Deteksi Begal untuk Senjata Api dan Senjata Tajam Pendek dengan SSD MobileNet

Abdullah Hadi
Telkom University
School of Computing
abdullahhadi@student.telkomuniversity.ac.id

Otniel Abiezer
Telkom University
School of Computing
otnielabiezer@student.telkomuniversity.ac.id

Rachmat Dwi Putra
Telkom University
School of Computing
rachmatdp@student.telkomuniversity.ac.id

Abstrak

Begal merupakan kejahatan yang sering terjadi di negara kita. Kejahatan ini merugikan karena dapat mengakibatkan kerugian harta sampai dengan kehilangan nyawa. Untuk itu diperlukan deteksi begal untuk menghindari hal tersebut. Salah satu cara deteksi begal yaitu dengan melakukan deteksi objek berupa senjata yang dibawa oleh begal tersebut seperti pisau dan pistol. Salah satu algoritma deteksi objek yang terkenal adalah SSD MobileNet. Model ini cukup cepat dalam melakukan deteksi objek dan cukup ringan untuk dibawa-bawa (mobile). Kami membangun sistem deteksi begal dengan menggunakan SSD MobileNet dengan input berapa videostream dari CCTV dan output berupa bunyi alarm jika terdeteksi pisau atau pistol. Hasil dari sistem yang kami bangun didapatkan nilai akurasi 0.820, rerataan IoU sebesar 0.612, serta mAP COCO dengan threshold IoU=0.5 memiliki nilai sebesar 0.875.

Kata kunci : begal, deteksi object, mobilenet, single shot detector, tensorflow object detection api, senjata api, senjata tajam

1. Pendahuluan

Kejahatan dengan senjata merupakan hal yang tidak lepas, terutama pada negara kita. Kejahatan tersebut sering disebut sebagai begal. Begal bisa terjadi kapanpun dan dimanapun. Oleh karena itu, diperlukan bantuan teknologi untuk membantu masalah begal.

Salah satu tanda jika begal itu terjadi, yaitu adanya senjata tajam atau senjata api. Oleh karena itu, kami akan mendeteksi dua objek tersebut. Begal sering terjadi di jalanan, sistem yang akan kami buat diharapkan akan dicoba deploy pada wilayah tersebut.

Di sini kami memberikan solusi untuk adanya sistem deteksi begal yang memfokuskan pada alat senjata api dan senjata tajam pendek, di mana jika sistem melihat adanya senjata tersebut maka akan dibunyikan suatu suara warning. Selain suara warning, hasil lokalisasi berupa bounding box pada video juga diperlukan. Hal ini bertujuan untuk menindaklanjuti suatu kejadian begal yang telah terjadi. Diperlukan pula bukti yang menunjukkan dimana senjata itu berada dan untuk kebutuhan forensic suatu lokalisasi objek kriminal.

2. Penelitian Terkait

Beberapa implementasi terkait masalah yang ingin diselesaikan, seperti pada penelitian [1] yang membuat sistem deteksi object terhadap senjata api dan api itu sendiri dengan arsitektur YOLOv3, mereka menguji dengan dataset IMFDB, UGR, dan FireNet dengan akurasi yang diperoleh 89.3%, 82.6% and 86.5%. Penelitian dari [2] membuat deteksi senjata berupa pistol dan pisau menggunakan YOLOv4 dengan akurasi 66.67% sampai 77.78% dengan menggunakan dataset dari Open Image V6. Kami akan mencoba melihat kedua paper tersebut untuk mempelajari lebih lanjut bagaimana mereka membuat sistem mereka.

Dari dua paper sebelumnya menggunakan YOLO, dan jika ditinjau kembali, banyak yang menggunakan YOLO, Namun terdapat satu jenis model yang juga cukup cepat dan dikatakan lebih akurat yaitu SSD MobileNet. Kami akan menggunakan model tersebut dan coba membandingkan dengan model YOLO yang kedua paper sebelumnya. Hasil dari sistem yang dibuat bisa tepat untuk mendeteksi lokasi senjata tajam ataupun senjata api.

