Klassifisering av bilder med nevrale nettverk

Siv Aalbergsjø

Dette prosjektet tar sikte på å bli kjent med ulike maskinlæringsmetoder for bildegjenkjenning. Det er blitt testet og sammenlignet ulike metoder for klassifisering av bildedata med mål om å forstå hvorfor noen av metodene er bedre til dette enn andre. Det er benyttet logistisk regresjon, og ulike arkitekturer for forovermatede nevrale nettverk (feed forward nevrale nettverk) (FFNN) og konvolusjonsnettverk (CNN). De ulike metodene og arkitekturene har vært testet ut på datasettet MNIST som er en samling håndskrevne siffer fra 0 - 9 og Fashion MNIST som er bilder av klesplagg. Som forventet presterte CNN bedre enn logistisk regresjon og FFNN på bildegjenkjenningsproblemet. Det ble funnet at logistisk regresjon ga en nøyaktighet på 96, 97% for MNIST, mens den beste FFNN med to skjulte lag med 500+150 noder ga en nøyaktighet på 97, 99% og det beste CNN-nettet med to konvolusjonslag med 8 + 16 filtere med påfølgende 2 skjulte koblede lag med 500+150 noder ga en nøyaktighet på 99, 02%. For Fashion MNIST ble det også funnet at CNN-metoden ga de beste resultene, med en oppnådd nøyaktighet på på 90, 56% for det største konvolusjonsnettverket. Det er sannsynlig at et større CNN tilpasset Fashion MNIST ville gitt en bedre modell for dette problemet.

I. INTRODUKSJON

Bildegjenkjenning er teknologi som har gjort store fremskritt og blitt stadig mer utbredt. Vi er blitt vant til at telefonene våre kan gjenkjenne elementer i bilder for oss, til og med kan gjenkjenne personene på bildene. Andre apper kan brukes til å gjenkjenne tekst og oversette denne for oss ved behov. I denne oppgaven skal jeg ta for meg noen av metodene som kan brukes for å gjenkjenne elementer i bilder gjennom såkalt overvåket læring, dvs. at metodene trenes på datasett hvor vi kjenner til hva som fines i bildene fra før med mål om å bli kjent med hvorfor noen maskinlæringsmetoder er bedre egnet til dette enn andre. Metodene som testes ut er logistisk regresjon, forovermatet nevralt nettverk (feed forward nevrale nettverk) (FFNN) og konvolusjonsnettverk (CNN). Disse er eksempler på klassifiseringsmetoder, dvs. at det skal skilles mellom fler enn to ulike typer objekter/elementer i bildene.

Hensikten med denne oppgaven er å undersøke hvordan oppbygningen til bildegjenkjenningsmetodene påvirker hvor godt de løser sin oppgave. Metodene testes ut på et relativt enkelt problem, nemlig gjenkjenning av håndskrevne tall mellom 0 og 9, hvor hvert bilde kun består av ett siffer. Metodene testes ut og det sammenlignes ulike arkitekturer for de nevrale nettverkene for MNIST datasettet. Deretter testes metodene på det noe mer komplekse problemet som består av gjenkjenning av klesplagg i bilder, men fortsatt kun med ett klesplagg i hvert bilde.

I denne rapporten vil jeg først gjøre rede for metodene som er benyttet i klassifiseringen, styrker og svakheter ved disse og hvordan man bruker disse til å bygge modeller som kan gjenkjenne elementer i bilder. Deretter beskriver jeg datasettene som er benyttet i treningen av modellene og resultater fra andres utprøving med disse datasettene, før jeg beskriver min implementeringen av metodene. Deretter gjør jeg rede for utprøving de ulike metodene og ulike arkitekturer i de nevrale nettverkene på problemet med sifrene 0-9 og diskuterer hvordan oppbygningen av modellene påvirker hvor godt de ulike metodene løser oppgaven. Til slutt tester jeg de beste modellene på problemet med klesplaggene og diskuterer hva som er forskjeller og likheter i de to problemene.

Programkode, vedlegg og ytterligere plott finnes på https://github.com/sivgaa/fys-stk4155/tree/main/Project3.

II. KLASSIFISERINGSPROBLEMER OG LOGISTISK REGRESJON

Klassifiseringsproblemer går ut på å kategorisere tilfeller i to eller flere klasser basert på informasjon om tilfellene. Metodene som beskrives her baserer seg på at vi trener en modell gjennom å tilpasse den til tilfeller der vi kjenner kategoriene til de ulike tilfellene og at de ulike kategoriene er gjensidig utelukkende, dvs at tilfellene faller inn i nøyaktig én av klassene. Logistisk regresjon er en forholdsvis enkel måte å lage en modell som kan klassifisere data gjennom å gjøre prediksjoner av hvilken klasse noe tilhører (utdata) basert på gitt inndata. Logistisk regresjon er godt egnet for problemer hvor klassene kan separeres lineært (Raschka *et al.*, 2022). Binær logistisk regresjon (når det kun finnes to mulige utfall) baserer seg på å modellere sannsynligheten for de ulike utfallene. Til dette brukes sigmoidfunksjonen som tar verdier mellom 0 og 1,

$$\sigma(z) = \frac{1}{1+\exp(-z)} = \frac{\exp z}{1+\exp z},$$

hvor z er representert som $z = \beta_0 + \beta_1 x$, der β_0 og β_1 er parametere som må tilpasses i modellen. Da blir

$$p(x) = \frac{\exp(\beta_0+\beta_1 x)}{1+\exp)(\beta_0+\beta_1 x)}$$

som kan generaliseres til

$$p(\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}\boldsymbol{x} + \boldsymbol{b})}{1 + \exp(\boldsymbol{w}\boldsymbol{x} + \boldsymbol{b})},$$

i tilfellet hvor inndataene våre består av mer enn én parameter. I et datasett bestående av sammenhørende verdier $\boldsymbol{x_i}$ og y_i, vil da sannsynligheten for gitt utfall være

$$P = \prod_{i=1}^{n} \left[p(y_i = 1|\boldsymbol{x_i}, \boldsymbol{w}, \boldsymbol{b})\right]^{y_i} \left[1 - p(y_i = 1|\boldsymbol{x_i}, \boldsymbol{w}, \boldsymbol{b}))\right]^{1-y_i}.$$

Vi ønsker å finne verdiene $\boldsymbol{w}$ og $\boldsymbol{b}$ som gir størst mulig sannsynlighet, dvs at modellen vår passer best mulig med det som er observert. For å gjøre det enklere, så tar vi den naturlige logaritmen av denne funksjonen

$$\begin{aligned} log(P) = \sum_{i=1}^{n} (y_i \log p(y_i = 1|\boldsymbol{x_i}, \boldsymbol{w}, \boldsymbol{b}) \\ + (1 - y_i) \log \left[1 - p(y_i = 1|\boldsymbol{x_i}, \boldsymbol{w}, \boldsymbol{b})\right]. \end{aligned}$$

Hvis vi så setter inn $p(\boldsymbol{x})$ i denne får vi et uttrykk som kalles for *cross entropy* som benyttes som kost-funksjon i logistisk regresjon:

$$\mathcal{C}(\boldsymbol{w}, \boldsymbol{b}) = -\sum_{i=1}^{n} \left(y_i(\boldsymbol{w}\boldsymbol{x_i} + \boldsymbol{b}) - \log\left(1 + \exp\left(\boldsymbol{w}\boldsymbol{x_i} + \boldsymbol{b}\right)\right)\right).$$

Optimering, eller trening, av modellen består så i å finne det beste settet med parametere $\boldsymbol{w}$ og $\boldsymbol{b}$. Deretter kan modellen benyttes til å klassifisere ukjente tilfeller gjennom å beregne sannsynligheten $p(\boldsymbol{x})$ og en på forhånd bestemt terskelverdi for å angi hvilken klasse dne hører hjemme i.

A. Klassifisering med mer enn to klasser

Dersom klassifiseringsproblemet har mer enn to klasser, må modellen over tilpasses for å ta høyde for mer enn to utfall. Sannsynligheten for et gitt utfall blir nå

$$P = \prod_{i=1}^{n} \prod_{k=1}^{K} p_k(\boldsymbol{x_i}, \boldsymbol{w}, \boldsymbol{b})^{y_i},$$

hvor det er totalt K klasser. Sannsynlighetsmodellen må også tilpasses at det er flere klasser og dette gjøres gjennom å erstatte sigmodifunksjonen med *softmax*:

$$p_k(\boldsymbol{x_i}) = \frac{\exp\left(\boldsymbol{w_k}\boldsymbol{x} + \boldsymbol{b_k}\right)}{\sum_{l=1}^{K-1} \exp\left(\boldsymbol{w_k}\boldsymbol{x} + \boldsymbol{b_k}\right)}.$$

Totalt sett får vi da til slutt en kostfunksjon gitt ved:

$$\mathcal{C}(\boldsymbol{w}, \boldsymbol{b}) = -\sum_{i=1}^{n} \sum_{k=1}^{K} \log \frac{\exp\left(\boldsymbol{w_k}\boldsymbol{x} + \boldsymbol{b_k}\right)}{\sum_{l=1}^{K-1} \exp\left(\boldsymbol{w_k}\boldsymbol{x} + \boldsymbol{b_k}\right)}.$$

Denne kostfunksjoen kan så benyttes for å finne optimale verdier av modellparameterne. Men for å gjøre prediskjoner i flerklasseklassifisering, klassifisering med mer enn to utfall, må det beregnes en sannsynlighet for hver av klassene. Deretter kan man f.eks. benytte såkalt *one hot encoding* som vil si at man plasserer tilfellet i den klassen som har høyest sannsynlighet. På denne måten vil man i alle tilfeller tildele en klasse. Modellen vi da predikere kun én klasse selv om det var flere klasser som hadde høy sannsynlighet og også selv om ingen av klassene hadde høy sannsynlighet.

