Pekka Siltala-Li

Neuroverkot liikennemerkkien tunnistamisessa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tutkinto-ohjelman nimi

Insinöörityö

16.5.2022

Tiivistelmä

Tekijä:	Pekka Siltala-Li
Otsikko:	Neuroverkot liikennemerkkien tunnistamisessa
Sivumäärä:	24 sivua + 1 liitettä
Aika:	16.5.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistonkehitys
Ohjaajat:	Janne Salonen Mira Myllärinen

Miten liikennemerkkejä voidaan tunnistaa tekoälyn menetelmin yksinkertaista neuroverkkoa (R-CNN) käyttämällä.

Avainsanat: Liikennemerkit, ML, R-CNN, Neural Networks

Abstract

Author:	Pekka Siltala-Li
Title:	Traffic sign recognition using neural networks
Number of Pages:	24 pages + 1 appendices
Date:	15 April 2022
Degree:	Bachelor of Engineering
Degree Programme:	Computer science
Professional Major:	Software development
Supervisors:	Janne Salonen, Head of School (ICT) Mira Myllärinen, Principal Lecturer

---

How traffic signs can be identified by artificial intelligence methods using a simple neural network (R-CNN).

Keywords: Traffic signs, ML, R-CNN

Sisällys

Lyhenteet

ORM: *Object-relational mapping*. Ohjelmiston oliomallin mukaisen esityksen kuvaaminen relaatiotietokantamallin mukaiseksi esitykseksi ja kääntäen.

TKHJ: Tietokannan hallintajärjestelmä. Ohjelmisto tiedon tehokkaan hakemisen, säilyttämisen ja päivittämisen toteuttamiseksi.

ML: Machine Learning, koneoppiminen, tekoäly

KERAS: Avoimen lähdekoodin tekoälyä hyödyntävä korkean tason kirjasto

R-CNN: Region Based Convolutional Neural Network. Neuroverkkomalli, joka on erityisen hyvä käsitellessä visuaalista informaatiota

NumPy Python -kielen kirjasto, joka tarjoaa helpon tavan käsitellä erityisesti moniulotteisia taulukoita sekä matriiseja.

Sigmoid Sigmoidifunktio on funktio, jolla on ’S’ kirjaimen muotoinen käyrä

VGG16 ILSVR kilpailun 2014 voittanut R-CNN malli kuvien tunnistamiseen.

ZCA Tietojen muunnos siten, että sen kovarianssimatriisi
Σ on identiteettimatriisi. Siksi ZCA-valkaisu (engl. *whitening*) korreloi kuvan piirteisiin.

1 Johdanto

Neuroverkot mahdollistavat monenlaisia asioita. Tässä opinnäytteessä pyritään tarkastelemaan tekoälyn menetelmiä liikennemerkkien tunnistamisessa, sekä tarkastelemaan miten tällaisen yksikertaisen konenäön malli voidaan rakentaa nykyaikaisen ohjelmistokehityksen välinein.

Liikennemerkkien kuvadata saatiin Väyläviraston palvelusta. Siellä on kaikki suomessa käytössä olevat liikennemerkit. Pian projektin edessä selvisi, että käyttämällä kaikkia liikennemerkkejä niiden augmentointi (eli pienten muunnosten ja virheiden lisääminen mallin opettamiseksi) vaati paljon enemmän resursseja kuin oli käytettävissä.

Tässä opinnäytteessä käytetään vain pientä osajoukkoa liikennemerkeistä, jotka on satunnaisesti valittu varoitus ja kieltomerkeistä.

Täydelliseen toteutukseen pitäisi huomioida myös katumerkinnät sekä merkeissä olevat tekstit, joka sinänsä on oma kokonaisuutensa (esimerkiksi lisäkilvessä olevassa kellonajassa tai opastekilven paikkakunnassa voi olla lähtökohtaisesti mikä tahansa nimi tai numerosarja)

2 Historiaa

Neuroverkot jäljittelevät hieman ihmisaivojen toimintaa, jossa aivojen *neuronit* kytkeytyvät muihin synapsien välityksellä. Aivojen sähkökemialliset prosessit ja valtava joukko kytkentöjä muodostaa valtavan monimutkaisen ristiin kytketyn verkon aistien sensorisen informaation käsittelyyn. Tekoälyn kehityksessä on sen alkuvaiheissa ajateltu, että tietojenkäsittelyn menetelmin voidaan jossain määrin jäljitellä tätä toimintaa.

Keinotekoisessa neuroverkossa, jotka voivat olla varsin yksinkertaisia aina sellaisiin asti joissa on kerroksittain tuhansia neuroneita. Ihmisaivojen

kyvykkyyteen on kuitenkin pitkä matka, aivoissa yhdellä synapsilla voi olla miljoonia yhteyksiä ja neuronien kokonaismäärä on sata miljardia.

Neuronin pääpiirteenä on sen toimintaa määrittelevä funktio. Jos se on lineaarifunktio, neuroni voi ratkaista tällöin vain lineaarisia ongelmia. Tällä on myös joitakin harvinaisempia sovelluksia, joukolla neuroneita voidaan tällöin mallintaa suurta loogisten operaatioiden (AND, OR, NOT, XOR) verkkoa. Kaikkien muiden paitsi eXclusive OR-operaattorin mallinnukseen riittää yksi neuroni.

Neuroverkkojen kehitys lähti McCulloch ja Pitts mallista, josta johti Hebb oman versionsa vuonna 1949. Tämän hyvin yksikertaisen ajatuksen päälle neuronista jatkoi Hebb 1949. Siinä oli kuitenkin ongelmana, että neuronien välistä liikennettä suosivat painoarvot kykenivät vain nousemaan, eivätkä koskaan laskemaan. Tämän korjasi Rosenblattin ajatus perseptronista 1958. Tässä vaiheessa kyseessä olivat kuitenkin vain lineaarifunktiolla toimivat neuroverkot, jolloin ne kykenivät ratkaisemaan vain lineaarisia ongelmia.

Hopfieldin esitellessä 1982 sigmoid-funktion, ja Rumelhart McClellandin esitellessä takaisinkytkennän (engl. *backpropgation)* asiat muuttuivat. Tämä mahdollisti sekä epälineaaristen ongelmien ratkaisemisen, että tuloksessa olevan virheen propagoimisen verkossa takaisinpäin niin, että se vaikutti neuronien välisiin painoihin.

