Klassifisering av bilder med nevrale nettverk

Siv Aalbergsjø

I denne oppgaven er det 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 ble testet på datasettene MNIST og Fashion MNIST. Som forventet presterte CNN bedre enn logistisk regresjon og FFNN på klassifisering av bildedata. Det ble funnet at logistisk regresjon ga en nøyaktighet på 96, 97% for MNIST, mens den beste FFNN med to skjulte lag ga en nøyaktighet på 97, 99% og det beste CNN-nettet med to konvolusjonslag ga en nøyaktighet på 99, 02%. For Fashion MNIST ble det også funnet at CNN 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. F.eks. har vi via våre mobiltelefoner tilgang til *Artstorakel* (Artsdatabanken, nd) som kan identifisere arter fra norsk natur gjennom et bilde. 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. 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 to eller flere typer objekter/elementer i bildene.

Hensikten med oppgaven er å undersøke hvordan oppbygningen til klassifiseringsmetodene påvirker hvor godt de løser oppgaven med klassifisering av bilder. 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. Deretter testes metodene på det 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 klassifisere 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. Så 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 overvåket læring. Logistisk regresjon er en forholdsvis enkel måte å lage en modell som kan klassifisere data. Metoden er godt egnet for problemer hvor klassene kan separeres lineært (Raschka *et al.*, 2022). Metoden er beskrevet i mer detalj i Vedlegg A. I logistisk regresjon benyttes en sannsynlighetsmodell for å regne ut sannsynligheten for at datapunktet vårt hører til i en gitt klasse. I flerklasseproblemer benyttes *softmax*-funksjonen til dette. Da er sannsynligheten for hver klasse gitt ved:

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

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. Optimering, eller trening, av modellen består så i å finne det beste settet med parametere $\boldsymbol{w}$ og $\boldsymbol{b}$. Totalt sett får vi en kostfunksjon gitt ved:

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

kalt *cross entropy*. Når beste parametere er funnet kan modellen benyttes til å klassifisere ukjente tilfeller gjennom å beregne sannsynligheten $p(\boldsymbol{x})$. 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):

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

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

Gradientmetoder er gjort redefor i prosjekt 2.

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 representerer inversen av den andrederiverte av funksjonen og utgjør en metaparameter i algoritmen. **b** oppdateres på tilsvarende måte.

Stokastisk gradient descent (SGD) er en måte redusere tidskostnaden på. Her evalueres gradienten i hver iterasjon på en tilfeldig valgt batch av dataene. I minibatch stokastisk gradient descent (Raschka, nd) deles datasettet inn i minibatcher, og iterasjonene deles inn i *epoker*. For hver iterasjon benyttes én minibatch, mens en epoke utgjør antallet iterasjoner som trengs for å gå gjennom hele datasettet. 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 beregne denne andrederiverte, tilnærmes denne gjennom metaparameteren som kalles *læringsraten* η. 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 lærignsraten, se Vedlegg B for matematiske uttrykk.

IV. NEVRALE NETTVERK

Forovermatede 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 hver lag i det nevrale nettverket kalles i det følgende for **z**, mens utdataene fra hvert lag betegnes med **a**. Inndataene til modellen,**x**, kan betraktes som utdata fra det nulte laget. Mellom to lag gjøres det først en lineær operasjon på dataene hvor utdata fra lag (i-1) blir til inndata til lag (i), 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 som gir modellens prediksjoner. Lagene mellom inndatalaget og utdatalaget kalles skjulte lag. Et nevralt nettverk kan ha mange eller få skjulte lag. Trening av nettverket består i å optimere **W** og **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 et klassifiseringsproblem. For de skjulte lagene derimot, står vi fritt til å velge dimensjon og hvor mange lag vi ønsker. Hvor stort og dypt nettverk som trengs (antall noder og antall lag), avhenger av kompleksiteten i problemet som skal modelleres. Oppbygnignen av kalles arkitekturen til det nevrale nettverket.

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.

