

Járművek trajektóriájának előrejelzése machine learning modellekkel

PÉTER BENCE MÉRNÖKINFORMATIKA BSC 6. FÉLÉV*, Széchenyi István Egyetem, Hungary

DR. HORVÁTH ANDRÁS, Széchenyi István Egyetem, Hungary

AGG ÁRON PHD HALLGATÓ, Széchenyi István Egyetem, Hungary



Az ITS (intelligent transportation system) egyre nagyobb teret hódít napjainkban és rengeteg különböző területen alkalmazzák ezeket a rendszereket. A közlekedési csomópontok elemzése egy frekvenciált terület az ITS alkalmazásában. Célunk, gépi látás és gépi tanulás felhasználásával, közlekedési csomópontok elemzésének automatizálása és felgyorsítása. A kutatásban lefektetett alapgondolatokat, kifejlesztett keretrendszert és a felmerülő problémák megoldásait, a gyakorlatban balesetek megelőzésére, renitens viselkedések kiszűrésére és forgalomirányító rendszerek támogatására lehet

Authors' addresses: Péter Bence Mérnök informatika BSc 6. félév, Széchenyi István Egyetem, Győr, Hungary; Dr. Horváth András, Széchenyi István Egyetem, Győr, Hungary; Agg Áron PhD hallgató, Széchenyi István Egyetem, Győr, Hungary.

2023. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in , <https://doi.org/10.1145/nnnnnnn.nnnnnnn>.

használni. A kutatásban egy trajektória osztályozó módszert ismertettünk, amely objektumdetektálás és objektumkövetés segítségével elemzi a közlekedési csomópontokban elhaladó járművek mozgását. A mozgásuk alapján klaszterezi a trajektóriákat, majd gépi tanulás segítségével predikciót ad az újonnan belépő járművek kilépési pontjára. A módszerhez 6 különböző közlekedési csomópontban készített saját videó adatbázisunkat használtuk fel. A tesztelt klaszterezési mód-szerek közül (OPTICS, BIRCH, KMeans, DBSCAN) az OPTICS algoritmus bizonyult legjobbnak trajektórák klaszterezésére. Összehasonlí-tottunk több különböző klasszifikációs módszert a legpontosabb predikció eléréséhez, amelyek: KNN, SVM, GP, DT, GNB, MLP, SGD. A tanul-mányban bemutatott eljárások közül az SVM adta a legpontosabb 90%-os eredményt.

ACM Reference Format:

Péter Bence Mérnök informatika BSc 6. félév, Dr. Horváth András, and Agg Áron PhD hallgató. 2023. Járművek trajektóriájának előrejelzése machine learning modellekkel. 1, 1 (March 2023), 6 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

CONTENTS

Abstract	1
Contents	2
1 Bevezetés	2
2 Kapcsolódó kutatások	2
2.1 YOLO	2
2.2 DeepSORT	3
3 Adathalmazok kialakítása	3
3.1 Adatstruktúra	3
3.2 Objektumdetektálás	4
3.3 Objektumkövetés	4
4 Klaszterezés	4
4.1 Adattisztítás	4
4.2 Feature vektorok	4
4.3 Klaszterezési algoritmusok	4
4.4 Paraméterek kiválasztása	4
5 Klasszifikáció	4
5.1 Multiclass	4
5.2 Binary	4
5.3 OneVsRest	4
5.4 Machine Learning modellek	4
5.5 Feature vektorok	4
5.6 Pontosság mérése	5
References	5

