Szájmaszk detektálás

Tálos Botond

Bevezetés

A projekt célja, hogy valós időben, video stream-ből kiragadott képeken megtaláljon emberi arcokat, majd eldöntse, hogy az arcokon van-e szájmaszk. A projektnek nem célja, hogy a felismerést végző program megítélje a maszk hordás helyességét (kilógó orr). Elsősorban a "visel-e maszkot?" kérdésre igyekszik minél magabiztosabb igen/nem választ adni. A fejlesztés során az arc detektáláshoz egy előre betanított neurális háló került felhasználásra, míg a szájmaszk kérdésre egy általam betanított háló válaszol. Ehhez egy publikus dataset nyújtott segítséget, amely elkülönítve tartalmaz maszkot hordó és nem hordó emberekről képeket.

Kulcsszavak: szájmaszk, neurális háló, vírus

1 Elméleti háttér

1.1 Objektum detektálás

Az első lépés, hogy a video streamből kiragadott képet megkapja egy Deep Nerual Network(DNN). A deep szó arra utal, hogy a neurális hálónak sok rétege van. Egy sikeres betanítást követően, az egyes rétegek különböző méretű jellemzők(features) detektálásra lesznek "fogékonyak".

Arcfelismerés esetén ilyen jellemző lehet egy rétegben pl. az emberi fej formája, míg egy későbbi rétegben már elkülönítve beszélhetünk pl fülről, orról. Összefoglalva, egy dnn hierarchikus módon képes complex jellemzőket magától megtalálni, amely alkalmassá teszi az összetett és nagyfokú eltérésekkel rendelkező objektumok detektálására. [8]

Az arc detektor az úgynevezett **Single Shot-Multibox Detector** módszeren alapul és ResNet-10 architektúrája van. [3]

A ResNet architektúra az úgynevezett **vanishing gradient** problémát oldja meg. Ezzel akkor lehet találkozni, amikor egy háló nagyon mély és a kiszámított gradiens nagyon közel van nullához.

Ennek oka, hogy egy mély háló esetén a gradiens értékének kiszámításához sokszor kell alkalmazni a lánc szabályt, ami azt jelenti, hogy sokszor szorzunk össze eleve pici számokat, aminek a végeredménye egy még apróbb szám. Mivel a gradiens befolyásolja a tanulás mértékét, egy aprócska érték akár meg is akadályozhatja a tanulás folyamatát.

A ResNet megoldása a következő. Ne végezzünk **back propagation-t** a hálón. Helyette ugorjunk néhány layer-t, magyarán hagyjuk ki a középen lévőket. Ezzel elkerülhető az exponenciális csökkenés a gradiens értékében, és kellően nagy marad a tanuláshoz. [23][27]

A gradient azért kiemelt szerepű, mert ennek segítségével lehetséges egy neurális háló tanítása. Lényegében egy függvény globális minimumát keressük. Ezt a folyamatot sokféleképpen is megtehetjük attól függően, hogy mekkora lépésekkel közelítjük, ezek a lépések változnak-e a tanulás során, egyszerre mennyi adatot mutatunk a hálónak stb.

Viszont mindegyik esetben közös a **gradient descent**, azaz a gradiens értékének felhasználása a minimum keresése során. A minimumot pedig azért keressük, mert a függvénynek(neurális hálónak) ott lesz a legkisebb hibája. [4]

Objektum detektálás és kép osztályozás között az egyik fontos jellemző, hogy előbbi **folytonos** jellegű, míg utóbbi **diszkrét**. Egy kép osztályozásához elegendő azt megmondani, hogy x előre definiált osztály közül, melyikbe tartozik a kép(pontosabban szólva, melyikbe a legvalószínűbb, hogy tartozik).

Tehát a döntés eredménye egy diszkrét érték. Objektum detektáláshoz, ez nem elegendő, mivel arra vagyunk kíváncsiak, hogy az objektum, a kép mely részén található. Jellemzően egy erre fejlesztett neurális háló azt mondja meg, hogy hol található az objektum helyét jelölő téglalap bal felső és jobb alsó csúcsa. [22]

1.2 Kép osztályozás

Második lépés, hogy az előző háló által megtalált arcokat megkapja egy Convolutional Neural Network(CNN), amely egy intervallumból kiragadott érték formájában közli, hogy mennyire magabiztos abban, hogy az arcon maszk található-e.

