

Dokumentation für klassische und deep learning Methoden der Augmentation

Basic approaches

Erstellung der Pfade (Path creating script)

Zum ersten Schritt muss die *basic_img_augmentation.py* Datei im Ordner *basic_approaches* ausgeführt werden.

Nach der Programmausführung werden zwei json-Dateien erstellt, die aus der Liste der Pfade bestehen. Jede Pfad ist die Adresse, wo das jeweilige Bild platziert ist. Im ersten Fall sind Bilder und Masken mit dem Vordergrund als *citysc_fgPaths.json* benannt. Sie haben “person” Klasse.

Darüber hinaus bestehen diese Bilder aus Personen, mindestens eine davon grösser als *obj_bg_ratio* Verhältnis. Im zweiten Fall besteht die zweite Datei (*citysc_bgPaths.json*) aus den Bildern und Masken, die den Hintergrund bezeichnen und “ground”, “road” und/oder “sidewalk” Klasse/-n haben.

Erstellung des Datensatzes (Basic image augmentation script)

Allgemeine Beschreibung des Programms

Zum zweiten Schritt wird der Datensatz aus den klassisch augmentierten Bildern und entsprechenden Masken erstellt. Die Anzahl der Datensatzbilder muss festgelegt werden, die als *dataset_size* initialisiert ist. Zum Weiteren ist es die Pfade anzupassen, die für den zu erstellenden Datensatz (Bilder und Masken) und json-Dateien sind. Dann müssen die Listen mit Pfaden zu den Vordergrund- und Hintergrundbildern erstellt werden.

Zum Beispiel:

```
python      basic_img_augmentation.py      --dataset-size      11      --output-path
/home/admin1/Programming/HIWI_MRT/gp-gan_augmentation/basic_approaches/D
ataBase
                                         --fg
/home/admin1/Programming/HIWI_MRT/gp-gan_augmentation/basic_approaches/cit
```

```
ysc_fgPaths.json --bg  
/home/admin1/Programming/HIWI_MRT/gp-gan_augmentation/basic_approaches/cit  
ysc_bgPaths.json --process 4
```

Besonderheiten und Funktionen

Das Programm wählt ein zufälliges Bild und eine entsprechende Maske als Vordergrund- Vordergrundbild (Abb. 1) und ein anderes Bild und seine Maske als Hintergrund- Hintergrundbild (Abb. 2).



Abb. 1: Vordergrundbild (oben), Vordergrundmaske (unten)

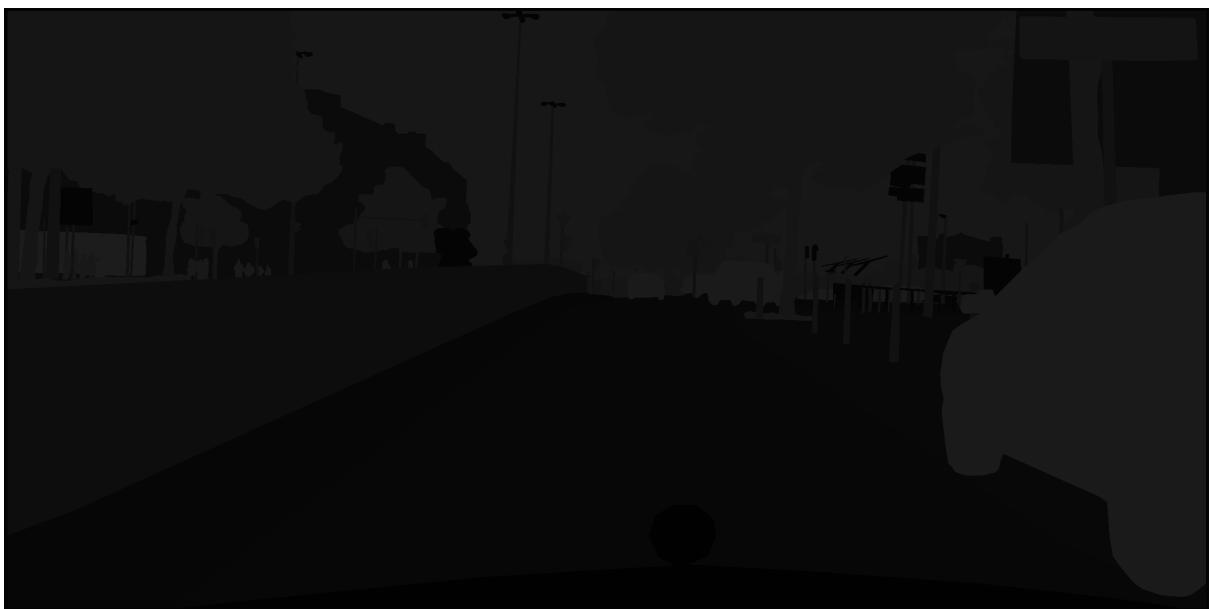


Abb. 2: Hintergrundbild (oben), Hintergrundmaske (unten)

Zum nächsten Schritt werden die Funktionen für die Augmentierung eingesetzt. Das gewählte Vordergrund-, Hintergrundbild und die entsprechenden Masken werden zufällig horizontal oder vertikal gespiegelt.

