Find objects in real estate images with convolutional neural networks

OSKAR RÅHLÉN OCH SACHARIAS SJÖQVIST

Degree project in Computer Science
Date: May 5, 2019
Supervisor: Handledare
Examiner: Examinator
School of Electrical Engineering and Computer Science
Swedish title: Hitta object i fastighetsbilder med convolutional neural networks

Abstract

English abstract goes here

Sammanfattning

Svenskt sammanfattning

Contents

Chapter 1

Introduktion

Koppla första mening till titel. söka på speicfika saker i en annons. mysig inledning. Deeplearning bildanalys - google grejer. Ai och hur automatisering vuxit fram. Lite fakta på hur viktig sökfunktioner är för att hitta bostäder? Hur många sökningar /tidsenehet. Om folk enklare hittar bostad på ett mer effektivt sätt så effektiviseras hela köp och sälj processen och därmed hela marknaden.

Just nu (april 2019) finns över 20 000 bostadsrätter (https://www.hemnet.se/statistik) till salu på Sveriges största mäklarsite (https://www.hemnet.se/om), där nästan hälften av dessa ligger i Stockholm. Dit kommer 2,8 miljoner (https://www.hemnet.se/om) unika besökare varje vecka och gör tillsammans över 2000 sökningar i minuten. Detta ställer höga krav på filtreringsfunktionerna för att potentiella köpare snabbt ska kunna hitta sin drömbostad. Idag erbjuds redan filtreringsfunktionerna ”antal rum”, ”boarea”, ”pris” samt ”område”. Det finns också en fritextsöka där man kan söka på vad mäklaren har skrivit i texten. Exempel på sökord är ”balkong”, ”kakelugn” och ”sjöutsikt”.

Problemet med att söka i mäklartexter är att det tvingar mäklaren att i texten nämna samtliga attribut som han eller hon vill göra sökbara. Detta gör att vissa attribut kan bli utelämnade vilket i sin tur gör fritextrutan sämre.

Exempelvis är det ofta självklart om det finns en balkong eller kakelugn på mäklarbilderna men det är inte alltid mäklaren väljer att skriva detta i den löpande texten. Det hade därför varit intressant att utifrån mäklarbilderna automatiskt generera ett sökindex där varje bild knyts till ett antal attribut. Dessa attribut skulle sedan kunna användas för att lägga till filtreringsalternativ i sökfunktionen och därmed skapa en bättre användarupplevelse.

Det skulle också gå att använda sig av denna metod för att knyta attribut såsom skick och typ av rum (Sovrum, badrum eller kök) till varje bild. Kombinationen skulle göra sökningar såsom ”Kök i gott skick” möjligt.

Denna data kan också vara användbar vid värdering av bostäder, då det enkelt skulle gå att jämföra priser på bostäder med olika attribut, tillexempel lägenheter med sköutsikt mot lägenheter utan sköutsikt eller lägenheter med ett kök i gott skick mot lägenheter med ett kök i sämre skick.

VET INTE RIKTIGT HUR JAG SKA FORMULERA MIG HÄR!! Känns Som ett halvbra stycke För att på ett effektivt och precist sätt kunna knyta attribut till mäklarbilder behöver man använda sig av någon form av automatisk bildbehandling. Detta går att göra med hjälp av maskininlärning och djupa neurala nätverk. Sättet som detta görs på är att man tränar ett antal bilder som innehåller det givna attributet och ett antal som inte innehåller attributet. Modellen tränas sedan med hjälpt av ett djupt neuralt nätverk. Efter varje träning så testas modellen för att mäta hur många träningar som krävs för bästa resultat. När modellen är färdigtränad kan man använda denna genom att skicka in en bild som input och modellen berättar om bilden innhåller attributet eller inte.

1.1 Syfte

Syftet med denna studie är att titta på hur olika maskininlärningsmetoder kan användas för att hitta relevanta nyckelord till bilder på lägenhetsannonser.

1.2 Frågeställning

Går det med nuvarande verktyg inom maskininlärning hitta attribut i bilder från lägenhetsannonser?

1.3 Avgränsningar

För att sätta en rimlig avgränsningar på denna uppsats kommer tre olika sorters attribut undersökas och jämföras med tre olika sorters deeplearningmodeller. Attributen som kommer– användas är ”Balkong”, ”Kakelugn” samt ”Typ av rum” (Kök, badrum eller sovrum) och deelearningmodellerna som kommer användas är ”keras”, ”smartAI”, samt ”blabla”. Anledningen till att flera attribut eller flera sorters modeller inte valdes är att det är resurskrävande att både samla in och sortera testdata samt att implementera modellerna. Anledningen till att färre inte valdes var för att få spridning på attributen och modellerna, för kunna svara på om klassifiseringen fungerar i allmänhet och inte bara i enstaka testfall. En annan avgräsning som gjort är att mäklarbilderna

som används endast kommer från bostadsrätter, då det kan skilja sig ganska mycket på hur villor och bostadsrätter ser ut.

Chapter 2

Bakgrund

2.1 Maskininlärning

Maskininlärning är ett sätt att programmera datorer så de optimerar ett problem med hjälp av exempeldata eller tidigare erfarenhet [1]. Maskininlärning är ett aktivt forskningsfält inom datalogi. Lärandeprocessen för en maskininlärningsalgoritm kan se ut på följande sätt: Ett datorprogram har som uppgift att lära sig en förutbestämd uppgift T, hur väl den gör detta, algoritmens prestanda, är måttet P. Till sin hjälp har datorprogrammet tidigare erfarenhet E. Om prestandan P blir högre då vi tillhör erfarenheten E till programmet, då lär sig programmet.

Maskininlärningsmodeller kan delas in i två kategorier, övervakat lärande (eng. supervised learning) och oövervakat lärande (eng. unsupervised learning) [2]. Vid övervakat lärande så är erfarenheten E redan kategoriserad och uppmärkt för uppgiften T, vid oövervakat lärande så är den inte det. Ett exempel är en maskininlärningsmodell som har som uppgift T att kategorisera bilder i vissa förutbestämda kategorier. Vid övervakat lärande kommer erfarenhet E bestå av bilder som redan är uppmärkta med rätt kategori. Modellen ska då med hjälp av erfarenheten kunna kategorisera ny data som inte är uppmärkt.

2.2 Artificella neurala nätverk

En samling algoritmer inom maskininlärning med många tillämpningsområden är artificella neurala nätverk (ANN). Enligt Goodfellow, Bengio, and Courville [3] så var från början ANN ett försök till att bygga en digital modell av det biologiska neuronsystemet. Forskningen inom områden avvek dock snabbt från att efterlikna den biologiska varianten och fokuserade istället på

att öka prestandan på maskininlärningsproblem. De minsta byggstenarna i ANN, neuronen, fungerar fortfarande som dess biologiska variant. En neuron får signaler in och skickar sedan signaler ut via synapserna. Hur stark utsignal bör vara simuleras i en dator med hjälp av vikter (eng. weights), även kallat för parametrar. Varje neuron i nätverket har sina tillhörande vikter. När ett ANN tränas så görs detta genom att justera dessa vikter [3]. De viktade insignalerna bör även nå upp till en viss tröskel för att en utsignal ska skickas, detta simuleras i ANN med aktiveringsfunktioner. ANN är en acyklisk graf som består av dessa artificiella neuroner.

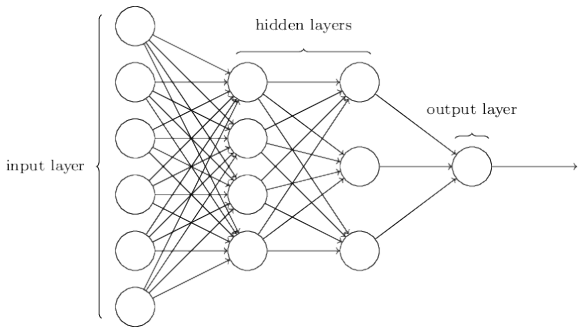

Figure 2.1: En illustration över ett artificiellt neuralt nätverk [4].

Ett ANN består först av ett indatalager (eng. input layer). Indatalagret representeras av en neuron per kännetecken (eng. features) i indatan. Då indata är bilder motsvarar en neuron i indatalagret en pixel i bilden. Indatalagret är sedan ihopkopplat med ett nytt lager med neuroner. Det kan vara flera lager efter varandra och dessa kallas för de gömda lagren (eng. hidden layers). De gömda lagren kan bestå av godtyckligt många neuroner och de behöver inte ha samma antal neuroner. Efter de gömda lagren kommer utdatalagret (eng. output layer). Antalet neuroner i utdatalagret bör motsvara formen på den utdata vi vill ha. Om modellen ska kategorisera bilder i tio olika kategorier, bör utdatalagret innehåller tio neuroner. Det är värdet dessa neuroner i utdatalagret har som är svaret från vårt ANN.

Ett ANN blir bättre på att utföra sin uppgift genom att ändra sina vikter. Detta sker i två steg, feed-forward pass och backward pass, även kallat för backpropagation [1]. Vid feed-forward skickas träningsdata (som kommer från tidigare erfarenhet E) in i nätverket för att få ut ett svar. Då träningsdatan är uppmärkt så jämförs svaret från nätverket med det korrekta svaret. Utifrån antal fel och hur fel nätverket hade, beräknas en kostnad. Denna kostnad kan beräknas på olika sätt men vanligtvis används funktionen cross-entropy loss [3]. Målet är att träna modellen så denna kostnad blir så låg som möjligt. Nästa

steg är backward pass. Vi vill ändra vikterna så att kostnaden blir lägre, detta görs genom att beräkna gradienten av kostnaden med avseende på alla vikter. Vi tar sedan ett steg åt motsatt håll till gradienten för att minska kostnaden. Storleken på detta steg kallas för inlärningstakt (eng. learning rate).

