Scikit-Learn Till Convolutional Neurala Nätverk: En Analys På Klassificeringsmodeller Inom Astronomi

Eric Torsson

EC Utbildning

Examensarbete

2024-05

Abstract

The objective of this thesis is to take a closer look at different ways to apply and use different classification models from Scikit-Learn and Tensorflow and compare those two methods against each other. We are going to use two different datasets, one which contain numerical data about different galactic phenomena, in this case stars, galaxies and quasars, and another dataset which contains images of galaxies and stars.

This is to see if it is worth the time investment of taking pictures and labeling them and training a Convolutional Neural Network to correctly classify the images or if the numeric data is enough for the problem at hand. The aim of this thesis is to first and foremost answer the question how the performance between the two methods differs, and then to see if it is possible to achieve 90% accuracy with both methods.

Innehållsförteckning

1 Inledning

Vare sig man är en entusiastisk astronomiforskare, en nyfiken observatör vid natthimlen eller någon som inte allt är intresserad av rymden, så är det inte en orimlig observation att rymden är både mystiskt och även nästa steg för mänskligheten.
Civilisationer som sträcker sig från forntiden till vår moderna civilisation har alltid haft någon form av fascination eller nyfikenhet om universum. Genom tiden så har astronomer kartlagt och observerat diverse himlakroppar och andra fenomen i universum, men det är fortfarande mycket som är okänt även för de trogna moderna astronomerna.
Även efter århundranden av utforskning och upptäckt så är vår förståelse om universum fortfarande väldigt begränsad. En sådan begränsning ligger i hur stor rymden är, avståndet mellan galaxer, planeter, stjärnor och andra himlakroppar kan sträcka sig över flera miljarder ljusår från varandra. Samtidigt som vår förståelse om andra komplexa astronomiska fenomen såsom svarta hål än idag ställer till hinder för moderna astronomer.

Det stora syftet med uppsatsen är ta en djupare titt på diverse klassificeringsmodeller genom att kolla närmare på två sätt att tillämpa klassificering på liknande problem. Är lönsamheten att ta bilder och märka bilderna för neurala nätverk någonting som verkligen behövs, eller räcker det med att förlita sig på numeriska data om stjärnor och galaxer för att skapa en stark och förlitlig modell motiverar en utforskning.

I denna uppsats så ska jag försöka svara på dessa frågor:

- **Hur jämförs prestandan hos traditionella maskininlärning klassificeringsmodeller med convolutional neurala nätverk (CNN) inom ramen för astronomi?**
- **Är det möjligt att uppnå målet på 90% pricksäkerhet från båda modellerna?**

2 Teori

I detta kapitel så går vi genom teorin bakom de olika modeller från Scikit-Learn som kommer testas, kort om vad neurala nätverk är och hur ett Convolutional Neural Network fungerar samt vilka utvärderingsmått som kommer användas.

2.1 Scikit-learn Klassificering Modeller

Scikit-learn har många olika modeller att välja mellan allt från klassificeringsmodeller till regressionsmodeller, klustringsmetoder och mycket mer, i denna del så kollar vi på några klassificeringsmodeller som kommer användas i detta arbete.

2.1.1 Random Forest

Random Forest är en *supervised* modell där de märkta datapunkterna är målet för modellen. Random Forests är en meta-estimator modell, det vill säga en ensemblemetod som passar data in i flera beslutsträdklassificerare på olika delar av datamängden och sedan använder sig av medelvärdet på de delarna för att förbättra den prediktiva noggrannheten samt kontrollera så modellen inte överanpassar på data.

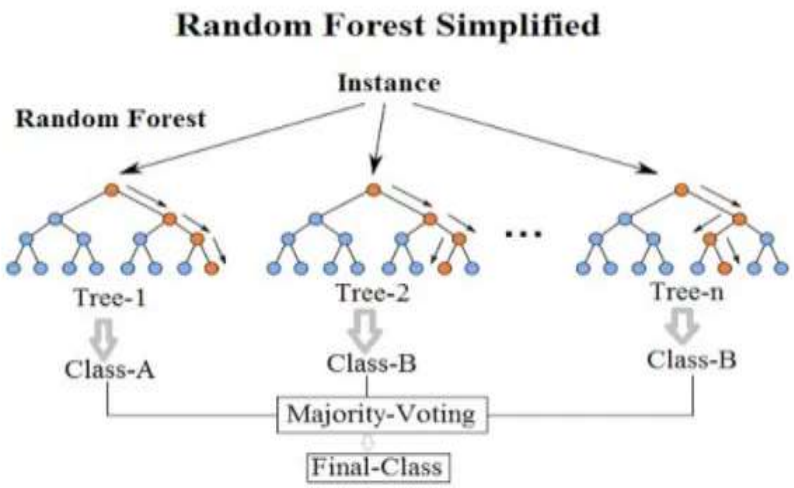
2.1.1.1 Regularisering i Random Forest

Regulariserings tekniker används i random forest för att kontrollera modell komplexitet och överanpassning. Att regularisera modeller har en stark koppling till valet av hyperparametrar, ett sätt är att begränsa antalet noder eller själva djupet av träden i modellen för att förhindra överanpassning och modell komplexitet.
Ett annat sätt kan vara att man optimerar parametern som styr antalet funktioner som slumpmässigt väljs för att bygga varje träd. Detta kan göras med hjälp av k-fold korsvalidering.

2.1.1.2 Välja Hyperparameter i Random Forest

Det finns ett flertal sätt att välja hyperparametrar till en Random Forest modell. En av dem är att använda sig av **Grid Search**, vilket innebär att man definierar ett rutnät med olika hyperparametrar och sedan söker genom hela rutnätet efter den bästa kombinationen av hyperparametrar.
En annan teknik är att använda sig an **Random Search**, där man slumpmässigt väljer olika hyperparameter kombinationer sen redan är fördefinierade. **Random Search** tillskillnad från **Grid Search** testar inte alla olika kombinationer. Olika hyperparameterar man kan testa mellan inkluderar men inte begränsat till:

- **n_estimators –** Bestämmer hur många träd som ska finnas i modellen.
- **max_depth –** Bestämmer djupet varje träd ska vara.
- **min_samples_split –** Sätter minimum på hur många datapunkter krävs för att splittra en nod i ett träd.



Figur 1: Illustrerar hur en Random Forest modell predikterar klasser. (Koehrsen, William. Random Forest Simple Explanation. Medium, 2017)

2.1.2 Support Vector Machines (SVM)

SVM är en algoritm där målet är att hitta ett hyperplan i N-dimensioner där N är antalet variabler eller ”features” som används för att klassificera datapunkter. Hyperplanet är en gräns som separerar och hjälper klassa olika datapunkter, där punkter som hamnar på olika sidor av hyperplanet kan hänföras som olika klasser. Det finns många möjliga hyperplan att välja mellan, så målet är att hitta det planet som har maximummarginal mellan datapunkterna för de olika klasserna.

