Project 3: De Appel Industrie

Team: Kajal, Marina, Hans, Hans, Hossein

Onderbouwing van de verschillende stappen en keuzes die wij gemaakt hebben

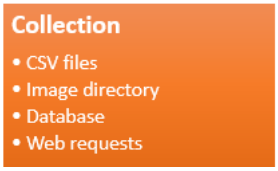
De 1ste stappen die wij hebben genomen zijn:

- GIT inrichten.
- Teams inrichten.
- Business vraag inventariseren: die is weergegeven in project omschrijving.
- Stappenplan maken. Dit plan is een Word document: Project 3 – plan van aanpak, te vinden.

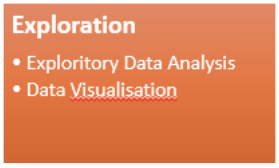
Dit document geeft de onderbouwing weer voor de belangrijkste keuzes die wij hebben gemaakt gedurende het project. Daarnaast zijn er verschillende overzichten van testresultaten geïntegreerd.

Dit project bestaat uit meerdere onderdelen. Het doel is om de volgende onderdelen stuk voor stuk werkend te krijgen en ze op een logische manier aan elkaar te knopen in een bruikbaar 'proof of concept'.

- Afwijkingen in appels herkennen
- Keuring uitvoeren op een of meerdere appel batches
- Statistieken over de batches weergeven
- Statistieken uitvragen via chat



Wij hebben voor de data collectie een project dataset gekregen. Die staat, onder Project 3, in de "data" folder. De structuur is hetzelfde: Train en test folders waarin de verschillende images van appel klassen (Normal, Scab, Rot en Blotched) staan. Onder data hebben wij tevens de toevoegingen geplaatst die wij later in het project hebben gekregen.



De observaties over de "kwaliteit" van de dataset:
Er zitten files van verschillende formaten in.
Er zitten foto's en plaatjes (tekeningen) in.

De hoeveelheden van de verschillende klassen zijn anders.

a. Er zitten dubbele foto's en plaatjes in.
b. Er zijn relatief weinig files/images.
c. De plaats van het onderwerp (de appel) is vaak verschillend in een plaatje.
d. Soms staan er meerdere appels op 1 plaatje.
e. De achtergronden van de foto's/plaatjes zijn niet hetzelfde.
f. Er staat in een aantal gevallen een soort watermerk in de afbeelding.

Ten 1ste hebben wij, op basis van de kleine dataset en de grote hoeveelheid verschillende data kwaliteitsaspecten (zie punt 2), de set handmatig opgeschoond. Zie bijgaande tabel:

Mogelijk probleem	Actie	Code - manual	Volgorde
Bewerkte plaatje - appel met pijltjes	Outlier	Manual	1
Geen foto (plaatje/schilderij)	Outlier	Manual	1
Juiste categorie	Verplaatsen	Manual	2
Identieke foto's inclusief augmentations	Doublures eruit halen	Te automatiseren	2
Identieke plaatjes in dezelfde dataset	Doublures eruit halen	Te automatiseren	2
Identieke plaatjes in verschillende datasets	Doublures eruit halen	Te automatiseren	2
Al bewerkte foto's resized	1 size genereren		3
Appel in boom	Label img gebruiken		4
Deel van appel zichtbaar door kaders	Label img gebruiken		4
Deel van appel zichtbaar door ander object	Label img gebruiken		4
Verschillende achtegronden	Label img gebruiken		4
Meerdere appels op 1 afbeelding	Label img gebruiken		4
Kleur	Moeten testen of kleur invloed heeft		5

Uiteindelijk hebben we niet alle voorgenomen acties uitgevoerd. We hebben handmatig alle deel-sets nagelopen en een aantal foto's verwijderd – zie overzicht. Met name de acties die we met Label Img hadden willen uitvoeren, zijn niet uitgevoerd omdat we zijn gaan "croppen"? Tijdens het vergelijken van de train-resultaten van model-trainingen op basis van de 'ongeschoonde' en 'geschoonde' data bleek dat het model doorgaans beter presteerde als het was getraind op 'ongeschoonde' data.
Overigens hebben we nog wel een .py bestand gemaakt (appelcropper.py) waarmee we snel grote hoeveelheden appels van hun achtergrond konden ontdoen.