3. Data

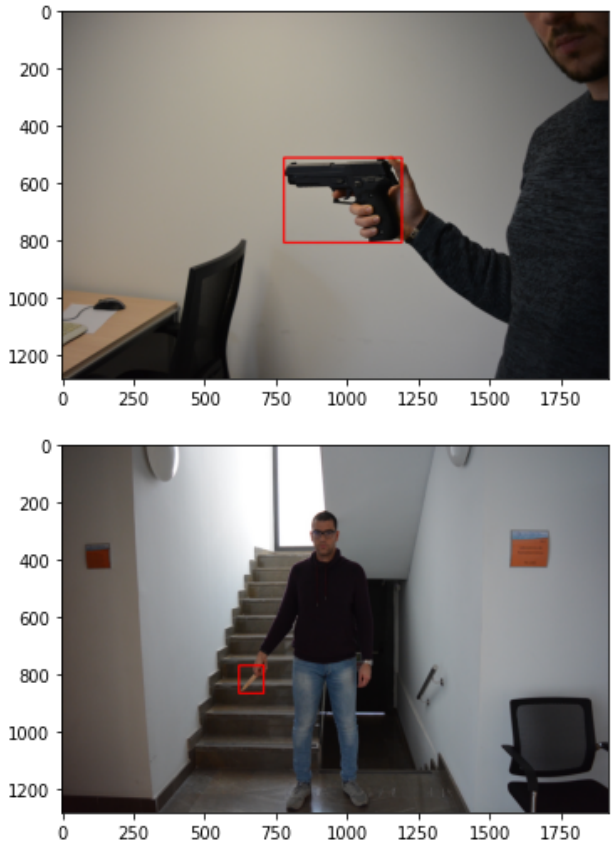
Untuk data kami sudah menemukan beberapa, untuk sekarang yang mungkin akan kami gunakan adalah dari *Andalusian Research Institute in Data Science* and Computational Intelligence [3], mereka mengumpulkan dataset gambar pistol dan senjata tajam pendek atau senjata tajam. Data tersebut sudah diberi anotasi untuk sistem deteksi object. Dataset tersebut dipilih karena data tersebut *open-source*, sudah dianotasi dan juga sudah melakukan *split* untuk mempermudah. Dataset tersebut

juga dibuat oleh instansi pendidikan sehingga menjamin kualitas data tersebut.

Secara rinci, kami menggunakan dataset tersebut pada bagian Weapons and Similar Handled Objects yang khusus untuk deteksi objek yang dinamakan Sohas Weapon Detection. Dataset ini dipilih karena memilih hasil yang lebih baik dibandingkan dengan menggunakan dataset terpisah antara gambar pisau dan gambar pistol. Dalam dataset ini mengandung objek lain selain pistol dan pisau seperti smartphone, bill, purse, dan card sehingga kami melakukan drop dengan data yang tidak digunakan.

Format dari semua gambar-gambar tersebut adalah JPG dengan ukuran yang berbeda-beda. Untuk anotasi pada gambarnya, menggunakan format XML yang berisi tentang jenis senjata yang ada pada gambar, ukuran gambar (height dan width), serta koordinat dari bounding box tersebut (koordinat X dan Y). Contoh dari dataset berupa gambar dan juga anotasinya dapat dilihat pada Gambar 1.

Dataset tersebut sudah melakukan pemisahan antara data train dengan data test baik sebagai gambar, maupun sebagai anotasi. Untuk banyaknya data pada masing-masing kelas dapat dilihat pada Tabel 1.



Gambar 1 Sampel gambar dan anotasi dari dataset

Tabel 1 Banyaknya data pada train dan test

Kelas	Data Train	Data Test
Pisau	1825	452
Pistol	1357	80

Pada bagian Metode dan Sistem, dijelaskan tentang konfigurasi pipeline di mana salah satu opsi konfigurasi adalah adanya augmentasi data saat training. Secara default, file konfigurasi pipeline yang didapatkan sudah memiliki augmentasi data berupa random horizontal flip dimana akan membalik gambar secara horizontal. Selain itu, ada augmentasi random crop image yang memotong sebagian gambar secara acak.