III. OPTIMERING AV PARAMETERE

For å finne de optimale parameterne benyttes gradientmetoder, som går ut på å iterativt oppdatere **w** og **b** gjennom å ta steg i motsatt retning av gradienten (Raschka *et al.*, 2022):

$$\boldsymbol{w} = \boldsymbol{w} + \Delta\boldsymbol{w}$$

$$\boldsymbol{b} = \boldsymbol{b} + \Delta\boldsymbol{b}.$$

I gradient descent (GD) tar oppdateringen utgangspunkt i å taylorutvikle kostfunksjonen til andre orden slik at

$$\begin{aligned} \boldsymbol{w}^{n+1} &= \boldsymbol{w}^n - (\frac{d^2C}{d\boldsymbol{w}^2})^{-1}\Big|_{\boldsymbol{w}=\boldsymbol{w}^n} \frac{dC}{d\boldsymbol{w}}\Big|_{\boldsymbol{w}=\boldsymbol{w}^n} \\ &= \boldsymbol{w}^n - \eta^{(n)}\boldsymbol{g}^{(n)} \end{aligned}$$

hvor $\boldsymbol{g}$ er gradienten og η kalles for læringsraten (læringsrate) og representerer inversen av den andrederiverte av funksjonen. Læringsraten utgjør en metaparameter i algoritmen, og kan tilnærmes på ulike måter, eller bare gis en verdi. **b** oppdateres på tilsvarende måte. Dette er gjort rede for i prosjekt 2.

For store datasett, vil det å regne ut gradienter være en tidkrevende prosess, og stokastisk gradient descent (SGD) er en måte redusere tidskostnaden på. I SGD evalueres gradienten i hver iterasjon på en tilfeldig valg undergruppe av dataene, en såkalt batch. Minibatch stokastisk gradient descent er den mest brukte varianten av SGD (Raschka, nd). Her deles datasettet inn i minibatcher, og iterasjonene deles inn i *epoker*. For hver iterasjon benyttes én minibatch til å beregne gradienten, mens en epoke utgjør så mange iterasjoner at hele datasettet er benyttet. På den måten forteller antall epoker i praksis hvor mange ganger man har løpt gjennom hele datasettet i optimeringen. Etter hver epoke kan det så gjøres en vurdering av om optimeringen er konvergert.

A. Adam

I GD trengs et uttrykk for den andrederiverte av kostfunksjonen mhp parameteren som skal optimeres. For å

unngå å måtte beregne denne andrederiverte, tilnærmes denne gjennom metaparameteren som kalles *læringsraten*. Ofte gis denne som en tallverdi som må tilpasses til problemet. Den kan også variere gjennom optimeringen (iterasjonene) f.eks. ved at den gjøres mindre og mindre etter hvert som man har gjort flere iterasjoner slik at det tas kortere og kortere skritt fra én iterasjon til den neste. Adam (Kingma and Ba, 2017) er en algoritme SGD inkludere gradienten fra forrige iterasjon i første og andre potens i tillegg. Dette kan betraktes som en tilpasning av læringsraten.

Adam flere metaparametere, η, ρ_1, ρ_2, og vi ser at denne metoden benytter både gradienten i første potens (m), med minne fra forrige iterasjon, og gradienten i andre potens (s) for å justere steget i $\boldsymbol{w}$:

$$\boldsymbol{m}^{(i)} = \rho_1 \boldsymbol{m}^{(i-1)} + (1-\rho_1)\boldsymbol{g}^{(i)}$$

$$\boldsymbol{s}^{(i)} = \rho_2 \boldsymbol{s}^{(i-1)} + (1-\rho_2)(\boldsymbol{g}^{(i)})^2$$

$$\boldsymbol{m}^{(i)} = \frac{\boldsymbol{m}^{(i)}}{1-\rho_1^i}$$

$$\boldsymbol{s}^{(i)} = \frac{\boldsymbol{s}^{(i)}}{1-\rho_2^i}$$

$$\boldsymbol{w}^{(i+1)} = \boldsymbol{w}^{(i)} - \eta^{(i)} \frac{\boldsymbol{m}^{(i)}}{\sqrt{\boldsymbol{s}^{(i)}+\delta}}.$$

δ er her et lite tall som legges til for å unngå problemer med å dele på 0. Og igjen, vil det være et helt tilsvarende sett med ligninger for $\boldsymbol{b}$, og det er vanlig å benytte de samme metaparameterne for å optimere både $\boldsymbol{w}$ og $\boldsymbol{b}$.

IV. NEVRALE NETTVERK

Feed forward nevrale nettverk (FFNN) er gjort grundigere rede for i prosjekt 2, men her vil jeg oppsummere og gjengi det som er viktig for helheten i denne oppgaven. I et FFNN gjøres dataene våre gjennom flere lag, gjennom gjennom lineære og ikke-linere operasjoner (Goodfellow *et al.*, 2016). Selve modelleringen i et FFNN er ikke tilpasset problemet som skal modelleres, og metoden er derfor fleksibel og kan benyttes på mange ulike typer problemer.

Inndataene til det nevrale nettverket kalles i det følgende for $\mathbf{z}$, mens utdataene fra hvert lag betegnes med $\mathbf{a}$. Inndataene våre,$\mathbf{x}$, kan betraktes som at de utjgør utdata fra det nulte laget. Mellom to lag gjøres det først en lineær operasjon på dataene hvor indata til lag (i) produseres lages basert på utdata fra lag (i-1), på denne måten

$$\boldsymbol{z}^{(i)} = \boldsymbol{W}^{(i)}\boldsymbol{a}^{(i-1)} + \boldsymbol{b}^{(i)}.$$

Deretter foretas en ikke -lineær operasjon når det produseres utdata fra laget gjennom en aktiveringsfunksjon

$$\boldsymbol{a}^{(i)}(\boldsymbol{z}).$$

Til slutt er det et utdatalag, og input i dette laget utgjør modellens prediksjoner. Lagene mellom inndatalaget og utdatalaget kalles skjulte lag, og et nevralt nettverk kan ha mange eller få slike, og disse kan ha mange eller få noder (dimensjonen til z). Trening av nettverket består i å optimere $\mathbf{W}$ og $\mathbf{b}$ for at prediksjonene skal bli så gode som mulig.

Hvert lag i et FFNN kan ha ulike dimensjoner. Dimensjonen til inndatalaget må svare til informasjonen vi har i datasettet vårt, antall features, og utdatalaget må ha samme dimensjon som det finnes klasser, dersom nettverket skal brukes til å modellere et klassifiseringsproblem. For de skjulte lagene, derimot, står vi fritt til å velge dimensjon og hvor mange lag vi ønsker, oppbygnignen av disse kalles arkitekturen til det nevrale nettverket. Hvor stort og dypt nettverk som trengs, avhenger av kompleksiteten i problemet som skal modelleres.

Lagene i et FFNN sies å fullt koblede. Det betyr at hver node i hvert lag er koblet direkte til samtlige noder i laget før og etter. Dermed vil store, fleksible, nevrale nettverk være regnetunge, særlig når det er store datasett som skal modelleres.

A. Trening av nevrale nettverk

I et nevralt nettverk initieres modellen ved at $\boldsymbol{W}$ og $\boldsymbol{b}$ initieres med (tilfeldig) valgte verdier. DERetter består trening av nettverket i å terativt propagere fremover gjennom lagene og gjøre prediksjoner med modellen, beregne kostfunksjonen og så beregne gradientene til parameterne gjennom å propagere bakover gjennom nettverket, for så å oppdatere $\boldsymbol{W}$ og $\boldsymbol{b}$ og gjenta prosessen til vi oppnår konvergens.

Gradienten

$$W^{(i)} = W^{(i-1)} - \eta \frac{dC}{dW}$$

beregnes gjennom såkalt tilbakepropagering. For hvert lag i det nevrale nettverket kan denne uttrykkes som

$$\frac{dC}{dW} = \frac{dC}{da}\frac{da}{dz}\frac{dz}{dW}$$

Gradienten vil for hvert lag (bakover) avhenge av lagene foran:

$$\frac{dC}{da_n} = \frac{dC}{da_{n+1}}\frac{da_{n+1}}{dz_{n+1}}\frac{dz_{n+1}}{da_n}.$$

Når de deriverte er beregnet i tilbakepropagerinen kan $\mathbf{W}$ oppdateres ved bruk av gradientene og valgt gradientmetode. $\boldsymbol{b}$ vil oppdateres parallellt med $\boldsymbol{W}$ helt analogt.

B. Aktiveringsfunksjoner, kostfunksjoner, nøyaktighet i nevrale nettverk

I tillegg til å kunne variere antall skjulte lag og noder per lag, påvirker valg av aktiveringsfunksjoner og hvilken kostfunksjon som benyttes funksjonen til et nevralt nettverk.

Kostfunksjonen må tilpasses problemet som skal løses, og for klassifiseringsproblemer vil samme kostfunksjon kunne benyttes som for logistisk regresjon. Tilsvarende må aktiveringsfunksjonen for utdatalaget tilpasses typen prediksjoner som skal gjøres, og softmax kan benyttes for flerklasseklassifisering, på samme måte som for logistisk regresjon. Vi ser dermed at logistisk regresjon utgjør det samme som et FFNN uten noen skjulte lag mellom hvor mang går direkte fra inndata til utdata gjennom en lineær operasjon $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$ med tilpassede prametere og deretter anvender softmax og beregner cross entropy.