1 BEVEZETÉS

A városok növekedése egyre nagyobb forgalomhoz vezet, ami a balesetek, forgalmi dugók számát növeli és a levegő minősége is romlik. Az ITS (intelligent transportation system) fejlesztése a városokban erre megoldást jelenthet. Ez magába foglalja az információs és kommunikációs technológiák, mint például szenzorok, kamerák, kommunikációs hálózatok és adat elemzés fejlesztését. 5G hálózatokon keresztül, ezek a technológiák összeköthetők a közlekedési eszközökkel. Ehhez okos forgalomirányítási rendszerek kifejlesztésére van szükség, amik információval tudnak szolgálni a járművekbe szerelt informatikai rendszereknek. A legértékesebb információt a közlekedésben résztvevő járművek jelen és jövőbeli pozíciója jelenti. Pontos és gyors trajektória előrejelző rendszerek kifejlesztése egy nagy kihívás és egyre növekszik irántuk a kereslet. E kutatási terület kiforratlanságából eredően, kevés létező keretrendszer és adathalmaz található, így a tanító adathalmaz gyűjtése, adatok kinyerésének formátuma, tárolása és mérőszámok kifejlesztése (amivel a tesztelni kívánt modellek pontosságát tudjuk mérni) is a kutatáshoz tartoznak. Ebben a kutatásban erre a problémára törekszünk egy módszertant és keretrendszert kifejleszteni, emellett klaszterezési és klasszifikációs algoritmusokat tesztelni. A tanító adatok előállításához, objektumok detektálására a YOLOv7 [Wang et al. 2022] konvolúciós neurális hálót használtuk, ez a konvolúciós neurális háló architektúra nem csak nagy pontosságot hanem sebességet is nyújt nekünk. Emellett képkockáról képkockára követni is kell tudni a detektált objektumokat. Erre is sok megoldás található

manapság, erre a feladatra a DeepSORT [Wojke and Bewley 2018] nevezetű algoritmust használtuk, ez kálmán filtert és konvolúciós neurális hálót használ az objektumok követésére. A tanító adatok 6 különböző helyszín forgalmát tartalmazzák. Minden helyszín más tulajdonságokkal bír, ezért nem lehet generalizálni a tanítási folyamatot, nem lehet egy univerzális modellt betanítani ami minden közlekedési helyszínre alkalmazható egyaránt. A klaszterezés során megpróbáljuk minél pontosabban meghatározni a be és kimeneti pontok által leírt klasztereket, amelyek majd alapul szolgálnak a klasszifikáció tanítása során. Több fajta klaszterezési algoritmust megvizsgáltunk a kutatás során, KMeans, OPTICS [Ankerst et al. 1999], BIRCH [Zhang et al. 1996] és DBSCAN [Ester et al. 1996][Schubert et al. 2017]. A klasszifikációhoz bináris klasszifikációs modelleket kombinálunk, így több klasszos klasszifikációs modelt kapunk. Minden bináris modelnél, egy klassz az összes többivel szemben van betanítva. A modellek pontosságának kiértékelésére 3 mérőszámot alkalmaztunk, amik az *Accuracy Score*, *Balanced Accuracy Score* [Brodersen et al. 2010] és *Top-k Accuracy Score*. Mindegyik mérőszám kiszámolásához *K-Fold Cross-Validation* [Anguita et al. 2012] metódust alkalmaztunk, ahol $K = 5$.

2 KAPCSOLÓDÓ KUTATÁSOK

Sok ITS-el kapcsolatos kutatásban tárgyalják a forgalom folyás (traffic flow) előrejelzését. [Paul et al. 2017] összehasonlítja az eddig kutatott és használt modellek, mint például Kalman Filtering, k-nearest neighbor (k-NN), mesterséges neurális hálók, stb., pontosságát és sebességét, ezen modellek tovább-kutatását, mivel egyre növekednek a különböző szenzorok által begyűjtött traffic flow adatok, így ez a terület belépett a *Big Data* korszakába. [Rossi et al. 2021] is a traffic flow előrejelzését és generálását tárgyalja, Floating Car Data (FCD) adathalmazokon betanított, Hosszú-Rövid-Távú memóriájú és Generatív versengő hálókkl.

2.1 YOLO

”You look only once” (YOLO) egy state-of-the-art, valós idejű objektum detektáló rendszer. Legfrissebb változata a YOLOv7 felülmúlja sebességben és pontosságban a modern konvolúciós hálókat (lásd 1 2 3). Beágyazott rendszerekben és videokártyákon is egyaránt jó a teljesítménye, ezért az ITS területén alkalmazható.

