Cours 5: PMC avec Keras

Author: Ricardo Vallejo

```
L'objectif de ces exercices est de pratiquer le perceptron simple et le
perceptron multicouche en utilisant la
bibliothèque Keras.
Soit l'ensemble des données Segmentation (disponible aussi sur Lea) qui
comprend 2310 observations d'images décrites par 19 variables
(caractéristiques) et leurs variable cibles (classe).
```

1. Téléchargez le contenu de la base de données.

In [409]:

```python
import pandas as pd
import matplotlib.pyplot as plt
import statistics
import numpy as np
import scipy.stats
import seaborn as sns

# 1. Téléchargez le contenu de la base de données iris

data = pd.read_excel("segmentation.xlsx")
pd.set_option('display.max_rows', None)
data.head(5)
```

Out[409]:

	region_centroid_col	region_centroid_row	region_pixel_count	short_line_density_5	short_lin
0	218	178	9	0.11	
1	113	130	9	0.00	
2	202	41	9	0.00	
3	32	173	9	0.00	
4	61	197	9	0.00	

In [410]:

```python
dataX = data.loc[:, data.columns != 'classe']
dataY = data[['classe']]
```

In [411]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2310 entries, 0 to 2309
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   region_centroid_col   2310 non-null   int64  
 1   region_centroid_row   2310 non-null   int64  
 2   region_pixel_count    2310 non-null   int64  
 3   short_line_density_5  2310 non-null   float64
 4   short_line_density_2  2310 non-null   float64
 5   vedge_mean            2310 non-null   float64
 6   vegde_sd              2310 non-null   float64
 7   hedge_mean            2310 non-null   float64
 8   hedge_sd              2310 non-null   float64
 9   intensity_mean        2310 non-null   float64
 10  rawred_mean           2310 non-null   float64
 11  rawblue_mean          2310 non-null   float64
 12  rawgreen_mean         2310 non-null   float64
 13  exred_mean            2310 non-null   float64
 14  exblue_mean           2310 non-null   float64
 15  exgreen_mean          2310 non-null   float64
 16  value_mean            2310 non-null   float64
 17  saturation_mean       2310 non-null   float64
 18  hue_mean              2310 non-null   float64
 19  classe                2310 non-null   object 
dtypes: float64(16), int64(3), object(1)
memory usage: 361.1+ KB
```

2. Procédez à une standardisation des données

In [412]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
datax_std = scaler.fit_transform(dataX)
dfx_std =  pd.DataFrame(datax_std, columns = [dataX.columns])
dfx_std.head(5)
```

Out[412]:

	region_centroid_col	region_centroid_row	region_pixel_count	short_line_density_5	short_lin
0	1.276189	0.949736	0.0	2.410668	
1	-0.163336	0.114538	0.0	-0.357047	
2	1.056833	-1.434058	0.0	-0.357047	
3	-1.273827	0.862737	0.0	-0.357047	
4	-0.876244	1.280336	0.0	-0.357047	

3. Déterminez les différentes classes

In [413]:
```
data.groupby('classe').size()
```

Out[413]:
```
classe
brickface    330
cement       330
foliage      330
grass        330
path         330
sky          330
window       330
dtype: int64
```

4. Considérez une partition de 70% pour l’entrainement.

In [414]:
```
df_std_tot = pd.concat([dfx_std, dataY], axis = 1)
df_std_tot =  df_std_tot.set_axis([data.columns], axis=1, inplace=False) #pd.
df_std_tot.head(5)
```

Out[414]:

	region_centroid_col	region_centroid_row	region_pixel_count	short_line_density_5	short_lin
0	1.276189	0.949736	0.0	2.410668	
1	-0.163336	0.114538	0.0	-0.357047	
2	1.056833	-1.434058	0.0	-0.357047	
3	-1.273827	0.862737	0.0	-0.357047	
4	-0.876244	1.280336	0.0	-0.357047	

