

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Sedevcic, Kevin

**Razpoznavanje hrane na podlagi slik z
nevronskimi mrežami**

MAGISTRSKO DELO
MAGISTRSKI PROGRAM DRUGE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: izr. prof. dr. Matej Kristan

Ljubljana, 2017

UNIVERSITY OF LJUBLJANA
FACULTY OF COMPUTER AND INFORMATION SCIENCE

Sedevcic, Kevin

**Image-Based Food Recognition Using
Neural Networks**

MASTER'S THESIS
THE 2ND CYCLE MASTER'S STUDY PROGRAMME
COMPUTER AND INFORMATION SCIENCE

SUPERVISOR: izr. prof. dr. Matej Kristan

Ljubljana, 2017

COPYRIGHT. The results of this master's thesis are the intellectual property of the author and the Faculty of Computer and Information Science, University of Ljubljana. For the publication or exploitation of the master's thesis results, a written consent of the author, the Faculty of Computer and Information Science, and the supervisor is necessary.

©2017 SEDEVICIC, KEVIN

ACKNOWLEDGMENTS

To my family which believed in me, to my friends who have always supported me, to professor P.Rogelj who has pointed me into the right direction, to professor M.Kristan who gave me a chance, to G.Stampalia who aided in the writing of this thesis and to everyone else who has made this possible.

Sedevcic, Kevin, 2017

"Every journey begins with a single step."

— Lao Tzu

Contents

Povzetek

Abstract

Razširjeni povzetek	i
I Kratek pregled sorodnih del	ii
II Predlagane metode	iv
III Implementacija metod	v
IV Eksperimentalna evaluacija	vii
V Glavni rezultati	ix
VI Sklep	x
1 Introduction	1
1.1 Related Works	3
1.2 Our Contributions	6
1.3 Thesis Outline	7
2 Artificial Neural Networks	9
2.1 Activation Functions	10
2.2 Learning	14
3 Convolutional Neural Networks	35
3.1 CNN Building Blocks	36
3.2 CNN Architectures	41

CONTENTS

4 Recurrent Neural Networks	55
4.1 RNN Architecture	57
4.2 Application to Image Captioning	58
5 Image-Based Food Recognition	69
5.1 Method	69
5.2 Implementation	72
5.3 Visual Model Implementation	74
5.4 Captioning Model Implementation	75
5.5 Region Proposal Model Implementation	76
6 Experimental Evaluation	79
6.1 Food Image Dataset	79
6.2 Data Augmentation	82
6.3 Food Caption Dataset	82
6.4 Performance Measures	83
6.5 Results	84
7 Conclusions	95
7.1 Future Work	97

List of used acronyms

acronym	meaning
ANN	artificial neural network
CNN	convolutional neural network
RNN	recurrent neural network
ILSVRC	imagenet large-scale visual recognition challenge
LSTM	long short term memory
ReLU	rectifier linear unit
MSE	mean squared error
CE	cross entropy
SGD	stochastic gradient descent
RMS	root mean square
SVM	support vector machine
PCA	principal component analysis
BN	batch normalization
FC	fully connected
SRN	simple recurrent network
BPTT	backpropagation through time
BRNN	bidirectional RNN
MIL	multiple instance learning
ME	max entropy
LM	language model
MERT	minimum error rate training
DMSM	deep multimodal similarity model
NIC	neural image caption
GT	ground truth

Povzetek

Naslov: Razpoznavanje hrane na podlagi slik z nevronskimi mrežami

V magistrski nalogi ukvarjamо s klasifikacijo slik in sestavo opisov slik, in sicer smo implementirali tri metode z nevronskimi mrežami (klasifikacija hrane, sestava opisov hrane in sestava opisov hrane z regijami), ki so bile naučene in testirane na dveh podatkovnih zbirkah, prvi, razdeljeni na 21 kategorij hrane s 1470 slikami, in drugi, podatkovni zbirki opisov (2 kategoriji in 5 opisnih stavkov na sliko). Prva implementirana metoda – metoda klasifikacije hrane – uporablja arhitekturo mreže GoogLeNet-Inception-v3 (že naučena na zbirki ILSVRC), ki je bila dodatno naučena na naši podatkovni zbirki hrane, na kateri dosega 82,4 % točnosti top-1 in 98 % točnosti top-5. Druga metoda – metoda sestave opisov – uporablja arhitekturo mreže Show and Tell, ki jo je inicializiral naš model klasifikacije hrane in doseže 23,3 točk perpleksnosti. Metoda ne omogoča sestave popolnih opisov slik, ko je na sliki dva ali več objektov, zato smo implementirali še metodo, ki bi izpisala vsebovanost objektov na slikah. Pri tretji metodi – metodi za sestavo opisov slik z regijami – je uporabljen isti vizualni model kot v prejšnjih dveh metodah, le da klasificira regije vhodne slike. Rezultat evaluacije nad isto podatkovno zbirko je 86,5 % točnosti top-1. Dodatna evaluacija, s katero smo testirali količino razpoznavanih objektov v slikah z več različnimi objekti hrane, je pokazala, da metoda prepozna 64 % objektov na slikah.

Ključne besede

računalniški vid, računalništvo, strojno učenje

Abstract

Title: Image-Based Food Recognition Using Neural Networks

In this thesis we address the problems of image classification and image captioning with three implemented methods with neural networks (food classification, food captioning and food captioning by region-proposal). The methods were trained and tested on a 21-category food image dataset with 1470 images and a 2-category food caption dataset with 750 caption sentences. The first method—food classification method—uses the architecture of the GoogLeNet-Inception-v3 model trained on our food dataset, achieving a top-1 prediction accuracy of 82.4% and top-5 prediction accuracy of 98%. The second method—food captioning method—uses the Show and Tell architecture trained on our food caption dataset, achieving a perplexity score of 23.3. Our food visual model was used to classify the input images, but the overall results did not meet expectations, as the model does not correctly caption images containing multiple foods. The third method—food captioning with region proposal—uses our food classification method to classify images and performs better than the food-classification method alone, achieving a prediction accuracy of 86.5%. Additionally, this third method summarizes the contents of images containing different types of food with an accuracy score of 64%.

Keywords

computer vision, computer science, machine learning

Razširjeni povzetek

Računalništvo se že dalj časa razvija po vzoru biologije. Nekateri vidiki človeka so precej raziskani, medtem ko so človeški možgani še pomanjkljivo odkriti. Študija, ki je popolnoma spremenila način reševanja problemov na področju računalniškega vida, je povezana z delovanjem nevronskih povezav v možganih oziroma s t. i. nevronskimi mrežami. Že leta 1949 je kanadski nevropsiholog Donald O. Hebb, poznan tudi kot oče nevronskih mrež, v svoji knjigi izjavil (Hebb, 1949 [30]), da se povezave med nevroni okrepijo vsakič, ko so uporabljenе. Ideja, poznana tudi kot Hebbovo učenje, je kasneje postala tudi osnova za implementacijo nevronskih mrež v računalniške sisteme. Prva nevronška mreža je bila uspešno implementirana leta 1960 na stanfordski univerzi (Widrow & Hoff, 1960 [68]). Zmogljivost takratnih računalniških sistemov je bila precej omejena, zato implementiranih nevronskih mrež niso obširno uporabljali v praktičnih primerih. Ena izmed uporabljenih rešitev je lahko napovedala le naslednji bit iz sekvence bitov na telefonski liniji, kar je bila rešitev za odmev pri telefonskih klicih. Kljub uspešnosti eksperimentov z nevronskimi mrežami je raziskovalni sektor do približno leta 2010 obvladovala tradicionalna von Neumanova arhitektura. Leto 2012 pa je prineslo cunami idej na osnovi nevronskih mrež, ki so izboljšale rezultate na številnih področjih, npr. detekcije prometnih znakov (Ciresan, 2012 [7]), detekcije hišnih številk (Goodfellow, 2013 [23]), prepoznavanja rokopisa (Greff, 2015 [25]), prepoznavanja objektov in govora (Grave, 2013 [24]), detekcije raka na dojkah (Abdel-Zaher, 2016 [2]) itd. Naprave, kot so pametni telefoni, so dandanes zelo poceni si jih lahko privošči vsak. V zadnjih letih so

se razvile številne računalniške aplikacije, kot so nevronske mreže, ki lahko npr. pripomorejo slabovidnim ljudem razumevanje vsebine slik, uporablajo se lahko kot pripomoček pri oceni vnosa zaužite hrane ali pa pri prepoznavanju določenih tipov objektov, ki so značilni za določen kraj. Zgoraj navedeni primeri temeljijo na razpoznavanju in klasifikaciji slik, ki sta dve temeljni nalogi umetnih nevronskih mrež. V magistrski nalogi se ukvarjam s problemom razpoznavanja hrane na podlagi slik in sestavo opisa hrane na podlagi slike. Klasifikacijo bomo izvedli s konvolucijskimi nevronskimi mrežami (CNN), sestavo opisov pa s ponavljajočimi se nevronskimi mrežami (RNN). Dodatno bo implementirana še metoda, ki iz vsake slike obreže manjše dele in jih naknadno klasificira. Dobro klasificirane regije bodo uporabljeni za sestavo opisa vsebine slik. Pri vseh treh metodah bosta uporabljeni še dve novi podatkovni zbirki hrane, ena pri klasifikaciji, druga pri sestavi opisov.

I Kratek pregled sorodnih del

Pri klasifikaciji slik je nevronske mreže uspešno uporabilo že več raziskovalcev (Krizhevsky et al., 2012 [42], Zeiler & Fergus, 2013 [75], Simonyan & Zisserman, 2014 [58], Szegedy et al., 2014 [63], He et al., 2015 [29] and Huang & Liu, 2016 [34]). AlexNet (Krizhevsky et al., 2012 [42]) je prva konvolucijska nevronska mreža, ki se je povzpela na vrh primerjalnega testa ILSVRC in presegla tradicionalne metode računalniškega vida. Leta kasneje so se na primerjalnem testu ILSVRC povzpele novejše in bolj prefinjene mreže. Zeiler & Fergus, 2013 [75], Simonyan & Zisserman, 2014 [58] and Szegedy et al., 2014 [63] so odkrili, da so pri razvoju konvolucijskih nevronskih mrež manjša konvolucijska jedra učinkovitejša od tistih v AlexNetu (velikost se je pri novejših spremenila s 7×7 na 3×3 in 1×1). He et al., 2015 [29] so dokazali, da lahko še boljše rezultate dosežemo s povezavami med različnimi plastmi. Huang & Liu, 2016 [34] pa sta pokazala, da so lahko konvolucijske nevronske mreže sestavljene le iz konvolucijskih plasti. Najboljši metodi, ki sta bili testirani na podatkovni zbirki ILSVRC, sta mreža ResNet (He et al., 2015 [29])

in nova verzija GoogLeNeta (Szegedy et al., 2015 [64]). Kompleksnejše arhitekture sicer pripomorejo k vedno boljšim rezultatom, a s seboj prinašajo tudi težave. Metode DropOut (Hinton et al., 2012 [32]), DropConnect (Wan et al., 2013 [67]) in Batch Normalization (Ioffe & Szegedy, 2015 [35]) rešujejo prekomerno učenje mreže na določenih primerih (angl. overfitting) in tako izboljujejo delovanje na testnih primerih. Druga metoda, ki rešuje prekomerno učenje mreže, je umetno povečevanje vzorcev (angl. data augmentation). Z naključnim zrcaljenjem, obrezovanjem, spremembom velikosti, barve in osvetljenosti slik se težava prekomernega učenja zmanjša. Izmed zgoraj navedenih prispevkov o arhitekturi konvolucijskih mrež je najboljši pristop omenjen v Szegedy et al., 2014 [63], kjer se vsaka slika obreže naključno (velikost obreza v intervalu 8% - 100% in sprememba v razmerju slike v intervalu 76% - 133%), pri tem pa se spremenijo osvetljenost, kontrast in barva (v intervalu 50% - 150%). Raziskovalci (Vinyals et al., 2014 [66], Mao et al., 2014 [47], Fang et al., 2014 [18] and Karpathy et al., 2014 [37]) so v zadnjih letih težavo opisovanja slik uspešno premostili. Za pridobitev glavnih lastnosti iz vhodnih slik vse metode uporabljajo konvolucijske nevronske mreže. Mao et al., 2014 [47] uporablja dodatno plast, s katero za sestavo opisov povezuje slovarske podatke in ponavljajoče se nevronske mreže. Fang et al., 2014 [18] klasificira regije slik z besedami (glagoli, pridevniki in samostalniki), nato z modelom podobnosti (angl. multimodal similarity model) regijam dodaja besede. Besede so dodane, tako da se nova beseda najbolje ujema z ostalimi besedami, že prisotnimi v regiji. Na koncu faze sestavljanja se stavki razvrstijo, pri čemer so izbrani le najboljši. Karpathy et al., 2014 [37] združijo rezultate iz vizualnega dela z dvosmerno ponavljajočo se nevronsko mrežo. Model je naučen le na regijah originalne slike in na delih povedi, ki opisujejo izbrano regijo. Vinyals et al., 2014 [66] so edini, ki so za sestavo stavkov uporabili novo strukturo (angl. long short term memory) ponavljajočih se nevronskih mrež, ki rešuje časovne odvisnosti. Njihov pristop je na primerjalnem testu MS COCO dosegel najvišje točke. V magistrski nalogi se osredotočamo le na slike hrane in ne rešujemo splošnih težav v

zvezi s klasifikacijo slik ali sestavo opisov slik. Pregledane so bile štiri večje podatkovne zbirke hrane, in sicer UEC-Food-100 (Matsuda et al., 2012 [48]), UEC-Food-256 (Kawano & Yanai et al., 2014 [38]), FOOD-101 (Bossard et al., 2014 [3]) ter UniMib2016 (Ciocca et al., 2017 [6]), od katerih pa nobena ni bila izbrana kot podlaga za naše delo. UEC in UniMib2016 so podatkovne zbirke, ki vsebujejo več različnih objektov hrane na vsaki sliki, medtem ko FOOD-101 vključuje veliko japonskih in korejskih jedi, ki jih težko razlikujemo. Tudi na področju razpoznavanja hrane je bilo izvedenih več raziskav (Kagaya et al., 2014 [36], Kawano & Yanai, 2015 [73], Lu, 2016 [26] and Ciocca et al., 2017 [6]). Kagaya et al., 2014 [36] je testiral konvolucijsko nevronske mreže za razpoznavanje slik, če vsebujejo hrano ali ne. Kawano & Yanai, 2015 [73] sta izjavila, da dodatno učenje mreže, ki je že bila naučena na podobnih kategorijah, povečuje natančnost razpoznavanja. Lu, 2016 [26] je primerjal konvolucijsko nevronske mreže s tradicionalno metodo (angl. bag of features) in dokazal, da je nevronska mreža nedvomno boljša. Ciocca et al., 2017 [6]) so uporabili konvolucijsko nevronske mreže za razpoznavanje svoje podatkovne zbirke hrane UniMib2016 s točnostjo 79%.

II Predlagane metode

V naslednjem poglavju razpravljamo o treh metodah: (a) konvolucijski nevronske mreže, uporabljeni pri klasifikaciji slik hrane, (b) ponavljajoči se nevronske mreže za sestavo opisov slik ter (c) metodi, ki opisuje vsebino slik z obrezovanjem regij in njihovo naknadno klasifikacijo s konvolucijsko nevronske mreže iz točke (a). Prvi dve metodi pri učenju uporabljata dve na novo generirani podatkovni zbirki hrane.

II.I Metoda vizualnega modela

Uporabili smo arhitekturo konvolucijske mreže GoogLeNet-Inception-v3 (Szegedy et al., 2015 [64]), ki jo odlikuje zelo velika natančnost pri primerjalnem testu ILSVRC. Inception-v3 je različica originalne mreže GoogLeNet, ki je na

primerjalnem testu ILSVRC dosegla 3.58% top-5 napako. Pozitivna lastnost konvolucijskih mrež je možnost prenosa naučenih značilnosti (angl. transfer learning) med enakimi mrežami. Tudi v pričujočem delu smo uporabili že naučeno mrežo na podatkovni zbirki ILSVRC in na svoji podatkovni zbirki hrane ponovno naučili le zadnji dve plasti.

II.II Metoda modela za sestavo opisov

Zaradi najboljših rezultatov na primerjalnem testu MS COCO in nove strukture (long short term memory) pri ponavljajoči se nevronski mreži je bila izbrana Googlova arhitektura Show and Tell (Vinyals, 2014 [66]). Za vizualni del smo uporabili na naši podatkovni zbirki hrane naučeno konvolucijsko mrežo, ki inicializira arhitekturo Show and Tell. Inicializirana arhitektura je naknadno naučena na zbirki hrane z opisi.

II.III Metoda modela za sestavo opisov z regijami

Tretja metoda uporablja na podatkovni zbirki hrane naučeno konvolucijsko mrežo, ki dobi regije slik in jih klasificira. Metoda generira 100 naključnih regij vhodne slike velikosti v intervalu 45% – 100%. Če je regija razpoznana s točnostjo nad 70%, je sprejeta, sicer se zavrže. Sprejete kategorije hrane nato tvorijo obnovo vsebine vhodne slike.

III Implementacija metod

Naš razpoznavalec in opisnik hrane je implementiran v Pythonu z ogrodjem TensorFlow. Zaradi svoje nekonvencionalne strukture TensorFlow uporablja le specifičen tip datotek, imenovan TFRecords.

III.I Implementacija vizualnega modela

Ker TensorFlow upošteva le določeno vrsto datotek, se podatkovna zbirka s pomožno funkcijo (*convert_data.py*) pretvori v dve skupini s končnico TFR-

cord. Prva skupina je namenjena učenju mreže, druga pa testiranju točnosti. Izbira slik je določena z naključnim mešanjem slik po kategorijah. Za tem je treba nastaviti hiperparametre za učenje: zastave za upravljanje učenja, optimizacijske zastave, zastave stopnje učenja, zastave podatkovne zbirke in zastave za naknadno učenje, ko se uporablja prenos naučenih značilnosti. Po koncu nastavite naložimo mrežo ter podatkovno zbirko s slikami in ustreznimi oznakami. Pri učenju mreže uporabimo metodo povečevanja vzorcev. Učenje lahko prekinemo predčasno ali po določenem številu korakov. Po končanem učenju se ustvari kontrolna točka in vizualni model se lahko oceni na testnih primerih.

III.II Implementacija modela za sestavo opisov

Pri implementaciji modela sestave opisov smo morali najprej sestaviti podatkovno zbirko stavkov, ki opisujejo specifične slike (uporabili smo pomožni vmesnik *Caption_GUI.py*). Na enak način kot pri vizualnem modelu smo podatke pretvorili v končnico TFRecord in sestavili še eno datoteko, ki vsebuje število ponavljanj vseh besed v opisih slik (*word_counts.txt*). Zgoraj naštete datoteke so omogočile začetek učenja. Enako kot pri vizualnem modelu je bilo treba določiti nekatere parametre, in sicer pot, na kateri se nahajajo datoteke TFRecord, pot do že naučenega vizualnega modela in imenik slik za učenje. Po končanem učenju smo lahko pognali vrednotenje učenja (*evaluate.py*), ki je pokazalo, kolikšno perpleksnost ima jezikovni model. Model se lahko še dodatno testira (*run.inference.py*), tako da sestavi opis za vhodno sliko.

III.III Implementacija modela za sestavo opisov z regijami

Pri implementaciji modela z regijami je najprej treba slike pretvoriti v združljiv format za TensorFlow. Nato so informacije podatkovne zbirke in vizualni model naloženi. Funkcija, ki generira regije, posreduje vizualnemu modelu 100 naključno izbranih regij (s funkcijo *tf.image.sample_distorted_bounding_box*).

Velikost vsake regije je nato spremenjena glede na zahteve vizualnega modela (velikost vhodne slike je $299 \times 299 \times 3$) in je posredovana čez mrežo za klasifikacijo. Napovedana točnost in kategorija vsake slike se obdržita, če je točnost višja ali enaka 70%. Po končani klasifikaciji regij se izpišejo točnosti in kategorije.

IV Eksperimentalna evaluacija

V naslednjem poglavju so opisani podatkovne zbirke hrane, način umetnega vzorčenja slik, uporabljeni meritve kakovosti modelov in rezultati meritev.

IV.I Podatkovna zbirka hrane

Slike, ki spadajo v končno verzijo sestavljene podatkovne zbirke hrane, smo prenesli iz prosto dostopne podatkovne zbirke OpenImages (Krasin [41]). OpenImages je zbirka, sestavljena iz več kot 10 milijonov slik in več kot 6000 kategorij. Slike so dostopne prek url naslovov, zato so bile s spleta prenešene s pomožnim Pythonovim skriptom (*downloader_from_csv.py*) in umeščene v kategorije. Ker so bile nekatere slike prisotne v več kot eni kategoriji, so morali biti dvojni ročno izbrisani ali ročno umeščeni v le eno od več kategorij (*dups_finder.py*). Dobljena podatkovna zbirka je štela 8000 slik in 45 kategorij. Testiranje te podatkovne zbirke pa ni doseglo visokih rezultatov. Razlog so bile nekatere slike, ki niso ponazarjale objektov označenih kategorij (npr. slika tržnice, označena kot kategorija jabolko) in neuravnoveženo število slik za vsako kategorijo (nekatere kategorije so vsebovale 800 slik, druge pa manj kot 50). Zadnja popravljena verzija podatkovne zbirke hrane je vsebovala 21 kategorij, 70 slik za vsako kategorijo in 1470 slik skupno.

IV.II Umetno povečevanje vzorcev

Zaradi izbora kompleksnejše arhitekture je bila podatkovna zbirka še vedno premajhna, zato smo izbrali umetno vzorčenje, s katerim smo povečali število

slik. Vzorčenje je sledilo vzorčenju Szegedyjeve raziskave (Szegedy et al., 2014 [63]), kjer se slike obrežajo v intervalu 8% - 100% originalne slike, pri čemer se razmerje med stranicami spremeni v intervalu 76% - 133%, slikam pa se doda fotometrično popačenje, pri čemer se osvetljenost, barva in kontrast naključno spreminjajo v intervalu 50% - 150%. Poleg pojasnjene vzorčenja smo vsem slikam spremenili velikost, tako da ima najdaljša stranica 300 pikslov dolžine. S spremembou velikosti smo zmanjšali velikost podatkovne zbirke z 2 GB na 85 MB.

IV.III Podatkovna zbirka opisov slik

Za učenje modela sestave opisov iz slik smo izbrali dve kategoriji iz podatkovne zbirke hrane (jabolko in kruh) in za vsako sliko napisali 5 opisnih stavkov (s pomočjo *Caption_GUI.py* smo shranili stavke v formatu MS COCO) ter pri tem dobili 750 sestavljenih stavkov. Da bi preverili, ali se model lahko nauči relacij med več objekti, ko na primerih za učenje ni takih relacij, smo izbrali samo dve kategoriji. V ta namen smo naučeni model testirali na slikah, ki so vsebovale oba objekta podatkovne zbirke (jabolko in kruh). Sestava opisov je delovala samo na enem od dveh objektov, ki je v tistem trenutku v vizualnem modelu prevladal. Sestava večje podatkovne zbirke, ki bi vsebovala relacije med objekti (tj. da slika vsebuje več kot en objekt hrane in opis razloži relacijo med njimi), bi bila preveč zamudna, zato se razvijanje podatkovne zbirke ni nadaljevalo.

