CS6005 - DEEP LEARNING TECHNIQUES

MINI PROJECT 1 - IMAGE CLASSIFICATION USING CNN

Jerrick Gerald *20.04.2021*

2018103031 *'N' Batch*

HUMAN PROTEIN CLASSIFICATION

Project contents

Problem statement

Interpretation of images of cells has always played important roles in science and medicine. Proteins are “the doers” in the human cell, executing many functions that together enable life. Historically, classification of proteins has been limited to single patterns in one or a few cell types, but in order to fully understand the complexity of the human cell, models must classify mixed patterns across a range of different human cells. Images visualizing proteins in cells are commonly used for biomedical research, and these cells could hold the keys for the next breakthrough in medicine. This project focuses on using simple CNN model to classify the patterns of **protein expression** within **images** of **human** cells.

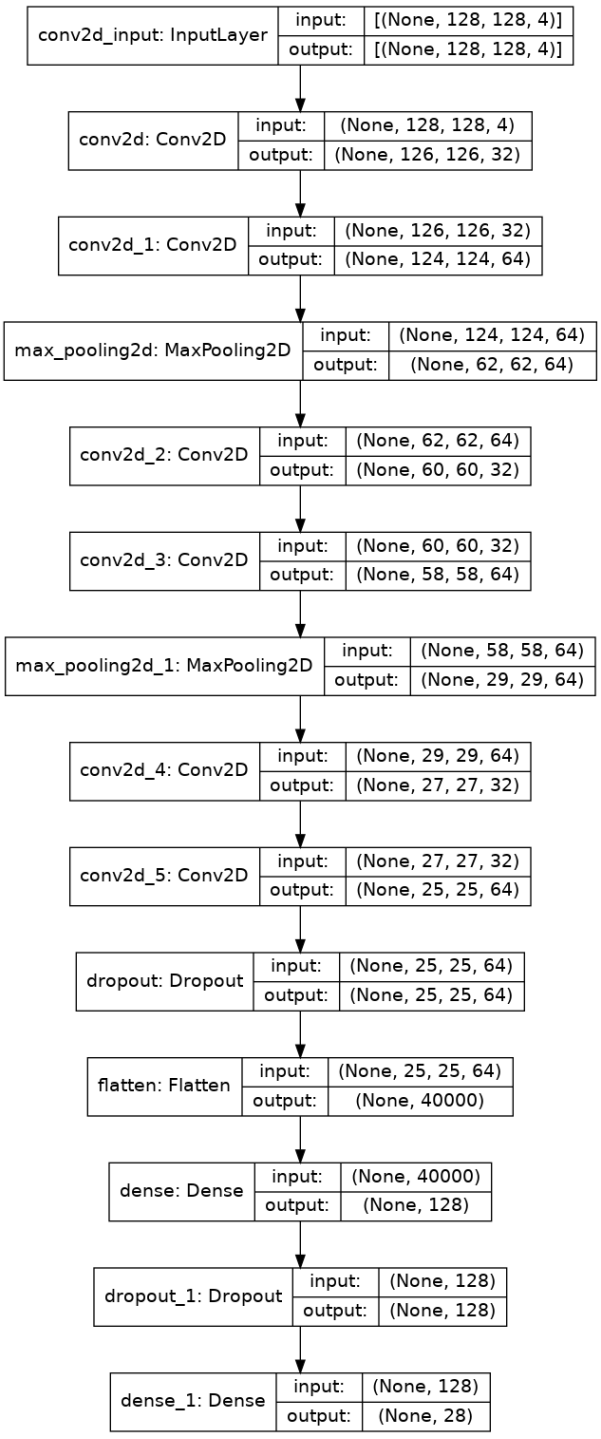
Dataset details

The Human Protein Atlas is a Sweden-based initiative aimed at mapping all human proteins in cells, tissues and organs. The dataset comprises of train and test folders with protein images in “png” format. Along with these folders, two “csv” files namely, “train.csv” containing filenames and labels for the training set and “sample_submission.csv” containing filenames for the test set, and a guide to constructing a working submission. The overall aim is to predict protein organelle localization labels for each sample. There are in total 28 different labels present in the dataset. All image samples are represented by four filters (stored as individual files), the protein of interest (green) plus three cellular landmarks: nucleus (blue), microtubules (red), endoplasmic reticulum (yellow). The green filter should hence be used to predict the label, and the other filters are used as references.

Modules

1. Input visualization & EDA
2. Image preprocessing
3. Model definition
4. Training & Evaluating models
5. Making sample predictions

Model summary




```python
def create_model():
    model = Sequential()
    model.add(Conv2D(32, kernel_size=(3, 3),
                     activation='relu',
                     input_shape=input_shape))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(32, kernel_size=(3, 3),
                     activation='relu',
                     input_shape=input_shape))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(32, kernel_size=(3, 3),
                     activation='relu',
                     input_shape=input_shape))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='sigmoid'))

    model.compile(loss=keras.losses.binary_crossentropy,
                  optimizer=keras.optimizers.Adam(lr=1e-3, decay=1e-3 / epochs)
                  metrics=['accuracy', f1])
    return model
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 32)      1184
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 124, 124, 64)      18496
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 62, 62, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 60, 60, 32)        18464
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 58, 58, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 29, 29, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 27, 27, 32)        18464
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 25, 25, 64)        18496
_________________________________________________________________
dropout (Dropout)            (None, 25, 25, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 40000)             0
_________________________________________________________________
dense (Dense)                (None, 128)               5120128
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 28)                3612
=================================================================
Total params: 5,217,340
Trainable params: 5,217,340
Non-trainable params: 0
_________________________________________________________________
```