In [415]:
```
from sklearn.model_selection import train_test_split

train, test = train_test_split(df_std_tot, test_size = 0.3, stratify = df_std
```

5. Vérifiez la taille de l’échantillon d’entrainement et de test par classe.

In [416]:
```
train.reset_index(drop = True, inplace = True)
test.reset_index(drop = True, inplace = True)
```

In [418]:
```
train.head(4)
```

Out[418]:

	region_centroid_col	region_centroid_row	region_pixel_count	short_line_density_5	short_lin
0	0.330215	1.019336	0.0	2.410668	
1	0.179408	0.166738	0.0	-0.357047	
2	1.303608	-0.059461	0.0	-0.357047	
3	1.262479	-0.755459	0.0	-0.357047	

In [363]:
```
# train.groupby('classe').size()
# test.groupby('classe').size()
# TODO
```

6. Développez un perceptron simple et une architecture séquentielle (activation='softmax', optimizer='adam')

6.1. Data Model Train Encoding

In [419]:
```
from sklearn.preprocessing import OneHotEncoder

#encodage de classes
encoder =OneHotEncoder()
encodedTrainTarget = encoder.fit_transform(train[["classe"]])
labelsTrain=pd.DataFrame(encodedTrainTarget.toarray(), columns=encoder.catego
labelsTrain.head(5)
```

Out[419]:

	brickface	cement	foliage	grass	path	sky	window
0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	1.0
4	0.0	0.0	0.0	0.0	0.0	0.0	1.0

In [420]:
```
# Concatenation
df_train_tot=pd.concat([train,labelsTrain],  axis=1)
df_train_tot.head(5)
```

Out[420]:

	region_centroid_col	region_centroid_row	region_pixel_count	short_line_density_5	short_lin
0	0.330215	1.019336	0.0	2.410668	
1	0.179408	0.166738	0.0	-0.357047	
2	1.303608	-0.059461	0.0	-0.357047	
3	1.262479	-0.755459	0.0	-0.357047	
4	1.111672	-1.086059	0.0	-0.357047	

5 rows × 27 columns

6.2. Data Model Test Encoding

In [421]:
```
#encodage de classes
encoder =OneHotEncoder()
encodedTestTarget = encoder.fit_transform(test[["classe"]])
encodedTestTarget

labelsTest=pd.DataFrame(encodedTestTarget.toarray(), columns=encoder.categori
labelsTest.head(5)
```

Out[421]:

	brickface	cement	foliage	grass	path	sky	window
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	0.0	0.0	0.0	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	1.0	0.0	0.0	0.0

In [422]:

```
# Concatenation
df_test_tot=pd.concat([test,labelsTest],axis=1)
df_test_tot.head(5)
```

Out[422]:

	region_centroid_col	region_centroid_row	region_pixel_count	short_line_density_5	short_lin
0	-0.231885	-0.216061	0.0	-0.357047	
1	0.576991	-1.155658	0.0	-0.357047	
2	0.343925	0.497337	0.0	-0.357047	
3	-1.150439	-0.529260	0.0	-0.357047	
4	1.509255	2.185133	0.0	-0.357047	

5 rows × 27 columns

In [423]:

```
df_test_tot.drop(columns=['classe'],inplace=True)
df_test_tot.head(5)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py:4153: Per
formanceWarning: dropping on a non-lexsorted multi-index without a level pa
rameter may impact performance.
  obj = obj._drop_axis(labels, axis, level=level, errors=errors)
```

Out[423]:

	region_centroid_col	region_centroid_row	region_pixel_count	short_line_density_5	short_lin
0	-0.231885	-0.216061	0.0	-0.357047	
1	0.576991	-1.155658	0.0	-0.357047	
2	0.343925	0.497337	0.0	-0.357047	
3	-1.150439	-0.529260	0.0	-0.357047	
4	1.509255	2.185133	0.0	-0.357047	