I et nevralt nettverk initieres modellen ved at $\boldsymbol{W}$ og $\boldsymbol{b}$ initieres med (tilfeldig) valgte verdier. Trening av nettverket består i å iterativt propagere fremover gjennom lagene og gjøre prediksjoner med modellen, beregne kostfunksjonen og så beregne gradientene til parameterne gjennom å propagere bakover gjennom nettverket, såkalt tilbakepropagering (Rumelhart and Williams, 1986), for så å oppdatere $\boldsymbol{W}$ og $\boldsymbol{b}$ med en optimeringsalgoritme og gjenta prosessen til vi oppnår konvergens. Dette er beskrevet nøyere i prosjekt 2.

A. Aktiveringsfunksjoner, kostfunksjoner, nøyaktighet i nevrale nettverk

I tillegg til å variere antall skjulte lag og noder, påvirker valg av aktiveringsfunksjoner kostfunksjon 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*.

For de skjulte lagene står man fritt til å velge ulike typer aktiveringsfunksjoner.ReLU er en mye benyttet aktiveringsfunksjon:

$$a_{ReLU}(z) = max(0, z).$$

Denne funksjonen tar samme verdier som inndataene dersom disse er positive, men gir 0 hvis inndataene er negative. Dersom man skal ha mange lag i nettverket sitt, er dette en god aktiveringsfunksjon citeRaschkaSebastian2022MLwP.

En annen mye benyttet aktiveringsfunksjoner hyperbolsk tangens:

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

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 (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 andelen riktige prediksjoner. Det kan også være interessant å lage en såkalt *feilmatrise* som inneholder informasjon om andel rett og galt klassifiserte tilfeller for hvert utfall.

B. Konvolusjonsnettverk

Konvolusjonsnettverk (CNN) kan sies å utgjøre en utvidelse av FFNN hvor det er lagt til flere lag mellom inndataene og de fullt koblede lagene. Konvolusjonslagene er ikke fullt koblede. Hver node i et lag forholder seg kun til et utvalg av nodene i laget før, de som ligger nær hverandre. 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.

Bilder kan ha høy oppløsning, hver piksel utgjør en feature i datasettet. Fordi CNN utgjør en systematisk reduksjon i dimensjonen av dataene, gjennom å evaluere piklser sammen i grupper, kan disse nettverkene være mer effektive for å gjøre f.eks. bildegjenkjenning. Derfor forventes det at et mindre CNN kan gjøre en bedre jobb med bildegjenkjenning enn et større FFNN.

1. Konvolusjoner

I en konvolusjon anvendes et *filter* (kernel) på inndataene for å produsere utdata. Dette er en lineær operasjon som utføres på bare en del av inndataene om gangen og erstatter den lineære operasjonen mellom fullt koblede lag i et FFNN. For et bilde kan vi betrakte dette som et filter av en gitt dimensjon som legges over et bilde og flyttes systematisk over bildet. Det kan benyttes mange filter i hvert lag. Fordi filtert 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).

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

Fordi filtert har en utstrekning som er større enn én piksel, vil dimensjonen til det nye bildet være mindre enn det opprinnelige, dersom det ikke er mulig å plassere filtert 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.

2. Pooling

Konvolusjonen, kan i praksis utgjøre en utvidelse av datamengden 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 piksler uten overlapp. Dette vil utgjøre en reduksjon i datamengden 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 ekstrahere informasjon om strukturer i et bilde uavhengig av deres plassering. Dette kan benyttes som inndata til et FFNN eller en logistisk regresjon for å klassifisere bildet som analyseres.

V. DATASETT SOM ER BENYTTET

I denne oppgaven er det benyttet to datasett. MNIST (Lecun *et al.*, 1998) er et datasett som består av 70 000 bilder av håndskrevne siffer mellom fra 0 til 9. Datasettet er sortert i 60 000 bilder i et treningssett, og 10 000 bilder i et testsett. Bildene har oppløsning på 28×28 piksler 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 ble laget i forbindelse med at konvolusjonsnettverk ble presentert som en effektiv og nøyaktig metode for klassifisering av bilder (Lecun *et al.*, 1998). Datasettet er senere benyttet til benchmarking av ulike maskinlæringsalgoritmer for klassifisering av bilder (LeCun *et al.*, nd).

