GRP07 - ITMAL O4

Brug af Machine Learning Algoritme på eget datasæt

Gruppemedlemmer	Studienummer
Lars Bo Christiansen	201805398
Bjarke Vangsgård	201805703
Mathias Vraa	201810583

Kursuskoordinator: Carsten Eie Frigaard, Peter Ahrendt

Ingeniørhøjskolen Aarhus

Maj 2021

Indhold

Indledning og problemstilling

Målet med denne opgave er at udvikle en algoritme, der kan se forskel på billeder af forskellige frugter. Dette problem blev valgt, da image recognition er en sted, hvor Mashine Learning er meget effektivt. Samtidigt er der så stor forskel på frugter, at det burde være muligt at udvikle en model, der kan se forskel på dem.

På tabel 1 ses arbejdsfordelingen for O4. P står for primær og S for sekundær. Hvis der står flere P på et afsnit, så er arbejdet ligeligt fordelt.

Afsnit	Bjarke	Bo	Mathias
Problemstilling	P		
Datasæt	P	P	P
Analyse af dataset	P	P	P
Algoritme	P		
Pipeline	P		
Performance metrics			P
Optimering		P	
Over- og Underfitting	P	S	
Konklusion	P	P	P

Tabel 1: Oversigt over arbejdsfordeling. P(Primær), S(Sekundær)

Datasæt

Til O4 bruges datasættet Fruit 360, Version 2020.05.18.0[1]. Datasættet indeholder ca. 90000 .jpg filer af forskellige frugter, hvor hvert billed har størrelsen 100*100 pixels med værdier i rgb. Billederne er pre processeret, hvor der er blevet fjernet baggrund. Datasættet er også er delt op i et træningssæt og et testsæt.
Billederne er lablet og der er i alt 131 forskellige klassificeringer. Nogle frugter har flere forskellige tilhørende klasser, som for eksempel Apple Golden og Apple RED1.
Billederne er taget ved at tage en video af en langsomt roterende frugt med et videokamera. Baggrunden var et hvidt stykke papir. Men da der var forskel på belysning er billedet blevet beskåret så kun frugten er tilbage. Hvorefter de beskårede pixels sættes til 255.
Formålet med dataene er at opbygge en "multiclass image classifier" der kan kende forskel på de forskellige typer af frugter. Som nævnt før omfanger datasættet 131 forskellige klassifikationer, der er stor forskel mellem mange frugterne så det burde være muligt at få en effektiv classifier.

Datasættet har som udgangspunkt allerede en opdeling i Trænings og Test set, hvor billederne i Test sættet er uddrag af den førnævnte drejende frugt. Dvs. billederne i Test sættet ligner billederne i Træningssættet, ikke kun fordi det viser den samme frugt. Rotationen af frugten kan her nogle gange ikke være nok for at skabe en tydelig forskel i billederne. Dette kunne så føre til at den brugte ML Algoritme bliver trænet med billeder der er for nær på Test sæt billederne, et eksempel kan ses på figur 1 og figur 2.

Figur 1: Braeburn æble i Træningssæt

Figur 2: Braeburn æble i Testsæt

For at modvirke den potentielle overfitting pga. lignende samples i trænings og test sæt kan der overvejes flere mulige løsninger.
Den ene løsning bygger på at, datasættet supplerer ikke kun billeder af frugtene drejet rundt, men også samme antal af billeder af det samme æble drejet rundt i en roteret position. Dette betyder f.eks. for det ovenviste Braeburn æble, at den blev lægget på siden og så drejet rundt til sampling. Som et alternativt datasæt vil derfor være de normale billeder som træningsdata mens de roterede frugter repræsenterer testdataene.
En anden løsning til dette ville være at hente frugter fra et nyt datasæt og teste modellen på dette. Her ville der ikke være mulighed for at billederne ligner hinanden for meget, da der bruges helt nye frugter til det. Her kunne også baggrunden være noget andet for at teste selve objekt genkendelsen i algoritmen.

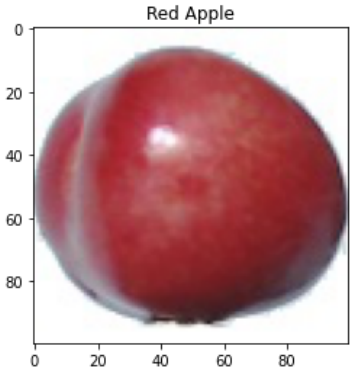
Analyse af dataset

For dataanalysen af datasættet der skal bruges til projektet blev der primært kigget på farver i et billede og deres position. Da der skal undersøges billeddata er der ikke meget at undersøge i de enkelte features, udover betragtningen af farverne. For koden til analysen kan der ses i Kode appendixen på listing 3.

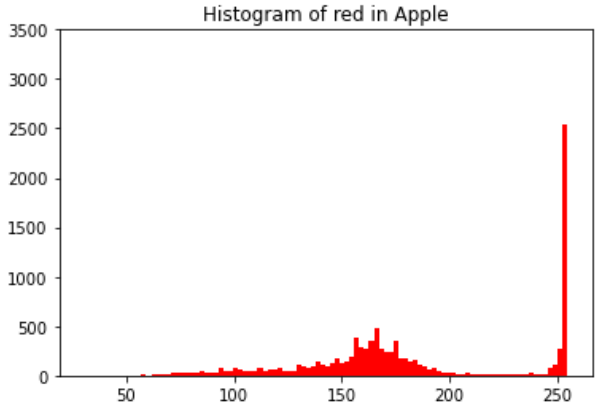
I det følgende vil farverne vises i histogrammer, der viser antallet af repræsentering af farven i billedet, og et grayscale af de enkelte farver vises, der viser repræsentering og positionen sammen. Der undersøges to forskellige billeder, som det første ses der på et rødt æble og bagefter på en grøn mango. Der indsættes ikke nogen kode til dette afsnit, da koden består af importeringen af dataene vha. sklearns load_files() funktionen der shuffler datasættet med det samme. For så at kunne udvælge et billede og indlæse det bruges der imread() funktionen fra skimage library. Herefter består koden af at slice datasættene og plotte de forskellige features og dele vha. pyplots imshow() funktionen eller hist() funktionen.