5 rows × 26 columns

In [424]:
```
df_train_tot.drop(columns=['classe'],inplace=True)
df_train_tot.head(5)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py:4153: Per
formanceWarning: dropping on a non-lexsorted multi-index without a level pa
rameter may impact performance.
  obj = obj._drop_axis(labels, axis, level=level, errors=errors)
```

Out[424]:

	region_centroid_col	region_centroid_row	region_pixel_count	short_line_density_5	short_lin
0	0.330215	1.019336	0.0	2.410668	
1	0.179408	0.166738	0.0	-0.357047	
2	1.303608	-0.059461	0.0	-0.357047	
3	1.262479	-0.755459	0.0	-0.357047	
4	1.111672	-1.086059	0.0	-0.357047	

5 rows × 26 columns

PMC

In [425]:
```
X_train=df_train_tot[['region_centroid_col', 'region_centroid_row', 'region_p
       'short_line_density_5', 'short_line_density_2', 'vedge_mean',
       'vegde_sd', 'hedge_mean', 'hedge_sd', 'intensity_mean', 'rawred_mean',
       'rawblue_mean', 'rawgreen_mean', 'exred_mean', 'exblue_mean',
       'exgreen_mean', 'value_mean', 'saturation_mean', 'hue_mean']]

y_train=df_train_tot[['brickface', 'cement', 'foliage',  'grass', 'path', 'sk

X_test=df_test_tot[['region_centroid_col', 'region_centroid_row', 'region_pix
       'short_line_density_5', 'short_line_density_2', 'vedge_mean',
       'vegde_sd', 'hedge_mean', 'hedge_sd', 'intensity_mean', 'rawred_mean',
       'rawblue_mean', 'rawgreen_mean', 'exred_mean', 'exblue_mean',
       'exgreen_mean', 'value_mean', 'saturation_mean', 'hue_mean']]

y_test=df_test_tot[['brickface', 'cement', 'foliage',  'grass', 'path', 'sky'
```

In [426]:
```
X_train.shape
```

Out[426]: (1617, 19)

In [427]:
```
y_train.shape
```

Out[427]: (1617, 7)

In [428]:
```
encoder.categories_
```

Out[428]:
```
[array(['brickface', 'cement', 'foliage', 'grass', 'path', 'sky', 'windo
w'],
       dtype=object)]
```