For de skjulte lagene står man fritt til å velge ulike typer aktiveringsfunksjoner. Hyperbolsk tangentfunksjonen **tanh**, er en mye benyttet aktiveringsfunksjon (Raschka *et al.*, 2022)

$$a_{tanh}(z) = \frac{e^z - e^{-z}}{e^z - e^{-z}}.$$

Dette er en ikke-lineær funksjon som gir verdier mellom - 1 og 1. Ulempen med denne er at den kan gi problemer med for små gradienter i tilbakepropageringen, slik at optimeringen stopper opp. En annen mye benyttet funksjon, er ReLU-funksjonen:

$$a(z) = max(0, z).$$

Denne funksjonen tar samme verdier som inndataene dersom disse er positive, men gir 0 dersom inndataene er negative. Dersom man skal ha mange lag i nettverket sitt, er dette en bedre aktiveringsfunksjon, da denne ikke har de samme problemene med forsvinnende gradienter som tahn-funksjonen (Raschka *et al.*, 2022).

For klassifiseringsproblemer er det vanlig å beregne nøyaktighet i tillegg til kostfunksjonen. Nøyaktigheten kan uttrykkes som

$$\text{Accuracy} = 100\ \% \times \frac{\sum_{i=1}^{n} I(t_i = y_i)}{n}.$$

Denne teller rett og slett opp hvor mange ganger vi traff prediksjonen delt på antall prediksjoner som er gjort. Det kan også være interessant å lage en såkalt *feilmatrise* som inneholder informasjon om andel rett og galt klassifiserte tilfeller for hvert utfall.

C. Konvolusjonsnettverk

Konvolusjonsnettverk (CNN) utgjør en utvidelse av et FFNN hvor det er lagt til flere lag mellom inndataene og de fullt koblede lagene beskrevet over. Konvolusjonslagene er ikke fullt koblede, gjennnom at hver node i et lag forholder seg kun til et utvalg av nodene i laget før. Konvolusjonsnettverk er dermed tilpasset bildeanalyse fordi operasjonenen som gjøres på inndataene forholder seg til hvordan inndatane er ordnet, gjennom at bildepiksler i nærheten av hverandre blir evaluert sammen. Hovedprinsippet i CNN er at man først behanlder dataene gjennom et filter (en *konvolusjon*), deretter anvendes en aktiveringsfunksjon, før man trekker ut inforamsjon om dataene gjennom en såkalt *pooling* som utgjør en reduksjon i dimensjonaliten til problemet. Dette kan gjentas flere ganger, før datane til slutt sendes gjennom et fullt koblet nettverk for klassifisering. Parameterne i CCN optimeres gjennom iterativ fram- og tilbakepropagering, som beskrevet over.

Fordi CNN utgjør en systematisk reduksjon i dimensjonen av dataene, en slags filtrering, kan disse nettverkene være mer effektive for å gjøre f.eks. bildegjenkjenning. Bilder har ha høy oppløsning, hver pixel utgjør en feature, men det er sannsynlig at det vil være mest relevant å sammenligne en pixel med andre pixler i nærheten, heller enn alle pixler i bildet. Derfor forventes det at et mindre CNN kan gjøre en bedre jobb med bildegjenkjenning enn et større FFNN.

1. Konvolusjoner

I en konvolusjon anvendes en *kernel* på inndataene for å produsere en output. Dette er en lineær operasjon som utføres på bare en del av inndataene om gangen og erstatter den lineære operasjonen mellom fullt koblede lag. For et bilde kan vi betrakte dette som et filter av en gitt dimensjon som legges over et bilde. Det kan benyttes mange ulike filtere/kernels i hvert lag av et CNN. Fordi filteret initieres tilfeldig vil hvert filter fange opp ulike typer variasjoner i bildet. Optimeringen består av å forsterke filterne slik at de forsterker og fanger opp relevante"variasjoner i bildet. Dette ligner måten vi mennesker analyserer bilder på, vet at vi oppfatter fasonger/variasjoner som f.eks. hjørner, buer, fargeendringer osv. (Raschka *et al.*, 2022).

Filteret flyttes systematisk rundt i hele bildet og genererer på denne måten et nytt bildesom inneholder resultatet av konvolusjonen. Både dimensjonen til filteret og hvor langt filteret flyttes mellom hver anvendelse, også kalt *stride*-paramteeren, vil påvirke hva som fanges opp i bildet. Dersom filteret er stort, vil det fange opp elementer med større utbredelse i det opprinnelige bilde, mens et lite filter vil kunne fange opp mindre detaljer. Sammenhengen mellom dimensjonen til filterne og strideparameteren vil avjgøre om det blir overlapp mellom der filterne anvenes.

Fordi filteret har en utstrekning som er større enn én pixel, vil dimensjonen til det nye bildet være mindre enn

det opprinnelige, dersom det ikke er mulig å plassere filteret "utenfor kanten av bildet". Ved å anvende såkalt *padding*, vanligvis gjennom å legge til nuller rundt bildet, kan dette problemet unngås (Hjorth-Jensen, 2021). Da blir det også mulig å fange opp mer av det som eventuelt skjer i kanten av bildet, fordi det kan samples også herfra.

Etter å ha foretatt en konvolusjon med ett eller flere filter, er det produsert ett eller flere nye bilder basert på inndataene. Disse bildene behandles så gjennom en aktiveringsfunksjon, gjerne ReLU eller tanh.

2. Pooling

Konvolusjonen beskrevet over, utgjør i praksis en utvidelse av datamaterialet gjennom at det dannes flere bilder. For å trekke ut den relevante informasjonen i de nye bildene og samtidig redusere dimensjonaliteten i dataene gjøres det en såkalt pooling. En vanlig form for pooling er å trekke ut maksimalverdien blant f.eks. kvadrater på 2×2 pixeler og denne ene verdien. Det er vanlig å ikke ha overlapp i dette tilfellet. Slik at eksempelet over vil utgjøre en redusjon i dimensjonalitet til $1/4$. Det kan også benyttes f.eks. gjennomsnittsverdi i en slik pooling, og dimensjonen kan tilpasses problemet. Pooling er med på å gjøre metoden invariant for små forflytninger i bildet (Goodfellow *et al.*, 2016).

Gjennom én eller flere påfølgende konvolusjoner og poolinger kan vi på denne måten ekstrahere informasjon om strukturer i et bilde (uavhengig av deres plassering). Dette kan benyttes som inndata til et FFNN eller en logistisk regresjon (et FFNN med kun utdatalag) for å klassifisere bildet som analyseres.

V. DATASETT SOM ER BENYTTET

For uttesting er det benyttet to ulike datasett. MNIST (Lecun *et al.*, 1998) er et datasett som består av 70000 bilder av håndskrevne siffer mellom fra 0 til 9. Datasettet er sortert i 60 000 siffer i et treningssett, og 10 000 siffer i et testsett. Bildene har oppløsning på 28×28 pixler og dermed 784 datapunkter per bilde.Fashion MNIST (Xiao *et al.*, 2017) består av like mange bilder og med samme oppløsning, men her er det bilder av ulike klesplagg. Begge datasettene består av svart-hvitt-bilder.

A. MNIST

MNIST-datasettet ble presentert i 1998 i et arbeid som også la fram konvolusjonsnettverk som en effektiv og nøyaktig metode for å gjøre klassifisering av bilder (Lecun *et al.*, 1998), og er senere benyttet til benchmarking av ulike maskinlæringsalgoritmer for bildegjenkjenning (Yann LeCun, nd). Med logistisk regresjon, og uten preprosessering av data er det funnet at klassifisering med MNIST-data gir en feilrate på 12%, noe som tilsvarer en nøyaktighet på 88%(Lecun *et al.*, 1998), mens for FFNN er de beste resultatene oppnådd med 5 skjulte lag (Cireşan *et al.*, 2010) med en feilrate på $0,35\%$. Av de mindre FFNN er det funnet at ett skjult lag med 300 noder uten preprosessering gir en nøyaktighet på $95,3\%$, 1000 noder gir en nøyaktighet på $95,7\%$ (Lecun *et al.*, 1998). Ved bruk av to skjulte lag med hhv 300 og 100 noder, ble det funnet nøyaktigheten øker til $96,95\%$, og $1000 + 150$ lag ga en nøyaktighet på $97,05\%$. Alle disse er gjort uten preprosessering av dataene og med MSE som kostfunksjon.

LeNet5, beskrevet lengere nede, er også benyttet for å gjøre klassifiseringer på MNIST dataene. Denne arktiekturen har to konvolusjonslag med hhv. 6 og 16 filtere og to påfølgende skjulte lag med hhv 102 og 84 noder. Med dette oppsettet ble det funnet at konvergens ble nådd med etter 10-12 epoker med en nøyaktighet på 99.05% (Lecun *et al.*, 1998).

B. Fashion MNIST

Fashion MNIST (Xiao *et al.*, 2017) ble laget senere for å utgjøre et mer komplekst problem enn MNIST, men ellers så likt som mulig. Bildene er tatt fra nettbutikken Zalando og hvert bilde er klassifisert som enten T-shirt/top", Trouser", Pullover", Dress", "Coat", SSandal", Shirt", Sneaker", Bageller "Ankle boot", slik at det også her er 10 klasser. Dette medfører at kode som brukes for å lage modeller for MNIST-settet kan direkte overføres til å brukes på Fashion MNIST (Fas, nd). Tidligere utprøvinger med CNN med 2 konvolusjonslag har gitt nøyaktighet på $91,6\%$ (Fas, nd).

VI. IMPLEMENTERING AV MODELLENE

Alle modellene er implementert i python ved bruk av pytorch (Paszke *et al.*, 2019), NumPy (Harris *et al.*, 2020) og Scikit-learn (Pedregosa *et al.*, 2011). Plot er laget med seaborn-pakken for visualisering av data (Waskom, 2021).

