Erkennung von kleinen Objekten mit einer RetinaNet Implementierung anhand eines Bienenbilder-Beispiels

1er Marcel Reineck
Hochschule München für angewandte Wissenschaften
München, Deutschland
reineck@hm.edu

2er Lorin Arndt
Hochschule München für angewandte Wissenschaften
München, Deutschland
larndt@hm.edu

3er Benjamin Eder
Hochschule München für angewandte Wissenschaften
München, Deutschland
beder@hm.edu

***Zusammenfassung*—Für das „Environment and Bee Monitoring"-Projekt der Hochschule München wird ein kamerabasiertes Überwachungssystem entwickelt, um die Veränderungen von Bienenpopulationen in Korrelation mit Umweltfaktoren zu quantifizieren. Dabei sind die genaue Erkennung und Lokalisierung der ein- und ausfliegenden Honigbienen entscheidend. Für die Detektion der kleinen Bienenobjekte wird in dieser Arbeit ein tiefes neuronales Netz auf Basis einer RetinaNet Architektur implementiert. Um das Potential des verwendeten Algorithmus zu analysieren, werden für den Anwendungsfall die signifikanten Hyperparameter der Netzarchitektur identifiziert und optimiert. Als Indikatoren für die Perfomance eines trainierten Modells werden die erreichten *Precision*, *Recall* und *average-precision* Werte auf einem Bienendatensatz evaluiert. Durch Zusammenführen der Optimierungsergebnisse konnte eine Hyperparameterkonfiguration der Netzwerkarchitektur gefunden werden, mit der das resultierende Modell einen AP-Wert von 99,6 % bei einer durchschnittlichen Rechenzeit von 42 ms pro Bild (24 fps) auf einer NVIDIA RTX 3080 Grafikkarte und einem Ryzen 7 5800x Prozessor von AMD erreicht.**

***Schlagworte* – Deep Learning, RetinaNet, Objekterkennung, Bienen, kleine Objekte**

I. Einleitung

Im Zuge der Veranstaltung *Deep Learning* an der Hochschule München für angewandte Wissenschaften im Wintersemester 20/21 versuchen wir einen Algorithmus zur Erkennung von Bienen zu entwickeln. Durch die neuen Erkenntnisse, wie beispielsweise die Netzarchitektur und gefundenen Hyperparameter, hoffen wir das Projekt *EnBeeMo* geeignet unterstützen zu können.

II. Methoden und Werkzeuge

Im ersten Schritt argumentieren wir die Methoden und Werkzeuge, welche im Laufe des Projekts verwendet werden.

A. Objekterkennungsmethode

Aktuelle Deep Learning Objekterkennungsmethoden können in Ein- oder Zweiphasen-Methoden untergliedert werden. Die Zweiphasigen, wie *Faster R-CNN*, versuchen vor der Klassifikation, Objektvorschläge in den gegebenen Bildern zu finden, während bei den Einphasigen, wie *SSD*, *YOLO* oder *RetinaNet*, die Vorschläge gleichzeitig extrahiert und klassifiziert werden. Allgemein gelten zweiphasige Methoden, verglichen mit einphasigen, als langsamer, bieten jedoch eine bessere Erkennungsleistung. [1, S. 21]

Bei der Erkennung von Bienen sind wir mit einer Vielzahl gleichförmiger Bilder konfrontiert. Auf jedem der Bilder sind eine Anzahl an ungefähr gleich dimensionierten Bienen zu sehen, welche jedoch gemessen an den Ausmaßen des Bildes relativ klein sind. Daher suchen wir gezielt nach geeigneten Methoden bzw. Architekturen zur Erkennung von **kleinen Objekten**.

In einer verwandten Arbeit wird ein Vergleich von Objekterkennungsmethoden bei kleinen Objekten angestellt. Dabei scheinen einphasige Methoden ähnliche *mean Average Precision*-Werte (*mAP*) wie zweiphasige zu erreichen [2, S. 12]. Als eine der aktuellen Lösungen — und wegen der besseren Performance des Single-shot-detectors gegenüber zweiphasigen Methoden — entscheiden wir uns für eine RetinaNet Architektur. Wir hoffen, dass die Einfachheit unseres Problems, verglichen mit einem COCO Datensatz, damit zu einem guten Ergebnis führt.

