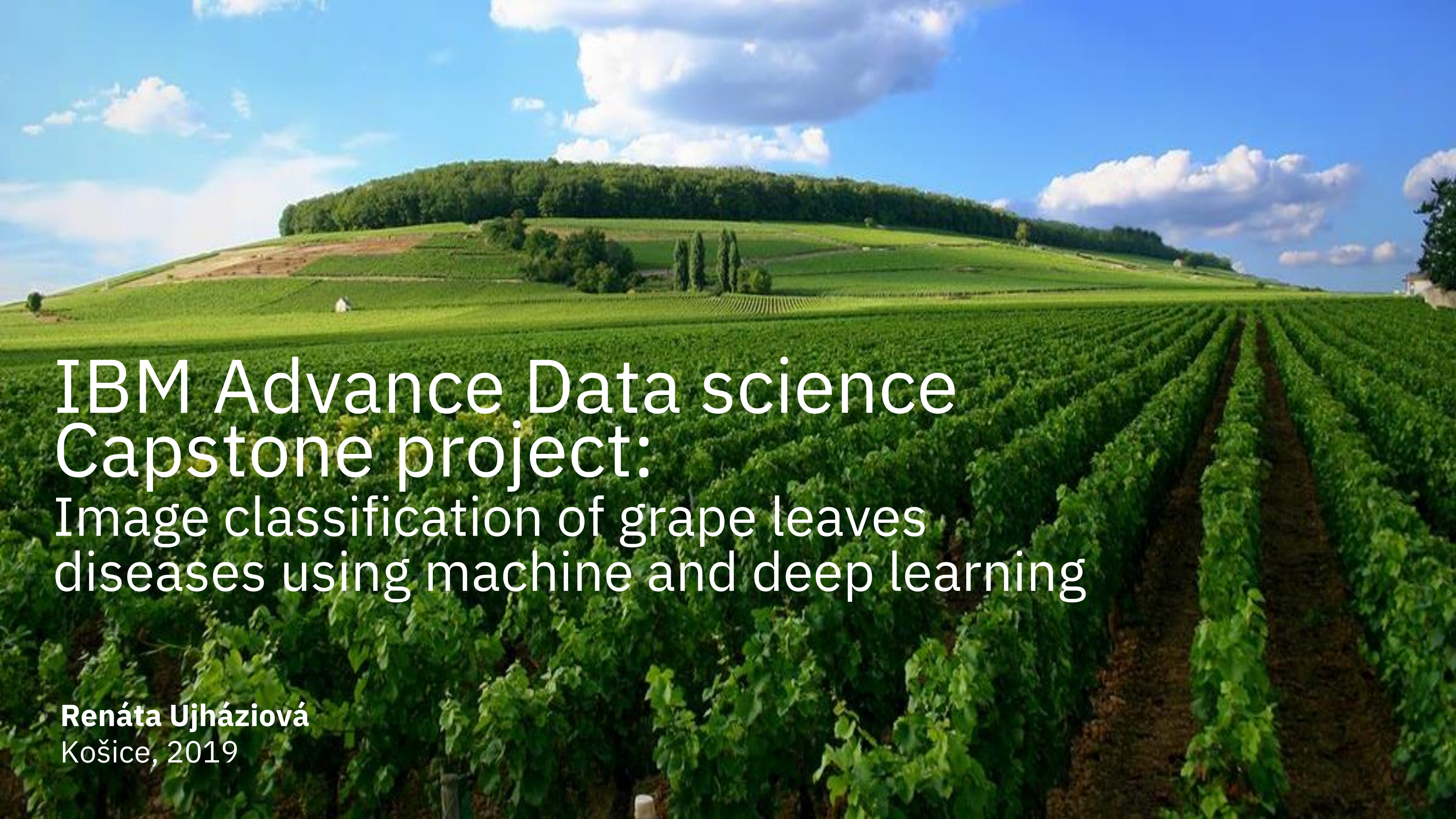


IBM Advance Data science Capstone project:

Image classification of grape leaves diseases using machine and deep learning

Renáta Ujháziová
Košice, 2019

The goal of the Capstone project

1. Prepare the image classification models in order to classify grape leaves images into 4 different classes.

2. Deploy the model in Watson studio using Watson Machine learning repository and create the REST API.

3. Show how to use the image classification REST API in the real-life use case.

Presentation for stakeholders

Introduction

Mr. Andrew's experience with grape diseases

Mr. Andrew is a 65-year-old owner of the Vineyard in South East of Slovakia. He was not in his vineyard for several days.
Although he did not notice anything at the last visit, now he found out that one part of his vineyard was infected with the esca disease in a very large extent and in another part of the vineyard the other disease called mite spread out.
He lost 30% of his grapes because he did not take a necessary action earlier.

Current problems of the Vineyard owners

Common questions

- How could I detect diseases in early stages and actively protect my Vineyard?
- How can I prevent the spread out of the grapevine disease?
- Can I do that with limited resources?