Perseptroneilla saadaan muodostettua nykyaikainen yksinkertainen neuroverkko. Joskin siinä on joukko ongelmia kuten paikallisen minimin optimointiongelma, näitä on ratkottu konjugaattigradienteilla ja käyttämällä stokastisia menetelmiä.

Siinä missä aivoissa synapsien välistä liikenteessä yhteyksissä toimivat aksonit ja dendriitti, neuroverkoissa samaa hallitaan painotuksilla. Kun neuroverkkoa opetetaan, sen painotukset suosivat joitakin yhteyksiä toisten kustannuksella, ja yksinkertaistettuna tämä tila on malli, jota voidaan hyödyntää.

Neuroverkoista on varsin erilaisia toteutuksia, niin täysin kuin osittain kytkettyjä, synkronisia tai asynkronisia, staattisia tai dynaamisia, ja tärkeimpänä ehkä ohjatun, että syväoppimisen (engl. supervised unsupervised) mallit. Toisessa meillä on opetusdatalle nimet, joita kutsutaan labeleiksi, kun taas syväoppimisessa malli itse päättelee korrelaatiot.

3 Liikennemerkit

Liikennemerkit ovat liikenteenohjauksen, liikenneturvallisuuden sekä reitittämisen perusasioita tieliikenteessä. Niiden käyttö perustuu tieliikennelakiin.

Liikennemerkeillä on erilaisia luokkia kuten vaara, kielto ja opastemerkit. Osa merkeistä on varsin ylikansallisia muodoltaan, joskin kansallisissa standardeissa väritys saattaa vaihdella. Ajoneuvon kuljettajilta vaaditaan kuljettatutkinnossa osoitus yleisimpien liikennemerkkien tuntemisesta.

Nykyaikaisen auton varusteisiin kuuluu erilaisia avusteita. Yksi yleistynyt lisävaruste on automaattinen liikennemerkkien tunnistus. Vielä jokin aika sitten nämä kuuluivat enemmän Premium -luokan ajoneuvojen varustukseen, mutta yleistyvät nyt myös keskiluokan autoissa.

Näiden etuna on, että ajoneuvon kuljettaja, vaikka jättäisi huomioitta varoitusmerkkiin, ajoneuvo voi huomata sen ja toimia tilanteen vaatimalla tavalla; esimerkiksi hidastamalla ajonopeutta.

Ongelmana vaihtelevassa ilmastossa ovat olleet sekä kameroiden peittyminen osin tai kokonaan lumeen tai tiestä nousevaan kuraan. Tässä opinnäytteessä kuvatuin menetelmiä jatkokehittämällä lienee mahdollista parantaa konenäön tunnistamista silloin, kun merkki tai näkijä näkee vain osan kohteesta.

3.1 Työvälineiden valinta

Ohjelmointikieleksi valittiin Python, koska se on varsin vakiintunut koneoppimisen kieli laajasta ja vakiintuneesta kirjastoista johtuen. Se on hyvin suosittu, että nopea oppia, jolloin opinnäytteen jatkojalostaminen opiskelumateriaaliksi ja hyöty mahdollisimman laajalle yleisölle tulee mahdolliseksi.

On mahdollista toteuttaa neuroverkko hyvin matalla tasolla, jolloin lähestymistapa on alhaalta ylös, mutta tässä kuvataan yleinen toiminta erikseen ja hyödynnetään korkeamman tason kirjastoja (Keras/Tensorflow)

4 Datan lataus

Sen jälkeen, kun varsinainen liikennemerkkien esimerkkidata on ladattu väylävirastolta, se pitää augmentoida. Augentoinnin syy on keinotekoinen opetusmateriaalin kasvattaminen. Mallia voi opettaa hyvin pienelläkin määrällä, mutta tällöin sen tarkkuus on hyvin huono.

Itse opetusdata on järjestetty seuraavasti

```
.
├── data
│   ├── augmented
│   │   ├── A4
│   │   ├── A6
│   │   ├── A8
│   │   ├── A9
│   │   ├── B1
│   │   ├── B3
│   │   ├── B4
│   │   ├── B5
│   │   ├── B7
│   │   └── C5
│   └── signs
│       ├── A4
│       ├── A6
│       ├── A8
│       ├── A9
│       ├── B1
│       ├── B3
│       ├── B4
│       ├── B5
│       ├── B7
│       └── C5
├── notebooks
```

Varsinainen ohjelmakoodi sijaitsee notebooks -hakemiston alla, ja käsiteltävät kuvat ovat jaoteltuja data -hakemiston alle. Alkuperäiset kuvat ovat data/signs -hakemistotason alla kukin omassa hakemistossaan. Keras tekee automaattisesti hakemistoista luokkia, joita voi hyödyntää jatkokehityksessä. Tällainen ratkaisu säästää sekä aikaa ohjelmistokehityksessä, että vähentää oman ratkaisun virheiden korjaamista. Toisaalta se sitoo organisoimaan asiat tietyllä tavalla.

Augmented -hakemistoon sijoitetaan valmiit muunnetut kuvat. Jokaista signs-hakemiston alta löytynyttä alihakemiston kuvaa kohden sijoitetaan vastaavaan

alihakemistoon joukko satunnaisesti halutulla tavalla muunneltuja (augmentoituja) kuvia.

4.1 Satunnaisen tiedon lisääminen kuviin – augmentointi

Augmentointi on tekniikka, jolla voidaan keinotekoisesti laajentaa tietojoukon kokoa luomalla tietoaineistoon muokattuja versioita. Tämä tekniikka on erityisesti käytössä visuaalista informaatiota käsitelevissä sovelluksissa.

Vahvistetun sekä syväoppimisen hermoverkkomallien opettamisessa suuret datajoukot korreloivat parempiin malleihin. Augmentaatiotekniikoilla voidaan laajentaa joukkoa pienillä muunnelmilla, jotka parantavat mallin osumatarkkuutta.

Augmentoitiin valittiin joukko muuttujia

Taulukko 1. Lisätyn todellisuuden muuttujat ja arvot

Metodi	**Arvo**
Rotation_range	180
Shear_range	0.2
Zoom_range	0.5
Rescale	1./255
Width_shift_range	0.2
Height_shift_range	0.2
Horizontal_flip	False
Brightness_range	0.1/1.8
Gaussian noise	0-50

Lisäksi kaikki kuvat skaalattiin staattiseen kokon 128 * 128 pikseliä. Opetuksessa käytettiin värinkuvia (RGB) joissa kolme kanavaa.

