Examensarbete

HÖGSKOLAN I SKÖVDE

JÄMFÖRELSE AV CNN MODELLER FÖR OBJEKTIDENTIFIERING OCH AUTOMATISK MARKERING

COMPARISON OF CNN MODELS FOR VISUAL OBJECT DETECTION AND LABELING

Examensarbete inom huvudområdet
Informationsteknologi
Grundnivå 30 Högskolepoäng
Vårtermin 2020

Albin Berg

Handledare: András Márki
Examinator: Henrik Gustavsson

Sammanfattning

En svårighet med att använda Artificiell Intelligens, är resurserna som krävs för att utföra beräkningarna under en acceptabel tidsram, men också med en bra träffsäkerhet. Målet med denna uppsats är att jämföra olika modeller av convolutional neural networks, mellan träffsäkerhet och hastighet, för att hitta den modell som är mest effektiv. Dessutom evalueras den mest effektiva modellen genom en webblösning, som kan markera bilder med text.

Resultatet visar att varje modell har olika fördelar i hastighet och träffsäkerhet, men att VGG16 har nära till bäst resultat utan de problem som andra modeller har.

Nyckelord: Artificiell Intelligens, Hastighet, Träffsäkerhet

Innehållsförteckning

1 Introduktion

Artificiell Intelligens spelar en stor roll i våra vardagliga liv. Det är något vi sällan tänker på, och i många fall vet vi inte ens om det. Något som de flesta kanske inte känner till, är att Artificiell Intelligens spelar en stor roll i saker som diagnostisering inom vården, eller för styrningen i ett flygplan. Det är ett kraftfullt verktyg som är relativt enkelt att sätta upp, och som har potential till väldigt många tillämpningar. Kanske sker den största tillämpningen av Artificiell Intelligens, på internet.

Med hjälp utav Artificiell Intelligens kan vi identifiera objekt och innehåll på internet, vilket tillämpas mer och mer. Detta kan vara önskvärt, då man vill identifiera skadlig innehåll på sociala medier, som kan behövas tas ner. Fördelen med att använda Artificiell Intelligens, är att vi inte behöver mänskliga resurser för att identifiera något manuellt. Detta har då potential att befria resurser.

Därför är det viktigt att hitta den AI lösning som använder minst resurser, med rätt träffsäkerhet. Den största “Neural Network” som är lämpad för att analysera bilder, är convolutional Neural Network (CNN). Målet är att tre olika CNN modeller testas, där hastighet och träffsäkerhet räknas. Den mest effektiva modellen används sedan i en webblösning, som kan markera ut en valfri uppladdad bild, med en text som beskriver vilken kategori den liknar mest som.

2 Bakgrund

2.1 Machine Learning

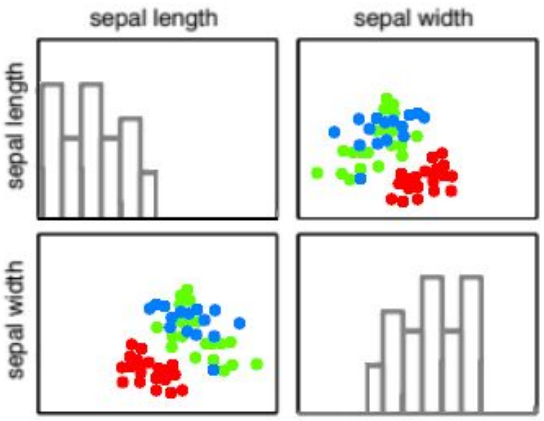
Machine Learning är en tillämpning av Artificiell Intelligens, och en samling algoritmer och statistiska modeller som ger systemet möjlighet att lära sig från erfarenhet utan någon explicit programmering. Istället så förlitar Machine Learning sig på mönster och slutledning. Metoden består huvudsakligen av tre subkategorier: Supervised Learning, Unsupervised Learning, Reinforcement Learning (Murphy, K. P. 2012).

2.2 Supervised Learning

Supervised learning är den form av Machine Learning som är mest tillämpad i praktiken (Murphy, K. P. 2012). Supervised learning är designad för att lära sig av exempel. Detta sker genom träningsdata från datasets, som säger åt algoritmen vilka mönster att söka efter. Beroende på vilket dataset vi använder, kommer modellen alltså att ge annorlunda resultat vid analys av bilder. Ett större dataset kan ge mer precision, men kan också vara långsammare som resultat.

Classifications spelar en stor del av Supervised Learning, där vissa kategorier skapas för att hitta specifika mönster av t.ex en bild.

Figur 1 Tre olika typer av blommor

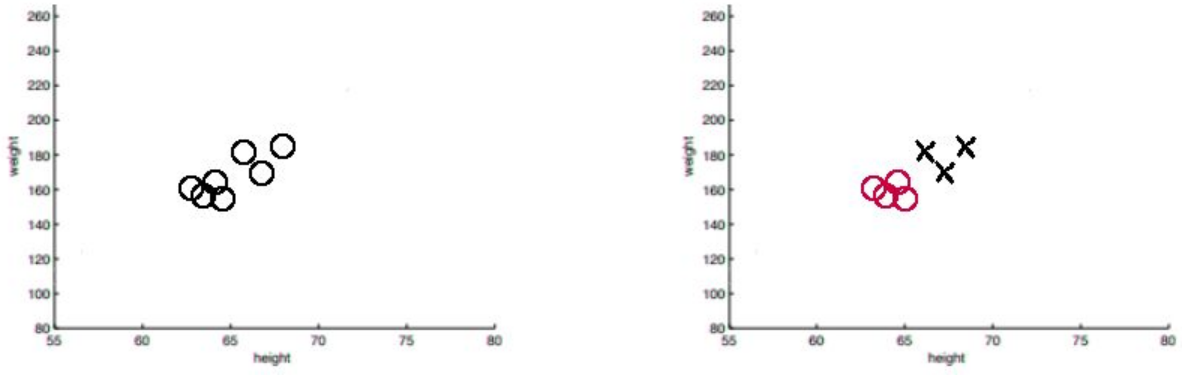


Figur 2 Visualisering av data för blommor. Det diagonala visar histogram med två olika egenskaperna. Resterande diagonaler visar plottar av olika par av egenskaper.

Figur 2.2 visar på classification, där målet är att skilja på tre olika blommor. Istället för att jobba direkt med bilddatan, har vi redan två egenskaper som speciellt bemärker denna typen av blomma. Vi har "Sepal length" och "Sepal width" som kännetecken av blomman helt enkelt. Denna "feature extraction" är en viktig uppgift, och dem flesta metoderna för machine learning använder egenskaper valda av människor. Man kan säga att feature extraction handlar om att en uppsättning rå data reduceras till mer hanterbara grupper för bearbetning. I detta fallet är dem hanterbara grupperna, kategorier som kännetecknar en blomma.