Too much time

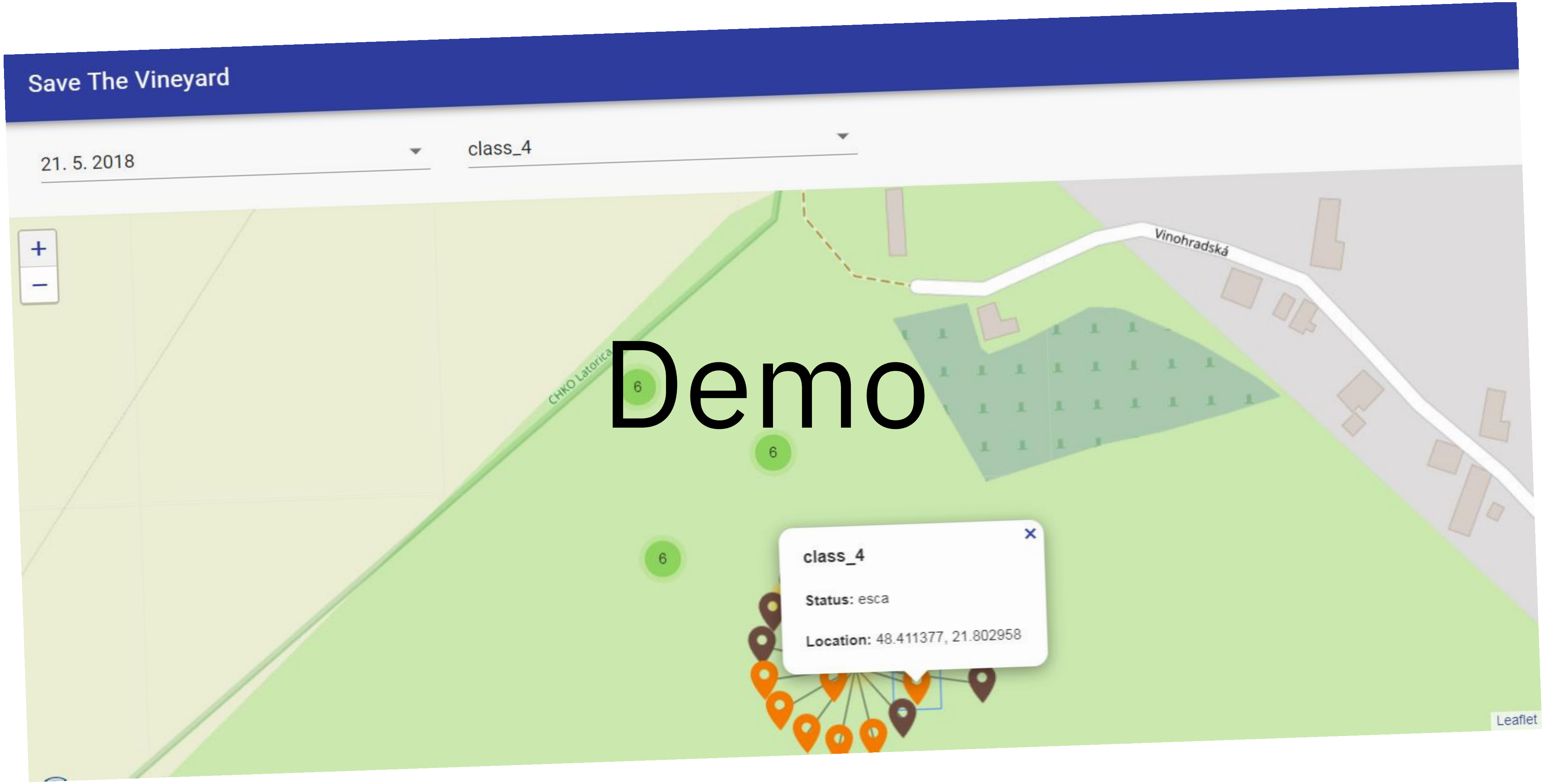
- Manually identify the locations with grapevine diseases.
- Take necessary actions at the right place.
- Recover from the disease.


What if…

There is cloud based solution

Save the Vineyard

Demo



Save The Vineyard
21. 5. 2018
class_4
Vinohradská
CHKO Latorica
6
6
6
class_4
Status: esca
Location: 48.411377, 21.802958
Leaflet

SMS notification with link to the application

https://save-the-vineyard-ui.eu-de.mybluemix.net/

Save the Vineyard's key features

Process Automation

Automatic retrieval of time, location and image data.

Machine Learning

Visual recognition to identify and classify grape leaves diseases.

Give initial feedback about the diseases spread.

SMS notification

SMS notification to the Vineyard owner in case of the grape leaves disease.

Data visualization

Data visualization with information about location and type of disease.

High level application architecture

IBM Cloud
is used to keep
all components.

Watson IoT
Is used to capture
images and obtains
GPS coordinates.

Cloudant
is used to store
captured images.

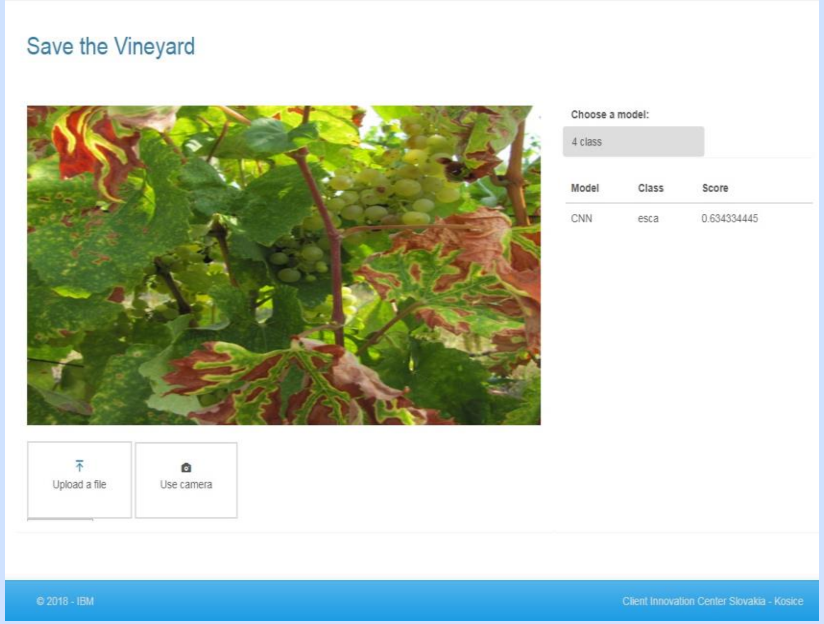
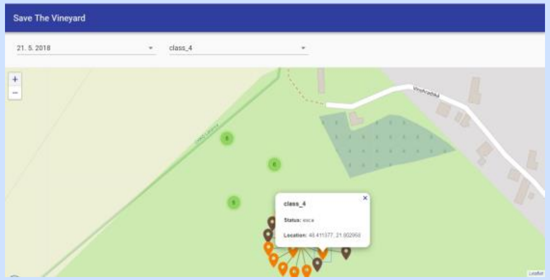


Image Classification REST API is used
to classify grape leaves diseases.