Huomioitavaa, että kun käytetään kameroita tai kuvanläheitä, jotka operoivat muun kuin näkyvän valon alueella, on tarkoituksenmukaista käyttää mustavalkokuvia opettamiseen koska niissä kanavia on vain yksi (harmaasävykuva)

Rotation range määrittää alueen, jolla kuvaa käännetään satunnaisesti keskipisteensä ympäri.

Shear range määrittää määrän (20 %) mitä kuvaa voidaan kääntää ”vetämällä kulmasta” jolla voidaan simuloida hieman tilannetta, että kuva nähdään viistosta.

Zoom range määrittää kuvan näkemistä eri etäisyyksillä.

Rescale määrittää millä väriarvojen alueella toimitaan. Koska RGB kuva on tässä tapauksessa 8-bittinen, jokainen kanava voi saada arvot 0–255.

Width_shift_range tarkoittaa, että kuvaa siirretään keskikohdasta sivuun. Myös niin että vain pieni osa kuvasta näkyy, jolla voidaan simuloida tilannetta, että nähtävä asia on vain osittain nähtävissä.

Height_shift_range tarkoittaa, että kuvaa siirretään keskikohdasta pystysuunnassa, aivan kuten mainittu edellä.

Horizontal_flip kontrolloi, tehdäänkö kuvasta peilikuva. Jossain tapauksissa siitä voi olla hyötyä, joten peiliobjektiivien kanssa kuvan näkyessä kennolle käänteisenä, tässä siitä ei nähty vastaavaa hyötyä.

Brightness_range määrittää vaihteluvälin kuvan kirkkaudelle. Tällä voidaan simuloida tilannetta, että merkki nähdään joko hyvin hämärässä tai hyvin valoisassa.

Gaussian_noise on apufunktio, joka lisää kuvaan kohinaa. Erilaisilla suodattimilla kuten kohina, voidaan simuloida tilannetta, että kuva on epätarkka johtuen

ympäristön ominaisuuksista (sumu, sade, lumisade). Tässä käytettiin perus-muotoista satunnaista kohinaa.

Augmentoinnin hyöty on myös siinä, että mallista ei tule ylisopiva (engl. overfit)

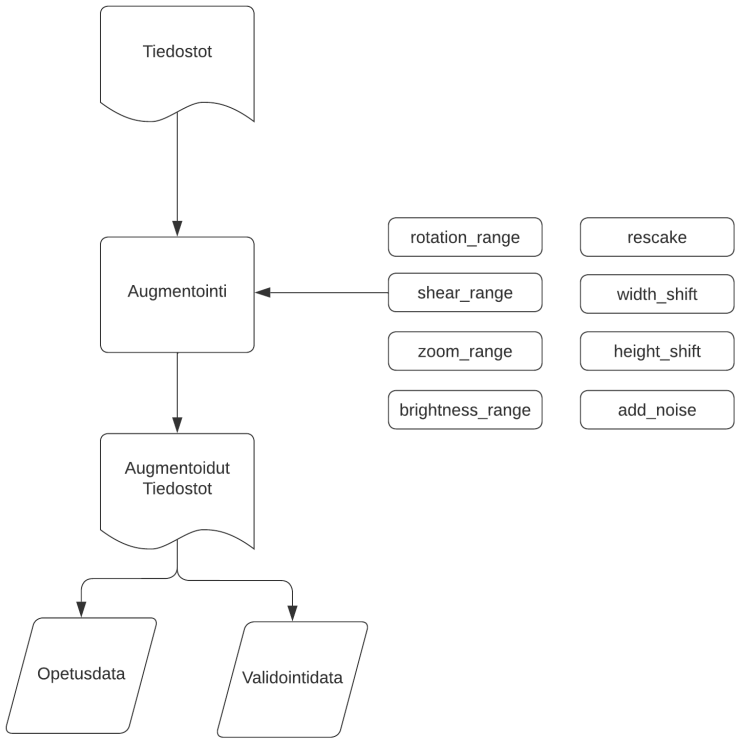
Augmentoinnin hyötyjä ovat

- näyte ja ominaisuuskohtainen vakiointi
- kuvan satunnainen kierto, siirtäminen, leikkaus tai zoomaus
- vaaka- ja pystysuora kääntö
- ZCA -valkaisu
- kuvan mittasuhteiden muuttaminen
- kuvien tallentaminen automaattisesti levylle

Jotkut kirjastot auttavat lisäksi kuvien tallentamisessa levylle. Keraksen kirjasto ei saa valtavaa määrää erilaisia temppuja, mutta sen etuna on esimerkiksi la-bel-tietojen käsittely suoraan alihakemistoista mikä helpottaa pienten ohjelmien kirjoittamista.

4.2 Tiedon prosessointi

Kuva 1 - latausprosessin prosessikuvaus



Kun data on saatu valmisteltua, se on vakiomuotoista ja valmis prosessoitavaksi.

Pythonissa luodaan ensin Datagen tietue. Augementointi auttaa myös välttämään mallin ylisopivuutta (overfit). Kuvatietojen lisääminen on yksi tapa kiertää ongelmaa, mutta ei täydellinen - sillä augmentoitujen kuvien välillä on edelleen korrelaatio.

```
datagen = ImageDataGenerator(
        rotation_range = 180,
        shear_range = 0.2,
        zoom_range = 0.5,
        rescale=1./255,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip = False,
        brightness_range = (0.1, 1.8),
        preprocessing_function=add_noise)
```

Kokeillaan luoda myös gaussian noise -kohinan luomiseen erillinen apufunktio

```
def add_noise(img):
    VARIABILITY = 50
    deviation = VARIABILITY*random.random()
    noise = np.random.normal(0, deviation, img.shape)
    img += noise
    np.clip(img, 0., 255.)
    return img
```

Alihakemistot käydään läpi yksitellen, määritellään pääosan augmentoinnista hoitava apufunktio gen_train_data, joka ottaa parametrikseen luettavien tiedostojen hakemiston, että kirjoitettavien tiedostojen hakemiston, johon augmentoiniin tulos tallennetaan.

```
def gen_train_data(indir, outdir):
    image_directory = indir
    SIZE = 128

dataset = []

    my_images = os.listdir(image_directory)
```

Tallennetaan myös paikalliseen muuttujiin sisään tulevien tiedostojen polku, että kuvien skaalauksessa käytettävä staattinen koko pikseleinä.