A CNN betanítása egy **supervized learning** formájában balósul meg, azaz a háló számára rendelkezésül bocsátott adatok előre meg lettek jelölve arra vonatkozóan, hogy maszkos vagy masznélküli képről beszélünk.

Egy CNN jellemzője, hogy 2 layer között nincs feltétlenül jelen az összes lehetséges összeköttetés. Ennek oka, hogy képek osztályozása esetén egy Fully Connected(FC) neurális háló rengeteg paramétert jelentene.

A teljes összeköttetés értelemszerűen több fine-tuning-ot is eredményez, azonban azt is figyelembe kell venni, hogy egy FC háló **overfitting-re** hajlamos. Egy kép osztályozásánál nem előnyös ha minden részletet figyelembe vesz a háló, amit csak talál a képen, hiszen egy objektum sokféle képpen szerepelhet a képen, így felesleges az utolsó részletig megfigyelni egy-egy képet. [19]

Tehát végeredményben az arc detektálás tartalmaz egy **regressziós** problémát, míg a maszk detektálás egy **osztályozási** probléma.

2 Megvalósítás

2.1 Arc detektálás

A video stream-ből érkező képeken egy előre betanított hálót használtam, a **caffemodel-ek** egyikét. A modellt ezen a linken lehet elérni és letölteni. A caffe egy deep learning framework. Előnye a sebesség, valamint a kötetlenség, abban az értelemben, hogy kapcsoló beállításával futtatható a modell cpu-n vagy gpu-n tetszés szerint [13]. Az általam használt modell ezen a linken érhető el.

A modell használatához egy úgynevezett **prototxt** is szükséges. Ez file a file leíja, hogy a caffemodel-nek milyen az archtektúrája: layer-ek száma, input neuronok száma, stb. Erre azért van szükség, hogy a projekt során használt opencv függvény be tudja tölteni a modellt későbbi hazsnálatra. [9] (A prototxt nem feltétlenül szükséges hogy ember számára olvasható txt legyen, itt most az.)

A beérkező kép mielőtt átmegy ezen a hálón, átméretezésre kerül, hogy megegyezzen a caffemodel input dimenzióival. Átméretezés mellett megváltoztatásra kerülnek a színcsatornák is a következő módon. Az RGB színcsatornákból kivonásra kerülnek az alábbi értékek: 104, 177, 123. Erre azért van szükség, mert a fényviszonyok, amelyben a kép készül, nagymértékben változhat. [21]

Ennek következtében a neurális háló neuronjaiban is más aktivitás értékek keletkeznek. Végeredményben ez nem kívánatos módon befolyásolja a detektálás eredményét. Annak érdekében, hogy különböző **fényviszonyok** mellett is jól teljesítsen a háló, előfeldolgozásra van szükség, amelynek eredménye egy kép, ahol az élek kevésbé függnek a megvilágítástól [21]. Az előfeldolgozás a 1. képlet alapján történik.

$$R = (R - \mu_R)/\sigma$$
$$G = (G - \mu_G)/\sigma$$
$$B = (B - \mu_B)/\sigma$$

FIGURE 1: Színcsatornák előfeldolgozása
Forrás: [21]

Az 1. képletben látható σ ebben projektben 1, tehát elhagyató, viszont említés céljából szerepel, ugyanis deep learning projektek során előfordulhat, hogy ettől eltér az értéke. Feladat függő. A μ értékeire azért esett a - 103, 177, 123 - választás, mert gépi látás projektek során gyakran alkalmazzák ezeket az értékeket.

Az eddig leírtakat a következő egy sor kód valósítja meg:

```
blob = cv2.dnn.blobFromImage(frame,
                             1.0,
                             (300, 300),
                             (104.0, 177.0, 123.0))
```

Ahol a 2. paraméter a σ, 3. a kívánt átméretezés, 4. pedig a 3 μ. Az így megkapott blob egy 4 dimenziós tömb, amely tartalmazza többet között a kép méreteit, valamint a csatornái számát [21].