Med logistisk regresjon, og uten preprosessering av data er det funnet at klassifisering av MNIST gir en feilrate på 12%, noe som tilsvarer en nøyaktighet på 88%(Lecun *et al.*, 1998). For FFNN er de beste resultatene oppnådd med 5 skjulte lag (Cireşan *et al.*, 2010) med en feilrate på $0,35\%$, dvs. $99,65\%$ nøyaktighet. Av de mindre FFNN er det funnet at ett skjult lag med 300 noder uten preprosessering gir en nøyaktighet på $95,3\%$, mens 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 $1000 + 150$ noder ga en nøyaktighet på $97,05\%$. Alle disse er gjort uten preprosessering av dataene og med MSE som kostfunksjon.

LeNet5, er en arkitektur for et CNN (beskrevet lengere nede), som også er benyttet for å gjøre klassifiseringer av MNIST. Denne arktiekturen har to konvolusjonslag med hhv. 6 og 16 filter og to påfølgende skjulte lag med hhv. 120 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 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*, *Sandal*, *Shirt*, *Sneaker*, *Bag* eller *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

I dette arbeidet er 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 nett (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 noe kode fra prosjekt 1 og 2.

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. 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 er 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 batchene. For hver batch gjøres prediksjoner (scores beregnes), kostfunksjonen be-

regnes og gradienter finnes gjennom tilbakepropagering før parameterne endres i et optimaliseringssteg. For hver epoke er det beregnet nøyaktighet for treningsdata og valideringsdata som er plottet og benyttet for å vurdere om modellen er konvergert og/eller overtrent.

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. Til slutt ble det gjort kjøringer med de beste arkitekturene fra CNN og FFNN, samt logistisk regresjon for både MNIST og Fashion MNIST. Her ble test-dataene 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.

A. Benyttede arkitekturer for nevrale nettverk

1. LeNet5

LeNet5 (Lecun *et al.*, 1998) er en CNN-arkitektur som er benyttet for å gjøre klassifisering av MNIST tidligere. Derfor ble det i denne oppgaven også implementert en arkitektur som ligner på denne. LeNet5 består av 7 lag, og aktiveringsfunksjonen som benyttes i de skjulte lagene er *tanh*. Det første laget er et konvolusjonslag med et filter med dimensjon 5×5, stride på 1 og 6 filter. 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 piksler 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 piksler 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.

2. FFNN

For FFNN ble det prøvd ut modeller med ett og to skjulte lag, og med ulike antall noder i de skjulte lagene. 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. 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 ble et oppsett med 120+84 skjulte noder testet, som tilsvarer de fullt koblede lagene som inngår i LeNet5-arkitekturen (Lecun *et al.*, 1998).

3. CNN

I tillegg til LeNet5, ble det prøvd ut konvolusjonsnettverk med én og to konvolusjoner, og kun ett fullt koblet lag etter konvolusjonslagene (utdatalaget). Dimensjon på filter, 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 filter, filterdimensjoner på 1, 3 og 5 piksler, og stride parametere på 1 eller 2 og padding tilpasset dimensjonen på filter og stride. For modellene med to konvolusjonslag ble det benyttet 1, 4, 8 og 16 filter i det siste av lagene, og halvparten i det første laget. Fullstendig oversikt over arkitekturene finnes i tabellene II og III.

VII. UTPRØVING AV MODELLER FOR MNIST

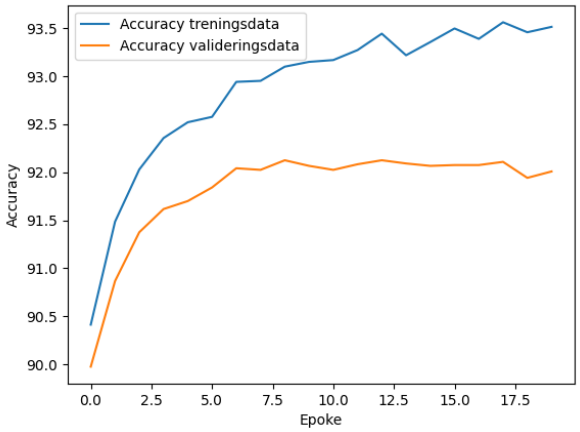
A. Logistisk regresjon

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 ikke en spesielt godt, men det er likevel bedre enn 88% 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.