In [430]:
```
X_test.shape
```

Out[430]:
```
(693, 19)
```

In [431]:
```
y_test.shape
```

Out[431]:
```
(693, 7)
```

In [433]:

```
# architecture séquentielle (activation='softmax', optimizer='adam')

from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential


PMC = Sequential( )
PMC.add(layers.Dense( units=y_train.shape[1] , activation='softmax' ) )
PMC.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accu
print('Number of layers :',len(PMC.layers))


# fit model
results = PMC.fit(X_train, y_train,validation_data=(X_test, y_test),epochs=30
```

```
Number of layers : 1
Epoch 1/30
51/51 [==============================] - 2s 18ms/step - loss: 1.9247 - ac
curacy: 0.2421 - val_loss: 1.6152 - val_accuracy: 0.4430
Epoch 2/30
51/51 [==============================] - 0s 7ms/step - loss: 1.5357 - acc
uracy: 0.4986 - val_loss: 1.4007 - val_accuracy: 0.5483
Epoch 3/30
51/51 [==============================] - 0s 3ms/step - loss: 1.3815 - acc
uracy: 0.5677 - val_loss: 1.2551 - val_accuracy: 0.5671
Epoch 4/30
51/51 [==============================] - 0s 3ms/step - loss: 1.2422 - acc
uracy: 0.6002 - val_loss: 1.1441 - val_accuracy: 0.6046
Epoch 5/30
51/51 [==============================] - 0s 3ms/step - loss: 1.1080 - acc
uracy: 0.6241 - val_loss: 1.0549 - val_accuracy: 0.6421
Epoch 6/30
51/51 [==============================] - 0s 2ms/step - loss: 1.0252 - acc
uracy: 0.6501 - val_loss: 0.9796 - val_accuracy: 0.6667
Epoch 7/30
51/51 [==============================] - 0s 3ms/step - loss: 0.9675 - acc
uracy: 0.6595 - val_loss: 0.9161 - val_accuracy: 0.6854
Epoch 8/30
51/51 [==============================] - 0s 2ms/step - loss: 0.9029 - acc
uracy: 0.7095 - val_loss: 0.8620 - val_accuracy: 0.7431
Epoch 9/30
51/51 [==============================] - 0s 2ms/step - loss: 0.8583 - acc
uracy: 0.7498 - val_loss: 0.8147 - val_accuracy: 0.7691
Epoch 10/30
51/51 [==============================] - 0s 4ms/step - loss: 0.8147 - acc
uracy: 0.7746 - val_loss: 0.7736 - val_accuracy: 0.7835
Epoch 11/30
51/51 [==============================] - 0s 2ms/step - loss: 0.7701 - acc
uracy: 0.7809 - val_loss: 0.7376 - val_accuracy: 0.8009
Epoch 12/30
51/51 [==============================] - 0s 2ms/step - loss: 0.7422 - acc
uracy: 0.7951 - val_loss: 0.7057 - val_accuracy: 0.8153
Epoch 13/30
51/51 [==============================] - 0s 4ms/step - loss: 0.7002 - acc
uracy: 0.8084 - val_loss: 0.6773 - val_accuracy: 0.8182
```

```
Epoch 14/30
51/51 [==============================] - 0s 2ms/step - loss: 0.6669 - acc
uracy: 0.8317 - val_loss: 0.6517 - val_accuracy: 0.8240
Epoch 15/30
51/51 [==============================] - 0s 3ms/step - loss: 0.6552 - acc
uracy: 0.8249 - val_loss: 0.6288 - val_accuracy: 0.8312
Epoch 16/30
51/51 [==============================] - 0s 3ms/step - loss: 0.6485 - acc
uracy: 0.8228 - val_loss: 0.6083 - val_accuracy: 0.8384
Epoch 17/30
51/51 [==============================] - 0s 3ms/step - loss: 0.5953 - acc
uracy: 0.8540 - val_loss: 0.5895 - val_accuracy: 0.8499
Epoch 18/30
51/51 [==============================] - 0s 3ms/step - loss: 0.5814 - acc
uracy: 0.8490 - val_loss: 0.5726 - val_accuracy: 0.8499
Epoch 19/30
51/51 [==============================] - 0s 3ms/step - loss: 0.5781 - acc
uracy: 0.8413 - val_loss: 0.5567 - val_accuracy: 0.8528
Epoch 20/30
51/51 [==============================] - 0s 3ms/step - loss: 0.5350 - acc
uracy: 0.8730 - val_loss: 0.5424 - val_accuracy: 0.8557
Epoch 21/30
51/51 [==============================] - 0s 2ms/step - loss: 0.5318 - acc
uracy: 0.8682 - val_loss: 0.5293 - val_accuracy: 0.8600
Epoch 22/30
51/51 [==============================] - 0s 3ms/step - loss: 0.5327 - acc
uracy: 0.8644 - val_loss: 0.5174 - val_accuracy: 0.8600
Epoch 23/30
51/51 [==============================] - 0s 3ms/step - loss: 0.5092 - acc
uracy: 0.8598 - val_loss: 0.5062 - val_accuracy: 0.8615
Epoch 24/30
51/51 [==============================] - 0s 3ms/step - loss: 0.4985 - acc
uracy: 0.8727 - val_loss: 0.4957 - val_accuracy: 0.8658
Epoch 25/30
51/51 [==============================] - 0s 3ms/step - loss: 0.4905 - acc
uracy: 0.8721 - val_loss: 0.4863 - val_accuracy: 0.8672
Epoch 26/30
51/51 [==============================] - 0s 3ms/step - loss: 0.4828 - acc
uracy: 0.8568 - val_loss: 0.4770 - val_accuracy: 0.8730
Epoch 27/30
51/51 [==============================] - 0s 3ms/step - loss: 0.4644 - acc
uracy: 0.8668 - val_loss: 0.4687 - val_accuracy: 0.8759
Epoch 28/30
51/51 [==============================] - 0s 3ms/step - loss: 0.4542 - acc
uracy: 0.8837 - val_loss: 0.4606 - val_accuracy: 0.8745
Epoch 29/30
51/51 [==============================] - 0s 3ms/step - loss: 0.4434 - acc
uracy: 0.8806 - val_loss: 0.4529 - val_accuracy: 0.8730
Epoch 30/30
51/51 [==============================] - 0s 3ms/step - loss: 0.4521 - acc
uracy: 0.8773 - val_loss: 0.4460 - val_accuracy: 0.8730
```