2.2 Maszk detektálás

2.2.1 Környezet

A projekt során python 3.8.3-t használtam. A háló betanítását GPU-n végeztem a CUDA Toolkit segítségével. Minden további package megtalálható a requirements.txt-ben.

2.2.2 Dataset

A projekt során a következő linken elérhető dataset-et használtam fel a maszk detektáló neurális háló betanítására. Ez önmagában 1916 maszkos és 1919 masz nélküli képet tartalmaz. Az úgynevezett data augmentation során a képek száma bővítésre került. Az új képek, az eredeti másolataiból keletkeztek, enyhe változtatással.

Ezáltal nőtt a dataset mérete, másrészt a változtatások révén - forgatás, zoom, eltolás - a neurális háló rendhagyó esetekre is fel tud készülni. Végeredményben toleránsabb lesz a fokozott sokszínűséggel rendelkező dataset-nek köszönhetően [20].

2.2.3 Hiperparaméterek

A háló betanítását az **Adam** algoritmussal végeztem. Adam a [6]. forrás szerint **Stochastic Gradient Descent**(SGD) algoritmus egy kiegészítése. Viszont SGD egy jellemzője, hogy egyszerre egy adatot választ ki véletlenszerűen, kiszámítja a modell hibáját az adat függvényében, majd módosítja a súlyokat.

Véleményem szerint pontosabb, ha SGD helyett a **Mini Batch Gradient Descent**(MBGD) algoritmus kiegészítéseként tekintjük, abból kifolyólag, hogy Adam, mini batch-ekkel dolgozik a működése során. Igaz, hogy ez a méret lehet 1, amely esetben valóban SGD-ként működik. Viszont SGD-hez hasonlítva átsiklunk azon a tulajdonságán, hogy Adam, mini batch-ekkel dolgozik [5].

Adam úgynevezett **adaptive learning rate**-t alklamaz, tehát a learning rate nem konstans a futása során. Az **initial learning rate** értéke $1e-4$, azaz innen indul ki a learning rate hiperparaméter. A változást, pontosabban szólva a csökkenést az úgynevezett **decay** határozza meg.

A learning rate szabályozása azért fontos, hogy az algoritmus minél közelebb kerüljön a **veszteség függvény** globális minimumához. A learning rate kezdeti magas értéke lehetővé teszi, hogy gyorsan működjön a program, abból kifolyólag, hogy a nagy lépések miatt, kevesebb itarációra van szükség. Viszont a nagy lépések nem alkalmasak finomhangolásra, amelyre szükség van, a tanulás vége felé járva [14]. Az itt leírtakat, jobban szemlélteti a 2. ábra.

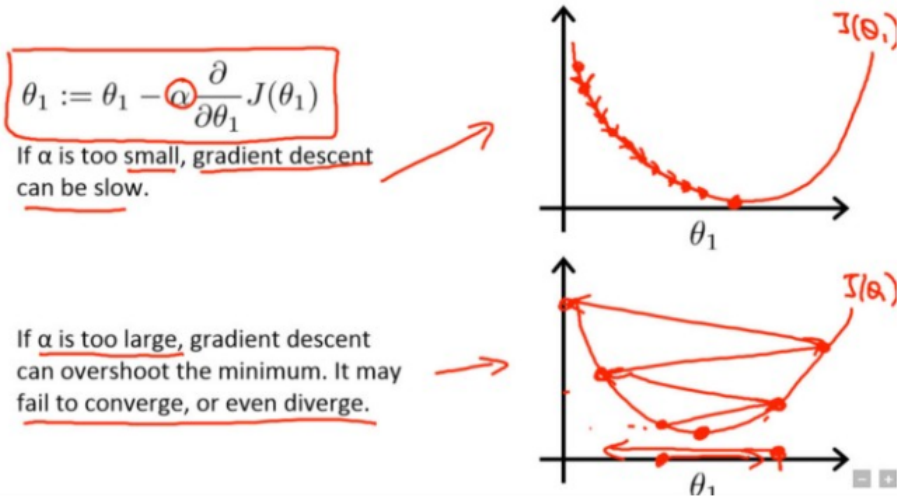

FIGURE 2: A lépés méretének jelentősége
Forrás: [14]