Figur 1a viser billedet af Red Apple der bliver undersøgt på farver i de følgende histogrammer, figur 1b, 1c og 1d. Der kan ses at der er en stor hvid rand rundt om frugten, og frugten er dominerende rød.

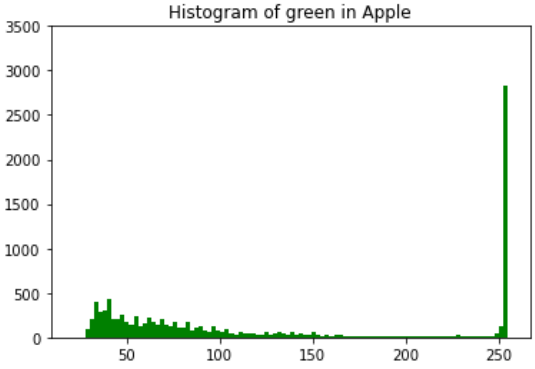
Histogrammerne kan ses til at have værdier mellem 0 og 255, der siger noget om intensiteten af farven, jo højere tallet desto mere af denne farve er der i en pixel. På alle tre histogrammer kan der ses en overvældende antal af pixels der indeholder max værdien, det er på grund af at i den hvide rand ligger der alle tre farver med maks intensitet.



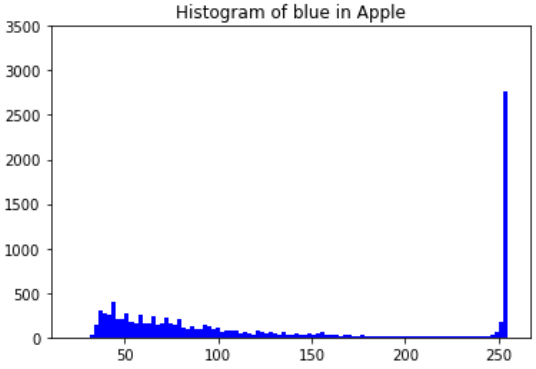
(a) Image Show af Red Apple der bliver undersøgt



(b) Histogram af den røde farve i Red Apple



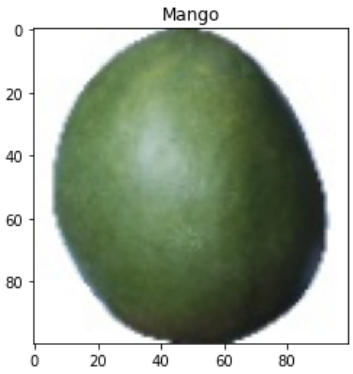
(c) Histogram af den grønne farve i Red Apple



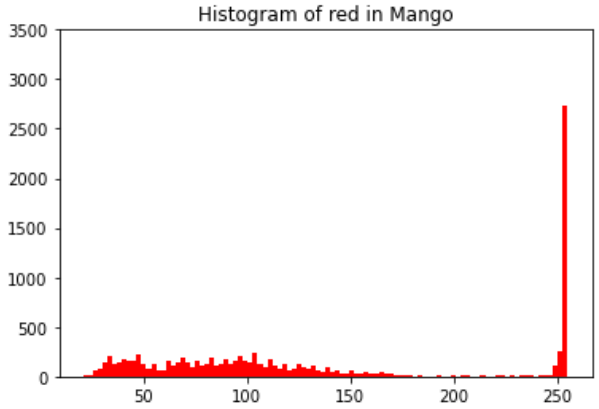
(d) Histogram af den blå farve i Red Apple

Figur 1: Analyse af æble billed

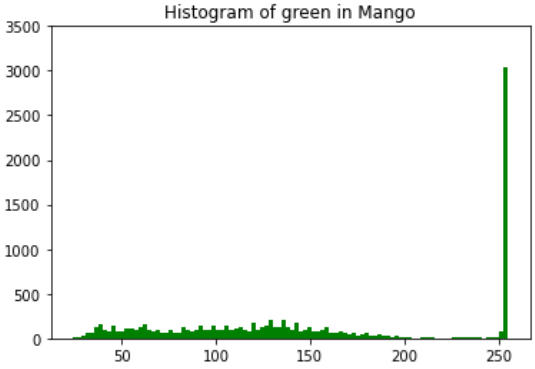
Herefter kan der ses at den røde farve har flere optrædelser ved intensitet 150-200, hvorved grøn og blå holder sig for det meste mellem 0-100. Det tyder så på at den røde farve er mere repræsenteret i selve billedet hvis man ser bort fra det hvide, hvilket stemmer god overens med forventningen fra figur 1a.
Der kigges nu på en mango i stedet for et æble. Der vælges at kigge på en mango, da den har en anden farve end det røde æble. Mangoen kan ses på figur 2a. Dette er mango billede 242_100.jpg.