4. Metode dan Sistem

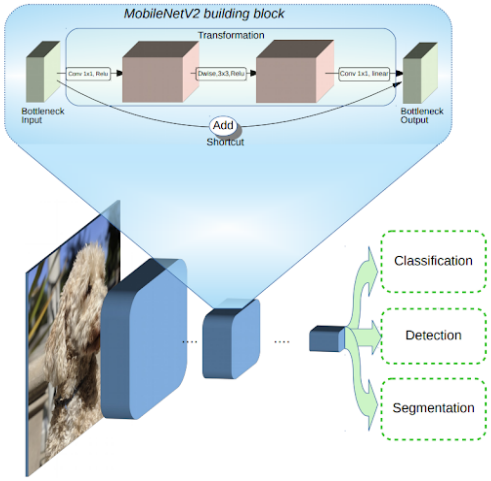
4.1 Tensorflow Object Detection API

Tensorflow Object Detection API (TFOD) adalah framework yang dibangun dengan Tensorflow untuk pembuatan model deteksi object yang lebih mudah [4]. Referensi code pertama kami [5] dimana source code yang diberikan berdasarkan pada TFOD, oleh karena itu tetap harus melakukan instalasi dependency yang ada pada TFOD. Pada referensi code [5] dan TFOD memberikan sebuah python script (file program python) untuk training dan evaluasi. Pada percobaan kami, semua python script yang dibutuhkan telah ditulis ulang dan disesuaikan fungsinya pada satu cell code dalam sebuah Interactive Python Notebook (file ipynb) yang pertama dibuat pada Google Colab. Hal tersebut dilakukan agar cukup satu file saja untuk semua proses pada penelitian ini.

4.2 SSD MobileNet

Metode yang digunakan adalah SSD MobileNet karena merupakan algoritma deteksi objek satu tahap berbasis framework klasifikasi/regresi yang lebih cepat dalam melakukan deteksi objek daripada algoritma dua tahap berbasis framework regional. Arsitektur SSD MobileNet yang kami buat adalah SSD MobileNet V2 karena untuk V2 memiliki akurasi yang lebih cepat daripada V1 [6].

MobileNet V2 adalah salah satu arsitektur convolutional neural network (CNN) berbasis ponsel yang dapat digunakan untuk mengatasi kebutuhan akan computing resource berlebih. MobileNet V2 merupakan penyempurnaan dari arsitektur MobileNet. Arsitektur MobileNet dan arsitektur CNN pada umumnya memiliki perbedaan pada penggunaan lapisan atau convolution layer. Convolution layer pada MobileNetV2 menggunakan ketebalan filter yang sesuai dengan ketebalan dari input image. MobileNetV2 menggunakan depthwise convolution, pointwise convolution, linear bottleneck dan shortcut connections antar bottlenecks [7].



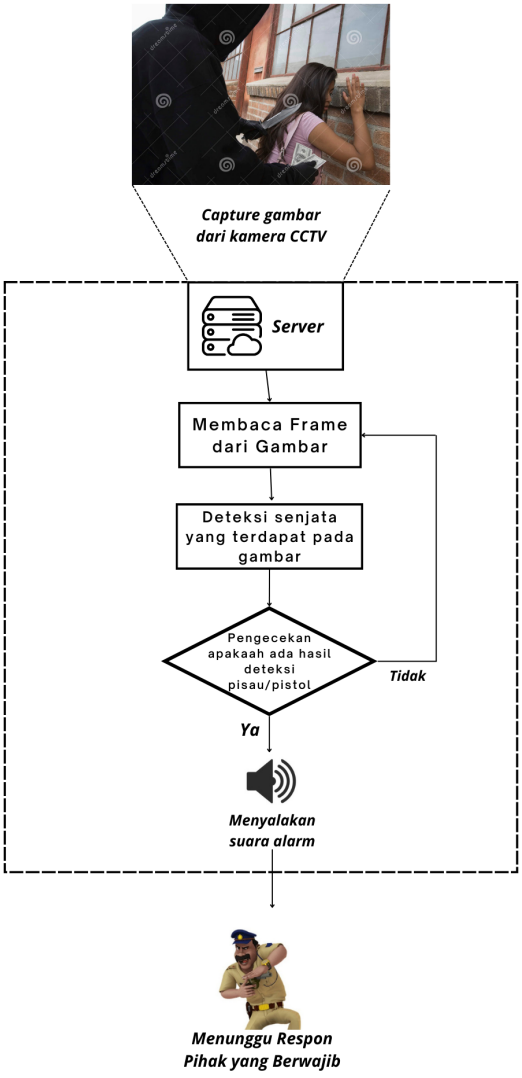
Gambar 2 Arsitektur MobileNetV2 dengan detail dari blok pembentukan konvolusi [8]

Gambaran arsitektur secara mudah tentang SSD MobileNetV2 dapat dilihat pada Gambar 2. Sementara arsitektur lengkap dapat dilihat pada Tabel 2. Pada setiap baris berisi satu atau lebih layer yang identik yang diulang sebanyak *n* kali. Semua layer dengan sequence yang sama memiliki output channel sebanyak *c*. Layer pertama dari setiap sequence memiliki stride *s*. Konvolusi spasial menggunakan kernel 3 x 3 dan expansion menggunakan factor *t* [9].

