README:

Programmierabgabe 1 - Gruppe 5

Klassifikation von Hunderassen mit Scikit Learn

Verwendung eine SVM mit Radial Basis Function Kernel.

von: Thomas Alpert und Lucas Späth

geschätzter Arbeitsaufwand: 20 Stunden (Blut, Schweiß und Tränen)

Ausführbares Jupiter Notebook : hunde_Aufgabe1.ipynb

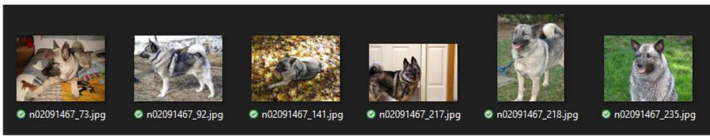
Zip-File einfach auspacken. Datei und Ordner Struktur:

```
.\annotation\Annotation
.\annotation\Annotation\n02091467-Norwegian_elkhound\annotionfiles
.\annotation\Annotation\n02095889-Sealyham_terrier\annotionfiles
.\annotation\Annotation\n02100583-vizsla\annotionfiles
.\annotation\Annotation\n02102973-Irish_water_spaniel\annotionfiles
.\annotation\Annotation\n02105056-groenendael\annotionfiles
.\images\Images
.\images\Images\n02091467-Norwegian_elkhound\images
.\images\Images\n02095889-Sealyham_terrier\images
.\images\Images\n02100583-vizsla\images
.\images\Images\n02102973-Irish_water_spaniel\images
.\images\Images\n02105056-groenendael\images
.\hunde_Aufgabe1.ipynb
.\hunde_Aufgabe2.ipynb
.\README.pdf
```

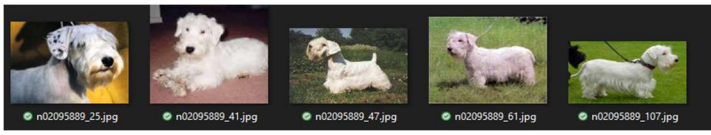
Auswahl der 5 Hunderassen:

Bei der Auswahl der Hunderassen wurde Wert draufgelegt, dass diese sich leicht "auf den ersten Blick" unterscheiden lassen können. Ein geeignetes Merkmal, um eine Hunderasse leicht zu unterscheiden wäre unter anderem die Fellfarbe. Daher wurden für die Klassifikation 5 Hunderasse mit unterschiedlicher Fellfarbe ausgewählt.

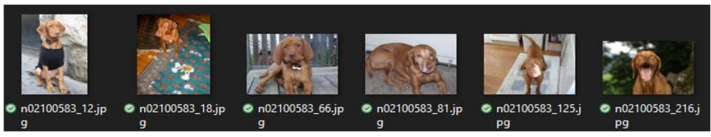
n02091467-Norwegian_elkhound → Fellfarbe grau



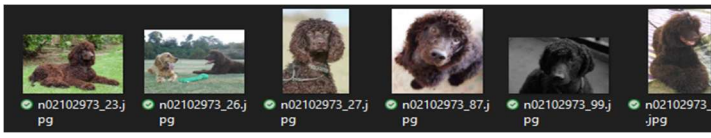
n02095889-Sealyham_terrier →Fellfarbe weiß



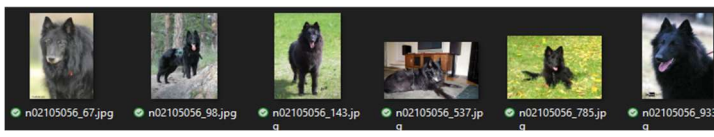
n02100583-vizsla →Fellfarbe orange-braun



n02102973-Irish_water_spaniel →Fellfarbe dunkel-braun



n02105056-groenendael → Fellfarbe schwarz



Merkmalsextraktion:

Zur Klassifizierung der Hunderassen müssen aus diesen (Bildern) Merkmale extrahiert werden. Dabei wurde sich dafür entschieden den **durchschnittlichen Farbwert (HUE)** und die **durchschnittliche Sättigung (Saturation)** der vorhandenen Hundebilder zu entnehmen. Anhand dieser Merkmale sollte es - auf Grund der Vorauswahl der Unterscheidbarkeit der Fellfarbe der Hunderassen - kein Problem sein für die SVM die Hunderassen ansprechend zu klassifizieren.

Merkmalsreduktion:

Der Mittelwert des Farbwerts und der Sättigung eines Bilder wird aus allen Pixeln im Bild generiert. Daher ist es wichtig, dass auf den Bildern möglichst viel "vom Hund" zu sehen ist. Alle weiteren Informationen wie der Hintergrund und alles, was nicht mit dem Hund zu tun hat, muss/sollte aus den Bildern entfernet werden. Da diese sonst die Mittelwerke verfälschen können.