Dataset on tyhjä lista, johon tallennetaan NumPy -muodossa olevat matriisit kuvien lukemisen jälkeen. Tämän jälkeen luetaan my_images muuttujaan sisään tulevien tiedostojen lista.

```
for image_name in my_images:
        image = io.imread(os.path.join(image_directory,image_name))
        image = Image.fromarray(image, 'RGB')
        image = image.resize((SIZE,SIZE))
        dataset.append(np.array(image))
```

Käydään läpi tiedostonimien lista yksitellen, lisäten ne koon muunnoksen jälkeen edellä esiteltyyn dataset -listaan.

Keraksen kirjastoissa on varsin monipuolinen joukko erilaisia apuvälineitä, tässä hyödynnämme flow -funktiota, joka tekee varsin paljon varsinaista työstä. Sen kutsuminen on varsin suoraviivaista.

```
x = np.array(dataset)
 i = 0

 for batch in datagen.flow(x, batch_size=1,
                           save_to_dir= outdir,
                           save_prefix='au_',
                           save_format='jpg'):
     i += 1
     if i >=1000:
         break
```

Käydään läpi dataset listaan osoittava muuttuja ja tämä tallentaa automaattisesti tuloksen kohdehakemistoon lisäten tiedostonimeen prefiksin ’au_' . Keras itsessään satunnaistaa tarvittavat operaatiot määritellyn datagen -olion mukaisesti, tässä muuttuja i kontrolloi iteraatioiden määrää määräten, että jokaisesta kuvasta tehdään tuhat (1000) erilaista versiota.

Lopuksi kutsutaan tätä sekä apufunktioita antamalle niille yksikertaisesti lähde- ja kohdehakemistojen nimet

```
gen_train_data('../data/signs/validation/A4','../data/augmented/A4')
gen_train_data('../data/signs/validation/A6','../data/augmented/A6')
….
```

Tuloksena projektin augmented -hakemistot täyttyvät satunnaisesti muunnelluista kuvista.

Tämän voisi tehdä myös yhdessä itse opetusvaiheen kanssa, mutta tässä on katsottu, että tarkoituksenmukaista luoda ensin staattinen opetusmateriaali. Ympäristöissä, joissa laskenta ja i/o -resurssit ovat korkeammalla tasolla, ja augmentointi ei kuluta merkittävästi aikaa, voidaan käyttää myös muita lähestymistapoja.

Kaiken kaikkiaan 10 eri liikennemerkin mallikuvista on muodostunut näin 10 000 kuvan materiaali mallin opettamista varten.

5 Datan lataus

Konemallien opettamisessa hyväksi usein osoittautuva jako on käyttää 80/20 jakoa, eli datasta 80 prosenttia käytetään mallin opettamiseen ja 20 prosentilla tehdään tarkistus vastaako se ennustettua.

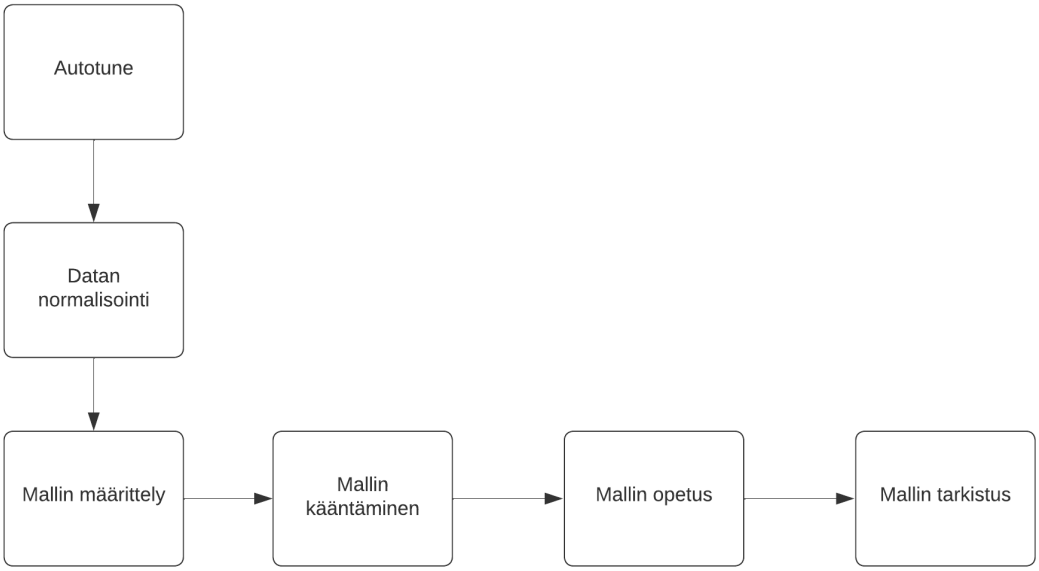
Tässä opinnäytteessä valittiin lähtökohdaksi käyttää 80 prosenttia augmentoidusta datasta opettamiseen, ja 20 prosenttia mallin validointiin.

Itse opetusprosessi voidaan jaotella siten, että ensin tapahtuu prosessin optimointi. Tässä vaiheessa opetusdataa sekoitetaan, ja sitä ladataan valmiiksi puskureihin nopeamman käsittelyn mahdollistamiseksi.

Datan normalisointivaiheessa kuvien intensiteettialueet asetetaan välinne 0..1. Näin saadaan kuvainformaation kontrasti maksimoitua opetusta varten.

Sitä seuraavat mallin määrittely, kääntäminen, opetus ja mallin tarkkuuden tarkistus vasten ennalta määriteltyä joukkoa validointidataa.

Kuva 2 - datan käsittelyn prosessimalli



Itse mallissa on kolme convulotional -tasoa ja sen poikkileikkaus näyttää tältä

Konvoluutiohermoverkot sopivat erityisen hyvin matriisimuotoisen informaation kuten sensoreilta tulevan kuvadatan analysoimiseen, kuin myös äänen tai tekstin analysointiin.

