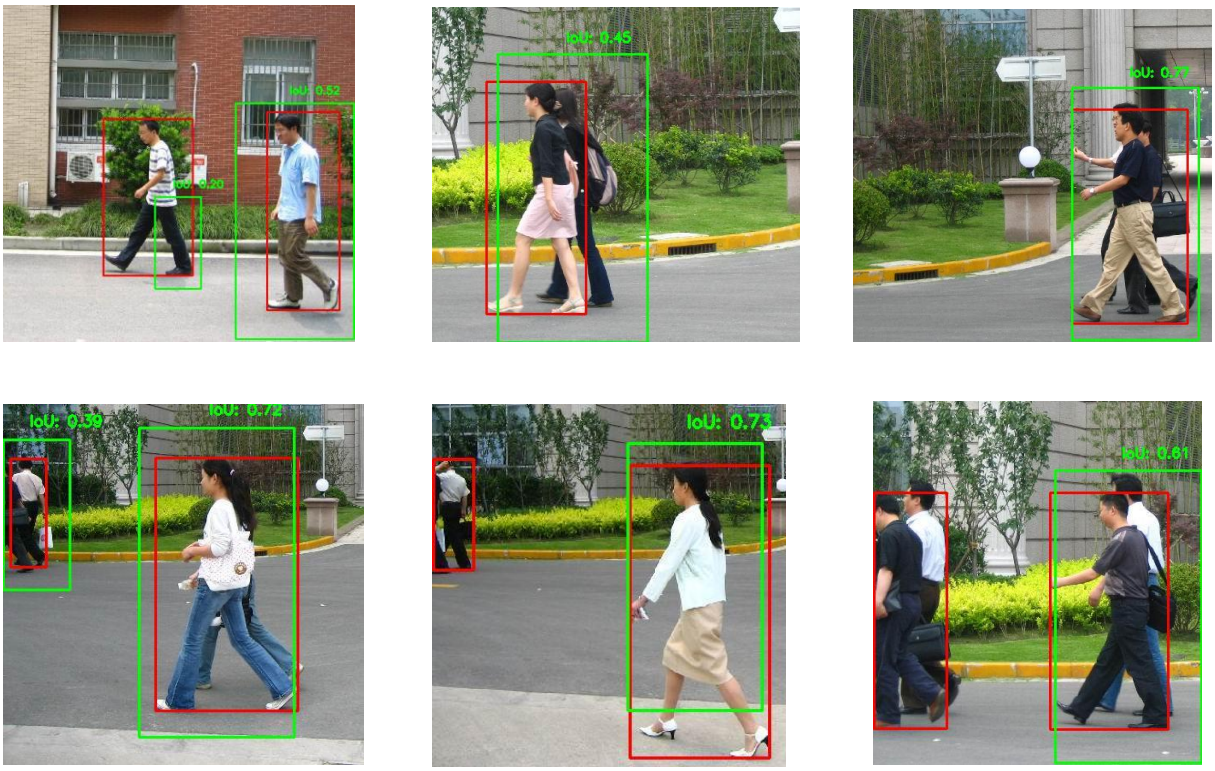

Figure 7: Benchmark model (example results)