Zu jeden Hundebild in diesem Datensatz gibt es eine entsprechende *Annotation* Datei. Diese *Annotation* Datei enthält Metadaten zu den Bildern. Unter anderem ist in jeder *Annotation*-Datei hinterlegt wo sich genau der Hund befindet. Diese Informationen sind als „Pixelquadrat“ vorhanden.

Die gesamten Hundebilder werden daher mithilfe der Informationen aus der jeweiligen *Annotation* Datei zuerst zurechtgeschnitten und in ein separaten Ordner gespeichert:

Beispiel zuschneiden der Bilder.

Links = Originalbild mit markiertem Pixelquadrat, in dem sich der Hund befindet, aus der *Annotation*-Datei

Rechts= zurechtgeschnittenes Bild

Vorbereitung

Nach dem alle Bilder zurechtgeschnitten werden, werde diese mit dem ***cv2-modul*** in ein Arrays gespeichert. In diesem 3-Dimensionalen Array stehen dann die Informationen der Bilder in Form von RGB-Werten .

Die Bildinformationen oder Bilder werden anschließende mit numpy ge“reshaped“ und auf eine einheitliche Bildgröße skaliert.

Anschließend kann mit numpy.mean der HUE und Saturation Wert als Feature Array entnommen werden.

Einteilung der Daten:

Die Daten (Features) werden vor der Einteilung in Train und Test Daten zunächst einmal zufällig sortiert. Das ist notwendig um unterschiedliche Ergebnisse zubekomme, da wir einen geordneten Datensatz haben.

Dann können die Daten in 75% Train Daten und 25% Testdaten aufgeteilt werden.

Erstellung der SVM

Das Programm wird mit Train und Testdaten gefüttert und mit der Support Vector Machine mit RBF Kernel kann eine Genauigkeit angegeben werden, zu wie viel Prozent die Testwerte tatsächlich richtig Klassifiziert worden sind.
Nach drei Durchläufen lässt sich sagen das ein Hundebild zu ca. 42-46% der richtigen Hunderasse klassifiziert werden kann.

```
Model accuracy: 0.4272300469483568

Cross-Validation-Score 3-fold
[0.33802817 0.43661972 0.33802817]
```

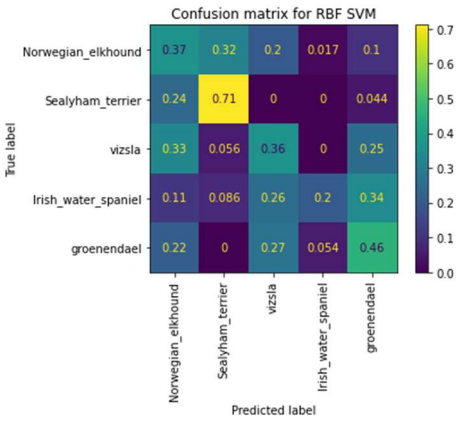
```
Model accuracy: 0.4694835680751174

Cross-Validation-Score 3-fold
[0.42253521 0.53521127 0.43661972]
```

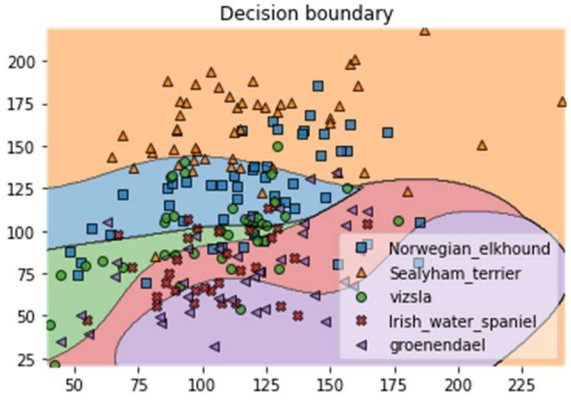
```
Model accuracy: 0.45539906103286387

Cross-Validation-Score 3-fold
[0.45070423 0.47887324 0.46478873]
```

Darstellung:

Zur geeigneten Darstellung wird eine Confusion Matrix erstellt. Die angibt zu wie viel Prozent ein Hundebild (y-Achse) welcher Hunderasse klassifiziert wird (x-Achse)

Auch die Desicion Boundary wird angezeigt, die aus dem SVM generiert wird. Die Bereiche der Hunderassen / Klassifikation aus den Testwerten sind im Schaubild zu erkennen.