Web application is used to
visualize data after classification.

Twilio
is used to send alert via
sms to the Vineyard owner.

Cloudant
is used to store data
after classification.

Node Red is used for orchestration.

Presentation for data scientist

Architectural choices

I used

- **for the raw data:** my own dataset of grape leaves.
- **for the data repository:** IBM Cloud object storage and Github repository.
- **for data exploration, data preprocessing and modeling:** Jupyter notebook with Python 3.5.
- **for the solution, the following Python libraries:**
  - NumPy
  - Pandas
  - Seaborn
  - Scikit learn
  - Xgboost
  - Keras
  - Watson-machine-learning-client

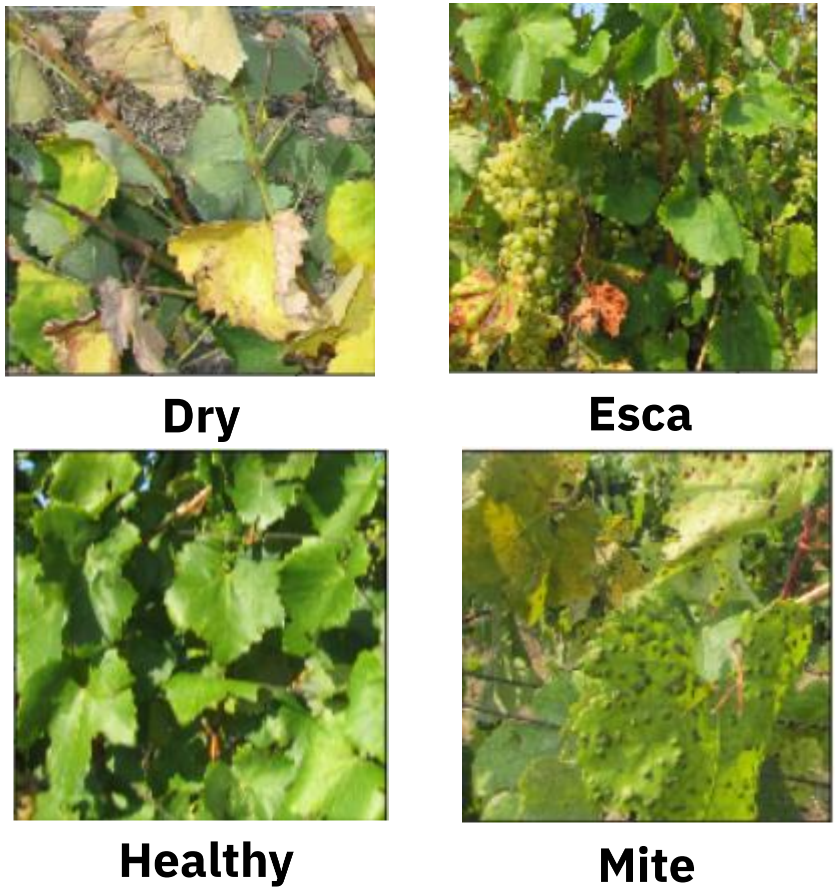
The dataset

The dataset contains 272 images.

The structure of the dataset:

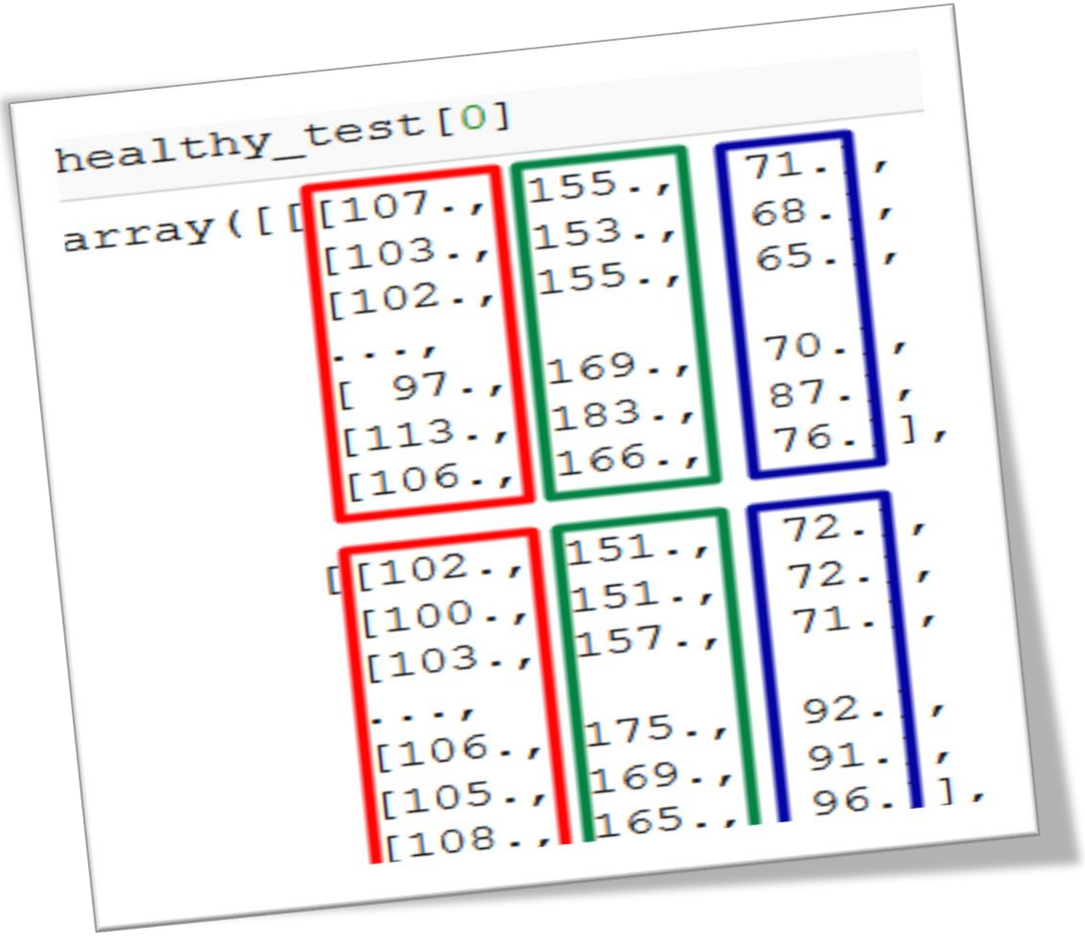
- Test_set (48 images)
- Training_set (200 images)
- Validation_set (12 images)