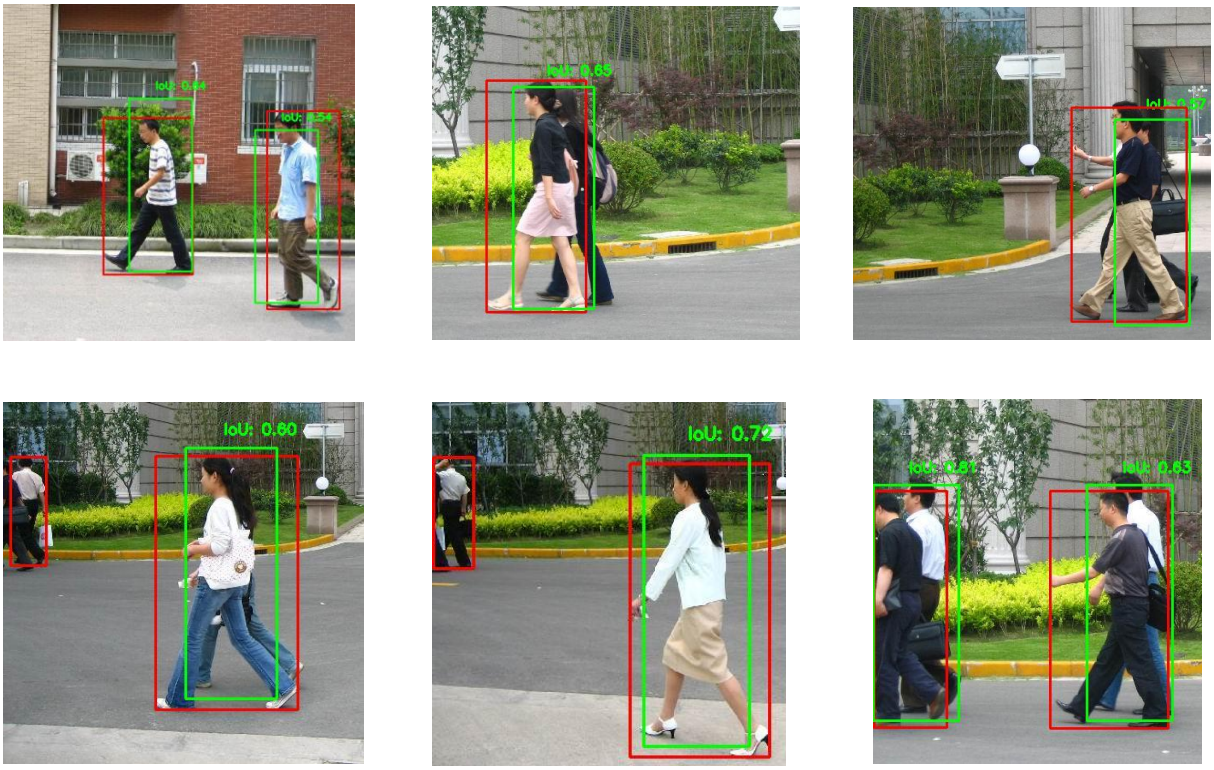

Figure 8: Created model (example results)

In *Figure 9* the detection results from both methods are listed, including all relevant metrics. Looking at the numbers, the chosen approach outperforms the OpenCV benchmark model and therefore the results are quite satisfying. However, there are also shortcomings of this approach. The computation time for the 20 test images was about 3 min 24 sec with a stride size of 50 pixels. Since we did not aim for real time performance this is a minor issue, but definitely a field that should be improved further. In comparison, the benchmark model needed only 7 seconds for the same task and is therefore much more efficient. To reduce the amount of time needed for a detection, there are several improvements that could be made to the model. For example selective search and a fast classifier cascade could be utilized to classify roi's faster and improve performance on negative images. With these approaches, the roi generation as well as the classification are able to speed up. However, the used sliding window together with the image pyramid is easy to program and very intuitive (why it was used in this project).

	Recall	Precision	Ture-Positive	False-Positive	False-Negative	Double-Detections
model	0.838710	0.81250	26.0	5.0	6.0	0.0
benchmark	0.515152	0.53125	17.0	16.0	15.0	0.0

Figure 9: Statistics for the evaluated models (iou-threshold 0.5)

Conclusion

The model crated in this project performed quite good on the test images and outperformed the chosen benchmark model. Although the results are satisfying, the models performance is far away from modern real time object detection systems using end to end trainable deep neural networks (Faster-RCNN[4], YOLO[5]). Nevertheless, the chosen approach to create an image classifier and use it as a pedestrian detector was successful.

References

[1] https://en.wikipedia.org/wiki/Computer_vision#:~:text=Computer%20vision%20is%20an%20interdisciplinary,human%20visual%20system%20can%20do.

[2] https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html

[3] https://www.cis.upenn.edu/~jshi/ped_html/

[4] https://arxiv.org/pdf/1506.01497.pdf

[5] https://arxiv.org/pdf/1506.02640.pdf