Arbeitsprobe: Zelldetektion und Klassifizierung mit Hilfe neuronaler Netze (Dave)

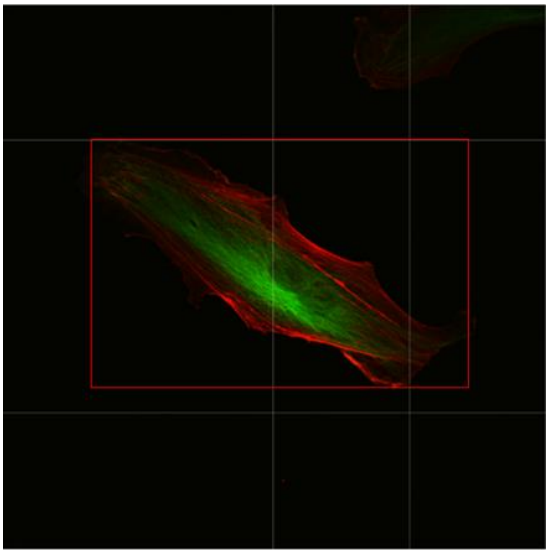
Im Folgenden finden Sie beispielhaft den Einsatz von zwei Neuronalen Netzen zur Lösung eines Bilddetektionsproblems. Die Netze wurden mit Hilfe der Pythonbibliotheken Keras und NumPy erstellt.

Die Arbeit bezieht sich direkt auf den Artikel: *Springer et al.* (2019) PLOS ONE 14(3): e0210570. https://doi.org/10.1371/journal.pone.0210570

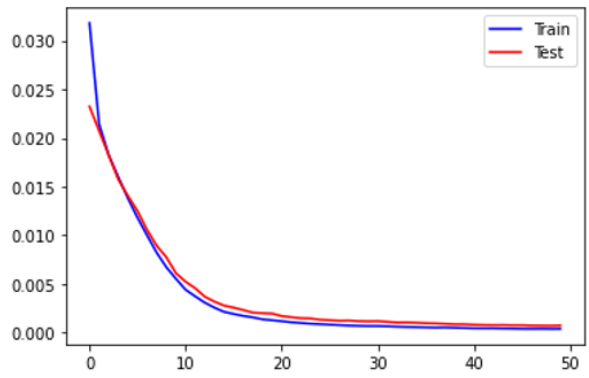
Problem: Biologische Zellen und Zellcluster lassen sich im Labor sehr einfach durch mechanische Dehnung stimulieren. Die dabei stattfindenden Prozesse sind hoch relevant (z. B. Herzzellen, Lungenzellen, etc.) und geben Aufschluss über intra- und interzelluläre Signalprozesse.

Ziel: Änderungen der dabei gefärbten Proteinstrukturen müssen detektiert und miteinander korreliert werden, was eine der größten Herausforderungen in der modernen Zellbiologie darstellt. In diesem Projekt sollen diese Arbeitsschritte von selbstlernenden Algorithmen übernommen und deren Einsatz am Beispiel des Aktinzytoskeletts (Dehnungs-sensible Biopolymerstruktur) in Zellen der Epidermis gezeigt werden.

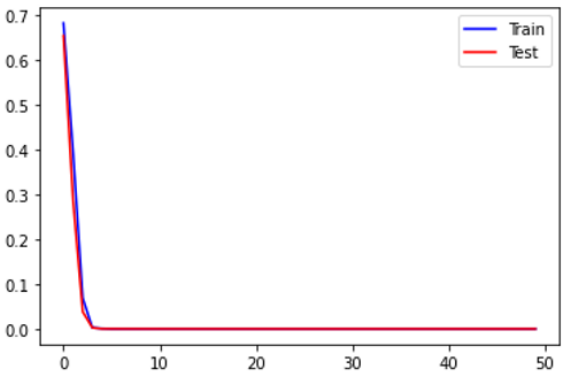
Datenbeschaffung: aus dem Repositorium zu Springer et al, einsehbar unter: https://idr.openmicroscopy.org/webclient/?show=project-505



Multifluoreszenzaufnahme einer Keratinozyte mit Bounding Box



Regression Bounding Boxes



Klassifikation

Loss-funktionen der Neuronalen Netze *links* zur Regression der Bounding Boxen, *rechts* zur Klassifikation der Zellen.