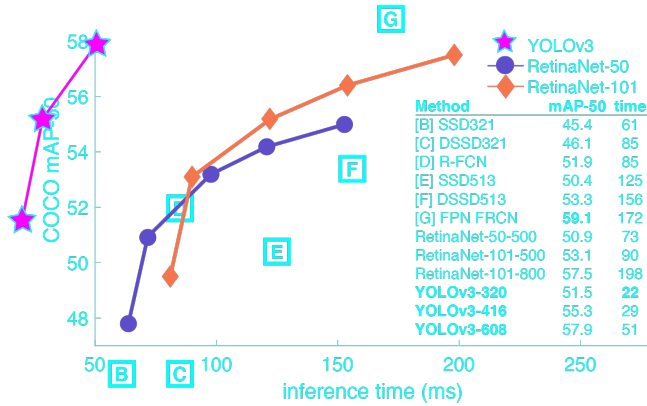


Fig. 1. YOLOv3 Performance

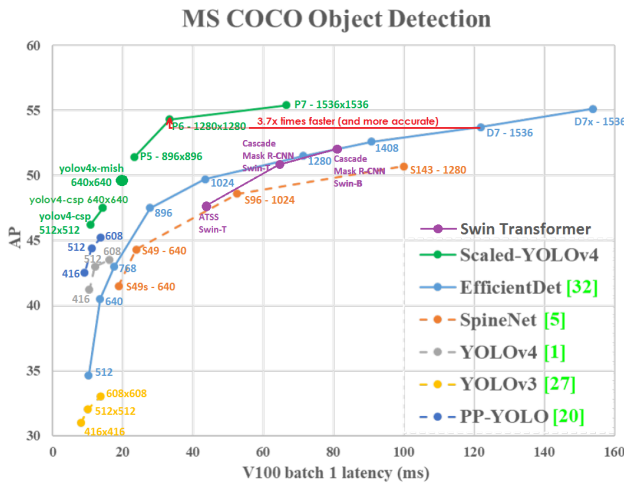


Fig. 2. YOLOv4 Performance

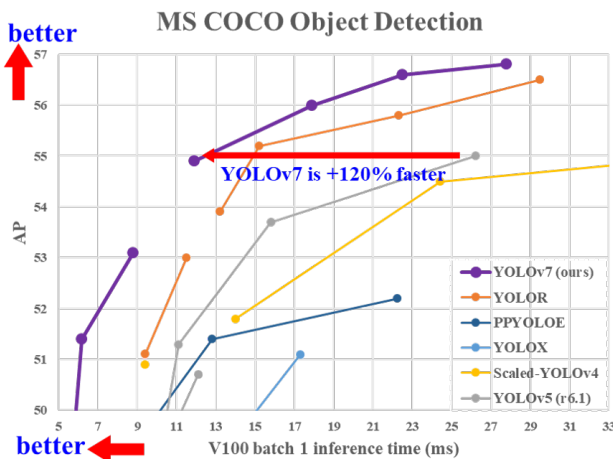


Fig. 3. YOLOv7 Performance

2.2 DeepSORT

Simple Online and Realtime Tracking with a Deep Association Metric (DeepSORT) a SORT algoritmus egy továbbfejlesztett változata. Az eredeti algoritmus sebességét törekednek növelni egy offline tanítási lépéssel. A betanított mély asszociációs metrika segítségével nagyban lecsökkentették az identitás váltásokat, és hosszabb elfedés után is követni tudja az objektumokat.

3 ADATHALMAZOK KIALAKÍTÁSA

A kutatás során saját adathalmazok kialakítására volt szükség. Az adatok begyűjtésére és eltárolására saját alkalmazást és keretrendszert fejlesztettünk ki. A szoftver keretrendszer python nyelven írtuk meg, a forráskód ezen a linken megtalálható http://github.com/Pecneb/computer_vision_research. A fejlesztés során a következő programkönyvtárakat használtuk OpenCV [Bradski 2000], Numpy [Harris et al. 2020], Pandas [pandas development team 2020], Scikit-Learn [Pedregosa et al. 2011], Matplotlib [Hunter 2007], SQLite [Hipp 2020], Joblib [Joblib Development Team 2020]. Az adathalmazokat SQLite adatbázisban és joblib fájlokban tároltuk el. Azért döntöttünk így, hogy két féle módon is eltároljuk az adathalmazokat, mert az SQL adatbázist univerzálisan bármilyen adatbáziskezelővel, vagy más programnyelvvél be lehet olvasni, viszont a joblib fájlokat sokkal gyorsabban be lehet tölteni pythonnal.