Vanligtvis delas träningsdatan in i högar, vilket kallas för mini-batch [3]. Istället för att beräkna gradienten för varje datapunkt i träningsdatan så beräknas gradienten för en hel mini-batch för att sedan använda sig av en genomsnittlig gradient. En epok (eng. epoch) är när nätverket har gått igenom alla datapunkter i träningsdatan. Hur många träningssteg detta innefattar beror på antal datapunkter samt storleken på mini-batch.

Stochastic gradient descent

Det finns olika algoritmer för exakt hur vikterna ska ändras när ett ANN tränas. En av de vanligaste algoritmerna är Stochastic Gradient Descent (SGD) [3]. Den bygger på feed-forward pass och backward pass men beräknar gradienten på ett fåtal slumpmässigt utvalda datapunkter istället för alla datapunkter i träningsdatan [5]. Anledningen är att effektisera modellen så träningen går snabbare. En variant är mini-batch SGD, där samma principer appliceras på en mini-batch istället för på hela träningsdatan.

En vidareutveckling är SGD med momentum [6]. Algoritmen med momentum kommer ihåg förändringen av vikterna vid föregående iteration och baserar nästa uppdatering på en linjärkombination av gradienten och den tidigare förändringen. Denna teknik har fått stor genomslagskraft inom forskningen av ANN [7].

2.3 Neurala nätverk med konvolution

Neurala nätverk med konvolution (eng. Convolutional Neural Network, CNN) är en viss typ av neurala nätverk som fungerar bra då indatan kan ses som ett rutnät [3]. Detta inkluderar data om tidsserier, som kan ses som ett endimensionellt rutnät, men också bilder, som kan ses som ett rutnät av pixlar. CNN introducerades för 20 år sedan men det har, på grund av begränsningar i hårdvara och arkitekturer, inte gått att träna djupa CNN förrän för några år sedan [8]. Enligt Huang et al. [8] har CNN idag blivit den dominanta tekniken inom maskininlärning för igenkänning av visuella objekt. Och enligt Simon, Rodner, and Denzler [9] är CNN som är förtränade på bilder från ImageNet grunden till majoriteten av state-of-the-art-modeller inom bildkategorisering.

2.3.1 Lager

Det finns några grundläggande lager som används i de flesta typer av CNN, oberoende av arkitektur. CNN kan användas för många olika typer av indata men vi har här valt att fokusera på dess tillämpningar med bilder. En vanlig uppbyggnad av ett CNN för bildkategorisering är indatalager, konvolutionslager, ReLU, föreningslager och sist ett FC-lager.

Indatalager

Detta lager håller de råa pixelvärdena från bilden via skickar in. I ett vanligt neuralt nätverk med resnet-arkitekturen så är det 224 x 224 x 3. Då är det en rektangulär bild med 224 pixlar i höjd, 224 pixlar i bred och 3 färgkanaler, RGB.

Convolutional Layer

Ett convolutional layer består av en mängd filter som har parametrar som går att träna. Varje filter är kvadratiskt och små mått i höjd och bredd, men har alltid samma djup som vår indata. En vanlig storlek på ett filter är 5x5x3, det vill säga 5 pixlar brett och högt och 3 färgkanaler. Vid forward pass så glider vi (convolve) detta filter över vår indata-bild. Vi beräknar skalärprodukten av filtret och den 5x5x3-bit av indatan som filter ligger på. Efter beräkningen så glider vi filtret åt sidan [10]. Vanligtvis en pixel, men det kan även vara flera. Utdata från varje filter blir en tvådimensionell activation map. Djupet på utdata motsvarar antal filter vi använt i vårt convolutional layer. Höjd och bredd på utdata beror på storleken på filter.

Storleken på utdata från ett convolutional layer beror på tre hyperparametrar: djup, stride och zero-padding. Djup motsvarar antal filter som används och blir djupet på vår utdata. Stride består hur många pixlar vi förflyttar oss vid varje glidning. Zero-padding bestämmer om vi ska bygga en ram runt bilden med nollor. Detta användas för att kunna behålla storleken på bilden igenom flera convolutional layers men är också bra för att inte bortse viktig information i utkanten av bilden.

Activation Layer / ReLU

Activation Layer består av en elementvis funktion. Det finns flera olika typer av dessa, men det som vanligtvis används inom bildkategorisering är Rectified Linear Units (ReLU). Den applicerar funktionen max(0,x) på varje element i

indata. Detta betyder varje element som är positivt är opåverkat och varje negativt värde får värdet noll. Utdata har samma storlek som indata.

Pooling Layer

Ett pooling layer applicerar en nedsampling (eng: downsampling) på indata. Detta sker längs de spatiala dimensionerna, bredd och höjd. Detta sker vanligtvis för att minska antalet parametrar i nätverket och för att minska på beräkningskraften som behövs. Det är även en teknik för att undvika overfitting. Vanligaste typen är max pooling, då man tar ett område, till exempel 2x2 pixlar och väljer den pixel med högst värde. Vanligast är max pooling på 2x2 pixlars filter med en stride på 2. Indata minskar då med 75%.

Fully-connected Layer

Fully-connected layers är den typen av lager som är vanligast i normala neurala nätverk. I detta lager har varje neuron en full koppling till alla aktiveringar från det tidigare lagret. Dessa har då vanligtvis en tillhörande matris med vikter och bias, vilket kan beräknas med matrismultiplikation. Storleken på utdata beror på storleken på viktmatrisen. Vanligtvis så byggs det sista FC layer upp så att det har samma storlek som vi vill ha kategorier. Om vi vill klassifiera en bild i tio kategorier, så kan utdata från vårt sista lager ha storleken 10x1, där varje element motsvarar sannolikheten för att bilden tillhör en viss kategori.

2.3.2 Tekniker inom djupinlärning

Det finns vissa tekniker som är vanliga inom djupinlärning och som oftast används ihop med CNN. Ett urval av dessa presenteras här.

Batch-normalisering

Att träna djupa neurala nätverk är komplicerat då distributionen av varje lagers indatavärden förändras under träning [11]. Detta leder till att träningen av nätverket tar längre tid, då vi behöver använda lägre träningssteg (eng: learning rate). Det leder också till att parameter initisieringen blir väldigt finkänslig. Enligt [11] kallas detta fenomen för internal covariate shift och löses genom att normalisera indata till varje lager. Metoden bygger på att normaliseringen blir en del av arkitekturen och där en normalisering sker för varje träningsminibatch. Denna typ av batch normalization tillåter oss att använda högre träningssteg och gör att initieringen inte blir lika viktig. Det fungerar även som regularization, vilket gör att andra tekniker för regularization kan uteslutas.

Batch-normaliserring har visat sig vara extra viktigt för lyckad träning och konvergens för större nätverk, som VGG19 [9]. Det har även visat sig att den högre träningssteget, som batch-normalisering tillåter, leder till en högre slutlig noggrannhet [9]. Detta har visat sig fungera både på AlexNet och VGG19.

Data augmentation

Djupa neurala nätverk behöver en stor mängd data för att lära sig effektivt. Insamlingen av denna data är oftast dyrt och tidskrävande. Data augmentation (Dataförstärkning) genom att öka datamängden artificiellt genom att göra förändringar på befintlig data [12]. Det har visat sig att generell data augmentation har ökat prestandan på convolutional neurala nätverk. En viktig del av data augmentation är att indata fortfarande ska tillhöra samma klass som den tillhörde innan. Inom bildkategorisering så sker förändringen med geometriska och fotometriska transformationer. Geometriska transformationer ändrar geometrin i bilden med målet att vårt CNN ska vara oberoende av objektets position eller orientering [12]. Dessa transformationer inkluderar att flippa, beskära, skala och rotera bilden. Fotometriska transformationer förändrar istället färgkanalerna och målet är att vårt CNN ska vara oförändligt till skillnader i belysning och färg.

2.3.3 Arkitekturer

Här kommer ett urval av de vanligaste arkitekturerna för CNN för bildkategorisering att presenteras. Nätverk med conv-lager har blivit djupare och djupare, från LeNet [13] med fem lager, VGG med 19 lager [14] och Residual Networks (ResNet) med över 100 lager [15]. Ett problem med detta är att viktig information i indata försvinner innan det hinner nå slutet av nätverket [8]. Det har därför kommit arkitekturen som försöker lösa dessa problem. ResNet löser det genom att förbikoppla vissa lager med identitetskopplingar. DenseNet skapar flera olika förbikopplingar inne i nätverket.

Resnet

Det finns flera olika varianter av ResNet, Resnet18, Resnet34, Resnet50, Resnet101 och Resnet152, beroende på storlek. Vi fokuserar här på Resnet18. ResNet bygger på en struktur med convolutional layers, pooling layers och fully-connected layers. Trenden har varit att CNN blir djupare med fler lager.

Men till slut kom modellerna till ett tak när det handlar om accuracy. ResNet är en lösning på det här problemet genom att introducera "identity

shortcut connections". Detta betyder att det finns kopplingar som hoppar över vissa convolutional layers. Det betyder att om modellen inte behöver utnyttja alla convolutional layers så kan den bara använda sig av identiteten istället för ett convolutional layer. Det betyder att nätverket borde kunna vara hur stort som helst, för det går alltid att bara använda indentiteten. Modellerna med residuala funktioner ska även vara enklare att optimera [15]. Resnet beskär bilderna till storleken 224 x 224 pixlar, och huvudsakliga målet var att kategorisera bland 1000 kategorier i imageNet 2012 classification dataset. Det finns flera olika versioner av ResNet, beroende på antal lager. ResNet-18 består totalt av 18 lager, när det första lagret är ett convolutional layer och därefter ett max pooling-lager. Därefter är det många convolutional layer där det även sker en nedsampling. Till sist är det ett average pooling-lager, ett fully connected-layer med output 1x1000 och ett softmax-lager, som gör att de olika outputvärdena kan ses som en distribution.