Programm 1: Vorprozessierung der Rohdaten

```
"""
Created on Wed Mar 31 14:00:52 2021
@author: dave-
"""
import matplotlib.pyplot as plt
import os
import skimage
from skimage import io, util

path_images = r'C:\Users\dave-\Documents\python_übungen\detektion_zellen\cells_control'
names_images = os.listdir(path_images) #Liste der Bildnamen
path_images_new = r'C:\Users\dave-\Documents\python_übungen\ detektion_zellen
\cells_control\\'
path_scaled_img = r'C:\Users\dave-\Documents\python_übungen\ detektion_zellen
\images_control\\'

#Zielmaße der Bilddateien
image_height = 600
image_width = 600
dpi = 96
figsize = (image_height/dpi, image_height/dpi) #umrechnung in inch



for i in names_images: #i sind Datei Namen
    j = names_images.index(i)
    image = io.imread(path_images_new + str(names_images[j]))

    #Bilder croppen:
    left = 345
    top = 17
```

```
    right = 1528
    bottom = 1195
    image = image[17:1195, 345:1528, :]

    #Bilder skalieren:
    image = util.img_as_ubyte(skimage.transform.resize(image, (image_height, image_width)))

    #Bilder anzeigen und abspeichern
    fig = plt.figure(figsize=figsize, dpi=dpi, frameon=False)
    ax = plt.Axes(fig, [0., 0., 1., 1.])
    ax.set_axis_off()
    fig.add_axes(ax)
    ax.imshow(image)

    #neuer Bildname, aus Gründen der Reihenfolge der Bildnamen wird bei 100 gestartet
    new_name = path_scaled_img + str(j+100)+"_control"

    fig.savefig(new_name, dpi='figure', bbox_inches='tight', pad_inches=0)
```