2.3 Unsupervised Learning

Med unsupervised learning kan vi få outputdata under rätt inställningar, utan någon som helst input. Målet är att hitta "intressanta strukturer" i data, vilket kallas för "knowledge discovery" (Murphy, K.P.2012). Det kan argumenteras för att unsupervised learning är mer typisk av mänsklig inlärning, eller inlärning hos djur. Det är även mer oberoende än supervised learning, då det inte finns ett krav på en mänsklig expert som manuellt märker datan. Istället fokuserar man på sambandet, och så kallade klusters. Klusters är egentligen grupperingar av data i vårt dataset, som är lika varandra.



Figur 3 (a) Höjd och vikt av några personer. (b) En möjlig klustring

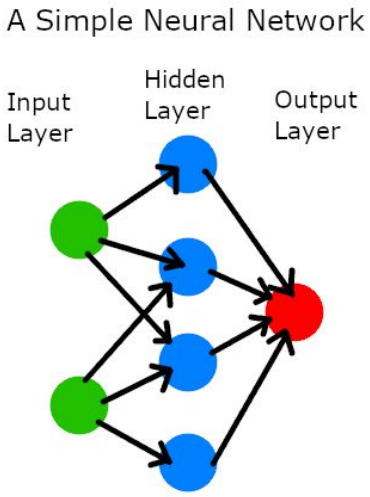
Supervised learning markerar oftast klusters i förhand, med förutbestämda namn. I exemplet av figur 2.3, så ser vi bara sambandet och mönstren. I detta exemplet markerar vi ett kluster.

2.4 Reinforcement Learning

Inom Reinforcement Learning, blir modellen som lär sig inte tillsagd vad den ska göra. Istället måste modellen upptäcka vilka handlingar som leder till bäst resultat genom att prova sig fram (Sutton, R. S. 2018). Det är dessa två egenskaper speciellt, som märker ut vad Reinforcement Learning handlar om. Reinforcement Learning ligger utom räckvidd för syftet av denna uppsats, och är inte direkt relevant för denna typ av uppgift. Istället är det Supervised Learning som är av mest intresse för arbetet.

2.5 Deep Learning

Deep Learning är en underkategori av maskininlärning, som i sin tur är en underkategori till Artificiell Intelligens. Ett artificiell neuronnät används, vilket är ett samlingsnamn på självlärande algoritmer som efterliknar funktionen i ett biologiskt neuronnät. En samling av anslutna enheter kallas för artificiella neuroner. Varje anslutning kan sedan skicka en signal till andra neuroner. Neuronen som får signaler processar sedan vidare och signalerar de neuroner som är anslutna.



Figur 4 Simplifierad vy av ett artificiellt neuralt nätverk

För att på ett enklare sätt förstå hur ett artificiellt neural nätverk fungerar, kan man likna det vid en produktionslinje. Vår input kan liknas vid ett material som förs vidare på ett rullband. Ibland stannar det för olika lager, som kan extrahera olika set av high level features. Första lagret kanske identifierar kanter, medan det andra laget

analyserar ljusstyrkan av dess pixlar. Tillslut när det sista lagret är uppnådd, har man detektorer som kan analysera komplexa kännetecken. Man kan lista ut en kombination av dessa kännetecken för att sedan ge ett värde till output lagret, och identifiera objektet.

Inom Deep Learning så finns bland annat deep neural networks, recurrent neural networks och convolutional neural networks (LeCun, Y., Bengio, Y., & Hinton, G. 2015). Convolutional neural networks, är en klass av deep neural networks.

2.6 Convolutional Neural Network

Inom Deep Learning är Convolutional Neural Network (CNN) en klass av Deep Neural Networks, och används oftast för att analysera bilder. CNN är kapabel till att uppnå rekordbrytande resultat när det kommer till objekt identifiering (Alex Krizhevsky, Ilya Sutskever och Geoffrey E. Hinton, 2012). Fördelen med CNN är att man kan upptäcka viktiga detaljer hos en bild, utan någon slags mänsklig övervakning. T.ex, om man matar in många katt eller hund bilder, så kan denna metod lära sig distinkta drag för varje klass helt själv.

2.7 Trained models

Vid användande av Convolutional Neural Network, så behöver systemet lära sig vilka objekt som är vad. För att göra detta behöver vi träna upp modeller på dataset. Tack vare modeller som VGG16, VGG19 och ResNet50, så kan vi träna upp dem för att få färdiga och kompletta resultat. Dessa modeller skiljer sig dock från varandra, med olika typer av lager och storlek hos modellen.

2.8 TensorFlow

TensorFlow är en gratis och open source mjukvara utvecklad av Google, som används för maskininlärning applikationer som Neural networks. TensorFlow innehåller en grupp av verktyg och bibliotek, som hjälper till med maskininlärning. Mjukvaran är skriven på Python, C++ och CUDA. TensorFlow används för att sätta upp våra CNN modeller, och även för att utföra mätningarna.

2.9 Flask

Flask är ett webbramverk skriven i Python, som används för att bygga webbapplikationer. Det är en av de mest populäraste webbapplikations ramverken för Python, och används för att koppla kod till webben. Det används för att skicka värdet från vår analys, till en HTML sida.

3 Problemformulering

Idag är det mer aktuellt än någonsin med Artificiell Intelligens, och dess roll i människors liv. Från livräddande mjukvara till smart homes finns det ett brett omfång av tillämpningar för tekniken. En tillämpning av AI är för att upptäcka skadligt innehåll, med hjälp utav objektidentifiering. Facebook använder sig utav AI för att göra just detta, och ta bort innehåll som anses vara skadligt (Mike Schroepfer, n.d.).

Problemet är att objektidentifiering är speciellt krävande, för en uppgift som kräver precis mätning. Precis som för livräddande utrustning finns det en väldigt liten marginal för fel, och ett högt krav på pålitlighet. Att det finns en liten marginal för fel, betyder att det är ännu viktigare att jämföra olika lösningar, för att hitta den som är mest effektiv. CNN valdes eftersom en variant av denna metod lyckades köras på en hastighet som är mer lämplig för realtime object identification, enligt Shaoqing, Kaiming, Girshick & Sun (2015). I denna studie så jämförs tre olika modeller inom metoden CNN, med hänsyn till snabbhet och träffsäkerhet. Den första modellen heter ResNet50, den andra modellen VGG19 och den tredje modellen heter VGG16.

Nollhypotesen är att ResNet50 modellen presterar bättre än VGG16 och VGG19 modellerna, när det kommer till att identifiera objekt. Alternativ hypotes är att ResNet50 modellen inte presterar bättre än VGG16 och VGG19 modellerna, när det kommer till att identifiera objekt.