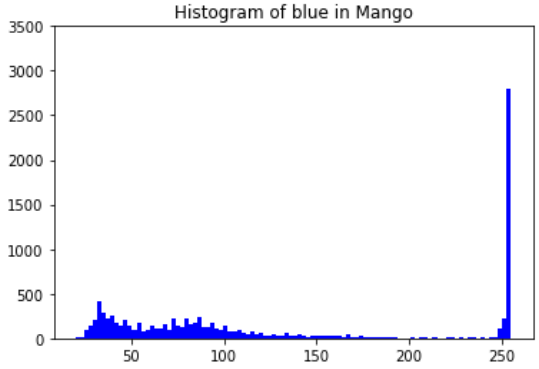
(a) Image Show af Mango der bliver undersøgt



(b) Histogram af den røde farve i Mango



(c) Histogram af den grønne farve i Mango



(d) Histogram af den blå farve i Mango

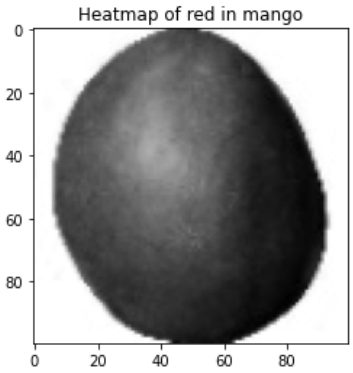
Figur 2: Analyse af mango billed

Der ses på figuren at mangoen har en del grønne pixels, samtidigt med at den er mere mørk i det end billedet af æblet. Dette betyder at der burde være en overvægt af grøn i billedet men hvor histogram værdierne vil ligge tættere på 0.

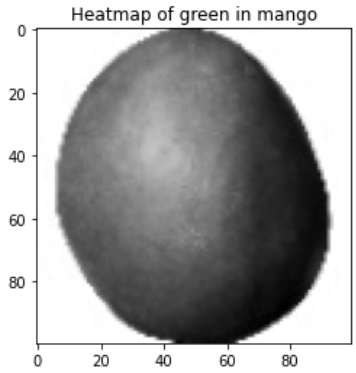
Der er ikke en bestemt værdi som histogrammet af grøn fordeler sig omkring, i stedet ses det at der den er mere ligeligt fordelt. Dog med en større koncentration af høje værdier end blå og grøn.

Histogrammerne for mango viser en større vægt af mørke pixels især blandt rød og blå.

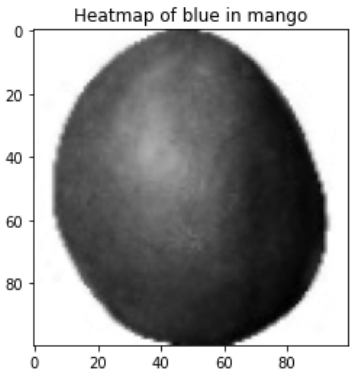
Hvis der så skal ses på heatmappet af de enkelte farver i mangoen blev der valgt at farverne vises som image show i grayscale. Her igen er en lyser pixel repræsentering for en højere intensitet af farven. Hvis man sammenligner heatmapperne i figurene 3a, 3b og 3c kan der meget hurtigt ses at heatmappen for den grønne farve er noget lyser en de andre to heatmapper, hvilket betyder at billedet er dominerende grøn. Da der kan ses på det originale Mango billede i figur 2a, kan der ses at observeringen er rigtigt.



(a) Heatmap af den røde farve i mango



(b) Heatmap af den grønne farve i mango



(c) Heatmap af den blå farve i mango

Figur 3: Heatmap af rød/grøn/blå farver i en mango

Algoritme

Da datasættet består af en masse forskelige frugter, er der behov for en multiclass classification. Datasættet er lablet og der er et kendt antal klasser. Dette gør det muligt at bruge supervised learning, men det ville også være muligt at bruge unsupervised clustering. Da supervised generalt giver en bedre accuracy, og datasættet ikke er alt for stort, bruges supervised.

Der er flere mulige klasser at vælge imellem, når det kommer til supervised klassificering. F.eks. Decision Trees, Support Vector Machines og Neural network modeller. Der vælges at kigge på SVM(Support Vector Modeller), da de er gode på mange dimentionelle datasæt. De virker også fint når der er flere features end samples, og da der ofte er mange features og få samples ved billed-genkendelse er det er godt valg. SVM er også effektiv, når det kommer til hukommelsesforbrug, dette er også vigtigt, når datasættet er stort.

Der vælges at bruge en SVC model. Da datasættet er i størrelsesordenen 10.000-100.000 samples er det værd at overveje fit-tiden. Dog har test kørsel vist at fit-tiden er indenfor en acceptable periode (Under 5 min efter PCA transformation).

Ulempen ved valget af en SVM er at det ikke er muligt direkte at få en sandsynligheds beregning på hver klasse, dette vil kræve yderligere arbejde. Hvis det ikke er muligt at reducere antallet af features i datasættet, ville valget af SVC modellen også give en meget lang fit-tid.

Pipeline

Der skal laves noget preprocessering af indput datasættet for at modellen kan beregnes. Dette skyldes især, at datasættet har mange features. Den overordnede pipeline ser ud som følgende:

1. Indlæsning træningdata
2. Indlæs test data
3. Lav data til matrix med dimensionerne 100*100*3
4. Flatten for at få to dimensionalt matrix til PCA
5. Brug PCA til at reducere antal features
6. Fit PCA med træningsdata og transformer data
7. Transformer testdata
8. Opret model og træn med træningsdata
9. Brug model på test data

Koden til punkt 1-4 kan ses i listing 5 og koden til 5-9 ses i listing 6

1 Indlæsning af data

Til indlæsning af data bruges load_files fra sklearn.datasets. Den indlæser filerne fra en given placering, den blander også filerne som default. Filerne bliver opdelt i data, target, targetnames og filenames. Se

2 Indlæsning af testdata

Herefter indlæses testdata på samme måde som træningsdata.