2.1.2.1 Kernel Trick i Support Vector Machines

Kernel Trick är ett grundkoncept i SVM modeller som ger möjligheten till att hantera icke-linjära beslutsgränser effektivt genom att anpassa data i en högre dimension. Detta görs med hjälp av en kernel funktion där den inre produkten beräknas mellan data punkter i den högre dimensionen. Det finns många olika kernels man kan använda sig av beroende på slutmålet:

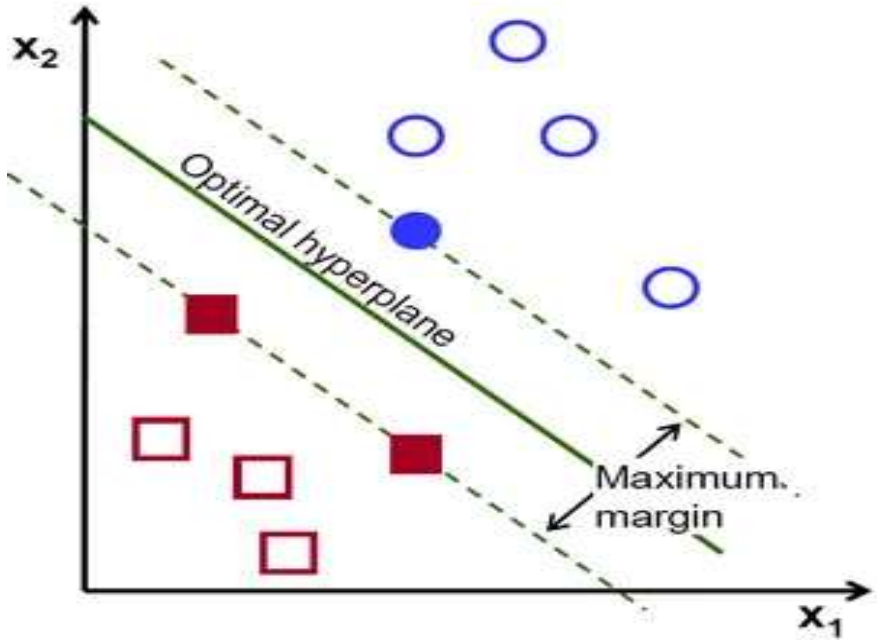
$$Linjär\ Kernel\colon\ K(x_i, x_j) = x_i^T \times x_i$$
$$Polynomisk\ Kernel\colon\ K(x_i, x_j) = \left(x_i^T \times x_j + x\right)^d$$
$$Sigmoid\ Kernel\colon\ K(x_i, x_j) = \tan\left(ax_i^T \times x_i + c\right)$$

Kernel Trick används först och främst för att hantera olinjära relationer mellan datapunkter, vilket betyder att data inte måste vara linjärt med varandra för att effektivt hitta ett hyperplan som passar. Eftersom SVM modeller jobbar i högre dimensioner så leder det till mer effektiva beräkningar även fast det inte görs en explicit transformering av data.

2.1.2.2 Optimering av Support Vector Machines

Optimering av SVM modeller görs först genom att välja den kernel som passar bäst för problemet och det dataset man har att jobba med. Sedan så väljer man de hyperparametrar man vill använda, här kan man även testa ett flertal olika hyperparameterar med hjälp av **Grid Search** eller **Random Search**. Exempel på hyperparametrar att välja mellan inkluderar men inte begränsat till:

- **C –** Kontrollerar avvägningen mellan att maximera marginal och minimera klassificerings error.
- gamma – Bestämmer hur mycket inflytande varje tränings exemplar har i en RBF kernel.



Figur 2: Visuell beskrivning hur ett hyperplan kan fördelas mellan två klasser. (Gandhi, Rohith. Support Vector Machine – Introduction to Machine Learning Algorithms. Towardsdatascience, 2018)

2.1.3 Logistic Regression

Logistic Regression är ett estimat av sannolikheten att en händelse inträffar, exempelvis om någon har ätit frukost eller inte. Det är lätt att missförstå och tro att Logistic Regression är en regressionsmodell, vilket det inte är, utan det är en statistisk modell som ofta används vid binär klassificering, men kan även tillämpas till flera klasser med hjälp av **One-vs-Rest (OvR)** eller **One-vs-All (OvA)** metoder.

2.1.3.1 Regularisering i Logistic Regression

Regularisering är en viktig del i många modeller, Logistic Regression är inget undantag. Målet är att undvika överanpassning och förbättra generalisering av modellen. Och mot det ändamålet så finns det två vanliga typer av regularisering.
Den första är **L2 Regularization (Ridge Regularization)** och den andra är **L1 Regularization (Lasso Regularization)**.
L1 regularisering använder en straffterm på kostfunktionen i modellen som är proportionell mot det absoluta värdet på beta (β) koefficienten. Medan L2 regularisering och använder sig av en straffterm på liknande sätt, men denna straffterm är proportionell mot roten ur magnituden av beta (β) koefficienten.

$$L1\ Regularisering\colon\ \lambda\sum_{j=1}^{n}|\beta_j|$$
$$L2\ Regularisering\colon\ \lambda\sum_{j=1}^{n}\beta_j^2$$

2.1.3.2 Funktionsskalning i Logistic Regression

Vanliga skalningsmetoder som används till många modeller är **Standardization (Z-score normalization)** och **Normalization (Min-Max scaling)**. Standardization är en statistisk metod som normaliserar datan genom att sätta medelvärdet till noll och standardavvikelsen till 1. Medan normalization skalar datan till ett värde mellan [0, 1].

2.1.4 K-Nearest Neighbor (KNN)

KNN är en *supervised* algoritm som används för både klassificerings och regressions problem. Denna algoritm antar att liknande saker existerar när intill varandra. Det som algoritmen försöker göra är att

sätta samma klasser i samma kluster, det vill säga att datapunkter som tillhör samma klass kommer därmed vara nära varandra i dessa kluster.

2.1.4.1 Avståndsmått i K-Nearest Neighbor

Att välja rätt avstånds mått är en viktig del när man använder KNN modeller. Det finns tre vanliga avståndsmått som används, Euclidean Distance, Manhattan Distance och Minkowski Distance.

1. Euclidean Distance
    - Vanligaste avståndsmåttet som anpassas
    - Beräknar en rak linje mellan två punkter i ett multidimensionellt utrymme

$$Euclidean\ Distance:\ d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

2. Manhattan Distance
    - Beräknar avståndet mellan två punkter genom att summera den absoluta skillnaden mellan punkternas koordinater
    - Hanterar extrema värden bättre än andra avståndsmått

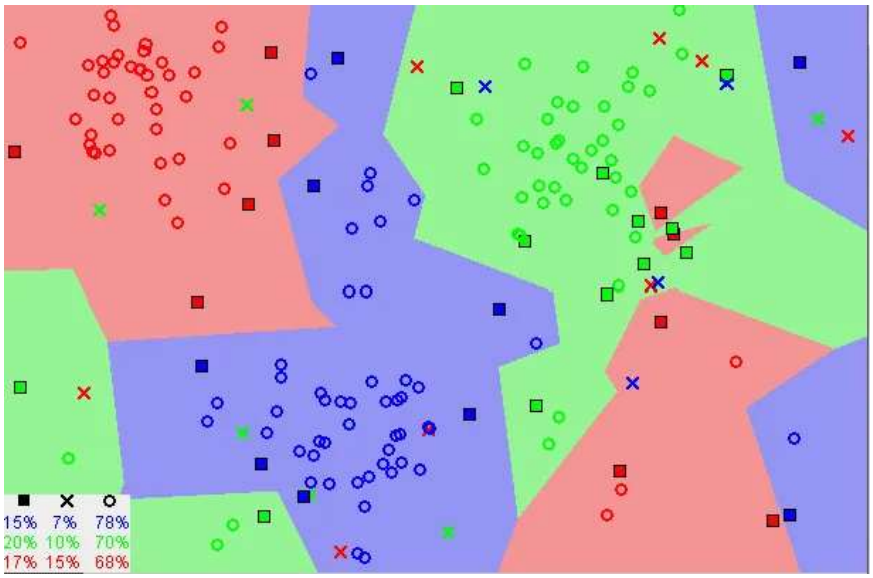
$$Manhattan\ Distance:\ \sum_{i=1}^{n}|p_i - q_i|$$

3. Minkowski Distance
    - Är en generalisering av benämnda avståndsmått

$$Minkowski\ Distance:\ D(X,Y) = \left(\sum_{i=1}^{n}|x_i - y_i|^p\right)^{\frac{1}{p}}$$

2.1.4.2 Att Välja K i K-Nearest Neighbor

Att välja K i KNN modeller så kan man använda sig av metoder som redan har nämnts, såsom **Grid Search** och **Random Search**. Men det är viktigt att ha i baktanke när man väljer K-värdet att när man gör en binär klassificering så är det vanligt att man väljer ett udda tal (1,3,5…) för att undvika oförutsägbart beteende från modellen. En annan regel man kan utgå ifrån är att använda det K-värde man får om man tar roten ur antalet datapunkter man har.