Följande frågor kommer besvaras i denna studie:

1. Hur snabbt kan modellerna identifiera objekt?
2. Hur träffsäkert kan modellerna identifiera objekt?
3. Vilken modell är mest effektiv?
4. Hur fungerar modellen i en webblösning?

För att svara på frågorna behöver vi ha möjlighet att implementera vissa delar:

1. Sätt upp ett CNN som är basis för att kunna analysera bilder och identifiera objekt
2. Implementera ResNet50, VGG19 och VGG16 modellerna
3. Skapa en webblösning som kan markera ut inmatade bilder med text

4 Metod

Metoden på detta arbetet är av typen experiment, med en empirisk mätning. Mätningarna behöver ske under identiska förhållanden, vilket fungerar speciellt bra under typ experiment. En empirisk mätning görs, eftersom vi vill kunna mäta skillnader i svarstider och noggrannhet. CIFAR-10 datasetet används som data för att identifiera objekten, dvs alla bilder som kommer analyseras och klassificeras. CIFAR-10 är ett märkt subset av 80 million tiny images datasetet, som var skapad av Alex Krizhevsky, Vinod Nair, och Geoffrey Hinton. CIFAR-10 datasetet innehåller 60 000 märkta bilder, som är placerade i 10 olika kategorier (Alex Krizhevsky).

Två typer av mätvärden finns. Först ett mätvärde som räknar ut längden på hela processen och tiden att gå igenom hela datasetet. Det andra värdet mäter träffsäkerhet, för att visa hur likt något är alla klasser sammanlagt. Detta värdet ger oss en idé hur noga mätningen är. En del av implementationerna kräver att vi har en webblösning för att presentera hur modellen fungerar i praktiken. En webbsida skapas där användaren kan ladda upp vilken bild som helst, där modellen märker ut bilden med text. Denna texten beskriver vilken av dem 10 kategorierna som bilden är mest lik.

4.1 Alternativa metoder

Enligt Wohlin m.fl. (2012), finns det två ytterligare metoder förutom experiment. Dessa metoder är fallstudie, och enkätstudie. Till skillnad från experiment som är en kontrollerad studie, är fallstudie en observationsstudie där flera evidenskällor används för att undersöka en teori. Fallstudier används ofta för att följa eller etablera en relation mellan olika attribut, och har en lägre nivå av kontroll i jämförelse av ett experiment (Wohlin m.fl (2012)). Eftersom detta arbetet kräver en hög nivå av kontroll för vår testmiljö och testfaktorer, blir alltså experiment mer lämplig metod att använda.

En enkätstudie är en studie som oftast används i retrospekt, när t.ex ett verktyg eller teknik har använts ett tag. Det primära sättet att samla data är genom kvantitativ eller kvalitativ data, som t.ex intervjuer eller enkäter (Wohlin m.fl (2012)). Detta görs genom att ett prov tas som representerar den population som ska studeras. Resultatet av enkäten blir sedan analyserat för att dra förklarande slutsatser. Eftersom denna uppsatsen handlar om teknik som ska utvärderas på ett tekniskt sätt, blir det mindre

relevant med någon slags användarutvärdering kring funktionalitet. Att utvärdera svarstider genom experiment, blir mer lämpligt än genom åsikter.

4.2 Etiska aspekter

Eftersom studien är väldigt driven av data, så kan det finnas bekymmer när det kommer till människors integritet. CIFAR-10 innehåller inga bilder på människor, och endast 10 klasser används. CIFAR-10 är ett subset av tiny images dataset, bilder som är samlade av Alex Krizhevsky, Vinod Nair, och Geoffrey Hinton (Alex Krizhevsky).

Ingen människa eller djur blev skadade eller berörda som resultat av denna studie. Allt bildmaterial som presenterats i denna studie är mitt egna, och berör inget copyrights skydd.

5 Genomförande

5.1 Litteraturstudie

För att enklare utveckla och använda CNN modeller, så används Keras för att simplifiera processen av att skriva kod. Keras är ett open-source neural-network bibliotek skrivet i python, och är ett väldigt populärt alternativ. Det används ofta när det kommer till bygga, träna, evaluera och köra neural networks, men är även flexibel nog för att utveckla ett brett utbud av neural network architectures med (Géron, A. (2019)).

En inspirationskälla för studien var en artikel från Facebook (Wu, Carole-Jean, et al.(2019)), som beskriver användandet av AI för bl.a object detection. Artikeln beskriver hur machine learning förser ett brett utbud av möjligheter som driver user experience, som med att ranka posts, förstå innehåll och object detection. Machine Learning används av de mesta Facebook services (Wu, Carole-Jean, et al.), där alla varianter av machine learning modeller används.

5.2 Litteraturssökning

I detta kapitlet kommer litteratur för verktygen som användes för uppgiften att listas, men också en kort beskrivning över vad man kan förvänta sig från att ta del av denna information.

- **TensorFlow**

TensorFlow Guide - Den officiella dokumentationen för TensorFlow kan hittas på addressen https://www.tensorflow.org/guide. Webbsidan ger en ganska omfattande dokumentation angående TensorFlow. T.ex finns det information om hur man installerar TensorFlow, hur man sparar en modell, bästa sätt att optimera för bättre prestanda, m.m. I dagsläget är den version som använts för projektet TensorFlow 2.2.

- **Keras**

Keras Overview - En enkel att förstå och snabb överblick över Keras, kan hittas på addressen https://www.tensorflow.org/guide/keras/overview. Man går igenom bland annat hur man bygger en enkel modell, till hur man tränar ,evaluerar, sparar och återställer en modell. Bra om man vill ha en grundlig förståelse över hur Keras fungerar.

- **MatPlotLib**

MatPlotLib Documentation - Den officiella dokumentationen för MatPlotLib hittas på addressen https://matplotlib.org/contents.html, och ger en väldigt omfattande bild av hur MatPlotLib fungerar. Innefattar allt från installation, till API och moduler. Beskriver det mesta om parametrar och liknande, och är väldigt användbar.

5.3 Implementation

I den första iterationen av projektet, implementerades färdigtränade modeller med hjälp utav Tensorpack. Efter att en del problem kom upp med att skapa mätningsdatan, bestämdes det istället att skapa ett nytt github projekt. Själv tränade modeller innebär mer mätningsdata om träningsprocessen, och en enklare implementation med mindre problem i överlag.