Auf dem Vordergrundbild werden die Objekte mit der Klasse "Person" erkannt und eine zufällige Person wird gewählt. Auf dem Hintergrundbild wird eine zufällige Position (innerhalb der Klassen "road", "ground" und "sidewalk") gefunden, worauf die ausgewählte Person angelegt wird. Vor dem Anlegen wird die Person entsprechend verkleinert oder vergrößert.

Die Größe der Person wird folgendermaßen ausgerechnet: Es wurde bemerkt, dass die Größe der Personen an den bestimmten Standorten auf dem Bild durchschnittlich einen Wert beträgt. Es wurde angenommen, dass die Größe einer Person im ersten Fall bei $y = 1000 \text{ px}$ 800 px beträgt. Im zweiten Fall ist eine Person bei $y = 540 \text{ px}$ 170 px groß. Das Gleichungssystem macht das deutlich:

$$h = y^*a + b$$

h – Größe einer Person (px)

y – Position (px)

$$800 = 1000^*a + b$$

$$170 = 540^*a + b$$

Bei der Lösung des Gleichungssystems können die Unbekannten a und b gefunden werden. Dann kann die Größe der Person für jede Position auf dem Bild ausgerechnet werden.

Da nicht jedes Bild (Abb. 3) im *cityscapes* Datensatz perfekt annotiert ist, wurde entschieden die Funktion *Mattig* (<https://github.com/pymatting/pymatting>) einzusetzen, die die entsprechenden Masken selbst generiert.



Abb. 3: Darstellung des cologne_000056_000019_leftImg8bit.png Bildes mit der angelegten Maske

Die *Matting* Funktion ermöglicht es, die genaueren Masken zu erstellen. Darüber hinaus werden die schrägen Kanten zwischen des Objekts und Hintergrunds mittels des *alpha*-Kanals beim Anlegen des Objekts (Vordergrundbild) auf den Hintergrund (Hintergrundbild) (Abb. 4) vermieden.



Abb. 4: Darstellung des Bildes, nachdem das Objekt auf den Hintergrund angelegt und die *Matting* Funktion eingesetzt wurde

Den entsprechenden Skript kann man in GitLab

https://gitlab.mrt.uni-karlsruhe.de/MRT/private/roesch/ius/image_data_augmentation.git

branch *add_pure_python_scripts* finden.

Problem von matting Funktion

Es wurde aber gemerkt, dass viele Bilder Artefakten haben. Sie produziert die matting Funktion. Das Problem ist, dass wir für matting Funktion TrimapMasken erstellen:

https://gitlab.mrt.uni-karlsruhe.de/MRT/private/roesch/ius/image_data_augmentation.git

branch *add_pure_python_scripts*

image_data_augmentation/basic_approaches/geometric_transformations.py

def trimap_creator(mask) (Abb.5)

```

227     def trimap_creator(mask):
228         kernel_value, iteration_value = blur_parameter_finder(mask)
229         eros_iter_value = iteration_value
230         dil_iter_value = iteration_value
231
232         kernel = np.ones((kernel_value, kernel_value), np.uint8)
233
234         copy_er_mask = mask.copy()
235         mask_erosion = cv2.erode(copy_er_mask, kernel, iterations=eros_iter_value)
236         mask_erosion = np.where(mask_erosion == 0, 0, 255).astype(np.uint8)
237
238         copy_dil_mask = mask.copy()
239         mask_dilation = cv2.dilate(copy_dil_mask, kernel, iterations=dil_iter_value)
240         mask_dilation = np.where(mask_dilation == 0, 0, 128).astype(np.uint8)
241
242         added_masks = cv2.add(mask_erosion, mask_dilation)
243
244     return added_masks

```

Abb. 5: Die *matting* Funktion

Ab und zu sind die TrimapMasken nicht passend. Aus diesem Grund treten Artefakten (Abb. 6) auf



Abb. 6: Darstellung der Artefakten, nachdem die *matting* Funktion ausgeführt wurde

Dann wurden die Masken auf die entsprechenden Bilder mittels der *mask_checking/mask_checking.ipynb* Funktion angelegt und geprüft, wie genau die Masken annotiert sind.