Alexnet

Alexnet är en enklare variant av CNN och består av totalt åtta lager [16]. De första fem lagrena är convolutional layers, där vissa av dem har max pooling-lager emellan sig. De tre sista är fully connected-layers, där det sista har output 1x1000 för att motsvara klasserna i imagenet. Det sista lagret är också softmax. Alexnet använder sig av aktiveringsfunktionen ReLU. AlexNet vann ILSVRC 2012. Indata har storleken 224x244. De olika lagrena består av 11x11, 5x5, 3x3, convolutions, max pooling och ReLU aktiveringsfunktioner. Det tränades ursprungligen med SGD med momentum.

VGGNet

Enligt [14] så har VGG högre noggrannhet än Alexnet och uppnådde år 2015 state-of-the-art noggrannhet på ILSVRCs klassificering och lokaliseringsuppgifter. Alla conv-lager följer samma design som den i Alexnet [14].

Enligt [14] ser arkitekturen ut som följande: Under träning så består indata av RGB-bilder av fixerad storlek 224 x 224. Den enda förbehandlingen är att medelvärdet av RGB-värdena som beräknas på träningsmängden subtraheras från varje pixel. Bilden skickas sedan igenom en stack av conv-lager med filter av storlek 3 x 3. I en av konfigurationerna används även 1 x 1-filter, som kan ses som en linjär transformation av indatakanalerna. Kliven, Stride, är 1 pixel och padding beror på filterstorlek, men ska se till att indata och utdata är i samma storlek. Padding är 1 pixel vid 3 x 3-filter. Spatial pooling sker med fem stycken max-pooling lager, där vissa följer efter conv-lager. Det är

inte alla conv-lager som följs av max-pooling. Max-pooling sker med ett 2 x 2-filter med stride 2. En stack av conv-lager, som är olika djup beroende på vilken typ av VGG, följs sedan av tre stycken FC-lager. De första två har 4096 kanaler var och det sista har 1000-kanaler, för att motsvara klasserna i ILSVRC-problemet. Det sista lagret är ett soft-max-lager. Alla gömda lager är utrustade med ReLU icke-linjäritet.

Densenet

Tidigare forskning har visat att nätverk bestående av conv-lager kan vara djupare, har högre noggrannhet och är mer effektiva att träna om det finns kortare vägar mellan indata och utdata [8]. Därför skapades DenseNet, vilket kopplar ihop varje lager i nätverk med varje lager framför. Det betyder att om ett tradionellt nätverk med conv-lager har totalt L antal lager, så har det även L kopplingar. DenseNet har istället L(L+1)/2 antal direkta kopplingar. Enligt [8] så har DenseNet uppnått signifikanta förbättringar på Cifar-10 och ImageNet jämfört med andra state-of-the-art-modeller. Det betyder att alla lager har en direkt koppling till alla lager framför. DenseNet har olika intern arkitektur beroende på storlek. Gemensamt är dock att indata består av RGB-bilder av storlek 224 x 224. Därefter kommer en stack med conv-lager där första lagret har filterstorlek 7 x 7 och stride 2. Efterkommande conv-lager kommer i par, där det första i paret har storlek 1 x 1 och den andra har filterstorlek 3 x 3. Stride 1 och padding för att behålla storleken på indata. Mellan dessa lager sker först max-pooling med storlek 3 x 3 och stride 2 och sedan average pooling med storlek 2 x 2 och stride 2. Sista lagret, som är klassificeringslagret, består av global average pooling med storlek 7 x 7 och ett FC-lager med 1000 kanaler som ett soft-max-lager.

Inception-v2

Inception har en mycket lägre beräkningskostnad än VGG [17]. Detta har lett till att den används i big-data-sammanhang, där modeller med större beräkningskraft inte har kunnat användas. Inception-v2 tar som indata RGB-bilder i storleken 299 x 299. Inception-v2 består av en stack med conv-lager där alla har filterstorlek 3 x 3 med stride 1 eller 2. Det finns även mellan conv-lagren ett max pooling med storlek 3 x 3 och stride 2. Padding är så data ska behålla storleken. Därefter kommer tre stycken inception-lager. Ett inception-lager är ett lager som kombinerar resultatet från flera olika conv-lager i olika storlekar (vanligtvis 1 x 1, 3 x 3 och 5 x 5) ihop med ett max pooling-lager. Efter inception-lagren kommer klassificeringslagren som består av ett max pooling

med storlek 8 x 8, ett FC-lager med 2048 kanaler och sist ett FC-lager med 1000 kanaler med softmax.

Inception-v2 är unikt då det består av två utdatalager vid träning. [17] Det primära lagret är ett linjärt lager i slutet av nätverk medan det sekundära lagret kallas för auxiliary output. Vid testning så används bara det primära lagret.

2.4 Transfer Learning

Enligt [18] så har många djupa neurala nätverk som har tränats på naturliga bilder visar att de har en gemensam sak. Det är att i de första lagren så lär de sig egenskaper som motsvarar Gabor-filter och färgklickar. Gabor-filter är ett typ av filter som används för att beskriva textur. Det har även visat sig att hur vikterna i dessa första lager är beror inte på någon specifik datamängd eller uppgift utan är istället generell oberoende på datamängd. Dessa egenskaper behöver sedan formas från generella till specifika egenskaper för den specifika datamängden [18]. Detta sker i de sista lagren, men exakt vart har det inte studerats om. De första lagrena kallas därför för generella och det sista för det specifika.

Det intressanta med generella och specifika lager är att det blir möjligt att implementera transfer learning. I transfer learning så tränar vi först ett basnätverk på en bas-datamängd och en basuppgift för att sedan överföra de tränade vikterna till ett annat målnätverk som tränas med mål-datamängden och måluppgifen [18]. När måldatamängden är mycket mindre än basdatamängden så kan detta vara ett kraftfullt verktyg för att träna ett stort nätverk utan överanpassning (eng: overfitting). Vanligaste sättet är att träna ett basnätverk och sedan kopiera över dess n första lager till de n första lagren i målnätverket. De övriga lagren i målnätverket initieras slumpmässigt och tränas sedan för måluppgiften. Man kan välja att även träna de kopierade lagren när man tränar nätverket för måluppgiften. Det kallas för att finjustera (eng: fine-tune) lagren till den nya uppgiften. Man kan även låta bli att träna de kopiera lagren utan istället bara träna det sista klassificeringslagret. Detta kallas för att frysa lagren och brukas kallas för feature extraction.

Om det är bäst att träna alla lager (finetuning) eller frysa de kopierade lagren och träna sista lagret (feature extraction) beror på storleken på måldatamängden samt antal parametrar i de första n lagren. Om måldatamängden är liten och antal parametrar är stort, så leder fine-tuning oftast till överanpassning. Det är därför bättre med frysta lager. Om måldatamängden däremot är stor eller om antal parametrar är litet så kan finjustering användas för att få en högre noggrannhet än vad som hade varit möjlig med frysta lager [18].

2.4.1 ImageNet

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) är en utmaning som testar algoritmer för objektigenkänning och bildkategorisering i stor skala [19]. En anledning till detta är att utveckla en stor datamängd som redan är uppmärkt, för att minska barriär för att utveckla bättre modeller genom att ta bort processen med att märka upp bilder. Den andra anledningen är att mäta utveckling för datorseende. Huvudutmaningen består i att bestämma vilken klass en bild tillhör av 1000 klasser. ImageNet är en bilddatabas med uppmärkta bilder som används i tävlingen. Databasen består av strax över 1.2 miljoner bilder [20].

Det har blivit väldigt vanligt att för-träna ett bildigenkänningsnätverk på ImageNet för att det ska lära sig bra generella egenskaper [20]. Man använder då utmaningen ILSVRCs 1000 kategorier som basuppgift för att bygga upp ett basnätverk. Detta har visat sig fungera väldigt bra i många områden. Anledningen till detta diskuteras men både att basdatamängden är väldigt stor och att basuppgiften består av många olika kategorier är två anledningen som tillhör till dess generella egenskaper [20].

Chapter 3

Metod

Detta kapitel kommer att beskriva metoden som användes för att ge mäklarbilder attribut.

3.1 Datainsamling

För att kunna träna modeller för de olika attributen ”balkong”, ”kakelugn” samt ”typ av rum” behövdes ett antal bilder som både uppfyllde dessa attribut samt ”negativa” bilder som inte uppfyllde dessa attribut. Detta gjordes med ett pythonscript som heter Google Images Download (https://github.com/hardikvasa/google-images-download) och tar ett sökord, hur många bilder som ska laddas ner (X) samt från vilken källa bilderna ska hämtas från som input och ger tillbaka en mapp med de första X antal bilderna som hittas på Google från den givna källan.

Bildkällan som valdes var https://hemnet.se då det är Sveriges största bostadsförmedlingssite med mängder av bilder.

Sökorden samt antalen som användes för detta beskrivs här:

Sökord	Antal
balkong hemnet	400
balkong vasastan	400
balkong inspiration	400
hemnet vardagsrum	400
hemnet badrum	400
Hemnet braskamin	400
hemnet öppen spis	400
vardagsrum braskamin	400
hemnet hall	400
hemnet kök	400
hemnet uteplats	400

Eftersom denna data inte är validerade mäklarbilder med det attribut som sökes påbörjades ett valideringsarbete där alla bilder manuellt validerades och sorterades enligt följande:

- Alla bilder som inte uppfattades som mäklarbilder enligt definitionen ovan raderades
- Alla bilder som inte var tagna på lägenheter raderades
- Alla bilder med en eldstad las i en mapp med eldstäder
- Alla bilder med en balkong las i en mapp
- Alla bilder på ett kök las i en mapp
- Alla bilder på ett vardagsrum las i en mapp
- Alla bilder på ett badrum las i en mapp
- Negativa bilder skapades för samtliga mappar. Alltså skapades en mapp med bilder utan eldstad, en mapp med bilder utan balkong etc. Dessa skapades med hjälp av bilder i övriga mappar.