Figur 3: Exempel som illustrerar att oftast så befinner sig liknande datapunkter nära varandra. (Harrison, Onel. Machine Learning Basics with the K-Nearest Neighbor Algorithm. Towardsdatascience, 2018)

2.1.5 Decision Tree

Decision Tree är som det låter, det är en modell som liknar ett träd, lite som Random Forest, skillnaden är att denna modell fungerar lite som ett flödesschema där varje nod i trädet representerar ett test, och varje ”gren” i trädet representerar resultatet på testet medan varje ”löv” i grenen representerar klassen.

2.1.5.1 Pruning av Decision Trees

Pruning används för att undvika överanpassning genom att reducera allt ljud runt datan. Med hjälp av pruning så ger man trädet en mer simpel struktur i stället för en mer komplicerad struktur, samtidigt som man bevarar noggrannheten vid prediktion.
Det finns olika tekniker man kan anpassa, ett av dem är ”Pre-pruning (Early Stopping)”, där man slutar växa trädet innan den blir alltför komplex.

2.1.5.2 Hantering av Kategoriska Variabler

Decision Trees kräver numeriska variabler, det vill säga att man inte kan rakt av använda sig av exempelvis ”bil”, ”lastbil”, ”grön” i modellen. Dessa kallas för kategoriska variabler och kräver att man anpassar olika funktions-encodings vilket betyder att man mappar kategoriska variabler till numeriska variabler. Det finns flera sätt man kan göra detta på, exempelvis så kan man använda sig av One-Hot Encoding där varje distinkt kategorisk variabel görs om till binära variabler.
Ett vanligt exempel är om du har tre färger, röd, blå och grön och anpassar One-Hot Encoding så kommer de nu representeras med ettor och nollor:

$$\begin{aligned} Röd &= [1, \quad 0, \quad 0] \\ Blå &= [0, \quad 1, \quad 0] \\ Grön &= [0, \quad 0, \quad 1] \end{aligned}$$

I exemplet så ser vi tre unika färg klasser och tre unika matriser som visar vilken färg det är numerisk. Andra vanliga metoder som används är Ordinal Encoding och Frequency Encoding, oftast så beror det på vad för dataset man har att jobba med när man bestämmer vilken metod man kommer använda.

2.2 Neurala Nätverk

2.2.1 Grunderna i Neurala Nätverk

Neurala nätverk är baserade på hur den mänskliga hjärnan är uppbyggt och fungerar. Det vill säga att det finna olika lager där varje lager har en viss mängd neuroner (nodes) som är sammankopplade med neuronerna i nästa lager.

2.2.2 Convolutional Neural Network (CNN)

CNN tillskillnad från vissa andra neurala nätverk, använder sig av tre-dimensionella data vid bildklassificering och andra objekt igenkänning. Denna form av nätverk är en typ av ”feed-forward” neuralt nätverk, vilket betyder att information i nätverket går bara åt ett håll, framåt från noderna. CNN modeller innehåller även lager som andra neurala nätverk oftast inte har, dessa är pooling lager, convolutional lager, stride och padding vilket kommer fördjupas på i senare avsnitt. Men det är inte alltid som dessa lager används, mycket beror som sagt på andra faktorer som modell komplexitet och hur datan ser ut och hur mycket man faktiskt har att arbeta med.

2.2.2.1 Arkitektur av Convolutional Neural Networks

Arkitekturen i CNN modeller kan variera väldigt mycket, strukturen på modellen beror mycket på slutmålet, hur komplext problemet i fråga är, hur mycket data man har, och beräknings resurserna på datorn. I arkitekturen så ingår hur många lager som ska ingås, hur många noder varje lager ska få, och hur sammankopplade noderna ska vara mellan varandra.

- **Input lager:** Tar emot data
- **Gömda lager:** Alla lager som ligger mellan input och outputlagret där alla beräkningar görs på datan
- **Output lager:** Sista lagret som producerar det slutgiltiga resultatet av nätverket

Varje lager innehåller en mängd noder, dessa försöker replikera en biologisk hjärna, där varje nod får en inputsignal och gör en beräkning på den signalen och sedan producerar en outputsignal.

2.2.2.1.1 Convolutional Layers

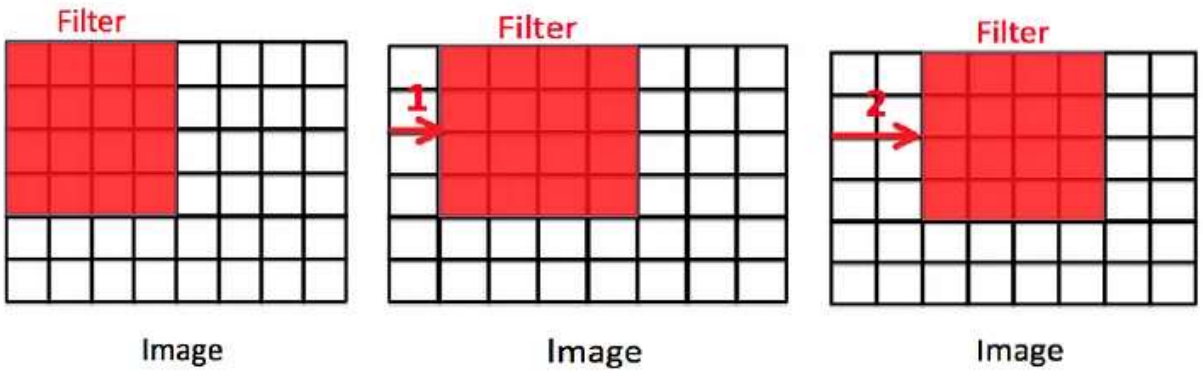
Convolutional lager är byggblocken till CNN modeller. Det är dessa lager som för de stora beräkningarna framåt i nätverket. I varje lager så görs det en dot-produkt mellan två matriser, där den ena matrisen har lärbara parametrar detta kallas oftast för kernel, medan den andra matrisen är en begränsad del av vad modellen ser.

2.2.2.1.2 Stride and Padding

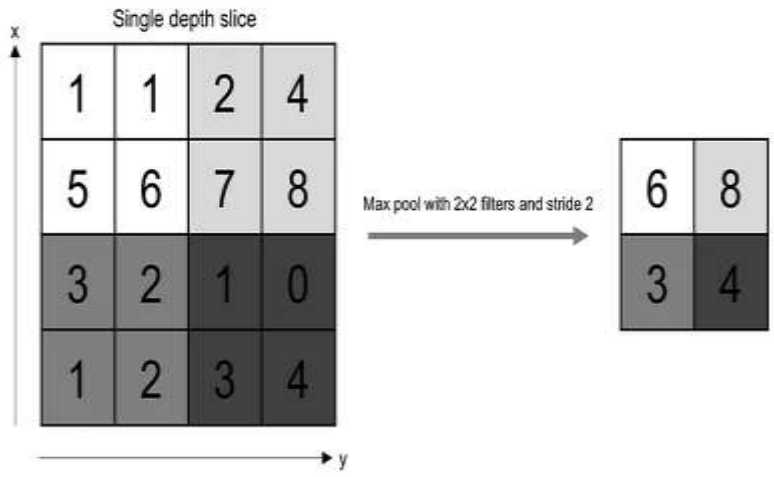
Padding då lägger man till en extra rad av pixlar till en bild innan man gör några beräkningar på den, detta hjälper med att bevara kanterna av bilderna och leder till att bilderna inte krymper för mycket. Stride däremot visar hur många pixlar ett filter rör sig vid varje beräkning, genom att ändra värdet på stride så kan man bestämma vilken skala output bilden kommer att ha.