B. Architektur und Implementierung

Vor Beginn der Implementierung haben wir uns darauf verständigt das Framework PyTorch zu verwenden. Als Startpunkt für die Implementierung haben wir die relevanten Teile des Quellcodes von https://github.com/yhenon/pytorch-retinanet verwendet und in unserem Jupyter Notebook zusammengefasst. Eine grobe Abbildung der verwendeten Architektur ist in Abbildung 1 zu finden.

Dabei haben die dargestellten Faltungsebenen *conv1, ..., conv4* des Klassifikations- und Regressionsmodells jeweils eine Ausgabe der Größe $WxHx256$. Lediglich die letzen Ebenen unterscheiden sich in ihrer Ausgabegröße. So liefert das Regressionsmodell $WxHx(4 * \text{Anzahl Anchor Boxen})$ —

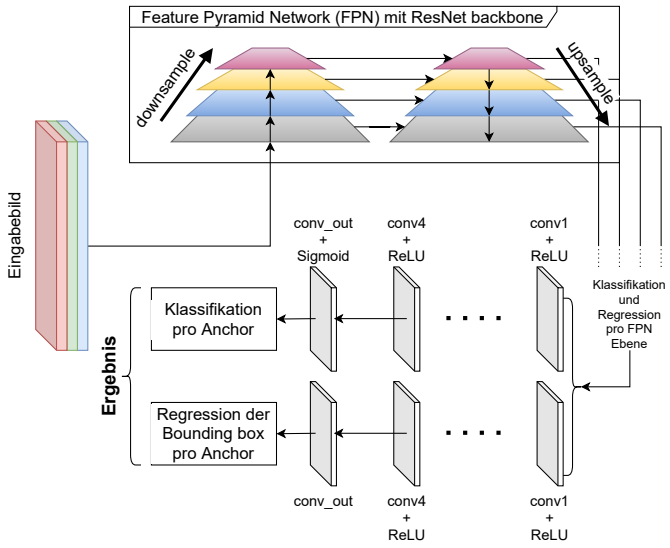

Abbildung 1. Darstellung der verwendeten RetinaNet Meta-Architektur zur Erkennung von Bienen Bounding Boxen.

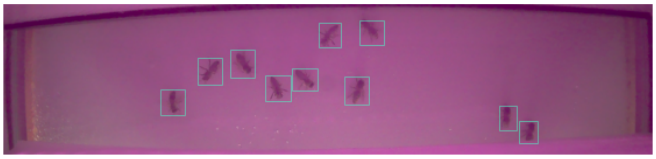

Abbildung 2. Beispiel aus dem gegebenen Datensatz. Die blauen Boxen um die Bienen herum sind die *Ground Truth* Bounding Boxen.

also die Bounding Boxen — und das Klassifikationsmodell $WxHx(\text{Anzahl Klassen} * \text{Anzahl Anchor Boxen})$ — also die Klassifikation — als Ergebnis. Dabei ist anzumerken, dass wir in unserem Falle nur eine Klasse *Biene* haben. Das Klassifikations- und Regressionsmodell erhält als Eingabe die Ausgabe des Feature Pyramid Networks (FPN) auf Basis eines *ResNet* Backbones. Dabei erhält das FPN wiederum ein Bild zur Verarbeitung.

Im weiteren Verlauf dieser Arbeit werden auf die einzelnen Aspekte wie das Backbone, FPN und die Anchor Boxen der Implementierung weiter eingegangen, welche wir verändert haben, um die Erkennung von Bienen in unserem Datensatz zu optimieren.

C. Bienen Datensatz und Vorverarbeitung

Ein Teil der Arbeit bestand darin, die bereitgestellten Bienenbilder einzulesen und vorzuverarbeiten. Da wir sowohl auf AWS, als auch lokal testen wollen, entwickeln wir ein PyTorch Dataset, welches die Daten entweder von dem bereitgestellten S3 Bucket oder aus dem lokalen Dateisystem beziehen kann. Ein Beispiel eines gegebenen Bienenbilds mitsamt *Ground Truth* Bounding Boxen ist in Abbildung 2 dargestellt.