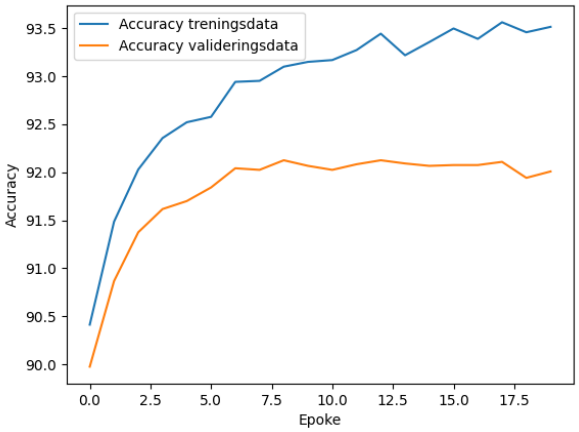
Det ble tatt utgangspunkt i eksempelkode jeg fant på medium.com (Poyekar, 2024) som ble tilpasset til mitt bruk. Koden er kjørt i Google Colab, og AI-en der tilbyr seg uoppfordret å bidra til å både forstå og tilpasse kode. Jeg har gladelig trykket tab når den har foreslått kode som jeg uansett hadde tenkt å skrive inn, f.eks. for å lage plott. Jeg også bedt AI-en om å forklare meg kodebiter, og bedt den om å foreslå kodebiter for å lagre plot osv. på Google drive istedenfor å google meg fram til dette. Forslagene var tidvis gode, kan man vel si. Jeg har også gjenbrukt kodebiter fra prosjekt 1 og 2 for å lage plott

og lignende.

All koden ble laget med utgangspunkt i kode for et FFNN, som så enten ble omgort til en logistisk regresjon ved å fjerne de skjulte lagene eller til et CNN ved å legge til konvolusjonslag i forkant. Koden baserer seg på bruk av innebygde klasser og funksjoner i pytorch. Den lager en klasse for det nevrale nettverket (eller den logistiske regresjonen). Nøyaktigheten beregnes eksplisitt i en egen funksjon, men det kunne sikkert vært benyttet en innebygget pytorch funksjon til dette. Datasettene som benyttes likker også tilgjengelige via pytorch dataloader. Etter at data er lastet inn og parametere er satt initialiseres nettverket og det settes cross entropy loss som kostfunksjon og Adam som gradientmetode for optimaliseringen. Deretter trenes modellen gjennom å løpe gjennom et gitt antall epoker og for hver epoke løpe gjennom alle batchene for å gjøre prediksjoner (beregne scores), beregne kostfunksjonen og finne gradienter gjennom tilbakepropagering før parameterne endres i et optimaliseringssteg. For hver epoke er det beregenet nøyaktighet for treningsdata og valideringsdata som er plottet og benyttet for å vurdere om modellen er konvergert og over/under-tilpasset.

Denne oppgaven har fokus på å teste ut ulike arkitekturer, mens andre metaparametere er holdt konstant. Det er benyttet SGD med batch-strørrelse på 64 bilder. Det ble gjort utprøvinger med 20 epoker i evalueringen av de ulike arkitekturene, for å se om/når de konvergerte og vurdere overtilpassing. Optimeringsalgoritmen som er benyttet er Adam (Kingma and Ba, 2017), med en læringsrate $\eta = 0,001$ og $\rho_1 = 0,9$ og $\rho_2 = 0,999$ som er defaultverdier i pyroch sin implementering av Adam. Det er benyttet cross entropy loss som kostfunksjon og softmax som aktiveringsfunksjon i siste lag. ReLU er benyttet som aktiveringsfunksjon for skjulte lag, der ikke annet er oppgitt eksplisitt. For konvolusjonsnettverkene er det max-pooling som er benyttet der ikke annet er oppgitt.

For begge datasettene ble treningsdataene splittet i et treningssett (80%) og et valideringssett (20%) som ble benyttet i utprøvingen av arkitekturene. Uttestingen av ulike arkitekturer ble gjort med MNIST-datasettet. Til slutt ble det gjort kjøringer med de beste arkitekturene fra konvolusjonsnettverk og FFNN, samt logistisk regresjon for både MNIST og Fashion MNIST hvor testdataene ble benyttet for å vurdere kvaliteten på modellene. Hensikten med dette var både å se hvor gode metodene var når det ble bentyttet testdatasett for MNIST, men også for å få et inntrykk av hvor godt de metodene som fungerte for MNIST gjorde det for Fasion MNIST.



Figur 1 Nøyaktighet som funksjon av epoke for logsitisk regresjon på MNIST-datasettet.

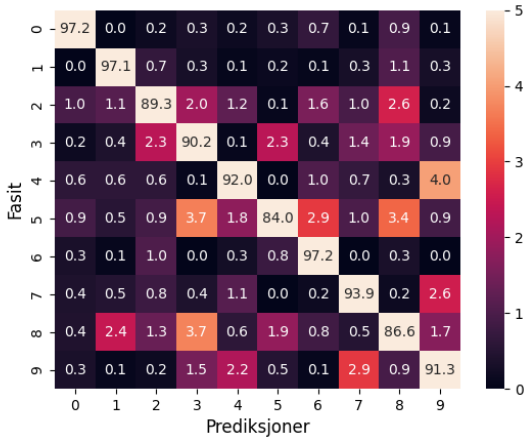
VII. UTPRØVING FOR MNIST MED LOGISTISK REGRESJON OG FFNN

For FFNN ble det prøvd ut modeller med ett og to skjulte lag, og med ulike antall noder i de skjulte lagene. Resultatene av utprøvingene for FFNN og logistisk regresjon er gjengitt i tabell I.

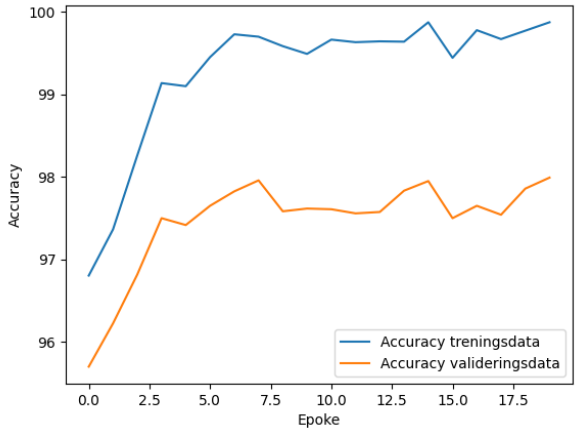
For logistisk regresjon ble det etter 20 epoker oppnådd en nøyaktighet på $93,51\%$ for treningsdataene og $92,01\%$ for valideringsdataene. Dette er jo ikke en spesielt god modell, men det er likevel bedre enn de 88%-ene som er funnet tidligere, da ved bruk av MSE som kostfunksjon (Lecun *et al.*, 1998). At det er lite forskjell på nøyaktigheten for treningsdataene og valideringsdataene tyder på at modellen i liten grad er overtilpasset. Figur 1 viser nøyaktigheten som funksjon av epoker for treningsdata og valideringsdata med logistisk regresjon. Vi ser at modellen etter 7-8 epoker slutter å gjøre det bedre på valideringssettet, mens nøyaktigheten fortsetter å øke for treningssettet. Dette tyder egentlig på en overtilpassing av modellen.

Figur 2 viser feilmatrisen etter 20 epoker med logistisk regresjon. Vi ser av figuren at modellen gjør det best for tallene 0, 1 og 6, men dårligst for 5 og 8. Tallet 5 ser ut til å blandes spesielt med 3, 6 og 8, mens 8 blandes med 2, 3, 5 og 9. Det er vanskelig å tillegge dette mye mening, men vi ser hvertfall at det er litt ulikt hvordan modellen gjør det for ulike tall.

For FFNN med ett skjult lag ble det valgt å teste modeller med 50, 300 og 1000 noder i det skjulte laget for å kunne sammenligne med tidligere rapporterte resultater, i tillegg til et ganske mye mindre nettverk. For tolags-modellene ble det valgt 300+100 og 500+150 noder i de skjulte lagene for å kunne sammenligne med tidligere rapporterte resultater, i tillegg til et oppsett med 120+84 skjulte lag, som tilsvarer de fullt koblede lagene



Figur 2 Feilmatrise for valideringsdataene etter 20 epoker med logistisk regresjon på MNIST-datasettet.



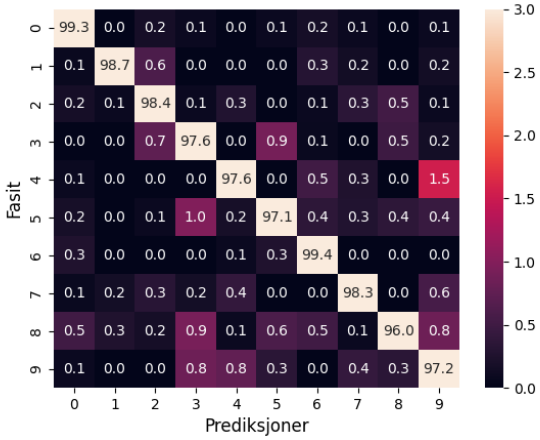
Figur 3 Nøyaktighet som funksjon av epoke for modell FFNN 5 (to lag med 500+150 noder) for MNIST.

som inngår i LeNet5-arkitekturen (Lecun *et al.*, 1998).

Som vi ser av tabell I oppnår vi med FFNN en nøyaktighet på nesten 100% for treningsdataene, mens den ligger en god del lavere for valideringsdataene. Likevel oppnår vi for både ett og to skjulte lag resultater som er sammenlignbare og til og med noe bedre enn de som er blitt funnet tidligere med tilsvarende arkitekturer (Lecun *et al.*, 1998). Som forventet gjør de minste nettverkene en noe dårligere jobb en de større, hvor det minste nevrale nettet gir en nøyaktighet for valideringsdataene på $96,97\%$, mens modell 5, som er det største nettverket, med to skjulte lag med $500 + 150$ noder, gir en nøyaktighet på $97,99\%$. Dersom man hadde økt til 5 lag, er det grunn til å tro at man kunne fått enda bedre resultater, selv med FFNN (Cireşan *et al.*, 2010).