The dataset is available:
https://github.com/RenataUjhaziova/dataset WS_DvsEvsHvsM_512-512_git.git

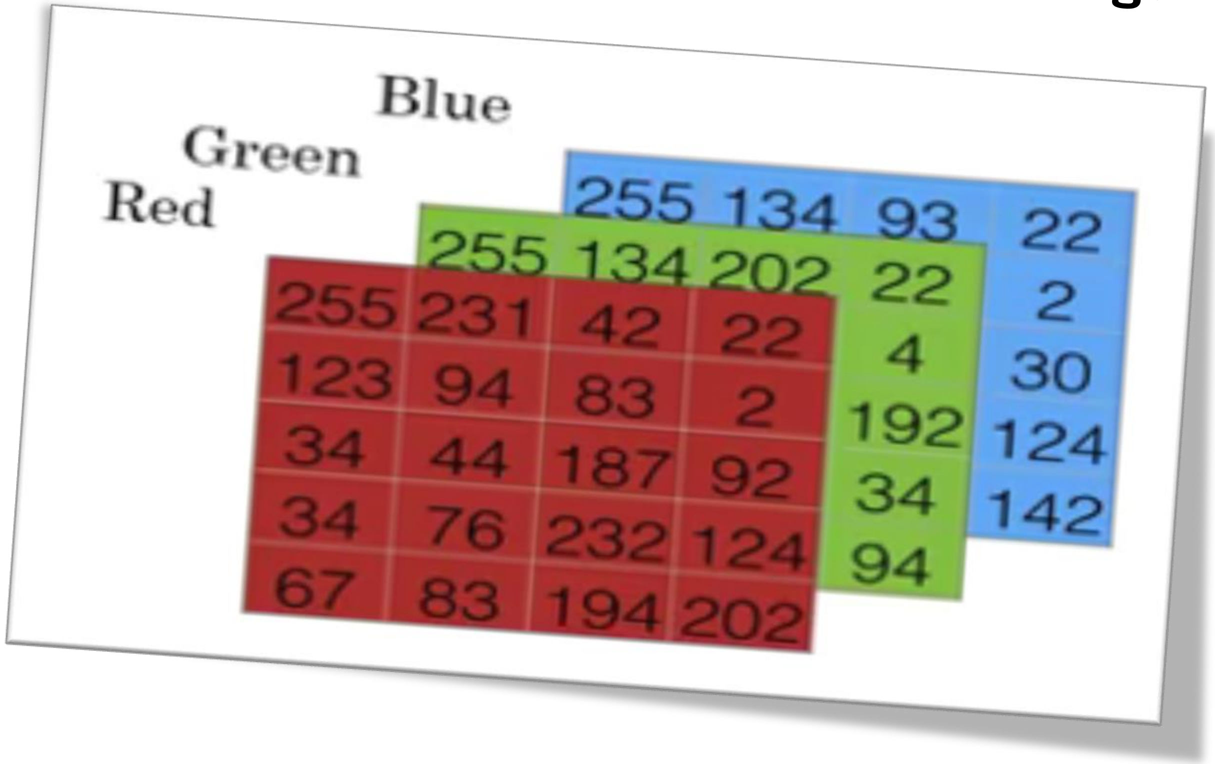


Data understanding

Color images are stored as **RGB values of pixels** (description of picture).