Kod skapades för att kunna träna upp en modell, enligt dataset CIFAR10[1]. Detta skedde under python med modulen tensorflow, som är speciellt anpassat för att hantera AI kod. Genom processen av att träna upp modellen, mäts träffsäkerheten mot validation data, men också training data. Denna processen upprepas i cykler (epochs), för att fortsätta träna modellen att känna igen data.

```
def unpickle(file, encoding='bytes'):
    with open(file, 'rb') as f:
        di = pickle.load(f, encoding=encoding)
    return di
print(os.listdir("cifar-10-batches-py"))

batches_meta = unpickle(f"cifar-10-batches-py/batches.meta",
encoding='utf-8')
label_names = batches_meta['label_names']

batch_labels = []
batch_images = []
```

Figur 5 Python kod för uppackning av dataset

[1] https://github.com/b17albbe/CNN/commit/12a5b2f4aa5eb8ca4890e4fe11f89cffbaeeea1f

```
for n in range(1, 6):
    batch_dict = unpickle(f"cifar-10-batches-py/data_batch_{n}")
    # Add labels to the list of batch labels
    batch_labels.append(batch_dict[b'labels'])

    # Load the images, and resize them to 10000x3x32x32
    data = batch_dict[b'data'].reshape((10000, 3, 32, 32))
    # Modify axis to be 10000x32x32x3, since this is the correct
order for keras
    data = np.moveaxis(data, 1, -1)
    batch_images.append(data)

labels = np.concatenate(batch_labels, axis=0)
images = np.concatenate(batch_images, axis=0)

test_dict = unpickle(f"cifar-10-batches-py/test_batch")
test_labels = np.array(test_dict[b'labels'])
test_images = test_dict[b'data'].reshape((10000,3,32,32))
test_images = np.moveaxis(test_images, 1, -1)
```

Figur 6 Python kod för inläsning av bilder

```
# We normalize the input according to the methods used in the paper
X_train = preprocess_input(images)
y_test = to_categorical(test_labels)

# We one-hot-encode the labels for training
X_test = preprocess_input(test_images)
y_train = to_categorical(labels)
```

Figur 7 Python kod för tilldelning av testdata och träningsdata

```
# Train the model
#history = model.fit(
#    x=X_train,
#    y=y_train,
#    validation_split=0.1,
#    batch_size=256,
#    epochs=30,
#    callbacks=[checkpoint],
#    verbose=1
#)
```

Figur 8 Python kod för träning av modell

Accuracy data sparas i en fil, som sedan visualiseras genom matplotlib. Eftersom träning av modeller tar lång tid, valdes 30 som värdet för mängder epoker, som en bra balans mellan tid och statistisk precision.

- **Webblösning**

För att genomföra webblösningen, så skapades kod för att analysera bilder, och ge ut ett värde[2]. Detta kopplades sedan till en webbsida med hjälp utav flask, som tillåter användaren att ladda upp bilder som input till CNN. Värdet från analysen kan sedan visas på webbsidan, och markera ut bilden.

Figur 9 Webblösning för analys av bild

```
# load an image from file
    image = load_img('static/img.png', target_size=(32, 32))
```

[2] https://github.com/b17albbe/CNN/commit/2ad064e69206b41b6ce656da8a0c5d34395b721c

```
    # convert the image pixels to a numpy array
    image = img_to_array(image)
    # reshape data for the model
    image = image.reshape((1, image.shape[0], image.shape[1],
image.shape[2]))
    # prepare the image for the VGG model
    image = preprocess_input(image)
    # predict the probability across all output classes
    yhat = model.predict(image)
```

Figur 10 Python kod för analys av bild

Värdet från bildanalysen, visar på om hur nära bilden är någon av dem 10 klasserna som modellen är upptränad på. Namnet på den klass som bilden är närmast, skickas vidare till Flask implementationen. Flask lösningen tar värdet, och skriver ut genom en vanlig HTML sida.

```
<!doctype html>
        <title>Upload new File</title>
        <h1>Upload new File</h1>
      <h1> Image is: {{ classification }} </h1>
        <form method=post enctype=multipart/form-data>
          <input type=file name=file>
          <input type=submit value=Upload>
        </form>
```

Figur 11 Python kod för HTML

5.4 Progression

Genom arbetets gång, så har en hel del ändrats när det kommer till genomförandet. Som sagt tidigare så var det från början av genomförandet, tänkt att använda pretrained models, för att undvika behöva träna upp modellerna och därmed spara tid. Detta var från den första iterationen av github projektet, och kan ses i filen vgg.py [3]. Implementationen av pre trained models och mätningen, orsakade många problem, och därför beslutades det att träna upp modellerna manuellt istället. Bland annat var det begränsat med pretrained models för att kunna mäta data, då man inte kunde mäta träningsprocessen. Dessutom fanns det problem med att jämföra modeller, genom att använda ett sånt stort dataset, som alla pretrained models var upptränade på. Eftersom modellerna var upptränade på så många bilder, fanns det väldigt många

[3] https://github.com/b17albbe/imageAI/commit/bd0a9e956a4e9782c3bf24c824b0a22cf8298e04

kategorier. Om man skulle jämföra precision mellan kategorier, skulle det därför bli krångligt med antalet av dessa.

Varav beslutet att träna upp modellerna, byttes också datasetet till CIFAR10 i ett nytt projekt[4]. Detta var för att spara tid i träningsprocessen, då CIFAR10 är ett mycket mindre dataset i jämförelse med ImageNet, datasetet som var tänkt att användas från början. Med mindre dataset finns det mindre bilder, som modellen måste analysera och träna sig upp på.

5.5 Pilotstudie

För att säkerställa att den koden som skrivs fungerar som den ska, så genomfördes en pilotstudie. Detta tillåter oss att bara mäta en modell, för att verkställa att mätningarna fungerar. Därför använder jag mig bara utav VGG16 modellen, då de olika modellerna fungerar på liknande sätt. Accuracy av image classification, samt tiden att utföra operationen, mäts för att verkställa att mätning metoderna fungerar. För den senare kompletta undersökningen, så kommer experimentet utföras med alla modeller, för att få en större jämförelse mellan dem olika modellerna.