Med unntak av den minste modellen, er alle de nevrale nettene konvergert etter omtrent 10 epoker. Figur 3 viser nøyaktigheten som funksjon av epoker for treningsog valideringsdataene. Vi ser at modellen gjør det jevnt bedre for treningsdataene enn valideringsdataene, men at forskjellen mellom de to virker å være stabil. Det virker derfor som at modellen er noe overtrent, i at den har en bias mot treningsdataene. Tilsvarende plott for de andre modellene som er FFNN finnes i Vedlegg A.

Figur 4 viser feilmatrisen for modell 5. Her ser vi at også denne modellen er best til å gjenkjenne 0-ere og 6-ere, men at den også gjør det jevnt mye bedre på alle siffer enn modellen laget med logistisk regresjon. Igjen er det også 8 som peker seg ut som det vanskeligste sifferet å bestemme, og dette blandes bl.a. med 5. Feilmatrisene for de andre FFNN-modellene finnes også i Vedlegg A.



Figur 4 Feilmatrise for valideringsdataene etter 20 epoker for modell FFNN 5 (to lag med 500+150 noder) for MNIST.

VIII. UTPRØVING FOR MNIST MED KONVOLUSJONSNETTVERK

Det ble prøvd ut konvolusjonsnettverk med én og to konvolusjoner, og max pooling mellom konvolusjonen, og kun ett fullt koblet lag etter konvolusjonslagene (utdatalaget). Dimensjon på filtere, stride og padding ble valgt etter anbefalinger funnet i forelesningsnotatene for kurset FYS-STK4155(Hjorth-Jensen, 2021). Det ble testet modeller med 1, 4, 8 og 16 filtere.

A. Konvolusjonsnettverk med ett konvolusjonslag

Resultatene av utprøvingene med konvolusjonsnettverk med ett konvolusjonslag er gjengitt i tabell II. Her ser vi at at selv ett konvolusjonslag gir resultater gir bed-

Modell	Tid for trening (min)	Nøyaktiget treningsdata (%)	Nøyaktiget valideringsdata (%)	Antall skjulte lag	Noder per skjulte lag
Logistisk regresjon	7	93,51	92,01	0	
1	9	99,53	96,97	1	50
2	*mangler*	99,92	97,88	1	300
3	14	99,82	97,66	1	1000
4	11	99,81	97,65	2	300+100
5	12	99,88	97,99	2	500+150
6	10	99,81	97,51	2	120+84

Tabell I Logistisk regresjon og FFNN-modeller. Nøyaktigheten er oppgitt etter 20 epoker for MNIST.

Modell	Tid for trening (min)	Nøyaktiget treningsdata (%)	Nøyaktiget valideringsdata (%)	Antall filtere	Dimensjon filter	Stride	Padding
1	9	92,56	91,83	1	3	1	1
2	10	94,65	93,64	1	5	1	2
3	9	89,91	89,73	1	5	2	2
4	9	11,27	11,11	1	1	1	0
5	11	97,32	95,84	4	3	1	1
6	12	98,84	97,67	4	5	1	2
7	9	97,36	96,72	4	5	2	2
8	10	93,07	92,11	4	1	1	0
9	12	99,1	97,78	8	3	1	1
10	13	99,28	97,9	8	5	1	2
11	10	98,58	97,58	8	5	2	2
12	-	-	-	8	1	1	0
13	16	99,69	98,06	16	3	1	1
14	18	99,82	98,22	16	5	1	2
15	11	99,22	97,95	16	5	2	2
16	15	93,81	92,62	16	1	1	0
14 modifisert	19	99,74	98,45	16	5	1	2

Tabell II CNN-modeller med ett konvolusjonslag. Nøyaktigheten er oppgitt etter 20 epoker. Der det står ble ikke modellen benyttet for MNIST.

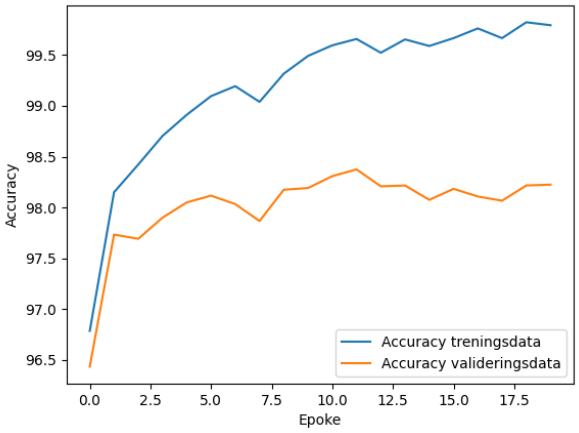
re enn de største FFNN beskrevt over, men dette gjelder ikke alle modellene.

Modellene med filter av dimensjon 1 ga elendige resultater. Det ble noe bedre med flere filtere, men likevel så dårlig at modell 12 ikke ble kjørt, da det ikke ble vurdert som hensiktsmessig. Det er ikke veldig overraskende at disse modellene var dårlige, da de i praksis ikke gjør noen konvolusjon, og heller ikke er hverken fullt koblede eller litt koblet på tvers, med unntak av poolingen som i praksis bare fjerner 1/4 av informasjonen.

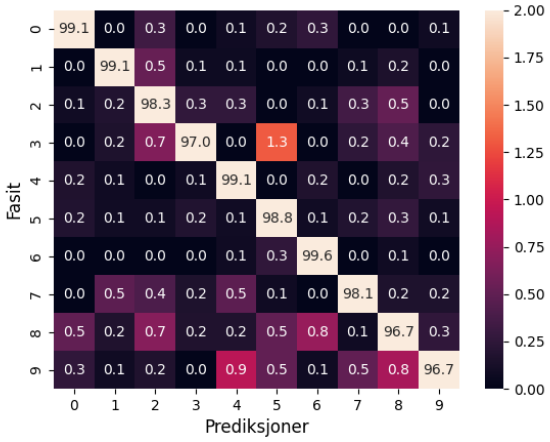
Modellene med stride 2 ga også betrakelig dårligere resultater enn modellene med stride 1, men til gjengjeld var de mye raskere å trene. Dette skyldes nok rett og slett at stride 2 medfører en større reduksjon i datamengde, og at vi til en viss grad kan "hoppe overelementer i bildet. Dette er nok mer aktuelt for MNIST-dataene enn for andre bilder, da disse har svært grov oppløsning. Likevel er det vert å merke seg at disse modellene var svært mye raskere enn de andre modellene, og hvis vi sammenligner det største av disse modellene (16 filtre av dimensjon 5, med stride 2 og 2 pixler padding) med andre modeller som tok like lang tid (11 minutter), ser vi at modellen med stride 2 hadde en nøyaktighet på evalueringsdataene på $97,95\%$ mens modellen med kun 4 filtre av dimensjon 3 hadde en nøyaktighet på kun $97,67\%$ som faktisk er dårligere. Siden disse modellene uansett trenes raskt, er ikke det relevant her, men det kan være noe å ta med seg hvis man skal trene modeller for større og evt. flere bilder.

Vi ser at det å bruke et filter av dimensjon 5, med padding på 2 og stride på 1 ga de beste resultatene for ett filter. Vi ser også at dette er de modellene det tok lengst tid å trene, men at denne tiden ikke er mye lengere enn for de andre modellene med gitt antall filtere. Vi ser også at flere filtere ga bedre modell, og Modell 14 med 16 filtere ga de beste resultatene, med en nøyaktighet på $98,22\%$ for valideringsdataene.

Figur 5 viser nøyaktigheten for både treningsdataene og valideringsdataene som funksjon av epoker for modell 14, den beste modllen. Vi ser at også denne modellen ser



Figur 5 Nøyaktighet som funksjon av epoke for modell CNN 14 (ett konvolusjonslag, 16 filter, filterdimensjon 5, stride 1 og padding 2) for MNIST.



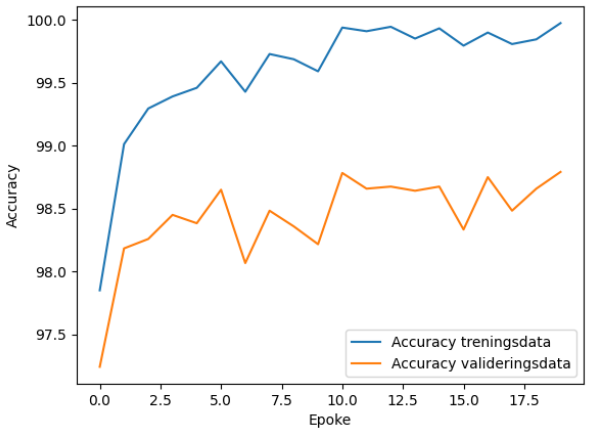
Figur 6 Feilmatrise for valideringsdataene etter 20 epoker for modell CNN 14 (ett konvolusjonslag, 16 filter, filterdimensjon 5, stride 1 og padding 2) for MNIST.

ut til å være overtrent, da nøyaktigheten på valideringsdataene ikke øker, selv om nøyaktigheten på treningsdataene ser ut til å fortsette å øke, selv etter 20 epoker. Tilsvarende plott for de andre modellene med ett konvolusjonslag finnes i Vedlegg B.