In [ ]:

In [435]:

```
prediction = PMC.predict(X_test)
```

In [ ]:

```
```

In [436]:

```
# Applying the function on input class vector
from tensorflow.keras.utils import to_categorical


classes_x = np.argmax(PMC.predict(X_test), axis=-1)
y_pred = to_categorical(classes_x, num_classes = 7, dtype ="int32")

print(y_pred)
```

```
[[1 0 0 ... 0 0 0]
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 0 0 1]
 ...
 [0 1 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 [0 0 0 ... 0 0 1]]
```

In [ ]:

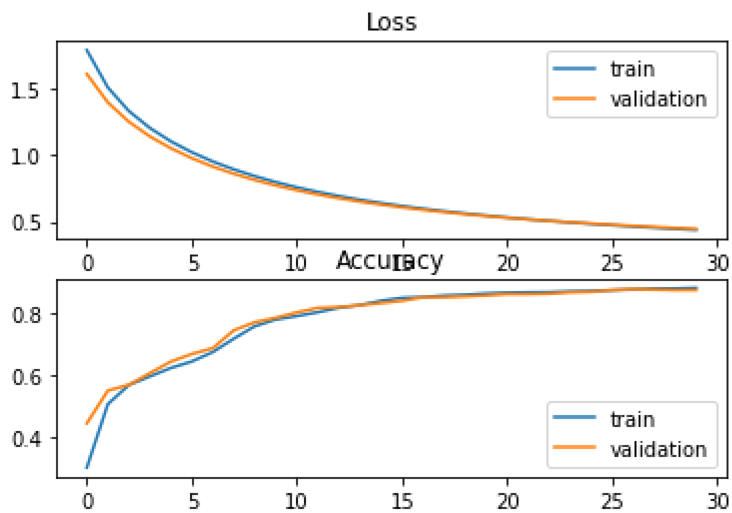
```
ypred=encoder.inverse_transform(y_pred)
ytest=encoder.inverse_transform(y_test)
```

In [396]:

```
```

In [438]:

```
# plot loss during training
plt.subplot(211)
plt.title('Loss')
plt.plot(results.history['loss'], label='train')
plt.plot(results.history['val_loss'], label='validation')
plt.legend()
# plot accuracy during training
plt.subplot(212)
plt.title('Accuracy')
plt.plot(results.history['accuracy'], label='train')
plt.plot(results.history['val_accuracy'], label='validation')
plt.legend()
plt.show()
```



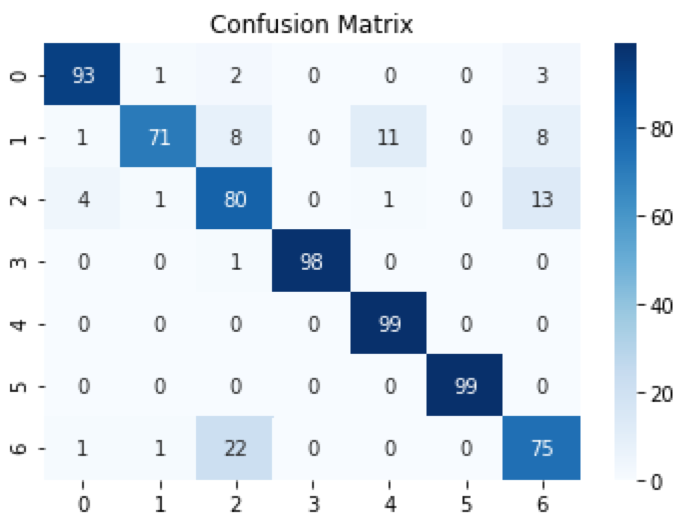
7. Représentez la matrice de confusion et évaluez les performances en utilisant classification_report..