Ne ratkaisevat ongelman, jossa täysin kytketyn neuroverkon neuronit ovat täysin toisiinsa kytkettyjä. Tällöin 256 x 256 mustavalkoinen kuva sisältää jo

256*256 = 65536 pikseliä. Tällaisen käsittelyyn vaadittavan täysin kytketyn neuroverkon koko olisi varsin suuri, esimerkiksi 1000 neuronin kerros toiseen 1000 neuronin kerrokseen vaatii miljoona liitosta. Tällainen on myös laskennallisesti erittäin raskasta.

5.1 Mallin opettaminen

Kerrotaan että opetusdatan nimeäminen otetaan automaattisesti hakemistojen nimistä, kuvat ovat värillisiä, työstetään 32 kuvan joukoissa, koko jne. Validation_split kontrolloi kuinka suurta joukkoa kaikista kuvista käytetään. Se on arvo välillä 0 ..1 eli 0.8 tarkoittaa 8/10 tai 80 %. Kerromme myös, että kyseessä on nimenomaan opettamiseen tarkoitettu joukko.

```
training_dataset = tf.keras.preprocessing.image_dataset_from_direc-
tory(
     training_directory,
     labels='inferred',
     label_mode='int',
     class_names=None,
     color_mode='rgb',
     batch_size=32,
     image_size=(128, 128),
     shuffle=False,
     seed=None,
     validation_split=0.8,
     subset='training',
     interpolation='bilinear',
     follow_links=False
)
```

Esittelemme toisen joukon validointia varten, jota käytetään mallin tarkkuuden testaamiseen. Tulostamme myös hieman tietoa joukoista mitä lataamme.

```
validation_dataset = tf.keras.preprocessing.image_dataset_from_direc-
tory(
     validation_directory,
     labels='inferred',
     label_mode='int',
     class_names=None,
     color_mode='rgb',
     batch_size=32,
     image_size=(128, 128),
     shuffle=False,
     seed=None,
    validation_split=0.2,
     subset='validation',
     interpolation='bilinear',
     follow_links=False
)

class_names = training_dataset.class_names
print(class_names)
```

Tässä vaiheessa ohjelma tulostaa meille, kuinka montaa tiedostoa se aikoo käyttää opettamiseen sekä validointiin, ja montako eri luokkaa se on tunnistanut. Luokkien nimet vastaavat Väyläviraston liikennemerkkien virallista nimikointia.

Found 10000 files belonging to 10 classes. Using 8000 files for training.
Found 10000 files belonging to 10 classes. Using 2000 files for validation.
['A4', 'A6', 'A8', 'A9', 'B1', 'B3', 'B4', 'B5', 'B7', 'C5']

Voimme tarkastella osajoukkoa jostakin kuivasta ja todeta että siellä on eritavoin muunneltuja versioita samasta liikennemerkistä

Kuva 3 - muunneltuja liikennemerkkejä

Mallin opettamisessa alussa käytettiin kolmea Convulotional -tasoa, joiden aktivointifunktiona oli RelU. Näiden välissä oli Max2Dpooling -taso. Sen tehtävänä on alas samplaus, eli syötteen (korkeus ja leveys) muokkaus jokaiselle sisääntulolle. Tavallaan CNN toimii siten, että osa kerroksista löytää kuvasta yksityiskohtia, ja isommat kerrokset puolestaan kokoavat ne yhteen.

Lopuksi mallissa on kaksi täysin kytkettyä dense -tasoa, jossa biasit painotetaan kahdessa täysin kytketyssä tasossa ja lopuksi tuotetaan tulos luokkien määrään.

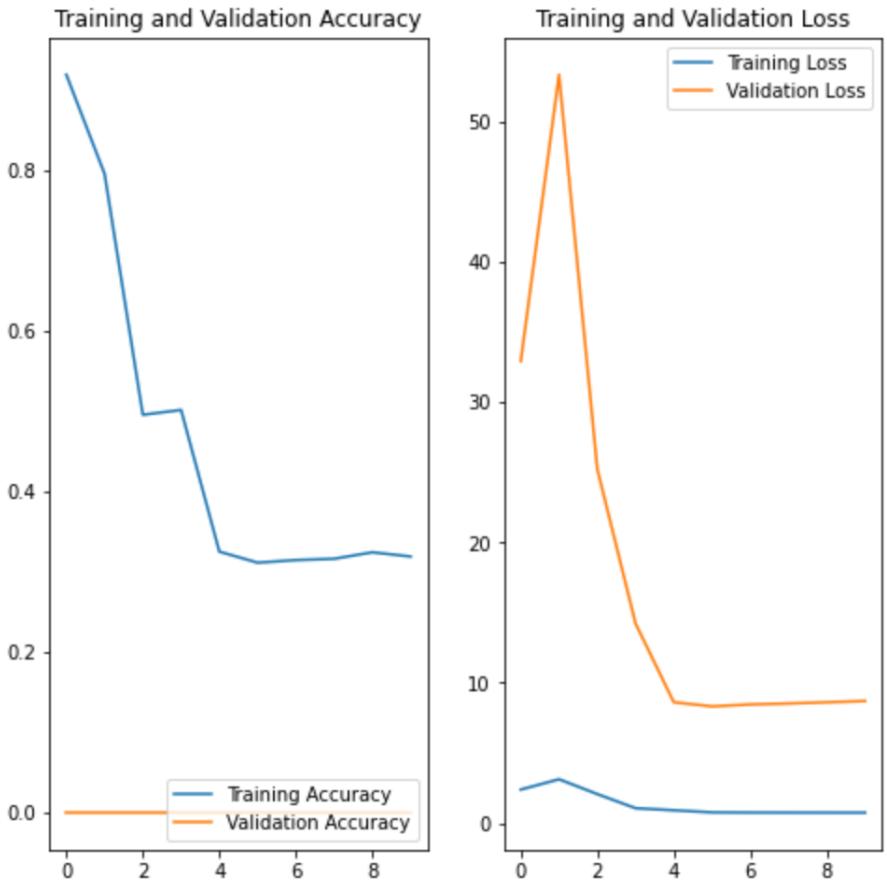
Monimutkaisemmissa valmiissa malleissa tasoja saattaa olla paljonkin, esimerkiksi Googlen kuvantunnistuksessa käyttämässä mallissa on satoja tasoja.