A decay mértékét a következőképpen határoztam meg:

```
decay = init_lr / epochs
```

Tehát a kezdeti learning rate és az **epoch** szám hányadosa. Az epoch számot 20-ra állítottam, azaz ennyiszer találkozik a model az összes adattal. Látható, hogy az epoch szám növelésével, a decay is csökken, végeredményben pedig a gradient descent sebessége is csökken, ami a tanulás sebességét határozza meg.

2.2.4 Veszteségfüggvény

A veszteség kiszámítására az úgymevezett **binary cross-entropy** függvényt használtam. Nevéből is adódóan olyan becslésekre használható fel, ahol két osztály létezik.

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^{N} y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

FIGURE 3: Binary cross-entropy
Forrás: [11]

- N: adatok száma
- y_i: target label, azaz 0 vagy 1 attól függően, hogy az i-dik adat egy maszkos kép-e
- p(y_i): a modell megfelelő osztályra vonatkozó becslése, tehát a 0 és 1 közötti magabiztossága

Vegyük észre, hogy az összeadás két tagja közül az egyik mindig kiesik, ugyanis az egyik tagban y_i, míg a másikban $1 - y_i$ szerepel. Továbbá a mínusz előjel sem véletlen a szumma előtt, mivel a 10-es alapú logaritmus negatív értékeket vesz föl, az x $\in$ [0, 1] intervallumban. [17]

A veszteségfüggvény a 4. ábrán látható módon vesz fel értékeket. [11]

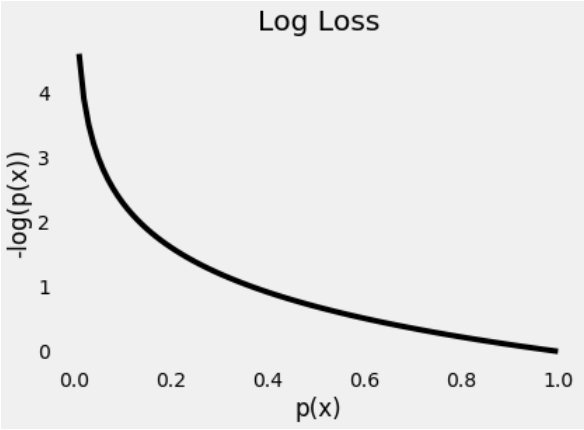

FIGURE 4: Veszteség függvény
Forrás: [11]

A függvény jelentősége abban rejlik, hogy a **modell büntetése exponenciálisan** nő a tévedéssel.

2.2.5 Train test split

A target vektor a **one-hot encoding** segítségével készült el, azaz minden target vektor pontosan egy db 1-es értéket tartalmaz, a többi 0. Ennek az a szerepe, hogy a veszteséget ki lehessen számolni a **becslés** és a target különbségével. [17]

A modell tanítását megelőzően a dataset-et két csoportra osztottam. 80%-át tanításra, fennmaradó 20%-át tesztelésre használtam. A felosztáshoz a *scikit-learn* könyvtár *train_test_split* függvényét alkalmaztam [1]. A felosztás **stratification-nel** történt, azaz a maszkos és masz nélküli képek aránya megegyezik a split után keletkező test és train halmazban [2].

2.2.6 Háló architektúra

A maszk detektálás során alkalmazott neurális háló egy része a **MobilNetV2** nevet viseli. Ez egy konvolúciós neurális háló, amelyet a Google fejlesztett ki. Rendeltetése, hogy

hatékonyságának köszönhetően kis számítási kapacitás mellett (mobil telefon) is használható legyen. [24]

Mivel egy CNN-ről van szó, a projekt szempontjából egyik lényeges tulajdonsága, hogy a filterek(vagy kernelek) értékeit magától találja ki, tehát nem egy ember által előre meghatározott értékek szerepelnek a filterekben. [7]

A háló "elejét", azaz a **base** részét előre beállított súlyokkal inicializáltam. Base alatt azt a részt értem, amely először találkozik az inputtal és a súlyai nem változnak, azaz a tanulás nem befolyásolja. A base-re kerül a **head**, amely tanulásra van kijelölve.

A base súlyok *lefagyasztását*, azért lehet megtenni, mert az alacsony szintű layer-ek nagyon általános jellemzőket detektálnak, míg a magas szintű layer-ek specifikus jellemzőket találnak meg. Tehát ez a háló működőképes a lefixált alacsony szintű súlyok és a feladatra alakított magas szintű súlyok mellett.[12]

A háló szerepe a a programban a 5. ábrán látható jól.

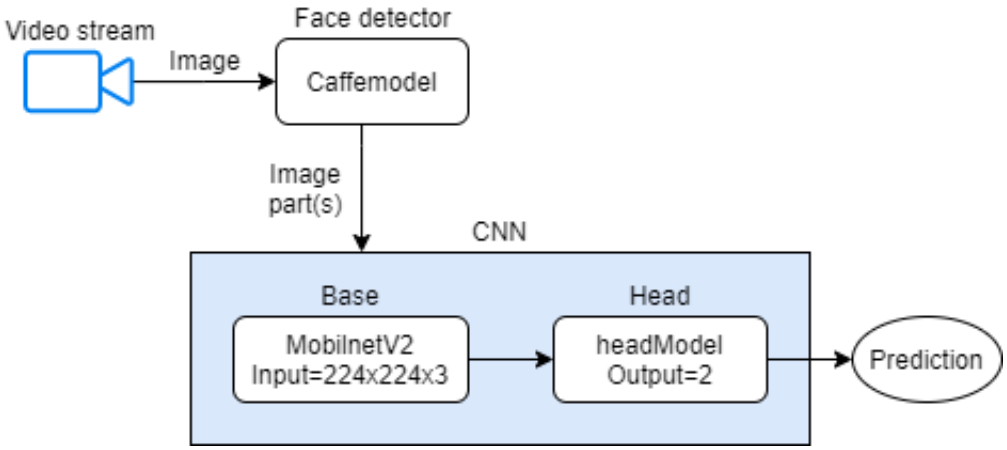

FIGURE 5: A CNN helye a programban

A head részben egy **pooling layer** is található, amelynek köszönhetően a CNN **translation invariance** előnyhöz jut. Magyarul a detektálás sikeressége nem függ az objektum elhelyezkedésétől a képen.[18]

Mivel a pooling layer egy 2D-s tömb, ki kell **lapítani**, ahhoz, hogy a következő layer neuron számára fogyasztható legyen. Ez a **flatten** lépés. [26]

Ezt követi egy 128 neuronból áll layer, ahol a neuronok **relu** aktivációs függvénnyel rendelkeznek. Relu egy nem lineáris függvény, ezáltal az egész háló fel van vértezve nemlinearitással. Ennek az a jelentősége, hogy komplex tudást - jelen esetben képek - nem lehet átadni egy hálónak, mivel a komplexitás rendszerint nem lineáris. [15]

A hálóban az úgynevezett **dropout** módszer is alkalmazva van, tehát tanulás közben adott valószínűséggel figyelmen kívül hagy súlyokat, azaz nem változik az értékük. Ezzel

megakadályozható, hogy a háló **overfitting** áldozata legyen. [16]

Háló legvégén pedig a 2 db output neruon található **softmax** aktivációs fügvénnyel. Softmax következtében a 2 neuron output összege - azaz "maszk" és "nincs maszk" valószínűség összege - mindig 1 lesz. Mivel egy arcon a maszk viselése és nem viselése egymást kizáró állapot, így előnyösebb a softmax egy olyan fügvénnyel szemben, ahol nincs megszorítás az összegre. [25]

2.2.7 Tanulás

FIGURE 6: Tanulás az eltelt epoch függvényében

3 Tesztelés

A tesztelés során a következő szempontok alapján értékeltem ki a programot: megvilágítottság, arc szöge, arcok száma, egyéb zavaró tényezők(pl. kilógó orr, maszk színe, emberi arccal hasonlatos objektumok). Egy tesztesetnél a sikeres és a sikertelen a követezőt jelenti.