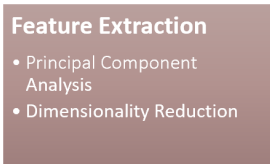
Daarnaast hebben we een aantal transformations gedaan op de train-set onder andere om:

- Het aantal images te vergroten.
- Ze identiek in grootte te maken.
- Zie de lijst hieronder (Kan je hier niet een stukje code uit het notebook plakken? Niet alle genoemde transformations komen me bekend voor. Bovendien zijn sommige van de transformations bedoeld als '**preprocessing**' – om de verwerking van de plaatjes door pytorch goed te laten verlopen (bv resize, rescale), en andere bewerkingen zijn uitgevoerd bij wijze van '**augmentation**'. Die behandelingen, rotaties, flippen, kleur aanpassen etc moeten het systeem

voeden met veel meer verschillende afbeeldingen dan dat we in werkelijkheid hadden).

-

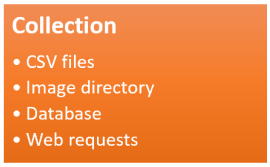
```
# Resize images to 224x224 pixels
# Randomly crop and resize images
# Randomly flip the images horizon
# Randomly rotate the images by 36
# Randomly adjust brightness, cont
# Convert images to tensors
# Normalize the images
```



Een Wikipedia quote over Feature Extraction:

“Bij machinaal leren, patroonherkenning en beeldverwerking begint kenmerkextractie bij een initiële set meetgegevens en bouwt afgeleide waarden (kenmerken) op die bedoeld zijn om informatief en niet-redundant te zijn, om de volgende leer- en generalisatiestappen te vergemakkelijken en in sommige gevallen te leiden tot betere menselijke interpretaties. Kenmerkextractie is gerelateerd aan dimensionaliteitsreductie.[1].”

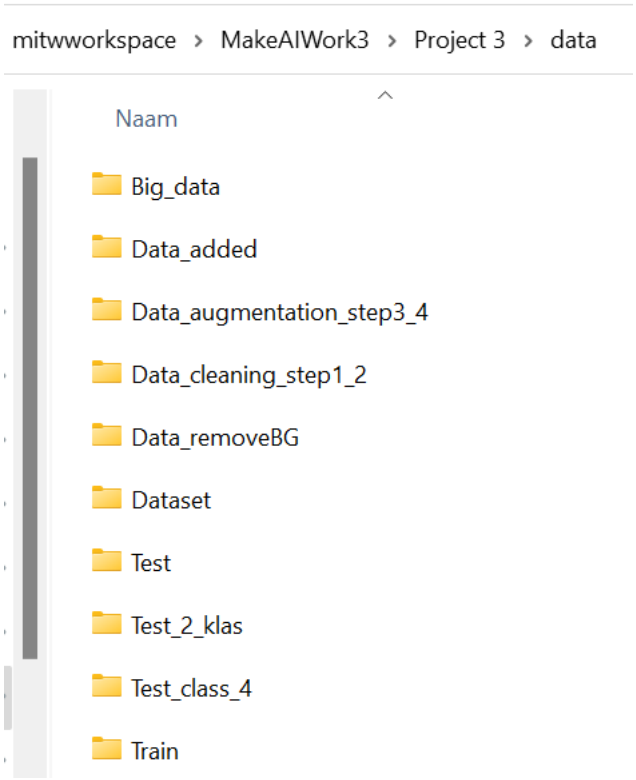
In ons geval heeft het Neural Net dit gedaan.



Zoals aangegeven onder punt 2 is het aantal files wat we hebben relatief klein. Daarnaast is de verdeling niet optimaal voor de opdracht. In ieder geval als we zouden willen werken met een classifier die gebruik maakt van een getal (uitkomst van combinatie model module en AQL classifier, dat wordt vergeleken met een AQL van 0,4% goede appels om het als “een goede appelbatch te kunnen labelen”. Dit heeft enerzijds een effect op het lerend vermogen. Wij hebben gekeken naar meerdere opties op dit probleem op te lossen. We hebben ervoor gekozen om een pre-trained model (Resnet) te gebruiken waardoor het kleine lerend vermogen enorm wordt versterkt. Het gebruik van transferlearning (de bestaande weights and bias van Resnet hergebruiken voor het trainen van onze eigen data) is enorm krachtig. Zelfs met een kleine dataset komen er nog redelijke resultaten uit. Daarnaast hebben wij er voor gekozen om plaatjes toe te voegen aan onze dataset. Deze plaatjes hebben wij van Kaggle).

Al de data files zijn in verschillende “Image Directories opgeslagen. De naam van de folder refereert aan de origine van de dataset. Ter herkenning: de plaatjes die van

Kaggle afkomstig zijn in aparte mappen binnen de dataset opgeslagen. Verder hebben wij een CSV file gebruikt om de resultaten van de training op te slaan.