3.1 Adatstruktúra

Az adatstruktúrát SQL schema-ként, és python class-ként is definiáltuk.

```
CREATE TABLE IF NOT EXISTS objects (
    objID INTEGER PRIMARY KEY NOT NULL,
    label TEXT NOT NULL
);

CREATE TABLE IF NOT EXISTS detections (
    objID INTEGER NOT NULL,
    frameNum INTEGER NOT NULL,
    confidence REAL NOT NULL,
    x REAL NOT NULL,
    y REAL NOT NULL,
    width REAL NOT NULL,
    height REAL NOT NULL,
    vx REAL NOT NULL,
    vy REAL NOT NULL,
    ax REAL NOT NULL,
    ay REAL NOT NULL,
    FOREIGN KEY(objID) REFERENCES objects(objID)
);

CREATE TABLE IF NOT EXISTS metadata (
    historyDepth INTEGER NOT NULL,
    futureDepth INTEGER NOT NULL,
    yoloVersion TEXT NOT NULL,
    device TEXT NOT NULL,
    imgsize INTEGER NOT NULL,
    stride INTEGER NOT NULL,
```

```
confidence_threshold REAL NOT NULL,
iou_threshold REAL NOT NULL
);
```

Minden követett objektum egyedi azonosítóval lett ellátva. Az objektumhoz tartozó detektálások külön táblába lett kiszervezve, ahol az *objID* idegen kulcsal kapcsoljuk az *objektumok* táblához. Egy objektumhoz az egyedi azonosítón kívül tartozik egy *label* amit a YOLO objektum detektálótól kap, ez lehet pl. autó, személy, teherautó, stb. Az objektumokhoz tartozó detektálások tartalmazzák a képkocka számát, amikor a detektálás történt, a konfidenciát, hogy mennyire biztos az objektumfelismerő a hozzárendelt *label*-ben, az objektum *X, Y* koordinátáját, az objektum szélességét és magasságát, sebességét és gyorsulását. Ezek mellett még a konfigurációs adatokat is külön táblában tároljuk, hogy később meg lehessen ismételni a detektálást.

3.2 Objektumdetektálás

Az objektumdetektáláshoz a fent említett YOLO modellt használtuk. Kutatásunk kezdetekor, a YOLO 4-es verziójával kezdtünk dolgozni, de később átváltottunk a jobb pontosságot és sebességet ígérő 7-es verzióra.

3.2.1 YOLOv4. YOLO 4-es verzióját, C-ben implementálták. Hogy fel tudjuk használni, írunk kellett egy python API-t, ami meg tudunk hívni a detektáló programunkban.

3.2.2 YOLOv7. A YOLOv7 viszont már python-ban implementálták amihez már sokkal könnyebb volt API-t programozni és használni. Emellett, gyorsaságban és pontosságban is felülmúlta a 4-es verziót (lásd 3).

3.3 Objektumkövetés

3.3.1 DeepSORT.

4 KLASZTEREZÉS

A klaszterezés segítségével lehet az adathalmazból alóállítani a klaszifikáció alapjául szolgáló klasszokat. Ahhoz, hogy az a rengeteg trajektóriából és detektálásból számunkra felhasználható információ keletkezzen, meg kell határoznunk feature vectorokat, amik a trajektóriákra jellemző értékeket tartalmaznak. Ebben a feature térben fogja a klaszterező algoritmus megtalálni az egymáshoz közeli, hasonló trajektóriákat.

4.1 Adattisztítás

A klaszterezés előtt a nyers adatokat fel kell dolgoznunk, hogy az esetleges hibás, zajos detektálások, trajektóriák miatt kapjunk fals klasztereket.

4.1.1 Trajektóriák kiválasztása. Az objektum detektálás és követés nem tökéletes, rossz fényviszonyok, hosszabb eltakarások miatt a trajektóriák megszakadhatnak ezért ki kell választani az egyben maradt trajektóriákat. Első soron kiválasztjuk azokat, amelyek belépési és kilépési pontjai a kép relatív széleitől egy meghatározott határértékre vannak.

4.1.2 DeepSORT pontatlanság. Kutatásunk során azt tapasztaltuk, hogy a DeepSORT és a YOLO pontatlanságai felerősítik egymást. A YOLO hajlamos néha táblákat vagy rendőrlámpákat autóknak nézni, és ekkor a DeepSORT is elkezd követni. Egy olyan hibáját is felfedeztük a DeepSORT-nak, hogy egy objektumról áttapad a követés egy másik objektumra, ami fals trajektóriákat hoz létre. A DeepSORT-nak lehet finomhangolni a paramétereit, ami nem bizonyult akkora javulásnak, ezért utólagos szűréssel kellett korrigálnunk ezt a hibát. ...