Programm 2: Detektion/Klassifikation Keratinozyten

```
"""
Created on Tue Mar 30 14:28:00 2021

@author: dave-
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
import tensorflow as tf
import keras
import os
from skimage import io
import xmltodict


############################################################
#Erstellen der labels:

y = np.empty((160, 4)) # Bounding Boxes
yt = np.zeros((160, 1), dtype=np.int8) # Zustand Zelle

zustand_zelle = ["control", "stretched"]

path_labels = r'C:\Users\dave-\Documents\python_übungen\ detektion_zellen\labels'
dirs_labels = os.listdir(path_labels)
path_new = r'C:\Users\dave-\Documents\python_übungen\ detektion_zellen\labels\\'

for i in dirs_labels:
    j = dirs_labels.index(i)
    #Reihenfolge labels:
```

```
    #print(dirs_labels[j])
    with open(path_new + i) as fd:
        a = xmltodict.parse(fd.read())
    #Parameter Bounding Box:
    xmin = int(a['annotation']['object']['bndbox']['xmin'])
    ymin = int(a['annotation']['object']['bndbox']['ymin'])
    xmax = int(a['annotation']['object']['bndbox']['xmax'])
    ymax = int(a['annotation']['object']['bndbox']['ymax'])
    #Zielvektor: y[,0] ist x-Koordinate der Mitte der Box
    y[j, 0] = xmin + (xmax-xmin)/2
    #y[,1] ist y-Koordinate der Mitte der Box
    y[j, 1] = ymin + (ymax-ymin)/2
    #y[,2] ist die Breite der Box
    y[j, 2] = xmax-xmin
    #y[,3] ist Höhe der Box
    y[j, 3] = ymax-ymin
    #label Zustand Zelle
    type_zelle = a['annotation']['object']['name']
    if type_zelle == 'stretched':
        yt[j, 0] = int(1)
        #Null steht für Kontrolle (ohne Präkonditionierung)

#Kontrolle:

path_images = r'C:\Users\dave-\Documents\python_übungen\detektion_zellen\cells_control'
names_images = os.listdir(path_images) #Liste der Bildnamen
path_images_new = r'C:\Users\dave-\Documents\python_übungen\ detektion_zellen
\cells_control\\'
path_scaled_img = r'C:\Users\dave-\Documents\python_übungen\ detektion_zellen
\images_control\\'

plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1)
a = io.imread(path_images_new + str(names_images[0]))
plt.imshow(a, cmap="gray")
x1 = y[0, 0] - y[0, 2]/2
Breite = y[0, 2]
y1 = y[0, 1] - y[0, 3]/2
Höhe = y[0, 3]
rect = Rectangle((x1, y1), Breite, Höhe, linewidth=1, edgecolor='r', facecolor='none')
ax.add_patch(rect)
plt.title(zustand_zelle[int(yt[0])])
plt.axis("off")
plt.show()
plt.close()

#############################################################################
#Erstellung von X

path_images = r'C:\Users\dave-\Documents\python_übungen\ detektion_zellen\images'
names_images = os.listdir(path_images) #Liste der Bildnamen
path_images_new = r'C:\Users\dave-\Documents\python_übungen\ detektion_zellen\images\\'
```

```
a = io.imread(path_images_new + str(names_images[0]))
#320x320 pixel, 4 Kanäle
a = np.array([a])

X = a

#über for-schleife alle weiteren Bilder dem array hinzufügen:

for j in range(159):
   #Reihenfolge Bilder:
   #print(names_images[j])
   b = io.imread(path_images_new + str(names_images[j+1]))
   #erste Bild wird übersprungen
   b = np.array([b])
   X = np.append(X, b, axis=0)

################################################################
#Skalieren X und y

#Nur ein Kanal (Aktin), da nur dieses Signal für Unterscheidung interessant ist!
X = X[:,:,:,0]

#Data Augmentation (jedes Bild um einen Pixel verschoben, Labels beibehalten):
X1 = np.zeros(shape=(160, 600, 600), dtype=np.int64)
for i in range(160):
   X1[i] = np.roll(np.ravel(X[i]), -1).reshape(600,600)

X = np.append(X, X1, axis=0)
y1 = y
y = np.append(y, y1, axis=0)
yt1 = yt
yt = np.append(yt, yt1, axis=0)

#One-Hot-encoder
yt=tf.keras.utils.to_categorical(yt, 2)

#Kontrolle
plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1)
a = io.imread(path_images_new + str(names_images[0]))
img = X[0].astype(np.uint8)
plt.imshow(img, cmap="gray")
x1 = y[0, 0] - y[0, 2]/2
Breite = y[0, 2]
y1 = y[0, 1] - y[0, 3]/2
Höhe = y[0, 3]
rect = Rectangle((x1, y1), Breite, Höhe, linewidth=1, edgecolor='r', facecolor='none')
ax.add_patch(rect)
plt.axis("off")
plt.show()
plt.close()

plt.figure(figsize=(10, 10))
```

```
ax = plt.subplot(1, 1, 1)
a = io.imread(path_images_new + str(names_images[0]))
img = X[160].astype(np.uint8)
plt.imshow(img, cmap="gray")
x1 = y[160, 0] - y[160, 2]/2
Breite = y[160, 2]