3 Datatransformation til billed 100*100*3

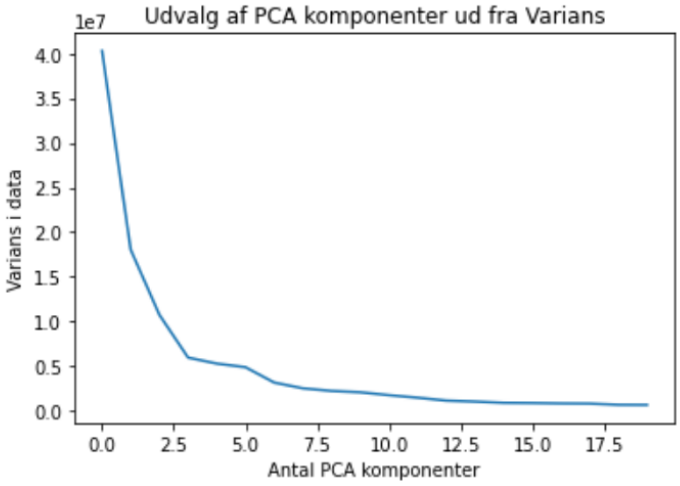
Data er af typen string, dette skal laves om til et matrix. Her bruges imread() fra imageio til transformationen.

4 Flatten data

Herefter bliver skal data laves til til et to dimentionelt matrix. Dette gøres, da sklearn.decomposition.PCA kun tager to dimentionelle indput. Til at udflade arrayet bruges flatten() fra numpy.

5 PCA oprettelse

Der bruges en PCA på datasættet, da der er 30000 features i datasættet, hvilket vil tage alt for lang tid at lave en model på. Samtidigt er der ikke meget information i mange af disse features. For at finde ud af hvor mange komponenter, er der blevet lavet et plot af varians kontra dimensioner. Dette plot kan ses på figur 1. Her ses at en næsten alt variansen er beskrevet i de første 8 dimensioner. Derfor vælges der at bruge 8 dimensioner i PCA'en.



Figur 1: Plot af varians kontra antal componenter

6 PCA træning og transformation

Det næste trin er at fitte PCA'en med data. Herefter transformeres datasættet så der nu er et datasæt med 8 features i stedet 30000.

7 Transformation af testdata

Testdata bliver ligesom træningsdata lavet til et to dimensionalt matrix. Herefter bliver testdata transformeret med den PCA, som blev trænnet med træningsdata.

8 Oprettelse af model

I dette trin oprettes SVC modellen, med de fundne parametre. fit() functionen bruges til at træne med træningsdata, og dens tilhørende targets.

9 Brug model på test data

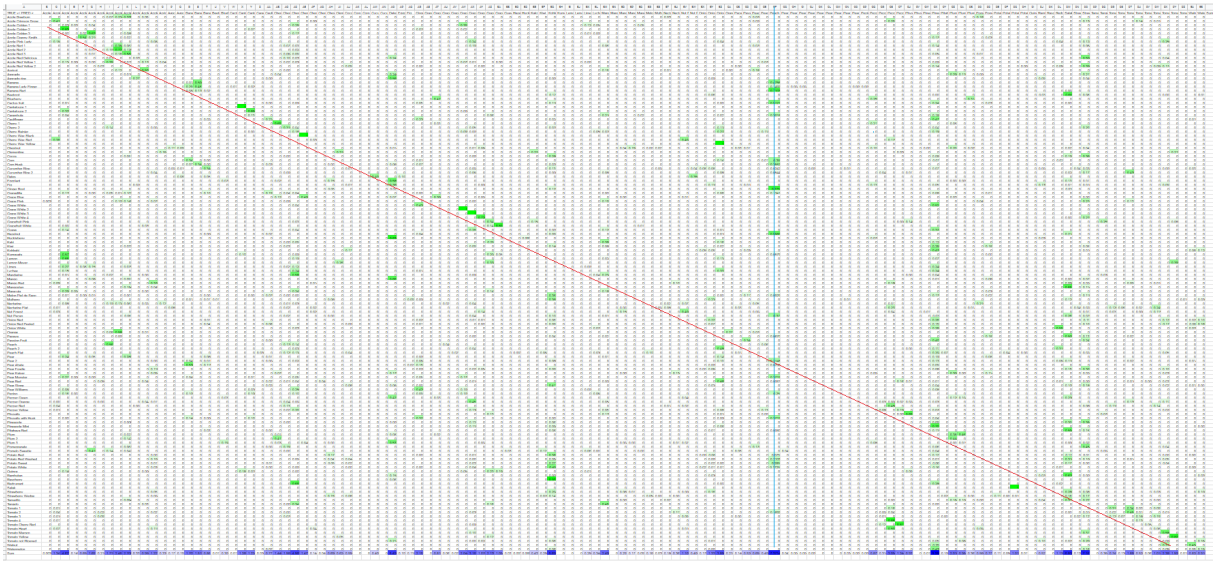
Når modellen er trænnet skal den bruges på testdata. Her bruges predict().

Performance metrics

Til at måle performance af modellen bruges en confusion matrix. Scikit funktionen plot_confusion_matrix() bruges, som tager modellen samt noget test data som parametre.

En confusion matrix bruges fordi der gerne vil ses hvilke frugter modellen har svært ved at predicte. Der er f.eks mange forskellige slags æbler med en form der ligner hinanden, så det vil gerne ses om modellen kan adskille dem. Det vil også gerne ses om modellen laver nogle indlysende fejl, som f.eks hvis den predicter en gul banan til at være et grønt æble.