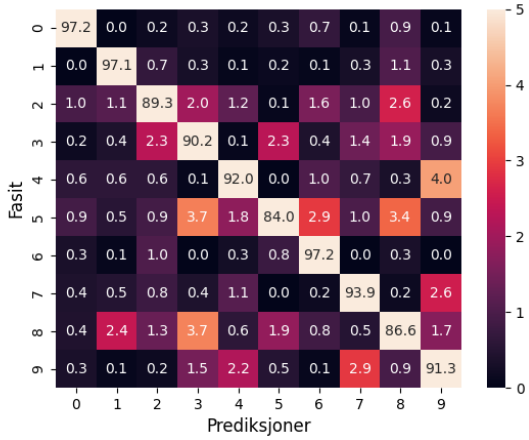
Figur 2 viser feilmatrisen etter 20 epoker med logistisk regresjon. Vi ser av figuren at modellen gjør det best for tallene 0, 1 og 6, men dårligst for 5 og 8. Tallet 5 ser ut

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 for MNIST. Nøyaktigheten er oppgitt etter 20 epoker.



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



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

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.

B. FFNN

Som vi ser av tabell I oppnår vi med FFNN en nøyaktighet på nær 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 med, 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 dårligere jobb enn de større, hvor det minste nevrale nettet gir en nøyaktighet for valideringsdataene på 96, 97%, mens FFNN 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 trenings- og valideringsdataene for FFNN Modell 5. 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 ikke overtrent. Tilsvarende plott for de andre modellene som er FFNN finnes i Vedlegg C.

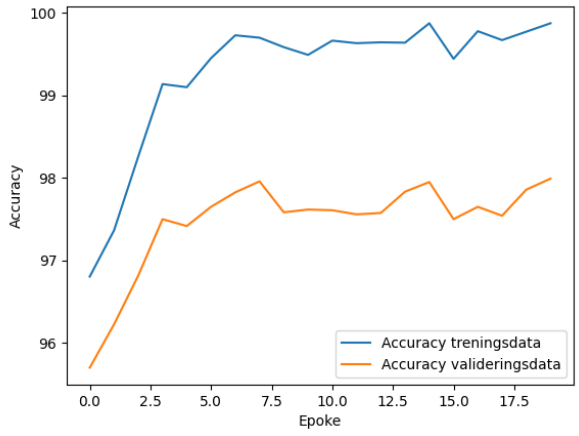
Figur 4 viser feilmatrisen for FFNN Modell 5. Her ser vi at også denne modellen er best til å gjenkjenne 0-ere og 6-ere, men at den også gjør det jevnt mye bedre på alle siffer enn modellen laget med logistisk regresjon. Igjen er det 8 som peker seg ut som det vanskeligs å gjenkjenne. Feilmatrisene for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

C. Konvolusjonsnettverk med ett konvolusjonslag

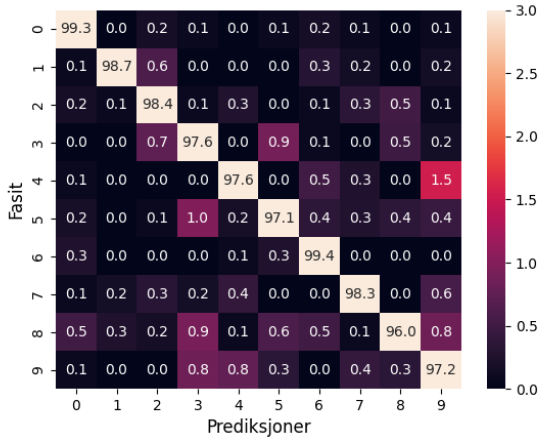
Oversikt over arkitekturer med ett konvolusjonslag er gitt i tabell II sammen med resultatet av utprøvingene. Her ser vi at at selv ett konvolusjonslag ga bedre resultater enn de største FFNN beskrevt over, selv om dette

Modell	Tid for trening (min)	Nøyaktiget treningsdata (%)	Nøyaktiget valideringsdata (%)	Antall filter	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 for MNIST. Nøyaktigheten er oppgitt etter 20 epoker. Der det står ble ikke modellen benyttet.



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



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

ikke gjelder alle modellene.