Es wurde festgestellt, dass die meisten Masken sieht gut annotiert sind. Es wurde entschieden, die matting Funktion nicht mehr zu benutzen, weil die Artefakten auftreten. Um die Kanten zwischen dem Vorder- und Hintergrund weich zu machen, setzen wir eine neue Funktion, die den *Alpha*-Kanal einführt (Abb. 7).

```
259
260 def borderBlender(foreground, background, alpha):
261     foreground = foreground / 255
262     background = background / 255
263
264     alpha = cv2.bitwise_not(alpha)
265     alpha = alpha / 255
266
267     return ((alpha * foreground + (1 - alpha) * background)*255).astype(np.uint8)
268
```

Abb. 7: Einführung des Alpha-Kanals in
basic_approaches/geometric_transformations.py

Im Endeffekt bekommen wir sehr gute Ergebnisse (Abb. 8):





Abb. 8: Darstellung eines generierten Bildes mittels der *Alpha*-Kanal

GP-GAN

Um Kontrast und Helligkeit des Vordergrundes anzupassen, benutzen wir die GP-GAN Methode (https://github.com/visitor9999th/Tensorflow_GP-GAN)

Generate images script

Für das Training des GP-GAN Netzes muss der Datensatz erstellt werden

Z.B:

```
python      GP-GAN/generate_images.py      --dataset_size      31      --dataset_dir  
'/home/admin1/CITYSCAPES_DATASET'                      --save_dir  
'/home/admin1/gp-gan_augmentation/GP-GAN/DataBase'
```

Als Ergebnis werden destination und source Bilder erstellt. Destination Bilder sind Original Bilder. Source Bilder sind augmentierte Bilder, wo die Personen auf dem Bild zufällige Kontrast und Helligkeit haben. Das Programm erstellt die Bilder (Abb. 9) mit dem *shape* 64*64 px. Ansonsten wird das Training viel länger dauert. Wenn man *shape* 256*256 px oder mehr macht, tritt der Fehler beim Training später auf.



Abb. 9: Destination Bild



Abb. 9: Source Bild

Write tf records script

Weiter erstellen wir *.tfrecords* Dateien für das Training aus dem Datensatz, der früher erstellt wurde.

Z.B.:

```
python           write_tf_records.py          --dataset_dir  
'/home/admin1/Programming/HIWI_MRT/gp-gan_augmentation/GP-GAN/DataBase/t  
rain_data'
```

Train blending gan script

Danach folgt das Training von GP-GAN

Z.B.:

```
python           GP-GAN/train_blending_gan.py      --train_data_root  
'/home/admin1/gp-gan_augmentation/GP-GAN/DataBase/train_data/train.tfrecords  
--val_data_root  
'/home/admin1/gp-gan_augmentation/GP-GAN/DataBase/train_data/val.tfrecords  
--save_folder 'output_path' --experiment "experiment_name"
```

Das Programm erstellt die Ordner *output_path/experiment_name*, wo deep learning Modelle und Daten vom Verlauf des Training für TensorBoard erstellt werden.

Alternativ können Sie die entsprechenden Ordner auch selbst erstellen und ein trainiertes neuronales Netzmodell dort ablegen (Ask Kevin Roesch, kevin.roesch@kit.edu):

```
gp-gan_augmentation/GP-GAN/output_path/experiment_name/GP-GAN_2021-06-1  
3-07-35-07.ckpt-215170410005.data-00000-of-00001  
gp-gan_augmentation/GP-GAN/output_path/experiment_name/GP-GAN_2021-06-1  
3-07-35-07.ckpt-215170410005.index  
gp-gan_augmentation/GP-GAN/output_path/experiment_name/GP-GAN_2021-06-1  
3-07-35-07.ckpt-215170410005.meta
```

Zur Ausführung schreibe im Terminal z.B:

```
python           GP-GAN/run_gp_gan.py          --generator_path  
'output_path/experiment_name/GP-GAN_2021-06-13-07-35-07.ckpt-215170410005'
```

Als Ergebniss werden neue Bilder und Masken (Abb. 10) erstellt



Abb. 10: Die generierten Bilder nach dem GP-GAN Prediction

Herausforderung und noch zu lösende Schwierigkeiten

Momentan generiert das aktuelle Programm ab und zu die Bilder und jeweilige Masken, die nicht realistisch aussehen.

Größe der Person nicht realistisch

In einigen Fällen sehen die Personen nicht realistisch (Abb. 11) aus, nachdem das Programm sie verkleinert oder vergrößert hat.