Ved å se på feilmatrisene for modell 14, figur 6, ser man at også for denne modellen er prediksjonene av tallene 0, 1 og 6 mest presise, mens 3, 8 og 9 her skiller seg ut som dårlige. Spesielt kan man legge merke til at tallet 9 ofte feilpredikeres som 4 og at 2 ofte feilpredikeres som 8. Det er fristende å tenke at dette kan skyldes at f.eks. 4 og 9 er topologisk like, og at et 2-tall blir et 8-tall hvis man bare trekker en diagonal strek fra starten til slutten av 2-tallet slik at man lukker løkken. På den annen side er både 2 og 4 på et vis kantete, mens hverken 8 eller 9 er det, så sånn sett er det litt pussig å tenke på at modellen blander disse. Men for å vite om denne typen resonnement har noe for seg, må man undersøke filterene i modellen nærmere, og det er ikke gjort her. Feilmatrisene til de andre modellene med ett konvolusjonslag finnes også vedlagt i Vedlegg B, for den glade spekulant.

Det ble også testet en modell som var Modell 14, men med 2 skjulte lag på toppen, inspirert av LeNet5. Det ble da valgt to skjulte lag med 500+150 noder, da dette var FFNN-modellen som ga best resultater. Denne er lagt inn i nederste linje i tabell II, kalt Modell 14 modifisert. Denne modellen ga noe dårligere resultater for treningsdataene, men likevel bedre for valideringsdataene, noe som kan tyde på at denne modellen er mindre overtrent, og det ble funnet en nøyaktighet på valideringsdataene på 98, 45%.

Figur 7 og 8 viser henholdsvis nøyaktigheten som funksjon av epoke og feilmastrisen etter 20 epoker for den modifiserte versjonen av modell 14. Her kan vi se at modellen ser ut til å være ganske konvergert etter 10-12

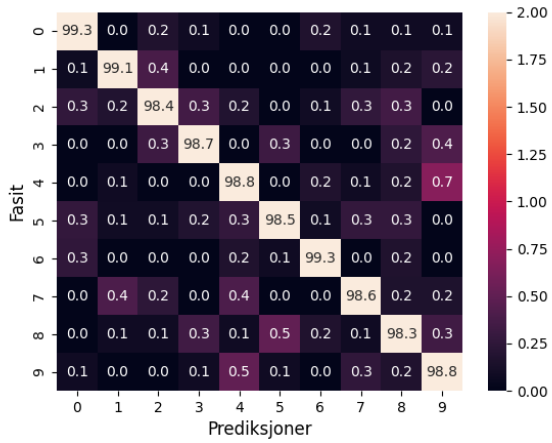


Figur 7 Nøyaktighet som funksjon av epoke for modell CNN 14 modifisert (ett konvolusjonslag, 16 filter, filterdimensjon 5, stride 1 og padding 2) utvidet med to fullt koblede skjulte lag på toppen med 500+150 noder for MNIST.

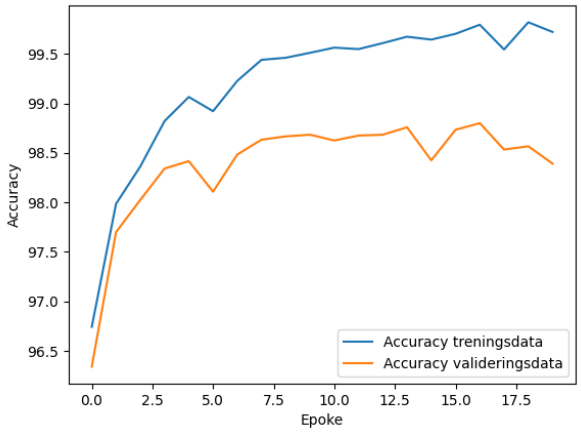
epoker. Feilmatrisen viser forstatt at 0,1 og 6 er de enklestesiffrene å predikere, men vi kan legge merke til at feilprediksjonene på de andre tallene er mer eller mindre borte i denne modellen. Dette tyder på at det har noe for seg å legge på fullt koblede skjulte lag på toppen av et konvolusjonsnettverk.

B. Konvolusjonsnettverk med to konvolusjonslag

For modellene med to konvolusjonslag ble det også benyttet 1, 4, 8 og 16 filtere i det siste av lagene, og halvparten i det første laget. Resultatene er gjengitt i tabell III. I tilfellet med to lag, ble ikke modellene med filter av dimensjon 1 benyttet, da resultatene fra utprøvingene



Figur 8 Feilmatrise for valideringsdataene etter 20 epoker for modell CNN 14 modifisert (ett konvolusjonslag, 16 filter, filterdimensjon 5, stride 1 og padding 2) utvidet med to fullt koblede skjulte lag på toppen med 500+150 noder for MNIST.



Figur 9 Nøyaktighet som funksjon av epoke for modell CNN 30 (to konvolusjonslag, 8+16 filter, filterdimensjon 5, stride 1 og padding 2) for MNIST.

med ett lag ble såpass dårlige som de ble. Dette gjaldt også modellene med filter av dimensjon 5 og stride 2.

Vi ser at to lag ikke utgjør en klar forbedring fra ett lag is seg selv, men at de tilsvarende modellene blir bedre med to lag. Den beste modellen er modell 30 som har hhv. 8 og 16 filtere i de to lagene, med filterdimensjon på 5, stride 1 og padding 2 i begge lag. Da oppnås en nøyaktighet på $98,39\%$ for valideringsdataene som er noe bedre enn tilsvarende modell med kun ett lag.

Ikke alle modellene rekker å konvergere i løpet av 20 epoker, men ved å plotte nøyaktighet som funksjon av epoker for test- og valideringsdata, se Vedlegg C, så det ut til at jo større modellene var, jo lettere konvergerer de. Modell 30, som peker seg ut som den beste av modellene med 2 konvolusjonslag uten fullt koblede skjulte lag ser ut til å konvergere etter rundt 15 epoker, se figur 9. Overraskende nok, så er denne modellen dårligst på å gjenkjenne siffrene 7 og 8(!), se feilmatriser for to-lags konvolusjonsnettverk i vedlegg C. Så det virker som at det å legge på et ekstra konvolusjonslag gjør at modellen sorterer på andre egnskaper ved siffrene enn ved bare ett lag, i den grad dette kan tillegges noe mening.

Det ble også gjort utprøvinger med et arkitektur som ligner på LeNet5 (Lecun *et al.*, 1998), som er benyttet på dette datasettet tidligere. LeNet5 er arkitektur for et konvolusjonsnettverk bestående av 7 lag, og aktiveringsfunksjonen som benyttes i de skjulte lagene i LeNet5 er *tanh*. Det første laget er et konvolusjonslag med et filter med dimensjon 5×5, stride på 1 og 6 filtere. Det er også lagt ett lag med padding (padding=2) rundt bildet i dette laget. I det neste laget gjennomføres en gjennomsnittspooling hvor det tas gjennomsnitt over 2×2 pixeler uten overlapp, slik at oppløsningen reduserers fra 28×28 til 14×14. Det tredje laget er et nytt konvolusjonslag med 16 kernels av dimensjon 5×5, uten padding og med stride 1, mens det fjerde laget er en ny gjennomsnittspooling over 2×2 pixeler slik at dimensjonen av hvert bilde reduseres til 5×5. Femte og sjette lag fullt koblede lag med hhv 120 og 84 noder. Deretter følger utdatalaget, hvor softmax benyttes som aktiveringsfunksjon. Dette ble implementert i min kode som beskrevet her. Min implementering av LeNet5 ga en modell med nøyaktighet på $98,44\%$ for valideringsdataene, mot de $99,05\%$ som er oppgitt i litteraturen (Lecun *et al.*, 1998). Forskjellene her *kan* skyldes at jeg har benyttet en annen kostfunksjon enn MSE, og at jeg har brukt en annen læringsrate i optimeringen. Men det er betryggende at det ikke ble heeelt annerledes i det minste.

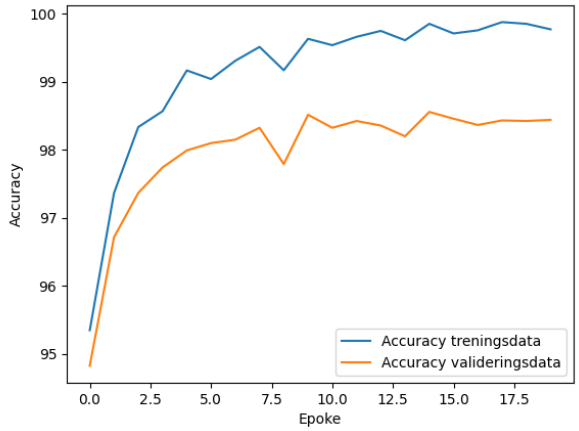
Figur 10 viser nøyaktigheten som funksjon av epoke for min implementering av LeNet5-arkitekturen. Her ser vi at vi ser ut til å nå konvergens etter 12-15 epoker, mot de 10-12 som er rapporter i litteraturen (Lecun *et al.*, 1998). Dette kan igjen skyldes at jeg har implementert en litt annen optimeringsalgoritme enn det som er benyttet der. Feilmatrisen for LeNet5-modellen indikerer at den er jevnt god på å preikere alle tall, men noe svakere på 5 og 9 (se Vedlegg C).

Inspirert av LeNet5, ble det også laget en modell som baserte seg på den beste av mine 2-lags konvolusjonsmodeller med den beste av mine 2-lags FFNN på toppen (500+150 noder), kalt *modell 30 modifisert*. Dette ga en nøyaktighet på $99,02\%$ for valideringsdataene, tilsvarende det som tidligere har blitt funnet med LeNet5 (Lecun *et al.*, 1998). Men vi kan legge merke til at tiden det tar å trene denne modellen er en del lengere enn det tar å trene LeNet5-nettverket.