Modellene med filter av dimensjon 1 ga elendige resultater. Det ble noe bedre med flere filter, 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 over elementer i bildet. Dette er nok mer aktuelt for MNIST-dataene enn for andre bilder, da disse har svært grov oppløsning. Likevel er det verdt 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) med andre modeller som tok like lang tid, ser vi at modellen med stride 2 hadde en bedre nøyaktighet (se tabell II). 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 av tabell II at CNN Modell 2 med dimensjon 5, med padding på 2 og stride på 1 ga de beste resultatene

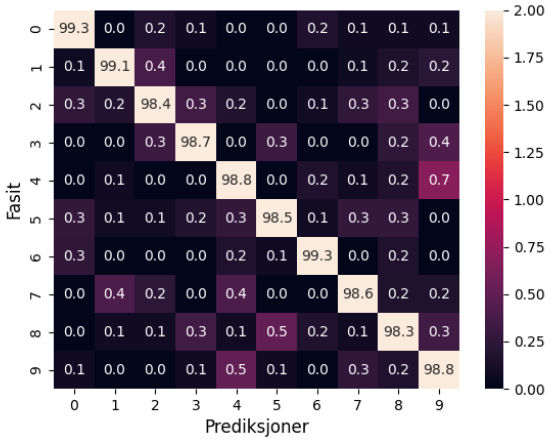
for ett filter. Vi ser også at flere filter ga bedre modell, og CNN Modell 14 med 16 filter ga de beste resultatene, med en nøyaktighet på 98, 22% for valideringsdataene.

Plott av nøyaktigheten til treningsdataene og valideringsdataene som funksjon av epoker for CNN med ett lag finnes i Vedlegg D. CNN Modell 14 ser 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.

Ved å se på feilmatrisene for CNN Modell 14 (se Vedlegg D) ser man at også her er prediksjonene av tallene 0, 1 og 6 mest presise, mens 3, 8 og 9 skiller seg ut som dårlige. Feilmatrisene til de andre modellene med ett konvolusjonslag finnes også vedlagt i Vedlegg D, for den glade spekulant.

Det ble også testet en modell som var CNN 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 5 feilmastrisen etter 20 epoker for den CNN Modell 14 modifisert. Denne viser at også her er 0, 1 og 6 de enkleste å predikere, men vi kan legge merke til at feilprediksjonene på de andre tallene er mer eller mindre borte med denne modellen. Dette tyder på at det har noe for seg å legge på fullt koblede skjulte lag på toppen av et konvolusjonsnettverk. Av plott av nøyaktigheten som funksjon av epoke (se Vedlegg D) kan vi se at modellen ser ut til å være ganske konvergert etter 10-12 epoker.



Figur 5 Feilmatrise for valideringsdataene etter 20 epoker for CNN Modell 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.

D. Konvolusjonsnettverk med to konvolusjonslag

I tilfellet med to konvolusjonslag, ble ikke modellene med filter av dimensjon 1 benyttet, da resultatene fra utprøvingene med ett lag ble såpass dårlige som de ble. Dette gjaldt også modellene med filter av dimensjon 5 og stride 2. Oversikt over arkitekturene som ble benyttet og resultatene er gjengitt i tabell III.

Vi ser at to lag ikke utgjør en klar forbedring fra ett lag i seg selv, men at de tilsvarende modellene blir bedre med to lag. Den beste modellen er CNN Modell 30 som har hhv. 8 og 16 filter 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 bedre enn tilsvarende modell med kun ett lag.

Ikke alle modellene rekker å konvergere i løpet av 20 epoker (se plott av nøyaktighet som funksjon av epoke i Vedlegg E). Men det ser ut til at jo større arkitekturene er, jo lettere konvergerer de. CNN Modell 30 peker seg ut som den beste av modellene med 2 konvolusjonslag uten fullt koblede skjulte lag. Denne ser ut til å konvergere etter rundt 15 epoker.

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. CNN Modell 30 predikterer overraskende nok best tallene 1, 2 og 6 og er dårligst på å gjenkjenne siffrene 7 og 8, se feilmatriser for to-lags konvolusjonsnettverk i Vedlegg E.

Min implementering av LeNet5 ga en 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 *heeeelt* annerledes i det minste.