- Sikeres: maszkkal és maszk nélkül is helyes eredményt ad.
- Sikertelen: ha a 2 eset közül bármelyikre hibás eredményt ad vagy nem történik detektálás.

Megvilágítottság

A 1. táblázathoz egy rövid magyarázat.
Lux: a megvilágítottság mérésére használt SI mértékegység. [10] A mérést azon a pozíción végeztem, ahol a detektálandó objektum helyezkedett el. (Másik lehetőség a kamera pozíciója lett volna)

TABLE 1: Megvilágítottság teszt

Környezet	Lux	Detektálás
kültér nappal	1537	Sikeres
beltér nappal	552	Sikeres
beltér éjszaka	27	Sikeres

Arc szöge

Ezt a tesztet több szögben, valamint maszkkal és maszk nélkül végeztem el. 0° jelenti azt, amikor arc szembe van a kamerával, 90° pedig a profil nézetet.

TABLE 2: Arc szöge teszt

Maszk	Szög	Detektálás
Van	0°	Sikeres
	45°	Sikeres
	90°	Sikertelen
Nincs	0°	Sikeres
	45°	Sikeres
	90°	Sikeres

A 2. táblázatban látható, hogy profilból nem sikerült elérni a kívánt működést, amikor az illető maszkot hord. Úgy gondolom ennek, oka, hogy maszk nélkül profilból továbbra is látszik a száj egy része, valamint egy szem. Maszk hordása esetén kizárólag egy szem áll rendelkezésére az arcdetektornak.

Arcok száma

Itt arra voltam kíváncsi, az arcok számának növelésével hogyan romlik a video stream folytonossága. A következő számokkal teszteltem a programot.

- 1: Folytonos stream, maszk eltávolítás/felvételre ember által nem érzékelhető idő alatt reagál.
- 2: Nincs különbség 1-hez képest.
- 6: A stream folytonossága megszűnik, szaggatás tapasztalható.
- 18: Itt telefonról mutattam egy képet a kamera elé. Nem nevezhető video stream-nek.
- 32: Szintén telefon segítségével oldottam meg a számot. A válaszidő itt már egyáltalán nem kielégítő.

Érdemes megemlíteni, hogy a folytonosság romlik, viszont a detektálások sikeressége független az arcok számától.

Egyéb zavaró tényezők

Amennyiben a video stream-be kerülő ember kilógó orral hordja a maszkot, a program továbbra is azt mutatja, hogy maszkot hord az illető. Az orr láthatósága nem befolyásolja maszk detektálást. Ellenben, ha az illető lehúzza az állára már jelez, hogy nem visel maszkot. Nem figyeli továbbá, hogy a maszk rögzítve van-e akár fülön vagy bármely más helyen.

A maszk színével kapcsolatban azt próbáltam, hogy kifordítva is felveszem a maszkot. Így teszteltem kék és fehér színre. Mindkettőre detektálás sikeres volt.

Az arc detektálásnak határozottan vannak gyengéi...

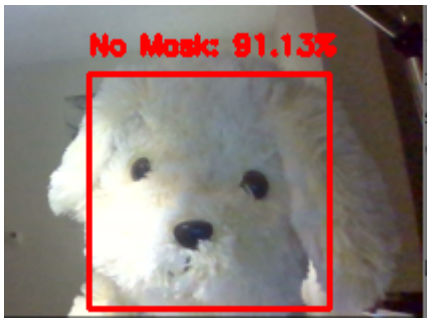

FIGURE 7: Plüss kutya maszk nélkül

4 Felhasználói útmutató

A program webkamerával rendelkező laptopon lett kipróbálva. Ettől eltérő setup-on nem. A maszk detektáláshoz a video_mask_detection.py code-on kívül szükség van a megfelelő caffemodel-re, prototxt-re, valamint mask detetktor modelre. A github repositoryban minden szükséges file megtalálható.