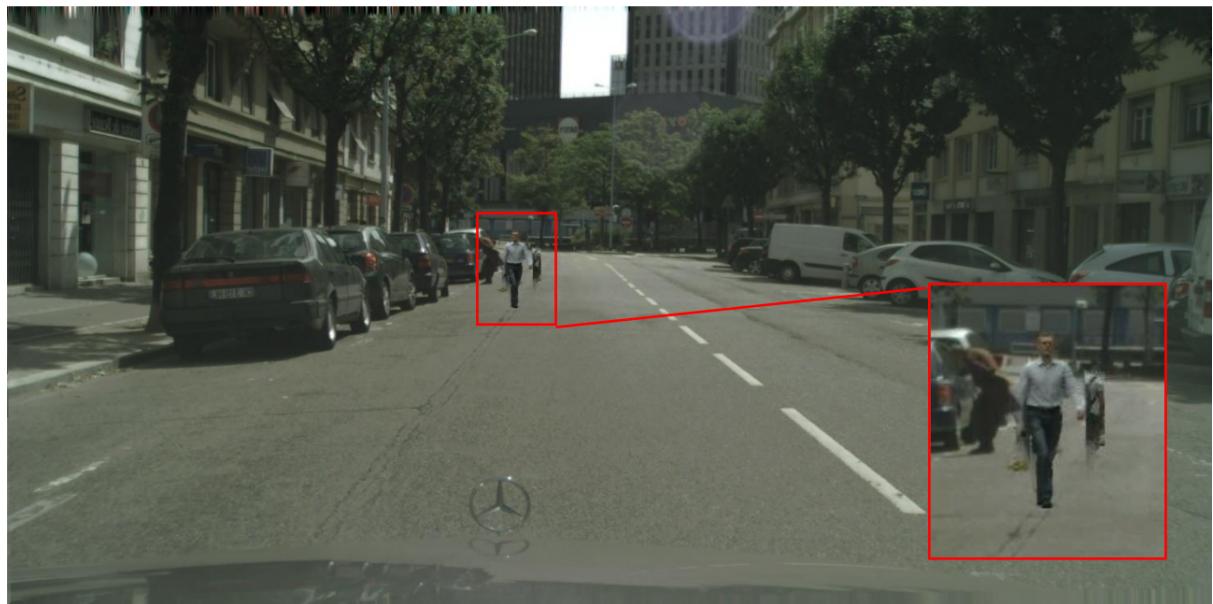


Abb. 11: Wegen ihrer Größe sieht die Person nicht realistisch aus

Verdeckung

Die angelegten Personen können die Objekte (Abb. 12) verdecken.

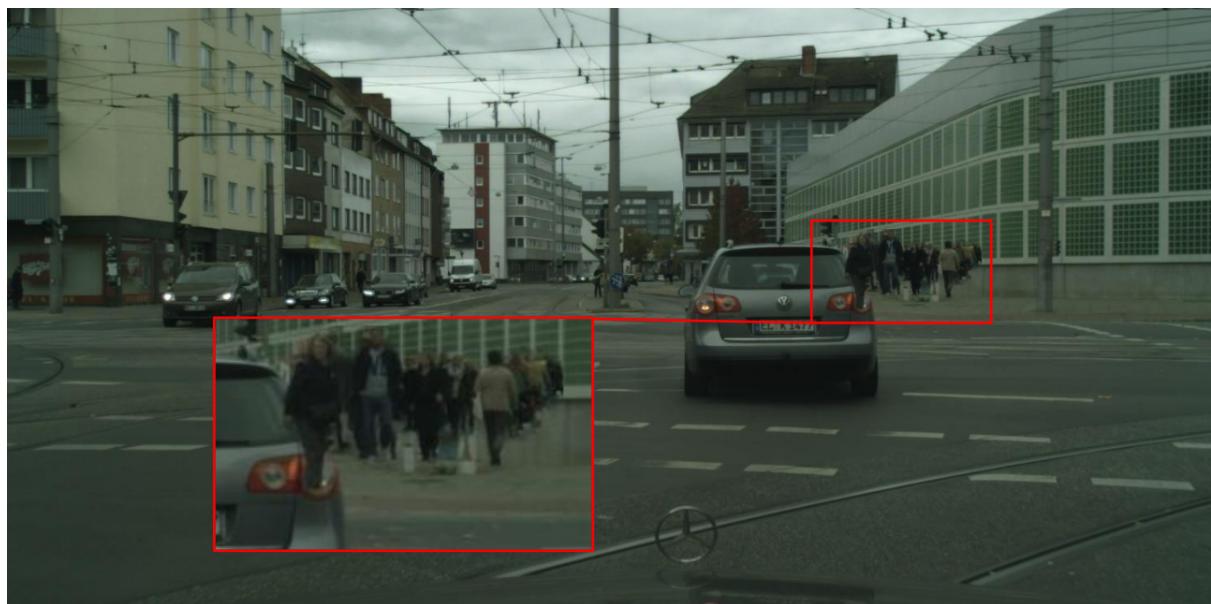


Abb. 12: Verdeckende Objekte (oben), verdeckte Objekte (unten)

Manchmal erstellt GP-GAN nicht realistische Bilder (Abb. 13)



Abb. 13: Das erstellte Bild von GP-GAN, das nicht realistisch aussieht