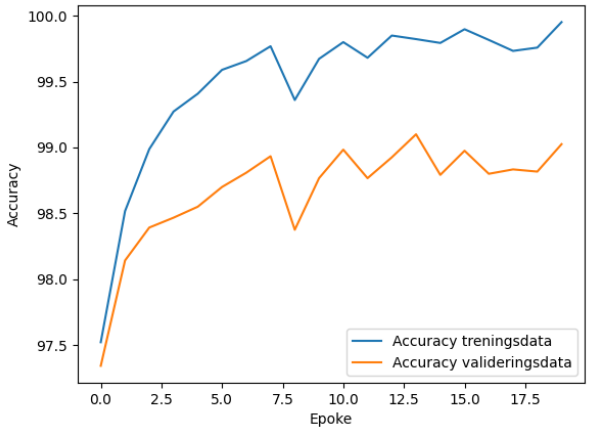
Nøyaktigheten som funksjon av epoke for min implementering av LeNet5-arkitekturen finnes i Vedlegg E. Denne tyder på konvergens etter 12-15 epoker, mot de 10-12 som er rapporter i litteraturen (Lecun *et al.*, 1998). Dette kan skyldes at jeg har implementert en litt annen optimeringsalgoritme enn det som er benyttet der. Feilmatrisen for LeNet5-modellen indikerer at den er jevnt god på å predikere alle tall, men noe svakere på 5 og 9 (se Vedlegg E).

Inspirert av LeNet5, ble det også laget en modell som baserte seg på den beste av mine 2-lags konvolusjonsmodeller (CNN Modell 30) med den beste av mine 2-lags FFNN på toppen (500+150 noder), kalt *CNN Modell 30 modifisert*. Denne 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.

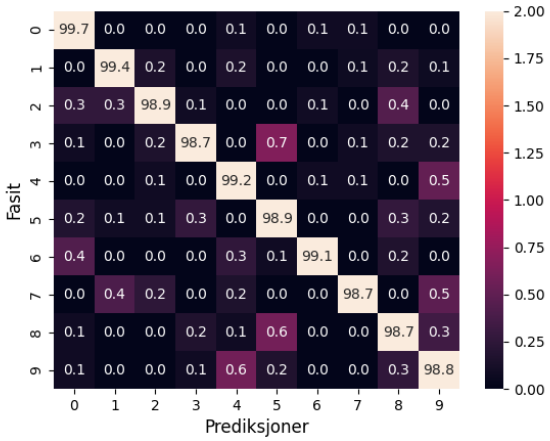
Figur 6 viser nøyaktigheten som funksjon av epoker for

Modell	Tid for trening (min)	Nøyaktiget treningsdata (%)	Nøyaktiget valideringsdata (%)	Antall filter	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 6 Nøyaktighet som funksjon av epoke for CNN Modell 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.

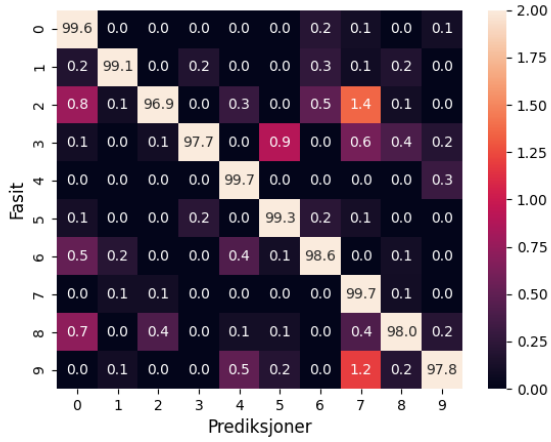


Figur 7 Feilmatrise for valideringsdataene etter 20 epoker for CNN Modell 30 modifisert (to konvolusjonslag, 8+16 filter, filterdimensjon 5, stride 1 og padding 2, utvidet med to fullt koblede skjulte lag på toppen med 500+150 noder) for MNIST.