IV.IV Meritve

Pri vizualnem modelu je natančnost učenja izmerjena s točnostjo top-1 in top-5. Med testiranjem modela vizualni model klasificira vsako sliko z določeno kategorijo. Če je izbrana kategorija enaka resnični kategoriji slike (angl. ground truth), je klasifikacija uspela. Količina uspelih klasificiranih slik se na koncu testiranja deli s številom slik v testni zbirki; rezultat je točnost top-1. Pri točnosti top-5 pa se upošteva rezultat klasificiranja mreže kot točen, če je

ena izmed petih najvišjih klasificiranih kategorij enaka resnični kategoriji. Pri modelu sestave opisov slik se upošteva meritev perpleksnosti. Perpleksnost določa povprečno število možnih besed, ki model izbira pri vsaki iteraciji (če je perpleksnost enaka pet, vsakič ko model sestavi novo besedo, izbira med petimi drugimi). Pri sestavi opisov slik z regijami se uporablja enaka meritev kot pri vizualnem modelu, saj se uporablja tudi enak model. Ker je dodatna lastnost modela sestava opisov vsebovanih kategorij hrane v slikah, se mera točnosti uporablja še za število napovedanih kategorij (če slika vsebuje štiri kategorije, model pa jih razpozna le tri, je meja točnosti 75%).

V Glavni rezultati

Naslednje poglavje predstavlja glavne rezultate meritev treh na novo implementiranih metod. Učenje vizualnega modela hrane je potekalo z gradientno metodo (angl. SGD), velikost serije slik s 64 slikami, optimizacijsko funkcijo ADAM, eksponentnim manjšanjem stopnje učenja (z eksponentom manjšanja 0.96) in faktorjem 0,00004 za regularizacijo manjšanja uteži. Končni rezultati so dosegli točnost top-1 82,4% in točnost top-5 98% na testni zbirki slik. Če smo testno zbirko spremenili, tako da so bile slike rotirane pod kotom 30 stopinj, se je točnost top-1 znižala na 65%. Zaradi tega in dodatne možnosti uporabe modela v praktičnih primerih smo umetnemu vzorčenju dodali še naključne rotacije slik (v intervalu ± 30), kar je izboljšalo robustnost vizualnega modela na novih testnih primerih skoraj do nivoja originalnih rezultatov (točnosti top-1 79% in točnosti top-5 96%). Model sestave opisov slik je bil naučen na novi zbirki slik dveh kategorij s sestavljenimi opisi. Perpleksnost modela na testni zbirki je bila precej nizka, 23,3, v primerjavi z izhodiščnim modelom Show and Tell (Vinyals, 2014 [66]), kjer je bil rezultat testiranja na naši zbirki nad 900 perpleksnosti. Kljub visoki perpleksnosti je model Show and Tell sestavil zelo natančne opise slik. Razlika je v učni množici, saj je število učnih opisov in slik veliko manjše od učne množice MS COCO, na kateri je bil naučen model Show and Tell. Tretji model je pravzaprav nad-

gradnja prvega z dodano metodo obrezovanja regij slik. Model ni bil dodatno naučen, saj uporablja že naučeni vizualni model iz prve metode. Rezultati na podatkovni zbirkki hrane so rahlo višji s točnostjo top-1 86,5 %. Model smo dodatno testirali na slikah z več kot enim objektom hrane. Odstotek razpoznavanih kategorij slik je 64 %.

VI Sklep

V magistrski nalogi smo predstavili glavne koncepte umetnih nevronskeih mrež, ki so bili podlaga za implementacijo treh metod. Prva metoda je model za klasifikacijo hrane z inicializirano arhitekturo in parametri modela GoogLeNet-Inception-v3. Model je bil naknadno naučen na novi zbirkki hrane z 21 kategorijami in 1470 slikami in je dosegel 82,4 % točnosti top-1 in 98 % točnosti top-5 (model GoogLeNet je na primerjalnem testu ILSVRC dosegel 96,42 % točnosti top-5). Druga metoda je model za sestavo opisov slik, ki uporablja arhitekturo modela Show and Tell; arhitektura je bila inicializirana z našim na novo naučenim vizualnim modelom in naknadno še z zbirkko opisov hrane dveh kategorij. Naučeni model je dosegel mejo perpleksnosti 23,3, medtem ko je izhodiščni model dosegel več kot 900 točk; kljub visoki perpleksnosti sta oba modela natančno sestavlja opise slik. Tretja metoda je nadgradnja prve z metodo obrezovanja regij vhodne slike. Metoda je na testni zbirkki hrane dosegla top-1 točnost 86,5 %, pri razpoznavanju količine objektov iz slik pa točnost 64 %.

VI.I Nadaljnje delo

Umetno vzorčenje slik je ena izmed najučinkovitejših metod pri učenju modelov. Žal nam ni znano, katere metode dosežejo pri določenih slikah boljše rezultate. Izvesti bi bilo treba natančnejšo raziskavo, v okviru katere bi bilo specifičnim slikam dodeljeno specifično vzorčenje (sadnje in zelenjavo se fotografira pod različnimi zornimi koti, slike avtomobilob in hiš pa večinoma od spodaj navzgor.). Prihodnja ideja pri umetnem vzorčenju je torej dodeljeva-

nje specifičnim kategorijam specifično vzorčenje. Uporabljene konvolucijske mreže so mreže, ki so bile prvič implementirane leta 2014 in pri tej magistrski nalogi niso bile dodatno spremenjene. Zaradi hitrega razvoja novih arhitektur in idej v konvolucijskih nevronskeih mrežah je ta arhitektura že zastarella. V prihodnosti bi se zato lahko posvetili posodabljanju arhitekture. Kar zadeva zbirke slik z opisi za model sestave opisov, se lahko razširijo s slikami, ki vsebujejo kombinacije več objektov hrane. Tako kot nove slike, bi morali tudi novi opisi vnašati relacije med več kategorijami hrane. Nov test na tako razširjeni zbirki bo pokazal, ali so hipoteze o učenju relacij med objekti na slikah točne ali ne.

Chapter 1

Introduction

Increasingly, scientists of all kinds have been gaining an understanding of many aspects of the human being. Yet, the human brain continues to be something of a mystery. Progress has been significant in this area, originating from sciences that have tackled the issue from apparently opposite sides: for example, psychology and neuroscience. Significant discoveries have been made in our knowledge of the division of the organ's structural and functional areas, or of how signals are transmitted through the nervous system. In the latter case, for example, it is now known that signals travel via billions of neurons clustered together through specialized connections called synapses. Impulses starting as sensory signals from organs such as the eyes, ears, tongue, skin, etc. flow into the system through electrochemical impulses that solicit certain neuron responses. These transformed impulses can then travel to the destination in the right area of the brain for processing. These neuron formations are called neural networks. The question that computer scientists asked themselves was: what if the idea of these interconnections within the brain could be "translated" into computer science?

Artificial neural networks constitute one answer to such question. And, slowly but surely, the idea of this particular analogy has worked its way into the practical science of computers. In 1949, Canadian Neuropsychologist Donald O. Hebb, considered in many ways the father of neural networks,

stated in his book (Hebb, 1949 [30]) that neural pathways are strengthened each time they are used; this idea, later known as Hebbian learning (Hebb, 1949 [30]), would become the basis for the development of artificial neural networks (ANNs). Several unsuccessful experiments occurred in the 1950s, such as Nathaniel Rochester’s at the IBM research laboratories (Rochester et al., 1955 [49]). In 1960, the first working artificial neural networks were developed at Stanford University by Bernard Widrow and Marcian Hoff (Widrow & Hoff, 1960 [68]). These model networks were named “ADALINE” and “MADALINE.” At that time, the computational power of computers was weak, and the newly-created networks could do little more than read streaming bits from a phone line, and predict the next bit. While this may seem like a meager result, it represented a significant advancement from the ordinary technology of the time, for it could be used to solve the echoing in telephone lines, a bothersome flaw in that period.

Despite the early success of these neural network experiments, traditional von Neuman architecture—based on the model developed in 1945 by Hungarian-American mathematician and physicist John von Neumann and others, a model also known as Princeton architecture—took over the computing scene, and neural-network research was temporarily halted. This conventional computing worked better for defined problems that could be solved by a set of if/then rules (or with the top-down approach); furthermore, at that time a lot of software still needed to be developed, and von Neumann architecture helped to accelerate the process.

As times changed, there were enormous developments in computer hardware. In the new Millennium, computers acquired a greater computational power, and with it the possibility of implementing bigger and more sophisticated networks. Accordingly, there was a boom in funding for ANNs. The 2010s brought about the so-called “tsunami” of ideas in this particular field: many methods were discovered for addressing tasks such as traffic-sign detection (Ciresan, 2012 [7]), house-number detection (Goodfellow, 2013 [23]), handwriting recognition (Greff, 2015 [25]), object and speech recognition

(Grave, 2013 [24]), breast-cancer detection (Abdel-Zaher, 2016 [2]), and many more, thus defeating by a huge margin the conventional computer vision and speech processing algorithms.

These days, image-captioning devices such as smartphones have become inexpensive and within everyone’s reach, and many applications are being developed for those tiny computers. Connecting smartphones to ANNs can, for example, help visually-impaired people understand what they are looking at. Image-classification ANNs can help with dietary assessments (we all have busy schedules and the first thing that usually goes out of our daily plan is a healthy diet): there is nothing easier than taking a picture of what you are eating to automatically keep track of nutrition intake. Furthermore, if we take a trip to a place we have never been before, a recognition tool can help us by giving us information about the things we are seeing, such as animals, foods, plants, etc. Obviously, such a tool can also be used for educational purposes, for it allows a child to have the information about surrounding just a click (or a picture) away. All the above applications are based on image recognition and image classification, tasks on which the modern ANNs have been shown to perform exceptionally well. This thesis addresses the problems of image classification and image captioning that are solved with convolutional neural networks (CNNs) and recurrent neural networks (RNNs) respectively. A region-proposal method is also added to the CNN to create summaries of image content. In this work three methods are implemented using state-of-the-art architectures (GoogLeNet and Show and Tell for the visual model and the captioning model respectively) trained on two newly created food datasets in order to extend the research in food classification and captioning.

1.1 Related Works

Several researchers—Krizhevsky et al., 2012 [42], Zeiler & Fergus, 2014 [75], Simonyan & Zisserman, 2014 [58], Szegedy et al., 2015 [63], He et al., 2016 [29]

and Huang & Liu, 2016 [34]—have already used neural networks on image classification challenges, achieving great results.

AlexNet, the network of Krizhevsky et al., 2012 [42], is the first deep convolutional neural network to test the classification challenge on the ILSVRC-2012 benchmark [56], outmatching the conventional algorithms by a big margin. After this, the other above-mentioned papers achieved even better results, changing the preprocessing methods and the architecture of their neural networks. In Zeiler & Fergus, 2014 [75], Simonyan & Zisserman, 2014 [58] and Szegedy et al., 2015 [63], it was discovered that the best method to develop a CNN is to use smaller convolutional kernels. In those papers the sizes of the kernels changed from 7×7 -sized kernels used in AlexNet to kernels of size 3×3 and 1×1 . He et al., 2016 [29], with their shortcuts between layers, proved that deeper networks can be developed, with even better results. The last paper of Huang & Liu, 2016 [34] discussed an idea of concatenated blocks that consist of convolutional layers only.

Every network was developed to be the next state-of-the-art in image classification tasks. This brought about better results, but issues too. Dropout methods were first used by Krizhevsky et al., 2012 [42] and proved to be extremely effective in solving overfitting. In He et al., 2016 [29], a new regularization method called batch normalization was used, which proved to be even more effective against the overfitting problem.

Data augmentation methods have proven to be an aid to networks in gaining higher scores and diminishing overfitting. Several approaches that consisted in random flipping, cropping, re-scaling and colorbrightness perturbations were used in all the mentioned papers. From the results stated in Szegedy et al., 2015 [63], the most useful approach consisted in random-cropping the images (where the scales and the aspect ratios are in a range of 8% – 100% and 76% – 133%) and adding a photometric distortion (where the brightness, contrast and color are manipulated in a range of 50% – 150%).

The architectures that scored highest in the ILSVRC benchmark for image classification were an adjusted version of GoogLeNet defined in Szegedy

et al., 2015 [64] and He et al., 2016 [29], which achieved almost the same results.

Several researchers—Vinyals et al., 2015 [66], Mao et al., 2014 [47], Fang et al., 2015 [18] and Karpathy et al., 2015 [37]—have recently addressed the task of image captioning successfully through deep models. All of them use a CNN part for taking features out of the input images. Mao et al., 2014 [47] uses a multimodal layer that joins together a word-embedding layer (a one-hot-word vector that contains the words with the highest semantical meaning) and an RNN layer for generating the output sentence. Fang et al., 2015 [18] propose an approach that collects a set of words that are more likely to describe image regions (i.e. nouns, verbs and adjectives). With a deep multimodal similarity model each image with its text fragments is mapped into a vector. The sentences are generated by adding more words to those text fragments. The words are selected from a set of words that are the most likely to follow the previous word in the sentence. At the end the sentences are re-ranked and the best one is chosen. Karpathy et al., 2015 [37] propose a pipeline approach that joins the CNN results with the results of a bidirectional recurrent neural network for generating image descriptions. Differently from others, they train the network only on portions of images and parts of the training sentences. From the results on the MS COCO dataset this idea outperformed the previous two. Only Vinyals et al., 2015 [66] use long-short term memory (LSTM) cells in their RNN part, which is a newer structure for RNNs that solves time dependencies. It is the current state-of-the-art solution on the image captioning MS COCO benchmark [45].

This work does not aim to solve general image classification and captioning problems, but it addresses specific food-recognition issues. Due to the lack of food datasets Matsuda et al., 2012 [48] assembled the UEC-food-100 dataset. Two years later an extended version came out, the UEC-food-256 extended by Kawano & Yanai et al., 2014 [38]. The largest dataset ever built is FOOD-101 Bossard et al., 2014 [3] with 101 categories and 1000 images per category. Ciocca et al., 2017 [6] built UniMib2016 dataset of segmented

images of a canteen tray with 73 classes and 1000 images in total. The UEC and UniMib2016 datasets have different plates and types of foods on each image, which is not suitable for use in the experiments described in this thesis. FOOD-101 is mostly composed of Japanese and Korean foods that are similar and difficult to differentiate. Every image in these datasets represent cooked foods only.

Researchers Kagaya et al., 2014 [36], Kawano & Yanai, 2015 [73], Lu, 2016 [26] and Ciocca et al., 2017 [6] applied a CNN to food images classification. Kagaya et al., 2014 [36] used a CNN for recognizing images between two categories (food and non-food categories). The network was trained on 10 food categories and thus the network recognized only the images belonging to those categories as opposed to every other category. Kawano & Yanai [73] found out that fine tuning a model that was trained with an additional set of classes than those in the food dataset boosted the results, gaining a 78.77% top-1 error for UEC-food-100 dataset and a 67.57% top-1 error for the UEC-food-256 dataset. Lu, 2016 [26] compared a basic 5-layer CNN against a conventional bag of features and an SVM, stating that the CNN performed drastically better (56% against 90% of the CNN). Ciocca et al., 2017 [6], used a CNN that picked the segmented foods of the canteen trays of the UniMib2016 dataset and classified them with an accuracy of 79%.

1.2 Our Contributions

Our contribution is five-fold. The first contribution is a deep convolutional neural network trained on our food dataset that achieves competitive performance in food image recognition compared to the results on the architectures from the ILSVRC benchmark (Krizhevsky et al., 2012 [42], Zeiler & Fergus, 2014 [75], Simonyan & Zisserman, 2014 [58], Szegedy et al., 2015 [63], He et al., 2016 [29] and Huang & Liu, 2016 [34]). The architecture used is the GoogLeNet (with inceptionv3 module) with a modified preprocessing approach.

The second contribution is a recurrent neural network for automatic image captioning (Show and Tell model) trained on a food-specific image-caption pair dataset, which is a novelty in the image captioning field (the python GUI for captions in the MS COCO format is freely available).

The third contribution is a concise overview of the evolution of the most important CNN and RNN architectures from 2012 to 2016 in image recognition and image captioning tasks.

The fourth contribution is a region-proposal CNN for classifying regions of images and creating summaries of food present in each image.

The fifth contribution is a new 21-class food dataset that does not consist of cooked foods only or images with multiple food objects (it also contains raw foods such as fruits and vegetables). The images are taken from the OpenImages dataset (Krasin et al., 2017 [41]), from Google and Flickr images with creative commons attributions for a total of 1470 images spread through 21 categories. Furthermore a guide that defines how to implement the work of this thesis step by step is available on GitHub¹. Everything will be publicly available for further development.

1.3 Thesis Outline

The remainder of the thesis is composed of four chapters. Chapter 2 is an introduction to ANNs where the activation and cost functions, the different weight initialization and update methods and the different approaches for regularization and overfitting problem solving are defined. Chapter 3 is about CNNs with a brief overview of the building blocks: convolution, non-activation and pooling layers are discussed there. The second part of the chapter is an overview of the most influential CNN architectures from 2012 to 2016. Chapter 4 presents RNNs, their basic concepts and an evolution of the RNN architecture with LSTM cells. The second part of the chapter discusses the four image captioning implementations that achieved

¹<https://github.com/KaneFury/Tensorflow-image-classification-and-captioning>

the highest score against the MS COCO captioning benchmark. Our food recognition network is presented in Chapter 5. The experimental evaluation and its results are explained in Chapter 6 and the conclusions are drawn in Chapter 7.

Chapter 2

Artificial Neural Networks

Figure 2.1 shows the smallest computational block in an ANN, the neuron¹. It is connected to multiple other neurons with weighted connection (the higher the weight, the stronger the connection and the more information is passed through). Weights have an important role here. They are learned by the network through a training process where they change in accordance with the importance of each input for each expected output. Adjusting the weights, i.e. the tracing back and correcting the information from output to input to adjust the learning is known as backpropagation (see chapter learning). Neu-

¹“neuron” was the early name given by scientist to the smallest unit in the ANN; with the evolution in the naming convention it could be called as a unit.

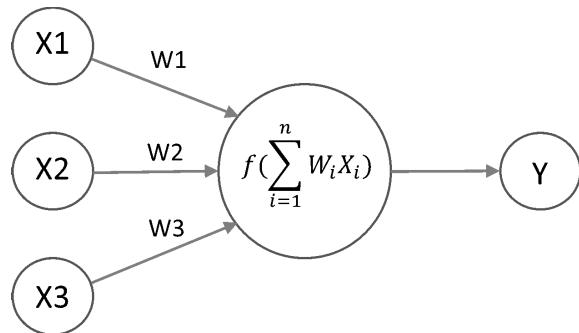


Figure 2.1: Simple neural network.

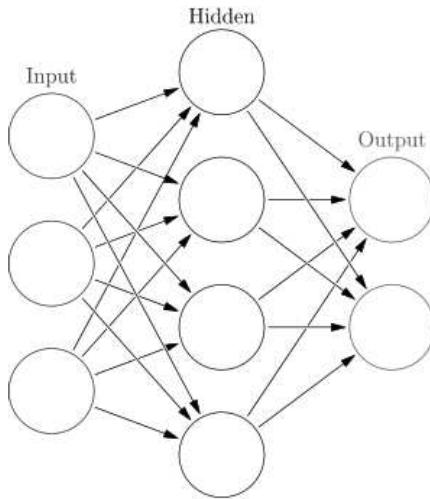


Figure 2.2: Simple neural network.

ron activations in the brain are the consequence of electrochemical impulses; in an ANN this activation is recreated by non-linear activation functions that are explained in the next sections.

A network is created by connecting together lots of neurons. Such network is split into three parts (see Figure 2.2). The input/output layer, where the neurons are connected with external sources and one or more hidden layers is where the actual prediction of the expected output happens. The networks prediction of an output based on a feeding of information to is termed forward propagation. These networks are the basic concept of the idea called deep learning.

2.1 Activation Functions

As mentioned above, every neuron in a layer applies a function over the weighted sum of the inputs of the previous layers's neurons. A higher output means that a neuron is activated (or important) and carries good information. In this case, the output can be anything ranging from $-\infty$ to ∞ . How does

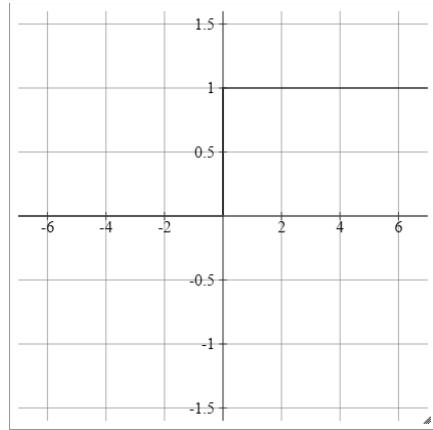


Figure 2.3: Step activation function.

one decide if a neuron is good enough to be activated? In the past the first function used were functions with a threshold (see Figure 2.3). Let's suppose that every value above a given threshold is activated and every value below is not; i.e. values of 1 are applied to activation and 0 to non activation, a really simple solution that has its drawbacks. A network that needs to chose between two outputs, "Yes" or "No," can be managed by such a function. As the problems assigned to ANNs become more complex and as the tasks became more elaborate, this type of solution soon became obsolete, precisely because of its simplicity. For the new ANNs a more complex approach with more intermediate steps was needed: an analog function. The following Sections 2.1.1, 2.1.2, 2.1.3, 2.1.4 will explore the most used activation functions in the field of ANNs with their properties.

2.1.1 Linear Function

Another activation function that was used in the past is the linear function, i.e.,

$$f(x) = cx. \quad (2.1)$$

The output in this case is a function of weighted input, where the weight defines the function's steepness. This equation (2.1) has an analog range of values from which the highest one can be chosen. Such function is easily implemented, but has its limitations. As mentioned in the introduction, an ANN learns by backpropagating the error through the weights with gradient descent methods. But as seen in the Function (2.1) the gradient of x in this case is constant and does not depend on changes of the input δx . Another drawback of linear functions is that if one is used over and over again on the same data as in a multi-layered network the output will still be linear. That means that everything could be summarized with just one layer, and in doing so, the possibility of stacking layers of network, which gives better performances in ANN fields (Krizhevsky et al., 2012 [42], Goodfellow, 2013 [23], Simonyan et al., 2014 [58], Szegedy et al., 2015 [63], He et al., 2016 [29]) is lost. Using non-linear functions is the solution.

2.1.2 Sigmoid Function

The first non-linear type is the sigmoid function, i.e.,

$$f(x) = \frac{1}{1 + e^{-x}}. \quad (2.2)$$

As shown in Figure 2.4, the value of the function between $x = -2$ and $x = 2$ is very steep. This means that small variations of x yield big differences in the output y . Here the gradient of x will be very big while at the extremes it will be really small, so small that it is almost imperceptible; this is known as the problem of the vanishing gradient. Every value that approaches an end is likely to stay there. There are positive properties too: the sigmoid function solves the problem of linearity and has a range of values between 0 and 1, hence multiple layers can be stacked. This means that ∞ or $-\infty$ values do not need to be handled. This activation function is no longer favored and is being replaced by the hyperbolic tangent and the rectified linear unit.