4.2 Feature vektorok

Klaszterezéshez 4 és 6 dimenziós feature vektorokat használtunk. A 6 dimenziós vektorokat a DeepSORT hibájának a kiszűrésére hoztuk létre, de a kifejlesztett szűrő hatékonyságban bizonyult, ezért maradtunk a 4 dimenziós feature vektorok mellett. (lásd ...)

4.3 Klaszterezési algoritmusok

A legjobb eredményeket az OPTICS [Ankerst et al. 1999] algoritmus adta. ...

4.4 Paraméterek kiválasztása

A megfelelő paraméterek kiválasztása a klaszterezéshez igen fontosnak bizonyult. Ezt a gyűjtött adathalmazokon kézzel kellett finomhangolnunk. ...

5 KLASZIFIKÁCIÓ

5.1 Multiclass

A több klasszos klaszifikálás egy olyan feladat, amikor több mint 2 klassz van, és minden feature vektor csak egy klasszba tartozhat.

5.2 Binary

A bináris klaszifikáció a többklasszossal szemben, csak 2 klassz között dönt. 2 klassz között sokkal pontosabban el lehet dönteni, hogy a feature vektor melyikbe tartozik.

5.3 OneVsRest

Bináris klaszifikációs modellek kombinációjából, egy több klasszos modellt állítottunk össze, ami ... %-al pontosabb eredményt adott.

5.4 Machine Learning modellek

Kutatásunk során több klaszifikációs modellt is teszteltünk, ezek közül a két legjobban teljesítő algoritmust tárgyaljuk, KNN és SVM modellt. ...

5.5 Feature vektorok

Ahogy klaszterezésnél is, fontos meghatározni egy olyan feature vektort, ami a legjobban jellemzi a trajektóriát. 7 fajta feature vektorral teszteltük a kiválasztott klaszifikációs modelleket, a legjobban az 1. és 7. verzió teljesített. ...

5.5.1 Adatdúsítás. Hogy minél pontosabban reprezentáljuk a valódi idejű futást, a gyűjtött trajektóriákból nem trajektóriánként egy future vektort generáltunk. Ezeknek a számosságát a feature vektort felépítő algoritmusban szabtuk meg.

5.6 Pontosság mérése

A pontosság mérésére háromféle metrikát használtunk.

Accuracy Score. Ha \hat{y}_i az i . minta predikciója és y_i a hozzátartozó valódi érték, akkor az eltalált predikciók és összes predikció hányadosa, amit így lehet leírni:

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

Balanced Accuracy. amit ezért használtunk, hogy az adathalmaz kiegyensúlyozatlansága miatt ne kapjunk fals pontosságot. Ha minden klasszra egyenlően jól teljesít a klasszifikációs modellünk, akkor a sima *Accuracy*-t kapjuk vissza. Ha a teszt adathalmaz kiegyensúlyozatlansága miatt az egyik klassznak jobb a pontossága mint egy másiknak, akkor ezt az értéket elosztja a klasszok számával. Ha az y_i a valódi értéke az i . mintának, és w_i a hozzátartozó súly, akkor ezt a súlyt a következőképpen korrigáljuk:

$$\hat{w}_i = \frac{w_i}{\sum_j 1(y_j = y_i) w_j} \quad (2)$$

ahol $1(x)$ a karakterisztikus függvény. Adott a \hat{y}_i predikció az i . mintának, így a balanced accuracy-t így definiálhatjuk:

$$\text{balanced-accuracy}(y, \hat{y}, w) = \frac{1}{\sum \hat{w}_i} \sum_i 1(\hat{y}_i = y_i) \hat{w}_i \quad (3)$$

Top-K Accuracy. az *Accuracy Score* egy generalizált változata. A különbség az, hogy a predikció akkor számít igaznak, ha beletartozik a k legmagasabb valószínűségű predikciók közé. Ha $\hat{f}_{i,j}$ a i . mintának a j . legmagasabb predikciója, és y_i a hozzátartozó valódi predikció, akkor az eltalált predikciók és az összes minta hányadosát így lehet definiálni:

$$\text{top-k accuracy}(y, \hat{f}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \sum_{j=1}^k 1(\hat{f}_{i,j} = y_i) \quad (4)$$

ahol k a megengedett találgatások száma, és $1(x)$ a karakterisztikus függvény.

5.6.1 Adathalmaz szétválasztás. A pontosság méréséhez el kell választanunk egy teszt adathalmazt a tanító adathalmaztól, mivel ha azon az adathalmazon tesztelünk, amin tanítottunk akkor tökéletes pontosságot kapnánk eredményül. Ezt *Overfitting*-nek hívják. A szétválasztást egy általunk implementált algoritmus végzi, ami véletlen szám generátort használ a minták kiválasztásához. Az algoritmusnak meg lehet adni paraméterként a tanító adathalmaz méretét, és egy *seed* értéket, ami azért fontos, hogy meg lehessen ismételni a szétválasztást.

5.6.2 Cross Validation. A túltanítás elkerülése érdekében, alkalmaztunk egy elterjedt metódust a cross-validációt. ...

5.6.3 Teszthalmazos validáció. Hogy meggyőződjünk arról, hogy biztosan nem tanítottuk túl a modellünket, egy olyan teszt adathalmazon is le kell tesztelnünk, amit nem használtunk fel cross-validáció alatt.

5.6.4 Modellek tárolása. A betanított modelleket joblib file-ként tároltuk el, amit később python-nal tudunk betölteni.

REFERENCES

- D. Anguita, Luca Ghelardoni, Alessandro Ghio, L. Oneto, and Sandro Ridella. 2012. The 'K' in K-fold Cross Validation. In *The European Symposium on Artificial Neural Networks*.
- Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. 1999. OPTICS: Ordering Points to Identify the Clustering Structure. *SIGMOD Rec.* 28, 2 (jun 1999), 49–60. <https://doi.org/10.1145/304181.304187>
- G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).
- Kay Henning Brodersen, Cheng Soon Ong, Klaas Enno Stephan, and Joachim M. Buhmann. 2010. The Balanced Accuracy and Its Posterior Distribution. In *Proceedings of the 2010 20th International Conference on Pattern Recognition (ICPR '10)*. IEEE Computer Society, USA, 3121–3124. <https://doi.org/10.1109/ICPR.2010.764>
- Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (Portland, Oregon) (KDD '96)*. AAAI Press, 226–231.
- Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020), 357–362. <https://doi.org/10.1038/s41586-020-2649-2>
- Richard D Hipp. 2020. SQLite. <https://www.sqlite.org/index.html>
- J. D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. <https://doi.org/10.1109/MCSE.2007.55>
- Joblib Development Team. 2020. *Joblib: running Python functions as pipeline jobs*. <https://joblib.readthedocs.io/>
- The pandas development team. 2020. *pandas-dev/pandas: Pandas*. <https://doi.org/10.5281/zenodo.3509134>
- Anand Paul, Naveen Chilamkurti, Alfred Daniel, and Seungmin Rho. 2017. Chapter 8 - Big Data collision analysis framework. In *Intelligent Vehicular Networks and Communications*, Anand Paul, Naveen Chilamkurti, Alfred Daniel, and Seungmin Rho (Eds.). Elsevier, 177–184. <https://doi.org/10.1016/B978-0-12-809266-8.00008-9>
- F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
- Luca Rossi, Andrea Ajmar, Marina Paolanti, and Roberto Pierdicca. 2021. Vehicle trajectory prediction and generation using LSTM models and GANs. *PLOS ONE* 16, 7 (07 2021), 1–28. <https://doi.org/10.1371/journal.pone.0253868>
- Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. 2017. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Trans. Database Syst.* 42, 3, Article 19 (jul 2017), 21 pages. <https://doi.org/10.1145/3068335>
- Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. 2022. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv preprint arXiv:2207.02696* (2022).
- Nicolai Wojke and Alex Bewley. 2018. Deep Cosine Metric Learning for Person Re-identification. In *2018 IEEE Winter Conference*

- on Applications of Computer Vision (WACV)*. IEEE, 748–756.
<https://doi.org/10.1109/WACV.2018.00087>
- Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* (Montreal, Quebec, Canada) (*SIGMOD '96*). Association for Computing Machinery, New York, NY, USA, 103–114. <https://doi.org/10.1145/233269.233324>