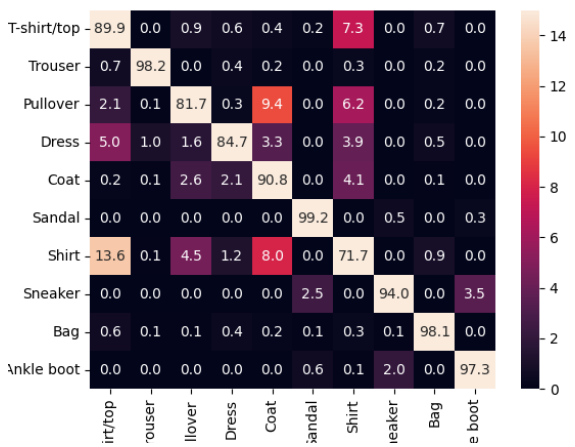
den modifiserte modell 30. Denne modellen ser ut til å være ganske konvergert etter 10-12 iterasjoner. Dessuten ser vi at selv om det er bedre nøyaktighet for treningsdataene enn for valideringsdataene, så følger disse hverandre, slik at den ikke virker overtrent. Feilmatrisen er vist i figur 7, og vi ser at også denne modellen gjør det klart best for 0, 1 og 6 og dårligst for 5 og 9. Men modellen har en presisjon på 97% eller bedre for alle siffer.

VIII. 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 *CNN Modell 30 modifisert*. Denne modellen så ut til å være konvergert etter 10-12 epoker, og det ble derfor besluttet å prøve ut dette oppsettet med 12 epoker på testdata for både MNIST og Fashion MNIST. I tillegg ble det besluttet å gjøre tilsvarende utprøvinger med den beste arkitekturen



Figur 8 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) for MNIST.



Figur 9 Feilmatrise for testdataene etter 12 epoker for CNN Modell 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 Fashion MNIST

for FFNN (FFNN Modell 5) og med logistisk regresjon, for sammenligning.

A. MNIST

Resultatene av utprøvningene på MNIST er vist i tabell IV. Det blir klart at CNN gir den beste modellen for å kjenne klassifisere bildene i MNIST. Det virker ikke å være nevneverdig forskjell i nøyaktigheten for valideringsdataene og testdataene for noen av metodene. Dette 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 modellene ikke her helt konvergert. I Vedlegg F finnes plott av nøyaktighet som funksjon av epoker, og disse tyder likevel på at kjøringene er konvergert.

Figur 8 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 0, 1 4 og 7, men også at 2 og 9 relativt ofte feilpredikeres som 7. 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, noe som kan tyde på overtrenthet. 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 F), ser vi at disse modellene ikke ser ut til å være helt konvergert etter 12 epoker. Dette 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 overtrent, 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. En forskjell på mitt arbeid og tidligere arbeider er at jeg har splittet treningsdataene inni test- og valideringssett. Dette var ikke nødvendig å gjøre for Fashion MNIST, siden jeg ikke har gjort valideringer med dette datasettet, en ble bare gjort fordi jeg gjenbrukte samme kode som hadde vært benyttet for MNIST.

Av feilmatrisene for modellene (se Vedlegg F) 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 9. 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.

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

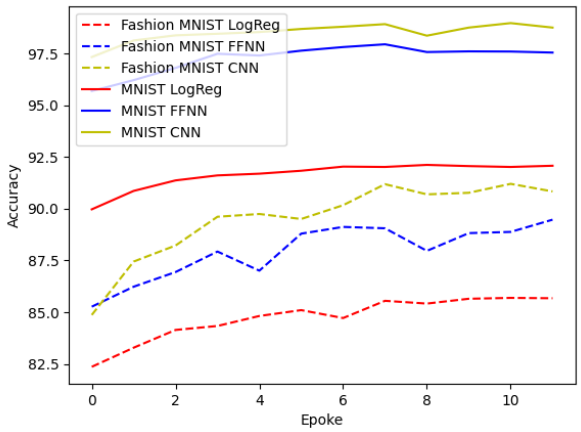
Tabell IV Resultater av utprøvinger av logistisk regresjon, FNN Modell 5 og CNN Modell 30 modifisert etter 12 epoker på MNIST.

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, FNN Modell 5 og CNN Modell 30 modifisert etter 12 epoker på Fashion MNIST.