I varje mapp låg det slutligen runt 400 bilder.

När detta var slutfört delades varje mapp i ytterligare två delar, en som kallades ”training” och en som kallades ”validation”. I training palcerades de första 80% av bilderna och i validation de sista 20%. Detta gjordes för att kunna testa hur modellerna presterade när de var färdigtränade med 20% av den totala bildmassan.

3.2 Modeller

För att träna modellerna användes Python 3.6 med deeplearningramverket Pytorch 1.0.0. Modellerna tränades med hjälp av pytorch i de olika ramverken ”Resnet”, ”Alexnet”, ”VGG-11”, ”Densenet” samt ”inception V3”. Efter varje träningsrund (Epoch) kördes en validering för att se hur förbättringskurvan ser ut efter varje Epoch och när modellen börjar bli ”overfitted”.

Dessa modeller kördes ett antal epochs? och bla bla bla bla

3.3 Uppföljning

När modellerna var färdigtränade gjordes ett utdrag av statistik för att kunna jämföra modellerna sinsemellan. Datan som hämtades för varje modell var

3.4 Hårdvara

Hårdvaran som användes när modellerna tränades var följande:

3.4.1 Graphical processing unit (GPU)

GPUn som användes var av modell Tesla K80 och hade tillgång till 128GB grafikminne.

3.4.2 Central processing unit (CPU)

CPUn som användes hade fyra kärnor

3.4.3 Random Access Memory (RAM)

Det fanns tillgång till 61GB RAM-minne.

Hur vi gått tillväga. Vilka dataset, hur implementation gått till (verktyg, klassificerare, parametrar), hur vi valt features. Hur evalueringen har gått till (traning, test, validation set).

Chapter 4

Resultat

Prestandan av de olika modeller kommer presenteras här.

4.1 Balkonger

Här nedan presenteras resultatet av den binära klassificeringen av balkonger i mäklarbilder. Resultatet består av två grafer per modell, som visar hur kostnaden från kostnadsfunktionen samt noggrannheten för både vårt träningssset och evalueringsset. Det finns även en sammanställning av de högst uppnådda noggrannheten och hur lång tid modellerna tog att träna upp. Vi kommer även titta på både när alla lager var frysta och när alla lager tränades.

4.1.1 Feature extraction

Vi kan se kostnaden för båda tränings och valideringsdatan i figur 4.1 för varje epoch. Vi kan även se i figur 4.2 hur träffsäkerheten för de olika modeller var på balkonger.

Modell	Tid	Max. noggrannhet
Resnet	3m 37s	94.63
Alexnet	3m 18s	94.27
VGG-11	6m 21s	93.86
Densenet	5m 59s	96.94
Inception V3	9m 04s	95.80

Table 4.1: Sammanställning av feature extraction för balkonger

Modell	Tid	Max. noggrannhet
Resnet	06m 06s	96.56
Alexnet	03m 37s	95.41
VGG-11	15m 55s	97.32
Densenet	13m 55s	97.70
Inception V3	21m 54s	98.09

Table 4.2: Sammanställning av finetuning för balkonger

Modell	Tid	Max. noggrannhet
Resnet	02m 06s	80.50
Alexnet	01m 55s	80.50
VGG-11	03m 45s	77.35
Densenet	03m 35s	73.58
Inception V3	05m 15s	79.24

Table 4.3: Sammanställning av feature extraction för eldstäder

4.1.2 Finetuning

Vi kan se hur kostnadsfunktionen såg ut för varje epoch i figur 4.3 och hur träffsäkerheten var vid finetuning i figur 4.4

4.2 Eldstäder

4.2.1 Feature extraction

Vi kan se kostnaden för båda tränings och valideringsdatan i figur 4.5 för varje epoch. Vi kan även se i figur 4.6 hur träffsäkerheten för de olika modeller var på eldstäder.

4.2.2 Finetuning

Vi kan se hur kostnadsfunktionen såg ut för varje epoch i figur 4.7 och hur träffsäkerheten var vid finetuning i figur 4.8

Modell	Tid	Max. noggrannhet
Resnet	03m 31s	79.24
Alexnet	02m 08s	77.35
VGG-11	08m 56s	81.13
Densenet	07m 55s	85.53
Inception V3	12m 54s	83.64

Table 4.4: Sammanställning av finetuning för eldstäder

Modell	Tid	Max. noggrannhet
Resnet	02m 33s	93.75
Alexnet	02m 18s	93.22
VGG-11	04m 31s	93.75
Densenet	04m 20s	96.87
inception V3	06m 28s	93.75

Table 4.5: Sammanställning av feature extraction för rum

4.3 Rum

4.3.1 Feature extraction

Vi kan se kostnaden för båda tränings och valideringsdatan i figur 4.9 för varje epoch. Vi kan även se i figur 4.10 hur träffsäkerheten för de olika modeller var på rum.

4.3.2 Finetuning

Vi kan se hur kostnadsfunktionen såg ut för varje epoch i figur 4.11 och hur träffsäkerheten var vid finetuning i figur 4.12

Modell	Tid	Max. noggrannhet
Resnet	04m 16s	96.35
Alexnet	02m 34s	94.79
VGG-11	11m 02s	97.91
Densenet	09m 53s	97.91
Inception V3	16m 10s	97.39