Die entstandene Dataset Klasse übernimmt auch die Vorverarbeitung der Bienenbilder. Zum einen verkleinern wir die Bilder auf eine maximale Seitengröße von 1024 Pixeln, transformieren also die Originalgröße von $1613 * 371$ auf $1024 * 236$ Pixel.

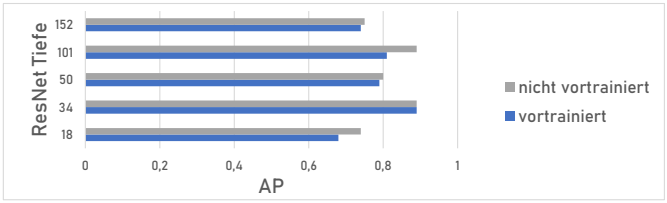

Abbildung 3. Average Precision (AP) auf dem Testdatensatz für verschiedene ResNet Backbone Tiefen nach 10 Epochen Training.

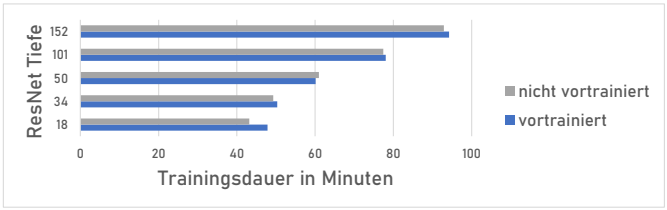

Abbildung 4. Trainingsdauer von 10 Epochen und verschiedenen ResNet Backbone Tiefen.

Außerdem werden die Bilddaten normalisiert, indem die RGB Werte durch 255 geteilt und mittels vordefinierter Mittelwerte und Standardabweichungen angepasst werden.

D. Training und Testaufteilung

Es stehen 2000 Bienenbilder zur Verfügung, wobei bei einem Bild die Ground Truth Bounding Boxen — also das Labeling — fehlen. Wir entscheiden uns für eine Aufteilung von 80 % Trainingsdaten und 20 % Testdaten, also 1599 Bilder für das Training und 400 Bilder für die Evaluation.

III. Optimierung

Im nachfolgenden Abschnitt vertiefen wir die vorgestellte RetinaNet Architektur und stellen, neben unseren architektonischen Änderungen, unser Vorgehen bei der Hyperparameteroptimierung vor. Grundsätzlich haben wir — wenn in den Unterkapiteln nicht anders dargestellt — die Standardparameter der verwendeten Implementierung nicht weiter angepasst. Wir nehmen uns die Teile *ResNet Backbone*, *FPN*, *Anchor boxen*, *Focal loss Funktion*, *Lernrate*, ... sukzessive vor und verbessern so das Modell schrittweise, wobei wir mit den Teilen anfangen, die aus unserer Sicht die größten Auswirkungen auf die anderen Bestandteile haben.

A. ResNet Backbone

Die verwendete Implementierung ermöglicht die Verwendung von verschiedenen ResNet-Modellen mit unterschiedlichen Ebenentiefen $18, 34, 50, 101$ und 152 als Basis für das FPN. Ein tieferes ResNet Modell bedeutet generell eine bessere mögliche Accuracy. Allerdings bedeutet das gleichzeitig auch eine langsamere Inferenz- und Trainingszeit [3, S. 6]. Daher wollen wir für unser Problem die bestmögliche Option finden.

Zum Testen lassen wir das Netz mit unterschiedlichen ResNet Backbone Konfigurationen auf den Trainingsbildern bei einem fixem Seed trainieren. Dabei betrachten wir sowohl

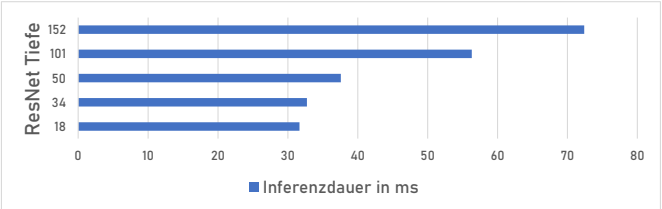

Abbildung 5. Inferenzdauer gemittelt über 100 Bilder bei verschiedenen ResNet Backbone Tiefen.

vortrainierte, als auch untrainierte ResNet Modelle. Im Diagramm 3 präsentieren wir die erhaltene *Average Precision* (AP) auf den Testdaten. Dabei fällt auf, dass ResNet-34 die beste Performance bei vortrainierten wie auch untrainierten Modellen liefert. Weiter zeigen wir die Auswirkungen auf die Trainings- und Inferenzzeit in den Abbildungen 4 und 5. Insgesamt scheint ResNet-34 die beste Wahl zu sein, wenn man zwischen Laufzeit und Erkennungsperformance abwägt. ResNet-18 ist im Gegensatz etwas schneller, bietet jedoch eine deutlich geringere Erkennungsleistung, während höhere Modelltiefen zwar ähnliche Ergebnisse liefern, jedoch deutlich mehr Zeit beanspruchen.