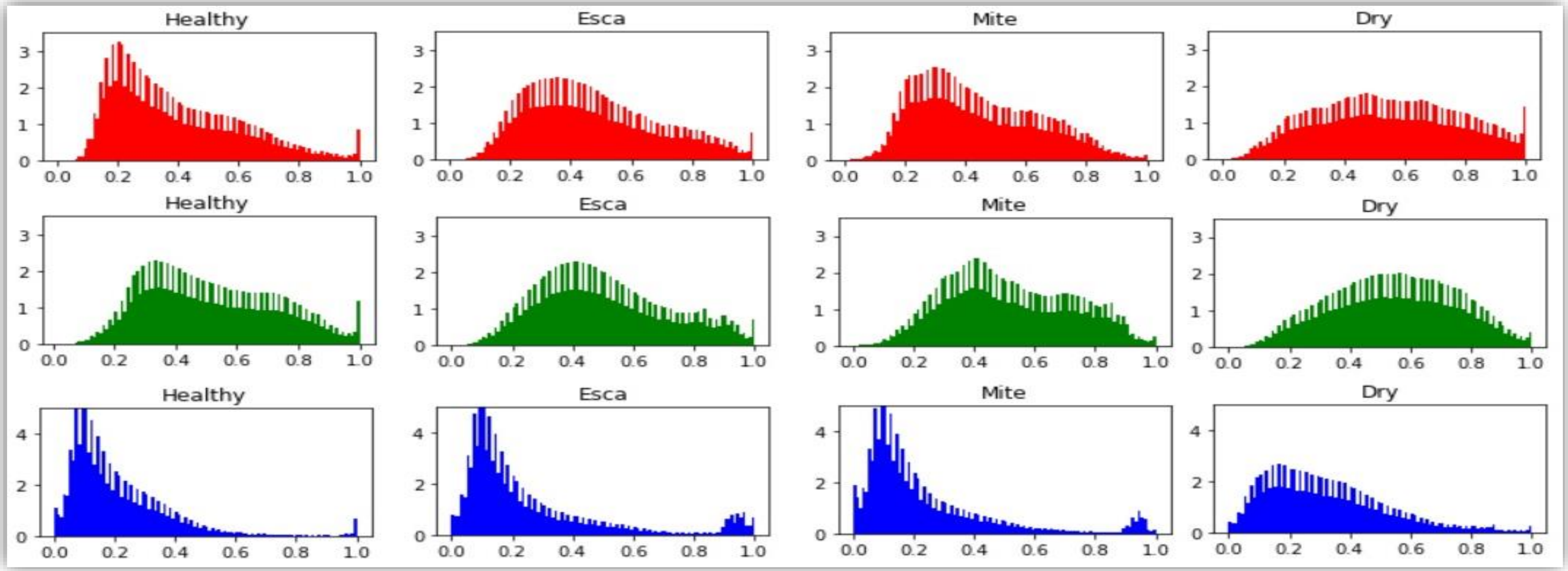


Pixel intensity of color channels of an image



Data exploration

Comparison of pixel color intensities across different classes



Feature extraction

There are different ways of extracting features from picture:

- extracting the **average color** in each of the three channels (RGB)
- extracting **dominant colors** from picture
- extracting features based on **histograms of pixels color intensities**

```
def append_to_df(img_array, df, label):
    for image in img_array:
        hist_b = np.histogram(image[:,:,0], bins=30, density=True, range=(0,1))[0]
        hist_g = np.histogram(image[:,:,1], bins=30, density=True, range=(0,1))[0]
        hist_r = np.histogram(image[:,:,2], bins=30, density=True, range=(0,1))[0]
        hist = np.append(np.append(hist_r, hist_g), hist_b)
        df = df.append({'hist_data': hist, 'label': label}, ignore_index=True)
    return df
```

Feature creation

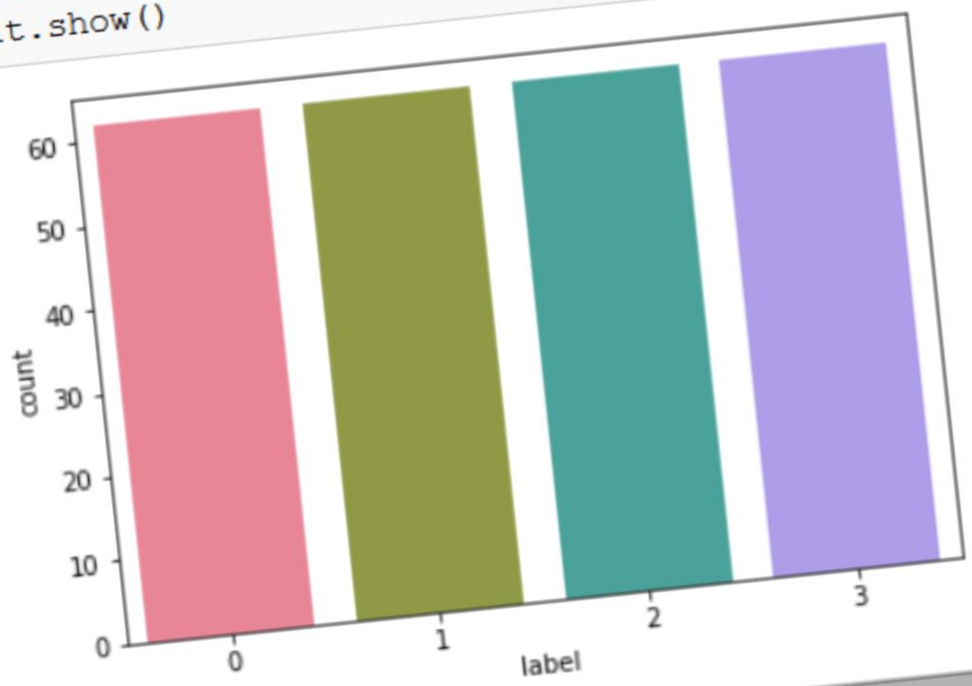
```python
data = pd.DataFrame(columns=['hist_data', 'label'])
data = append_to_df(dry_train, data,0)
data = append_to_df(dry_test, data,0)
data = append_to_df(esca_train, data,1)
data = append_to_df(esca_test, data,1)
data = append_to_df(healthy_train, data,2)
data = append_to_df(healthy_test, data,2)
data = append_to_df(mite_train, data,3)
data = append_to_df(mite_test, data,3)
data
```

```python
data.head(10)
```

	hist_data	label
0	[0.0048065185546875, 0.0128173828125, 0.050582...	0
1	[0.094757080078125, 0.2579498291015625, 0.8288...	0
2	[0.0450897216796875, 0.14041900634765625, 0.52...	0
3	[0.0597381591796875, 0.2088546752929687 5, 0.62...	0
4	[0.0040054321289062 5, 0.040512084960937 5, 0.42...	0
5	[0.791015625, 1.9091033935546875, 3.6917495727...	0
6	[0.0056076049804687 5, 0.0470352172851562 5, 0.4...	0
7	[0.025291442871093 75, 0.0902938842773437 5, 0.5...	0
8	[0.771560668945312 5, 2.153091430664062 5, 3.805...	0
9	[0.21881103515625, 0.4827117919921875, 1.17919...	0

Description of the dataset

```python
# Classes distribution plot
sns.countplot(x='label', data=data, palette='husl')
plt.show()
```




```python
# column names
column_names = data.columns
column_names
```

```
Index(['hist_data', 'label'], dtype='object')
```

```python
# dataset shape
data.shape
```

```
(248, 2)
```

```python
# null values check
data.isnull().sum()
```

```
hist_data     0
label         0
dtype: int64
```