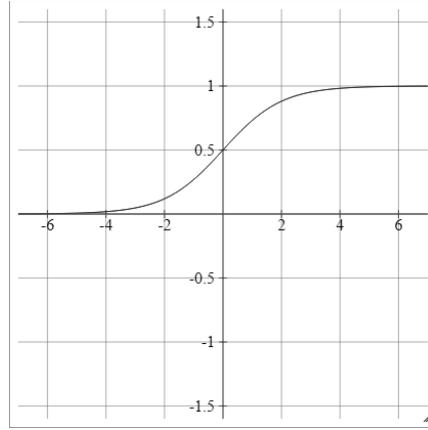


Figure 2.4: Sigmoid activation function.

2.1.3 Hyperbolic Tangent: Tanh

Similar to its predecessor, the hyperbolic tangent, i.e.,

$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1, \quad (2.3)$$

has the same properties as the sigmoid with a slight difference in the slope steepness and the range which goes from -1 to 1 (see Figure 2.5). The vanishing gradient problem still persists and limits the quality of of ANN results.

2.1.4 Rectified Linear Units: ReLU

As seen in Figure 2.6 the function is composed of two linear pieces. Everything that is lower than or equal to zero would be zero and everything above zero remains the same. Nonetheless the ReLU function, i.e.,

$$f(x) = \max(x, 0), \quad (2.4)$$

is not linear. Another point that should be discussed here is the sparsity of the activation. A lot of neurons being zero means that they will not activate. Imagine now a big ANN with lot of neurons. A function like sigmoid

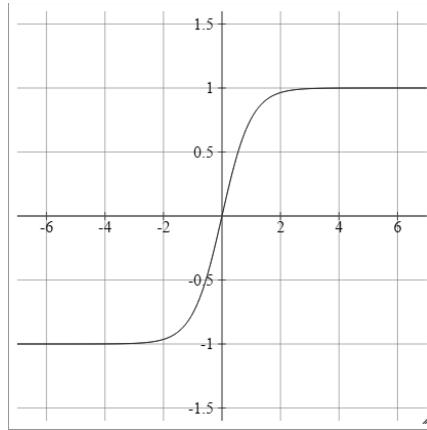


Figure 2.5: Hyperbolic tangent activation function.

would have dense neuron activation while ReLU would have a smaller number of activated units. This smaller number yields faster and more efficient computation (Glorot et al., 2011 [21]).

Like everything else, ReLU has its drawbacks. The zero-valued horizontal line in Figure 2.6 for every $x \leq 0$ is similar to the extremes in sigmoid and hyperbolic tangent functions, with the difference that here the gradient is actually zero. This means that those non-activated neurons will stop responding to variations in error/input and will not move from there. This is called the "dying ReLU problem." A possible solution to this is to lift up/-down the horizontal line by a minimum amount (example: $f(x) = 0,001x$), allowing the values of neurons to still move and maybe activate in a second moment. This solution is known as leaky ReLU. In the next section the most commonly-used cost functions within ANNs will be reviewed.

2.2 Learning

It has been mentioned that ANNs learn from being given training data; such learning process, however, is not as straightforward as it seems. Training data, cost functions, backpropagation methods and other learning parameters

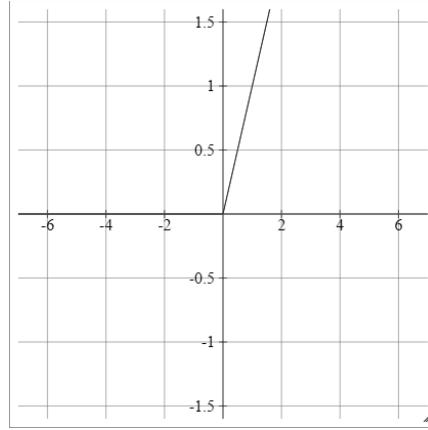


Figure 2.6: Rectified linear unit activation function.

need to be adjusted for reaching the best training performances. In the next sections all the methods for achieving the best results will be addressed.

2.2.1 Cost Functions

When training, the network tries to understand what the given input is, and predicts an output. Learning in this case means finding the output that is optimal compared to the ground truth of the given input. Given a class of functions F and a function $f^* \in F$ that is as similar as possible to the ground truth, a cost function is used $C : F \rightarrow \mathbb{R}$ such that for the given function f^* there is no better solution in class F :

$$C(f^*) \leq C(f) \forall f \in F.$$

Before going into details, the cost function must satisfy two main requirements. First, a cost function should produce a non-negative value. Second, the cost function value should become as smaller as possible when the predicted output of the network is reaching the ground truth, 0 being the perfect match.

The notational convention used in the following cost functions is of the form $C(h(x), y)$, where C is the cost of the predicted output of the network

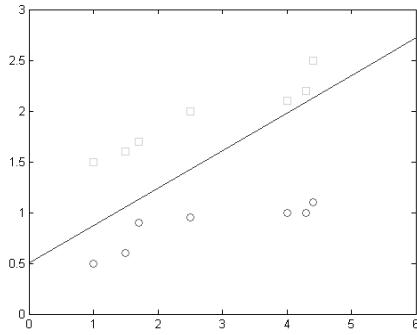


Figure 2.7: An example of how a linear model classifies two kinds of data.

prediction $h(x)$ regarding the ground truth y . The most used cost functions are reviewed below.

Mean Squared Error (MSE)

The mean squared error is also known as quadratic cost, which is the difference between the network prediction outputs $h(x)$ and the ground truth y . The MSE is defined as:

$$C_{MSE}(h(x), y) = \frac{1}{m} \sum_{i=0}^m (h(x^i) - y^i)^2. \quad (2.5)$$

The letter i indicates different input samples and m defines the number of classes. The gradient of this cost function with respect to the predicted output of an ANN is:

$$\Delta_x C_{MSE} = (h(x) - y) \cdot \quad (2.6)$$

The MSE is used in linear regression models. In Figure 2.7, that shows two classes of data, red circles and green squares, a linear regression model tries to categorize the data in classes using just a linear function. This is no longer used in ANNs because logistic regression models have turned out to be better suited to this kind of task.

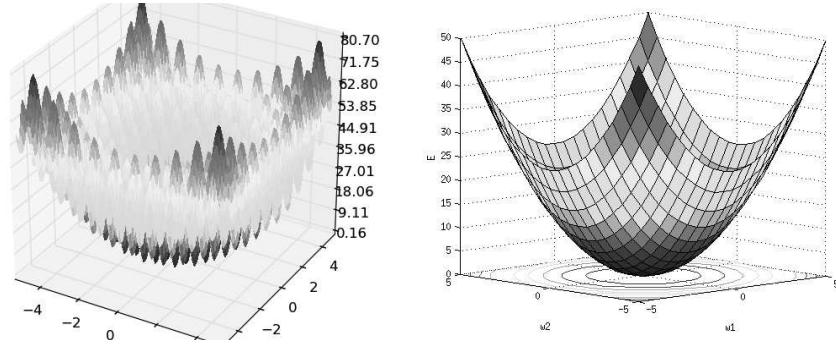


Figure 2.8: On the left there is a non-convex function [11] that has lot of local minimum and local maximum values, on the right instead a convex function [9] that is smooth all the way down to the global minimum. Images taken from Wikimedia Commons.

In addition, there are two problems associated with this cost function: (a) the problem of convexity and (b) high-value adjustments. In the case of (a), a non-convex function (see the left plot in Figure 2.8) has lots of local minimums and maximums so it can become harder to find the global minimum when using gradient descent methods. In the case of (b), as seen in Figure 2.10, if a prediction is slightly off the optimum (this would be $x = 1$) the cost function will be extremely high, so the adjustment of the network would be more laborious or practically impossible.

Cross-Entropy (CE)

As discussed above, the problem with the MSE in linear regressions is that it classifies small and large margins of error in its predictions with similar high costs. A way of differentiating the two error margins would be to use the cross entropy (CE) in a logistic regression model having the cost function as

$$C_{CE}(h(x), y) = - \sum_{i=0}^m y^i \ln(h(x^i)) + (1 - y^i) \ln(1 - h(x^i)), \quad (2.7)$$

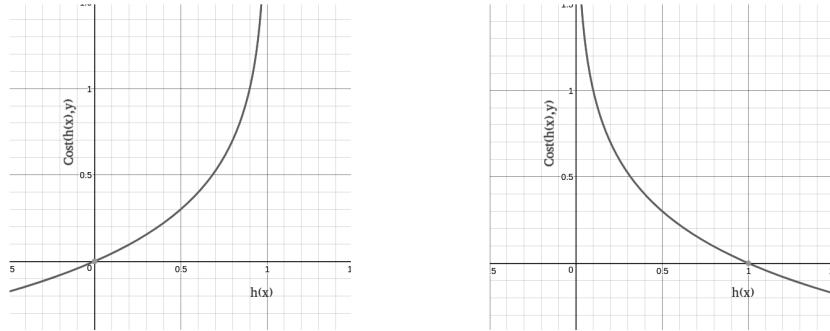


Figure 2.9: On the left the $-\log(1 - x)$ function is plotted, where $y = 0$; on the right, the $-\log x$, where $y = 1$. The x -axes are the predictions of the network $h(x)$ while the y -axes are the cost-function values.

where $h(x)$ are the hypotheses of the network for the given training example of the form:

$$h(x) = \frac{1}{1 + e^{-wx}}.$$

The gradient with respect to the predicted output is then:

$$\Delta_x C_{CE}(h(x), y) = \frac{(h(x) - y)}{(1 - h(x))h(x)}. \quad (2.8)$$

A closer look at equation (2.7) may suggest that the equation can be divided into two parts:

$$C(h(x), y) = \begin{cases} -\ln(h(x^i)), & \text{if } y = 1. \\ -\ln(1 - h(x^i)), & \text{if } y = 0. \end{cases}$$

Both equations can now be plotted as seen in Figure 2.9, where the plot on the left signifies $y = 0$ while the plot on the right signifies $y = 1$. In a binary classification task this means that a predicted class ($h(x)$) that is not the same as the ground truth (y), the cost would be extremely high (asymptotic to $+\infty$); a predicted class ($h(x)$) that matches the ground truth (y) would have a cost of zero considering the cost function.

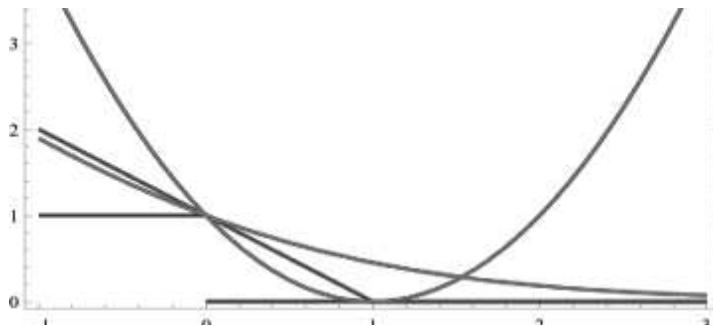


Figure 2.10: Plot of loss functions for classification task between two categories. Blue is the 0–1 indicator function. Green is the square loss function. Yellow is the logistic loss function. Purple is the hinge loss function. Image taken from Wikimedia Commons [10].

Hinge Loss

Another cost function worth mentioning is the hinge loss, i.e.,

$$C_{HL}(h(x), y) = \max(0, 1 - h(x)y). \quad (2.9)$$

A linear model with this function is known as a support vector machine. Even though SVMs come from a different community than neural networks the loss function behaves similarly to the logistic regression model explained above. In fact looking at Figure 2.10 the hinge loss and the log-loss are almost the same.

Softmax Regression

In the previous paragraphs three functions were discussed for linear and logistic regression. Those functions work only with binary classification problems, where the output y can only be of two classes, for example $y \in \{1, 0\}$. In recent years, the need for numerous output categories in the task of image classification (Deng, 2009 [14]) has become the norm. Consequently, softmax regression (or multinomial logistic regression) has came into use. It allows the handling of $y \in \{1, \dots, K\}$, where K is the number of output categories.

The cost function is then defined as:

$$J(\Theta) = -\left[\sum_{i=1}^m \sum_{k=1}^K 1\{y^{(i)} = k\} \log \frac{\exp(\Theta^{kT} x^i)}{\sum_{j=1}^K \exp(\Theta^{jT} x^i)}\right], \quad (2.10)$$

where \exp should be interpreted as e^x , Θ are the parameters of the model and the equation $1\{\}$ evaluates to 1 when there is a true statement in the brackets, 0 otherwise. The derivative of this cost function comes next:

$$\Delta_{\Theta^{(k)}} J(\Theta) = -\sum_{i=1}^m [x^{(i)}(1\{y^{(i)} = k\}) - P(y^{(i)} = k | x^{(i)}; \Theta)]. \quad (2.11)$$

Equation (2.11) cannot be minimized analytically but it can be solved with an optimization algorithm using its derivatives as described in Section 2.2.3.

2.2.2 Network Weight Initialization

Before the start of the training the network should have the weights initialized. In this section some of the common methods are presented.

All-Zero Initialization

An idea of initialization could be to set every weight to zero. Here are two negative consequences of doing so.

- The fact that activation functions like sigmoid (2.2) or ReLu (2.4) have a weight that equals 0, entails being stuck with a dying weight problem.
- If, on the other hand, every weight is the same, the update process will adjust all weights by the same amount. This means that every weight will be changed by the same gradient. This particular behavior causes the network to have what is known as symmetric weights and is one of the problems that needs to be avoided when initializing a network (Chapter eight of Goodfellow, 2016 [22]).

That said, zero initialization is not a used approach in ANN.

Random Initialization

Random initialization is a straightforward approach that differs from the previous one and solves the problem of symmetry in that the weights are randomly initialized with a small variance. Thus, every neuron will behave and train differently. There are several different ways of choosing the random parameters; one such method is the Xavier initialization.

Xavier Initialization

Xavier initialization is the most widely used method of initialization. It uses random initialization with some constraints that prevent the weights from being too small (dying weights) or too big (activation saturation). It helps the input to reach deep into the network without being suppressed in the first layers. The weights are taken from a zero-mean Gaussian distribution where the best variance stated by Glorot & Bengio's [20] is

$$Var(W) = \frac{2}{n_{in} + n_{out}},$$

where W is the initialization distribution of the neuron taken into consideration, n_{in} is the number of neurons as input and n_{out} the number of neurons as output.

Shallow-to-Deep Net Initialization

Another different approach to initialize the weights was used by Simonyan & Zisserman, 2014 [58] who used a random initialization for the most shallow architecture they had (11 layers). After training the network, they used the updated weights for the initialization of a deeper architecture. The additional layers of the deeper networks were initialized randomly. They repeated the weight transfer until they were able to train their deepest 19-layer network.

2.2.3 Network Weight Update

As mentioned in the cost functions Section 2.2.1, gradient calculations are needed; the reasons why those calculations are important are addressed in this section. Finding the best parameters of an ANN means achieving the best results that the network can yield. Starting with a random weight initialization will cause the network to give a non-optimal performance. Adjusting the weights in a proper way, on the other hand, will help the network learn the features needed to give an optimal performance. The most commonly used optimization techniques are based on gradient descent, which updates the weights in accordance with the following equation (Bottou & Léon, 2012 [4]):

$$w_{ji} = w_{ji} + \Delta w_{ji}, \quad (2.12)$$

where Δw_{ji} is the gradient of the cost function as shown in:

$$\Delta w_{ji} = -\eta \frac{\delta C_x}{\delta w_{ji}}. \quad (2.13)$$

The gradient is calculated regarding the weight w_{ji} multiplied by the learning rate factor η .

Updating the weights based on a single sample x , however, is usually not desirable because is too specific and misleading. A more efficient update will be based on a batch of samples, called mini-batch gradient descent (or stochastic gradient descent), that updates the weights more smoothly. The size of the batches is usually a power of 2 (32, 64, 128, 256, 512) (Goodfellow, 2016 [22]), where batches that are too large lead to the sharpest minimum and hence to lower generalization and lower performances (Keskar et al., 2016 [39]). Some of the commonly used optimizers are explained below.

Momentum

Since the objective function usually does not have the perfect convex shape of a function (check Figure 2.8 depicting a convex function), an SGD could get stuck in one of the local minimums. To overcome this, a momentum

factor is introduced that is multiplied by the gradient of the previous time step, i.e.,

$$\Delta w_{ji}^{(t)} = -\eta \frac{\delta C_x}{\delta w_{ji}} + \alpha \Delta w_{ji}^{(t-1)}. \quad (2.14)$$

The momentum $\alpha \in [0, 1]$ needs to be chosen according to the learning rate, which is usually around 0.9 (Qian, 1999 [54]). Large learning rates with a high momentum have too big a step and will not reach the global minimum (Sutskever, 2013 [61]). Another method that is suitable to SGDs is the AdaGrad method.

AdaGrad

AdaGrad (Duchi et al., 2011 [17]) adaptively chooses the learning rate for each parameter separately, rather than using a single update for all parameters. Dean et al., 2012 [13] found that AdaGrad greatly improved the robustness of SGDs and used it for training large-scale neural networks at Google.

The learning rates are set as:

$$\eta_{i,K} = \frac{\eta}{\sqrt{\sum_{j=1}^K \Delta w_{i,j}^2}}, \quad (2.15)$$

where $\eta_{i,K}$ be the learning rate of the weight $w_{i,j}$, where i stands for the number of weights at the iteration j . The $\Delta w_{i,j}$ represents the gradient of the weights at a single iteration. η in the nominator is a factor, bigger than the commonly used η set as 0.01. The update rule for the weight will then be:

$$w_{i,j}^{(t+1)} = w_{i,j}^{(t)} - \eta_{i,j} \Delta w_{i,j}^{(t)}. \quad (2.16)$$

This method has its good properties as it eliminates the manual tuning of the learning rate η and it updates every weight w properly with its own learning rate. If AdaGrad has a weakness is tendency to accumulate the sum of gradients throughout every iteration, which can shrink the learning rates so much that the weights cannot learn anymore. The following methods aim to solve this problem.

AdaDelta

AdaDelta (Zeiler, 2012 [74]) tries to solve the problem of the accumulated sum of gradients by considering only a window l of past gradients. In doing so, the sum of gradients will not sum up indefinitely but will stay close to a high number. But storing the sum of l previous gradients is inefficient, so AdaDelta instead introduces a decaying factor of running averages, noted as $E[\Delta w_{ij}^2]$, that is updated at every time step as in the next equation:

$$E[(\Delta w_{ij})^2]^{(t+1)} = \eta E[(\Delta w_{ij})^2]^{(t)} + (1 - \eta)(\Delta w_{ij})^{2(t)}, \quad (2.17)$$

where η is the learning rate factor (used as decay factor) and $(1 - \eta)(\Delta w_{ij})^{2(t)}$ is the updated value that is added to the decaying running average. That noted, the learning rate will change to:

$$\eta_{i,K} = \frac{\eta}{\sqrt{E[(\Delta w_{ij})^2]}}. \quad (2.18)$$

Thus the root MSE of the gradient can be written with the shorthand RMS. From the above examples it is evident that the factor η is still present in the equation. In order to remove it and in order to fix the problem of unit number mismatch (see section 3.2 of Zeiler, 2012 [74]) the same sort of running averages replaces η with the parameter update as $E[w_{ij}^2]$. This replacement can be inserted as the numerator of equation (2.18) above of the previous time step as:

$$\eta_{i,K} = \frac{RMS[w_i]^{(t-1)}}{RMS[\Delta w_i]^{(t)}}. \quad (2.19)$$

In the last equation it is not necessary to set the global learning rate η because it has been eliminated from the update rule.

RMSprop

Another approach similar to that of AdaDelta is the RMSprop (Hinton, 2012 [31]). In fact this method was developed at the same time as AdaDelta when both approaches were intended to be solutions for the problem of learning rate shrinkage. The running average and the update rule in RMSprop

are identical to the first updates of AdaDelta ((2.17) and (2.18)) choosing $\gamma = 0.9$ and $\eta = 0.001$.

Adaptive Moment Estimation (ADAM)

One of the newest methods for adaptive learning is the ADAM method conceived by Kingma & Ba, 2014 [40]. It holds the decaying average of past squared gradients v_t like AdaDelta and RMSprop; it additionally holds a decaying average of past gradients m_t , i.e.,

$$m_{(t)} = \beta_1 m_{(t-1)} + (1 - \beta_1) \Delta w_{(t)}, \quad (2.20)$$

$$v_{(t)} = \beta_2 v_{(t-1)} + (1 - \beta_2) \Delta(w_{(t)})^2. \quad (2.21)$$

As stated by Kingma & Ba, due to the initialization of vectors as 0's, all the numbers are biased towards zero. Computing the bias-corrected estimates $\hat{m}_{(t)} = \frac{m_{(t)}}{1-\beta_1(t)}$ and $\hat{v}_{(t)} = \frac{v_{(t)}}{1-\beta_2(t)}$ the update rule can be estimated:

$$w^{(t+1)} = w^{(t)} - \frac{\eta}{\sqrt{v^{(t)}} + \epsilon} \hat{m}_{(t)}. \quad (2.22)$$

The proposed values for the equations are: $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

Previous sections described some of the most used methods for weight update. There are also other methods, such as for example:

- NAG, nesterov accelerated gradient (Sutskever, 2013 [60], Ruder, 2016 [55])
- Nadam, that joins the ADAM optimizer with NAG (Dozat, 2015 [16], Ruder, 2016 [55])
- Exponential Decay Learning Rate (Senior et al., 2013 [57])
- Power Scheduling (Senior et al., 2013 [57])
- Performance scheduling (Senior et al., 2013 [57])

Considering what has been said so far, the methods reviewed in this section outline some of the improvements of optimizers over the last decade. First, the momentum method adds a portion of the previous gradient velocity

to skip possible local minimums and reach the best optimization. With AdaGrad, the problem of the manual learning rate setting was solved. AdaDelta and RMSprop were similarly developed to overcome the problem of learning rate shrinkage and finally the Adam optimizer, with its bias correction, outperforms the previous two at the end of training when gradients become sparser (Kingma & Ba, 2014 [40], Ruder, 2016 [55]). The best choice so far seems to be Adam. All the above methods are then used in the process of learning with backpropagation.

Backpropagation

The weighted sum of the inputs is computed for every neuron in the network; an activation function is applied to such sum and the result is passed on to the next neuron in the next layer. This cycle is known as a forward-pass. At the beginning of the learning process the network knows little or nothing, so the outputs in the last layer (predicted categories) will be wrong. Every neuron can be represented in a form such as

$$y = \sum (f(x_i w_i) + b_i),$$

where f is an activation function applied to the elementwise multiplication between the input x and the weights w . Additionally a bias term b is added. Adjusting the parameters of the equation aids the network to predict correct outputs. One of the known algorithms to achieve this is the backpropagation algorithm, which computes the partial derivatives of the cost function regarding the weights $\frac{\delta C}{\delta w}$ and the biases $\frac{\delta C}{\delta b}$ from the output layers to the first hidden layer. Then using the derivatives it adjusts the weights and biases for achieving a better output. Here the optimizers come in handy to help the learning process. The learning finishes when the loss function is minimum for every possible training example on which the network is learning. The present work will not go into the mathematical proofs which are already nicely discussed in Nielsen, 2015 [50].