2.2.2.1.3 Pooling Layers

Pooling lager ersätter en output i nätverket vid vissa delar. Detta görs för att sedan ta fram sumerisk statistik från andra outputs i närheten. Med hjälp av ett pooling lager så hjälper man i att reducera storleken av bilder vilket leder till mindre krav av hårdvaran i nätverket.



Figur 4: Visar hur stride rör sig över en matris. (Pandey, Kumar, Abhishek. Convolution, Padding, Stride, and Pooling in CNN. Medium, 2020)



Figur 5: Illustrerar hur pooling fungerar med en 2x2 matris som filter. (Mishra, Mayank. Convolutional Neural Networks. Towardsdatascience, 2020)

2.2.2.1.4 Dense Layers

Dessa lager hänvisas oftast till helt anslutna lager. Dense lager är ännu en viktig del i många neurala nätverk. Varje nod i detta lager är ansluten till noderna i lagret framför samt bakom, som sedan ger en output från varje nod som beräknas med hjälp av en vägd summa av inputs.

2.2.2.1.5 Batch Normalization

Användning av batch normalization görs för att förbättra stabiliteten av nätverket och öka hastigheten av inlärningsprocessen genom att normalisera aktiveringarna i varje lager.

2.2.2.1.6 Connected Layers

I helt anslutna lager så är noderna anslutna till både noderna som befinner sig i lagret framför såväl som i lagret bakom.

2.2.2.1.7 Aktiverings Funktion

Det finns många aktiverings funktioner man kan använda sig av till neurala nätverk, en aktiverings funktion tillämpas oftast på varje lager i nätverket. Alla har olika ändamål, och man måste inte alltid ha samma aktiverings funktion på varje lager men det är rekommenderat att försöka hålla sig till samma funktion för hela nätverket. Exempel på aktiverings funktioner är ”Sigmoid”, ”Tanh” och ”ReLU”.
ReLU är den vanligaste funktionen som används, den är väldigt tillförlitlig och hjälper till med att accelerera konvergens i nätverket väsentligt.

2.3 Utvärderingsmått

När man anpassar maskininlärning så finns det ett stort urval av utvärderingsmått att välja mellan. Mycket är helt beroende på vad det är för problem man försöker lösa och vad för modell man tänker använda.
I detta fall så anpassas *supervised* klassificeringsmodeller, vilket innebär att utvärderingsmåtten inte är detsamma som exempelvis regressionsmodeller. Måtten som används är accuracy, precision, återkallelse, F1-score, confusion matrix och ROC-AUC kurvan.
När det handlar om klassificerings problem så finns det vissa problem som man måste vara extra försiktig med när man tillämpar dessa utvärderingsmått. Exempelvis om det dataset man använder är ojämnt fördelade så kan en hög accuracy vara vilseledande då modellen kan prediktera att allt är majoritetsklassen – vilket leder till en hög accuracy.
Inom sjukhussektorn så kan fel leda till dyra konsekvenser om modellen predikterar fel, exempelvis

så kan man få falska positiva och falska negativa prediktioner från modellen vilket kan göra att någon som behöver hjälp inte får hjälp, medan någon som inte behöver hjälp får hjälp. Precision och återkallelse ger en bra bild på hur väl modellen hanterar dessa typer av fel, medan accuracy inte gör det.

2.3.1.1 Accuracy (Noggrannhet)

Accuracy är en av de mer vanliga utvärderingsmåtten som används vid klassificering. Accuracy motsvarar den andelen rätt predikterade klasser i ett urval.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

2.3.1.2 Precision

Precision kvantifierar noggrannheten av modellens korrekta prediktioner för en specifik klass. Detta görs genom att ta antalet positiva prediktioner genom positiva och falsk-positiva prediktioner.

$$Precision = \frac{TP}{TP + FP}$$

2.3.1.3 Recall (Återkallelse)

Återkallelse är den andelen av positiva klasser som predikterades korrekt bland alla instanser av en viss specifik klass.

$$Recall = \frac{TP}{TP + FN}$$

2.3.1.4 F1-score

F1-score är det harmoniska medelvärdet mellan återkallelse och precision, där ett högre värde motsvarar en bättre modell. F1-score ger ett singulärt värde för att se prestandan på modellen.

$$F1 = \frac{2 \times TP}{2 \times TP + FP + FN} \qquad F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

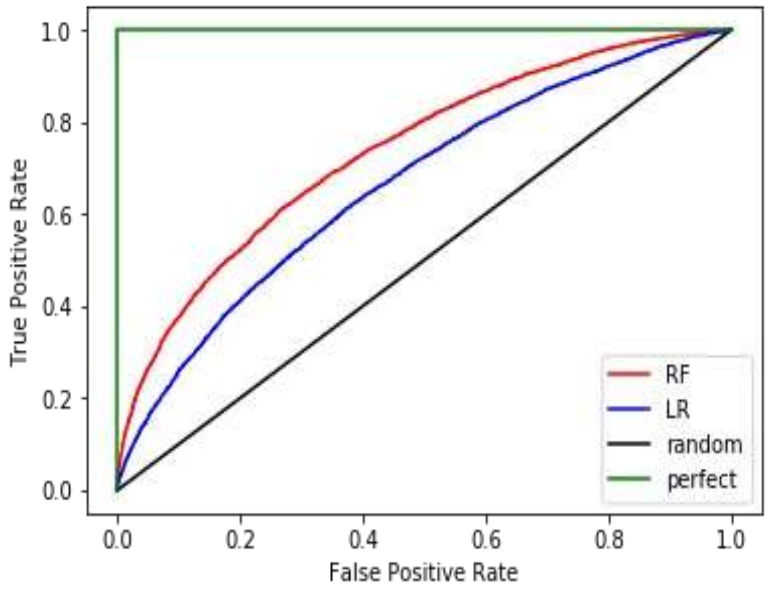
2.3.1.5 Confusion Matrix

Confusion Matrix används för att visuellt illustrera hur en modell predikterade. Det man kan bedöma från en sådan matris är precision och återkallelse, genom att visa summan av riktig positiv, falsk positiv, falsk negativ, riktig negativ

	Predicted Positive	**Predicted Negative**
Actual Positive	TP	FN
Actual Negative	FP	TN

2.3.1.6 Receiver Operating Characteristic – Area Under the Curve (ROC-AUC)

ROC-AUC är ett annat prestandamått för diverse klassificerings problem som fungerar på ett flertal modellinställningar. ROC är en sannolikhetskurva medan AUC representerar graden eller mått på separerbarhet. Genom att använda sig av ROC-AUC kurva så kan man se hur bra modellen är på att särskilja klasser. Desto högre AUC, desto bättre predikterar modellen korrekt på olika klasser.



Figur 6: Exempel på hur en ROC-AUC kurva ser ut. (Long, Andrew. Understanding Data Science Classification Metrics in Scikit-Learn in Python. Medium, 2018)

3 Metod

Detta kapitel kollar vi lite närmare på verktygen och datauppsättningen som användes till projektet.

3.1 Verktyg

Genom hela projektet så användes programmeringsspråket Python med hjälp av ramverk som Scikit-Learn, Tensorflow och Keras i en Jupyter Notebook miljö i VSCode. Tensorflow och Keras används i samband med varandra, där Tensorflow används som backend för effektiva beräkningar och optimeringar, medan Keras arbetar mer på en hög nivå för neurala nätverks API genom att göra processen från skapandet till distribueringen av modellen simplare.
Scikit-Learn på andra sidan är ett simpelt och effektivt ramverk fokuserat på maskininlärning för diverse *supervised* och *unsupervised* algoritmer.

3.2 Datainsamling

I denna uppsats så används två olika dataset för att besvara på frågorna. Det ena innehåller endast numeriska data om olika galaxer, stjärnor och kvasar. Medan det andra innehåller faktiska bilder på olika galaxer och stjärnor, däremot inga bilder på kvasar.
Båda datauppsättningar är tagna ifrån Kaggle från två olika källor. Den numeriska datauppsättningen ”Stellar Classification Dataset - SDSS17” och datauppsättningen med bilder ”Star-Galaxy Classification Data” innehåller riktiga data om diverse himlakroppar och båda är byggda för klassificeringsproblem.