C. Sammenligning av modellering på MNIST og Fashion MNIST

Figur 10 viser nøyaktigheten for valideringsdataene for alle de tre modellene (logisitisk regresjon, FFNN Modell 5 og CNN Modell 30 modifisert) som funksjon av epoker i de endelige 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 piksler gjør en bedre jobb enn generelle FFNN. Dette tyder på at det er gevinst i å evaluere piksler i nærheten av hverandre i sammenheng, slik filtrene i et CNN gjør.

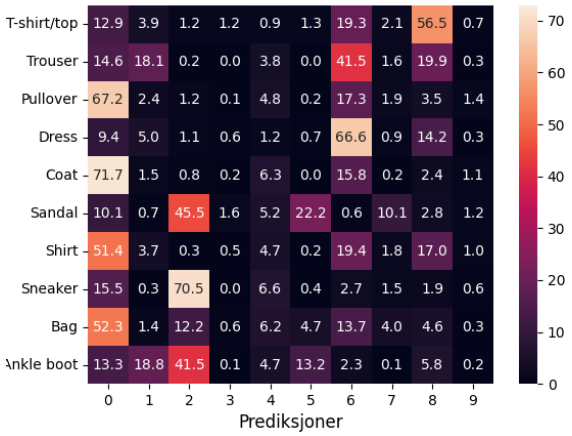


Figur 10 Sammenligning av nøyaktighet for valideringsdataene for de tre endelige modellene (logistisk regresjon, FFNN Modell 5 og CNN Modell 30 modifisert) for MNIST og Fashion MNIST.

Bare for gøy matet jeg Fashion MNIST datasettet inn i modellen som var trent på MNIST. Feilmatrisen for prediksjonene er vist i figur 11. Her ser vi at sko (sandaler, sneakers og ankelboots) og vesker blir klassifisert som tallet 2, mens de fleste andre klær blir klassifisert som enten 0, 6 eller 8. Dette er ikke 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 i det minste).

Men, dette illustrerer tydelig at selv om koden benyttet for å lage modeller for MNIST og Fashion MNIST er identisk, så blir modellene som trenes ulike fordi de er trent på ulike data. Og det viser viktigheten av at nevrale nettverk som er trent til å utføre den oppgaven de 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 viste en feilmelding istedenfor 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).



Figur 11 CNN Modell 30 modifisert, trent på MNIST benyttet til å gjøre prediksjoner på Fashion MNIST. NB!: Dette er ikke meningsfylt, men illustrerende for at modellen må brukes på samme problem som den testes på.

IX. 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 for klassifisering av bilder. De ulike metodene og arkitekturene har vært testet ut på datasettet MNIST (Lecun *et al.*, 1998). 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 filter 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).

Ved utprøving av de ulike arkitekturene ble det tydelig at konvolusjonsnettverk hadde en klar fordel i å gjenkjenne elementer i et bilde fremfor FFNN. Dette skyldes filterne i konvolusjonsnettverkene som evaluerer pikslene, ikke en og en, men i sammenheng med omkringliggende piksler, og dermed kan gjenkjenne strukturer i bildet.

Det kan være fristene å spekulere i hva som er årsakene til at noen tall feilpredikeres oftere og med preferanser til andre tall. For å finne ut det, må man undersøke filterne. Hvorvidt de strukturene som gjenkjennes av filterne sammenfaller med det vi mennesker legger vekt på når vi skal klassifisere tall, er uvisst.

De fleste modellene gjenkjente enklest tallene 0, 1 og 6, og hadde ofte problemer med særlig talle 9. Men vi så også at hvilke tall som var enklest å kjenne igjen endret seg når vi gikk fra ett til to konvolusjonslag. For den endelige modellen, som bare var kjørt over 12 epoker istedenfor 20 (som i utprøvingene) var det 0,4 og 7 som utpekte seg som de som ble best gjengkjent. Det kan derfor også virke som at det (selv om modellen ser ut til å være konvergert) skjer endringer i filterne underveis i optimeringen som påvirker klassifiseringen.

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.
Y. LeCun, C. Cortes, and C. J. Burges (n.d.), "The mnist database of handwritten digits," .
A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Anti-

ga, 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).

H. G. Rumelhart, D., and R. Williams (1986), "Learning representations by back-propagating errors," Nature **323** (6088), 533–536.

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].