The learning of a network is usually carried out on a subset of the data that can be forwarded to the network. Take for example a network that needs to classify chair and car images. One such network should train on a small number of images from those two categories first; the network should then be tested. A subset of training images is known as a training set. Images that are not part of the training set are part of a validation or test set. This means that the network will train on the same images until its learning no longer converges to the minimum. This also means, that the network will learn to recognize and classify only the images that are highly similar to the training set. This limitation on the part of the network is known as overfitting and will be explained in the next section with the inclusion of methods to prevent it.

2.2.4 Overfitting and Regularization

The ability of the network to predict good outputs on new inputs is called generalization of the learning process. Having a network that performs well in every circumstance (training and validation/test sets) means that the network has the minimal generalization error, which is the test error on new inputs. Figure 2.11 shows two models of the network ability to classify the available data: the black line represents a smooth fit, whereas the green line depicts an overfitted classification which fits perfectly on all the data. From an analytical point of view the black model has a lower accuracy in fitting the data, but on the other hand it will fit new data better than the green model. The techniques used for diminishing overfitting, which will be explained in the next sections, are:

- Data augmentation
- Early stopping
- Dropout
- DropConnect
- L2 & L1 normalization
- Batch Normalization

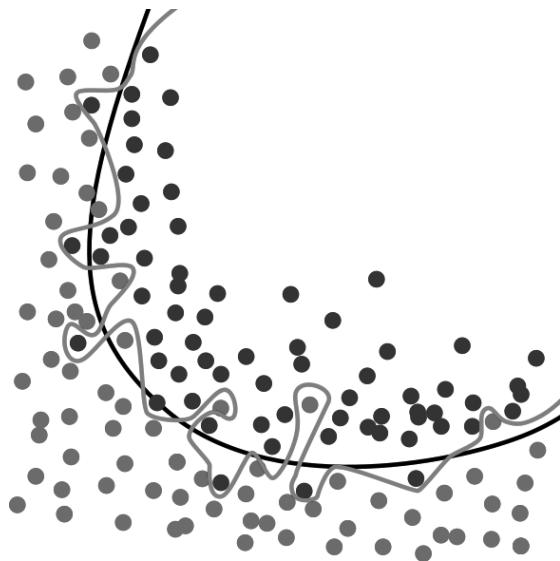


Figure 2.11: Plot of two types of data (red and blue dots) for binary classification. The two lines represent two models that classify the data. The black line is a regularized model, the green an overfitted model. Due to the perfect match, the green model is likely to have a higher error rate on new data. Image taken from Wikimedia Commons [8].

Data Augmentation

There are two solutions to this inconvenience. Either a larger dataset with more different samples can be used or a data augmentation procedure must be carried out. Briefly, data augmentation is an approach that transforms the training data so that the network can train on samples that the network views as different (Krizhevsky et al., 2012 [42], Zeiler & Fergus, 2014 [75], Szegedy et al., 2015 [63], Wong et al., 2016 [71]. The most popular practices are horizontal and vertical flipping, random crops, color jittering, rotation and scaling. During the years when the neural network "tsunami" flooded the fields of image classification and recognition, new methods of augmentation came out.

In 2012, Krizhevsky et al. [42] proposed the approach of image translation and horizontal reflections extracting random patches (original image size: 256×256 , crop: 224×224) from the original images. The intensities of the RGB channels of the images, a process known as PCA whitening. Therefore to each RGB image pixel $I_{xy} = [I_{xy}^R, I_{xy}^G, I_{xy}^B]^T$ the following quantity is added $[p_1, p_2, p_3][\alpha_1 \lambda_1, \alpha_2 \lambda_2, \alpha_3 \lambda_3]^T$ where p_i and λ_i are the eigenvectors of the 3×3 covariance matrix of the RGB pixel values. According to Krizhevsky, this approach reduced the top-error rate by 1%. Zeiler & Fergus, 2014 [75] and Szegedy et al. 2015 [63] used similar approaches with some adjustments in numbers and ways of cropping. The best cropping approach was the one by Szegedy et al. They resized their images to 4 scales: 256, 288, 320 and 352. From those images the center, the left side and the right side (top and bottom for portrait images) were extracted. These were resized, cropped and mirrored out of a total of 144 crops. Szegedy et al. architecture won the ILSVRC classification challenge in 2014 with a 5.6% top-5 error.

Wong et al., 2016 [71] is another approach, instead of augmenting the data before the training (in the so called data space) proposed the possibility of augmenting the data in the feature space. The results do not improve the performance of neural networks but they reduce the overfitting problem.

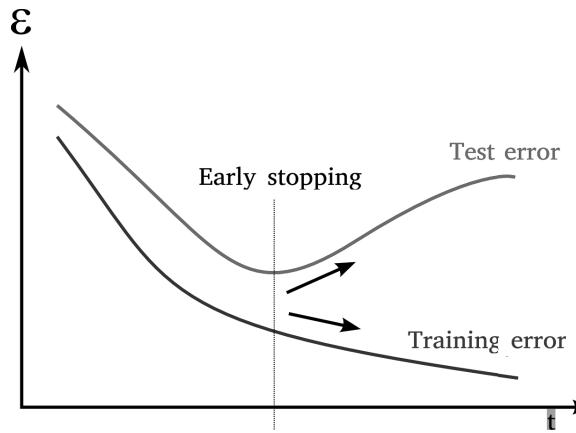


Figure 2.12: Plot of error functions. The blue line represents the training error, the red line the test error and the vertical dotted line the latest moment to avoid overfitting with the early stopping approach. The training error is still dropping giving false information of the actual test error.

Early Stopping

Figure 2.12 shows that after a certain point in time the training error is still converging to the minimum, but the test error starts to diverge. In that moment the network is starting to overfit. The simplest solution to avoid the use of data augmentation or other methods to prevent overfitting is to stop the training process right before the test error starts rising.

Dropout

Due to the more complex network architectures that have been developed in the last years, the number of parameters in such architectures is really high. Complex networks tend to learn more and tend to overfit a lot when fed with small datasets. Dropout is a method proposed by Hinton et al., 2012 [32] and additionally researched by Srivastava et al., 2014 [59] that discards some of those parameters and prevents overfitting. For every training case, the probability that a given hidden unit will be randomly left out is 0.5 (or with

$0 < p < 0.5$). For every new training case the omitted units are restored. Additionally the DropOut method improves performance due to its smaller computational needs. Consider the usual equation for a neural network

$$y = f(Wx), \quad (2.23)$$

where x represents the inputs, W the weights and f an activation function which gives the output y . Using DropOut the equation changes to

$$y = m * f(Wx), \quad (2.24)$$

where m is a binary matrix (of 1s and 0s) whose elements are taken from a Bernoulli distribution with probability p . Due to the properties that many activation functions have $f(0) = 0$. The equation can be rewritten as:

$$y = f(m * (Wx)). \quad (2.25)$$

With the last equation (2.25) the calculations of the activation function for all zero-valued outputs are straightforward, resulting in an acceleration of the calculation process.

A drawback of the DropOut method is that the output of the neurons still needs to be calculated. A different approach that achieves a better performances is the DropConnect method.

DropConnect

As the name suggests, DropConnect is similar to the DropOut method, but uses a more generalized approach that drops connections instead of neuron outputs (Wan et al., 2013 [67]). Instead of setting neuron outputs to 1 or 0, it randomly drops connections (weights) with a 0.5 probability during the training stage. The output of a layer with DropConnect would be written similarly to the one for DropOut, but multiplying the weights and the binary matrix first:

$$y = f((m * W)x). \quad (2.26)$$

According to Wan et al., [67] the DropConnect method achieves better performances than the DropOut method with the test set.

Weight Decay

Instead of cutting out neurons or connections, there is another way to solve overfitting and improve performance. If a network overfits, this means either that the network is too complex for that kind of data or that there are too few data on which the network can train. The problem of sparse data is solved through data augmentation (see Section 2.2.4). As for the problem of the network's complexity, one does not need to drop out units; instead, weights can be penalized with a regularization term that imprints some additional noise onto the data. Large weights can be penalized in their update phase using constraints on their squared values (L2 regularization) or on their absolute values (L1 regularization). Hence, the weights will be properly updated in the backpropagation process. Adding a parameter to the cost function will solve the issue. The cost function below is the cross-entropy method with an added term for regularization (2.13) (Nielsen, 2015 [50]):

$$C = -\frac{1}{n} \sum_{x_j} [y_j \ln a_j^L + (1 - y_j) \ln(1 - a_j^L)] + \frac{\lambda}{2n} \sum_w (w^2), \quad (2.27)$$

where λ is the L2 regularization constant and n the number of training samples (more information can be found in Chapter seven of Goodfellow, 2016 [22] and in Chapter three of the Nielsen, 2015 [50]). The update rule of the weights will then be:

$$w' = (1 - \frac{\eta\lambda}{n})w - \eta \frac{\delta C}{\delta w}. \quad (2.28)$$

The L1 normalization, instead, takes into account the absolute values of the weights:

$$C = -\frac{1}{n} \sum_{x_j} [y_j \ln a_j^L + (1 - y_j) \ln(1 - a_j^L)] + \frac{\lambda}{n} \sum_w |w|. \quad (2.29)$$

The update rule will then be:

$$w' = w - \frac{\eta\lambda}{n} sgn(w) - \eta \frac{\delta C}{\delta w}, \quad (2.30)$$

where $sgn(w)$ is the sign of the value of the weights. In both expressions we can see that both L1 and L2 norms shrink the weights; the way they

work, however, is different. The L1 norm regularizes the weights by the same amount every time; L2 regularizes proportionally to the actual weight.

The above two methods allow a weight to be very large. The Max Normalization method is used to prevent this; this method sets an upper bound for the weight's magnitude; this limit needs to be respected at the end of every weight update.

Batch Normalization

A method that has completely changed the ways of training and regularization is Batch Normalization (BN) (Ioffe & Szegedy, 2015 [35]). In the following we explain the main benefits of using this method in deep learning.

The most used optimizers for network weight update were summarized in Section 2.2.3 (e.g. Momentum). Those methods are based on SGD optimization, which works with batches of input samples rather than with one sample only; this improves the network learning curve, but brings about some problematic aspects. The distribution of samples varies across different batches. This difference creates a sort of training redundancy where the network parameters constantly need to be adjusted based on every new distribution; this adjustment is time consuming. The phenomenon of changes in distributions in the hidden layers is known as "Internal Covariance Shift" (Ioffe & Szegedy, 2015 [35]).

It was already stated by Wiesler & Nye in 2011 [69] that the network learns faster if the inputs are decorrelated, and have (a) zero-means and (b) unit-variances (i.e. are prewhitened). The calculations for whitening are complex and time consuming, especially in large networks such as the ones widely used for addressing tasks of image classification (Krizhevsky, 2012 [42], Goodfellow, 2013 [23], Simonyan, 2014 [58], Szegedy, 2015 [63], He et al., 2016 [29]). The BN approach addresses this problem by simplifying the calculations. The outline of the BN is given in Algorithm 1.

The steps listed in Algorithm 1 show that no complex calculations are needed. In fact, one does not need to calculate a covariance matrix for all

Algorithm 1 In steps (a–d) the mean μ of every input x (a) is calculated. In (b) step the variance σ^2 is calculated using the previously obtained mean. Every input is then normalized, \hat{x} (c). In the last step (d) the output is generated, by scaling the original input x with η and shifting it with β .

- (a) $\mu_\beta \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$
 - (b) $\sigma_\beta^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_\beta)^2$
 - (c) $\hat{x}_i \leftarrow \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}}$
 - (d) $y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i)$
-

the inputs x , which on the contrary would be mandatory when using the whitening approach.

The BN approach is used as an additional layer that is inserted between two layers of an ANN, but this cannot be done with every layer: fully-connected and convolutional layers are the layers that earn more. If BN is a layer, then, it needs to suit the forward and backward passes. In fact what is written in Algorithm (1) is already the forward pass.

Some problems could arise with the backpropagation methods. Ioffe & Szegedy, 2015 [35] lists the derivatives that are proof that the backpropagation methods can be used. Stated so, the BN layers can be used in both passes and are in fact exceptional in gaining faster training rates. Additionally, BN is a regularization technique too, and thus can limit the use of dropout.

The techniques and methods to make an ANN learn has been discussed in this chapter. Having laid down the theoretical background, we proceed in the next chapter to discuss different types of ANNs: convolutional neural networks and recurrent neural networks and their practical usage today.

Chapter 3

Convolutional Neural Networks

The ANNs usually work on a set of inputs that are stored within a vector. Convolutional neural networks (CNN) differ from typical ANNs in that they are structured to work on images, but they are similar enough to allow the theory of the previous chapter to be used.

With the advent of CNNs, some problematic aspects arose within this new kind of data. Every image has its height, width and depth (i.e. channels as RGB) and holds more data to be processed by the network. The conventional approaches for limiting the usage of data and accelerate the training process were no longer enough.

An idea of the bulk of the processing can be given by calculating how many weights are needed to elaborate a vector against an image. For example: a vector of length 200 and an image of size $200 \times 200 \times 3$ (height, width and RGB channels respectively) will be compared. Taking into consideration just one fully connected layer, which is a layer where every neuron is connected with every neuron in the next layer, the number of weights for a vector will be 200, while it will be 120000 for the image; that is 600 times more than the vector. This is computationally too expensive even for the computers of today. A different method for getting less information per each neuron was a must.

As the idea for ANNs was inspired by the biological aspect of the human

brain; the idea for CNNs was inspired by the same aspect, but this time from the organization of the animals visual cortex, where individual cortex neurons stimulate different parts of the receptive field. From this idea new types of layers were introduced for elaborating just part of the entire image: convolution and pooling layers. Some terminology and a set of concepts will be briefly defined before going into the layer's details.

First, the word "convolutional" in CNNs expresses the use of convolutions in the network architecture. This expression comes from digital signal processing and is a technique that combines 2 signals into a third. In CNNs the convolution is used with images, combining two images (or parts of them) into a third one. In CNNs the first image is usually the input image and the second image is a filter or kernel. The output is known as an activation map that is forwarded into the next network layer.

Kernels or convolutional filters are important in convolution calculations. The result of the multiplication of the kernel and the input image shows which parts of the image activate the most based on that particular kernel (in fact, the term activation maps refers to this aspect.) In the learning process, different filters will be used and consequently different kernel-specific regions will be activated that are specific to the input image. A set of kernels used in image processing is shown in Figure 3.1.

3.1 CNN Building Blocks

3.1.1 Convolution Layer

The convolution layer is the first of the new types of layers to address the amount of data in CNNs. It is always used as the first hidden layer after the input. As the name suggests, the convolution layer uses convolutions in its calculations. A simple example of a layer calculation is shown in Figure 3.2 where the kernel, the small 3×3 yellow matrix (b), is convolved with the

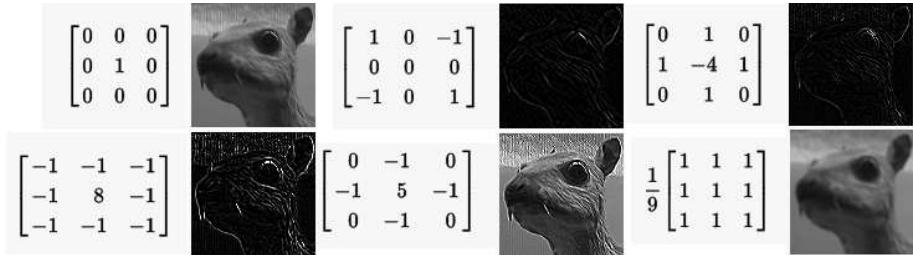


Figure 3.1: Example of image processing kernels. Respectively, from top left to bottom right: identity, three different edge detection, sharpen and box blur kernels. Images taken from Wikimedia Commons [70].

$5 \times 5 \times 3$ blue matrix (a). The convolution calculations can be written as:

$$y_{i'j'k'} = \sum_{ijk} w_{ijk} * x_{i+i'j+j'k}, \quad (3.1)$$

where y is the output of the sum of the element-specific multiplication (noted as $*$) of the image pixels (receptive field) x by a filter with weights w . The equation is similar to the equation of ANN (2.23), the only difference being the input volume (or depth) noted with the subscript $(\cdot)_k$.

One important thing that needs to be taken into account is the kernel depth. If the input image has 3 channels (as RGB) the kernel with which it will be multiplied must have the same depth of 3. The kernel multiplication process with every piece of the image is called kernel sliding. In the multiplication process the kernel size needs to be accurately chosen to fit the image size. Zero-padding and stride parameters can be set to adjust such incongruences.

The zero padding is shown in Figure 3.2 (in the first matrix (a)) where 0s are added all around the image. This method is additionally used to maintain the same size of the input and output volumes between layers.

The stride parameter, instead, indicates how many pixels are skipped between kernel multiplications. For example: a 5×5 matrix with kernel 3×3 and stride 1 will compute 9 kernel multiplications, 3 within rows and 3

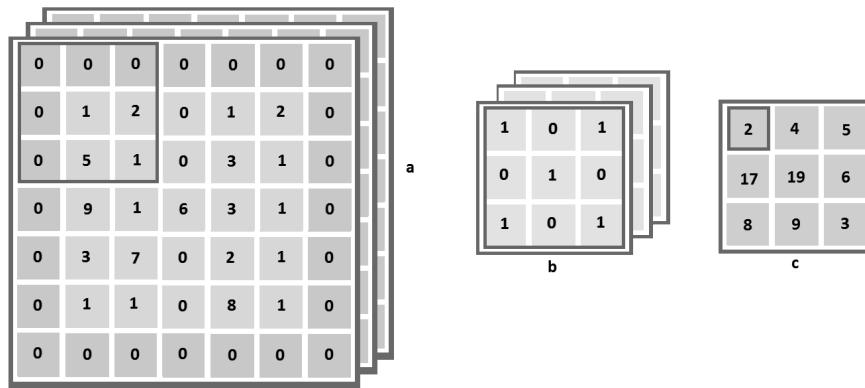


Figure 3.2: The first matrix (a) represents an RGB image $5 \times 5 \times 3$ (in light blue color), with 1 additional line around it for zero padding (light grey 0s). The second matrix (b) represents the filters or kernels (in light green color) that are multiplied by the (a) matrix. The last matrix (c) is the activation map of multiplication between the first two matrices with a stride of 2 with just the first filter out of three (b). This means that the matrix (b) is multiplied three times for every row and column of the matrix (a). The red square on the matrix (a) shows the receptive field.

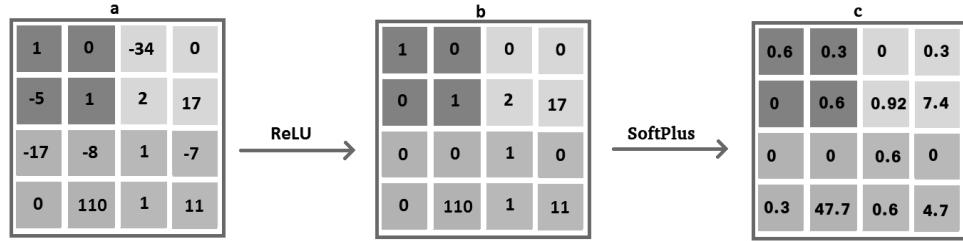


Figure 3.3: Example of ReLU and SoftPlus layer outputs. The matrix (a) is a slice of the activation map, the matrix (b), is the activation of the ReLU layer. The negative values are set as 0s. The matrix (c) is the activation of the SoftPlus layer.

within columns; the same multiplication with stride 2, instead, will compute only 4 multiplications.

3.1.2 Non-Activation Layers

After every convolution layer, a non-linear activation layer is used. This means that we use a non-linear function for every input of the previous layer. One such function is the ReLU that was already explained in Section 2.1.4. The SoftPlus is another function that is smoother than ReLU and is used in CNNs (Glorot et al., 2011 [21]), the equation follows:

$$f(x) = \ln(1 + e^x), \quad (3.2)$$

in softplus the derivative is the known sigmoid function, i.e.,

$$\delta f(x) = \frac{1}{1 + e^{-x}}. \quad (3.3)$$

An example of the functionality of the ReLU and Softplus layer is shown in Figure 3.3.

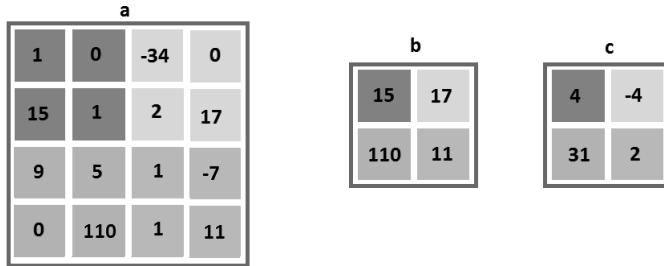


Figure 3.4: Max and average pooling with size 2×2 and stride 2. The matrix on the left (a) represents an activation map. The matrix in the middle (b) is the result of the max pooling (for red values in (a) the result is the maximum value of numbers: 1, 0, 15 and 1). The last matrix (c) is the average of the numbers' sum in each pool respectively.

3.1.3 Pooling Layer

The pooling layer, also known as down-sampling layer, helps to reduce the input volume of the previous layer. It does not change the depth of the activation maps, but it reduces their width and height. Is it usually inserted after a convolution layer. The process of down-sampling, even though it loses information, has two positive aspects: first, less information means faster computation/calculations, and second, the process works against the overfitting problem (already discussed in Section 2.2.4). The pooling works similarly to the convolution. The size of the pool and the operation of choice are defined. The pool slides over the input activation map with a selected stride and creates the output. The output is calculated using two approaches: average and max pool. The first approach selects the average of all the values in the pool, while the second selects the maximum value. Max pool is preferred in practice, due to its simplicity, its computational performance and its better practical results. Figure 3.4 is shown how those two variants work.

3.2 CNN Architectures

The first CNN ever developed was the LeNet-5 by Yann LeCun et al. in 1998 [44]. The main objective of this CNN was to recognize handwritten digits. The architecture consists of 7 layers in the following order: convolution, pooling, convolution, pooling, fully connected, fully connected and classification. The input layer is of the size of a 32×32 grayscale image. The actual sizes of the handwritten digits are no bigger than 20×20 pixels. The bigger size is chosen carefully so that the contours and the edges of the digits can be taken into the centers of receptive fields, when the convolution occurs. The first layer uses 6 kernels of size 5×5 having a feature map output of the size of $28 \times 28 \times 6$. The subsampling layer thereafter has a pool of size 2×2 without pool overlapping. The output of this last layer is then 14×14 which is multiplied by a trainable coefficient; a trainable bias is added and a sigmoidal function is applied for non-linearity. The next convolution layer with 16 kernels is not connected with every feature map slice of the previous one, but specific kernels are connected just to previous specific feature map 5×5 patches. Thus, the number of calculations and parameters is lower and the symmetry of the network is broken. The next subsampling layer does the same thing as the previous one having 16 feature maps of the size 5×5 . The third-to-last layer, called fully connected (FC) layer, is actually a convolution layer that maps the 5×5 previous layer patches to a single 1×1 . It has 120 kernels, hence the output is $1 \times 1 \times 120$. The second-last layer (also FC) just connects the previous one to 84 features. The output layer classifies 10 digits.