1. **Numerisk Data om Galaxer, Stjärnor och kvasar (Stellar Classification Dataset - SDSS17)**
   Som nämnt tidigare så är datauppsättningen tagen från Kaggle av Fedesoriano (2022). Detta dataset innehåller 100 000 observationer i rymden som är tagna av SDSS (Sloan Digital Sky Survey), och är framtaget för att försöka klassificera stjärnor baserat på deras egenskaper. Datauppsättningen innehåller 17 unika funktioner och 1 kolumn dedikerad till klasser.

2. **Bilder på Galaxer och Stjärnor (Star-Galaxy Classification Data)**

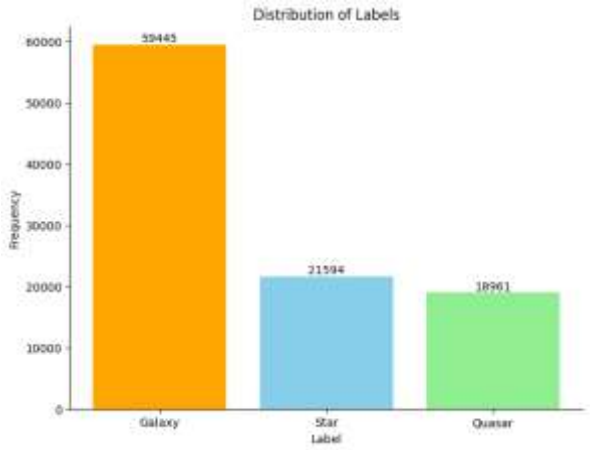
Detta dataset är också från Kaggle och var gjord av Divyansh Agrawal (2021) för ett projekt vid Aryabhatta Research Institute of Observational Sciences (ARIES), Nainital, India. I denna datauppsättning så finns det två klasser av bilder, den ena innnehåller 942 bilder på galaxer och den andra innehåller 3 044 bilder på stjärnor.

3.3 Utforskning av Data

Eftersom i denna uppsats så användes två olika datauppsättningar så behövdes båda utforskats. Datauppsättningen med numeriska data innehöll 100 000 unika observationer om olika rymdobjekt, medan datauppsättningen med bilder innehöll 3 986 unika bilder på stjärnor och galaxer.
Det är viktigt att uppmärksamma här den stora skillnaden i data som bägge datauppsättningarna består av.

3.3.1 Stellar Classification Dataset - SDSS17 (numerisk)

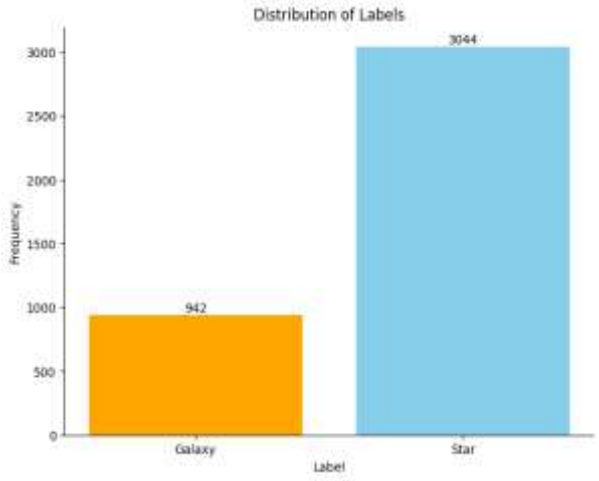
När man tar en närmare titt på distributionen av klasser i den numeriska datauppsättningen så ser vi att det finns tre unika klasser varav galax-klassen har en stor majoritet, vilket betyder att SMOTE eventuellt behöver anpassas. SMOTE är en översampling teknik, vilket betyder att man skapar nya datapunkter från minoritetsklasserna med hjälp av den data som finns om minoritetsklasserna.



Figur 7: Klass skillnad i "Stellar Classification Dataset - SDSS17" datauppsättning

3.3.2 Star-Galaxy Classification Data (bilder)

Datauppsättningen med bilder innehåller endast två klasser, en klass med stjärnor och en klass med galaxer. Här är distributionen också ojämn vilket inte alltid är så bra vid klassificering. Däremot så kommer bilderna gå genom ett neuralt nätverk, och neurala nätverk är lättare att ställa in hyperparametrar att ha denna klassobalans i åtanke vid träning.



Figur 8: Klass skillnad i ”Star-Galaxy Classification Data” datauppsättning

3.4 Data Förberedelse

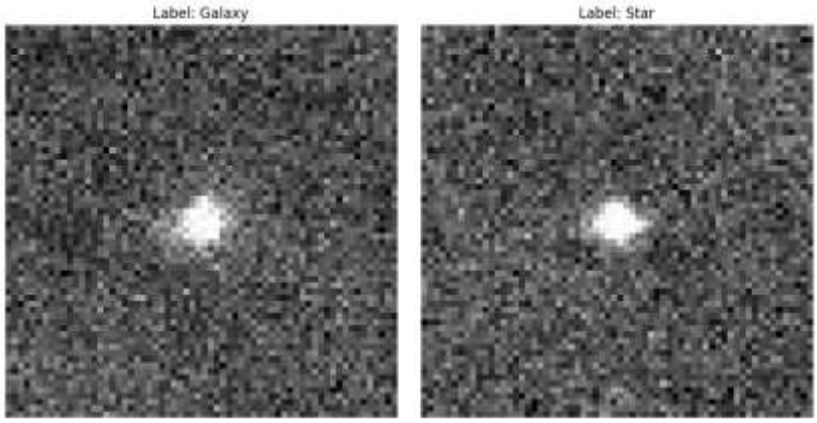
Med tanke på att datauppsättningarna som används har helt olika egenskaper jämfört med varandra så betyder det även att förberedelsen innan modellering är olika för båda.

3.4.1 Stellar Classification Dataset - SDSS17

Det första som gjordes på den numeriska datauppsättningen var är använda LabelEncoder från Scikit-Learn på ”class” kolumnen, det vill säga märka de datapunkterna i den kolumnen med antingen 2, 1 eller 0 beroende på vilken klass de tillhör. Detta är för många algoritmer hanterar numeriska värden bättre än kategoriska värden.
Sedan så fanns det många kolumner som inte bidrog med mycket mot slutmålet, vilket ledde till att dessa kolumner togs bort, majoriteten av kolumnerna som togs bort var ID kolumner.
Därefter så var det dags att ta hand om klassobalansen i datauppsättningen, och detta gjordes med hjälp av SMOTE från imblearn ramverket, vilket ledde till en jämn balans mellan klasserna.

3.4.2 Star-Galaxy Classification Data

Samma bemärkning gjordes på denna datauppsättning, men i stället för att använda LabelEncoder så märktes bilderna direkt i en egen variabel när datan laddades in.
Bilderna var alla redan i samma storlek på 64x64 pixlar, så nästa steg var att se till att alla bilder var i gråskala med endast en färgkanal i stället för tre.
Sedan så användes en funktion som heter ”to_categorical” vilket helt enkelt är en funktion i Tensorflow som fungerar som One-Hot Encoding från Scikit-Learn. Därefter så normaliserades pixlarna i bilderna så alla är på liknande skala vilket leder till en stabilare och snabbare träningsprocess.