B. Feature Pyramid Network

Wie in Abbildung 1 zu erkennen ist, liefert der Output eines Feature-Pyramid-Network (FPN) den Input der Modelle für Regression und Klassifikation. Die FPN Architektur mit dem ResNet34-Backbone kombiniert auf ihren Ebenen niedrig aufgelöste, semantisch starke Merkmale mit hoch aufgelösten, semantisch schwachen Merkmalen über einen Top-Down-Weg und laterale Verbindungen. Dadurch können auf den Ebenen des FPNs, Objekte unterschiedlicher Größe detektiert werden [4, S. 3]. In der Regel wird bei der Implementierung von FPNs die erste Ebene C2 zugunsten der Rechendauer vernachlässigt. Da sie nach der Theorie allerdings für die Lokalisierung von sehr kleinen Objekten verantwortlich ist, wird für die Erkennung der Bienenobjekte ihr Einfluss untersucht. Zusätzlich wird die Notwendigkeit der beiden letzten Schichten P6 und P7 unter Berücksichtigung der durchschnittlichen Inferenzdauer auf einem Bild ermittelt. Für die Untersuchung wird das RetinaNet mit drei möglichen FPN-Konfigurationen trainiert und das resultierende Modell auf den Testdaten evaluiert.

Es ist in Abbildung 6 zu erkennen, dass durch Hinzufügen der Ebene C2 bei moderatem Anstieg der Inferenzdauer ein AP-Wert von 96 % erreicht werden konnte. Im Vergleich zu den anderen Testläufen kann daraus geschlossen werden, dass durch die erhöhte Anzahl von kleinen Objekten im Datensatz die erste Ebene des FPNs für eine gute Erkennungsleistung benötigt wird. Ein Einfluss von P6 und P7 auf die Inferenzzeit des Modells lässt sich nicht feststellen. Die gewählte Konfiguration für das FPN ist demzufolge inklusive der Ebenen C2, P6 und P7.

C. Anchor boxen

Anchor Boxen sind Boxen mit einer festen Größe, die das Modell verwendet, um die Bounding Box eines Objektes

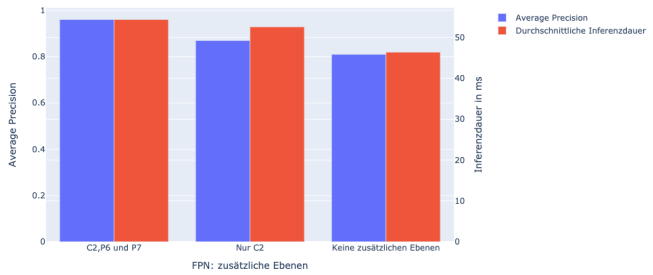

Abbildung 6. Average Precision und Interferenzdauer von verschiedenen FPN-Konfigurationen.

vorherzusagen. Pro Ebene der Feature Map werden neun Anchor Boxen initialisiert, wovon die Größe jeder einzelnen Box abhängig von drei Anchor Ratios und drei Anchor Scales ist [5]. Während des Trainings wird versucht, die Anchor Box mit der höchsten Überlappung (IoU) zur Bounding Box eines Ground Truth Objektes zu finden. Die Anchor Box mit der höchsten Überlappung mit einem Objekt ist dann verantwortlich für die Vorhersage der Klasse und Position des Objektes im Bild [6]. Da die richtige Wahl der Anchor Box Größen maßgeblich für die Vorhersage des Modells ist, wurde empirsch ermittelt, welche drei Anchor Ratios und welche drei Anchor Scales den höchsten AP-Wert erreichen. Nachfolgend sind die Ergebnisse bei 10 Epochen und einem random Seed aufgezeigt:

Anchor Scales	Anchor Ratios	**Ø Precision**	**Ø Recall**	**Ø AP**
$0.5, 1, 2^{1/3}$	$0.5, 1, 2$	$0,901$	$0,887$	**0,963**
$1, 2^{1/3}, 2^{2/3}$	$0.5, 1, 2$	$0,898$	**0,899**	$0,816$
$2^{1/3}, 2^{2/3}, 2$	$0.5, 1, 2$	**0,906**	$0,898$	$0,955$
$2, 2.5, 3$	$0.5, 1, 2$	$0,882$	$0,839$	$0,855$

Die Verwendung von $2^{1/3}, 2^{2/3}, 2$ für die Anchor Ratios und 0.5, 1, 2 für die Anchor Scales erzielte bei uns mit unter die höchsten Precision und Recall Werte und auch einen Average Precision Wert von 95.5 %. Die Anchor Ratios mussten nicht angepasst werden, da sie bereits passend zu den Bienenlabels sind.

D. Focal Loss Funktion

RetinaNet verwendet *Focal Loss* als Loss-Funktion [7, S. 3]. Dabei lassen sich der *focusing Parameter* γ und der davon abhängige α Parameter optimieren [7, S. 3]. Im Paper werden bereits einige Werte zur Focal Loss Funktion vorgeschlagen, welche wir ebenfalls für die Optimierung ausprobieren wollen [7, S. 6]. Nachfolgend sind die Ergebnisse bei 10 Epochen und einem fixen Seed aufgezeigt:

γ	α	**Precision**	**Recall**	**AP**	**Loss**
$0,0$	$0,75$	$0,91$	$0,88$	$0,75$	$0,37$
$0,1$	$0,75$	$0,91$	$0,86$	$0,75$	$0,38$
$0,2$	$0,75$	$0,92$	$0,86$	$0,70$	$0,38$
$0,5$	$0,50$	$0,92$	$0,86$	$0,74$	$0,37$
$1,0$	$0,25$	$0,92$	$0,87$	$0,74$	$0,37$
$2,0$	$0,25$	$0,92$	$0,86$	$0,73$	$0,37$
$5,0$	$0,25$	$0,92$	$0,87$	$0,73$	$0,38$

Aus den Ergebnissen ist ersichtlich, dass die Anpassung der beiden Parameter in unserem Fall keine Auswirkungen hat. Wir belassen daher die Werte auf den Standardeinstellungen $\gamma = 2,0$ und $\alpha = 0,25$.

E. Gradientenabstiegsverfahren

Während des Trainings des Modells beläuft es sich mathematisch auf ein Optimierungsproblem. Ein Verfahren, um dieses Problem zu lösen, ist das Gradientenabstiegsverfahren. Wir entschieden uns den Adaptive Moment Estimation (Adam) Optimizer mit dem Stochastic Gradient Descent (SGD) Optimizer zu vergleichen. Dafür wurde der Precision, Recall und AP-Wert bei unterschiedichen Lernraten ermittelt. In der nachfolgenden Tabelle sind die Ergebnisse bei 10 Epochen, die Anchor Ratios und Scales aus Kapitel III-C und einem fixen Seed aufgezeigt. Es ist zu erkennen, dass der SGD Optimizer mit einer Lernrate von $1e^{-3}$ den höchsten Average Precision Wert von 82.2 % erreicht hat. Daher wurde für den SGD Optimizer entschieden.

Lernrate	Optimizer	**Ø Precision**	**Ø Recall**	**Ø AP**
$1e^{-3}$	Adam	$0,897$	$0,901$	$0,788$
$1e^{-4}$	Adam	$0,894$	$0,906$	$0,757$
$1e^{-5}$	Adam	$0,892$	**0,908**	$0,757$
$1e^{-3}$	SGD	$0,893$	$0,903$	**0,822**
$1e^{-4}$	SGD	$0,897$	$0,899$	$0,752$
$1e^{-5}$	SGD	**0,900**	$0,898$	$0,737$

Anschließend wurde empirisch ermittelt, ob durch eine Veränderung der Lernrate ein höherer Average Precision Wert erreicht werden kann. Nachfolgend ist eine Auflistung der durchschnittlichen Ergebniswerte der Lernraten $1e^{-2}$, $5e^{-2}$, $1e^{-3}$, $5e^{-3}$ und $1e^{-4}$ mit unterschiedlichen Seeds und den Anchor Ratios und Anchor Scales aus Kapitel III-C aufgezeigt:

Lernrate	**Ø Precision**	**Ø Recall**	**Ø AP**
$1e^{-2}$	$0,896$	$0,892$	**0,936**
$5e^{-2}$	**0,921**	$0,887$	$0,906$
$1e^{-3}$	$0,893$	**0,903**	$0,822$
$5e^{-3}$	$0,888$	$0,876$	$0,921$
$1e^{-4}$	$0,897$	$0,899$	$0,752$

Es ist erkennbar, dass bei einer Änderung der Lernrate zu $1e^{-2}$ eine Performanceverbesserung des Average Precision zum Wert 93.6 % möglich ist.

IV. Zusammenfassung und Ausblick

Die Herausforderung der vorliegenden Problemstellung bestand in der Erkennung von vielen kleinen Objekten derselben Klasse auf einem Inputframe. Eine State-of-the-Art Recherche hat gezeigt, dass für derartige Probleme effiziente, einphasige Methoden gute Ergebnisse erzielen können. Deshalb entschieden sich die Autoren dieser Arbeit für eine auf der Funktionsweise eines Single-Shot-Detectors basierenden RetinaNet Architektur. Zunächst wurden grundlegende architektonische Veränderung, wie die Anpassung des verwendeten Backbone Netzwerkes, sowie die Anzahl der FPN-Ebenen, untersucht. Anschließend wurden die Einflüsse der Hyperparameter, Anchorboxenrate, Focussing Parameter sowie der Lernrate des verwendeten Optimierers in definierten Intervallen

Abbildung 7. Erkannte Bienen auf einem Testbild. Die roten Bounding Boxen repräsentieren die Vorhersage mit dem erreichten Konfidenzniveau.

auf die Modelperformance analysiert. Mit der Kombination aus den gefundenen Werten für die Hyperparameter konnte ein Model trainiert werden, welches für die Bienenerkennung eine Precision von 89,9 %, einen Recall von 89,2 % und einen AP-Wert auf dem Testdatensatz von 99,6 % erreicht. Ein exemplarisches Testergebnis des Models ist in Abbildung 7 visualisiert.

Für weitere Optimierungsansätze könnte in folgenden Experimenten der Design Space mit zusätzlichen Hyperparametern und größeren Intervallen erweitert werden. Für die Potentialanalyse zu einer pareto-optimalen Netzwerkarchitektur können auf Basis dieser Arbeit multi-kriterielle Ansätze der Optimierung, wie die Hyper Space Exploration angewendet werden. In einem weiteren Schritt kann dann das gefundene Model in die EnBeeMo-Einheit integriert und in der Live-Umgebung getestet werden.

A. Arbeitsanteile der Autoren am Projekt

Abschließend wollen wir noch anmerken, dass die Arbeitslast im Projekt über den gesamten Arbeitszeitraum gleichmäßig verteilt worden ist, sodass alle Autoren gleichmäßig daran beteiligt sind.

Literatur

[1] X. Jiang, A. Hadid, Y. Pang, E. Granger, and X. Feng, Eds., *Deep Learning in Object Detection and Recognition*. Springer Singapore, 2019. [Online]. Available: https://doi.org/10.1007/978-981-10-5152-4

[2] N.-D. Nguyen, T. Do, T. D. Ngo, and D.-D. Le, "An evaluation of deep learning methods for small object detection," *Journal of Electrical and Computer Engineering*, vol. 2020, p. 3189691, Apr 2020. [Online]. Available: https://doi.org/10.1155/2020/3189691

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[4] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," *CoRR*, vol. abs/1612.03144, 2016. [Online]. Available: http://arxiv.org/abs/1612.03144

[5] S. Humbarwadi, "Object detection with retinanet." [Online]. Available: https://keras.io/examples/vision/retinanet/

[6] E. D. GmbH, "How single-shot detector (ssd) works?" [Online]. Available: https://developers.arcgis.com/python/guide/how-ssd-works/

[7] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *CoRR*, vol. abs/1708.02002, 2017. [Online]. Available: http://arxiv.org/abs/1708.02002