För att beräkna hur snabbt modellen kan analysera bilderna, så används funktionen time. Tiden som mäts är från när bilden laddas in, till när probabiliteten har räknas ut, i.e classification.

```
start = time.time()
```

Figur 12 Python kod för start av tidtagning

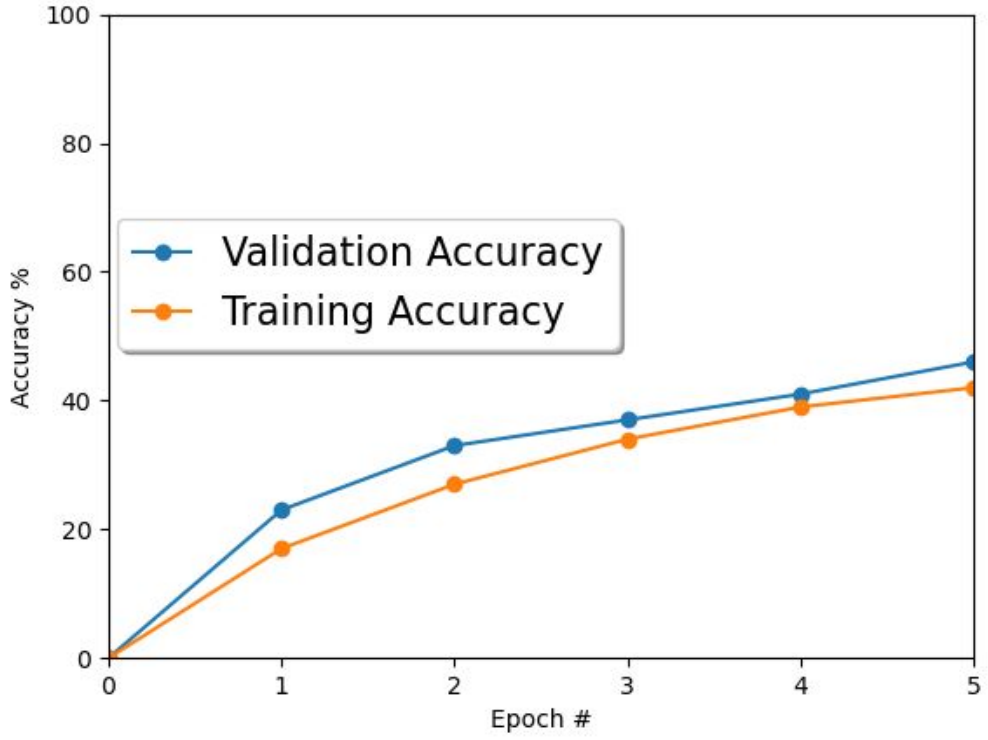
[4] https://github.com/b17albbe/CNN/commit/12a5b2f4aa5eb8ca4890e4fe11f89cffbaeeea1f

```
# Train the model
#history = model.fit(
#    x=X_train,
#    y=y_train,
#    validation_split=0.1,
#    batch_size=256,
#    epochs=5,
#    callbacks=[checkpoint],
#    verbose=1
#)
```

Figur 13 Python kod förupptäning av modell

```
done = time.time()
elapsed = done - start
print(elapsed)
```

Figur 14 Python kod för stopp av tidtagning, och utskrivning av tid



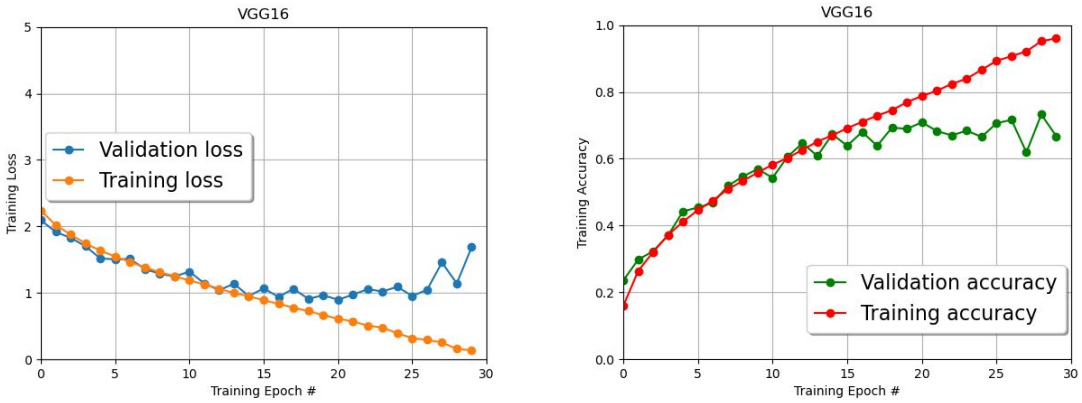
Figur 15 Graf av pilotstudie mätning

6 Resultat

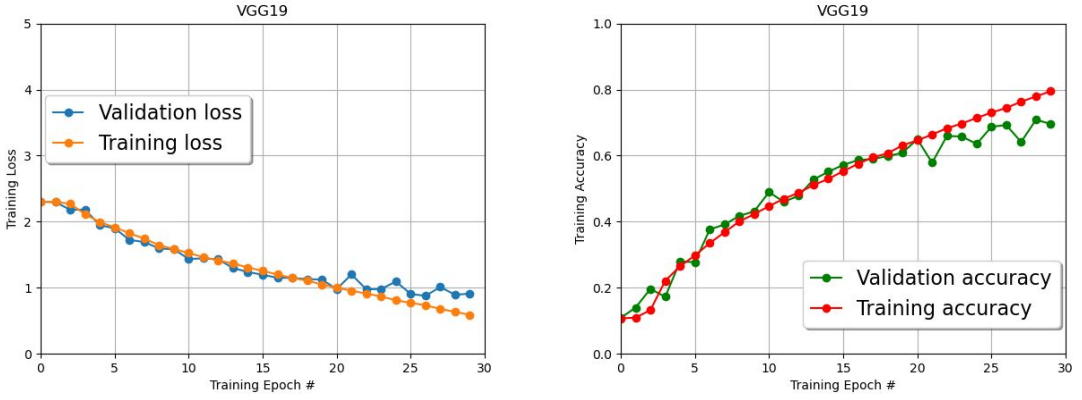
Detta kapitlet kommer att analysera och presentera det resultat som samlades i studien. Resultatet för CNN modellerna kommer presenteras genom grafer, därefter presenteras även tiderna för varje mätning.

6.1 Presentation av data

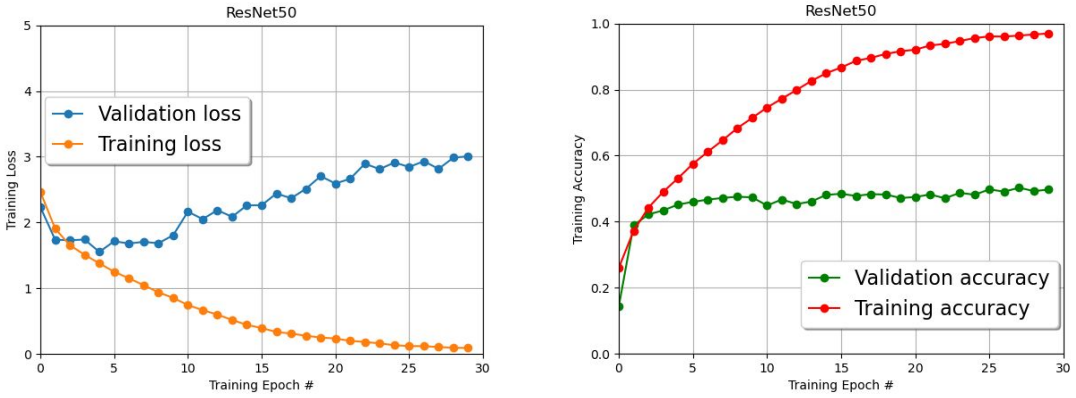
Datan från mätningarna visualiserades med grafer, och presenteras i detta under kapitlet. Graferna summerar träffsäkerhet genom träningen samt förlust, och en epok representerar en genomgång av det dataset som algoritmen går igenom.



Figur 16 VGG16 - Förlust och träffsäkerhet för varje epok genom träning.



Figur 17 VGG19 - Förlust och träffsäkerhet för varje epok genom träning.



Figur 18 ResNet50 - Förlust och träffsäkerhet för varje epok genom träning.

Tidsmätning från kompilering av modell, till träning av modell:

Modell	**Tid (sekunder)**
VGG16	40240.33609557152 (cirka 11 timmar)
VGG19	50915.91348218918 (cirka 14 timmar)
ResNet50	52851.55630111694 (cirka 14,5 timmar)

Mätning av test data och tid från kompilering till mätning:

VGG16	
Train Loss	0.1701684801992774
Test Loss	1.1977082255363465
Train Score	0.95984
Test Score	0.7122
Time	167.70080089569092 sekunder (avrundat till 2,795 minuter)

VGG19	
Train Loss	0.5430518135166168
Test Loss	0.9425076128005981
Train Score	0.80884
Test Score	0.6921
Time	195.11174058914185 sekunder (avrundat till 3,251 minuter)

ResNet50	
Train Loss	0.36202923179402946
Test Loss	2.8646291175842284
Train Score	0.92668
Test Score	0.5022
Time	94.26331377029419 sekunder (avrundat till 1,571 minuter)

- **Träningsmätning**

Figur 6.1 genom 6.3, visar på både tränings och validering träffsäkerhet, samt tränings och validerings förlust. VGG19 uppnådde den lägsta tränings träffsäkerhet, men hade en validerings träffsäkerhet som blev högst mellan modellerna. Detta blir mer tydligt när vi ser till förlust graferna. Där ser vi att VGG16 och ResNet50

upplever en större varians mellan tränings och validering förlust, än VGG19 som håller sig relativt stabilt och lågt.

ResNet50 visar på den högsta tränings träffsäkerheten, men har en låg validerings säkerhet. Ser vi till förlust grafen i figur 6.3, visas skillnaden tydligt. Förlusten för träning blir lägre, medan förlusten för validering planar tidigt ut och blir högre. VGG16 lider av något liknande, där tränings träffsäkerhet stiger, men validerings träffsäkerhet planar ut. Skillnaden är dock mindre, och validerings träffsäkerheten är högre.

- **Testmätning**

När det kommer till träffsäkerhet i testdatamätningen, så har VGG16 den högsta träffsäkerheten för både träningsdata och testdata. Näst högst träffsäkerhet inom träningsdata har ResNet50, men har också den lägsta test data träffsäkerheten med 0.5022. VGG19 har den näst högsta träffsäkerheten med test data, och den lägsta för träningsdata.

- **Tidsmätning**

Tidsmätningen visar på att den snabbaste modellen på att bli tränad är VGG16, medan den långsammaste modellen på att bli tränad är ResNet50. VGG19 har ett liknande resultat till ResNet50, men är lite snabbare. Ser vi till tidsmätningen av test datan, så har ResNet50 den snabbaste tiden med 94 sekunder, medan VGG19 erhåller den långsammaste tiden med 195 sekunder. VGG16 håller sig i mitten med 167 sekunder.

6.2 Analys

Mätningar i figur 6.3, visar en tydlig korrelation för ResNet50 modellen, där en hög träffsäkerhet för träningsdata, inte innebär en hög träffsäkerhet för valideringsdata. Figur 6.1 och 6.2 visar däremot ett något liknande samband, där träffsäkerheten för valideringsdata till störst del följer med träningsdata. Detta med ett litet undantag av VGG16, som har en validerings träffsäkerhet som börjar plana ut i slutet av mätningen. Ser vi till testdata mätningen har vi ett liknande sammanband mellan modeller, där VGG modellerna har högre validerings träffsäkerhet, varav ResNet50 har den lägsta.

När det kommer till tidsmätningar så finns det en varians av resultat när det kommer till testdata mätningarna, bland dem tre olika modellerna. Den långsammaste mätningen var från VGG19 med drygt 3 minuter, och tog dubbelt så lång tid att genomföra än med ResNet50. VGG16 hade en något snabbare mätning än VGG19, med drygt 2 minuter och 40 sekunder. ResNet50 tidsmätningen tog bara drygt 1 och en halv minut att genomföra, och var den snabbaste mätningen för testdatan.

Tidsmätningarna för träningen, visar dock en annorlunda bild. Den modell som tog längst tid för att träna upp var ResNet50, med drygt 14 och en halv timme från kompilering till uppträning. Detta följs upp av VGG19, som tog drygt 14 timmar att träna upp, varav VGG16 hade den kortaste tiden för att träna upp, med drygt 11 timmar.

Möjligen är det så att ResNet50 tar en längre tid att träna upp, men är mycket snabbare när det kommer till att analysera data. Detta verkar inte vara fallet för VGG19, som har både en långsam testdata mätning, men också en långsam träningsdata mätning. VGG16 som hade en snabb uppträningsprocess, har istället en mycket långsammare testdata mätning i jämförelse med ResNet50.

7 Diskussion

Detta kapitel kommer gå igenom en sammanfattning på alla upptäckter inom studien, och går även igenom tankar om etik, samhälle, och framtida arbeten.

7.1 Sammanfattning

Målet av denna studien var att lista ut hur snabbt modellerna kan identifiera objekt, hur träffsäkert modellerna kan identifiera objekt, och slutligen vilken modell som var mest effektiv.

Modellerna visar sig vara effektiva på olika områden av mätningarna. ResNet50 är snabbast för att mäta testdata, men är även den modell med lägst träffsäkerhet för valideringsdata och testdata, och är inte pålitlig nog för att markera ut bilder. VGG19 visar på den långsammaste mätningen för testdata, men den högsta träffsäkerheten för valideringsdata, nära uppföljd av VGG16. För allt som VGG19 kan göra bra, kan VGG16 göra nästan lika bra, utan dem större problem som kommer med det, t.ex långsammare uppträning och analys av data.

Jag tycker resultatet var speciellt överraskande, då jag inte förväntade att ResNet50 skulle ha en sådan låg träffsäkerhet för valideringsdata och testdata. Jag förväntande inte heller att ResNet50 skulle vara långsammast att träna upp, men samtidigt vara den snabbaste modellen i testdata mätningen.

Hypotesen löd följande “ResNet50 modellen presterar bättre än VGG16 och VGG19 modellerna, när det kommer till att identifiera objekt.”. Baserat på resultaten från detta experimentet, så går det att redogöra att hypotesen inte stämmer, varav den alternativa hypotesen att “ResNet50 modellen inte presterar bättre än VGG16 och VGG19 modellerna, när det kommer till att identifiera objekt.” stämmer. Nollhypotesen är motbevisad, och den alternativa hypotesen accepteras. Forskningsfrågan har därmed besvarats.

7.2 Etik och samhälle

Wohlin m.fl. (2012) tar upp en rad hot för validering, för bättre forskningskvalitet. Bland annat är en låg statistisk kraft ett hot mot validering, då det är den statistiska kraften som kan avslöja ett riktigt mönster i data. I detta fallet använder sig arbetet av en stor mängd data, när det kommer till analysen och uppträning av modellen. Ett annat hot för validering för experimentet är pålitligheten av åtgärderna som används. Detta kan bero på flera faktorer, som bl.a dålig frågeformulering eller dålig instrumentation. Eftersom vi använder oss av programmeringskod, blir det mer pålitligt, då vi inte behöver involvera mänsklig bedömning.

Med utveckling av automation, så kan det bli möjligt att en maskin blir ett mer attraktivt val till att utföra uppgifter, i jämförelse med människor. Detta skulle leda till att människor blir av med sina arbeten, vilken i sin tur leder till att dem blir av med sin lön, sjukvård och liknande. Detta är ett etisk aspekt som är värd att tänka på, när man utvecklar och gör automation mer effektiv.

Möjligheten till att analysera bilder och identifiera objekt, gör att det är möjligt att bli av med skadligt innehåll inom bl.a social media, vilket kan bidra till samhällsnyttan. I ett brett sammanhang kan objekt identifikation kopplas till en mängd samhällsnytta. Möjligheten till att se världen omkring oss utan ögon t.ex, kan bl.a hjälpa människor med nedsatt syn. Det kan även hjälpa utvecklingen med självkörande bilar, vilket jag också ser som en samhällsnytta.

7.3 Relaterat arbete

Det finns en annan studie som undersöker liknande faktorer, och dessutom med några modeller som är samma som för denna studien. Enligt Lee, C., Kim, H.J., & Oh, K.W. (2016), hittar man ett samband mellan träffsäkerhet och hastighet. Mer specifikt så finner man att ResNet50 tar längre tid på att träna och analysera bilder, varav VGG16 är snabbast, men har en lägre träffsäkerhet. En del av detta går emot resultatet i denna uppgiften, där ResNet50 har en lägre träffsäkerhet. Möjligen beror detta på att datasetet är mindre i storlek, vilket är sämre anpassat för något som ResNet50. En mer komplicerat modell kan vara mer passande till en högre resolution, och vice versa.

Det finns en till studie (M. M. Yapıcı, A. Tekerek and N. Topaloğlu (2019)) som undersöker prestanda mellan CNN modeller, bl.a VGG19 och ResNet50. Studien finner att den bästa prestandan går till DenseNet121, medan den näst bästa prestandan går till ResNet50. Den sämsta prestandan går till VGG19, som har ett ostabilt resultat, hög minne användning, och låg träffsäkerhet. Detta går även emot resultatet från studien, där ResNet50 har lägst träffsäkerhet, men VGG19 har bättre träffsäkerhet.

7.4 Framtida arbete

Det här arbetet har öppnat upp en hel del dörrar, när det kommer till framtida arbeten. I denna studien har bara convolutional neural network modeller mätts inom träffsäkerhet och tider. Men vid framtida arbeten är det fullt möjligt att undersöka flera typer av AI modeller och implementationer, och jämföra dem med varandra, för att hitta en mer effektiv lösning.

Genom att skapa webblösningen, känner jag att det finns så mycket mer möjligheter till implementationer som går att göras. I detta arbete var det enbart en bild som analyserades, men det är mycket möjligt att göra detta med ett mycket större antal bilder. Kanske är det möjligt att skapa en lösning för en hel webbsida, med ett mycket större dataflöde.

Jag tror också det hade varit intressant att utforska AI för områden som inte är enbart webbaserat, t.ex AI inom självkörande bilar eller medicin. Det är möjligt att en jämförelse mellan modeller i ett annat område, har annat krav på hastighet eller träffsäkerhet. Det hade varit intressant att jämföra modeller på enbart träffsäkerhet, om området prioriterade detta t.ex.

Referenser

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436-444.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

Schroepfer, M. New Progress in using AI to detect harmful content. Hämtad från

https://ai.facebook.com/blog/community-standards-report/

Ren, S., He, K., Girshick, R. and Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NIPS

Krizhevsky, A. CIFAR-10 and CIFAR-100 datasets. Hämtad från https://www.cs.toronto.edu/~kriz/cifar.html

Host, M., Ohlsson, M. C., Regnell, B., Runeson, P., Wesseln, A., & Wohlin, C. (2012). *Experimentation in software engineering*. Springer-Verlag Berlin.

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media.

Wu, C. J., Brooks, D., Chen, K., Chen, D., Choudhury, S., Dukhan, M., ... & Leyvand, T. (2019, February). Machine learning at facebook: Understanding inference at the edge. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (pp. 331-344). IEEE.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural Networks, 61, pp.85–117.

Lee, C., Kim, H.J., & Oh, K.W. (2016). Comparison of faster R-CNN models for object detection. 2016 16th International Conference on Control, Automation and Systems (ICCAS), 107-110.

Appendix A - othercifar.py

```
import os
import json
import pickle
import time

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def unpickle(file, encoding='bytes'):
    with open(file, 'rb') as f:
        di = pickle.load(f, encoding=encoding)
    return di

print(os.listdir("cifar-10-batches-py"))
batches_meta = unpickle(f"cifar-10-batches-py/batches.meta", encoding='utf-8')
label_names = batches_meta['label_names']

batch_labels = []
batch_images = []

for n in range(1, 6):
    batch_dict = unpickle(f"cifar-10-batches-py/data_batch_{n}")
```

```
    batch_labels.append(batch_dict[b'labels'])


    data = batch_dict[b'data'].reshape((10000, 3, 32, 32))

    data = np.moveaxis(data, 1, -1)
    batch_images.append(data)

labels = np.concatenate(batch_labels, axis=0)
images = np.concatenate(batch_images, axis=0)

test_dict = unpickle(f"cifar-10-batches-py/test_batch")
test_labels = np.array(test_dict[b'labels'])
test_images = test_dict[b'data'].reshape((10000,3,32,32))
test_images = np.moveaxis(test_images, 1, -1)

fig = plt.figure(figsize=(14,10))

for n in range(1, 29):
    fig.add_subplot(4, 7, n)
    img = images[n]
    plt.imshow(img)
    plt.title(label_names[labels[n]])
    plt.axis('off')
#plt.show()
```

```
start = time.time()

from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.utils import to_categorical

# We normalize the input according to the methods used in the paper
X_train = preprocess_input(images)
y_test = to_categorical(test_labels)

# We one-hot-encode the labels for training
X_test = preprocess_input(test_images)
y_train = to_categorical(labels)

from tensorflow.keras.applications.vgg16 import VGG16

model = VGG16(
   weights=None,
   include_top=True,
   classes=10,
   input_shape=(32,32,3)
)

# Expand this cell for the model summary
#model.summary()

from tensorflow.keras import optimizers
```

```
model.compile(
    loss='categorical_crossentropy',
    optimizer='sgd',
    metrics=['accuracy']
)

from tensorflow.keras.callbacks import ModelCheckpoint

checkpoint = ModelCheckpoint(
    'model.h5',
    monitor='val_accuracy',
    verbose=0,
    save_best_only=True,
    save_weights_only=False,
    mode='auto'
)

# Train the model
history = model.fit(
    x=X_train,
    y=y_train,
    validation_split=0.1,
    batch_size=256,
    epochs=30,
    callbacks=[checkpoint],
    verbose=1
)
```

```
done = time.time()
elapsed = done - start
print(elapsed)

#with open('history.json', 'w') as f:
#  json.dump(history.history, f)

#history_df = pd.DataFrame(history.history)
#history_df[['loss', 'val_loss']].plot()
#history_df[['acc', 'val_accuracy']].plot()

model.load_weights('model.h5')
train_loss, train_score = model.evaluate(X_train, y_train)
test_loss, test_score = model.evaluate(X_test, y_test)
print("Train Loss:", train_loss)
print("Test Loss:", test_loss)
print("Train F1 Score:", train_score)
print("Test F1 Score:", test_score)
done2 = time.time()
elapsed2 = done2 - start
print(elapsed2)
```

Appendix B - pyplotloss.py

```
import matplotlib.pyplot as plt

plt.plot([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29], [2.2342, 1.7394, 1.7300, 1.7413, 1.5528, 1.7151, 1.6809, 1.7060, 1.6852, 1.8058, 2.1651, 2.0521, 2.1870, 2.0871, 2.2616, 2.2631, 2.4404, 2.3695, 2.5121, 2.7049, 2.5930, 2.6647, 2.8941, 2.8117, 2.9125, 2.8452, 2.9311, 2.8159, 2.9865, 3.0057], marker='o')

plt.plot([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29], [2.4682, 1.9074, 1.6552, 1.5078, 1.3794, 1.2506, 1.1533, 1.0469, 0.9393, 0.8534, 0.7451, 0.6681, 0.5995, 0.5186, 0.4427, 0.3956, 0.3349, 0.3114, 0.2776, 0.2516, 0.2354, 0.2013, 0.1835, 0.1621, 0.1341, 0.1214, 0.1195, 0.1063, 0.0959, 0.0923], marker='o')

plt.axis([0, 30, 0, 5])

plt.ylabel('Training Loss')

plt.xlabel('Training Epoch #')

plt.grid(True)


plt.title('ResNet50')


# legend

plt.legend(('Validation loss', 'Training loss'),

        shadow=True, loc=(0.01, 0.58), handlelength=1.5, fontsize=16)


plt.show()
```

Appendix C - app.py

```
def analyze():
    from tensorflow.keras.preprocessing.image import load_img
    from tensorflow.keras.preprocessing.image import img_to_array
    import time
    from tensorflow.keras.applications.vgg16 import VGG16
    from tensorflow.keras.applications.vgg16 import preprocess_input
    import numpy as np
    from tensorflow.keras.applications.vgg16 import preprocess_input, decode_predictions

    model = VGG16(
        weights=None,
        include_top=True,
        classes=10,
        input_shape=(32, 32, 3)
    )

    model.compile(
        loss='categorical_crossentropy',
        optimizer='sgd',
        metrics=['accuracy']
    )

    # load the model
    model.load_weights('model.h5')
```

```
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.applications.vgg16 import preprocess_input

# load an image from file
image = load_img('static/img.png', target_size=(32, 32))
# convert the image pixels to a numpy array
image = img_to_array(image)
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# prepare the image for the VGG model
image = preprocess_input(image)
# predict the probability across all output classes
yhat = model.predict(image)

data = np.array2string(yhat)

datareplace = str(data).replace('[', '')
datareplace2 = str(datareplace).replace(']', '')

values = datareplace2.split()

spot = values.index(max(values))

if spot == 0:
    classification = 'airplane'
```

```
    elif spot == 1:
        classification = 'automobile'
    elif spot == 2:
        classification = 'bird'
    elif spot == 3:
        classification = 'cat'
    elif spot == 4:
        classification = 'deer'
    elif spot == 5:
        classification = 'dog'
    elif spot == 6:
        classification = 'frog'
    elif spot == 7:
        classification = 'horse'
    elif spot == 8:
        classification = 'ship'
    elif spot == 9:
        classification = 'truck'
    else:
        classification = 'error'

    return classification
```

```
from flask import Flask, render_template, redirect, url_for, request
```

```
# Route for handling the login page logic
app = Flask(__name__)

import os
from flask import Flask, flash, request, redirect, url_for
from werkzeug.utils import secure_filename

UPLOAD_FOLDER = 'static/'
ALLOWED_EXTENSIONS = set(['txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif'])

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def allowed_file(filename):
   return '.' in filename and \
         filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS



@app.route('/', methods=['GET', 'POST'])

def upload_file():
   classification = analyze()
   if request.method == 'POST':
      # check if the post request has the file part
      if 'file' not in request.files:
         flash('No file part')
         return redirect(request.url)
```

```
    file = request.files['file']
    # if user does not select file, browser also
    # submit an empty part without filename
    if file.filename == '':
        flash('No selected file')
        return redirect(request.url)
    if file and allowed_file(file.filename):
        filename = "img.png"
        file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
        return redirect(url_for('upload_file',
                                filename=filename))
return render_template('index.html', classification=classification)
```

Appendix D - index.html

```
<!doctype html>
    <title>Upload new File</title>
    <h1>Upload new File</h1>
        <h1> Image is: {{ classification }} </h1>
    <form method=post enctype=multipart/form-data>
      <input type=file name=file>
      <input type=submit value=Upload>
    </form>
```