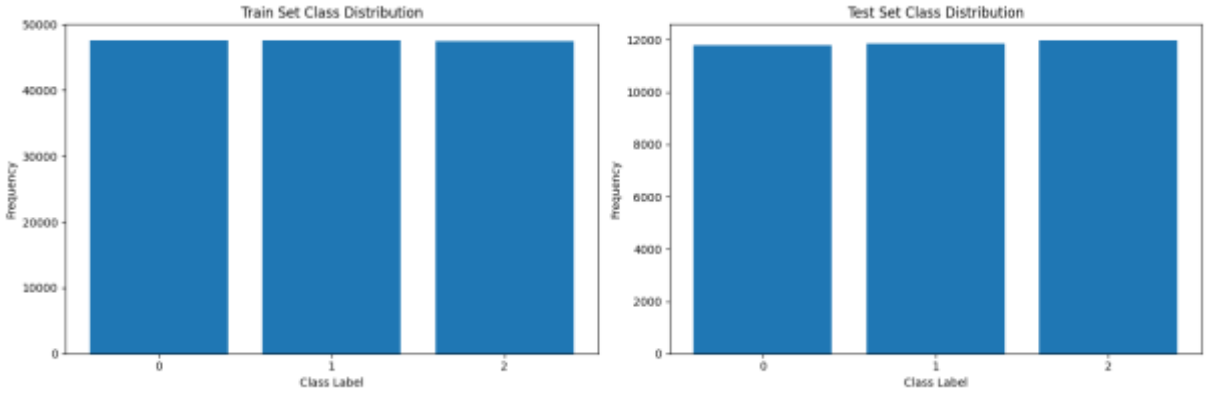
Figur 9: Figuren visar två bilder, en bild från varje klass i ”Star-Galaxy Classification Data” datauppsättningen

3.5 Träning och Test

Att dela data i en träning och ett test set är grundläggande när man vill testa sig på maskininlärning eller djupinlärning. I detta projekt så är detta inte ett undantag. Datauppsättningen med endast numeriska data delades upp i test och tränings data, medan datauppsättningen med bilder fick även validerings data.

3.5.1 Stellar Classification Dataset - SDSS17

Båda datauppsättningarna har en jämn distribution av klasser att träna sig på samt sedan testa sig på. Test datan består av 20% av hela datauppsättningen, medan tränings datan består av de resterande 80%.
Detta är en vanlig uppdelning som oftast används vid maskininlärning och ger modellerna rum att lära sig samt rum för att testa sig på.

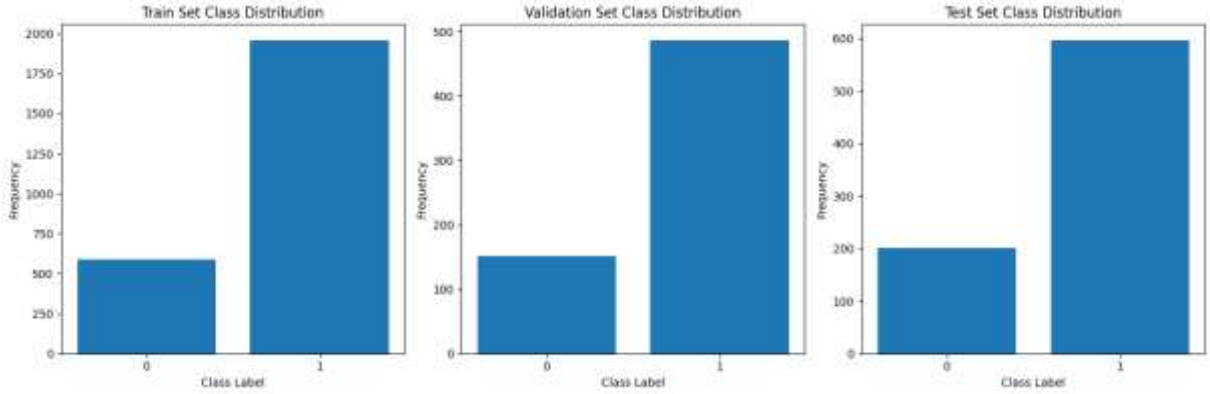


Figur 10: Bilden visar hur många av varje label (klass) som befinner sig i både test och träning datauppsättningen

3.5.2 Star-Galaxy Classification Data

Datauppsättningen med bilder har en ojämn klassdistribution jämfört med den numeriska datauppsättningen, och samtidigt färre datapunkter att lära sig ifrån. Men här finns det även ett validerings set som det neurala nätverket kommer testa sig mot under tiden den tränar på tränings setet.

Själva distributionen mellan klasserna i varje set är ungefär likadana, där 20% av klasserna tillhör klass 0 och resterande till klass 1. Testdatan består av 20% av träningsdatan och valideringsdatan består av 20% av resterande datan i träningsdatan.



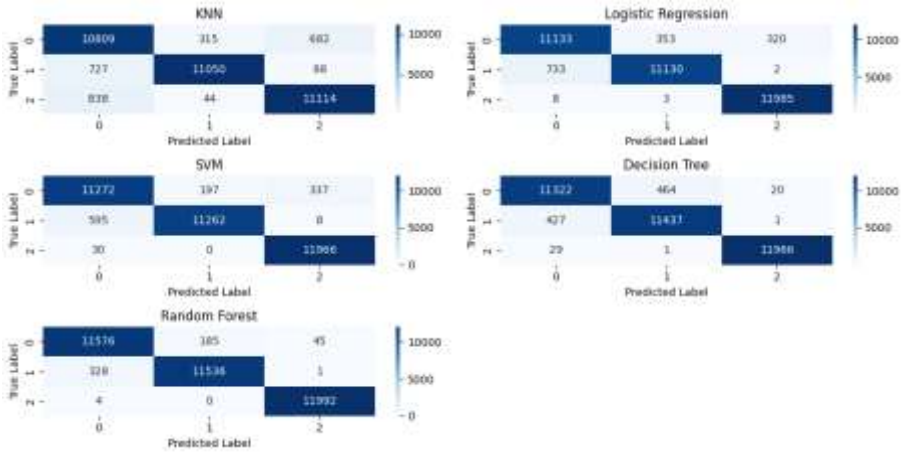
Figur 11: Visar klass skillnaden i träning, validering och testdatauppsättningarna

4 Resultat och Diskussion

I detta kapitel kommer jag presentera resultaten från modellerna, från båda datauppsättningarna, och visa de modeller som testades och slutligen den modellen som användes i slutet. Detta är eftersom jag bestämde mig för att testa fem olika modeller till den numeriska datan för att se vilken som skulle prestera bäst.
Varje delrubrik innehåller en figur för att visualisera modellernas prestanda och bättre kunna jämföra dem med varandra.