Efter gruppen fik resultater på det første RandomSearch af en SVC model, blev modellen med den bedste score målt i en confusion matrix. Et heatmap af dette plot kan ses på figur figur 1. På y-aksen er de faktiske frugter og på x-aksen er de korresponderende predictions til frugterne.



Figur 1: Et overview af et Confusion Matrix Heatmap af den bedste scoret SVC model.

Som det kan ses på heatmappen kan man godt spotte en tendenslinje (rød streg) med hensyn til korrekte predictions. Linjen er dog lidt svag, hvilket vil sige at der er meget plads til forbedring.

Ud over tendenslinjen, kan man også se nogle svage lodrette tendens linjer. Hvis vi f.eks kigger på "peach flat"(blå streg), kan vi se at mange frugter gerne vil predictes til at være lige præcis denne frugt. Den har altså en høj prediction rate og er derfor meget forstyrrende og trækker vores overall score ned. De tre mest forstyrrende frugter er Peach flat, Pineapple Mini og Strawberry Wedge.

Kigges det lidt nærmere på heatmappet, figur 2, kan der ses et par ting. Kigger vi på "Apple Brauburn"kan det ses at modellen meget gerne vil predicte den til at være et rødt æble - hvilket lidt giver mening da det jo rent faktik er et rødt æble. Modellen er også tilbøjelig at predicte den til at være en "Pomegranet", hvilket også giver lidt mening da den har lignende farve og form. Modellen vil dog aldrig predicte den til at være den sande frugt, hvilket er besynderligt.

	A	B	D	G	H	I	J	K	DC
1	TRUE v/ PRED >	Apple Braeburn	Apple Golden 1	Apple Granny Sm	Apple Pink Lady	Apple Red 1	Apple Red 2	Apple Red 3	Pomegranate
2	Apple Braeburn	0	0	0	0	0.07317073171	0.2347560976	0.3353658537	0.1890243902

Figur 2: Æble eksempel fra confusion matrix.

Kigger vi nærmere på bananer, kan man se lidt det samme resultat. Modellen vil gerne predicte en normal banan til at være af typen banen, men den er tilbøjelig til at predicte den til at være typen "Banana Lady Finger". Predictions for Banana Lady Finger er lidt delt mellem en Normal banan og en Lady Finger og Banana Red vil gerne være en Peach Flat. Peach flat er generelt meget forstyrrende for bananerne. Dette skyldes formodenligt at den, alt efter hvilken vinkel den er i, både kan være rød eller gul, og rund eller aflang.

		R	S	T	CF
		Banana	Banana Lady Fin	Banana Red	Peach Flat
18	Banana	0.01829268293	0.506097561	0	0.4756097561
19	Banana Lady Finger	0.2910447761	0.4875621891	0	0.1492537313
20	Banana Red	0.08841463415	0.1768292683	0.02134146341	0.7103658537

Figur 3: Banan eksempel fra confusion matrix

Men alt i alt, kan man se at modellen har formået at sætte lidt system i kaosset af frugter.

Optimering

Optimeringen i brug af SVC modellen kommer i form af en RandomizedSearch på hyperparametrene der bruges i modellen. Herved er det ikke sikker at der fås noget værdi ud af optimeringen af hyperparametrene på det regulere, da scoren på Default modellen allerede er meget høj (**0.96**). Her kommer det roterede datasæt til brug, da den gennemsnitlige accuracy er meget lavere med ca. **0.14** med brug af PCA'en og 8 komponenter.

For selve optimeringen vil der bruges SVC modellen som før med en række definerede hyperparametre der kan gennemsøges. Søgningen vil udføres vha. sklearn model_selection's RandomSearchCV, der udfører et fastlagt antal iterationer af tilfældig sammensatte hyperparametre.
Som generelle parametre til selve RandomSearchCV blev der defineret følgende:

- CV = 5 - Antal cross-valideringer der bliver gennemført i hver iteration
- n_iter = 100 - Antal iterationer der bliver testet
- scoring = 'f1_micro' - Scoring metoden der bliver brugt
- n_jobs = -1 - Antal parallele processer, -1 betyder alle mulige processorer

Ud over modellen og disse parametre blev der defineret en matrix af modellens hyperparametre, som det kan ses på listing 1. For at evaluere optimerings processen vil funktionerne fra ITMAL's Lektion 8[2] genbruges der generer en FullReport af RandomSearch'en. Evalueringen vil i stort del består af f1 scoringen der blev defineret. Koden til denne optimering kan ses i Kodeappendixen i listing 9 og listing 10

Listing 1: Definering af tuning parametre

```
tuning_parameters = {
    'C': [0.001, 0.01, 0.1],
    'gamma': [0.0001, 0.001, 0.01, 0.1, 1],
    'kernel': ('linear', 'poly', 'rbf', 'sigmoid'),
    'degree': [1, 2, 3, 4]
}
```

Resultaterne af optimeringsforsøget varierer stærkt, idet f1_micro scoren enten lå under 0.03 eller var på næsten 1. Dvs. der var ingen hyperparametre der kunne få modellen til at få en middelmæsig score e.l., der kunne kun opnås de ekstreme værdier.
Hvis man kigger på resultat listen af de forskellige sammensatte hyperparametre kan der hurtigt ses at alle iterationer der landede i den dårlige score ende, havde sat kernel hyperparametren til enten 'rbf' eller 'sigmoid'. Iterationerne med den høje score har så enten haft kernel parametren på 'linear' eller 'poly'.