By building upon this, the first and simplest of CNN architectures, other architectures were then developed and researched that covered the topics of image classification/recognition. Some of the most influencial are described in the following.

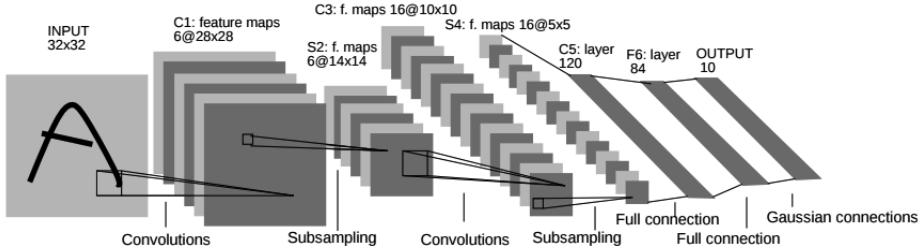


Figure 3.5: The architecture of the LeNet5 CNN. The input is a handwritten digit that is connected to the first convolution layer; after that, subsampling, convolution and FC layers are connected to achieve the desired classification. Image taken from LeCunn 1998, IEEE [44].

3.2.1 AlexNet

LeNet-5 was the first kind of CNN with good performance in handwritten digit classification; in 2012, however, Krizhevsky et al. [42] developed an architecture in a paper which became the most cited in the field of CNNs. This was the first architecture to outmatch the conventional algorithms in image classification by a large margin. In fact, Super Vision (the name given by the authors to the CNN model) won the 2012 ILSVRC classification challenge (Russakovsky et al. [56]) with only 16.42% of error (top-5). The model that placed second had an error of 26.17%, which is approximately 10% higher. The size of the ILSVRC dataset was, and still is, an important factor when such networks are learning. It is divided into 1000 categories with more than 1.2 million images; hence learning speed and memory constraint problems have to be addressed.

AlexNet (after the author's first name: Alex) yielded some improvements over the original idea of LeNet-5. Training speed was achieved by replacing the sigmoid activation function with the ReLU non-linearity function. In fact, training time was 6 times lower to achieve the same results. Furthermore, the architecture was split into two halves in order to spread the learning process over two GPUs. The split architecture is seen in Figure 3.6 which shows

the horizontal division. Krizhevsky et al. implemented a local response normalization (called "brightness normalization") that normalizes the neurons' activation based on their neighborhood. This approach decreased the top-1 error by 1.4%.

Overlapped pooling was another improvement over LeNet-5, which brought a 0.4% better score. In Section 3.1.3 an approach of pooling was defined with a stride parameter s of the same size as the kernel border; Krizhevsky et al., instead, used a stride parameter, that is smaller than the kernel size (for example: $s = 2$, $k = 3 \times 3$, s for stride, k for kernel, then one pixel is overlapping). Overfitting was a problematic aspects when such bigger networks came into use. Krizhevsky et al. solved this problem by introducing the DropOut technique and a data augmentation approach, which consisted of image translations, image flipping and image cropping.

AlexNet architecture consists of 5 convolution layers and 3 fully connected layers (see Figure 3.6). The input size is $224 \times 224 \times 3$, which means that the network processes RGB images. The first convolutional layer spreads the activation maps over two GPUs with 96 kernels (48 on each GPU) of the size $11 \times 11 \times 3$ with a stride parameter of 4. The second layer takes the normalized and pooled inputs and filters them with 256 kernels (128 on each GPU) of size $5 \times 5 \times 48$. The normalized and pooled units are forwarded to the third (384 kernels of size $3 \times 3 \times 256$) to the fourth (384 kernels of size $3 \times 3 \times 192$) and fifth convolution layers (256 kernels of size $3 \times 3 \times 192$), till they are again pooled and normalized before reaching the FC layers and a Softmax output of 1000 categories.

3.2.2 Zeiler and Fergus network (ZFnet)

ZFnet, from the name of its authors (Zeiler & Fergus, 2014 [75]), was the CNN that outmatched the AlexNet ILSVRC classification score with a top-5 error rate of 13.5%.

ZFnet architecture does not differ substantially from AlexNet architecture but optimizes it. Zeiler & Fergus introduced the cross-entropy cost function

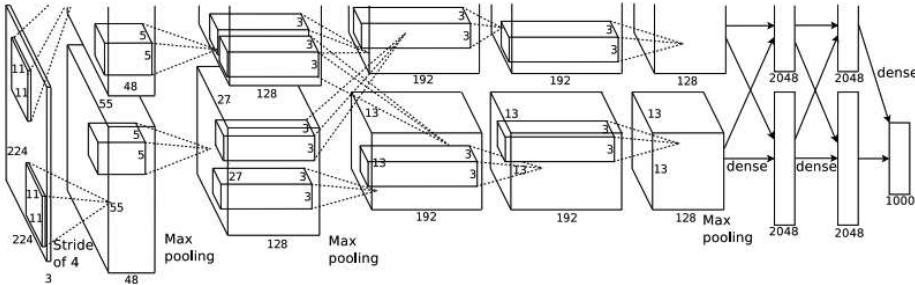


Figure 3.6: The architecture of AlexNet. The input layer takes $224 \times 224 \times 3$ -sized images. With the first convolution layer the kernels and the network are split into two lines (for double GPU training process). Max pooling and normalization layers are inserted after the first, second and fifth convolution layers. The last three are FC layers that are connected to a 1000-number classification vector. Image taken from Krizhevsky et al., 2012 [42].

and the learning process that is based on 128 images per batch (i.e. SGD). The Momentum update rule was used, with a Momentum factor of 0.9. The weights were initialized with the values of 10^{-2} . Image preprocessing, such as flipping, cropping and subtracting the per-pixel mean, were used. After some trials, it was discovered that certain filters prevailed with higher values over others, so a normalization approach was added to all those filters whose values exceeded an RMS of 10^{-1} . The new filter values were set to 10^{-1} accordingly.

An additional change over the AlexNet architecture was the joining of the third, fourth and fifth convolution layers over one GPU only and the switch to smaller convolution kernels (from 11×11 to 7×7 in the first layer and from 4 to 2 for the stride parameter). For more information see Figure 3.7. The different kernel size eliminated some artifacts created by the choice of the big stride and improved the overall performance. Another fact discovered by the authors was that a deep architecture improved performance; through trials with different CNN sizes they discovered that the best score was of the deepest network.

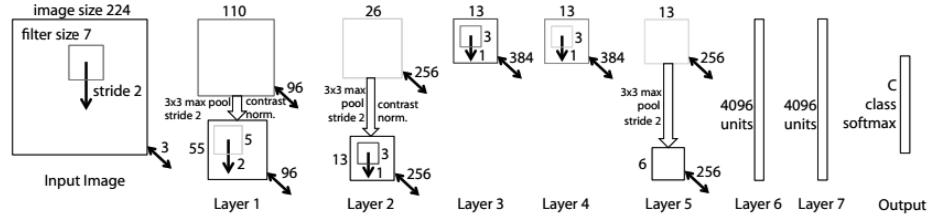


Figure 3.7: The architecture of ZFnet. The size of the input 224x224x3. The first kernels are sized 7x7, following 5 convolution and 3 FC layers. Max pooling and normalization are added after the first, the second and the fifth convolution layers. The FC layers are connected together to form a 1000-class classification vector. Image taken from Zeiler & Fergus, 2014 [75].

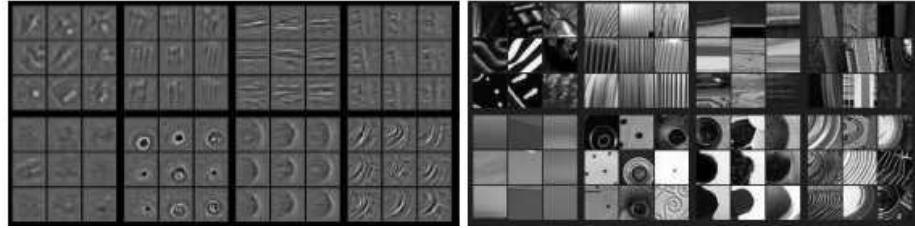


Figure 3.8: Visualization of kernels in the second convolution layer of the ZFnet. On the left, visualized kernels for the respective images on the right. Image taken from Zeiler & Fergus, 2014 [75].

The authors' discoveries yielded an interesting way of visualizing the learned filters. A network was created that reconstructed the results that came after convolution and pooling layers, from the feature-space back to the data-space, using the so-called DeconvNet. A visualization of their learned filters is visible in Figure 3.8.

3.2.3 Visual Geometry Group Network (VGGnet)

Since 2013, new ideas and architectures have arisen in the CNN field. VGGnet, from Simonyan & Zisserman, 2014 [58] shattered the previous year's

results. Intending to create a network that was deeper than the previous and as simple as possible, they developed different-sized networks the best of which was a 19-layer-deep network, that achieved 7.1% top-5 error score against the ILSVRC benchmark.

As a preprocessing method, only the approach of subtracting the mean RGB value from each pixel was used and no other structural changes were made. The main point of VGGnet architecture, instead, was to use only the smallest convolution filters (with receptive size 3×3 and a padding of one 0 to maintain the same width and height between the input and the output after a convolution layer). Such small filters can still gather the concepts of left/right, up/down and center, but in order to recreate the 5×5 or 7×7 receptive fields of some convolution layer filters, two or three conv layers (respectively) with 3×3 size must be used. Following this kind of logic, the depth grew and the number of parameters dropped. The max pooling is done with receptive fields of size 2×2 and a stride of 2.

The training phase is structured a little differently from the Krizhevsky approach in AlexNet. The training process included the batch size of 256, shallow-to-deep network weight initialization, weight decay with L2 regularization, dropout regularization and the images cropping approach (where the original images could have a border in the range of [256, 512] and then the images were cropped to the network input size of 224×224). The architecture of the deepest network is shown in Figure 3.9.

The number of parameters is one of the factors that has to be considered when using such deep networks. The deepest VGGnet has a total of 144 million parameters (weights).

3.2.4 GoogLeNet

Along with VGGnet, a completely different network appeared in the ladder of the ILSVRC benchmark. GoogLeNet (Szegedy et al., 2015 [63]), won the classification challenge with an astonishing 6.7% top-5 error rate, which beat VGGnet and proposed a different approach on how to build a CNN.

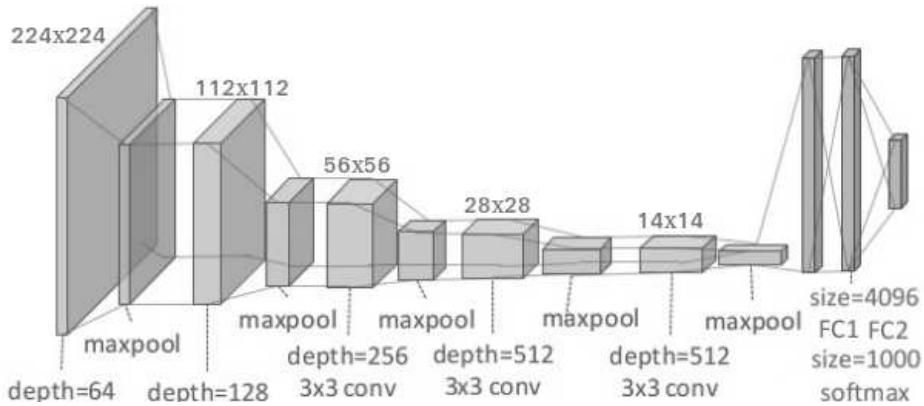


Figure 3.9: Architecture of VGGnet with 19 layers. With the use of 3×3 kernels and a padding of single 0's, the same kernel size and thus multiple convolution layers one after the other can be used without size adjustments. In sum: three convolution layers with 64 kernels, a maxpool layer, two more convolution layers with 128 kernels, a maxpool layer, four convolution layers with 256 kernels, a maxpool layer again and twice four convolution layers with 512 kernels and a maxpool layer at the end. The ending part consists of three FC layers and a Softmax for a 1000 classification vector. Image taken from Chang, 2016 [5].

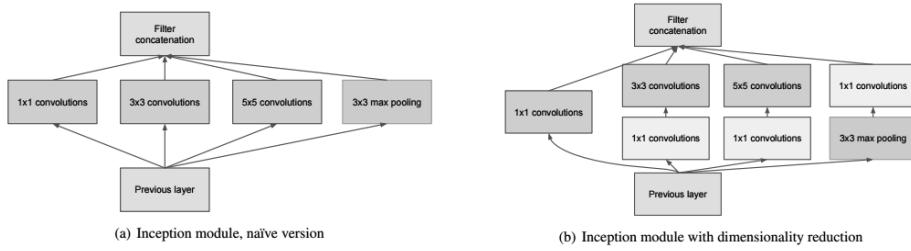


Figure 3.10: The inception modules. On the left (a) the naïve approach which has too many parameters. On the right (b), instead, the inception module with dimensionality reduction with 1×1 convolution layers. Images taken from Szegedy et al., 2015 IEEE [62].

Every CNN architecture of the previous sections had its layers built sequentially. Szegedy proposed calculating/executing some layers in parallel. The naïve approach (the authors' term for it) involved calculating three convolution layers (1×1 , 3×3 and 5×5 kernel sizes) and a max pooling layer from the same input and then linking everything together as a single output (see Figure 3.10a). The idea of concatenating those convolution-layer filter sizes was suggested by the fact that every filter picks different features from its inputs; using them all together helps the network pick all the features from the same input.

A reduction in computation resources was one of the main purposes of this network; thus, an idea different from that initial naïve approach had to be used. A convolution layer with 1×1 receptive fields solved this issue. It was used as dimensionality reduction for the convolution layers and hence the depth of each layer was diminished (see Figure 3.10b). GoogLeNet, in its last version has cut down the number of parameters to only a twelfth of the VGGnet, boosting the speed of training.

Like the previous models, GoogLeNet was trained with SGD with a momentum factor of 0.9, a decresing learning rate factor of 4% for every 8 epochs, a cropping approach for data augmentation as described in Figure 3.11 and a dropout method with 70% of dropped units. After the ILSVRC

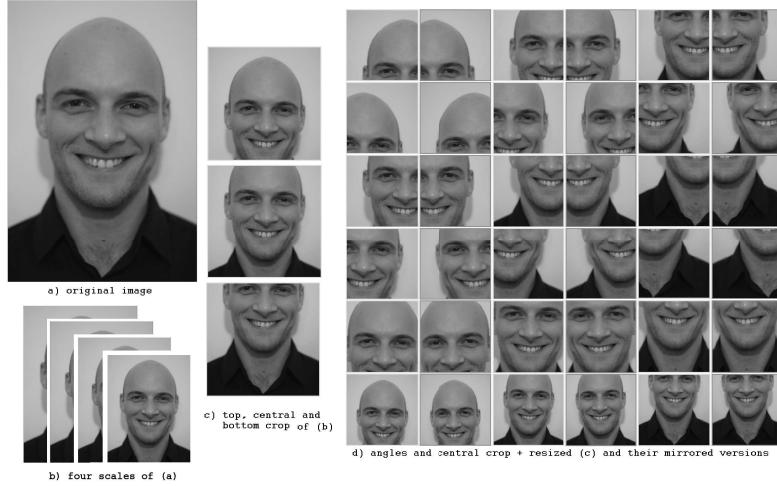


Figure 3.11: Cropping approach used in GoogLeNet. Top left, the original image (a). Under that (b), four scales of the original image, where the smallest border is set to: 256, 288, 320 and 352. From every scale, the left, middle and right squares are taken (top, middle and bottom squares for a portrait image, as in this case (c)). In the last step (d), from every square, the angles and the middle parts are taken (with the size of 224×224), and the original image resized to the same size as the others (shown in the first, third and fifth columns); additionally, every image is mirrored (shown in the second, fourth and last columns) for a total of 144 crops per input image.

competition submission of the model, Szegedy et al., discovered that a random cropping approach (where the scales and the aspect ratio are in a range of [8%, 100%] and [76%, 133%] respectively) and a photometric distortion (where the brightness, contrast and color are manipulated in a range [50%, 150%] as in Howard, 2013 [33]) is the best approach so far.

The architecture will not be shown or accurately described in the present work, because of its size. The authors state that the network has only 22 layers, counting the inception modules as only 1 layer. The number of layers actually exceeds 100 layers; hence, it is the deepest CNN architecture known for that period.

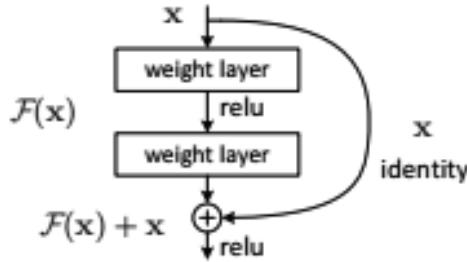


Figure 3.12: Residual learning. The x output is forwarded by identity mapping to the input of a layer in a deeper depth. Image taken from He et al. [27].

3.2.5 Residual Network (ResNet)

Another year passed after Simonyan et al. [58] and Szegedy et al. [63]; 2015 brought another improvement in CNN architectures. He et al. [29] presented a novel idea that outclassed every other CNN in image classification. The stated results of the 2015 ILSVRC classification benchmark were: a human-level-beating score of 3.57% top-5 error (humans are said to score between 5% and 10%) and the deepest CNN architecture ever developed, with 152 layers.

As the name suggests, the network is based on residual learning. The idea of that is to introduce a 'shortcut' that forwards the output of layer 'a' to the input of a layer 'b' that does not immediately follow 'a' (see Figure 3.12). Those forwarded activation maps are then summed with the dedicated input to layer 'b'. Considering x as an input to a layer, $F(x)$ the output of that layer, then the input to a new layer will be

$$F(x) + x, \quad (3.4)$$

when x and $F(x)$ have the same output the calculation is straightforward. If the sizes of the two parameters do not match, an identity map with zero padding is used to match the height and width of the activation map, and 1×1 convolutions are used to match the depth.

The image preprocessing and training process of ResNet is not substantially different from that of the previously discussed networks. A random multiscale image augmentation is used (shorter border of image in the range of [256, 480].) Crops of size 224×224 are randomly sampled, with or without horizontal flipping. A per-pixel mean subtraction and color perturbation are applied, as used in Krizhevsky, 2012 [42]. An SGD with a batch of 256, a weight decay of 0.0001 and a momentum of 0.9 are used for training; BN layers over DropOut technique are used to solve the problem of overfitting. The weights are initialized with the technique in He et al., 2015 [28] which slightly differ from the Xavier initialization due to the non-zero mean of the ReLU activation function.

The architecture follows the idea of the VGGnet network, but with more layers. For layers that have the same activation map size, the number of kernels remains unchanged, whereas when the feature map size is halved the kernel number is doubled to maintain the computational complexity. All the convolution layers use kernel sized 3×3 , except the first one which uses kernel sized 7×7 . The architecture is structured as follows: convolution layer (64 kernels), pooling layer, 6 convolution layers (64 kernels), 8 convolution layers (128 kernels), 12 convolution layers (256 kernels), 6 convolution layers (512 kernels), an average pool and the last fully connected layer for classification of 1000 categories. Every first convolution layer of each group of layers uses a stride of 2 to diminish the activation map height and width. After the first pooling layer, residual learning is introduced and is used for every two layers.

3.2.6 Dense Network (DenseNet)

After the advent of ResNet and its concept of identity mapping between non-sequential layers, an idea of concatenating more convolution-layer activation maps together was proposed by Huang & Liu, 2016 [34], called DenseNet. According to the authors, the DenseNet can achieve even bigger depth than ResNet with fewer parameters and a better validation error compared to the previously discussed ResNet of 152 layers against the ILSVRC benchmark.

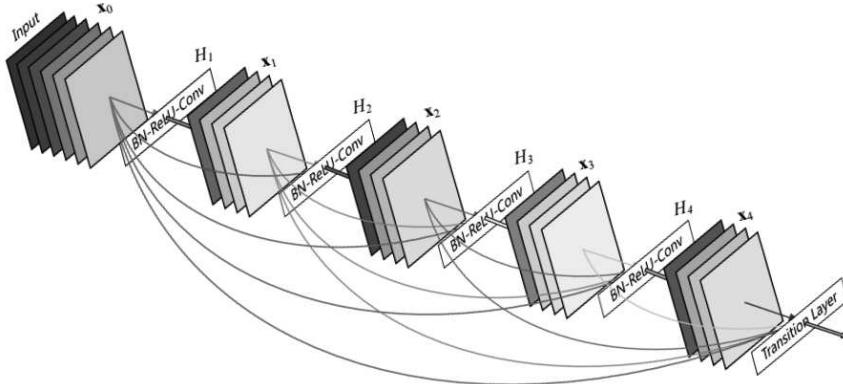


Figure 3.13: DenseNet dense blocks. The output of every layer are forwarded to every BN-ReLU-Conv layer as inputs to all other layers. Image taken from Huang & Liu, 2016 [34].

DenseNets (named after the dense connection among layers) embodies the idea that every activation map of every layer is connected with every layer that follows, given activation maps of the same size. Due to size constraints, groups of network layers called dense blocks (see Figure 3.13) are used to join together the layers with same-size activation maps (height and width). Between each group of activation maps and the subsequent groups there is a layer where BN, ReLU and a convolution layer (3×3 kernels) are used to address normalization, activation and activation map size. The output of every dense block is then forwarded to a transition layer that consists of a BN layer, a convolution layer (1×1 kernels) and an average pooling to diminish and normalize the information volume of the network. Dense blocks and transitional layers are then stacked until they reach a depth of hundreds of layers. One point that needs to be addressed with deeper architectures is the number of parameters that slow down the training process (backward pass) and the inference (forward pass) of the network. For that, a growth constant k is used that limits the number of activation maps that are forwarded among the dense-block layers (k 's used in the network trials are 16, 24, 32, 40 and 48).

Adding k activation maps after each layer in a dense block means that the last layer would have lk activation maps (where l is the number of layers in a dense block). To diminish the number of activation maps, the 1×1 convolution layers are limited to the maximum length of $4k$ activation maps. The problem of too many activation maps still persists for the input of the following dense blocks. A compression factor Θ is added at the end of every dense block in a range of $0,1$ and only $\Theta 4k$ activation maps are then forwarded to the next dense block through the transition layer.

The architecture used for ILSVRC benchmark has 4 dense blocks; the first convolution layer (kernel size of 7×7 with stride 2) creates $2k$ activation maps forwarded to a max pooling layer (with a 3×3 pool and stride 2). The activation maps (of size 56×56) are the input to 4 consecutive dense blocks and transitional layers. The last two layers are then an average pool (pool size of 7×7) with the output forwarded to a fully connected layer for classification of 1000 categories (for ILSVRC benchmark).

This network is the last of the CNN architectures that are discussed in this work. Only one of the architectures described in this chapter was the basis for the coding part of the present image classification project. The second part of the project consisted in creating an architecture that defines what images represent. Recurrent neural networks (RNNs) are the newest field in ANNs that best suited the latter purpose.