Tabel 2 Arsitektur lengkap MobileNet V2

Input	Operator	*t*	*c*	*n*	*s*
224^2 x 3	conv2d	-	32	1	2
112^2 x 32	bottleneck	1	16	1	1
112^2 x 16	bottleneck	6	24	2	2
56^2 x 24	bottleneck	6	32	3	2
28^2 x 32	bottleneck	6	64	4	2
14^2 x 64	bottleneck	6	96	3	1
14^2 x 96	bottleneck	6	160	3	2
7^2 x 160	bottleneck	6	320	1	1
7^2 x 320	conv2d 1x1	-	1280	1	1
7^2 x 1280	avgpool 7x7	-	-	1	-
1 x 1 x 1280	conv2d 1x1	-	k	-	

Pada TFOD terdapat model zoo yang memberikan model pretrain pada dataset COCO 2017, kami menggunakan model pretrain tersebut untuk mendapatkan sebuah konfigurasi pipeline untuk training dengan dataset kita, bentuk pipeline tersebut adalah sebuah file bernama pipeline.config, file tersebut berisi informasi seperti jumlah kelas, batch size, augmentasi data, learning rate, jumlah step training, dan hyperparameters lainnya. jenis pipeline yang kita gunakan adalah dari ssd_mobilenet_v2_fpnlite_640x640_coco17_tpu-8 yang bisa di akses pada model zoo TFOD, kita memilih jenis model tersebut karena memiliki performa terbaik dari model dengan Arsitektur SSD MobileNet V2 yang lain.



Gambar 3 Diagram sistem deteksi begal

4.4. Sistem yang Dibangun

Sesuai yang diusulkan pada bagian pendahuluan, sistem yang dibangun dimulai dengan menerima input sebuah frame dari CCTV yang terhubung dengan server. Server di sini sudah terdapat program dan model deteksi begal yang akan digunakan untuk mendeteksi senjata. Input frame sebelumnya adalah juga input frame pada model deteksi begal. Output dari model deteksi begal adalah koordinat bounding box objek senjata dan jenis senjata tersebut, yaitu pisau atau pistol.

Jika ada output dari model, maka sistem akan menyalakan suara alarm yang diharapkan akan menghimbau pihak yang berwajib seperti security. Jika tidak terdeteksi senjata, sistem akan terus memantau video yang ditangkap oleh CCTV. Video yang ditangkap CCTV juga akan disimpan pada local storage di mana video tersebut sudah tergambar bounding box senjata jika di dalam video tersebut terdapat senjata.

4.4. Metrik Evaluasi

Untuk evaluasi kami menggunakan metrik Akurasi (Q) seperti yang digunakan dalam kedua paper sebelumnya. Serta kami ingin mencoba jika berhasil menggunakan metrik mAP (mean Average Precision), metrik tersebut sering terlihat dalam referensi. Oleh karena itu kami juga mencobanya. Metrik yang juga sering dipakai dalam deteksi objek adalah IoU (Intersection over Union) karena metrik ini spesifik dalam evaluasi deteksi objek karena dapat dibedakan pada kasus klasifikasi objek yang tidak ada metrik ini. Untuk persamaan akurasi dapat dilihat pada persamaan (1), persamaan IoU pada persamaan (2), dan persamaan mAP pada persamaan (3)

$$Q = \frac{Jumlah\ klasifikasi\ benar}{Jumlah\ Data} \quad (1)$$

dimana jumlah klasifikasi yang benar adalah perjumlah dari True Positive (TP) dan True Negative (TN).

$$IoU = \frac{Area\ of\ overlap}{Area\ of\ union} \quad (2)$$

pada IoU, suatu deteksi dikatakan True apabila memenuhi suatu nilai IoU threshold.

$$mAP = \frac{1}{n}\sum_{k=1}^{k=n} AP_k \quad (3)$$

di mana AP_k adalah AP dari kelas k dan n adalah banyak kelas.