In [440]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classif

cm = confusion_matrix(ytest, ypred)

# Plot confusion matrix
import seaborn as sns
import pandas as pd
# confusion matrix sns heatmap
ax = plt.axes()
df_cm = cm
sns.heatmap(df_cm, annot=True, annot_kws={"size": 10}, fmt='d',cmap="Blues",
ax.set_title('Confusion Matrix')
plt.show()
```



In [441]:

```
print(classification_report(ytest, ypred))
```

```
              precision    recall  f1-score   support

   brickface       0.94      0.94      0.94        99
      cement       0.96      0.72      0.82        99
     foliage       0.71      0.81      0.75        99
       grass       1.00      0.99      0.99        99
        path       0.89      1.00      0.94        99
         sky       1.00      1.00      1.00        99
      window       0.76      0.76      0.76        99

    accuracy                           0.89       693
   macro avg       0.89      0.89      0.89       693
weighted avg       0.89      0.89      0.89       693
```

9. Développez un perceptron multicouche (2 couches cachées à 30 neurones, fonctions d’activation ReLu)

In [442]:

```
def create_model(optimizer='adam',activation='relu'):

    model = Sequential()
    model.add(layers.Dense( units=30,input_dim=X_train.shape[1] , activation=
    model.add(layers.Dense(30, activation=activation ) )
    model.add(layers.Dense( units=y_train.shape[1] , activation='softmax') )

    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metri
    return model

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)
```

In [443]:

```
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier

# create model
model = KerasClassifier(build_fn=create_model, epochs=30, batch_size=10, verb


# define the grid search parameters
#on va tester les fonctions d'activation et les optimizateurs
optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nada
activation = ['softmax', 'relu','sigmoid','softplus','softsign','tanh','selu'

param_grid = dict(optimizer=optimizer,activation=activation)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X_train, y_train)
grid_result
```

Out[443]:

```
GridSearchCV(cv=3,
             estimator=<keras.wrappers.scikit_learn.KerasClassifier object
at 0x000001B51FA49340>,
             n_jobs=-1,
             param_grid={'activation': ['softmax', 'relu', 'sigmoid',
                                        'softplus', 'softsign', 'tanh', 'se
lu',
                                        'elu'],
                         'optimizer': ['SGD', 'RMSprop', 'Adagrad', 'Adadel
ta',
                                       'Adam', 'Adamax', 'Nadam']})
```

In [ ]:

```


# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

10. . Représentez la matrice de confusion et évaluez les performances en utilisant classification_report.

In [408]:

```
matrix = confusion_matrix(ytest, ypred)
print(matrix)
print(classification_report(ytest, ypred))
```

```
[[93  1  2  0  0  0  3]
 [ 1 71  8  0 11  0  8]
 [ 4  1 80  0  1  0 13]
 [ 0  0  1 98  0  0  0]
 [ 0  0  0  0 99  0  0]
 [ 0  0  0  0  0 99  0]
 [ 1  1 22  0  0  0 75]]
              precision    recall  f1-score   support

   brickface       0.94      0.94      0.94        99
      cement       0.96      0.72      0.82        99
     foliage       0.71      0.81      0.75        99
       grass       1.00      0.99      0.99        99
        path       0.89      1.00      0.94        99
         sky       1.00      1.00      1.00        99
      window       0.76      0.76      0.76        99

    accuracy                           0.89       693
   macro avg       0.89      0.89      0.89       693
weighted avg       0.89      0.89      0.89       693
```

In [ ]: