JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 16 (2020) 227–232
Received: 25 July 2020
Accepted: 3 September 2020

Blender as a tool for generating synthetic data

Blender jako narzędzie do generacji danych syntetycznych

Rafał Sieczka*, Maciej Pańczyk

Department of Computer Science, Lublin University of Technology, ul. Nadbystrzycka 38, 20-618 Lublin, Poland

Abstract

Acquiring data for neural network training is an expensive and labour-intensive task, especially when such data is difficult to access. This article proposes the use of 3D Blender graphics software as a tool to automatically generate synthetic image data on the example of price labels. Using the fastai library, price label classifiers were trained on a set of synthetic data, which were compared with classifiers trained on a real data set. The comparison of the results showed that it is possible to use Blender to generate synthetic data. This allows for a significant acceleration of the data acquisition process and consequently, the learning process of neural networks.

Keywords: artificial neural networks; convolutional neural network; synthetic data; blender

Streszczenie

Pozyskiwanie danych do treningu sieci neuronowych, jest kosztownym i pracochłonnym zadaniem, szczególnie kiedy takie dane są trudno dostępne. W niniejszym artykule zostało zaproponowane użycie programu do grafiki 3D Blender, jako narzędzia do automatycznej generacji danych syntetycznych zdjęć, na przykładzie etykiet cenowych. Przy użyciu biblioteki fastai, zostały wytrenowane klasyfikatory etykiet cenowych, na zbiorze danych syntetycznych, które porównano z klasyfikatorami trenowanymi na zbiorze danych rzeczywistych. Porównanie wyników wykazało, że możliwe jest użycie programu Blender do generacji danych syntetycznych. Pozwala to w znaczącym stopniu przyśpieszyć proces pozyskiwania danych, a co za tym idzie proces uczenia sieci neuronowych.

Słowa kluczowe: sztuczne sieci neuronowe; konwolucyjne sieci neuronowe; dane syntetyczne; blender

1. Wstęp

W ostatniej dekadzie sztuczne sieci neuronowe znalazły zastosowanie w wielu dziedzinach. Jedną z nich jest widzenie komputerowe (ang. computer vision), gdzie głębokie uczenie (ang. deep learning), wykorzystywane jest przy rozwiązywaniu wielu problemów z tej dziedziny [1] m.in. detekcja obiektów, rozpoznawanie aktywności, twarzy i szacowanie pozycji człowieka [2–5]. Trenowanie jak i testowanie głębokich sieci neuronowych jest czasochłonne oraz kosztowne, ponieważ należy zebrać dużą ilość danych, następnie ręcznie je opisać, aby wykorzystać je w uczeniu nadzorowanym. Jest to problem występujący nie tylko przy głębokim uczeniu ale w szeroko pojętym uczeniu maszynowym (ang. machine learning) [6], mimo że istnieją łatwo dostępne, gotowe zbiory danych np.: VGG-Sound [7], RoadText [8], PandaSet [9], czy też COVID-CT [10]. Może się jednak okazać, że taki zbiór nie pasuje do problemu, który osoba trenująca próbuje rozwiązać, ponieważ zbiór jest zbyt mały lub słabej jakości aby uzyskać zadowalające wynik, bądź względy prawne nie pozwalają go wykorzystać. Obiecującym rozwiązaniem jest użycie automatycznie wygenerowanych danych syntetycznych. W zależności od rodzaju danych jakie są potrzebne - dane tabelaryczne, zdjęcia, wideo, dźwięk - proces generowania takich danych - jeśli możliwy, może być różny.