Programmierabgabe 2 - Gruppe 5

Klassifikation von Hunderassen mit Deep Learning

von: Thomas Alpert und Lucas Späth

geschätzter Arbeitsaufwand: 10 Stunden (Blut, Schweiß und Tränen)

Ausführbares Jupiter Notebook : hunde_Aufgabe2.ipynb

Zip-File einfach auspacken. Datei und Ordner Struktur:

```
.\annotation\Annotation
.\annotation\Annotation\n02091467-Norwegian_elkhound\annotionfiles
.\annotation\Annotation\n02095889-Sealyham_terrier\annotionfiles
.\annotation\Annotation\n02100583-vizsla\annotionfiles
.\annotation\Annotation\n02102973-Irish_water_spaniel\annotionfiles
.\annotation\Annotation\n02105056-groenendael\annotionfiles
.\images\Images
.\images\Images\n02091467-Norwegian_elkhound\images
.\images\Images\n02095889-Sealyham_terrier\images
.\images\Images\n02100583-vizsla\images
.\images\Images\n02102973-Irish_water_spaniel\images
.\images\Images\n02105056-groenendael\images
.\hunde_Aufgabe1.ipynb
.\hunde_Aufgabe2.ipynb
.\README.pdf
```

Datenvorbereitung:

Die Aufgabe 2 basiert auf demselben Datenbestand wie Aufgabe 1.
Das bedeutet, dass die ersten 5 Zellen des Jupiter-Notebooks identisch mit denen aus Aufgabe 1 sind.

Der einzige Unterschied der eingelesenen Daten in Aufgabe 2 ist, dass hier das gesamte Bild (und nicht wie in Aufgabe 1, einzelne Features aus den Bildern) verwendet wird.

Die eingelesenen (und bereits mit 75:25 gesplitteten) Bilddaten werden anschließend für ein neuronales Netzwerk zurecht-“geshapt“. Dabei werden mehrere Transformationen durchgeführt, um die Daten in die richtige Form für das neuronale Netzwerk zu bringen.

```
# Shape die Daten zurecht
X_train=X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2], 3)
X_train=X_train / 255.0
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2], 3)
X_test=X_test / 255.0

Y_train = tf.one_hot(Y_train.astype(np.int32), depth=5)
Y_test = tf.one_hot(Y_test.astype(np.int32), depth=5)
```

Beschreibung des CNN:
Der verwendete Code zeigt die Definition eines neuen neuronale Netzwerkmodells in TensorFlow mit der "tf.keras.models.Sequential()" Methode. Das Modell besteht aus einer Reihe von Convolutional Neural Network (CNN) Schichten, die zur Extraktion von Merkmalen aus Bilddaten verwendet werden. Die Schichten des Modells sind:

```
num_classes = 5  ## (Anzahl der Hunderassen)
epochs = 10

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.3),

    tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.30),

    tf.keras.layers.Conv2D(128, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.30),


    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(num_classes, activation='softmax'),
```

- Conv2D: Eine 2-dimensional Convolutional Layer, die 32,64,und 128 Filter mit Größe 3x3 und ReLU-Aktivierungsfunktion verwendet.
- MaxPooling2D: Eine Max-Pooling Layer, die die Bildgröße halbiert.
- Dropout: Eine Dropout Layer, die 30% der Eingaben zufällig ausschaltet, um Overfitting zu vermeiden.
- Flatten: Eine Flatten Layer, die die 2-dimensionalen Bilder in 1-dimensionalen Vektoren umwandelt.
- Dense: Eine Fully-Connected Layer, die 512 Neuronen und ReLU-Aktivierungsfunktion verwendet.
- Dense: Eine Fully-Connected Output Layer, die die Anzahl der Klassen entsprechend der "num_classes" Variable (Anzahl der Hunderassen = 5)verwendet und die Softmax-Aktivierungsfunktion verwendet

Bei der Auswahl der des Optimizer und der Loss-Funktion wurde sich für optimizer="adam" und loss="categorical_crossentropy" entschieden.
Mit dieser Kombination konnten die „besten“ Accuracys gewonnen werden.
Obwohl der Optimizer „sgd“ besser geeignet ist für einen kleinen Datensatz ließ sich mit dem „adam“ Optimizer bessere Werte erzielen.
Die Loss-Funktion wurde auf "categorical_crossentropy" gesetzt, da dieser gut die für kategoriale Klassifikationsaufgaben geeignet ist, bei denen die Labels in Form von One-Hot-Vektoren vorliegen.

```
model.compile(optimizer="adam",
              loss='categorical_crossentropy', metrics=['acc'])
```

Trainieren mit der CNN

Nach 15 Epochen des Trainierens (mit Train Daten), nähert sich das CNN einer Genauigkeit von ca. 0,87 und einem Loss-Faktor von 0,33 an.