Untuk metrik IoU dan mAP memiliki pengembangan pada MS COCO. IoU MS COCO memiliki threshold dengan rentang dari 0.5 sampai 0.95 dengan setiap langkah adalah 0.05 sehingga ada 10 nilai threshold atau dapat dinotasikan sebagai [0.5 : 0.05 : 0.95]. Informasi mAP pada MS COCO yang lengkap dapat dilihat pada Tabel 3.

Juga terdapat beberapa fungsi perhitungan loss, terdapat dua jenis fungsi loss yang menurut kita penting. Localization Loss adalah fungsi loss terhadap bounding box yang diprediksi model dengan bounding box aslinya. Classification Loss adalah loss dari kelas yang diprediksi dengan kelas aslinya. perhitungan loss tersebut menggunakan sum of squared errors.

Tabel 3 mAP pada MS COCO

Jenis mAP	**Keterangan**
AP_{coco}	Rata-rata dari 10 IoU [0.5 : 0.05 : 0.95]
$AP_{coco}^{IOU=0.5}$	Rata-rata dengan IoU = 0.5
$AP_{coco}^{IOU=0.75}$	Rata-rata dengan IoU = 0.75

5. Result and Experiments

5.1. Training Environment

Kode python notebook sudah didesain untuk penggunaan pada Google Colab agar kita bisa menggunakan GPU yang diberikan untuk training lebih cepat, tetapi terdapat limitasi penggunaan GPU pada setiap akun. Selama proses training kita sudah berganti ke 5 akun Google yang berbeda agar tidak terkena limit pemakaian GPU. Selama penggunaan Google Colab kita dapatkan bahwa jumlah step training yang kita inginkan tidak bisa dicapai olehnya karena masalah limit GPU. oleh karena itu kita memutuskan untuk training secara local atau dengan komputer kita sendiri.

Spesifikasi yang digunakan untuk training local adalah CPU Ryzen 5 4600H, GPU Nvidia GTX 1650 TI dengan VRAM 4GB, dan kapasitas RAM 16GB. Performa komputasi GPU local milik kami tetep jauh dari performa yang diberikan pada Google Colab, tapi yang bisa dipastikan adalah tidak adanya limit.

Tabel 4 Hyperparameters pada Konfigurasi Pipeline

Nama	**Nilai Hyperparameter**
batch_size	3
num_classes	2
num_steps	97200

5.2. Training Parameter

Parameters dan hyperparameters terletak pada file konfigurasi pipeline yang sebelumnya sudah dijelaskan pada bagian penjelasan metode SSD MobileNet. Pada konfigurasi pipeline berikut yang kami rubah dan tetapkan nilai hyperparameters pada pipeline.

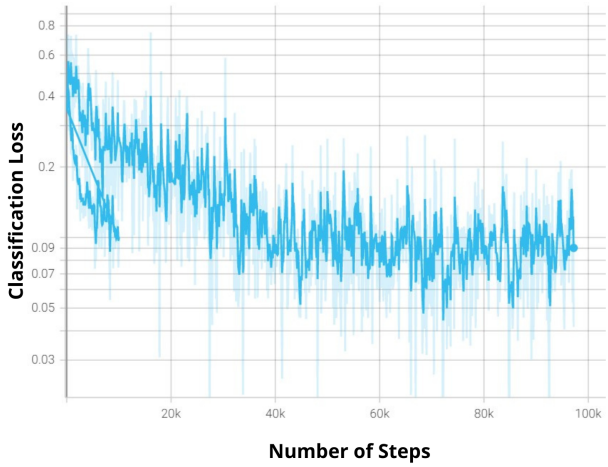
Pada batch_size kita tentukan nilai 3 dari hasil "trial and error", dimana kita awal awal mulai dengan nilai 12 untuk batch_size. Nilai 12 mampu dijalankan pada Google Colab, saat dijalankan di pada local (menggunakan komputer kami) tentu dia langsung memberikan error Out of Memory, memory disini merujuk pada memory GPU atau VRAM. Dari nilai 12 kita coba kurangkan dengan 1 dan dicoba training apakah ada error Out of Memory atau tidak, kita kurangkan dengan nilai 1 terus sampai didapatkan nilai batch_size 3 karena selama training tidak hadapkan error Out of Memory.