W przypadku wizji komputerowej, generacja danych syntetycznych, może odbywać się przy pomocy silników gier, bądź programów do grafiki 3D takich jak Unity 3D, Unreal Engine, Cinema4D, Blender [11]. Umożliwia to automatyzację procesu generowania i opisywania danych potrzebnych do treningu sieci. Przy użyciu tego typu programów powstało kilka zbiorów danych syntetycznych [12–22] zdjęć i filmów. W większości przypadków, te zbiory danych są kosztowne w wygenerowaniu. Wymagają od osoby tworzącej sceny, aby ta w szczegółach odpowiadała określonemu środowisku. Te zbiory danych były z powodzeniem używane do rozwiązywania geometrycznych problemów takich jak przepływ optyczny, sceniczny i szacowania pozycji kamery.

Dane syntetyczne zostały wykorzystane z sukcesem w pracy [23], gdzie zostało przedstawione użycie danych syntetycznych podczas trenowania modeli, do rozpoznawania tekstu na zdjęciach przedstawiających sceny ze-

*Corresponding author
Email address: rafal.sieczkaa@gmail.com (R. Sieczka)

wnętrzne. Dane syntetyczne w tym przypadku generowane są jako tekst nakładany na zdjęcia ze zbioru ICDAR 2003 i SVT. Podobne rozwiązanie zostało przedstawione w [24], jednak tym razem na zdjęcia nakładane są obiekty, a nie tekst. W tym celu zostały wykorzystane darmowe modele 3D CAD. Po wyrenderowaniu modeli z teksturami, w różnych pozach, obiekt zostaje naniesiony na różne tła. Na tak przygotowanych zdjęciach zostały wytrenowane detektory obiektów. Kolejne dwie prace [25, 26] także skupiają się na detekcji obiektów, używając przy tym Blendera i Unity 3D. Rozwiązanie wykorzystujące Blendera, podczas renderowania używa silnika Cycles, aby wyrenderowane obiekty były bardziej zbliżone do rzeczywistych obiektów, wpływa to jednak na prędkość generacji. Dane syntetyczne używane były również przy szacowaniu pozycji człowieka. W tym celu został stworzony zbiór realistycznych danych syntetycznych [27]. Istnieją jeszcze inne prace [28–32], które skupiają się na generacji danych syntetycznych lub ich wykorzystaniu do treningu modeli sieci neuronowych.

Przedstawione powyżej rozwiązania w dużej mierze skupiają się na uzyskaniu jak najbardziej realistycznych zdjęć obiektów, pomieszczeń, co odbija się na prędkości generacji takich danych. Przy małej liczbie zdjęć nie stanowi to dużego problemu, jednak kiedy potrzebujemy setek tysięcy zdjęć do treningu, ten czas się kumuluje. Skupiają się też one na jak najdokładniejszym odwzorowaniu otoczenia, w którym znajduje się obiekt, co również jest czasochłonne. Aby zaadresować te problemy, w tym artykule został przedstawiony sposób generowania danych syntetycznych, wykorzystując do tego program Blender, WeasyPrint, python-barcode. Do sprawdzenia użyteczności wygenerowanych danych, przygotowanym programem, zostały wytrenowane klasyfikatory etykiet cenowych na zbiorze treningowym danych syntetycznych i rzeczywistych. Przedstawione rozwiązanie wykorzystuje silnik renderujący Eevee, który zapewnia szybszy czas renderowania, w porównaniu do silnika Cycles. Otoczenie etykiet cenowych jest bardzo proste, aby nie tracić czasu na jego modelowanie.

2. Generacja danych i trening

W tej sekcji zostały opisane kroki w generacji danych syntetycznych etykiet cenowych oraz sposób w jaki zostały pozyskane dane rzeczywiste użyte w zbiorze ewaluacyjnym i testowym. Dodatkowo został opisany sposób trenowania sieci neuronowych z użyciem biblioteki fastai [33].