Mallia tarkastellessa näemme kunkin tason koon, sekä parametrimäärän. Tässä kuvauksessa näemme selvästi miten CNN parametrimäärä pysyy suhteellisen alhaisena aina Dense -kerroksille asti (jossa muodostuu edellisen kerroksen ja sen välillä niiden tulon määrä kytkentöjä, 16384*128 = 2097280)

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_13 (Rescaling)     (None, 128, 128, 3)       0

 conv2d_22 (Conv2D)           (None, 128, 128, 16)      448

 max_pooling2d_22 (MaxPoolin  (None, 64, 64, 16)        0
 g2D)

 conv2d_23 (Conv2D)           (None, 64, 64, 32)        4640

 max_pooling2d_23 (MaxPoolin  (None, 32, 32, 32)        0
 g2D)

 conv2d_24 (Conv2D)           (None, 32, 32, 64)        18496

 max_pooling2d_24 (MaxPoolin  (None, 16, 16, 64)        0
 g2D)

 dropout_2 (Dropout)          (None, 16, 16, 64)        0

 flatten_6 (Flatten)          (None, 16384)             0

 dense_12 (Dense)             (None, 128)               2097280

 dense_13 (Dense)             (None, 10)                1290

=================================================================
Total params: 2,122,154
Trainable params: 2,122,154
Non-trainable params: 0
```

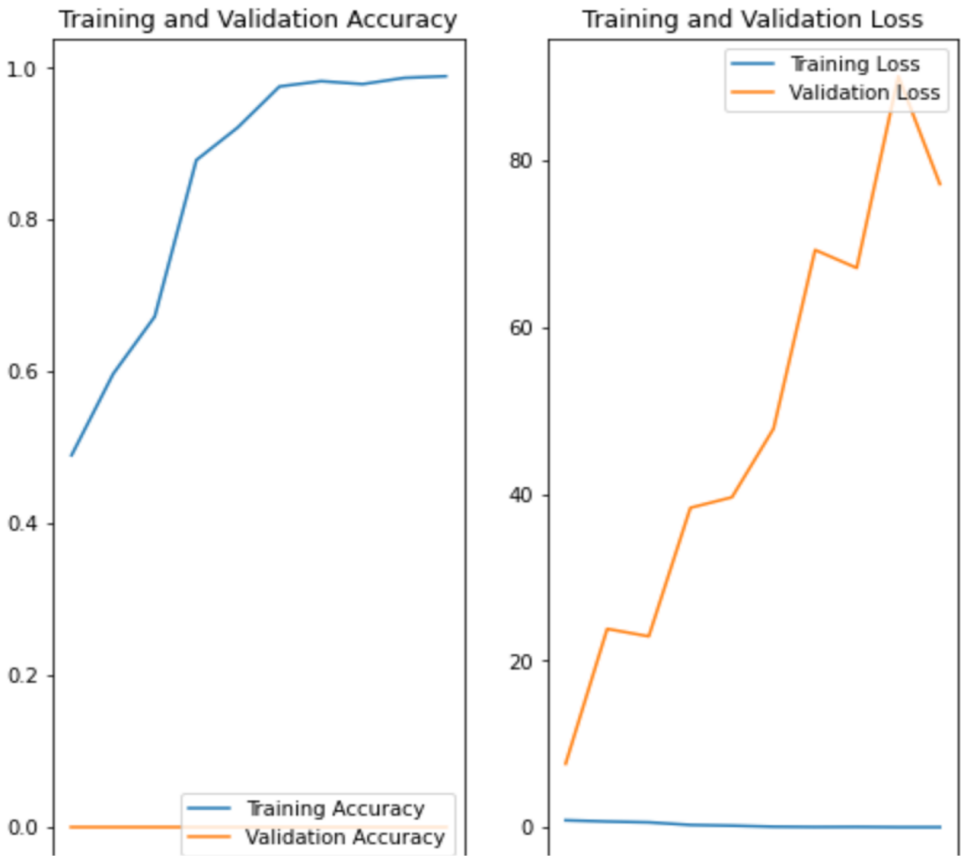
Kun mallia opetetaan maltillisella kymmenellä iteraatiokierroksella, saadaan varsin surkea tulos

Kuva 4 - ensimmäisen konfiguraation huono tulos



Ylisopiminen (overfit) ei ole tässä ongelma, mutta selkeästi ongelmaksi muodostuu tarkkuus. Tämän mallin tarkkuus on vain 0.3 eli noin 30 %. Se ei ole riittävä merkkien tunnistamisen vähimmäistarkkuudeksi (80–90 %)

Kun opetusmateriaalin määrä kaksinkertaistetaan, muuttuu tulos seuraavalla tavalla



Validation accuracy on huomattavan alhainen. Yleensä tästä selvitään siten, että lisätään merkittävästi opetusmateriaalia. Voidaan myös samanaikaisesti vähentää materiaalin entropian määrää.

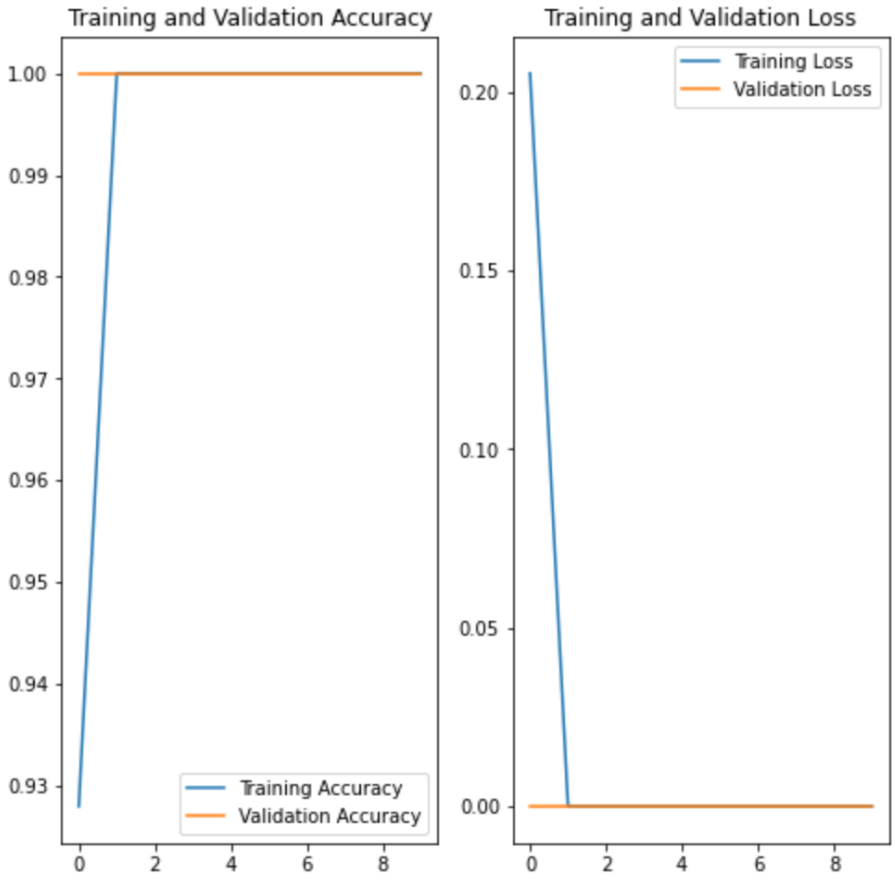
Materiaalin augmentointi lisää sen entropiaa, erilaisia permutaatioita alkuperäisestä versiosta. Ei ole kuitenkaan itsestään selvää, että kaikki muunnokset ovat samanarvoisia. Toiset ovat neuroverkolle helpompia ja toiset vaikeampia. Aivan kuten tietynlaiset kerrokset voivat olla parhaita juuri pystysuuntaisten linjojen löytämiseen kuvasta.

Asetetaan hypoteesi, että ongelma tässä muodostuu sekä merkkien entropiasta ja määrästä. Joitakin tuhansia on liian vähän, kun joukossa on liian paljon

augmentoitimuutoksia. Kun käytetään vain kymmenen asteen kiertoa ja matillista etäisyyden vaihtelua. Testataan hypoteesi.

Kuva 5 - kuva vähemmän muunnelluista kuvista

Kuva 6 - laskennan tulos muunnosten vähennyksen jälkeen



Hypoteesin testauksessa selviää, että vähentämällä muunnoksia liikaa, saadaan helposti ’overfit’ tilanne, joka sekin on varsin epäedullinen.

Ylioppiminen on sen vuoksi ongelmallista, että vaikka se sopisi erinomaisesti testidatan kanssa, se todennäköisesti ei toimisi yhtä hyvin uuden datan kanssa. Liikaa täydellisyyttä mallin kanssa siis kannattaa välttää, etenkin jos opetusdataa ei ole kovinkaan paljon.

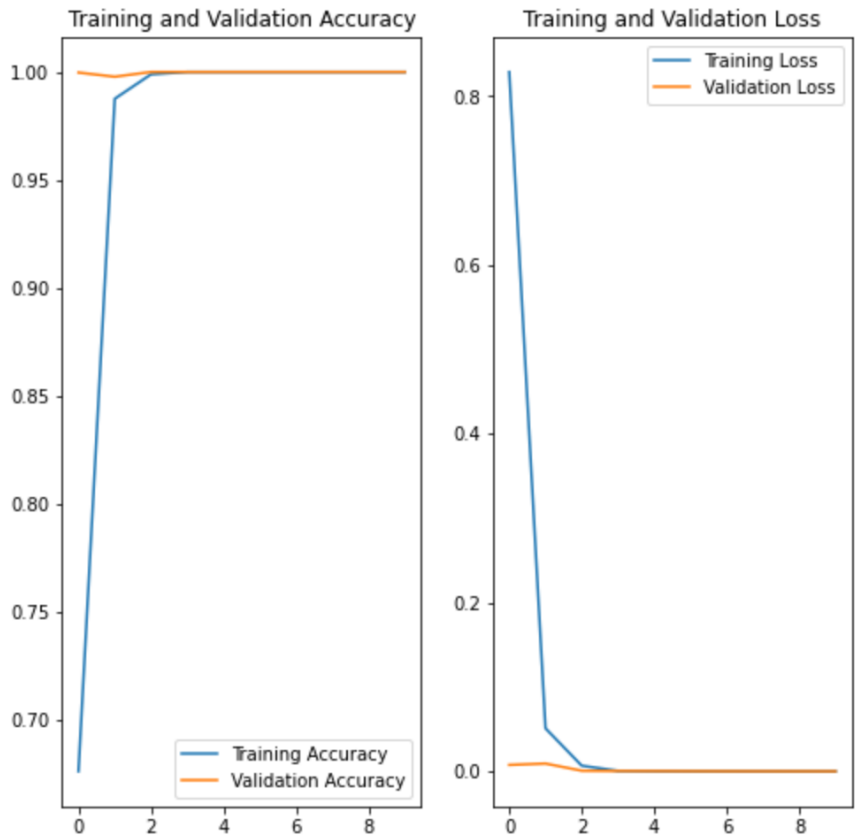
Epochien määrä on jatkuvasti pysynyt samana eikä muihin parametreihin koskettu. Tähänkin pätee vanha insinööritieteiden totuus, että mitataksesi jotakin muuta vain yhtä asiaa kerrallaan.