A program indítását parancssorból lehet megtenni a következő módon. (a requirements.txt-ben megtalálhatóak a futtatáshoz szükséges package-k python-on kívül)

```
python video_mask_detection.py
```

Lehetőség van a default-tól eltérő modelt, prototxt-t használni. Ezeket a megfelelő kapcsolók beállításával lehet megtenni, amelyről bővebb információt a következő parancs segítségével kaphatunk.

```
python video_mask_detection.py -h
```

References

[1] URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[2] URL: https://stackoverflow.com/questions/34842405/parameter-stratify-from-method-train-test-split-scikit-learn.

[3] Vardan Agarwal. *Face Detection Models: Which to Use and Why?* URL: https://towardsdatascience.com/face-detection-models-which-to-use-and-why-d263e82c302c.

[4] *Analyses of Deep Learning*. URL: https://stats385.github.io/learning.

[5] Jason Brownlee. *A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size*. URL: https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/.

[6] Jason Brownlee. *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. URL: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning.

[7] Jason Brownlee. *How Do Convolutional Layers Work in Deep Learning Neural Networks?* URL: machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/.

[8] Jason Brownlee. *What is Deep Learning?* URL: https://machinelearningmastery.com/what-is-deep-learning/.

[9] *Caffe model file formats*. URL: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781789137750/4/ch04lvl1sec31/caffe-model-file-formats.

[10] The Editors of Encyclopaedia Britannica. *Lux, unit of energy measurement*. URL: https://www.britannica.com/science/lux.

[11] Daniel Godoy. *Understanding binary cross-entropy / log loss: a visual explanation*. URL: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a.

[12] Vikas Gupta and Anastasia Murzova. *Keras Tutorial : Using pre-trained Imagenet models*. URL: https://www.learnopencv.com/keras-tutorial-using-pre-trained-imagenet-models/.

[13] Yangqing Jia. *Caffe*. URL: https://caffe.berkeleyvision.org.

[14] Suki Lau. *Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning*. URL: https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1.

[15] Z² Little. *Activation Functions (Linear/Non-linear) in Deep Learning*. URL: xzz201920.medium.com/activation-functions-linear-non-linear-in-deep-learning-relu-sigmoid-softmax-swish-leaky-relu-a6333be712ea.

[16] Cory Maklin. *Dropout Neural Network Layer In Keras Explained*. URL: https://towardsdatascience.com/machine-learning-part-20-dropout-keras-layers-explained-8c9f6dc4c9ab.

[17] Vlastimil Martinez. *Cross-entropy for classification*. URL: https://towardsdatascience.com/cross-entropy-for-classification-d98e7f974451.

[18] Divyanshu Mishra. *Translational Invariance Vs Translational Equivariance*. URL: https://towardsdatascience.com/translational-invariance-vs-translational-equivariance-f9fbc8fca63a.

[19] Santhosh Kumar Nandigama. *How is CNN better for an image classification problem over ANN?* URL: https://www.quora.com/What-is-the-difference-between-CNN-and-deep-NN-in-machine-learning.

[20] Lucas Robinet. *Data Augmentation and Handling Huge Datasets with Keras: A Simple Way*. URL: https://towardsdatascience.com/tagged/data-augmentation.

[21] Adrian Rosebrock. *Deep learning: How OpenCV's blobFromImage works*. URL: https://www.pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/.

[22] Adrian Rosebrock. *Object detection: Bounding box regression with Keras, TensorFlow, and Deep Learning*. URL: https://www.pyimagesearch.com/2020/10/05/object-detection-bounding-box-regression-with-keras-tensorflow-and-deep-learning/.

[23] Pablo Ruiz. *Understanding and visualizing ResNets*. URL: https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8.

[24] Mark Sandler and Andrew Howard. *MobileNetV2: The Next Generation of On-Device Computer Vision Networks*. URL: https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html.

[25] SuperDataScience Team. *Convolutional Neural Networks (CNN): Softmax & Cross-Entropy*. URL: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-softmax-crossentropy.

[26] SuperDataScience Team. *Convolutional Neural Networks (CNN): Step 3 - Flattening*. URL: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening.

[27] Chi-Feng Wang. *The Vanishing Gradient Problem*. URL: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484.