Selanjutnya, untuk num_classes kita berikan nila 2 karena jelas object yang kita ingin deteksi ada 2, yaitu senjata pisau dan pistol. Untuk num_steps adalah jumlah step training yang dilakukan, awal awal kita gunakan nilai 2000 steps, dari situ kita lihat dan menemukan bahwa setiap 1000 step ada penurunan loss sekitar kurang lebih 10%. Kita ingin mendapatkan serendah rendah mungkin oleh karena itu saat training ada fitur untuk save checkpoint yang diberikan oleh TFOD, dimana fitur ini menyimpan keadaan model saat training dan bisa dilanjutkan jika jumlah training step belum terpenuhi. Dari hal tersebut kita menambahkan jumlah training stepnya dengan merubah nilai num_steps pada pipeline dan kode training dengan nilai yang lebih besar dari nilai sebelumnya terus menerus sampai kita puas dengan nilai loss yang kami punya. Dari hasil eksperimen tersebut kita menemukan nilai num_steps 97200 yang menurut kami memuaskan. terkait experiment num_steps bisa kita lihat hasilnya nyatanya di bagian Hasil Training.

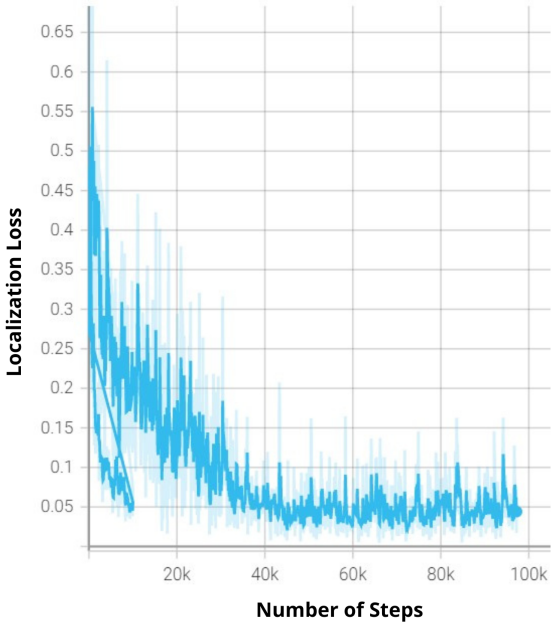
5.3. Hasil Training

Training dilakukan pada malam hari, dengan menggunakan fitur checkpoint kita bisa memberhentikan training pada pagi hari dan dilanjuti training pada malam harinya lagi. Secara total, training membutuhkan 1 hari dan 11 jam untuk mendapatkan hasil loss yang memuaskan bagi kami. Berikut adalah grafik perubahan nilai loss classification pada Gambar 4 dan localization pada Gambar 5. Bisa dilihat bahwa penurunan loss setiap 1000 step sekitar kurang lebih 10% yang dijelaskan pada bagian Training Parameter terlihat nyata dari gambar ini. Hasil akhir training kita mendapatkan nilai 0.04155 untuk Classification Loss dan nilai 0.01697 Localization Loss.

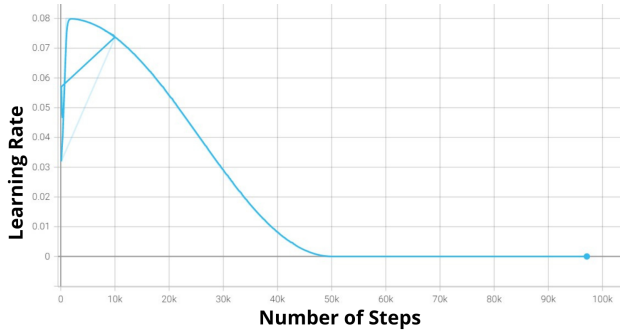
Ada juga grafik perubahan learning rate pada Gambar 6, learning rate pada pipeline konfigurasi dengan dibuat bisa berubah pada setiap step. Saat di nilai step 50.000 kita bisa melihat bahwa learning rate bernilai 0 dimana disini berdampak juga pada nilai loss saat training dimana di grafik Gambar 4 dan Gambar 5 pada saat step ke 50.000 juga tidak bisa turun lagi, melainkan memiliki fluktuasi naik turun yang tinggi karena tidak ada nilai learning rate untuk menghaluskan perubahan nilainya. Saat kita training dari step 50.000 sampai step terakhir dilakukan pada malam hari dimana kita tidak memonitor perubahannya, tetapi karena hasil loss sudah memuaskan kita tidak melanjuti training dengan nilai learning rate yang di atas 0.