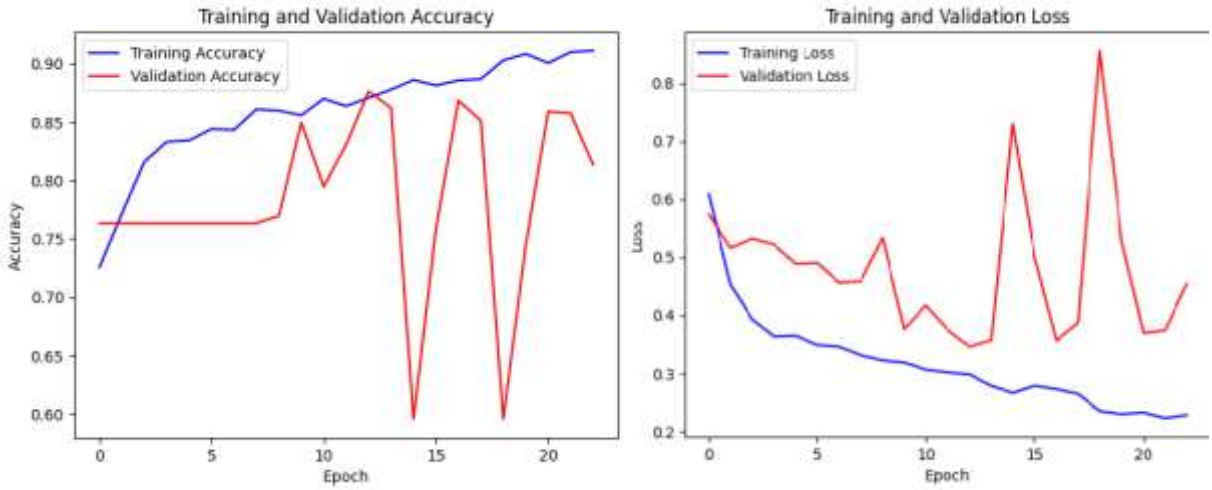
4.1 Modellutvärdering

I början så testades fem olika modeller som skulle användas med den numeriska datan för att sedan välja de tre bästa modellerna för vidare utforskning. I detta steg så standardiserades datan och sedan tränades modellerna för att hitta de bästa modellerna utan större hyperparameter inställning.
Av dessa fem modeller så var alla väldigt lika (se figur 13), där den enda som stack ut på den sämre sidan var KNN.
Så utifrån detta så fortsatte Random Forest, Decision Tree och SVM modellerna vidare till nästa steg i processen då dessa visade de bästa resultaten.
När det sedan kom till CNN nätverket så ritades pricksäkerheten, validering och tränings förlusten upp för att utvärdera hur bra nätverket presterade. Eftersom datan var väldigt obalanserad så var det svårt att hitta rätt arkitektur för nätverket för att undvika överanpassning samt att få modellen att konvergera (se figur 14).



Figur 12: Matris som visar ursprungliga resultaten från de fem valda modellerna

Figur 13: Matris som visar de numeriska utvärderingsmåtten från modellerna



Figur 14: Figuren visar prestandan på CNN nätverket under träning

4.2 Modellförbättring

Efter att de tre bästa modellerna valdes ut så tränades dessa tre modeller om på nytt men denna gång så användes cross-validation för att testa olika hyperparametrar för att ta fram den absolut bästa modellen.

Det visade sig att efter att efter att testa olika hyperparametrar så hamnade Random Forest modellen på topp (se tabell 1). Den vann inte med mycket, men av SVM och Decision Tree modellerna så kom Random Forest ut överst.

CNN nätverket gjordes ingen större förbättring eller förändring i arkitekturen, då större eller mindre nätverk gjorde varken någon större skillnad i slutresultatet och i vissa fall till och med försämrade modellen.

Tre bästa modeller efter GridSearchCV		
Modell	Best Params	Best Score
SVM	{'C': 1}	0,964996
Decision Tree	{'max_depth': 10, 'min_samples_split': 2}	0,976841
Random Forest	{'max_depth': None, 'n_estimators': 50}	0,982133

Tabell 1: Topp tre modellerna efter test på olika hyperparametrar

4.2.1 Slutgiltiga Modell

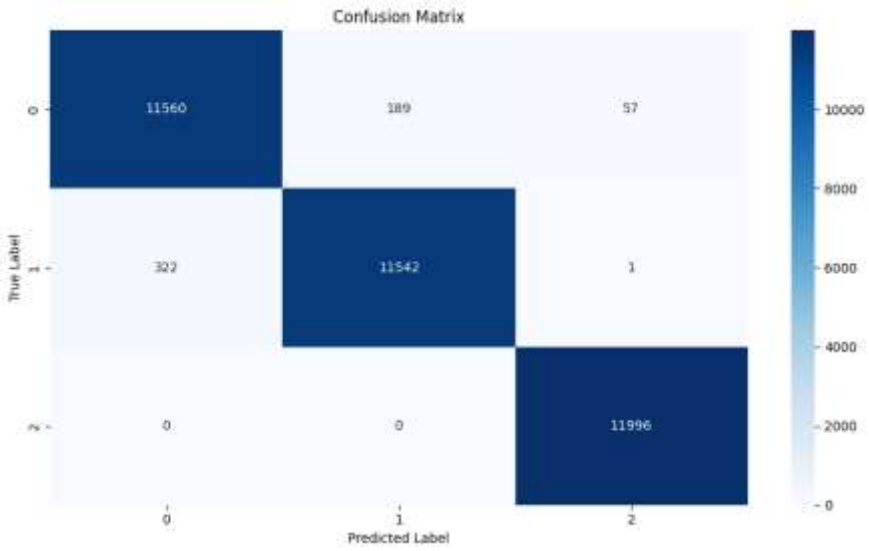
Efter att Random Forest visade sig vara den bästa modellen för problem till hand så tränades den om på nytt ännu en gång men denna gång så användes cross-validation för att testa ett större utbud av hyperparametrar för att leta efter den perfekta modellen.

När det var klart så hade modellen en 98% pricksäkerhet på testdatan med likadana resultat i precision, återkallelse och F1-Score (se figur 16). I confusion matrisen så ser man också att modellen presterade extremt bra när den predikterade klass 2 och 1, men hade lite mindre problem med klass 0 (se figur 15).

När CNN nätverket visade ett tillfredsställande resultat i träningen så testades modellen på testdata där pricksäkerheten kom upp till 80%. När man kollar närmare på prestandan av modellen så ser det inte lika bra ut, både figur 17 och 18 så ser man att modellen har problem med att prediktera rätt när bilden tillhör klass 0.

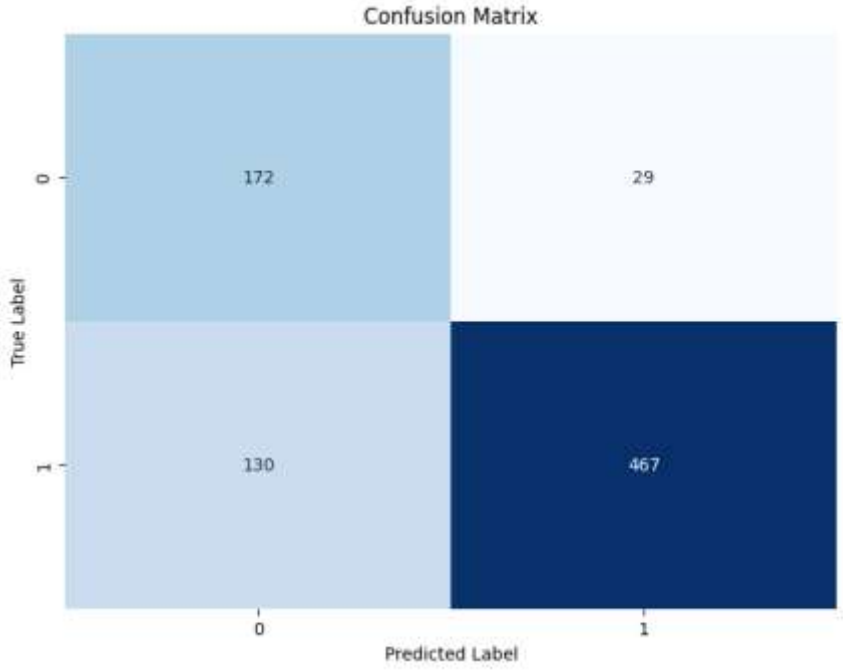
Pricksäkerhet slutgiltiga modeller	
Random Forest	98,4%
Neural Network (CNN)	80,0%