Tästä voi vetää myös johtopäätöksen, että liikennemerkkien opettaminen riittävillä resursseilla on varsin mutkatonta, mutta erilaisten variaatioiden kasvaessa myös opetusdatan luonti ja riittävät laskentaresussit ovat avainasemassa.

Saadaksemme selville, mikä operaatioista oli ”kallis” R-CNN mallille, käännettiin eri muunnoksia päälle ja pois -periaatteella kunnes tulos alkoi paranemaan.

Selvisi, että kun käännetään kaikki muu augmentointi takaisin päälle poissulkien kohina (gaussian noise) saadaan varsin erinomainen tulos hyvin vähillä opetuskierroksilla.

Kuva 7 - kuva tyydyttävästä opetustuloksesta



6 Johtopäätöksiä

Koska kohina on tekijä, joka vaikuttaa tekevän tunnistamisesta paljon haasteellisempaa, voimme tehdä tästä joitakin johtopäätöksiä.

Ensinkin tosielämän sovelluksissa kuvakennon herkkyys hämärissä olosuhteissa lisää kohinaa. Se johtuu kuvasensorin sähköisen herkkyyden ominaisuuksista. Aivan kuten hämärässä otetussa kännykkäkuvassa musta ei ole enää mustaa, vaan se koostuu värillisten pisteiden pilvistä.

Samoin monet ilmankehän ilmiöt voivat muistuttaa sitä kuten vesi- tai lumisade. Näin ollen ratkaisujen tehokkuuden hyviä mittareita ovat tilanteet, joissa kuvantunnistus altistuu sään tai hämärän olosuhteisiin.

Kirkkaassa säässä ja kohtuullisella optiikalla näiden mallien tehokkuus on huippuluokkaa, ja niin liikennemerkkien kuin rekisterikilpien automaattinen tunnistaminen on suhteellisen suoraviivaista.

Opettamalla mallia riittävästi on tätä varmasti mahdollista kompensoida, mutta samalla opetusmateriaalin määrää pitää merkittävästi lisätä.

Samoin riittävän hyvä neuroverkon rakentaminen ja opettaminen on haasteellista rajallisin resurssein. Suositteleva lähtökohta riittävälle tarkkuudelle saadaan käyttämällä valmiita neuroverkkomalleja kuten VGG16. Oppimistarkoituksessa oman rakentaminen toki puolustaa asemaansa, vaikka se olisi tehokkuudeltaan varsin vajavainen.

Lähteet

Aaltonen, Pietu. 2019. Tutkiva kirjoittaja ammattikorkeakoulussa. Opinnäytetyö. Metropolia Ammattikorkeakoulu. Theseus-tietokanta.

Alexia Audevart, Konrad Banachewicz, Luca Massaron. 2021. Machine Learning Using TensorFlow Cookbook. Packt Publishing.

Aziz, Lubna & Salam, Md Sah & Sheikh, Usman & Ayub, Sara. 2020. Exploring Deep Learning-Based Architecture, Strategies, Applications and Current Trends in Generic Object Detection: A Comprehensive Review. IEEE Access. 8. 170461-170495. DOI: 10.1109/ACCESS.2020.3021508.

Jason Brownlee. 2019. How to Configure Image Data Augmentation in Keras. Machine Learning Mastery, April 12th 2019. URI: https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

KC Tung. 2021. TensorFlow 2 Pocket Reference. O'Reilly Media.

Mohit Sewak, Md. Rezaul Karim, Pradeep Pujari. 2018. Practical Convolutional Neural Networks. Packt Publishing.

Matthew Moocarme, Mahla Abdolahnejad, Ritesh Bhagwat. 2020. The Deep Learning with Keras Workshop. Packt Publishing.

Oppivainen, Sanelma. 2020. Opinnäytetyön raportointiopas. Helsinki: Kaarikustantamo.

Soon Yau Cheong. 2020. Hands-On Image Generation with TensorFlow. Packt Publishing.

Lähdekoodi