Model selection

Machine learning

KNN

XGBoost

Deep learning

ANN

CNN model

CNN using VGG16

Model creation and training: KNN

Train & test split: 75:25

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
```

```
# Create the KNN classifier
KNN_model = KNeighborsClassifier()
```

```
# Train and evaluate.
KNN_model.fit(X_train, y_train)
```

```
# Select trained model.
y_pred_knn = KNN_model.predict(X_test)
```

```
# Check the accuracy of the trained model.
accuracy = accuracy_score(y_test, y_pred_knn)
print('Accuracy: %.2f%%' % (accuracy * 100.0))
```

Grid search & cross validation

```
# Create KNN pipeline, set up parameter grid.
KNN_model_gs = KNeighborsClassifier()
parameters = {'n_neighbors': [4, 6, 8, 10],
              'weights': ['uniform', 'distance'],
              'metric': ['euclidean', 'manhattan']}
```

```
# Search for the best parameters.
clf = GridSearchCV(estimator = KNN_model_gs,
                   param_grid = parameters,
                   verbose = 1,
                   cv = 5,
                   n_jobs = -1)

clf.fit(X_train, y_train)
```

```
# Display the accuracy of the best parameter combination
y_pred_cv = clf.best_estimator_.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred_cv)
print('Accuracy: %.2f%%' % (accuracy * 100.0))
```

Model creation and training: XGBoost

Train & test split: 75:25

```
# Import packages you need to create the XGBoost model.
from xgboost.sklearn import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
```

```
# Create the XGB classifier - xgb_model.
xgb_model = XGBClassifier(n_estimators=200,
                          objective= 'multi:softmax')
```

```
# Train and evaluate.
xgb_model.fit(X_train, y_train, eval_metric=['mlogloss'],
              eval_set=[((X_train, y_train)),(X_test, y_test)])
```

```
# Select trained model.
n_trees = 85
y_pred = xgb_model.predict(X_test, ntree_limit= n_trees)
```

```
# Check the accuracy of the trained model.
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: %.2f%%' % (accuracy * 100.0))
```

PCA for model tuning

```
# PCA
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components=10)
xgb_model_pca = XGBClassifier(n_estimators=n_trees)
pipeline = Pipeline(steps=[('pca', pca), ('xgb', xgb_model_pca)])
```

```
pipeline.fit(X_train, y_train)
```

```
y_pred = pipeline.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: %.2f%%' % (accuracy * 100.0))
```

Model creation and training: ANN

Train & test split: 75:25

```python
ANN_model = Sequential()
```

```python
# Adding the input layer and the first hidden layer
ANN_model.add(Dense(output_dim = 45,
                    init = 'uniform',
                    activation = 'relu',
                    input_dim = 90))
```

```python
# Adding the second hidden layer
ANN_model.add(Dense(output_dim = 45,
                    init = 'uniform',
                    activation = 'relu'))
```

```python
# Adding the output layer
ANN_model.add(Dense(output_dim = 4,
                     init = 'uniform',
                     activation = 'softmax'))
```

```python
# Compiling the ANN
ANN_model.compile(optimizer = 'adam',
                  loss = 'categorical_crossentropy',
                  metrics = ['accuracy'])
```

```python
# Fitting the ANN to the Training set
ANN_model.fit(X_train, y_train, batch_size = 10,
              nb_epoch = 30,
              validation_data=(X_test, y_test),
              callbacks=[plot_learning])
```

```python
score_test = ANN_model.evaluate(X_test, y_test, verbose=0)
```

```python
print("Loss = ", score_test[0], ", Accuracy = ", score_test[1])
```

```python
#predictions_test
predictions_test = ANN_model.predict(X_test)
```

```python
y_pred_test =[]
for i in range(len(predictions_test)):
    y_pred_test.append(np.argmax(predictions_test[i]))
print(y_pred_test)
```

```python
y_test_labels = [ np.where(r==1)[0][0] for r in y_test ]
y_test_labels
```

```python
ctr_test=0
for i in range(len(y_pred_test)):
    if y_pred_test[i] == y_test_labels[i]:
        ctr_test=ctr_test+1
res_test = ctr_test/len(y_pred_test)*100
print(res_test)
```

Model creation and training: ANN cross validation

Cross validation

```
from sklearn.model_selection import StratifiedKFold
```

```
seed = 7
np.random.seed(seed)
```

```
kfold = StratifiedKFold(n_splits=10, shuffle=True,
                        random_state=seed)
```

```
for train, test in kfold.split(X, y):
    cv_model = Sequential()
    cv_model.add(Dense(output_dim = 45,
                       init = 'uniform',
                       activation = 'relu',
                       input_dim = 90))
    cv_model.add(Dense(output_dim = 45,
                       init = 'uniform',
                       activation = 'relu'))
    cv_model.add(Dense(output_dim = 4,
                       init = 'uniform',
                       activation = 'softmax'))

    # Compiling the ANN model
    cv_model.compile(optimizer = 'adam',
                     loss = 'categorical_crossentropy',
                     metrics = ['accuracy'])

    # Fitting the ANN model
    cv_model.fit(X_train, y_train, batch_size = 10,
                 nb_epoch = 30, verbose=1,
                 validation_data=(X_test, y_test))
```

```
score_test_cv = cv_model.evaluate(X_test, y_test, verbose=0)
print("Loss = ", score_test_cv[0], ", Accuracy = ", score_test_cv[1])
```

Model creation and training: CNN VGG16

Train & test split

```
vgg16_model = keras.applications.vgg16.VGG16()
vgg16_model.summary()
```

```
#This is a Keras Functional API need to convert to sequential
type(vgg16_model)
#Iterate over the functional layers and add it as a stack
model = Sequential()
for layer in vgg16_model.layers[:-1]:
    model.add(layer)
```