Chapter 4

Recurrent Neural Networks

As the name suggests recurrent neural networks (RNNs) differ from CNN networks for a recurrent connection that forwards information back to the same neuron or neurons in the same layer. This means that RNNs can have cycles and are thus not acyclic graphs like CNNs.

The simplest idea of an RNN is the Simple Recurrent Network (SRN) depicted in Figure 4.1 (a). As it can be seen, x is the input to a hidden node h that has two output connections. One is a weighted connection that goes to output o , while the second one returns back into the same neuron. The second connection is called feedback connection, due to the amount of information passed at each time step. Notwithstanding its unusual shape, the SRN unfolds into a perfectly understandable network as in Figure 4.1 (b) where U , V and W are the weights for every input. The learning process of RNNs, known as backpropagation through time (BPTT), is similar to the well-known backpropagation algorithm of Section 2.2.3, but with an additional time constraint. Every node uses the same set of parameters, so computing a backpropagation at time step t means that the derivatives need to be calculated for every time step before and equal to t in order to update the parameters. For example, the following equation expresses the calculation of a hidden state at time step t :

$$h_t = f(V * h_{t-1} + U * x_t), \quad (4.1)$$

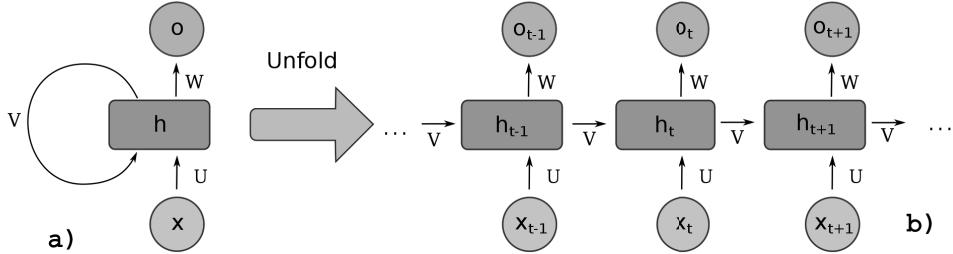


Figure 4.1: Simple RNN with its compacted version (a) and its unfolded version (b). The inputs x are represented in green, the hidden states h are represented by blue rectangles, the outputs o of each layer are represented in pink and U , V and W represent the input, hidden and output weight matrices respectively. Image taken from Wikimedia Commons [12].

where V and U are randomly initialized weight matrices for the feedback connection and the input respectively. h_{t-1} is the hidden state of the previous time step. o_t can be calculated as a Softmax of $W * h_t$ and is the output of the RNN at time step t . $f(\cdot)$ is one of the many choices in activation functions that were already explained in Section 2.1. The above equation constitutes a step in the forward pass.

For BPTT, every partial derivative of every time step with respect to every parameter in the network needs to be calculated. But calculating derivatives with long time dependencies can lead to problems such as vanishing or exploding gradients (Pascanu, 2012 [53]). If one considers the partial derivative of an activation function such as the sigmoid or the hyperbolic tangent, one can see that its gradient with respect to x on its sides is equal to 0. If this gradient is part of a chain of gradient calculations, this means that every gradient after it will still be 0. For the exploding gradient, instead, the difficulties are simpler: a gradient clipping (that limits the maximum value of a gradient) can solve the problem.

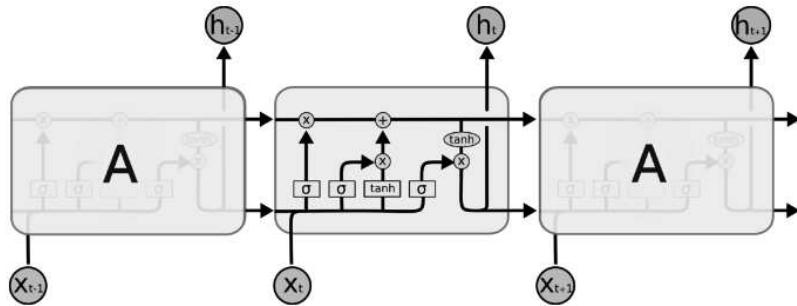


Figure 4.2: The image above shows how an LSTM hidden unit is defined and connected between two other hidden units. Image usage by permission of its author Christopher Olah [51].

4.1 RNN Architecture

The following section explains the architecture of Long Short Term Memory (LSTM) networks that addresses the vanishing gradient issue.

4.1.1 Long Short Term Memory (LSTM)

LSTMs (Figure 4.2) introduce a new structure that solves the time dependency problem and works well against vanishing gradients. A typical LSTM hidden cell is divided into 5 parts as described in Figure 4.3. The essence of its performance is the cell state connection (Figure 4.3a), which connects the input from the previous hidden cell to the next hidden cell with just an addition and a multiplication that can leave untouched, restrict or even block the flow of information of the previous hidden cell. Like the cell state connection, the forget gate (Figure 4.3b), the input gate (Figure 4.3c), the update section (Figure 4.3d) and the output gate (Figure 4.3e) work and learn together to output the best possible solution to the given problem. After the advent of LSTM, other variants were proposed. The 'peephole' connected variant has additional connections that join the state cell connection with every gates. In another variant the forget and input gate are joined together

so as to make the decisions symmetrical. Many other modifications were made through the last decades in Greff et al., 2015 [25], where 8 different variants were discussed and compared with mixed results.

LSTMs are widely used these days in the fields of speech recognition, handwriting recognition, image captioning (Vinyals et al., 2015 [66], Karpathy et al., 2015 [37], Donahue et al., 2015 [15], Xu et al., 2015 [72], Mao et al. 2015 [46]) and many others. The most influencial papers for the task of image captioning are discussed in the next section.

4.2 Application to Image Captioning

Below are discussed methods that use a CNN part and an RNN part linked together for generating image descriptions. In 2014 and 2015 different ways of addressing the task of image captioning were published; the most interesting are described in the following.

4.2.1 Multimodal RNN (m-RNN)

Mao et al., 2014 [47], proposed m-RNN, the first of its kind. It is divided into three main parts: a language model, an image model and a multimodal model. The language model learns how every word in the dictionary (which is built through the words acquired during the training phase) is connected together. The image model is a CNN and the multimodal part joins the previous two models in a one-layer representation. Mao et al.'s actual implementation consists of a deeper RNN than the one discussed in the introduction to RNNs in Chapter 4. Their idea introduces three more layers, two for word embedding and one multimodal layer with the following structure: input word, both embedding layers, recurrent layer, multimodal layer and output layer for the output word.

Those two embedding layers represent one of the main advantages of this network. They shrink the dense word-input vector (which is the size of the entire dictionary) to a word vector the size of 128 words. Secondly, the words

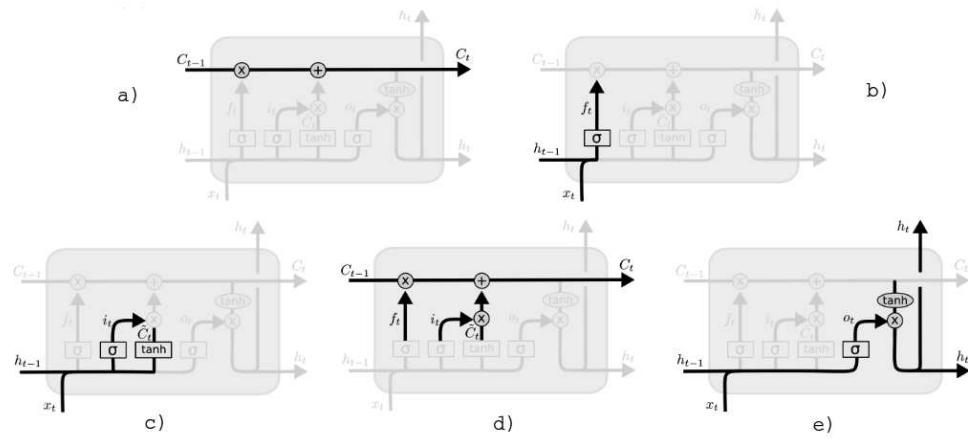


Figure 4.3: The images represent the concept on which the LSTM works. The cell state connection (a) has the role of carrying past information to the next hidden cell. The forget gate layer (b) uses a sigmoid function that outputs values in the range from 0 to 1, which are then multiplied by the cell state, and this operation determines whether past information is retained or not. The input gate layer (c) is split into a sigmoid function that decides which information needs to be updated and a hyperbolic tangent layer that creates a vector of new values. So the updated and the new values are then added to the cell state (d). The last image (e) represents the output gate where the output is filtered (sigmoid function) and rescaled in the range from -1 to 1 (hyperbolic tangent). Image usage by permission of its author Christopher Olah [51].

in the embedded vector are selected with the Euclidean distance: this means that only the words with the highest semantic value are selected.

The recurrent part has 256 dimensions and its hidden state $r(t)$ is calculated as:

$$r(t) = f_2(U_r r(t-1) + w(t)), \quad (4.2)$$

where f_2 is the ReLU activation function, U_r are the weights for the input and $w(t)$ is the input word at time step t . As shown by the equation 4.2, the value of $w(t)$ is added to the previous hidden state values and not multiplied by it (as in equation (4.1)). The use of ReLU turned out to be particularly profitable (low computational expenses, less prone to overfit the model), hence BPTT could be used without any adjustment for the recurrent layer depth.

The fourth building block of this architecture is the multimodal layer. It connects together the language model part (the second embedding layer and the recurrent layer) and the second-to-last layer of the AlexNet-architecture CNN. The multimodal output is then calculated as:

$$m(t) = g_2(V_w w(t) + V_r r(t) + V_i I), \quad (4.3)$$

where V are the weights for the respective layer outputs, I are the image features of AlexNet and g_2 is a scaled hyperbolic tangent $g_2 = 1.7159 \tanh(\frac{2}{3}x)$. The multimodal output is then forwarded to a Softmax layer to find the highest probability of the next word in the sentence.

The learning process with this architecture has been thought out well. The gradient of the backpropagation can update every weight from the language model (embedding and recurrent layer) to the image model (for the fine-tuning). The cost function of the model is an average log-likelihood based on the perplexity of the sentences with a regularization term as in

$$C = \frac{1}{N} \sum_{i=1}^N L \log_2 PPL(w_{1:L}^{(i)} | I^{(i)}) + \|\Theta_2\|^2, \quad (4.4)$$

where N is the number of words in the dictionary, PPL is the perplexity of

the sentence w given the image I and at the end there is the regularization term where Θ are the model parameters.

The model was evaluated against three different benchmarks (Flickr8k, Flickr30k and IAPR TC-12 benchmarks) with the BLEU score metric. The scores for IAPR TC-12 are 6.92 for perplexity, 39 for BLEU-1, 18 for BLEU-2 and 13 for BLEU-3. For Flickr8k dataset are 24 for perplexity, 58 for BLEU-1, 28 for BLEU-2 and 23 for BLEU-3. Against Flickr30k dataset for Perplexity, BLEU-1, BLEU-2 and BLEU-3 are 35, 55, 24 and 20 respectively shown in Table 4.1.

4.2.2 Google - Show and Tell

Google, after their impressive results in image classification with GoogLeNet, proposed an image captioning model called Show and Tell (Vinyals, 2015 [66]). The model consists of two parts: the CNN takes an image as input and forwards the output to the RNN. The recurrent part is built with LSTM cells, which are slightly simplified compared to the ones presented in Section 4.1.1. The LSTM cell input, output and forget gates are connected directly to the cell state connection (or update gate) with sigmoid functions. Additionally, the cell state has an update factor h that defines how much information of the previous state is retained with a hyperbolic tangent. The recurring connections are made with the output, which is connected with all the gates (input, output, forget and update) of the cell of the next time step.

Considering the unrolled LSTM network, the next three equations define the training procedure:

$$x_{-1} = CNN(I) \quad x_t = W_e S_t \quad p_{t+1} = LSTM(x_t). \quad (4.5)$$

The first equation represents the output from the CNN as input in the first block of LSTM cells. The second one represents a word in the ground truth sentence (starting and ending with two special strings: 'START' and 'END') at time step t multiplied by the word-embedding weights. The third equation represents the probability that the used word is suitable for that time step

and will be used in the loss function. The training phase aims to minimize the loss of the negative log-likelihood of the correct word for each time step as described in

$$C = - \sum_{t=1}^N \log p_t(S_t), \quad (4.6)$$

where N is the number of the words in the sentence, S_t is the word at time step t and p_t is the probability that the word is suitable. It is worth noting that the LSTM is trained for the next word after it has seen every word before that.

Differently from the previously discussed idea (m-RNN), the inference of this model was done in two ways: the first by sampling the first word according to the first probability p_1 and then continuously sampling all the words till one reaches the 'END' word or a limit length of the sentence; The second by using the beam search: for every time step the model retains k best sentences.

The model was tested on datasets PASCAL, Flickr8k, Flickr30k and SBU. The model was trained on the MS COCO dataset (Lin et al., 2014 [45]), which is the biggest dataset with the highest-quality image captions. The evaluation metric was not just BLEU as in Mao et al.'s, [47] previously discussed idea, but, due to BLEU's drawbacks, METEOR and CIDEr were also used. The results of BLEU-1 score for the PASCAL, Flickr8k, Flickr30k and SBU datasets, also reported in the paper (Vinyals, 2015 [66]), are 59, 66, 63, 28 respectively. The authors evaluated the model on the development set of the MS COCO dataset with additional metrics such BLEU-4, METEOR and CIDEr with the respective score of 27.7, 23.7 and 85.5.

4.2.3 Bidirectional RNN

Another idea, Karpathy et al., 2015 [37], which differs slightly from Mao et al., 2014 [47] and Vinyals et al., 2015 [66], is to create a combination of a deep neural network that associates portions of sentences with its location on the training image and a multimodal RNN that learns to create a sentence based

on the portions of sentence and the image regions where the portions belong. The approach of Karpathy et al., 2015 [37] differs in that the network learns on a different set of image-sentence pairs from the one that is originally given by the training datasets.

The original image and 19 best locations taken through a region-proposal CNN (Girshick, 2014 [19]) are forwarded to the CNN and transformed into a 4096-dimensional vector.

The sentence-generation model is a bidirectional RNN, which works in both directions of the sentence (from left to right and from right to left); thus it has two hidden layers, and every output word is chosen taking into account all the surrounding words. The BRNN model transforms each of the input words into the same dimensional space as that of the images.

A method was needed to compare the sentences with the words for the highest matching score. The authors proposed a score in the form of

$$S_{kl} = \sum_{t \in g_l} \sum_{i \in g_k} \max(0, v_i^T s_t), \quad (4.7)$$

where g_k and g_l are subsets of an input image k and subsets of an input sentence l . $v_i^T s_t$ is a dot product between the i -th region and t -th word and is considered the score amount. In this case the bigger the score amount the better the alignment between sentence l and image k . The above equation is then simplified so that for a given sentence word the region with the best score is taken:

$$S_{kl} = \sum_{t \in g_l} \max_{i \in g_k} v_i^T s_t. \quad (4.8)$$

The scores of single words for a given image mean that each specific region in an image is literally labeled with a word; thus, the network cannot create rich sentences with just these words. For that reason, a Markov Random Field is used which adds multiple corresponding words to the same image regions.

Over the original images and their training sentences and over smaller regions with their selected words a multimodal RNN is trained to generate

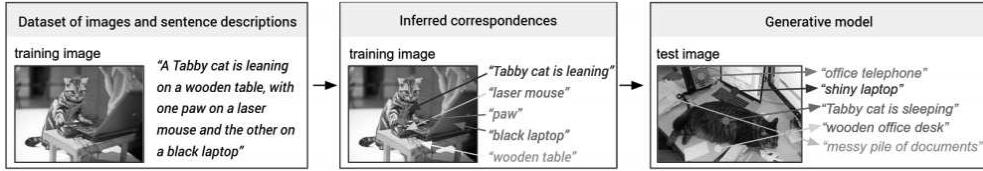


Figure 4.4: The first picture on the left represents the original training image with a training sentence. The middle picture represents the intermediate step where the regions of the image are connected with the best suitable words. The last picture is the result of the model on a test image. Image taken from Karpathy, 2015, IEEE [37].

new descriptions (see the image on the right in Figure 4.4). To rank the new generated sentences a beam search (with k factor 7) is used. The training uses SGD with a momentum factor of 0.9, DropOut regularization, a threshold for clipping the gradients and the RMSprop weight update rule.

The entire model was tested on Flickr8k, Flickr30k and MS COCO (Lin et al., 2014 [45]) datasets. It outmatched the mRNN model of Mao et al., 2014 [47] but it did not score better than Vinyals et al., 2015 [66] due to the less performing CNN. The results against the Flickr8k dataset are 57.9 for BLEU-1, 38.3 for BLEU-2, 24.5 for BLEU-3 and 16 for BLEU-4. The results for the other two datasets are summarized in Table 4.1.

4.2.4 Deep Multimodal Similarity Model

Fang et al., 2015 [18] proposed to create image captions similar to the one proposed by Karpathy et al., 2015 [37]. A pipeline model is used to join the language part and the visual part.

First, 1000 of the most common words are taken from the training captions while all the others are discarded. Then a Multiple Instance Learning (MIL) approach takes these words and a set of images. For every word, MIL takes as input sets of bounding boxes that are categorized as 'positive' (used for the training process) if the word is contained in the image description or

'negative' if not.

Instead of feeding the MIL approach with an original image, it is fed with the response map of a CNN. 144 crops are selected from the up-scaled response map (to 565 pixels on the shortest side) with the size of 224x224 (12 for every row and 12 for every column with a stride of 32). The result of this is a map of the original image where the regions that better fit the word have a higher probability.

To generate captions, a max-entropy (ME) language model (LM) estimates the highest probability of the next word based on all the previous words and on the words that have a high likelihood to the given words. The language model is trained by minimizing the log likelihood, i.e.,

$$C = \sum_{s=1}^S \sum_{l=1}^{\#(s)} \log Pr(w_l^{(s)} | w_{l-1}^{(s)}, \dots, w_1^{(s)}, < s >, V_{l-1}^{(s)}), \quad (4.9)$$

where s is the index of sentences, $\#(s)$ denotes the length of the sentence, w_l the word at time step l , $< s >$ the special word for the start of the sentence and V the ensemble of the words with the highest likelihood that need yet to be used in the sentence. For every image more sentences are generated. The sentences are generated word by word and only k best sentences are retained. Those k sentences are re-ranked using the MERT algorithm.

And for the last part of their project, the authors project they created a deep multimodal similarity model (DMSM) which is composed of two neural networks: one that learns the image-caption pairs and the second, a CNN, from which the features of the last FC layer are taken. Three additional FC layers are added on top of the CNN-extracted maps to make them of the same size as the first neural network. Then the best image captions are taken based on the cosine similarity between those two features' vectors.

The authors tested their model on MS COCO (Lin et al., 2014 [45]) dataset. The official results are 69.5, 29.1, 33 and 92.5 for BLEU-1, BLEU-4, METEOR and CIDEr metrics respectively.

Because all these works of image captioning were developed in the same period but did not use the same datasets, their results cannot be compared.

Table 4.1: Table summarizing the results on the image caption models on datasets Flickr30K and MS COCO for caption generation.

Models	Flickr30K				MS COCO					
	B-1	B-2	B-3	B-4	B-1	B-2	B-3	B-4	M	C
Mao et al.	55	24	20	-	-	-	-	-	-	-
Vinyals et al.	66	42	28	18	67	46	33	25	-	-
Karpathy et al.	57	37	24	16	63	45	32	23	20	66
Fang et al.	-	-	-	-	-	-	-	21	21	-

The results that can be compared are raked up in Table 4.1; as stated by the MS COCO benchmark the Google NIC model (Vinyals et al., 2015 [66]) outperforms every other model.

4.2.5 Captioning metrics

All those networks discussed in the previous section usually trains on image-caption pairs where they learn the features needed for generating new captions. The caption quality is evaluated by several measures like BLEU (Papineni et al., 2002 [52]), METEOR (Lavie et al., 2007 [43]) and CiDER (Vedantam et al., 2015 [65]).

BLEU (Papineni et al., 2002 [52]) is a metric used in machine translation. Considering a machine translated sentence and some reference sentences, the BLEU score is calculated by counting n-grams of sentences that match the same n-grams in the reference sentences. Smaller n-grams are penalized over the bigger one because they can just repeat words without semantic value, the bigger one, instead, tend to emphasize the fluency of the translation.

METEOR (Lavie et al., 2007 [43]) was developed for addressing the BLEU weaknesses. It calculates how many unigrams are matching in both, the machine generated and reference sentence. It maps unigrams of the generated sentence with the unigrams of the reference sentence. Giving higher score to those matches with least possible unigrams position crossings (called unigram

mapping crosses), thus generated sentences which have the same order of the words are classified as better. A lower score is assigned to unigrams that are mapped to its synonyms, plurals or singulars matches. The final score, additionally, is penalized if unigrams cannot be joined into n-grams (where two unigrams are mapped in sequence without map crossing can be joined into a bigram the same procedure is used for n-grams).

CiDER (Vedantam et al., 2015 [65]) is a slightly different metric from the previous two (BLEU and METEOR). It was created for evaluating image descriptions. The calculation of the score is done by weighting the value of each n-gram in the sentences (generated and reference sentences). The n-grams that are commonly used in all the images sentences in the training dataset are given a lower weight, conversely, the words that are used frequently in reference sentences are given a higher weight. The score is the finalized with the calculation of the cosine distance between the generated and reference sentences.

Chapter 5

Image-Based Food Recognition

This chapter gives an overview of our approach to address the automated food recognition problem. Many works (Kagaya et al., 2014 [36], Yanai & Kawano, 2015 [73], Lu, 2016 [26], Ciocca et al., 2017 [6]) used CNNs for food categorization, and the idea that is commonly stated by the authors is that food images are typically harder to classify because of their fine-grained properties. In the next sections the method and the implementation details are discussed.

5.1 Method

The objective of this thesis is to implement a food image recognition network, an image caption network and an image caption with region-proposal network using neural networks. Three networks will be used, for addressing these tasks. First, a CNN for the visual model that can learn the features needed to recognize food images, second, an RNN for the captioning model that can learn how to create descriptive sentences, and third, a region-proposal method that uses the same CNN for food recognition.

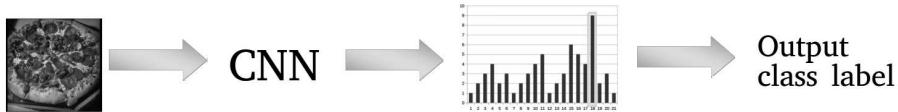


Figure 5.1: An input image is forwarded to the CNN. The last layer of the CNN shows the probabilities of each image class. The class with the highest probability is picked and printed.

5.1.1 Visual Model Method

The CNN architecture—GoogLeNet—was not developed from scratch, but was chosen from one of the previously discussed architectures. In fact, ResNet or DenseNet could have been chosen, due to their better performance in classification, but one year later the publication of GoogLeNet, Szegedy et al., 2015 [64], re-thought GoogLeNet’s architecture and made some changes¹. The new adjusted model (named Inception-v3) achieved 3.58% top-5 error against the ILSVRC benchmark, which is the same as the best score held by ResNets. One of the positive aspects of CNNs (and in general of ANNs) is transfer learning, where the learned weights of a trained architecture can be re-used for a different application. A GoogLeNet-Inception-v3 (Szegedy, 2016 [64]) pre-trained model (on ILSVRC benchmark dataset) was chosen. Due to the diversity between the datasets, a fine-tuning was needed. The fine-tuning was done by resetting the two lowest FC layers (as described in Figure 5.2) and training them from scratch on our new food dataset. Figure 5.1 schematically shows how this model works.