Tabell 2: Pricksäkerheten i %

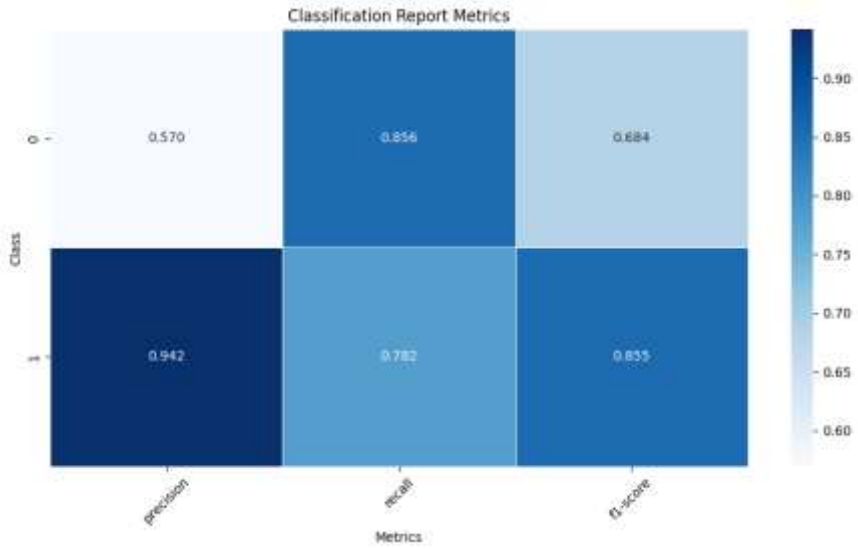


Figur 15: Figuren visar hur den slutgiltiga Random Forest modellen presterade

Figur 16: Numerisk utvärderingsmått på den slutgiltiga Random Forest modellen



Figur 67: Matrisen visar hur CNN nätverket presterade på testdata



Figur 18: Figuren visar numeriskt utvärderingsmått på CNN nätverket

5 Slutsatser och Förbättringar

I detta avsnitt kommer jag presentera och diskutera lite runt frågorna som ställdes i början av arbetet. Kommer diskutera valet av modeller, framtida förbättringar och saker som går att göra annorlunda.

Frågorna som ska besvaras:

- **Hur jämförs prestandan hos traditionella maskininlärning klassificeringsmodeller med convolutional neurala nätverk (CNN) inom ramen för astronomi?**
- **Är det möjligt att uppnå målet på 90% pricksäkerhet från båda modellerna?**

5.1 Slutsats

Den första frågan som ställdes: ”Hur jämförs prestandan hos traditionella maskininlärning klassificeringsmodeller med convolutional neurala nätverk (CNN) inom ramen för astronomi?” har ett relativt klurigt svar, eftersom båda modellerna hade två olika datauppsättningar. Men om man kollar på pricksäkerheten av slutmodellerna så skiljer det sig endast med ungefär 18%, vilket inte alls är mycket med tanke på att CNN modellen hade inte alls lika mycket data att arbeta med jämfört med Random Forest modellen.
Även fast det är en giltig ursäkt vilket ger ett skevt resultat i det stora hela, så vinner ändå Random Forest modellen men en relativt stor marginal.
”Är det möjligt att uppnå målet på 90% pricksäkerhet från båda modellerna?” Är den andra frågan som skulle besvaras, och i detta fall så uppnådde endast Random Forest modellen det målet med en 98% pricksäkerhet, men CNN modellen var inte alls långt ifrån med en pricksäkerhet på 80%, jag kommer gå genom närmare in på detta i respektive modellavsnitt.

5.1.1 CNN Modell

När jag byggde CNN modellen så gick jag efter ett ”trial and error” beteende för att kortsiktigt spara tid och datorresurser. Detta har lett till att slutmodellen är säkerligen inte den bästa modellen för denna uppgift, och det reflekteras även i pricksäkerheten jämfört med Random Forest modellen där den fall bakom med hela 18%.
Modellen presterade däremot väldigt bra med tanke på bristen på data som den hade att jobba med, men den grundläggande arkitekturen går att förbättras på andra sätt vilket jag går genom i nästa avsnitt för att få en stabilare modell.

5.2 Framtida Förbättringar

Till ett avslut så kommer jag gå genom några eventuella förbättringar som man kan ha i åtanke för framtiden. De två saker jag kommer nämna är valet av data samt användningen av Keras Tuner till neurala nätverket.

5.2.1 Data

Största hindret för denna uppsats vara datan, i och med att båda metoderna som utforskades använda två helt olika datauppsättningar, vilket kan leda till att man missleder och visar skeva resultat från modellerna.
Nästa steg vore att hitta en datauppsättning som har både bilder och numeriska data om diverse himlakroppar för att ge en bättre och starkare helbild i hur modellerna skiljer sig från varandra.

5.2.2 Keras Tuner

Med hjälp av Keras Tuner så kan man optimera hyperparametrarna i CNN modellen. Man kan tänka att Keras Tuner fungerar lite som Scikit-Learn cross-validation när den letar efter hyperparametrar. Keras Tuner tillåter en att hitta hur många lager som är optimalt i nätverket, man kan även hitta det optimala värdet för dropout, learning rate och mycket mer.
Detta är något som jag inte använde vilket skulle med stor sannolikhet förbättra modellens prestanda relativt mycket och kanske till och med uppnått målet av 90% pricksäkerhet.

5.2.3 Modell Optimering

Som nämnt i förra delkapitlet så skulle användningen av Keras Tuner för att optimera hyperparametrarna i neurala nätverket förmodligen förbättra slutresultatet genom att hitta de mest optimala lagerna, noder, learning rate och så vidare.
Men något annat som skulle kunna utforskas är att testa andra modeller som redan är tränade och klara och modifiera och optimera den modellen efter den datauppsättning som användes här, exempelvis ResNet-50 exempelvis.
ResNet-50 är en typ av CNN med 49 helt anslutna lager och är lätt att modifiera en förutbildad modell som är tränad på ImageNet datauppsättning.

Appendix A

Utvärderingsmått fem första modeller				
Modell	**Accuracy**	**Precision**	**Recall**	**F1-Score**
KNN	0,924468	0,925755	0,924446	0,924808
Logistic Regression	0,960215	0,960146	0,960044	0,959954
SVM	0,967281	0,967417	0,967149	0,967125
Decision Tree	0,973589	0,973488	0,973477	0,973481
Random Forest	0,984215	0,984172	0,984152	0,984143

Tabell 3: Resultat efter test på de fem olika klassificeringsmodellerna med extra decimaler

Sista modellen				
Modell	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Random Forest	0,984047	0,983992	0,98398	0,98397

Tabell 1: Resultat efter hyperparameter stämning på den bästa modellen

CNN model resultat				
	Precision	**Recall**	**F1-Score**	**Support**
0	0,569536	0,855721	0,683897	201,000000
1	0,941532	0,782245	0,854529	597,000000
accuracy	0,800752	0,800752	0,800752	0,800752
Macro avg	0,755534	0,818983	0,769213	798,000000
Weighted avg	0,847834	0,800752	0,811550	798,000000

Tabell 2: Resultat från CNN nätverket efter träning

Källförteckning

Deepchecks. (u.å). Hämtad 2024–05 från
https://deepchecks.com/glossary/normalization-in-machine-learning/

Divyansh Agrawal. (2021). Star-Galaxy Classification Data. Hämtad 2024–04 från
https://www.kaggle.com/datasets/divyansh22/dummy-astronomy-data/data

fedesoriano. (January 2022). Stellar Classification Dataset - SDSS17. Hämtad 2024–04 från
https://www.kaggle.com/fedesoriano/stellar-classification-dataset-sdss17

IBM. (u.å). Hämtad 2024–05 från
https://www.ibm.com/topics/logistic-regression

IBM. (u.å). Hämtad 2024–05 från
https://www.ibm.com/topics/convolutional-neural-networks

Keras. (2024). Hämtad 2024–05 från
https://keras.io/

MachineLearningMastery. (2017). Hämtad 2024–04 från
https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

Scikit-Learn. (2024). Hämtad 2024–04 från
https://scikit-learn.org/stable/

Tensorflow. (2024). Hämtad 2024–04 från
https://www.tensorflow.org/

Wikipedia. (2024). Hämtad 2024–05 från
https://en.wikipedia.org/wiki/Euclidean_distance

Wikipedia. (2024). Hämtad 2024–05 från
https://simple.wikipedia.org/wiki/Manhattan_distance

Wikipedia. (2024) Hämtad 2024–05 från
https://en.wikipedia.org/wiki/Minkowski_distance

Wikipedia. (2024). Hämtad 2024–05 från
https://en.wikipedia.org/wiki/Convolutional_neural_network