Ud fra disse meget forskellige resultater på basis af kernel'en var der tiltænkt en anden omgang af optimeringsforsøg. Dette var dog ikke muligt da GPU clusteret brød sammen da den var igennem 433/500 fits og bagefter ved PCA opsætningen allerede. Dermed kan der ikke kigges nærmere på resultaterne af 'rbf' og 'sigmoid' kernelen. Koden der var tiltænkt til det kan ses i Kodeappendixen listing 11

Over- og Underfitting

I Datasæt afsnittet blev der beskrevet, hvordan der blev lavet ændringer i datasættet for at forhindre potentiel overfitting. Dette skyldes at testsættet og trænningssættet lignede hinanden ufattelig meget. Her vil modellen så lære at genkende præcist, det æble der er trænnet på, men ikke andre æbler. Der blev kørt en SVC model på det originale datasæt, og dette gav en score på 1, hvilket viser at det var nødvendigt at lave ændringen. Her tales der om at modellen bliver overfittet i en høj grad.
Måden overfitting så kan ses typisk er at modellen scorer godt i valideringssættet men falder fra nå den skal testes på testsættet, i dette tilfælde blev scoren på testsættet så stor pga. at datasættene bestod af det samme æble og samplene lignede hinanden. I dette sammenhænge tales der stadigvæk om overfitting, selvom scorene ikke reflekterer det direkte.

Da datasættet er blevet opdelt, er det nede på kun at have ca. 300 samples af hver klasse. Dette betyder, at der kan ske en vis underfitting, da der ikke er nok data, til at formulere en ordentlig model. Dette kan ses i scoren på modellen der bruges på det nye roterede datasæt.

Selv efter opdelingen af datasættet lider datasættet stadigvæk under overfitting, nu igen med de typiske parametre at have en lav score på testsættet. Med andre ord så lærer modellen at genkender lige netop denne banan, og ikke andre bananer, eller den samme banan som er blevet mere moden. Dette er et betydeligt problem med datasættet, som er besværligt at ændre. Man kunne dog se på at tilføje modificeret data til trænningsættet. Her kunne man tage nogle originale billeder fra datasættet, og ændre farven lidt, fjerne dele af frugten, eller måske tilføje støj. Dette ville reducere overfittingen. En anden mulighed ville være at udvide datasættet med nogle nye samples der repræsenterer de samme klasser, men ikke er billede af den samme frugt, eller har en anden baggrund.

For at reducere overfitting i løbet af vores pipeline blev der brugt en PCA på datasættet, denne har reduceret antallet af features. Der blev brugt 8 komponenter, på basis af PCA analyse som det kunne ses i afsnit 5 under ”Pipeline”.

Konklusion

Datasættet, der blev valgt til denne opgave, er ikke optimalt. Det har givet flere problemer med overfitting, som er blevet forsøgt behandlet. Hvis opgaven skulle gentages vil det være oplagt at bruge et andet datasæt.

Fordelen med det valgte datasæt var at der ikke var behov for meget preprocessing i form af object-recognition, da dette allerede var gjort på forhånd. Ved valget af et andet datasæt, kunne det være nødvendigt selv at lave. Frugtene blev vist på en hvid baggrund, derfor var videre fremhævning af selve objektet overflødigt.

Valget af SVC-algoritment gav ikke problemer med trænningstiden. Det vil dog være oplagt at bruge en anden algoritme, hvis man vælger at bruge et andet frugt datasæt. Da hvis dette datasæt er større vil SVC give for lang trænningstid.

Confusion matricen derimod viser at scoren ikke nødvendigvis repræsenterer prediction kvaliteten fuldest. Selvom der opnås en score på næsten 1, kan der i confusion matricen ses, at det er langt fra at alle samples forudsiges til den rigtige frugt. Derfor kan der ikke direkte snakkes om overfitting her, men der burdes ses mere ind til forskellige performance metrics der kunne måle modellens resultater.

Når der ses på resultaterne af optimeringen kan der ses, at der også kan opnås en F1-score på ca. 1 på det roterede datasæt. Da optimeringen dog ikke forløb, som planlagt kunne der ikke kigges nærmere ind på dens resultater.
Når der kigges på resultaterne, der kom ud af den gennemførte optimering, kan der ses en stor forskel, når der enten bruges 'linear' og 'polynomial' eller 'rbf' og 'sigmoid'. Scoren bliver her enten maksimalt 0.03 eller næsten 1. Validitet af disse resultater kan så diskuteres idet det kunne tyde på en videre fejl i datasættet e.l.

Referencer

[1] Mihai Oltean. *Fruits 260*. https://www.kaggle.com/moltean/fruits. Accessed on 18/05-2021. 2020.

[2] Carsten Eie Frigaard. *L08: Regularisering, optimering og søgning*. https://blackboard.au.dk/webapps/blackboard/content/listContent.jsp?course_id=_145075_1&content_id=_2931033_1&mode=reset. Accessed on 18/05-2021, ITMAL tilgang på Blackboard forudsat. Apr. 2021.