5.1.2 Captioning Model Method

The choice of the RNN part was again pointed towards the Google model. Google’s Show and Tell (Vinyals, 2015 [66]) is not only the simplest of the models discussed in Section 4.2, but has also outperformed every other model

¹Convolution layers kernels were changed with 3×3 , $n \times 1$ and $1 \times n$ sized kernels.

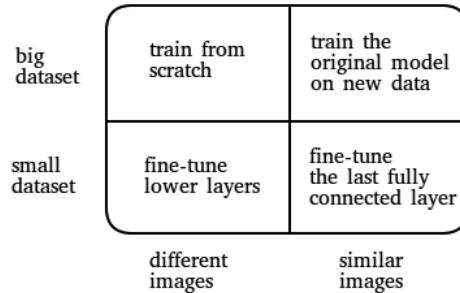


Figure 5.2: The image shows what needs to be done when fine-tuning a pre-trained model. If the new dataset is small and consist of similar images, only the last FC layer needs to be re-trained completely. In the case of different images, two or more FC layers should be re-trained. When the dataset is big enough and with similar images, it does not need to be fine-tuned, only train more on the new data. In cases where the new dataset is big and of a different kind, then the model needs to be trained from scratch.

on the MS COCO benchmark. As stated in Vinyals, 2015 [66] the model uses a CNN for the image part and an LSTM-RNN for the captioning part. Here our fine-tuned CNN model was used as input for the LSTM-RNN and the entire network was trained on image-sentence pairs. The last feature layer of the CNN forwards the information of the input image directly to the first LSTM cell of the RNN part that generates a word that will be forwarded to the next cell until a complete sentence is produced. A block scheme is given in Figure 5.3 which schematically shows how the part are connected together.

5.1.3 Region Proposal Method

Additionally to the previous two methods, a region-proposal method for summarizing food objects in images was implemented. In this method, 100 random regions of size in range from 45% to 100% of the original image are selected and forwarded to the newly food-trained visual model for classifica-

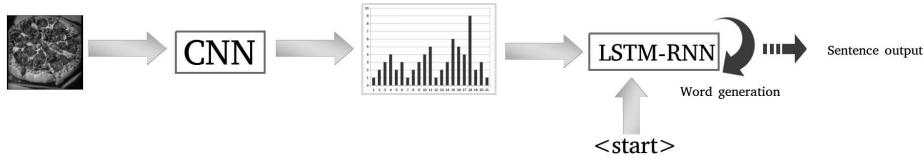


Figure 5.3: An input image is forwarded to the CNN. The last hidden layer of the visual model (a 2048-dimensional vector) is forwarded to the first layer of the LSTM network. The first LSTM cell get an additional word *< Start >* which is used to generate the first word of the caption sentence. Every generated word is the input to the next cell, too. This cycle is repeated till the word *< end >* is produced.

tion. The images that are classified with an accuracy over 70% are retained, the others are dropped. The retained categories with their probabilities are printed. Algorithm 2 defines the steps needed for classifying and caption an image with the region-proposal method.

5.2 Implementation

Our food recognizer and descriptor was implemented on a machine with OS Linux 16.04 LTS 64-bit, python version 3.4 and TensorFlow 1.0.1 with GPU support installed. TensorFlow, which is “an interface for expressing machine learning algorithms, and an implementation for executing such algorithms[. . . ,]” [1] aided the implementation of CNN models through an additional library (named Slim). TensorFlow is an open-source software library that uses computational graphs where the nodes represent mathematical operations and the edges are multidimensional tensors. Due to this unconventional structure, it can hold a specific type of input file only, called TFRecords.

Algorithm 2 In the algorithm are listed the most important steps of our region-proposal method. Before the for loop the image and the network are loaded. In the for loop 100 crops from the input image are created and classified. If the classification of each crop is high enough (over 70% of classification accuracy) the classification data are stored and printed in the end.

```
image_tensor ← decode_image("input_image.jpg")
visual_model ← get_network("Inception-v3")
for i = 1 to i = 100 do
    bbox ← create_distorted_bounding_box()
    cropped_images[i] ← crop(bbox, image_tensor)
    logits_layer ← visual_model(cropped_images[i])
    best_id ← argmax(logits_layer)
    probability ← max(softmax(logits_layer))
    if probability > 70 then
        probabilities[j] ← probability
        classes[j] ← id_to_class(best_id)
        best_images[j] ← cropped_images[i]
        j ← j + 1
    end if
end for
probabilities, classes, best_images ← fix_doubles(probabilities, classes,
best_images)
print probabilities, classes
```

5.3 Visual Model Implementation

The first requirement to be fulfilled was to convert the dataset to the compatible format. This was done with a helper function (*convert_data.py*) that takes two inputs: the name of the dataset, with which the dataset's pertinent information is obtained (such as split sizes, number of classes, the information that needs to be stored in the TFRecords etc..) and the image directory, that holds the images split in sub-directories named after the specific category name. The outputs are two groups of TFRecord files, one for the training set and one for the test set. Whether an image belongs in the training set or not is defined by the random shuffling of images in the sub-directories and by picking as many images as defined in the dataset's split size information. The remaining images are part of the test set. Once the dataset is in the right format, the training script can be run.

Before training the model, some parameters need to be defined and some methods implemented. Thanks to TensorFlow's Slim library, all of the weight update methods that were explained in Section 2.2.3 are implemented and one could be picked. The other parameters are divided into 5 categories: (a) flags used for training management, (b) optimization flags, (c) learning-rate flags, (d) dataset flags and (e) fine-tuning flags. (a) defines the number of steps needed for saving the training results (called checkpoints), the path to the images folder and additional parameters for defining how many clones of the same model are shared over the hardware. (b) defines the optimizer that needs to be used, the weight decay and all the other parameters used by each weight-update method. (c) defines the initial value, the type of learning rate and its parameters. (d) gives the name, split and the path to the directory of the dataset, and selects the model name and type of preprocessing. The last flags (e) define where to load the pre-trained models, where to store the new ones and which layer is to be re-trained from scratch. A main function loads all the information based on the given flags. The dataset and the network are loaded, then a dataset provider loads images with their ground truth labels. Over the picked images the preprocessing is done and a batch of preprocessed

images is created. The labels are stored as a one-hot vector and the batch of training images only is prefetched. The loss function is defined as cross entropy with a softmax layer and the optimizer is always checked for its learning rate. The gradient updates are created and the training is started with the *slim.learning.train* function. The length of the training process is defined by the maximum number of global steps the training should undergo, or, if no such number is defined, the training can be stopped manually, in which case the last checkpoint saved stores the latest version of the trained model.

In the middle or at the end of the training process, an evaluation script (*eval_image_classifier*) can be run to check the learning process of the model. In this script as in the training script, flags such as batch size, checkpoint path, evaluation directory path (for storing the results), dataset name, split name of the dataset and model name need to be defined. The next step is to load all the necessary data from the dataset (evaluation images, labels and split size) and the new checkpoint of the model. A batch of non-processed images is then used and forwarded through the network. For every image, the last layer (softmax layer for probabilities) is checked. If the index of the highest probability of the softmax vector is the same as the ground truth label number, then the prediction is correct. Differently from the training script, the metrics for evaluation are defined (accuracy as top-1 and recall-5) and are printed at the end of the evaluation process.

5.4 Captioning Model Implementation

For the image-captioning part, the procedure is similar to the visual model. First an MS COCO compatible dataset of image-caption pairs is created with an implemented GUI (*Caption_GUI.py*). Then, it is transformed into the train and validation split of a TFRecord-compatible format with a python script (*build_mscoco_data*). The script needs six input parameters: (a) the train image directory, (b) the validation image directory, (c) the training

caption file, (d) the validation caption file, (e) an output directory and (f) an output file for the word count. Additionally to the TFRecord files, the script outputs a text file (*word_counts*) that contains the repetitions of each word in the captions: a word that is repeated less than 4 times is classified as unknown.

With the data set, the training script (*train*) is run. The mandatory input parameters are the training TFRecord files, the pre-trained visual model and the directory of the training images. The first thing that needs to be done is building the Show and Tell model, after which the learning rate and the training operands (hyperparameters) are set, the pre-trained visual model is loaded and the training is started (with *tf.contrib.slim.learning.train* function).

The evaluation script (*evaluate*) in this case will not show the result of a generated sentence, but will calculate the perplexity over the test images and their captions. The inputs to the captioning model are the test TFRecord file, the directory of the pre-trained visual model and the directory of the test images. For the generation of captions, the inference script is used (*run_inference*). It builds the model, restores the parameters and generates the captions for the given image.

5.5 Region Proposal Model Implementation

The region-proposal model is a simple summarization of all the recognized object in the image. For that, the input image is loaded and converted to a compatible TensorFlow format (with *tf.image.decode_jpeg* function). The dataset information and the visual model are loaded soon after. With a cropping function (*tf.image.sample_distorted_bounding_box*) 100 random regions are selected from the loaded image. The crops are resized to the network input size ($299 \times 299 \times 3$) and forwarded through the network. The predictions are checked and if the maximum value is higher than 70% then the id of that value, the image and the value are stored. If there are two

categories with different values, only the highest is retained. At the end all the accuracies and class names are printed.

Chapter 6

Experimental Evaluation

This chapter gives an overview of the data used and the results of our three approaches: food classification, food captioning and food captioning by region proposal. In the following sections the dataset, the preprocessing approach, the performance measures and the different testing results are shown and discussed.

6.1 Food Image Dataset

Different food datasets are used by different authors. The most used are UEC-FOOD-100 (Matsuda et al., 2012 [48]), UEC-FOOD-256 (Kawano et al., 2014 [38]), FOOD-101 (Bossard et al., 2014 [3]) and UniMib2016 (Ciocca et al., 2017 [6]).

UEC-FOOD-100 and its larger variant UEC-FOOD-256 are datasets of images of common Japanese dishes, divided into 100 and 256 categories respectively. UniMib2016 has 1000 canteen tray images divided into 72 classes. The FOOD-101 dataset is the biggest food dataset created, with 101 food categories (Japanese and Korean foods), 1000 images per category.

This thesis entails the idea of creating a dataset of 'simple' foods. The first step was to find a larger dataset and to extract the specific types of images. OpenImages dataset (Krasin [41]) is one such dataset, with roughly

10 million images spanning 6000 categories. Fortunately, there is no need to download all the images. Google BigQuery online platform is a RestFul API with which the OpenImages dataset can be checked and SQL queries can be written to select specific images.

The OpenImages dataset is different from the datasets mentioned at the beginning of this section. In fact, it does not hold images but just the file-names, categories and URLs where the images reside. Without going into too much detail about the dataset composition, SQL queries were used to choose images that were labeled by humans as foods (this means that there is a 100% possibility that they were labeled correct). Additionally to the 'food' label, every food has other labels too (i.e. 'carrot', 'banana', 'meat', etc). Those additional labels were sorted by repetition numbers and those sub-categories that had more than 50 images were chosen.

For every food sub-category a *.csv* file was created that contained the images' metadata. With a python script (*downloader_from_csv.py*) the images were downloaded and stored in their dedicated folders. However, a problem arose. If one image was labeled with two chosen categories (example: an image labeled both ice_cream and chocolate) the duplicates needed to be either deleted or re-sorted manually (*dups_finder.py*). For every duplicated image, it was checked as to its suitability (left untouched) or unsuitability (eliminated) to the category. If the image suited two specific categories then it was deleted. The first version of the new dataset contained roughly 8000 images spread over 45 categories.

In a preliminary experiment, it was observed that the network performance fluctuated significantly (accuracy score between 50-65% on a test set with a 70%-30% train-test split). The problem was due to the unbalanced sorting of images per category in the dataset. In fact, some classes had 600 or more images while others had roughly 50. This difference causes the score on the test set to be unrealistic.

Thus, an effort was made to balance the dataset with 50 images per category, but after another training attempt no optimal scores were shown. The



Figure 6.1: Images taken from the categories of the new food dataset. The categories are left to right, first row: bread, broccoli, burger, apple, pineapple, cabbages, ice-cream. Second row: pasta, sausage, pineapple, grapes, pizza, rice, salad. Third row: soup, carrots, cookies, cheese, pie, meat and fish.

problem was discovered to be two-fold: first, a lot of images did not represent the categories to which they belonged (for example: an image labeled 'apple' represented a market with lots of people and just a small part of the image had apples), and second, specific categories were too similar to be labeled as different (for example limes and lemons). Thus a further modification was applied.

The third attempt for creating the dataset entailed: (a) removing bad representations of images, (b) joining/discardng almost-identical categories and (c) starting with a smaller dataset. The starting point was a six-category dataset and thirty images per category. All the new images that needed to be added were selected from 'www.flickr.com' or 'www.google.com' with creative commons attribution licenses. The testing of the network was optimal, with an accuracy of 100%. This result is expected on such dataset. The next step was aimed at enlarging the dataset. The last version of the dataset reached the maximum number of 21 categories and 70 images per category for a total number of 1470 images (see Figure 6.1). Once the final dataset's version was built, the preprocessing was chosen.

6.2 Data Augmentation

Our preliminary study showed that the dataset was still too small for the number of parameters in our architecture, which resulted in overfitting. As already discussed in Section 2.2.4, there are two solutions to overfitting. The possibility of a bigger dataset could not be taken into consideration due to time constraints (checking every image that fit the categories), therefore the data augmentation approach was used. Different authors proposed their approaches (see Section 3.2) where the most elaborate was GoogLeNet’s, which was chosen for the new dataset.

The data augmentation consisted in (a) random cropping of the original image in a range from 8% to 100%, (b) bilinear-interpolation rescaling of the cropped image, (c) random horizontal flipping, (d) brightness and saturation perturbation and (e) element-wise addition and multiplication (0.5 and 2 respectively).

Aside from such processing, all the images were externally resized to the longest border fixed to 300 pixels, maintaining the aspect ratio. This was done to diminish the size of the dataset from approximately 2GB to 85MB with 23 times fewer data.

6.3 Food Caption Dataset

Like every other model, the captioning model too needs data for training, this time in the form of sentences. For trial purposes, two categories (apple and bread) were chosen from the dataset. For those two categories and for each respective image, 5 training captions were written for a total of 750 sentences. A GUI caption helper was implemented in python (*Caption_GUI.py*), which aided the storing of image-caption pairs in the right MS COCO format. The captions and their images were split into a test set (20 image-caption pairs) and a training set (50 image-caption pairs).

After the set up, a preprocessing of the training data was needed. A dictionary of all the used words was created that assigned an integer-valued

ID to every word. If a word was repeated less than four times, it was categorized with an unknown character. After that, the image-caption pairs were transformed into TFRecord format so they can be used in the model training process through TensorFlow.

The captioning model learned some good features on those image-caption pairs but the idea of food captioning does not stop here. The generation of captions was supposed to work when objects of different categories were in the image. Unfortunately, testing the model on an image where an apple and a piece of bread were present returned a generated caption where just one of the two objects prevailed. There is an assumption on these non-working examples: the captioning model do not recognized the relations between different categories in the training images and must be trained to do so. On the other hand creating (and finding) such examples would be time consuming. Consider 21 categories where each category needs to have a training image with a relation to each of the other categories. This means 420 combinations and 2100 more training captions for just one degree of relation. Considering 2 or more degrees, the time expense would grow exponentially.

6.4 Performance Measures

Through this work different performance measures have been mentioned. In the next sections the measures used to quantify the score of our three approaches will be explained.

6.4.1 Visual Model Measures

The visual model scores are measured with top-1 and top-5 accuracies. The accuracies are calculated over the validation and test sets. When an image goes through the network, the network tries to predict the category to which the image belong. If the predicted category and the ground truth are the same, then the network has predicted the image correctly. The top-1 accuracy is calculated dividing the number of correctly predicted images by the

total number of images that went through the network and multiplying the quotient by 100 (for the percentage).

For the top-5 accuracy, the calculation is the same as in the top-1, but with the distinction that the predicted outputs are considered correct if one of the five highest predicted categories are equal to the ground truth. That said, the higher the accuracy the better the model prediction.

6.4.2 Captioning Model Measures

For measuring the language model the perplexity metric is used. It is calculated as the geometric average of the inverse probability of the words on the test data, i.e.,

$$ppl = \prod_{i=1} (P(w_i|h(w_i))), \quad (6.1)$$

where $h(w_i) = w_1, w_2, \dots, w_{i-1}$, P is the probability of a word w_i given all the previous words $h(w_i)$. When the perplexity metric is used as a measure of models, it shows how close the model is compared to the test data.

6.4.3 Region Proposal Model Measures

The measure used with the region-proposal method is composed of two metrics. The first is the accuracy of the visual model as explained in Section 6.4.1, which is used to classify regions of images. The second is a measure that classifies how many objects were recognized in the image (for example: an image with four objects where only three are classified correctly has a score of 75%).

6.5 Results

The following sections describe the results of image classification of the visual model (CNN), the caption generation of the captioning model (CNN + LSTM-RNN) and the summarization with the region-proposal model (Regions + CNN).

6.5.1 Food Classification Results

The GoogLeNet-Inception-v3 model [64] pre-trained on the ILSVRC dataset (Russakovsky, 2015 [56]) was applied to establish a baseline for the recognition performance. The difference in the number of classes between the food image dataset (defined in Section 6.1) and the ILSVRC dataset demanded that the last two hidden layers of the model were be re-learned from scratch. Testing the model, as soon as the last layers were reset, gave a poor test accuracy, which was expected. After that, the training was started on the food image dataset with the following hyperparameters: SGD with batch size of 64 images, an ADAM optimizer, a hand-tuned exponential learning rate decay and a weight decay regularization of 0.00004. As soon as the value of the total loss function halted, the training was stopped and the model was evaluated, achieving 82.4% score top-1 accuracy and 98% top-5 accuracy on the test set.

In the evaluation process, a confusion matrix was built, which is shown in Figure 6.2. The confusion matrix shows how well certain categories are learned and with which probability those images are predicted. Categories like pineapple, apple, banana, burger, cheese, grapes, ice_cream, pizza and rice are optimally learned and the evaluation shows high prediction accuracy. Other categories such as carrot, cookie, meat, pasta, pie, salad and sausage get confused with other categories.

Figures 6.3 and 6.4 depict some of the mislabeled images. Some of those images, in fact, resemble the predicted categories by the shape, positioning or colors of the objects. There are some images that unfortunately do not carry such similarities (for example the banana image in column one, row five in Figure 6.3 or the meat image in column three, row one in Figure 6.4). This is probably due to the preprocessing for the learning process that may take small pieces of images and apply color-brightness perturbation to them, which can change the original color of the object.

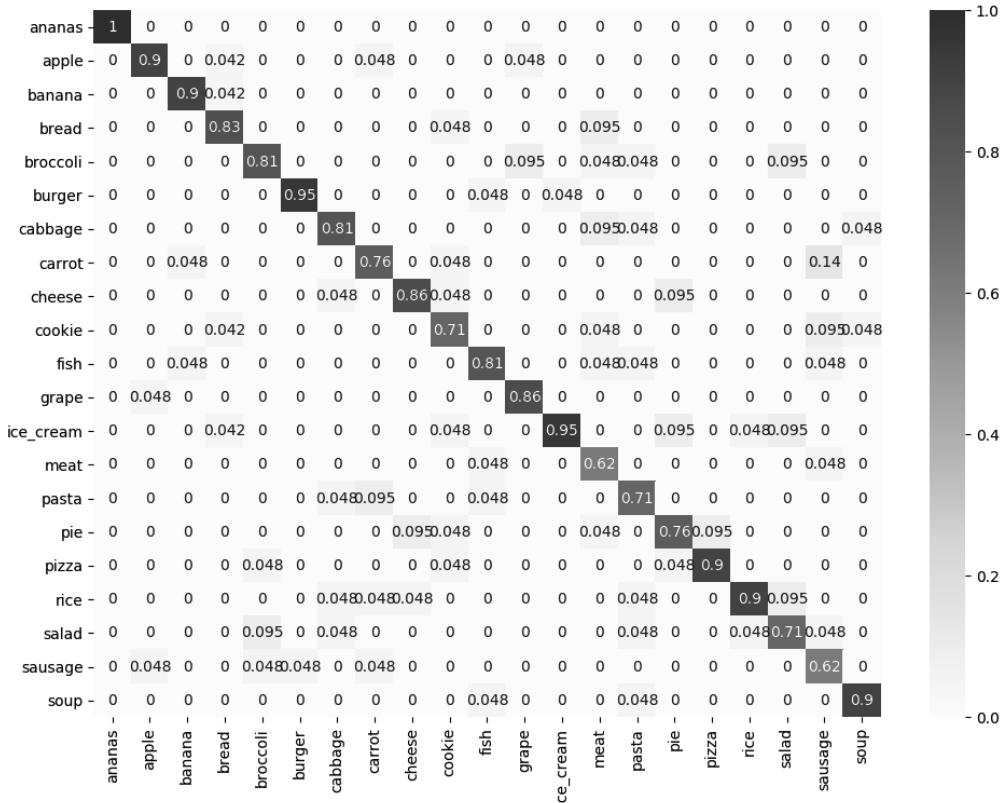


Figure 6.2: The image shows the confusion matrix built through the test set. On the left and the bottom borders the ground truth classes and the predicted classes are written on the left and the bottom border respectively. The bar on the right shows the probability that a given category was predicted correctly. For example the second number (0.9) in the top left portion over the diagonal represents the probability that the apple images were predicted as apples. In the grapes row in the same column the number is smaller, which means that apple images was wrongly predicted as grapes with that probability.