Gambar 4 Nilai Classification Loss pada setiap training step



Gambar 5 Nilai Localization Loss pada setiap training step



Gambar 6 Nilai Learning Rate pada setiap training step

5.4. Hasil Data Test

Untuk mengevaluasi hasil dari training model kami, pertama kami menggunakan nilai loss dan juga AP COCO yang telah diberikan oleh TFOD. Lebih lanjut, untuk mendapatkan akurasi, diperlukan confusion matrix terlebih dahulu. Dari confusion matrix akan didapatkan i TF dan TN secara langsung yang nilainya bisa dilihat pada Tabel 5, maka nilai akurasi didapatkan adalah dengan membagi nilai klasifikasi yang benar (TF ditambah TN) dengan banyaknya data seperti pada persamaan (1).

Tabel 5 Hasil confusion matrix

		Aktual		**Tidak Terdapat anotasi tapi diprediksi**
		knife	**pistol**	
Prediksi	**knife**	374	4	74
	pistol	4	63	13
Terdapat anotasi tapi tidak diprediksi		7	3	0

Kami mendapatkan confusion matrix dengan menggunakan script tambahan yang tersedia di Github oleh Valdarrama, dkk [10] sesuai dengan framework TFOD. Batas nilai IoU pada confusion matrix tersebut adalah 0.5, sehingga jika hasil IoU deteksi lebih atau sama dengan 0.5, maka akan dimasukan ke tahap pengelompokan pada confusion matrix. Hasil dari script tersebut juga memberikan nilai Precision setiap kelas yang dinamakan Precision@0.5IOU Pisau dan Precision@0.5IOU Pistol, dimana mereka berbeda dari $AP_{coco}^{IOU=0.5}$ yang melihat dari semua kelas.

Hasil evaluasi dengan confusion matrix dapat dilihat lebih lengkap pada Tabel 5 dan dengan metrik-metrik yang lain pada Tabel 6. Metrik yang ditandai (*) memiliki perbedaan dengan metrik AP sebelumnya karena dihitung dengan menggunakan script yang berbeda. Pada AP COCO sebelumnya dihitung dengan menggunakan TFOD, sedangkan pada Precision yang ditandai (*) dihitung dengan menggunakan script dari Valdarrama, dkk.

Tabel 6 Hasil dari masing-masing metrik

Metrik	**Hasil**
Akurasi	0.820
AP_{coco}	0.547
$AP_{coco}^{IOU=0.5}$	0.875
$AP_{coco}^{IOU=0.75}$	0.603
Rata-rata IoU semua kelas	0.612
Localization loss	0.161
Classification loss	0.249
Precision@0.5IOU Pisau (*)	0.971
Precision@0.5IOU Pistol (*)	0.900

Pada sisi akurasi, model kia memiliki akurasi yang lebih tinggi daripada penelitian [2] dengan akurasi 0.77 pada paper tersebut, sedangkan hasil yang kami dapatkan adalah 0.82 yang menggunakan metode YOLOv4. Selain itu, rata-rata IoU untuk semua kelas didapatkan nilai 0.612 yang dianggap baik karena suatu nilai IoU dikatakan bagus apabila memiliki nilai di atas 0.5 [11].

5.5. Hasil Deployment

Pada deployment, kami membuat script Python yang sesuai pada alur pada Gambar 3, di mana akan ada warning jika terdeteksi senjata pisau ataupun pistol. Script python tersebut kita beri nama file “realTimeOD.py”. Untuk melakukan simulasi pada Gambar 3, kami menggunakan kamera smartphone yang disambungkan pada laptop dan speaker dari laptop untuk alarm-nya sendiri. Konfigurasi yang diperlukan adalah mengatur input berupa kamera dan output berupa speaker. Agar dapat membunyikan suara, diperlukan Python library tambahan bernama PyAudio.

Hasil dari pengujian ini bisa dilihat pada video presentasi kami. dimana pada video tersebut ditampilkan laptop yang menjalankan sistem kami, dengan adanya sebuah orang yang memegang pisau di depan laptop.

Dengan menggunakan kamera laptop, dapat menangkap videostream dari kamera dan setiap frame dari videostream akan dijadikan sebagai input ke model deteksi kami. Dalam frame tertampil seseorang sedang memegang pisau, lalu pisau tersebut terdeteksi dan membunyikan alarm. Videostream yang diproses oleh sistem akan tersimpan dalam local storage, dimana videostream yang tersimpan sudah digambar sebuah bounding pada senjata yang terdeteksi untuk memperlihatkan dimana posisi senjata tersebut. Selanjutnya, videostream yang tersimpan bisa digunakan untuk analisis lebih lanjut untuk aparat atau untuk ranah forensik.