```
from keras.preprocessing.image import ImageDataGenerator
from skimage import io
import numpy as np
import os
from PIL import Image
import random



def add_noise(img):
    '''Add random noise to an image'''
    VARIABILITY = 50
    deviation = VARIABILITY*random.random()
    noise = np.random.normal(0, deviation, img.shape)
    img += noise
    np.clip(img, 0., 255.)
    return img

datagen = ImageDataGenerator(
        rotation_range = 25,
        shear_range = 0.1,
        zoom_range = 0.1,
        rescale=1./255,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip = False,
        brightness_range = (0.1, 1.8))
#        preprocessing_function=add_noise)

def gen_train_data(indir, outdir):
    image_directory = indir
    SIZE = 128

    dataset = []

    my_images = os.listdir(image_directory)

    for image_name in my_images:
        image = io.imread(os.path.join(image_directory,image_name))
        image = Image.fromarray(image, 'RGB')
        image = image.resize((SIZE,SIZE))
        dataset.append(np.array(image))


    x = np.array(dataset)
    i = 0

    for batch in datagen.flow(x, batch_size=1,
                          save_to_dir= outdir,
                          save_prefix='au_',
                          save_format='jpg'):
        i += 1
        if i >=2000:
            break

gen_train_data('../data/signs/validation/A4','../data/augmented/A4')
gen_train_data('../data/signs/validation/A6','../data/augmented/A6')
gen_train_data('../data/signs/validation/A8','../data/augmented/A8')
gen_train_data('../data/signs/validation/A9','../data/augmented/A9')
gen_train_data('../data/signs/validation/B1','../data/augmented/B1')
```

```
gen_train_data('../data/signs/validation/B3','../data/augmented/B3')
gen_train_data('../data/signs/validation/B4','../data/augmented/B4')
gen_train_data('../data/signs/validation/B5','../data/augmented/B5')
gen_train_data('../data/signs/validation/B7','../data/augmented/B7')
gen_train_data('../data/signs/validation/C5','../data/augmented/C5')

import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

training_directory = '../data/augmented'
validation_directory = '../data/augmented'

training_dataset =
tf.keras.preprocessing.image_dataset_from_directory(
    training_directory, labels='inferred', label_mode='int',
class_names=None,
    color_mode='rgb', batch_size=32, image_size=(128, 128),
shuffle=True, seed=1,
    validation_split=0.8, subset='training', interpolation='bilinear',
follow_links=False
)


validation_dataset =
tf.keras.preprocessing.image_dataset_from_directory(
    validation_directory, labels='inferred', label_mode='int',
class_names=None,
    color_mode='rgb', batch_size=32, image_size=(128, 128),
shuffle=False, seed=2,
    validation_split=0.2, subset='validation',
interpolation='bilinear', follow_links=False
)

class_names = training_dataset.class_names
print(class_names)

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in training_dataset.take(1):
  for i in range(4):
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.axis("off")

for image_batch, labels_batch in training_dataset:
  print(image_batch.shape)
  print(labels_batch.shape)
  break


AUTOTUNE = tf.data.AUTOTUNE
```

```
training_dataset =
training_dataset.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
validation_dataset =
validation_dataset.cache().prefetch(buffer_size=AUTOTUNE)

normalization_layer = layers.Rescaling(1./255)

normalized_ds = training_dataset.map(lambda x, y:
(normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]

num_classes = len(class_names)
img_height = 128
img_width = 128

model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  #layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])

model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

model.summary()

epochs=10
history = model.fit(
  training_dataset,
  validation_data=validation_dataset,
  epochs=epochs
)

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
```

```
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```