2.1. Generacja danych z wykorzystaniem Blendera

Przy tworzeniu programu do generacji danych syntetycznych pod uwagę brana była modularność projektu, aby w łatwy sposób móc dodawać generację innych obiektów niż etykiety cenowe. Dlatego obiekty, które są używane podczas generacji nie znajdują się w tym samym pliku blend co scena, w której są generowane. Pozwala to na łatwą zmianę obiektów, bez zaśmiecania głównej sceny oraz na wymianę sceny, jeśli np.: potrzebne są zdjęcia na zewnątrz lub wewnątrz budynku. Dla każdego typu obiektu tworzony jest osobny plik blend, w którym znajdują się obiekty tylko danego typu np.: samochody, książki, biurka itd. Pozwala to na łatwiejsze zarządzanie obiektami.

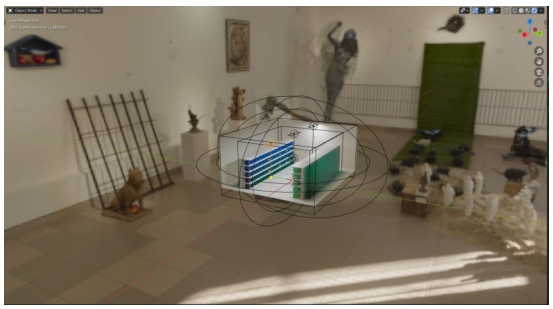
Aby było wiadomo, gdzie znajduje się jaki typ obiektu i mieć możliwość definiowania specjalnych zmiennych, zamiast ustawiać je na sztywno w kodzie, stworzony został plik json, w którym znajduje się konfiguracja. Przykład zawartości konfiguracji znajduje się na Listingu 1. W pliku json, niezależnie od typu obiektu znajdują się pola: type - typ obiektu, object_name - nazwa obiektu w pliku blend, blend_file - ścieżka do pliku blend, gdzie znajduje się model obiektu. Przy generacji wykorzystywany jest Eevee, który jest silnikiem renderującym czasu rzeczywistego. Skraca to znacząco czas potrzebny na wyrenderowanie zdjęcia nawet prostej sceny.

Listing 1: Przykładowy plik konfiguracyjny

```
{
"objects": [
    {
        "type": "price_label",
        "object_name": "electronic_label_1
            ",
        "template_file": "
            electronic_label_1.html",
        "css_file": "electronic_label_1.
            css",
        "price_label_type": "electronic",
        "blend_file": "etykiety.blend",
        "barcode": "ean8",
        "barcode_options": {
            "dpi": 300,
            "module_height": 3.0,
            "quiet_zone": 0.5,
            "write_text": false
        }
    }
],
"renderer_settings": {
    "engine": "evee",
    "width": 1920,
    "height": 1080,
    "samples": 64
}
}
```

Scena, w której renderowane były zdjęcia etykiet cenowych, została przedstawiona na Rysunku 1. W scenie została użyta mapa hdr jako tło. Scena składa się z prostego pomieszczenia, w którym znajdują się dwa regały sklepowe. Na regałach znajdują się różnego koloru sześciany. Zostały użyte 3 źródła światła typu Area.

Tekstury etykiet cenowych są generowane przy użyciu WeasyPrint, który potrafi zamienić plik html i css w zdjęcie. Dzięki temu możliwe jest stworzenie wielu różnych układów etykiet cenowych. Pozwala to również na



Rysunek 1: Scena, która była użyta przy generacji syntetycznych zdjęć etykiet cenowych

umieszczanie losowych wartości w polach etykiety takich jak: nazwa produktu, cena, waga itp. Kody kreskowe generowane są przy pomocy biblioteki python-barcode.

Proces generacji zdjęć etykiet cenowych można podzielić na 3 kroki.

1. Inicjalizacja programu
   W tym kroku, program wczytuje plik konfiguracyjny i tworzone są niezbędne obiekty generatorów, dla każdego typu obiektu. Zadaniem generatora jest wyrenderowanie zdjęć danego obiektu. Do pliku sceny zostają również dołączone wymagane obiekty z innych plików blend.
2. Tworzenie tekstur
   W tym kroku wybierana jest losowo jedna z etykiet cenowych. Dla wybranej etykiety cenowej, wczytywany jest plik html oraz css i na ich podstawie generowana jest tekstura etykiety cenowej. Przykładowa tekstura została przedstawiona na Rysunku 2.