Kode Appendix

Listing 2: Import de fleste funktioner

```
from sklearn.datasets import load_files
from os import getcwd
from imageio import imread, imsave
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
```

Listing 3: Load data og datasæt analyse

```
#%% Load data
# curDir = getcwd()
curDir = "./fruits-360/Training2"

dataSet = load_files(curDir, shuffle=False)

#%%

data = dataSet.data

testImg = data[0]

img = imread(testImg)

#%%

imageArray = []

for i in range(len(data)):
    imageArray.append(imread(data[i]))

#%%

flatImages = []

for i in range(len(data)):
    flatImages.append(imread(data[i]).flatten())

#%% Try with 2 components to visualize
pca = PCA(n_components=2)

pca.fit(flatImages)

inputData = pca.transform(flatImages)

#%%

v1 = pca.components_
plt.plot(pca.explained_variance_)
plt.title("Varianse")
plt.show()

```

```
#%% Projektion paa 2D til visualisering

plt.scatter(inputData[:,0], inputData[:,1])
plt.title("Data plot with 2 components")

#%% Plot based on type and color
n = 492;
applesGS = inputData[0:n-1,:]
applesRED = inputData[n:(2*n)-1,:]
banana = inputData[2*n:(3*n)-1,:]
eggplant = inputData[3*n:(4*n)-1,:]
lemon = inputData[4*n:(5*n)-1,:]
mango = inputData[5*n:(6*n)-1,:]

plt.scatter(applesGS[:,0], applesGS[:,1],color='g',linewidths=0)
plt.scatter(applesRED[:,0], applesRED[:,1],color='r',linewidths=0)
plt.scatter(banana[:,0], banana[:,1],color='y',linewidths=0)
plt.scatter(eggplant[:,0], eggplant[:,1],color='m',linewidths=0)
plt.scatter(lemon[:,0], lemon[:,1],color='b',linewidths=0)
plt.scatter(mango[:,0], mango[:,1],color='c',linewidths=0)
plt.legend(dataSet.target_names)
plt.title("Data sorted")
#This show that we need more components to classify the data.
```

Listing 4: Load data, preproccesering og fit/predict på standard datasæt

```
#%% Test with real data
#curDir = getcwd()
curDir = "./fruits-360/Training2"

trueFiles = load_files(curDir)

data2 = trueFiles.data


X = []
y = trueFiles.target
for i in range(len(data2)):
    X.append(imread(data2[i]).flatten())


#%% Now with 20 components to visualize
pca2 = PCA(n_components=8)

pca2.fit(X)

processedX = pca2.transform(X)

#%% Split in trainning and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(processedX,y,test_size
    =0.2)

#%%
import sklearn.svm as svm
clf = svm.SVC()

clf.fit(X_train,y_train)
```

```
32
33 #%%
34
35 y_pred = clf.predict(X_test)
36
37 s = clf.score(X_test,y_test)
38
39 print("Score: ", s)
```

Listing 5: Load roteret datasæt

```
1  #%% Test with rotated data
2  from sklearn.utils import shuffle
3
4  trainDir = "./fruits-360/rotated data/Training"
5  testDir = "./fruits-360/rotated data/Test"
6  print("set path done")
7  trainFiles = load_files(trainDir)
8  testFiles = load_files(testDir)
9  print("load files done")
10 dataTrain = trainFiles.data
11 dataTest = testFiles.data
12 print("get data done")
13
14
15 X_train = []
16 y_train = trainFiles.target
17 for i in range(len(dataTrain)):
18     X_train.append(imread(dataTrain[i]).flatten())
19 print("imread Train done")
20
21 X_test = []
22 y_test = testFiles.target
23 for i in range(len(dataTest)):
24     X_test.append(imread(dataTest[i]).flatten())
25 print("imread Test done")
26
27 # Shuffle
28 X_train, y_train = shuffle(X_train, y_train, random_state=42)
29 X_test, y_test = shuffle(X_test, y_test, random_state=42)
30
31 print("shuffle done")
```

Listing 6: PCA setup til brug med roteret data

```
1  #%% Now with 8 components
2  pca2 = PCA(n_components=8)
3  print("pca setup done")
4  X_train_trans = pca2.fit_transform(X_train)
5  print("pca fit transform training done")
6  X_test_trans = pca2.transform(X_test)
7  print("pca fit transform test done")
8
9  #%%
10 import sklearn.svm as svm
11
12 clf = svm.SVC()
13 print("SVC setup done")
```

```
clf.fit(X_train_trans,y_train)
print("SVC train done")
#%%

#y_pred = clf.predict(X_test_trans)
#print("SVC pred done")
s = clf.score(X_test_trans,y_test)
print("Score: ", s)
```

Listing 7: PCA setup med 20 komponenter til Varians plot

```
#%% Now with 20 components
pca3 = PCA(n_components=20)
print("pca setup done")
X_train_trans20 = pca3.fit_transform(X_train)
print("pca fit transform training done")
X_test_trans20 = pca3.transform(X_test)
print("pca fit transform test done")

#%%
import sklearn.svm as svm

clf20 = svm.SVC()
print("SVC setup done")
clf20.fit(X_train_trans20,y_train)
print("SVC train done")
#%%

#y_pred = clf20.predict(X_test_trans20)
#print("SVC pred done")
s = clf20.score(X_test_trans20,y_test)
print("Score: ", s)
```

Listing 8: Plot af Varians over PCA komponenter

```
plt.plot(pca2.explained_variance_)
plt.title("Udvalg af PCA komponenter ud fra Varians")
plt.xlabel("Antal PCA komponenter")
plt.ylabel("Varians i data")
plt.show()
```

Listing 9: Setup optimeringsvisualiserings funktioner fra MAL Lektion 8[2]

```
#%% Optimerings funktioner fra L8

from time import time
import numpy as np

from sklearn import svm
from sklearn.linear_model import SGDClassifier

from sklearn.model_selection import GridSearchCV, RandomizedSearchCV,
    train_test_split
from sklearn.metrics import classification_report, f1_score
from sklearn import datasets

currmode="N/A" # GLOBAL var!