```
#Since the model is already trained with certain weights,
# we dont want to change it. Let it be the same
for layer in model.layers:
    layer.trainable = False
```

```
# Complie the model
model.compile(Adam(lr=.00015),
              loss='categorical_crossentropy', metrics=['accuracy'])
```

```
model.fit_generator(train_batches, steps_per_epoch=20,
                    validation_data=test_batches,
                    validation_steps=12, epochs=20,
                    verbose=1, shuffle=False,
                    callbacks=[plot_learning])
```

```
scores_test = model.evaluate_generator(generator = test_batches_pred, steps=1)
print("Loss = ", scores_test[0], ", Accuracy = ", scores_test[1])
```

```
predictions_test = model.predict_generator(test_batches_pred, steps=1, verbose=0)
predictions_test
```

```
y_pred_test =[]
for i in range(len(predictions_test)):
    y_pred_test.append(np.argmax(predictions_test[i]))
print(y_pred_test)
```

```
ctr_test=0
for i in range(len(y_pred_test)):
    if y_pred_test[i] == y_test[i]:
        ctr_test=ctr_test+1
res_test = ctr_test/len(y_pred_test)*100
print(res_test)
```

Model creation and training: CNN Sequential

Train & test split

```python
# Initialising the CNN
model = Sequential()
```

```python
# Step 1 - Convolution
model.add(Conv2D(filters = 32, kernel_size=(3,3),
                 input_shape = input_shape,
                 activation = 'relu'))
```

```python
# Step 2 - Pooling
model.add(MaxPooling2D(pool_size = (2, 2)))
```

```python
# Adding a second convolution layer
model.add(Conv2D(filters = 64, kernel_size=(3,3),
                 activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Dropout(rate=0.20))
```

```python
# Adding a third convolution layer
model.add(Conv2D(filters = 128, kernel_size=(3,3),
                 activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Dropout(rate=0.20))
```

```python
# Step 3 - Flattening
model.add(Flatten())
```

```python
# Step 4 - Full connection
model.add(Dense(units = 128, activation = 'relu'))
model.add(BatchNormalization())
model.add(Dropout(rate=0.5))
```

```python
# Step 5 - Final layer
model.add(Dense(units = num_classes,
                activation = 'softmax'))
```

```python
model.fit_generator(train_batches, steps_per_epoch=7,
                    validation_data=test_batches,
                    validation_steps = 48,
                    epochs=200, verbose=1, shuffle=False,
                    callbacks=[plot_learning])
```