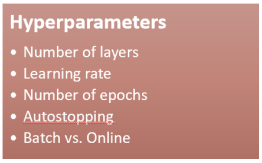


De train en test datasets zijn gegeven aan het begin van het project. Omdat de dataset heel klein was hebben we voor test en validatie initieel dezelfde dataset gebruikt. Voor de grote, via Kaggle, verkregen dataset hebben we een verdeling van 80%/10%/10% gebruikt.

Wij hebben voor dit probleem een classificatie gebruikt met een unsupervised learning model. Het plaatsje dat wordt aangeboden aan het model moet, middels een voorspelling, in 1 van 4 klassen (Normal, Blotched, Rot and Scab) worden ondergebracht.

Apple classifier:
Eerst alle images resizen naar 224 x 224, anders kan je er geen Tensors van maken.
Opbouw van de lagen in het model:

- Conv1 = 224 – 3 channels + 1 (altijd) + 2 (voor de stride)
- maxpool = 2 → 224/2 = 112
- Conv2 = 112 – 3 channels + 1 (altijd) + 2 (voor de stride)
- maxpool = 2 → 112/2 = 56
- Naar lineair: 32 x 56 x56, 128 met 32 output vorige laag en 56 x 56 zoals hierboven berekend. 128 is input voor volgende laag.



De keuze welke hyperparameters te gebruiken en welke dataset het beste te gebruiken is gebaseerd op een trainingsprogramma. De uitkomsten van iedere training zijn opgeslagen in een CSV, wat we vervolgens naar Excel hebben geconverteerd om de uitkomsten te analyseren. Zie bijgaande tabellen.

Onze uiteindelijk keuze is:
Werken met ResNet, met een learning rate van 0.0001 met 50 Epochs optimaliseert de Test accuracy.

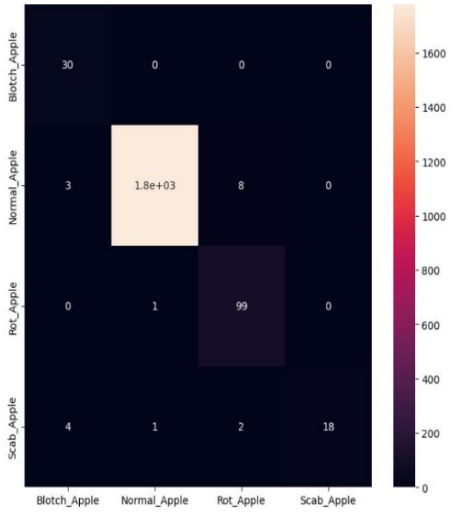
Gemiddelde van Test Accuracy	Kolomlabels Learning Rate/NN					
	resnet18				Team model	
Rijlabels Dataset	0.0001	0.001	0.01	0.1	0.0001	0.1
Train_Kaggle_Uncleaned_4_classes	99,1	97,9		58,0	85,4	
Train_Combination_source_Cleaned_4 classes	99,1		8,3		92,2	63,4
Train_ORIG_Cleaned2x_4_classes2		84,4				

Gemiddelde van Test Accuracy	Kolomlabels Epochs				
Rijlabels	1	2	10	50	100
resnet18					
Train_Kaggle_Uncleaned_4_classes	51,66		98,17	99,81	99,605

Performance Measure

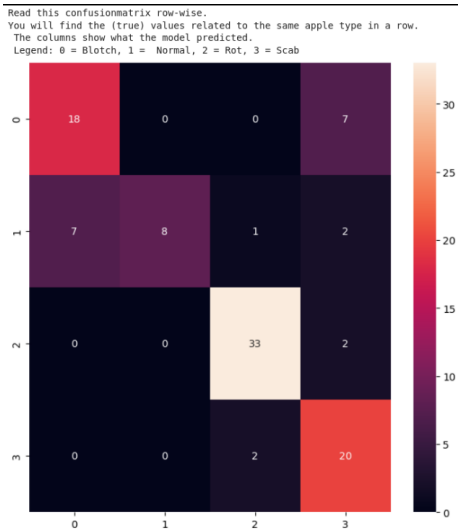
- Loss function / Accuracy
- False Positives
- False Negatives
- Confusion Matrix

De bijgaande matrix toont de goede uitkomsten op de diagonaal. Al de andere uitkomsten zijn fout. Het geeft een goede indicatie waar het model fouten maakt. Daarnaast geeft het inzicht in de kwaliteit/betrouwbaarheid van de uitkomst van het model.