```

```
15 def SearchReport(model):
16
17     def GetBestModelCTOR(model, best_params):
18         def GetParams(best_params):
19             ret_str=""
20             for key in sorted(best_params):
21                 value = best_params[key]
22                 temp_str = "'" if str(type(value))=="<class 'str'>" else ""
23                 if len(ret_str)>0:
24                     ret_str += ','
25                 ret_str += f'{key}={temp_str}{value}{temp_str}'
26             return ret_str
27         try:
28             param_str = GetParams(best_params)
29             return type(model).__name__ + '(' + param_str + ')'
30         except:
31             return "N/A(1)"
32
33     print("\nBest model set found on train set:")
34     print()
35     print(f"\tbest parameters={model.best_params_}")
36     print(f"\tbest '{model.scoring}' score={model.best_score_}")
37     print(f"\tbest index={model.best_index_}")
38     print()
39     print(f"Best estimator CTOR:")
40     print(f"\t{model.best_estimator_}")
41     print()
42     try:
43         print(f"Grid scores ('{model.scoring}') on development set:")
44         means = model.cv_results_['mean_test_score']
45         stds  = model.cv_results_['std_test_score']
46         i=0
47         for mean, std, params in zip(means, stds, model.cv_results_['params
               ']):
48             print("\t[%2d]: %0.3f (+/-%0.03f) for %r" % (i, mean, std * 2,
                   params))
49             i += 1
50     except:
51         print("WARNING: the random search do not provide means/stds")
52
53     global currmode
54     assert "f1_micro"==str(model.scoring), f"come on, we need to fix the
           scoring to be able to compare model-fits! Your scoreing={str(model.
           scoring)}...remember to add scoring='f1_micro' to the search"
55     return f"best: dat={currmode}, score={model.best_score_:0.5f}, model={
           GetBestModelCTOR(model.estimator,model.best_params_)}", model.
           best_estimator_
56 print("SearchReport done")
57 def ClassificationReport(model, X_test, y_test, target_names=None):
58     assert X_test.shape[0]==y_test.shape[0]
59     print("\nDetailed classification report:")
60     print("\tThe model is trained on the full development set.")
61     print("\tThe scores are computed on the full evaluation set.")
62     print()
63     y_true, y_pred = y_test, model.predict(X_test)
64     print(classification_report(y_true, y_pred, target_names))
65     print()
```

```
print("ClassificationReport done")
def FullReport(model, X_test, y_test, t):
    print(f"SEARCH TIME: {t:0.2f} sec")
    beststr, bestmodel = SearchReport(model)
    ClassificationReport(model, X_test, y_test)
    print(f"CTOR for best model: {bestmodel}\n")
    print(f"{beststr}\n")
    return beststr, bestmodel
print("FullReport done")
```

Listing 10: Optimering gennemgang nr. 1

```
#%% Optimering
import sklearn
# Setup search parameters
model = svm.SVC()
print("model setup done")
tuning_parameters = {
    'C': [0.001, 0.01, 0.1],
    'gamma': [0.0001, 0.001, 0.01, 0.1, 1],
    'kernel': ('linear', 'poly', 'rbf', 'sigmoid'),
    'degree': [1, 2, 3, 4]
}
print("tuning param setup done")

CV = 5
VERBOSE = 10

# Run RandomizedSearchCV for the model
start = time()
random_tuned = RandomizedSearchCV(
    model,
    tuning_parameters,
    n_iter=100,
    random_state=42,
    cv=CV,
    scoring='f1_micro',
    verbose=VERBOSE,
    n_jobs=-1,
    iid=True
)
print("Rand pipeline setup done ")
random_tuned.fit(X_train_trans, y_train)
t = time() - start
print("Fit done")
# Report result
b0, m0 = FullReport(random_tuned, X_test_trans, y_test, t)
print('done')
```

Listing 11: Optimernig gennemgang nr. 2

```
#%% Optimering

import sklearn
# Setup search parameters
model = svm.SVC()
print("model setup done")
C_range = np.logspace(-2, 10, 13)
```

```
gamma_range = np.logspace(-9, 3, 13)
tuning_parameters = {
    'C': C_range,
    'gamma': gamma_range,
    'kernel': ('rbf', 'sigmoid'),
}
print("tuning param setup done")

CV = 5
VERBOSE = 10



# Run RandomizedSearchCV for the model
start = time()
random_tuned = RandomizedSearchCV(
    model,
    tuning_parameters,
    n_iter=100,
    random_state=42,
    cv=CV,
    scoring='f1_micro',
    verbose=VERBOSE,
    n_jobs=-1,
    iid=True
)
print("Rand pipeline setup done ")
random_tuned.fit(X_train_trans, y_train)
t = time() - start
print("Fit done")

# Report result
b0, m0 = FullReport(random_tuned, X_test_trans, y_test, t)
print('done')
```

Listing 12: Setup model til confusion matrice

```
#%% Confusion Matrix
from sklearn.metrics import plot_confusion_matrix
import pandas as pd

# Run classifier, using a model that is too regularized (C too low) to see
# the impact on the results
classifier = svm.SVC(C=0.001, gamma=0.001, kernel='poly').fit(X_train_trans
    , y_train)
```

Listing 13: Plot og print til .csv fil af confusion matrix

```
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(classifier, X_test_trans, y_test,
                                 display_labels=trainFiles.target_names,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
```

```
11      disp.ax_.set_title(title)
12
13      # print(title)
14      # print(disp.confusion_matrix)
15
16      DF = pd.DataFrame(disp.confusion_matrix, columns=trainFiles.
            target_names, index=trainFiles.target_names)
17      DF.to_csv(title + ".csv")
```