```python
scores_test = model.evaluate_generator(generator = test_batches_pred, steps=1)
print("Loss = ", scores_test[0], ", Accuracy = ", scores_test[1])
```

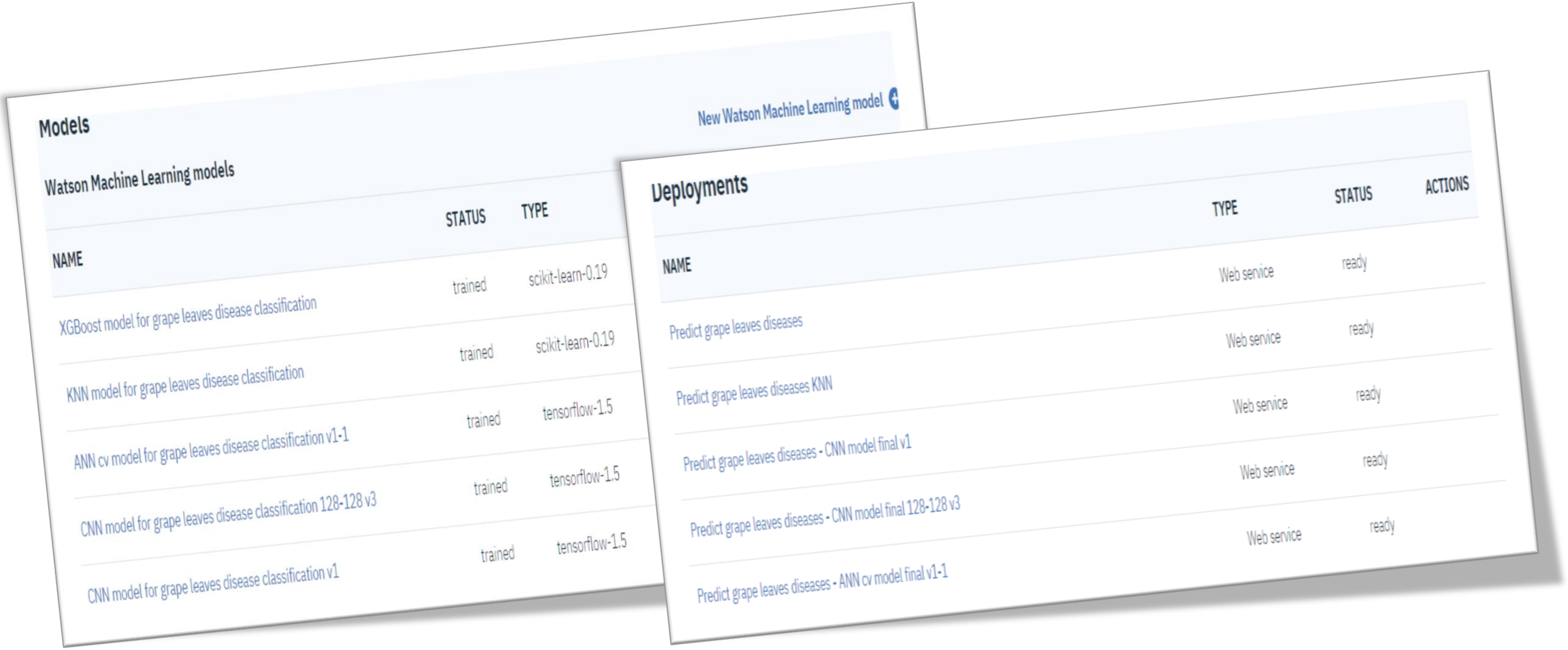
Model evaluation: ML models

Model name	Training method	Accuracy
KNN	Train & test set split	80.65%
	Grid search & cross validation	83.87%
XGBoost	Train & test set split	85.48%
	PCA for model tuning	88.71%

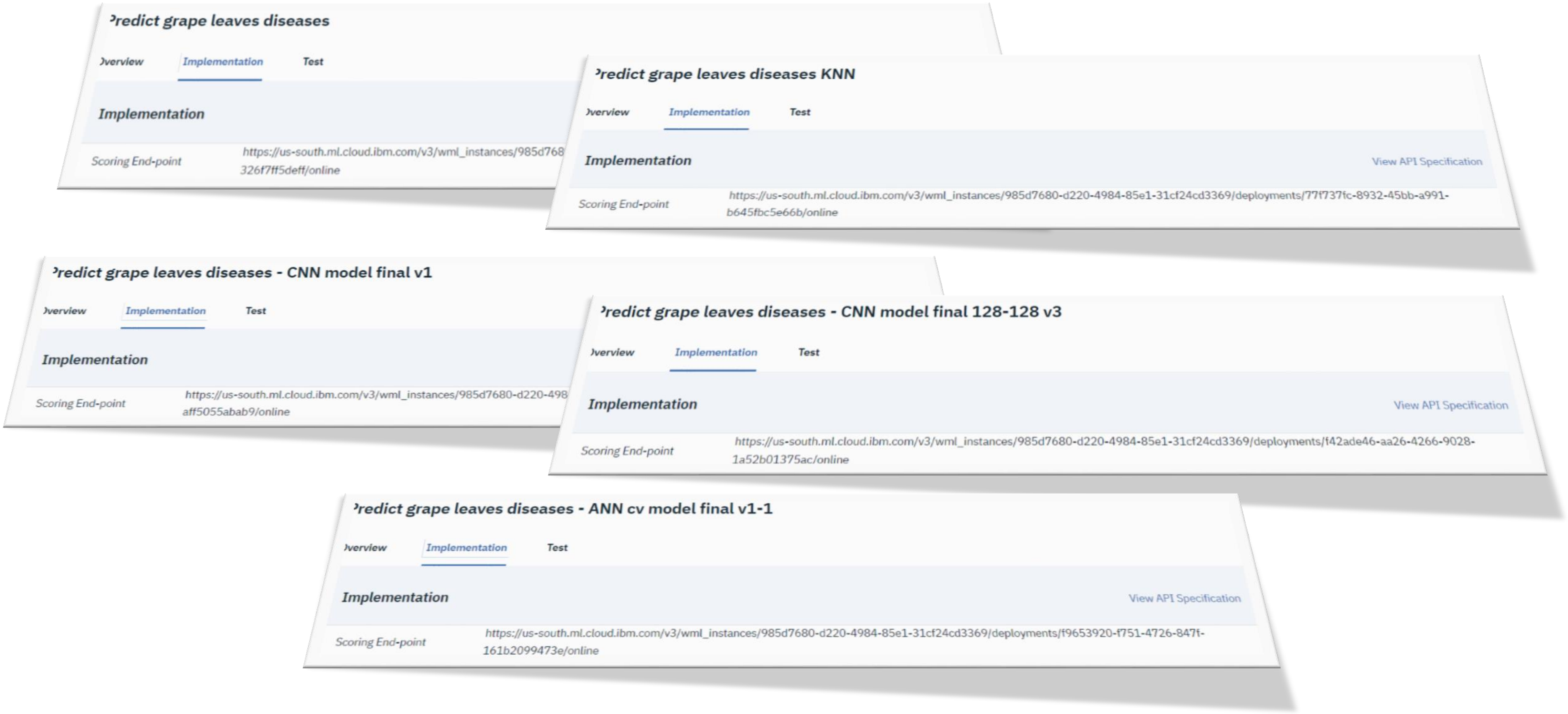
Model evaluation: DL models

Model name	Training method	Accuracy
ANN	Train & test set split	83.87%
	Cross validation	83.87%
CNN using VGG16	Train & test set split	79.16%
CNN model	Train & test set split	77.08%

List of deployed models in WML repository



Models REST APIs



Get the score using the model's REST API

```
img_file_name = "datasetWS_DvsEvsHvsM_512-512_ext_set/esca.IMG_7508.jpg"

import numpy as np
from PIL import Image

img = Image.open(img_file_name)
img = img.resize((224, 224))
img = np.array(img)
img = img.astype('float32') / 255.0
img = img.reshape((1,224, 224,3))
type(img)
```

```
img_to_list = img.tolist()
img_to_list
```

```
scoring_payload = {"values": img_to_list}
#print(scoring_payload)
```

```
model_deployment_endpoint_url = 'https://us-south.ml.cloud.ibm.com/v3/wml_instances/985d7680-d220-4984-85e1-31cf24cd3369/deployments/c3374942-d0f1-477b-9cde-aff5055abab9/online'
```

```
parms = { "wml_credentials" : wml_credentials, "model_deployment_endpoint_url" : model_deployment_endpoint_url }
```

```
from watson_machine_learning_client import WatsonMachineLearningAPIClient
client       = WatsonMachineLearningAPIClient( parms["wml_credentials"] )
model_result = client.deployments.score(model_deployment_endpoint_url, scoring_payload )
```

```
model_result
```


```
{'fields': ['prediction', 'prediction_classes', 'probability'],
 'values': [[[0.16954658925533295,
    0.44687438011169434,
    0.203651562333107,
    0.1799274981021881],
   1,
   [0.16954658925533295,
    0.44687438011169434,
    0.203651562333107,
    0.1799274981021881]]]}
```


```
img_class = model_result["values"][0][1]
img_class
```

```
1
```


THANK YOU