Performa Frame Rate Per Second (FPS) juga penting, dalam percobaan kami didapatkan rata rata 9.9 FPS pada resolusi gambar 1080 x 1080 pixels. Nilai FPS tersebut diperoleh dengan GPU yangs sama saat training. Pada percobaan kami, ditemukan suatu kekurangan di model kami, di mana jika objek senjata jauh dari kamera, akan mengakibatkan senjata tidak terdeteksi. Oleh karena itu, model tepat jika ukuran senjata tersebut berukuran ⅛ dari frame. Kami mengasumsikan bahwa hal ini dipengaruhi oleh dataset karena ukuran objek pada dataset kurang banyak mengandung objek yang berukuran kecil.

Untuk menjalankan script ini, diharapkan sudah meng-*install* library yang juga digunakan pada program training beserta memiliki folder yang berisi model yang di-*export*.

6. Kesimpulan

Dari hasil pengujian di atas, dapat disimpulkan bahwa telah berhasil dibuat sistem deteksi begal dengan akurasi sebesar 0.82, rerataan IoU sebesar 0.612, serta mAP pada MS COCO dengan threshold IoU=0.5 memiliki nilai sebesar 0.875. Namun terdapat kekurangan pada model kami, yaitu jika objek senjata jauh dari kamera akan mengakibatkan senjata tidak terdeteksi. Untuk penelitian selanjutnya diharapkan menggunakan dataset dengan berukuran anotasi ukuran senjata yang lebih bervariasi.

Daftar Pustaka

[1] P. Mehta, A. Kumar and S. Bhattacharjee, "Fire and Gun Violence based Anomaly Detection System Using Deep Neural Networks," 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), 2020, pp. 199-204, doi: 10.1109/ICESC48915.2020.9155625.

[2] W. E. I. B. W. N. Afandi and N. M. Isa, "Object Detection: Harmful Weapons Detection using YOLOv4," 2021 IEEE Symposium on Wireless Technology & Applications (ISWTA), 2021, pp. 63-70, doi: 10.1109/ISWTA52208.2021.9587423.

[3] F. P. Hernandez, & A. C. Lamas. (2022). OD-WeaponDetection [Online] . Available: https://github.com/ari-dasci/OD-WeaponDetection

[4] Abhimanyu. (2020). SSD-Mobilenet-Custom-Object-Detector-Model-using-Tensorflow-2 [Online]. Available: https://github.com/abhimanyu1990/SSD-Mobilenet-Custom-Object-Detector-Model-using-Tensorflow-2

[5] J. Huang et al., "Speed/accuracy trade-offs for modern convolutional object detectors," Nov. 2016. [Online] Available: https://github.com/tensorflow/models/tree/master/research/object_detection

[6] N. Nufus et al., "Sistem Pendeteksi Pejalan Kaki Di Lingkungan Terbatas Berbasis SSD MobileNet V2 Dengan Menggunakan Gambar 360° Ternormalisasi," Prosiding Seminar Nasional Sains Teknologi dan Inovasi Indonesia (SENASTINDO), vol. 3, pp. 123–134, Dec. 2021, doi: 10.54706/senastindo.v3.2021.123.

[7] Ae, Nasha Hikmatia, and Muhammad Ihsan Zul. "Aplikasi Penerjemah Bahasa Isyarat Indonesia menjadi Suara berbasis Android menggunakan Tensorflow." *Jurnal Komputer Terapan* 7.1 (2021): 74-83.

[8] R. O. Ekoputris, 'MobileNet: Deteksi Objek pada Platform Mobile'. 5 2018. [Online]. Available: https://medium.com/nodeflux/mobilenet-deteksi-objek-pada-platform-mobile-bbbf3806e4b3.

[9] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," Jan. 2018, [Online]. Available: http://arxiv.org/abs/1801.04381

[10] S. Valdarrama, Philip, and S. Lee, tf_object_detection_cm. 2020. [Online]. Available: https://github.com/svpino/tf_object_detection_cm

[11] Zitnick, C. Lawrence, and Piotr Dollár. "Edge boxes: Locating object proposals from edges." European conference on computer vision. Springer, Cham, 2014.