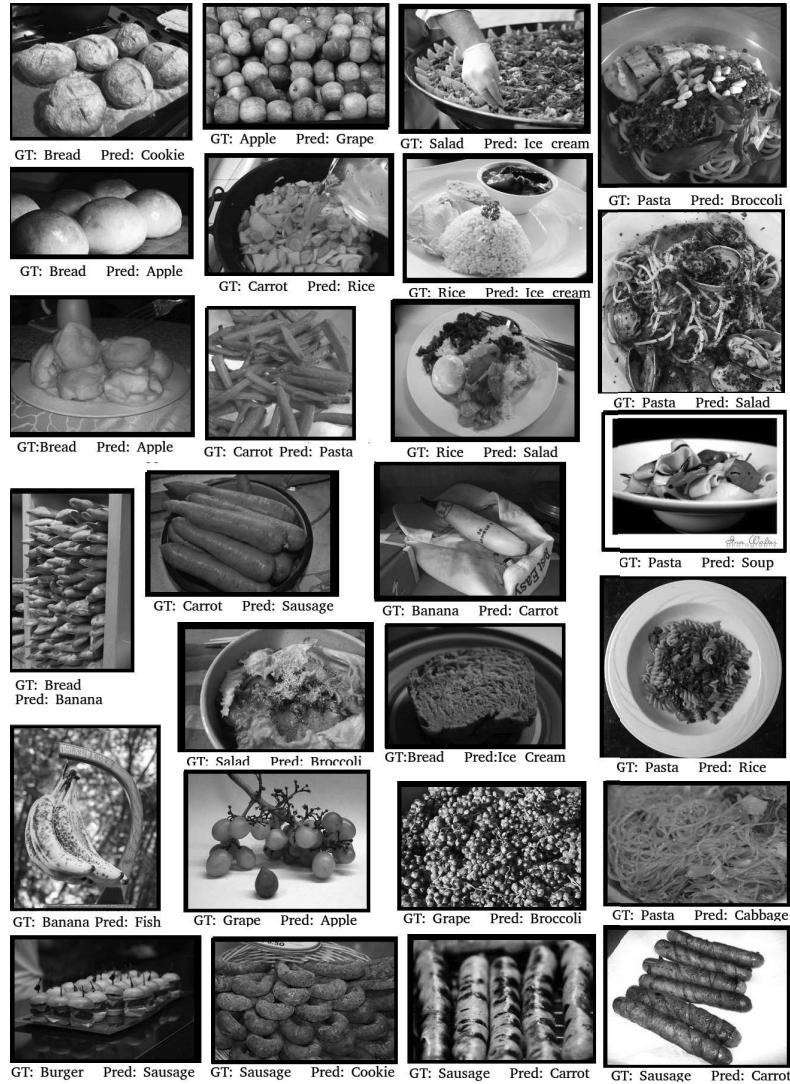


Figure 6.3: The figure shows some of the mislabeled images with the ground truth (GT) and the predicted (Pred) category.

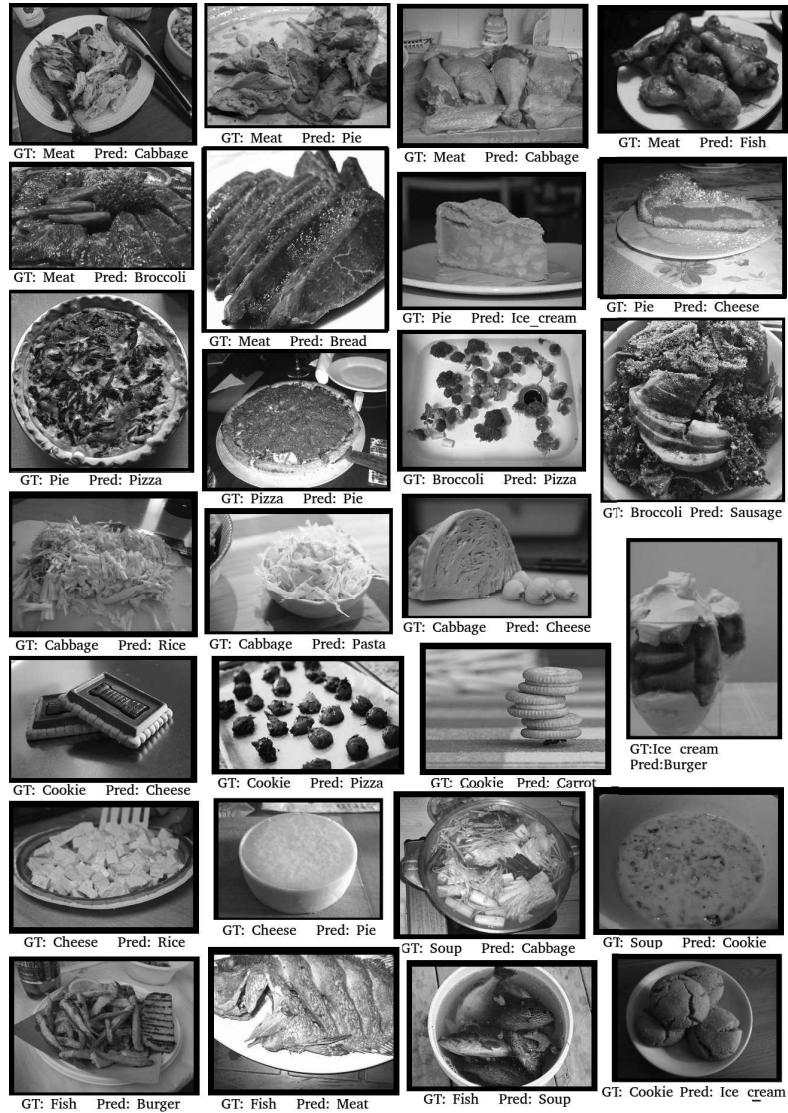


Figure 6.4: The figure shows some of the mislabeled images with the ground truth (GT) and the predicted (Pred) category.

It must be said, however, that the results of the test’s score cannot be considered reliable with such small datasets. In fact, the shuffling approach randomly splits the images into two sets. Different splits make the visual model train differently and the top-1 scores can change based on the images in the training and test sets. In other words, the top-1 score should have a variance due to the small number of training-test samples that makes the model less robust on different images.

If this food-classification model is used as a recognition tool in practice, the images that are forwarded through the network are rotated with different angles compared to the original test images. Testing the trained visual model on the test images with an additional rotation (30 degrees counterclockwise) made the top-1 accuracy score drop (from 82% to 65%). A solution to this drawback is to train the visual model with additional random rotations on the training images (in a range of ± 30 degrees). The evaluation of the newly trained visual model (where the preprocessing was changed with additional rotations) on the test set (either with or without rotations) does not yield the same original accuracies, but accuracies that are close enough nonetheless (79% for top-1 and 94% for top-5). In other words, adding rotations to the training preprocessing helps the network to be more robust in practical use.

6.5.2 Food Captioning Model Results

Google’s Show and Tell model (Vinyals, 2015 [66]) initialized with the GoogLeNet-Inception-v3 model (pre-trained on ILSVRC dataset) was applied to establish the baseline for the captioning performance. The baseline perplexity score of the Show and Tell model on the MS COCO dataset is 8.7. The evaluation of this baseline model on our dataset had a perplexity of 900+. This happened due to the diversity in the vocabulary file and training captions of the MS COCO dataset. After that, the training was started on our food caption dataset (discussed in Section 6.3). Our newly trained food visual model (whose results are discussed in Section 6.5.1) was used for the initialization of the parameters of the Show and Tell model. With such a small number of image-caption pairs, the training was fast (less than 10 minutes), with a nice convergence towards the minimum. The perplexity of this trained food-captioning model was approximately 23.3, which is slightly lower (and thus better) than the scores of the other models on the MS COCO and Flickr30K datasets (see Section 4.2). However, the number of images and training captions must be taken into account (fewer words in the vocabulary means fewer possible choices for the generated sentence).

Despite the high perplexity of the baseline model, the captions on the test images of the food caption dataset (discussed in Section 6.3) are still reasonably accurate (see first three sentences of each image in Figure 6.5) compared to the captions of the food-captioning model.

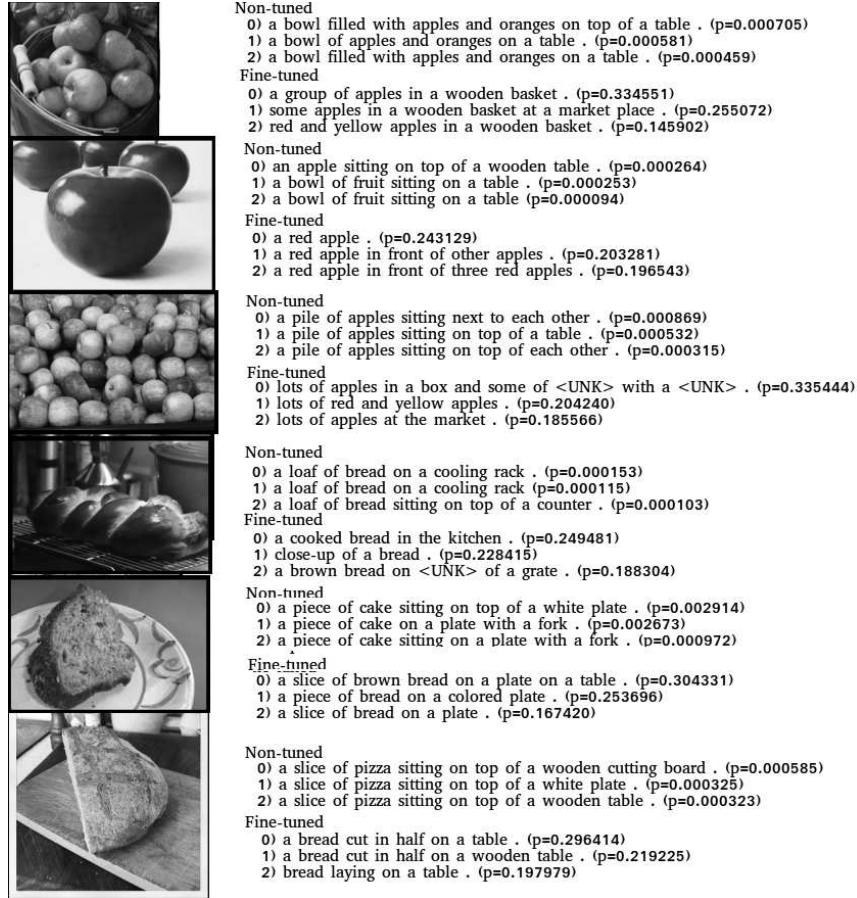


Figure 6.5: Six images part of the test set of our food-captioning dataset (defined in Section 6.3). On the right of each image there are three sentences created by the baseline captioning model and three more that use our food visual model. The numbers at the end of each sentence are the probability of that sentence. The words *<UNK>* are unknown words that are repeated less than four times in the training set.

6.5.3 Food Captioning by Region Proposals

As the last part of this work, a region-proposal model (defined in Section 5.1.3) was implemented and tested. The food visual model is the same and is defined in Section 5.1.1. The evaluation was performed on the test images of the food image dataset, achieving a score of 86.5% top-1 accuracy. The multiple crops of the region-proposal method increase the possibility of getting the prediction right. Hence, the prediction accuracy is higher than with the visual model only (84.2%).

Using the region-proposal model (see Section 5.1.3) not only allows images to be classified, but also their contents to be described. The images in the food dataset are images that contain only one type of food category per image. To test the description generation, images that contained different types of food per image (foods that both belong and do not belong to the food dataset) were selected. Unfortunately, due to the bigger cropping approach used (45% – 100%) the region-proposal model poorly predicts all the food categories present in the images, achieving an accuracy score of only 34%. This is due to the difference in size between the food objects in our food dataset and the images with multiple types of food (where each object is proportionally smaller to the quantity of objects in the image), but with an adjustment of the cropping area (from a range 45% – 100% to smaller crops of range 0.05% – 0.15%) the detection rise to 65%. Figure 6.6 shows an example.



Figure 6.6: The original image is forwarded to the region-proposal network that picks random smaller sub-regions and classifies those sub-regions. The sub-regions are classified by the food visual model. In this specific case the predicted categories were: grapes, banana and pineapple with 91%, 92% and 74% accuracies. The yellow boxes show the location of the picked categories.

Chapter 7

Conclusions

The first chapters of this work presented a basic introduction to the theory of artificial neural networks with an additional focus on convolutional and recurrent neural networks where the state-of-the-art research was reviewed in detail.

The basic knowledge gathered through this first part was used to address the problems of food classification and food captioning with three new implemented methods. The food-classification model is a CNN with the same architecture as the GoogLeNet-Inception-v3 model (Szegedy, 2015 [64]) but with a different preprocessing approach. The food-captioning model uses the architecture of the Show and Tell model (Vinyals, 2015 [66]) built in two parts: a CNN that classifies images and an LSTM-RNN that describes the contents of images with sentences. The food-captioning-by-region-proposal, that picks a specific number of sub-regions, uses the CNN of our first method to classify those regions and retain/discard predicted labels if the prediction score is higher than 70%.

The lack of larger food datasets consisting of common foods, one category per non-segmented image only, allowed the creation of two new food datasets. The first is a 21-category balanced food dataset with 1470 images. The second is built with two categories of the previous food dataset, where 5 training sentence were added to every image. This dataset is used to train

the food-captioning model.

The results of our methods were similar to those of the baseline models. The GoogLeNet-Inception-v3 model trained on the ILSVRC dataset achieved a top-5 accuracy of 96.42% against the ILSVRC benchmark, while our food-classification model trained on our own food dataset achieved 98% top-5 accuracy (and 82.4% top-1 accuracy). When the baseline-captioning model (Show and Tell model with the visual model trained on the ILSVRC dataset and the captioning model trained on the MS COCO dataset) was evaluated on our captioning dataset it showed a perplexity of over 900. After the Show and Tell architecture, initialized with our own food visual model, was trained on our food image dataset, the perplexity plummeted to 23.3. The region-proposal model was evaluated for image classification under the same circumstances as for the visual model with an even better classification accuracy score (86.5% top-1 accuracy). Furthermore, this model can predict different foods on the same image with a food recognition score of 64%.

A qualitative evaluation between the visual models cannot be done due to the fact that the models were trained on different datasets and thus an evaluation over the same dataset cannot be performed, but as we saw from the results in both cases on their respective datasets the classification accuracy is similar. Our food-classification model, unfortunately, still misclassifies some of the categories such as cookie, meat, pasta, salad and sausage, where the classification top-1 average per category is approximately 70%.

The evaluation comparison between the two captioning models, contrary to the two visual models, shows that both models create accurate captions on our food dataset. The model trained on the MS COCO gives even better results in terms of specific words used in the generated sentences, while our model seems to reproduce the training sentences. This most probably happens cause by the small variations in training sentences due to the small food caption dataset. Additional testing was performed on our food-captioning model, where we checked whether the model could create captions when the input image had more than one main food object in it, with scarce results.

Our model creates sentences on a single object contained in the image.

The evaluation of the region-proposal model showed that the food image classification ran better than our visual model alone and could also predict different foods in a single image. But, it could still predict categories that were not in the image when the visual model misclassified the sub-regions.

7.1 Future Work

The preprocessing of images seems to be one of the most important steps in getting the model to train well, but is still not completely understood. The ideas that are circulating at present are mostly based on random cropping and perturbations. With thorough research, new preprocessing methods could be proposed that are specific for specific images or purposes. For example, photographs of fruits or vegetables are taken in any possible position and from different vantage points, while photographs of cars, traffic lights or buildings are only taken top-side up. Thus, a preprocessing involving the rotation of images might help the task of image classification.

The CNN's architecture in our food classification and food captioning with region-proposal methods was not changed, due to time and hardware constraints and to layer dependencies, but with new architectures that are being created, an update will be possible (highway nets, dense nets, fractal nets...) that might yield even better performance.

The Show and Tell model still holds the first place on the MS COCO benchmark, but when its architecture was used in our food-captioning model, it did not do all we have hoped for. The captioning model may learn combinations of objects if the training set contains such combinations. So, one future task in our food captions dataset research could be to find and caption images containing different combinations of main objects.

Bibliography

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [2] Ahmed M. Abdel-Zaher and Ayman M. Eldeib. Breast cancer classification using deep belief networks. *Expert Systems with Applications*, 46:139 – 144, 2016.
- [3] Lukas Bossard, Matthieu Guillaumin, and Luc J. Van Gool. Food-101 - mining discriminative components with random forests. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI*, pages 446–461, 2014.
- [4] Léon Bottou. Stochastic gradient descent tricks. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade (2nd ed.)*, volume 7700 of *Lecture Notes in Computer Science*, pages 421–436. Springer, 2012.

- [5] Mark Chang. Applied deep learning 11/03 convolutional neural networks, 2016. [Online; accessed 10-August-2017].
- [6] Gianluigi Ciocca, Paolo Napoletano, and Raimondo Schettini. Food recognition: a new dataset, experiments and results. *IEEE Journal of Biomedical and Health Informatics*, 21(3):588–598, 2017.
- [7] Dan C. Ciresan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012.
- [8] Wikimedia Commons. File:overfitting.svg — wikimedia commons, the free media repository, 2015. [Online; accessed 16-August-2017].
- [9] Wikimedia Commons. File:error surface of a linear neuron with two input weights.png — wikimedia commons, the free media repository, 2017. [Online; accessed 9-August-2017].
- [10] Wikimedia Commons. File:loss function surrogates.svg — wikimedia commons, the free media repository, 2017. [Online; accessed 9-August-2017].
- [11] Wikimedia Commons. File:rastrigin function.png — wikimedia commons, the free media repository, 2017. [Online; accessed 9-August-2017].
- [12] Wikimedia Commons. File:recurrent neural network unfold.svg — wikimedia commons, the free media repository, 2017. [Online; accessed 31-August-2017].
- [13] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. Large scale distributed deep networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1223–1231. Curran Associates, Inc., 2012.

- [14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255. IEEE Computer Society, 2009.
- [15] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Trevor Darrell, and Kate Saenko. Long-term recurrent convolutional networks for visual recognition and description. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 2625–2634, 2015.
- [16] Timothy Dozat. Incorporating nesterov momentum into adam. 2015.
- [17] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.
- [18] Hao Fang, Saurabh Gupta, Forrest N. Iandola, Rupesh Kumar Srivastava, Li Deng, Piotr Dollár, Jianfeng Gao, Xiaodong He, Margaret Mitchell, John C. Platt, C. Lawrence Zitnick, and Geoffrey Zweig. From captions to visual concepts and back. In *CVPR*, pages 1473–1482. IEEE Computer Society, 2015.
- [19] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 580–587, 2014.
- [20] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

- [21] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey J. Gordon and David B. Dunson, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 315–323. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011.
- [22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [23] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet, and CA Google Inc., Mountain View. Multi-digit number recognition from street view imagery using deep convolutional neural networks. 2013.
- [24] A. Graves, A. r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013.
- [25] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *CoRR*, abs/1503.04069, 2015.
- [26] Hamid Hassannejad, Guido Matrella, Paolo Ciampolini, Ilaria De Munari, Monica Mordonini, and Stefano Cagnoni. Food image recognition using very deep convolutional networks. In *Proceedings of the 2Nd International Workshop on Multimedia Assisted Dietary Management, MADiMa ’16*, pages 41–49, New York, NY, USA, 2016. ACM.
- [27] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.

- [28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.
- [29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778, 2016.
- [30] Donald O. Hebb. *The organization of behavior: A neuropsychological theory*. Wiley, New York, June 1949.
- [31] Geoffrey Hinton. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent, 2012. [Online; accessed 14-August-2017].
- [32] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.
- [33] Andrew G. Howard. Some improvements on deep convolutional neural network based image classification. *CoRR*, abs/1312.5402, 2013.
- [34] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.
- [35] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456, 2015.
- [36] Hokuto Kagaya, Kiyoharu Aizawa, and Makoto Ogawa. Food detection and recognition using convolutional neural network. In *Proceedings of the 22Nd ACM International Conference on Multimedia, MM ’14*, pages 1085–1088, New York, NY, USA, 2014. ACM.

- [37] Andrej Karpathy and Fei-Fei Li. Deep visual-semantic alignments for generating image descriptions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 3128–3137, 2015.
- [38] Y. Kawano and K. Yanai. Automatic expansion of a food image dataset leveraging existing categories with domain adaptation. In *Proc. of ECCV Workshop on Transferring and Adapting Source Knowledge in Computer Vision (TASK-CV)*, 2014.
- [39] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016.
- [40] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.
- [41] Ivan Krasin, Tom Duerig, Neil Alldrin, Vittorio Ferrari, Sami Abu-El-Haija, Alina Kuznetsova, Hassan Rom, Jasper Uijlings, Stefan Popov, Andreas Veit, Serge Belongie, Victor Gomes, Abhinav Gupta, Chen Sun, Gal Chechik, David Cai, Zheyun Feng, Dhyanesh Narayanan, and Kevin Murphy. Openimages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from <https://github.com/openimages>*, 2017.
- [42] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.
- [43] Alon Lavie and Abhaya Agarwal. Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments. In

- Proceedings of the Second Workshop on Statistical Machine Translation*, StatMT '07, pages 228–231, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.
- [44] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.
 - [45] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.
 - [46] Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, Zhiheng Huang, and Alan L. Yuille. Learning like a child: Fast novel visual concept learning from sentence descriptions of images. *CoRR*, abs/1504.06692, 2015.
 - [47] Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, and Alan L Yuille. Explain images with multimodal recurrent neural networks. *NIPS Deep Learning Workshop*, 2014.
 - [48] Y. Matsuda, H. Hoashi, and K. Yanai. Recognition of multiple-food images by detecting candidate regions. In *Proc. of IEEE International Conference on Multimedia and Expo (ICME)*, 2012.
 - [49] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon. A PROPOSAL FOR THE DARTMOUTH SUMMER RESEARCH PROJECT ON ARTIFICIAL INTELLIGENCE. <http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html>, 1955.
 - [50] Michael A. Nielsen. Neural networks and deep learning. In "Neural Networks and Deep Learning". Determination Press, 2015. [Online; accessed 9-August-2017].
 - [51] Christopher Olah. Lstm-blog, 2017. [Online; accessed 31-August-2017].

- [52] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, ACL '02*, pages 311–318, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.
- [53] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012.
- [54] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.
- [55] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.
- [56] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.
- [57] Andrew Senior, Georg Heigold, Marc’arelio Ranzato, and Ke Yang. An empirical study of learning rates in deep neural networks for speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Vancouver, CA, 2013.
- [58] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.
- [59] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

- [60] Ilya Sutskever. *Training Recurrent Neural Networks*. PhD thesis, Toronto, Ont., Canada, Canada, 2013. AAINS22066.
- [61] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1139–III–1147. JMLR.org, 2013.
- [62] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.
- [63] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 00:1–9, 2015.
- [64] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2818–2826, 2016.
- [65] Ramakrishna Vedantam, C. Lawrence Zitnick, and Devi Parikh. Cider: Consensus-based image description evaluation. In *CVPR*, pages 4566–4575. IEEE Computer Society, 2015.
- [66] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 3156–3164, 2015.

- [67] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L. Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Sanjoy Dasgupta and David Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1058–1066. JMLR Workshop and Conference Proceedings, May 2013.
- [68] B. Widrow and M. E. Hoff. Adaptive switching circuits. *1960 IRE WESCON Convention Record*, pages 96–104, 1960. Reprinted in *Neurocomputing* MIT Press, 1988 .
- [69] S. Wiesler, R. Schlüter, and H. Ney. A convergence analysis of log-linear training and its application to speech recognition. In *2011 IEEE Workshop on Automatic Speech Recognition Understanding*, pages 1–6, Dec 2011.
- [70] Wikipedia. Kernel (image processing) — wikipedia, the free encyclopedia, 2017. [Online; accessed 22-August-2017].
- [71] Sebastien C. Wong, Adam Gatt, Victor Stamatescu, and Mark D. McDonnell. Understanding data augmentation for classification: When to warp? In *2016 International Conference on Digital Image Computing: Techniques and Applications, DICTA 2016, Gold Coast, Australia, November 30 - December 2, 2016*, pages 1–6, 2016.
- [72] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2048–2057, 2015.
- [73] Keiji Yanai and Yoshiyuki Kawano. Food image recognition using deep convolutional network with pre-training and fine-tuning. In *2015 IEEE International Conference on Multimedia & Expo Workshops, ICME Workshops 2015, Turin, Italy, June 29 - July 3, 2015*, pages 1–6, 2015.

- [74] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.
- [75] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, pages 818–833, 2014.