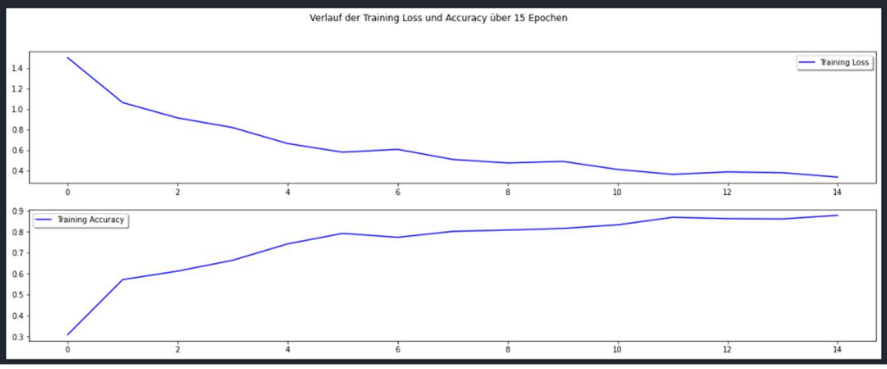
Der Verlauf kann mit dem folgenden Schaubild dargestellt werden:

```
history =model.fit(X_train, Y_train,
                   batch_size=batch_size,
                   epochs=epochs)
```

```
Epoch 11/15
20/20 [==============================] - 0s 23ms/step - loss: 0.4100 - acc: 0.8326
Epoch 12/15
20/20 [==============================] - 1s 30ms/step - loss: 0.3619 - acc: 0.8685
Epoch 13/15
...
Epoch 14/15
20/20 [==============================] - 0s 20ms/step - loss: 0.3780 - acc: 0.8607
Epoch 15/15
20/20 [==============================] - 1s 29ms/step - loss: 0.3349 - acc: 0.8779
```

```
fig, ax = plt.subplots(2,1, figsize=(20, 7))
ax[0].plot(history.history['loss'], color='b', label="Training Loss")
legend = ax[0].legend(loc='best', shadow=True)
ax[1].plot(history.history['acc'], color='b', label="Training Accuracy")
legend = ax[1].legend(loc='best', shadow=True)
fig.suptitle("Verlauf der Training Loss und Accuracy über " +str(epochs) + " Epochen" )
```

```
Text(0.5, 0.98, 'Verlauf der Training Loss und Accuracy über 15 Epochen')
```



Bei der Evaluierung mit den Testdaten und einer anschließenden 3-Fold-Kreuzvalidierung kommt man auf eine abschließende Genauigkeit von ca. 0,84%.
Auch eine Confusion Matrix wurde wieder erstellt. Die angibt zu wie viel Prozent ein Hundebild (y-Achse) welcher Hunderasse klassifiziert wird (x-Achse)

```
from sklearn.model_selection import KFold

# Definiere die Anzahl der Folds
num_folds = 3

# Erstelle ein int, um die Genauigkeit für jeden Fold zu speichern
acc =0

# Teile das Modell in Folds auf
folds = KFold(n_splits=num_folds, shuffle=True)

x_data = X_train
y_data = X_train

# Führe die Kreuzvalidierung durch
for train_index, test_index in folds.split(X_train):
    train_index = train_index.flatten()
    test_index = test_index.flatten()

    # Verwende die Indizes, um die Trainings- und Testdaten aufzuteilen
    x_train, x_test = X_train[train_index], X_train[test_index]
    y_train, y_test = y_data[train_index], y_data[test_index]


    model.fit(X_train, Y_train, epochs=3, verbose=0)
    accuracy = model.evaluate(X_test, Y_test)
    print(accuracy[1])
    acc = acc +accuracy[1]
# Berechne die Durchschnittsgenauigkeit
print("\nDurchschnittsgenauigkeit nach "+str(num_folds)+"-Fold-Validierung:", (acc / num_folds))
```

```
7/7 [==============================] - 0s 5ms/step - loss: 0.5828 - acc: 0.8592
0.8591549396514893
7/7 [==============================] - 0s 5ms/step - loss: 0.6189 - acc: 0.8310
0.8309859037399292
7/7 [==============================] - 0s 7ms/step - loss: 0.5621 - acc: 0.8498
0.8497652411460876

Durchschnittsgenauigkeit nach 3-Fold-Validierung: 0.8466353615125021
```

X-Achse = Predicted Labeles
Y-Achse = die echten Labels