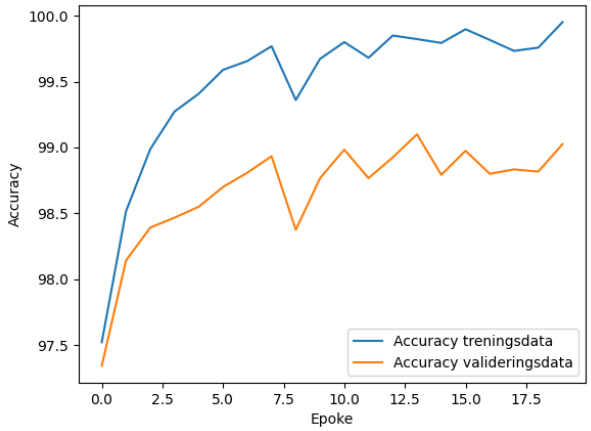
Figur 11 viser nøyaktigheten som funksjon av epoker for den modifiserte modell 30. Denne modellen ser ut til å være ganske konvergert etter 10-12 iterasjoner. Dess-

Modell	Tid for trening (min)	Nøyaktiget treningsdata (%)	Nøyaktiget valideringsdata (%)	Antall filtere	Dimensjon filter	Stride	Padding
17	11	11,27	11,11	1,1	3	1	1
18	12	93,67	93,03	1,1	5	1	2
19	-	-	-	1,1	5	2	2
20	-	-	-	1,1	1	1	0
21	12	97,76	97,11	2,4	3	1	1
22	13	98,42	97,76	2,4	5	1	2
23	-	-	-	2,4	5	2	2
24	-	-	-	2,4	1	1	0
25	13	99,08	98,08	4,8	3	1	1
26	14	99,16	98,20	4,8	5	1	2
27	-	-	-	4,8	5	2	2
28	-	-	-	4,8	1	1	0
29	16	99,43	98,10	8,16	3	1	1
30	18	99,72	98,39	8,16	5	1	2
31	-	-	-	8,16	5	2	2
32	-	-	-	8,16	1	1	0
LeNet5	17	99,77	98,44	6,16	5	1	2
30 modifisert	22	99,95	99,02	8,16	5	1	2

Tabell III CNN-modeller med to konvolusjonslag. Nøyaktigheten er oppgitt etter 20 epoker. Der det står ble ikke modellen benyttet for MNIST.



Figur 10 Nøyaktighet som funksjon av epoke for en arkitektur laget for å ligne LeNet5-arkitekturen. 2 konvolusjonslag og to fullt koblede konvolusjonslag for MNIST.

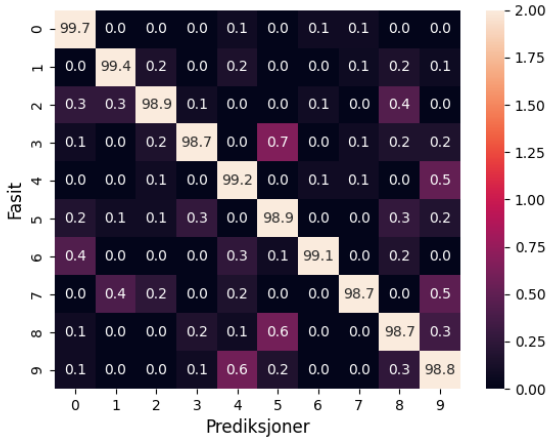


Figur 11 Nøyaktighet som funksjon av epoke for modell CNN 30 modifisert (to konvolusjonslag, 8+16 filter, filterdimensjon 5, stride 1 og padding 2) utvidet med to fullt koblede skjulte lag på toppen med 500+150 noder for MNIST.

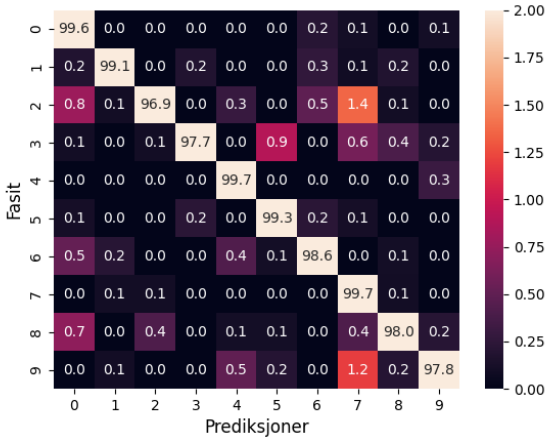
uten ser vi at selv om det er bedre nøyaktighet for treningsdataene enn for valideringsdataene, så følger disse hverandre. Dette tyder på at det er lite bias i modellen. Feilmatrisen er vist i firugr12, og vi ser at også denne modellen gjør det klart best for siffrene 0, 1 og 6 og dårligst for 5 og 9. Men modellen har en presisjon på 97% eller bedre for alle siffer.

IX. RESULTATER FOR MNIST OG FASHION MNIST

Under utprøvingene ble det funnet at en arkitektur med to konvolusjonslag og to påfølgende fullt koblede skjulte lag ga den beste modellen, kalt *modell 30 modifisert*. Denne modellen så ut til å være konvergert etter 12 epoker, og det ble derfor besluttet å prøve ut dette oppsettet på både MNIST testdata og Fashion MNIST. I tillegg ble det besluttet å gjøre tilsvarende utprøvinger med den beste arkitekturen for FFNN, modell 5, og med logistisk regresjon, for sammenligning.



Figur 12 Feilmatrise for valideringsdataene etter 20 epoker for modell CNN 30 modifisert (to konvolusjonslag, 8+16 filter, filterdimensjon 5, stride 1 og padding 2) utvidet med to fullt koblede skjulte lag på toppen med 500+150 noder for MNIST.



Figur 13 Feilmatrise for testdataene etter 12 epoker for modell CNN 30 modifisert (to konvolusjonslag, 8+16 filter, filterdimensjon 5, stride 1 og padding 2) utvidet med to fullt koblede skjulte lag på toppen med 500+150 noder på MNIST-data.

A. MNIST

Resultatene av utprøvningene på MNIST-dataene er vist i tabell IV. Det blir klart at CNN gir den beste modellen for å kjenne igjen siffrene i MNIST-datasettet. Fra tabellen ser vi blant annet at det ikke virker å være nevneverdig forskjell i nøyaktigheten for valideringsdataene og testdataene for noen av metodene, noe som tyder på at valget av modell ikke har bias grunnet valideringsdataene. Derimot ser vi også at nøyaktigheten etter 12 epoker er dårligere enn den var etter 20 epoker, sånn at det er naturlig å anta at modellen ikke her helt konvergert. I Vedlegg D finnes plott av nøyaktighet som funksjon av epoker, og disse tyder likevel på at kjøringene er konvergert.

Figur 13 viser feilmatrisen for testdataene for CNN modell 30 modifisert, som er den beste modellen. Her ser vi at modellen har størst presisjon på gjenkjennelse av sifferne 0, 1 4 og 7, men også at 2 og 9 relativt ofte feilpredikeres som 7. Det er ikke nødvendigvis enkelt å tolke dette, men det er verdt å merke seg at dette er annerledes enn det som ble funnet under utprøvingen etter 20 epoker.

B. Fashion MNIST

Resultatene av utprøvningene på Fashion MNIST er vist i tabell V. Her ser vi at modellene gjør en dårligere jobb på dette mer komplekse datasettet enn de gjør på MNIST. Det er også større forskjell på nøyaktigheten for treningsdataene og validerings og testdataene enn det er for MNIST. Men også for Fashion MNIST gir CNN-metoden et bedre resultat enn de andre metodene. Ved å se på hvordan nøyaktigheten for treningsdataene og valideringsdataene utvikler seg gjennom epokene (se plott i vedlegg D), ser vi at disse modellene ikke ser ut til lå være helt konvergert etter 12 epoker, noe som kan forklare hvorfor modellene er så dårlige. En utprøving med CNN modell 30 modifisert med 20 epoker ga en nøyaktighet på 91.20% for testdataene, selv om nøyaktigheten for treningsdataene ble 98, 36% noe som tyder på at denne modellen er overtilpasset, da den konvergerte etter ca 8 epoker. Tidligere beregninger med 2-lags konvolusjonsnettverk på Fashon MNIST har gitt nøyaktighet på 91, 6% (Fas, nd), noe som er svakt bedre enn det jeg får til her på tross av at denne arkitekturen egentlig er tilpasset MNIST.

Av feilmatrisene for modellene (se Vedlegg D) ser vi også tydelig at CNN-modellen gir en mye bedre presisjon i sine prediksjoner enn de to andre modellene. Feilmatrisen for CNN-modellen er gitt i figur 14. Her blir det tydelig at modellen helt klart er best på å kjenne igjen sandaler, bukser og vesker. Det er jo kanskje meningsfylt siden disse ikke ligner veldig på de andre typene plagg i bildene. Vi ser også at modellen oftest bommer ved å si at ting er T-skjorter/topper, jakker eller skjorter, mens skjorter, kjoler og gensere er det den synes er vanskeligst å kjenne igjen. Skjorter, T-skjorter/topper, jakker, gensere og kjoler er jo plagg som har ganske lik fasong, så dette er kanskje ikke helt overraskende.

C. Sammenligning av modellering på MNIST og Fashion MNIST

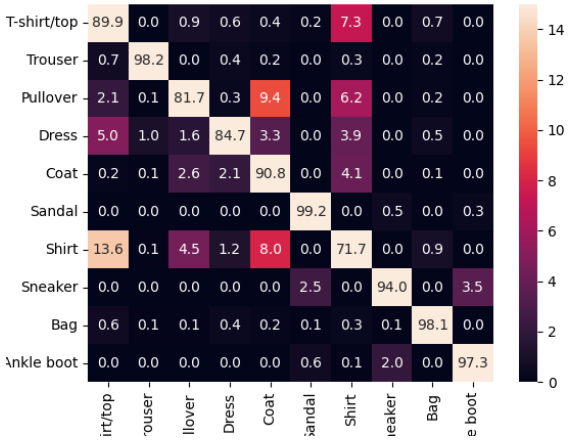
Figur 15 viser nøyaktigheten for valideringsdataene for alle de tre modellene (logisitisk regresjon, FFNN 5 og CNN 30 modifisert) som funksjon av epoker i de endelige

Modell	Nøyaktiget treningsdata (%)	Nøyaktiget valideringsdata (%)	Nøyaktiget testdata (%)
Logistisk regresjon	93,27	92,08	92,62
FFNN 5	99,63	97,56	97,96
CNN 30 modifisert	99,68	98,77	98,65

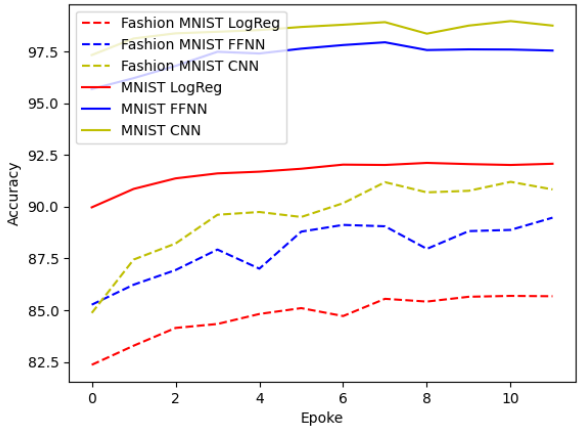
Tabell IV Resultater av utprøvinger av logistisk regresjon, den beste arkitekturen for FFNN (modell 5) og den beste arkitekturen for CNN (modell 30 modifisert) etter 12 epoker på MNIST datasettet.

Modell	Nøyaktiget treningsdata (%)	Nøyaktiget valideringsdata (%)	Nøyaktiget testdata (%)
Logistisk regresjon	86,92	85,67	84,56
FFNN 5	93,07	89,48	88,66
CNN 30 modifisert	95,64	90,84	90,56

Tabell V Resultater av utprøvinger av logistisk regresjon, den beste arkitekturen for FFNN (modell 5) og den beste arkitekturen for CNN (modell 30 modifisert) etter 12 epoker på Fashion MNIST.



Figur 14 Feilmatrise for testdataene etter 12 epoker for modell CNN 30 modifisert (to konvolusjonslag, 8+16 filter, filterdimensjon 5, stride 1 og padding 2) utvidet med to fullt koblede skjulte lag på toppen med 500+150 noder på Fashion MNIST

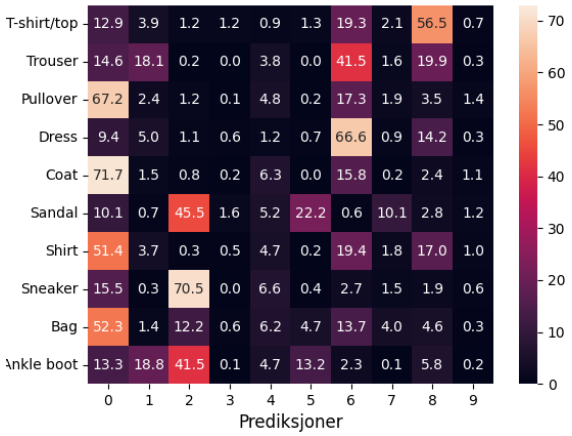


Figur 15 Sammenligning av nøyaktighet for valideringsdataene for de tre endelige modellene for MNIST og Fashion MNIST.

kjøringene. Vi ser tydelig at Fashion MNIST er et mer komplekst datasett som det er vanskeligere for modellene å gjøre prediksjoner på gjennom at det gjennomgående har lavere nøyaktighet enn prediksjonene på MNIST. For begge datasettene ser vi uansett at nevrale nettverk utgjør en klar forbedring i forhold til logistisk regresjon, og at konvolusjonsnettverk, som ser etter sammenhenger mellom nærliggende pixeler gjør en bedre jobb enn generelle FFNN.

Bare for gøy matet jeg Fashion MNIST datasettet inn i modellen som var trent på MNIST (CNN modell 30 modifisert). Konvolusjonsmatrisen for prediksjonene er vist i figur 16. Her ser vi at sko (sandaer, sneakers og ankle boots) og vesker blir vurdert som tallet 2, mens de fleste andre klær blir vurder som enten 0, 6 eller 8. Dette er ikke egentlig meningsfylt, men litt artig. Og det kan tyde på at noen av filterne som trenes i modellen for MNIST identifiserer noe som også utgjør grunnformer i klær og sko. Men hvorfor sko blir 2, er ikke åpenbart for meg likevel.

Dette illustrerer at selv om koden benyttet for å modellere på MNIST og Fashion MNIST er identiske, så blir modellene som trenes ulike, og det er viktig at nevrale nettverk som skal er trent til å utføre den oppgaven de



Figur 16 Kødd

skal brukes til. I min implementering blir prediksjonene satt til å være den klassen som har høyest sannsynlighet, uavhengig av hvor stor denne sannsynligheten er. En alternativ løsning kunne vært at modellen nektet å gjøre klassifiseringen dersom enten ingen sannsynligheter var over en gitt terskelverdi eller flere klasser hadde omtrent lik sannsynlighet. Eventuelt så kunne modellen istedenfor å foreta klassifiseringen oppgitt sannsyligheten for f.eks. de tre mest sannsynlige klassifiseringene. Dette kjenner vi fra tilgjengelige bilegjenkjenningsprogrammer som for eksempel Artsorakelet (Artsdatabanken, nd).

X. OPPSUMMERING OG KONKLUSJON

Hensikten med denne oppgaven har vært å prøve ut og sammenligne logistisk regresjon og ulike arkitekturer for FFNN og CNN til bruk i bildegjenkjenning. De ulike metodene og arkitekturene har vært testet ut på datasettet MNIST (Lecun *et al.*, 1998) ved å dele opp treningsdataene i et treningssett og et valideringssett som inneholdt 20% av de opprinnelige treningsdataene. Her ble det funnet at logistisk regresjon ga en nøyaktighet på $96,97\%$ på valideringssettet, mens den beste FFNN med to skjulte lag med 500+150 noder ga en nøyaktighet på $97,99\%$ og det beste CNN-nettet med to konvolusjonslag med $8 + 16$ filtere med påfølgende 2 skjulte koblede lag med 500+150 noder ga en nøyaktighet på $99,02\%$, som er sammnelignbart med det vi finner i litteraturen for tilsvarende modeller (Lecun *et al.*, 1998).

Disse modellene ble så anvendt på testsettene for MNIST og Fashion MNIST (Xiao *et al.*, 2017). Her ble det klart at modellene for MNIST-dataene presterer omtrent likt for valideringssettet og test-settet, noe som bidrar til gyldigheten av utprøvingene over. Dessuten ser vi tydelig at modellene presterer dårligere på det mer kompliserte Fashion MNIST-settet, som sannsynligvis krever en mer kompleks modell for å prestere godt. Likevel ble det oppnådd en nøyaktighet på $90,56\%$ for det største konvolusjonsnettverket, noe som er sammenlignbart med resultatene som er rapportert tidligere for Fashion MNIST (Fas, nd).

Modellene som er benyttet i dette arbeidet har benyttet gradientmetoden Adam, med defaultparameterne som ligger inne i pytorch. Det er ikke testet ut med ulike metaparametere, større arkitekturer, andre valg av aktiveringsfunksjoner, regulariseringsparameter osv. Det er heller ikke benyttet preprosessering av datasettet. Optimering av disse metaparameterne og bruk av preprosessering vil kunne bidra til å gjøre modellene enda bedre enn det vi ser her. Likevel kommer resultatene svært nært det som er vist tidligere i litteraturen, og bidrar til å belyse hvordan de ulike maskinglæringsalgoritmenes oppbygning påvirker funksjonaliteten til modellene som trenes.

REFERANSER

(n.d.), "Fashion-mnist," .
N. B. C. Artsdatabanken (n.d.), "Artsorakelet," .
D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber (2010), "Deep, big, simple neural nets for handwritten digit recognition," Neural Computation **22** (12), 3207–3220, https://direct.mit.edu/neco/article-pdf/22/12/3207/842857/neco_a_00052.pdf.
I. Goodfellow, Y. Bengio, and A. Courville (2016), *Deep Learning* (The MIT Press, Cambridge, Massachusetts).
C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant (2020), "Array programming with NumPy," Nature **585** (7825), 357–362.
M. Hjorth-Jensen (2021), "Applied data analysis and machine learning," .
D. P. Kingma, and J. Ba (2017), "Adam: A method for stochastic optimization," arXiv:1412.6980 [cs.LG].
Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner (1998), "Gradient-based learning applied to document recognition," Proceedings of the IEEE **86** (11), 2278–2324.
A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala (2019), "Pytorch: An imperative style, high-performance deep learning library," Cite arxiv:1912.01703Comment: 12 pages, 3 figures, NeurIPS 2019.
F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011), "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research **12**, 2825–2830.
B. Poyekar (2024), "Building simple neural networks using

pytorch (nn, cnn) for mnist dataset," Accessed at: 2024-11-20.

S. Raschka (n.d.), "How is stochastic gradient descent implemented in the context of machine learning and deep learning?" .

S. Raschka, Y. H. Liu, V. Mirjalili, and D. Dzhulgakov (2022), *Machine Learning with Pytorch and Scikit-Learn: Develop Machine Learning and Deep Learning Models with Python*, 1st ed. (Packt Publishing, Limited, Birmingham).

M. L. Waskom (2021), "seaborn: statistical data visualization," Journal of Open Source Software **6** (60), 3021.

H. Xiao, K. Rasul, and R. Vollgraf (2017), "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," arXiv:1708.07747 [cs.LG].

C. J. B. Yann LeCun, Corinna Cortes (n.d.), "The mnist database of handwritten digits," .