Table 4.6: Sammanställning av finetuning för rum

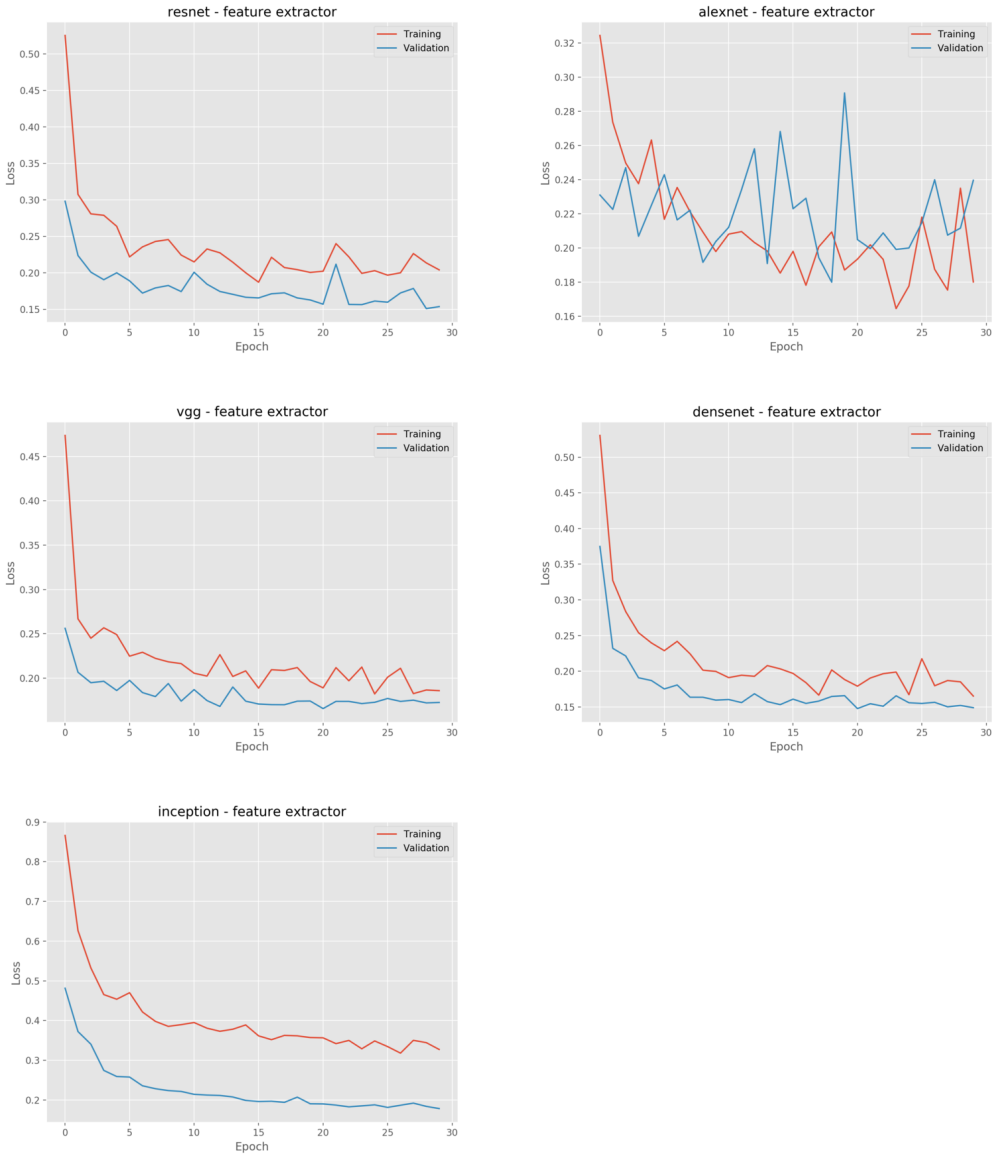

Figure 4.1: Kostnaden vid varje epoch för balkonger med feature extraction

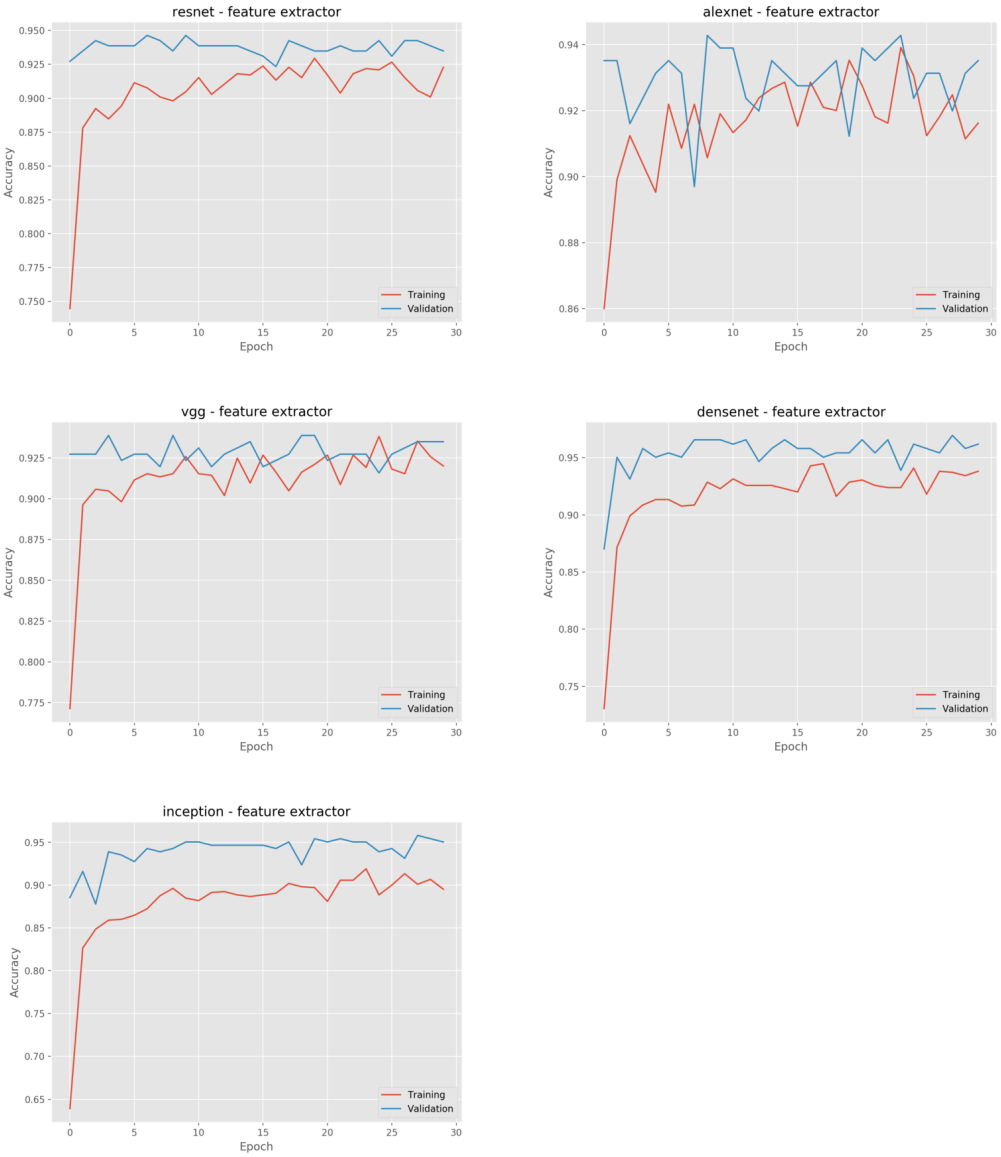

Figure 4.2: Träffsäkerhet för balkonger med feature extraction

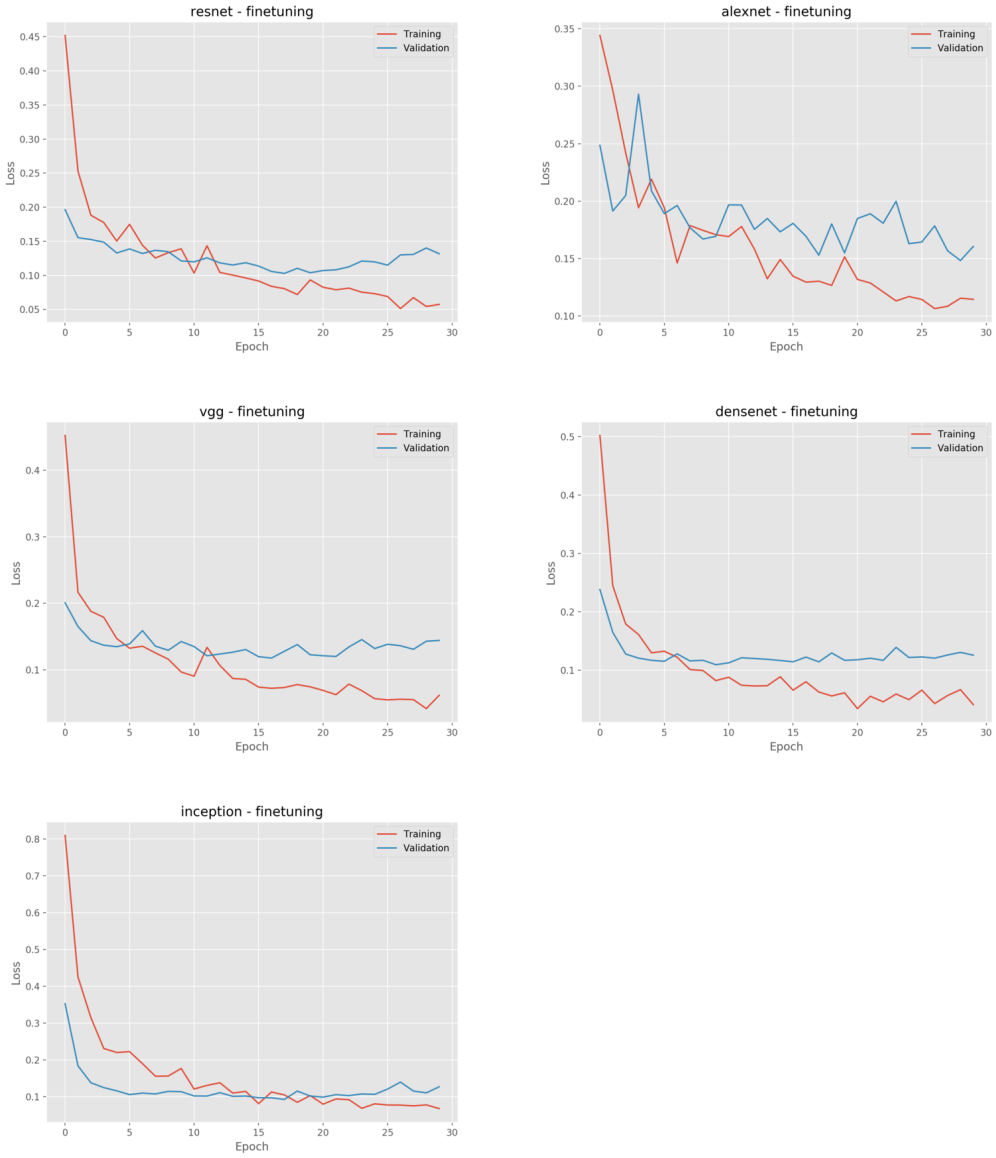

Figure 4.3: Kostnaden vid varje epoch för balkonger med finetuning

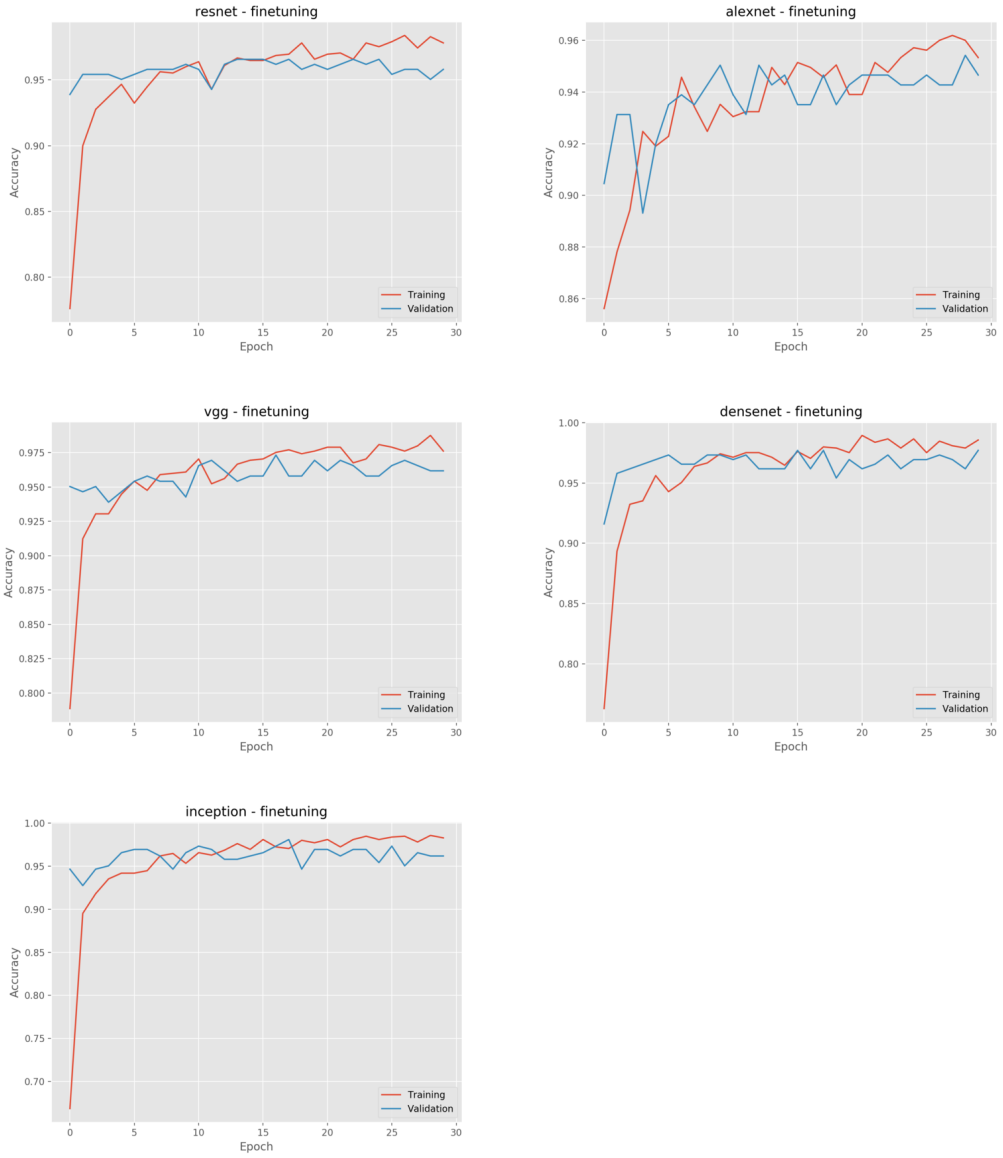

Figure 4.4: Träffsäkerhet för balkonger med finetuning

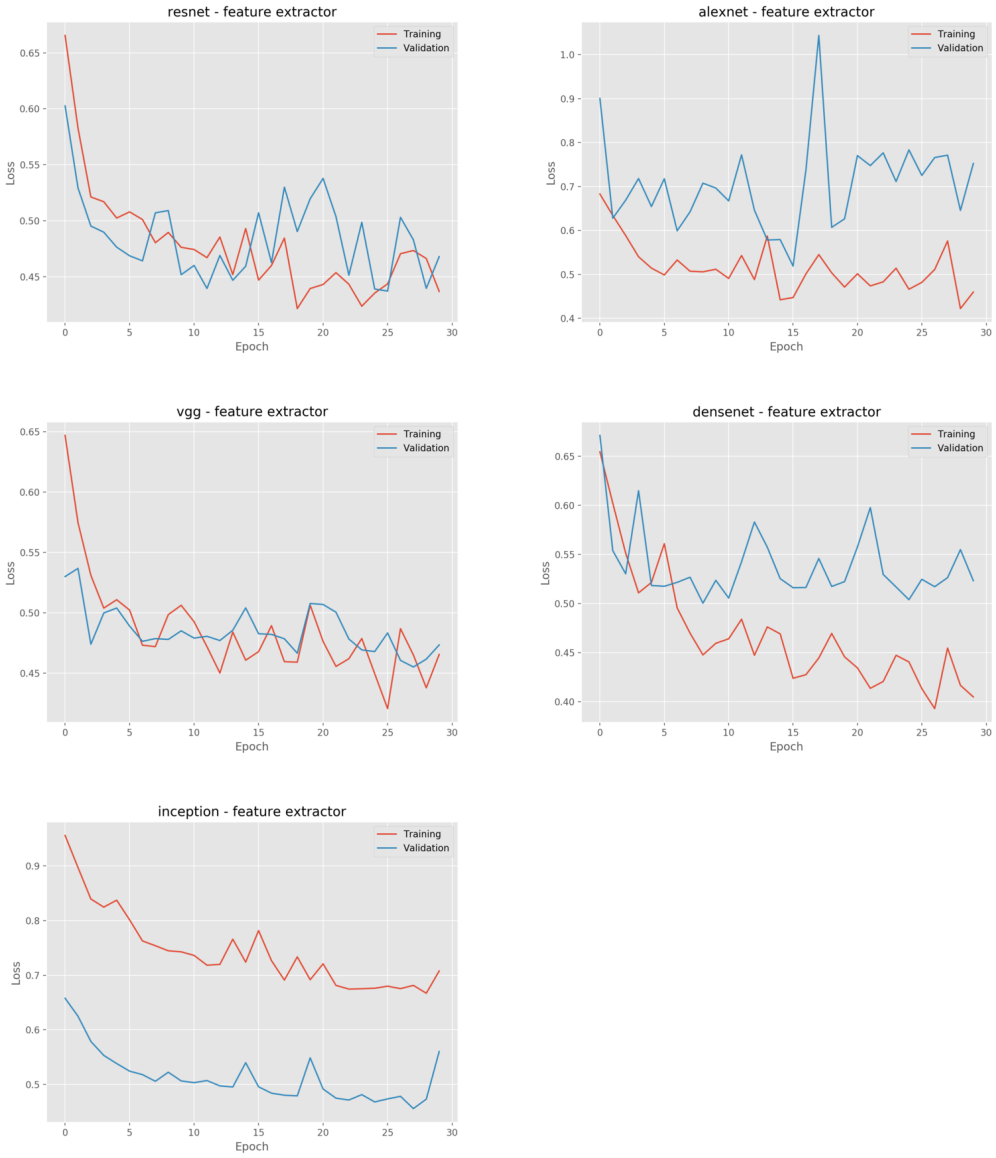

Figure 4.5: Kostnaden vid varje epoch för eldstäder med feature extraction

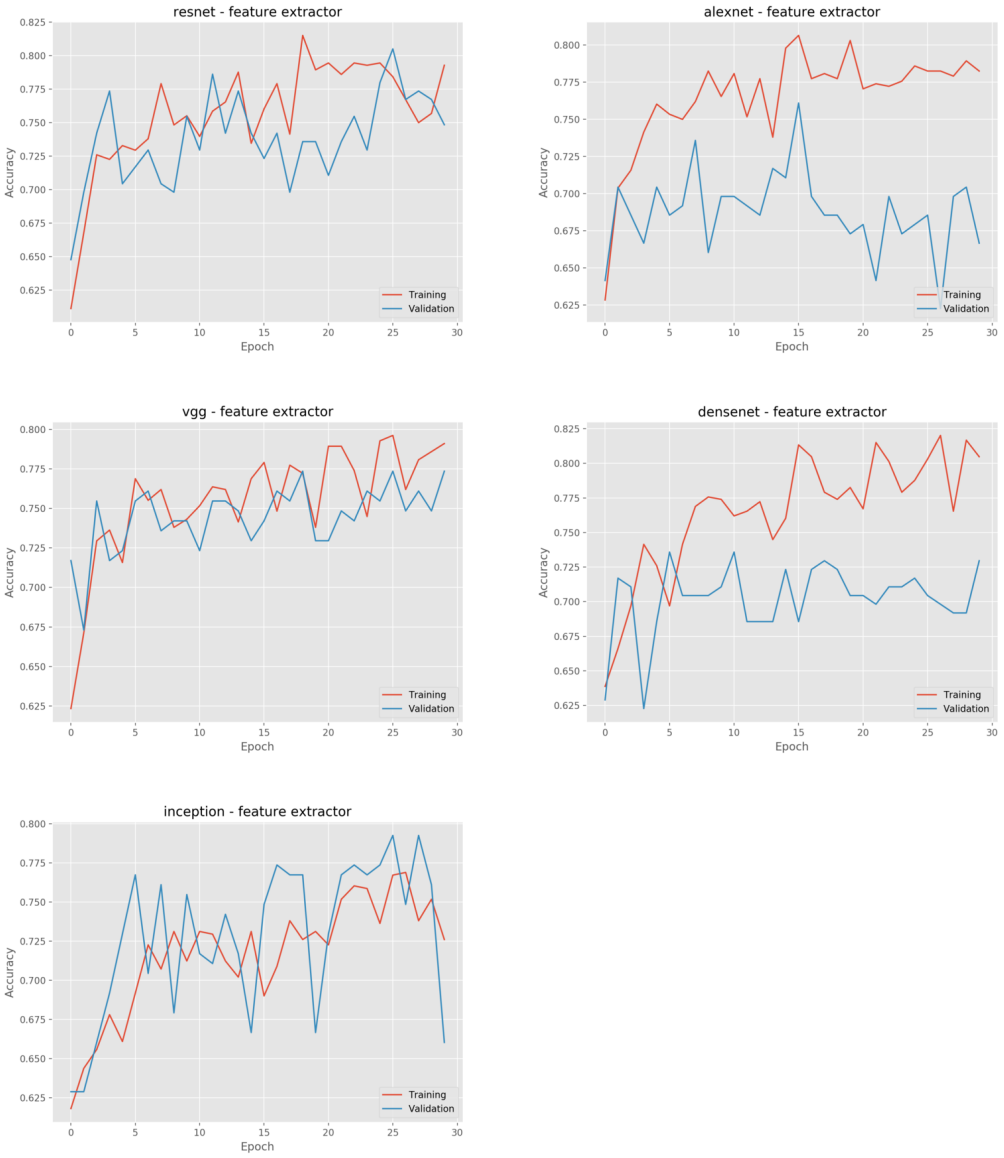

Figure 4.6: Träffsäkerhet för eldstäder med feature extraction

Figure 4.7: Kostnaden vid varje epoch för eldstäder med finetuning

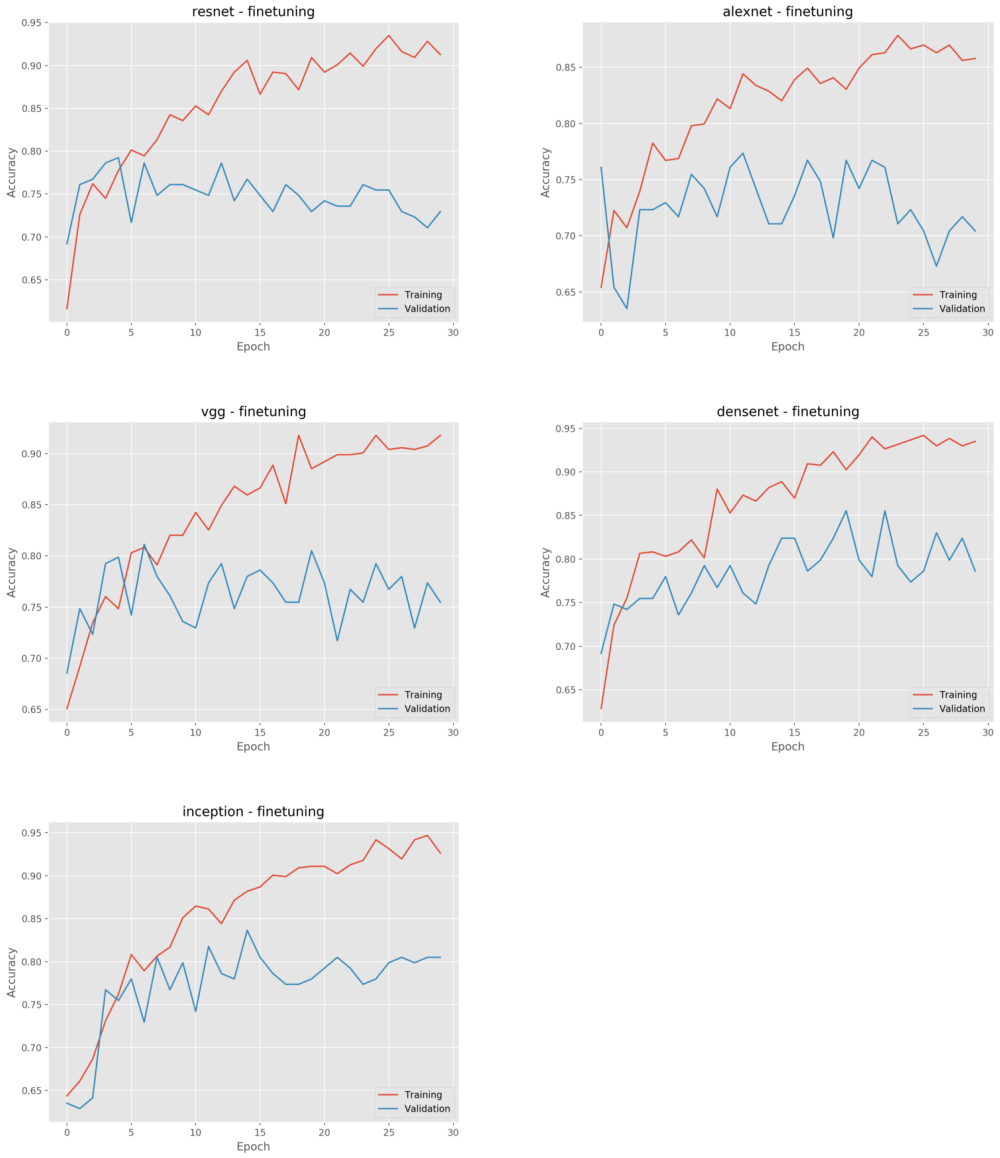

Figure 4.8: Träffsäkerhet för eldstäder med finetuning