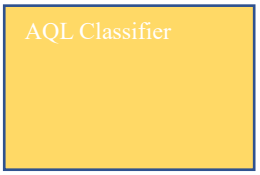


De uitkomst van het model (weights and biases) wordt gebruikt om in de AQL Classifier een image in de juiste klasse te plaatsen. Men pakt een sample van 32 willekeurige appels (1 run). Door dit eventueel een aantal keer (runs) te herhalen wordt een kwaliteits-overzicht van een appel batch gemaakt. Door het aantal appels / het percentage appels dat slecht' is te voorspellen kan er uiteindelijk een klasse aan een batch gegeven worden.

Onderstaande heatmap, die we hebben gemaakt aan het begin van het project, van de originele, ongeschoonde dataset.



Dit overzicht laat zien dat dit model, met de daarbij behorende dataset niet bruikbaar is voor "commercieel" gebruik als feed voor de kwaliteits-classifier.



Zie hieronder de klassen en kwaliteitsniveaus die worden gebruikt in de applicatie.

Keuring uitvoeren op een of meerdere appel *batches*
Model testen en kwalificeren adhv AQL: Acceptance Quality Limit
AQL is een protocol is een industriestandaard voor statistische kwaliteitscontroles.

We hanteren de volgende kwaliteitslabels:
1. *Klasse 1: kwaliteit appels goed genoeg om in supermarkt/groenteboer te liggen*
2. *Klasse 2: kwaliteit appels is goed genoeg voor verwerking in appelmoes*
3. *Klasse 3: kwaliteit appels is goed genoeg voor verwerking in stroop*
4. *Afgekeurd: kwaliteit is onvoldoende voor bovenstaande toepassingen*

Per batch:

Klasse	**AQL**	
Klasse 1	AQL <= 0.4	supermarkt
Klasse 2	0.4 < AQL < 6.5	*appelmoes*
Klasse 3	6.5 < AQL < 15.0	*stroop*
Klasse 4	AQL > 15.0	*afgekeurd*

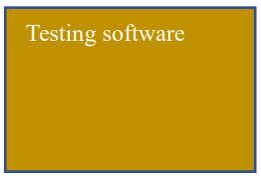
Hierover kunnen we, zodra we dit proces herhalen, statistieken uit halen.

De appel classifier kan worden aangesproken via een interface. Deze interface geeft, na input van een aantal velden, weer in welke klasse de batch valt.
Daarnaast geeft de interface zowel resultaten die gevoed worden door het Resnet Model als door het Team Model. Daardoor kunnen we goed zien dat het Resnet Model nauwkeuriger is.

Daarnaast kan er, via een geïntegreerde chatbot. Op basis van MiniLM en SBERT Vragen gesteld worden aan het systeem. Dit kunnen zowel kwalitatieve als kwantitatieve vragen zijn.

Het gaat hierbij om de herkenning van de semantische overeenkomst tussen zinnen die de gebruiker intypt en standaardzinnen waarop een standaardantwoord voorhanden is. Zo'n standaardantwoord kan variabele parameters bevatten zoals aantallen.

In eerste instantie was voor 'model = SentenceTransformer('distilbert-base-nli-mean-tokens')' gekozen. Daar zijn we op teruggekomen omdat de prestatie van de chatbot maar wel beter werd met SBert(MiniLM-L12-V2) . Dit model bleek namelijk beter in staat om sleutelwoorden in een vraag te identificeren. Helaas betekent deze keuze ook dat de chatbot iets minder snel reageert dan met 'distilbert-base-nli-mean-tokens'.



Uit de broncode hebben we een klein stukje code genomen, namelijk dat deel waar we op basis van de gewenste AQL en de uitslag van de appel-batch-keuring moeten bepalen welke eindbestemming die batch appels krijgt.
Allereerst hebben we dat specifieke stukje code voor test-doeleinden omgezet naar een functie: batch_qualifier(), opgenomen in de file 'aql_qualifier_code.py'.

Vervolgens hebben we een bestand 'aq_test.py' aangemaakt en daarin de functie 'testBatch_Qualifier()' geformuleerd.
In deze functie hebben we een aantal 'assert' testen opgenomen. Deze testen zijn vooral gericht op de grenswaarden van de oorspronkelijke functie. Zou op de een of andere manier een van deze grenswaarden veranderen, dan zou dat via een AssertionError melding onder de aandacht moeten worden gebracht.
Verder bevat dit .py-bestand ook nog de functie 'run_test()' dat de testen uit de testBatch_Qualifier uitvoert.

Ten slotte is er nog een ' main.py' waarin de functie 'testBatch_Qualifier' wordt aangeroepen. Zou 1 van de assert-testen falen, dan start de hele functie 'batch_qualifier' niet.