y1 = y[160, 1] - y[160, 3]/2
Höhe = y[160, 3]
rect = Rectangle((x1, y1), Breite, Höhe, linewidth=1, edgecolor='r', facecolor='none')
ax.add_patch(rect)
plt.axis("off")
plt.show()
plt.close()

#X normieren
Norm = X.max()#255
X = X/Norm

#X reshape
Anzahl_Datensätze = 320
Gesamtgröße = 600
X=X.reshape(Anzahl_Datensätze,Gesamtgröße*Gesamtgröße)

#y nomieren
ymax1 = y[:,0].max()
ymax2 = y[:,1].max()
ymax3 = y[:,2].max()
ymax4 = y[:,3].max()

y[:,0] = y[:,0]/ymax1
y[:,1] = y[:,1]/ymax2
y[:,2] = y[:,2]/ymax3
y[:,3] = y[:,3]/ymax4

#Train/test-Split
Test_Größe=0.2
X_train=X[:-int(len(X)*Test_Größe)]
print(X_train.dtype, X_train.shape)
X_test=X[int(Test_Größe-len(X)*Test_Größe):]
print(X_test.dtype, X_test.shape)
y_train=y[:-int(len(X)*Test_Größe)]
print(y_train.dtype, y_train.shape)
y_test=y[int(Test_Größe-len(X)*Test_Größe):]
print(y_test.dtype, y_test.shape)
yt_train=yt[:-int(len(X)*Test_Größe)]
print(yt_train.dtype, yt_train.shape)
yt_test=yt[int(Test_Größe-len(X)*Test_Größe):]
print(yt_test.dtype, yt_test.shape)

################################################################

#Neuronale Netze
```

```
model = keras.Sequential()
model.add(keras.layers.Dense(300, activation='sigmoid'))
model.add(keras.layers.Dense(50, activation='sigmoid'))
model.add(keras.layers.Dense(4,activation='sigmoid'))
optimizer=keras.optimizers.Adam(learning_rate=0.001)

model.compile(loss='mse',
          optimizer=optimizer,
          metrics=[tf.keras.metrics.MeanIoU(num_classes=2)])

history=model.fit(X_train, y_train, batch_size=20,
              epochs=50, verbose=1,
              validation_data=(X_test,y_test))

plt.plot(history.history['loss'], color='b', label='Train')
plt.plot(history.history['val_loss'], color='r', label='Test')
plt.legend()
plt.show()

#Klassifizierung
model2 = keras.Sequential()

model2.add(keras.layers.Reshape((Gesamtgröße, Gesamtgröße, 1),
input_shape=(Gesamtgröße*Gesamtgröße,)))
model2.add(keras.layers.Conv2D(8,(4,4),padding='same',activation='relu'))
model2.add(keras.layers.MaxPool2D(pool_size=(2, 2),padding='same'))
model2.add(keras.layers.Conv2D(8,(3,3),padding='same',activation='relu'))
model2.add(keras.layers.MaxPool2D(pool_size=(2, 2),padding='same'))
model2.add(keras.layers.Conv2D(16,(3,3),padding='same',activation='relu'))
model2.add(keras.layers.MaxPool2D(pool_size=(2, 2),padding='same'))
model2.add(keras.layers.Flatten())
model2.add(keras.layers.Dense(300, activation='relu'))
model2.add(keras.layers.Dense(300, activation='relu'))
model2.add(keras.layers.Dense(50, activation='relu'))
model2.add(keras.layers.Dense(2,activation='softmax'))
optimizer2=keras.optimizers.Adam(learning_rate=0.001)

model2.compile(loss='categorical_crossentropy',
          optimizer=optimizer2)

history2=model2.fit(X_train, yt_train, batch_size=20,
              epochs=30, verbose=1,
              validation_data=(X_test,yt_test))

plt.plot(history2.history['loss'], color='b', label='Train')
plt.plot(history2.history['val_loss'], color='r', label='Test')
plt.legend()
plt.show()

#################################################################
#plotten der ersten 25 Ergebnisse
examples = 25
```

```
y_pred = model.predict(X_test[0:examples])

y_pred[:,0] = y_pred[:,0] * ymax1
y_pred[:,1] = y_pred[:,1] * ymax2
y_pred[:,2] = y_pred[:,2] * ymax3
y_pred[:,3] = y_pred[:,3] * ymax4

yt_pred = model2.predict(X_test)

dim=(5, 5)
figsize=(10, 10)
path_images = r'C:\Users\dave-\Documents\python_übungen\ detektion_zellen \images'
names_images = os.listdir(path_images) #Liste der Bildnamen
path_images_new = r'C:\Users\dave-\Documents\python_übungen\ detektion_zellen \images\\'

fig=plt.figure(figsize=figsize)
for i in range(examples):

   ax= fig.add_subplot(dim[0], dim[1], i+1, xticks=[], yticks=[])
   a = io.imread(path_images_new + str(names_images[i]))
   ax.imshow(a, cmap='gray', interpolation='none')

   #label
   if yt_pred[i, 0] < yt_pred[i, 1]:
      label_pred = "control"
   else:
      label_pred = "stretched"
   plt.title(label_pred)

   x1 = y_pred[i, 0] - y_pred[i, 2]/2
   Breite = y_pred[i, 2]
   y1 = y_pred[i, 1] - y_pred[i, 3]/2
   Höhe = y_pred[i, 3]
   rect = Rectangle((x1, y1), Breite, Höhe, linewidth=1, edgecolor='g', facecolor='none')
   ax.add_patch(rect)

   plt.axis("off")
plt.tight_layout()
plt.show()
plt.close()
```