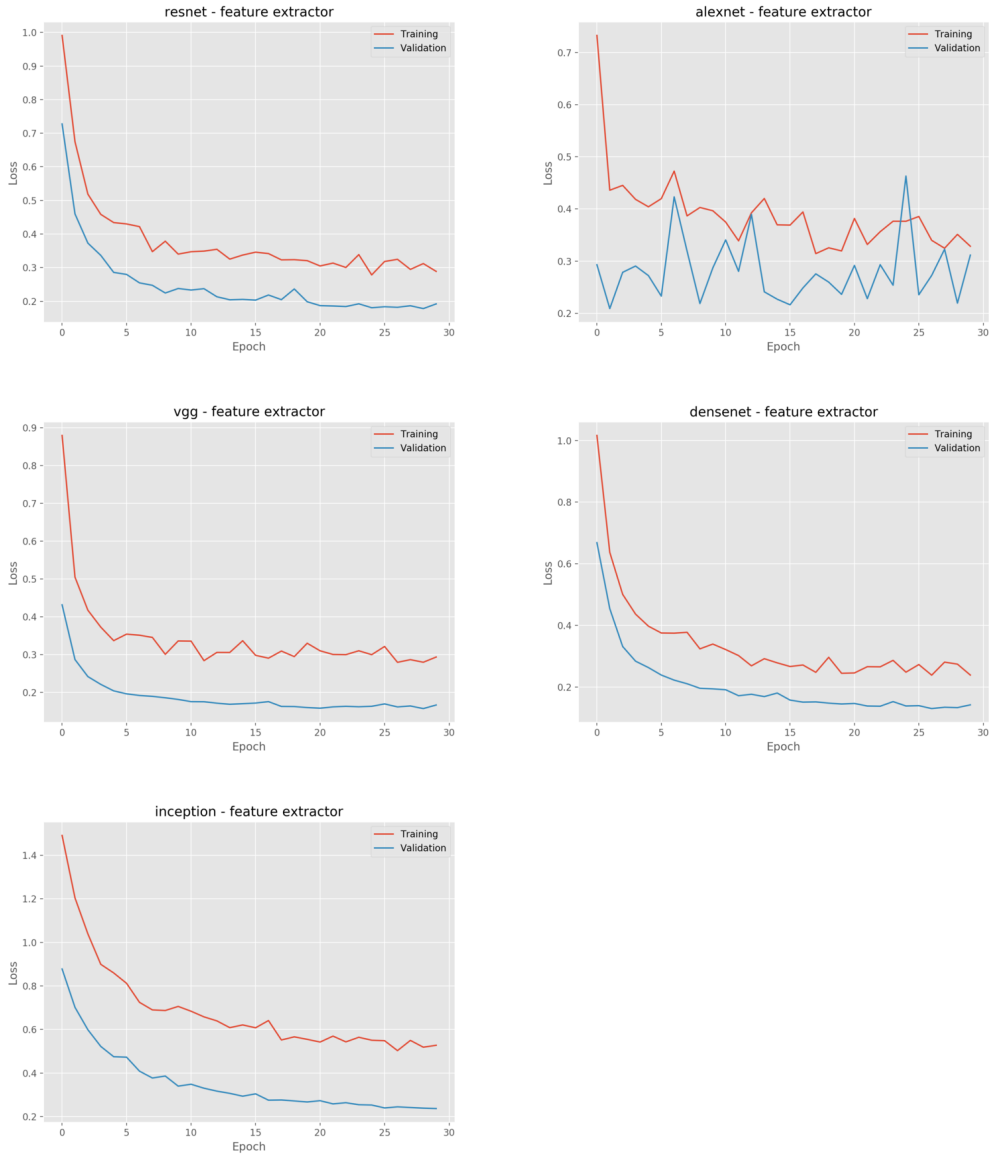

Figure 4.9: Kostnaden vid varje epoch för rum med feature extraction

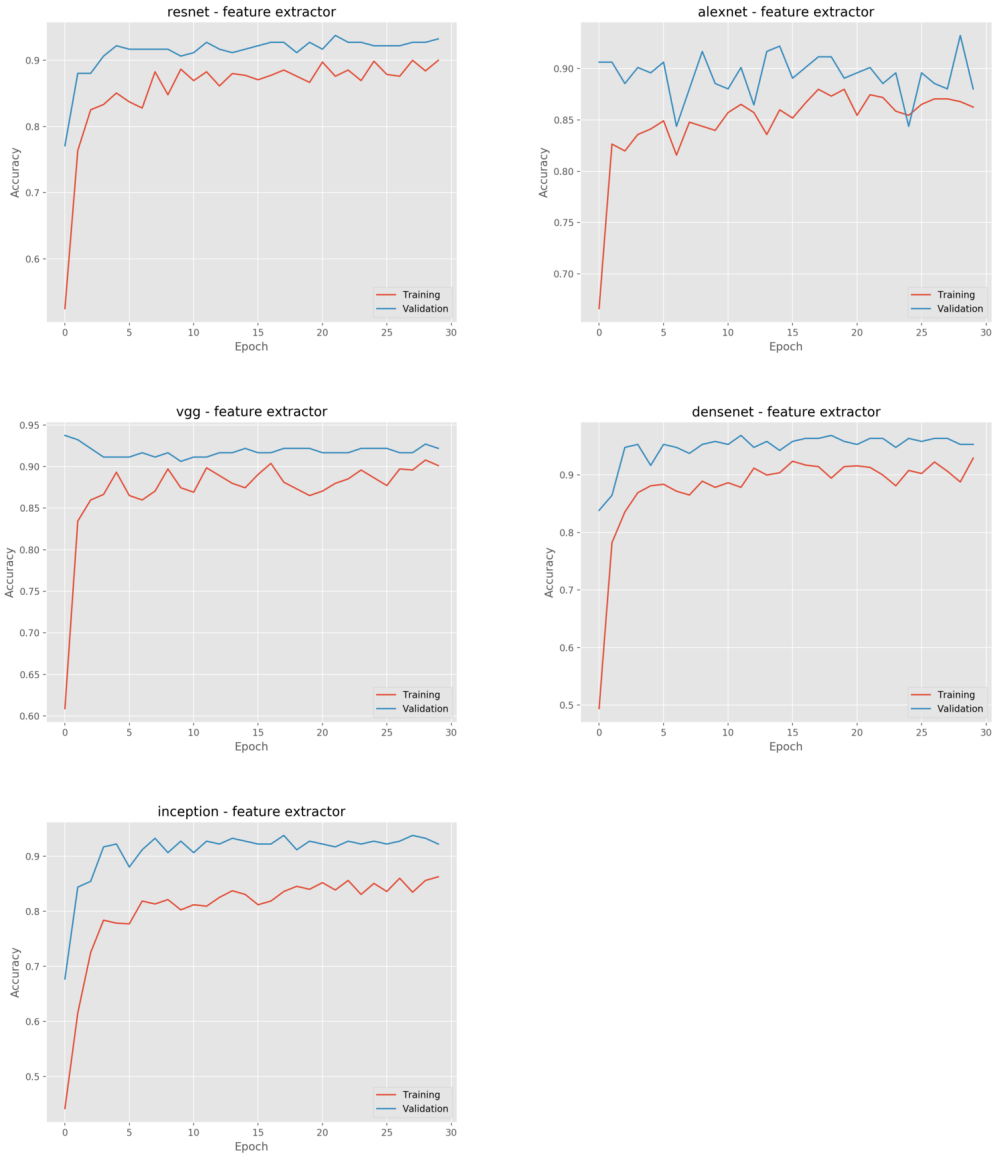

Figure 4.10: Träffsäkerhet för rum med feature extraction

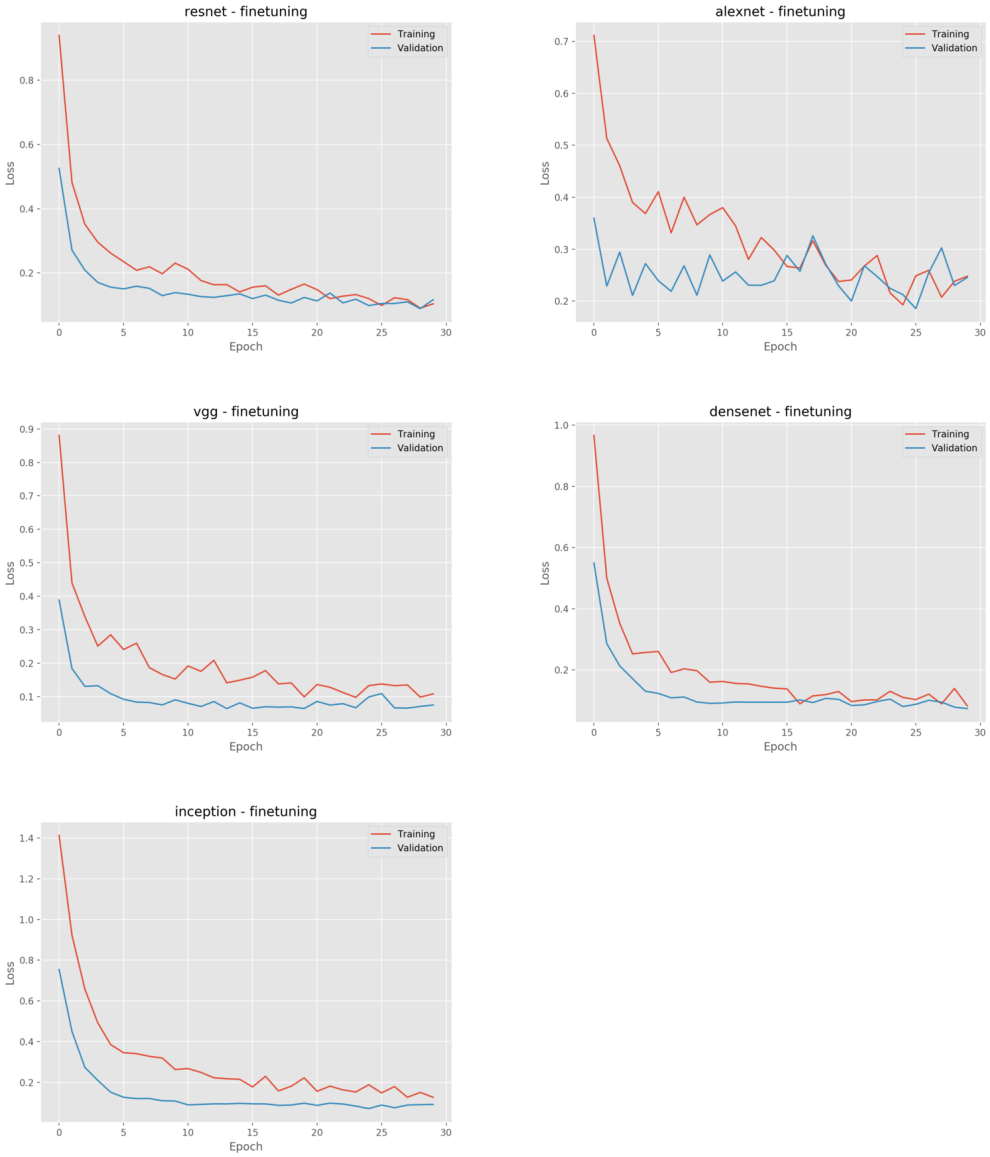

Figure 4.11: Kostnaden vid varje epoch för rum med finetuning

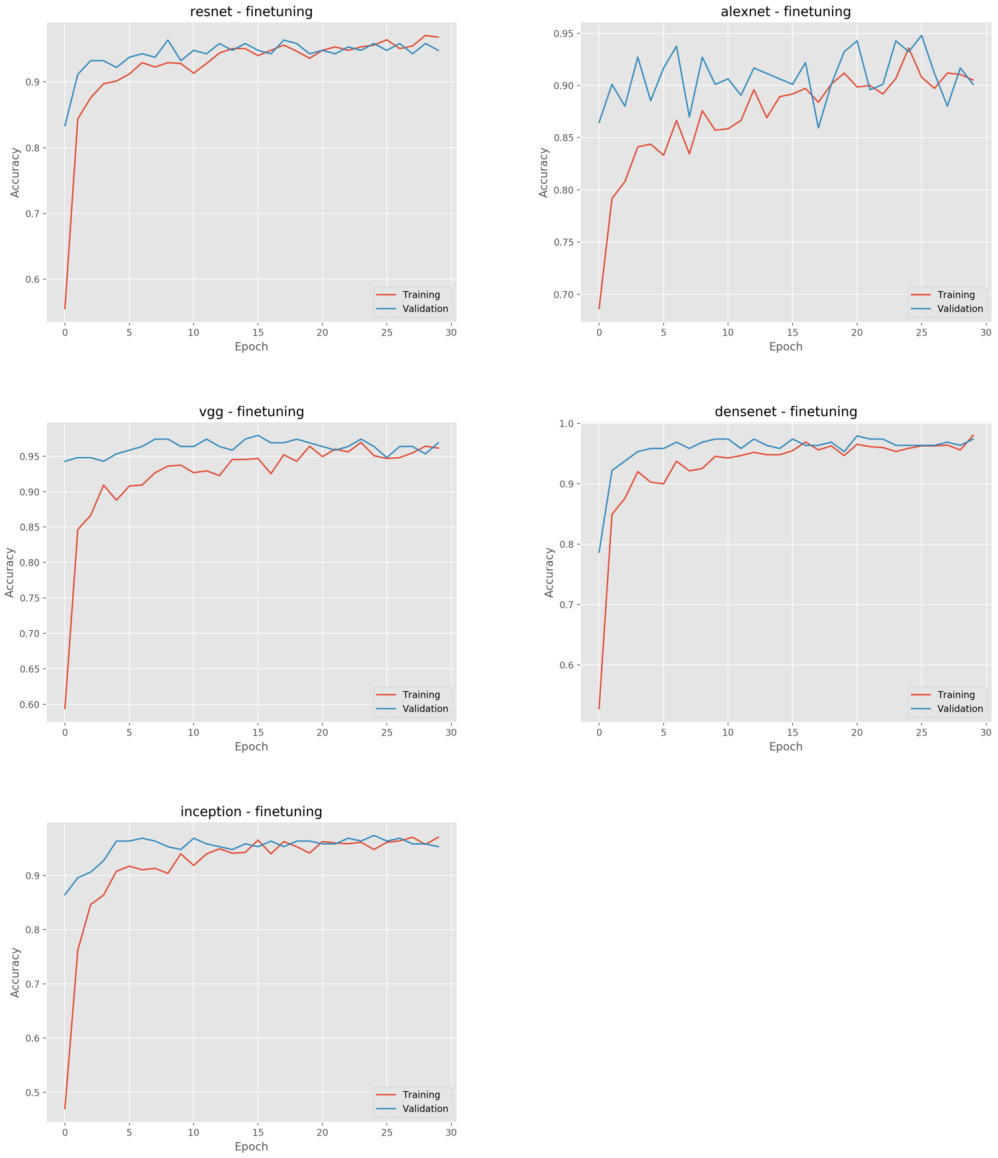

Figure 4.12: Träffsäkerhet för rum med finetuning

Chapter 5

Diskussion

Diskutera resultatet och hur olika delar kan ha påverkat eller påverkade. Diskutera eventuell framtida forskning. Begränsningar med resultatet. Etiska aspekter. Hållbarhet.

5.1 Fortsatt forskning

Vid värdering så är det också intressant att få ut attribut, så kan man räkna med det i värderingskalkylen.

Chapter 6

Slutsats

Slutsats av vad vi kom fram till.

Bibliography

[1] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2009.

[2] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[4] Michael A Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA: 2015.

[5] Léon Bottou. "Large-scale machine learning with stochastic gradient descent". In: *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[6] Ning Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural networks* 12.1 (1999), pp. 145–151.

[7] Ilya Sutskever et al. "On the importance of initialization and momentum in deep learning". In: *International conference on machine learning*. 2013, pp. 1139–1147.

[8] Gao Huang et al. "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[9] Marcel Simon, Erik Rodner, and Joachim Denzler. "Imagenet pre-trained models with batch normalization". In: *arXiv preprint arXiv:1612.01452* (2016).

[10] Kaiming He et al. "Spatial pyramid pooling in deep convolutional networks for visual recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 37.9 (2015), pp. 1904–1916.

[11] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[12] Luke Taylor and Geoff Nitschke. “Improving deep learning using generic data augmentation”. In: *arXiv preprint arXiv:1708.06020* (2017).

[13] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[14] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[15] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[17] Christian Szegedy et al. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[18] Jason Yosinski et al. “How transferable are features in deep neural networks?” In: *Advances in neural information processing systems*. 2014, pp. 3320–3328.

[19] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[20] Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. “What makes ImageNet good for transfer learning?” In: *arXiv preprint arXiv:1608.08614* (2016).

Appendix A

Appendix A