Rysunek 2: Przykładowa wygenerowana tekstura etykiety elektronicznej

3. Rednering zdjęcia
   Kiedy tekstura zostanie wygenerowana, wczytywana jest ona do programu blender i przypisywana do odpowiedniego materiału. Dany obiekt jest ustawiany w wyznaczonym miejscu, kamera zostaje ustawiona w losowym miejscu, w ustalonej odległości od obiektu, po czym wykonywany jest rendering zdjęcia. Po wyrenderowaniu, zdjęcie zostaje przycięte, w taki sposób aby etykieta zajmowała jak najwięcej miejsca na zdjęciu. Po zapisaniu zdjęcia, proces powraca do kroku 2, aż nie zostanie wygenerowana zadana liczba zdjęć.

Przykładowe wygenerowane zdjęcia różnych klas etykiet cenowych, zostały przedstawione na Rysunku 3.

Rysunek 3: Przykład wygenerowanych etykiet cenowych

2.2. Trenowanie modeli z wykorzystaniem biblioteki fastai

Zostały wytrenowane po 3 modele na zbiór danych syntetycznych i rzeczywistych (DenseNet121, ResNet50, VGG19) przy użyciu biblioteki fastai. Zbiór treningowy danych syntetycznych składa się z 5000 zdjęć, które były generowane w rozdzielczości 768x768px, zbiór danych ewaluacyjnych z 2016 zdjęć rzeczywistych, zbiór testowy z 1008 zdjęć rzeczywistych a zbiór danych rzeczywistych z 3070 zdjęć. Każdy ze zbiorów został podzielony na 3 klasy etykiet - paper, electronic i handwritten. Zdjęcia etykiet rzeczywistych zostały wycięte ze zdjęć półek sklepowych.

Modele zostały wytrenowane przy użyciu polityki 1 cyklu. Były one trenowane na zdjęciach ze zwiększającą się rozdzielczością - 32x32px, 64x64px i 128x128px. Na zdjęciach w zbiorze treningowym wykonane zostały augmentacje: crop_pad, flip_lr, symmetric_wrap, zoom, brightness oraz contrast. W czasie treningu, najlepszy punkt kontrolny był zapisywany i każdy kolejny trening na większej rozdzielczości zdjęć, dla danego zbioru i modelu, był wznawiany z tego punktu kontrolnego.

3. Wyniki

Dane syntetyczne zostały wygenerowane 10 razy po 5000 zdjęć, aby sprawdzić średni czas generacji zdjęć. Dla 5000 zdjęć średni czas generacji wyniósł 7507 sekund, czyli 2 godziny i 8 minut, natomiast średni czas renderowania jednego zdjęcia - używając do tego karty graficznej GTX 1660 Ti - wyniósł 1 sekundę, gdzie dla porównania, średni czas renderowania jednego zdjęcia, z użyciem silnika cycles, wyniósł 1 minutę i 2 sekundy. Podczas renderowania zdjęć, przy użyciu silnika cycles, wpływ na szybkość renderingu ma wiele ustawień, najważniejszymi z nich są ilość próbek i wielkość części zdjęcia, które w danej chwili jest renderowane. Mniejsza liczba próbek przyśpieszy renderowanie, jednak zdjęcie będzie gorszej jakości - bardziej ziarniste. Natomiast największy wpływ na czas generacji jednego zdjęcia miało renderowanie oraz tworzenie tekstury etykiety, dlatego całkowity czas generacji zdjęcia etykiety to suma czasu potrzebnego na generację tekstury i renderowanie zdjęcia.

W tabeli 1 zostały przedstawione procentowe wyniki dokładności sklasyfikowania etykiet cenowych, uzyskane na zbiorze testowym dla klasyfikatorów trenowanych na

danych syntetycznych i rzeczywistych.

Tabela 1: Procentowy wynik dokładności sklasyfikowania etykiet cenowych dla modeli trenowanych na danych syntetycznych i rzeczywistych

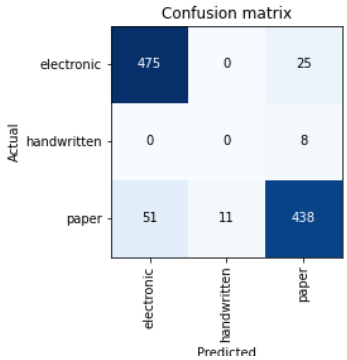
Zbiór danych	Architektura	Wynik precyzji na zbiorze testowym
Dane syntetyczne	ResNet50	91.5%
	DenseNet121	90.6%
	VGG19	70.5%
Dane rzeczywiste	ResNet50	98.9%
	DenseNet121	98.4%
	VGG19	98.1%

Różnica pomiędzy najlepszym klasyfikatorem, który został wytrenowany na danych rzeczywistych, a klasyfikatorem, który został wytrenowany na danych syntetycznych, wynosi 7.4%. Klasyfikatory ResNet50 oraz DenseNet121 trenowane na syntetycznych zdjęciach, uzyskały zbliżone wyniki, które można by polepszyć, dostosowując hiperparametry uczenia. Podczas treningu zmieniany był tylko współczynnik nauki (ang. Learning rate). Najgorszy wynik uzyskał klasyfikator oparty o architekturę VGG19.

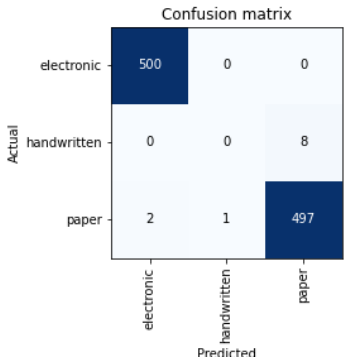
Mimo, że do renderowania etykiet nie był używany silnik Cycles, który renderuje zdjęcia, bardziej zbliżone do rzeczywistości, wyniki uzyskane przez najlepszy klasyfikator, trenowany na danych syntetycznych nie jest mniejszy od klasyfikatora trenowanego na danych rzeczywistych o 10%.

Patrząc na matrycę pomyłek dla najlepszego modelu trenowanego na danych syntetycznych (Rysunek 4) i matrycę pomyłek najlepszego modelu trenowanego na danych rzeczywistych (Rysunek 5), widać, że obydwa klasyfikatory, ani razu nie sklasyfikowały dobrze etykiet klasy handwritten. Przy klasyfikatorze trenowanym na danych rzeczywistych, wynika to z małej liczby zdjęć etykiet klasy handwritten w zbiorze treningowym. Przy klasyfikatorze trenowanym na danych syntetycznych, była wystarczająca liczba zdjęć tej klasy. Ten błąd wynika z tego, że syntetyczne dane etykiet klasy handwritten, za bardzo przypominały etykiety paper. Wpływ na to ma rodzaj użytej czcionki, która nie różni się znacząco od czcionek użytych przy etykietach paper.

Wpływ na wyniki uzyskane dla klasyfikatorów, trenowanych na danych syntetycznych ma również nałożenie tekstur na obiekty etykiet. Tekstury generowały się z losową nazwą produktu, ceną, kodem kreskowym itd. więc zdarzało się, że niektóre tekstury źle nakładały się na obiekty. Kolejnym czynnikiem mającym wpływ na wynik jest odwzorowanie układów występujących w rzeczywistości. Może się zdarzyć, że dany układ etykiety występuje rzadko w danych rzeczywistych, jednak w danych syntetycznych, występuje z częstotliwością zbliżoną do innych układów etykiet. W tym przypadku widać to na etykietach typu handwritten, których w zbiorach składających się z danych rzeczywistych jest bardzo mało, w porównaniu do innych klas etykiet, natomiast w zbiorze syntetycznym, występują porównywalnie często co inne klasy etykiet. Powoduje to, że taki klasyfikator ma większą szansę, źle sklasyfikować etykietę jako handwritten.



Rysunek 4: Matryca pomyłek dla modelu ResNet50 trenowanego na danych syntetycznych



Rysunek 5: Matryca pomyłek dla modelu ResNet50 trenowanego na danych rzeczywistych

4. Podsumowanie i wnioski

Przedstawiona metoda generacji danych syntetycznych przy użyciu Blendera, kiedy używamy do renderowania silnika Eevee, umożliwia szybką generację danych do treningu sieci neuronowych. Dzięki temu proces zbierania i opisywania danych staję się znacząco szybszy, co pozwala na szybsze dostarczanie wytrenowanych modeli.

Sieć wytrenowaną na danych syntetycznych, można użyć w dalszej części procesu, kiedy to duża liczba danych rzeczywistych będzie dostępna. Trzeba jednak zauważyć, że wytrenowanie sieci na danych syntetycznych, może się wiązać z większą ilością kroków, potrzebną do uzyskania wymaganych wyników.

Patrząc na uzyskane wyniki, można stwierdzić, że użycie silnika Eevee - szybszego ale renderującego mniej realistyczne zdjęcia - do renderowania syntetycznych zdjęć, pozwala na uzyskanie zadowalających wyników i znacząco skraca czas potrzebny na renderowanie.

Klasyfikator VGG19 uzyskał najgorszy wynik, kiedy został wytrenowany na danych syntetycznych, ponieważ jest to architektura, która posiada najwięcej parame-

trów, co może sugerować, że architektury posiadające bardzo dużą liczbę parametrów, uczą się gorzej na danych syntetycznych.

Literatura

[1] A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis, Deep learning for computer vision: A brief review, Computational intelligence and neuroscience (2018).

[2] Z. Cao, G. Martinez, T. Simon, S. Wei, Y. A. Sheikh, Openpose: Realtime multi-person 2d pose estimation using part affinity fields. IEEE Transactions on Pattern Analysis and Machine Intelligence (2019) 7291-7299.

[3] T Simon, H Joo, I Matthews, Y Sheikh, Hand keypoint detection in single images using multiview bootstrapping, CVPR (2017) 1145-1153.

[4] Z Cao, T Simon, S EnWei, Y. Sheikh, Realtime multi-person 2d pose estimation using part affinity fields, CVPR (2017) 7291-7299.

[5] S. En Wei, V. Ramakrishna, T. Kanade, Y. Sheikh, Convolutional pose machines, CVPR (2016) 4724-4732.

[6] Y. Roh, G. Heo, S. E. Whang. A survey on data collection for machine learning: a big data – ai integration perspective, IEEE Transactions on Knowledge and Data Engineering (2019).

[7] C. Xie, L. Vedaldi, P. Zisserman. Vgg-sound: A largescale audio-visual dataset (2020).

[8] S. Reddy, M. Mathew, L. Gomez, M. Rusinol, D. Karatzas., C. V. Jawahar, Roadtext-1k: Text detection and recognition dataset for driving videos, 2020 IEEE International Conference on Robotics and Automation (2020) 11074-11080.

[9] S. Hesai, Pandaset - public large-scale dataset for autonomous driving.

[10] J. Zhao, Y. Zhang, X. He, P. Xie. Covid-ct-dataset: a ct scan dataset about covid-19. arXiv preprint arXiv:2003.13865 (2020).

[11] Blender Online Community. Blender - a 3D modelling and rendering package. Blender Foundation, Stichting Blender Foundation, Amsterdam (2018).

[12] A.Tsirikoglou, J. Kronander, M. Wrenninge, J. Unger, Procedural modeling and physically based rendering for synthetic data generation in automotive applications, arXiv preprint arXiv:1710.06270 (2017).

[13] A. Gaidon, Q. Wang, Y. Cabon, E. Vig, Virtual worlds as proxy for multi-object tracking analysis, proceedings of the IEEE conference on computer vision and pattern recognition (2016) 4340-4349.

[14] M. Muller, V. Casser, J. Lahoud, N. Smith, B. Ghanem, Sim4cv: A photo-realistic simulator for computer vision applications, International Journal of Computer Vision, 126(9) (2018) 902-919.

[15] J. McCormac, A. Handa, S. Leutenegger, A. J. Davison, Scenenet rgb-d: 5m photorealistic images of synthetic indoor trajectories with ground truth, arXiv preprint arXiv:1612.05079 (2016).

[16] Y. Zhang, W. Qiu, Q. Chen, X. Hu, A. Yuille, Unrealstereo: Controlling hazardous factors to analyze stereo vision, in proceedings of International Conference on 3D Vision (3DV) (2018) 228-237.

[17] W. Qiu, A. Yuille, Unrealcv: Connecting computer vision to unreal engine, in proceedings of European Conference on Computer Vision (2016) (909-916).

[18] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, T. Brox, A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation, in Proceedings of the IEEE conference on computer vision and pattern recognition (2016) 4040-4048.

[19] P. Fischer, A. Dosovitskiy, E. Ilg, P. Hausser, C. Hazırba¸s, V. Golkov, P. van der Smagt, D. Cremers, T. Brox, Flownet: Learning optical flow with convolutional networks, in Proceedings of the IEEE international conference on computer vision (2015) 2758-2766.

[20] S. R. Richter, V. Vineet, S. Roth, V. Koltun, Playing for data: Ground truth from computer games, in European conference on computer vision (2016) 102–118.

[21] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, A. M. Lopez, The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes, IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016) 3234–3243.

[22] D. J. Butler, J. Wulff, G. B. Stanley, M. J. Black, A naturalistic open source movie for optical flow evaluation, in proceedings of Computer Vision – ECCV (2012) 611–625.

[23] M. Jaderberg, K. Simonyan, A. Vedaldi, A. Zisserman, Synthetic data and artificial neural networks for natural scene text recognition, arXiv preprint arXiv:1406.2227 (2014).

[24] X. Peng, B. Sun, K. Ali, K. Saenko. Learning deep object detectors from 3d models, in Proceedings of the IEEE International Conference on Computer Vision (2015) 1278-1286.

[25] P. S. Rajpura, H. Bojinov, R. S. Hegde, Object detection using deep cnns trained on synthetic images, arXiv preprint arXiv:1706.06782 (2017).

[26] K. Wang, F. Shi, W. Wang, Y. Nan, S. Lian, Synthetic data generation and adaption for object detection in smart vending machines, arXiv preprint arXiv:1904.12294 (2019).

[27] G. Varol, J. Romero, X. Martin, N. Mahmood, M. J. Black, I. Laptev, C. Schmid. Learning from synthetic humans, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2017) 109–117.

[28] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, S. Birchfield, Training deep networks with synthetic data: Bridging the reality gap by domain randomization, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (2018) 969–977.

[29] C. Mitash, K. E. Bekris, A. Boularias, A self-supervised learning system for object detection using physics simulation and multi-view pose estimation, in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2017) 545-551.

[30] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, P. Abbeel, Domain randomization for transferring deep neural networks from simulation to the real world, in proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2017) 23-30.

[31] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, A. M. Lopez, The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes, IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016) 3234–3243.

[32] H. Hattori, V. N. Boddeti, K. M. Kitani, T. Kanade, Learning scene-specific pedestrian detectors without real data, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2015) 3819–3827.

[33] J. Howard, S. Gugger. fastai: A layered api for deep learning, Information 11(2) (2020) 108.