Coding snapshots

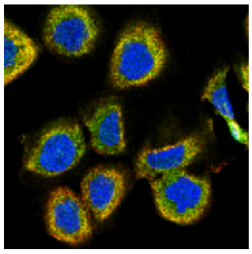
```
name_label_dict = {
0:  'Nucleoplasm',
1:  'Nuclear membrane',
2:  'Nucleoli',
3:  'Nucleoli fibrillar center',
4:  'Nuclear speckles',
5:  'Nuclear bodies',
6:  'Endoplasmic reticulum',
7:  'Golgi apparatus',
8:  'Peroxisomes',
9:  'Endosomes',
10:  'Lysosomes',
11:  'Intermediate filaments',
12:  'Actin filaments',
13:  'Focal adhesion sites',
14:  'Microtubules',
15:  'Microtubule ends',
16:  'Cytokinetic bridge',
17:  'Mitotic spindle',
18:  'Microtubule organizing center',
19:  'Centrosome',
20:  'Lipid droplets',
21:  'Plasma membrane',
22:  'Cell junctions',
23:  'Mitochondria',
24:  'Aggresome',
25:  'Cytosol',
26:  'Cytoplasmic bodies',
27:  'Rods & rings' }
```

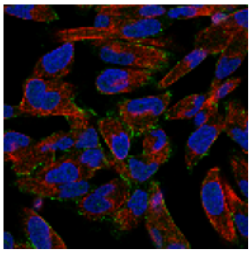
```
SIZE = (128,128)
def get_labels(img):
    return list(map(int, train[img].split(' ')))
# We'll fusion images
def open_multilayer_image(path, test=False):
    fullpath = root_train_directory+path
    if test:
        fullpath = root_test_directory+path
    red = plt.imread(fullpath+"_red.png")
    red = cv2.resize(red, SIZE)
    green = plt.imread(fullpath+"_green.png")
    green = cv2.resize(green, SIZE)
    blue = plt.imread(fullpath+"_blue.png")
    blue = cv2.resize(blue, SIZE)
    yellow = plt.imread(fullpath+"_yellow.png")
    yellow = cv2.resize(yellow, SIZE)
    ni = np.zeros((SIZE[0],SIZE[1],4), 'uint8')
    ni[..., 0] = red*255
    ni[..., 1] = green*255
    ni[..., 2] = blue*255
    ni[..., 3] = yellow*255
    return ni
```

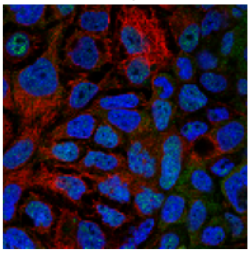
MODULE 1 - For the training to work, we need to merge all the layer of each image and turn them into rgby images. We'll also resize the images to make

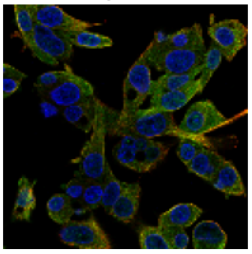
```
import cv2
import matplotlib.pyplot as plt
from random import randrange
from textwrap import wrap

#print('labels:',get_labels(img_name))

fig=plt.figure(figsize=(20, 30))

columns = 5
rows = 4
for i in range(1, columns*rows +1):
    num = randrange(len(train_img_names))
    img_name = train_img_names[num]
    img = open_multilayer_image(img_name)
    sub = fig.add_subplot(rows, columns, i)
    # make title
    title = ''
    for label in get_labels(img_name):
        title+=name_label_dict[label]+', '
    sub.set_title("\n".join(wrap(title[:-2],25)))
    plt.axis('off')
    plt.imshow(img[...,:-1])
plt.show();
```

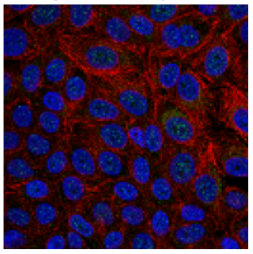
The input is visualized along with the ground truth labels

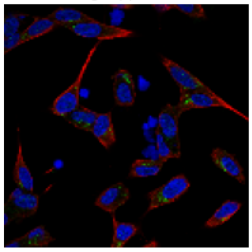

Microtubules

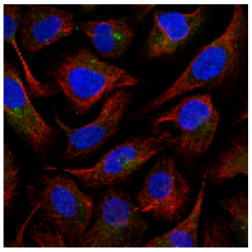

Plasma membrane

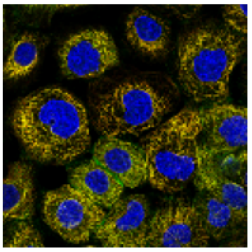

Nuclear speckles

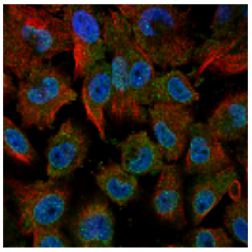

Cytosol

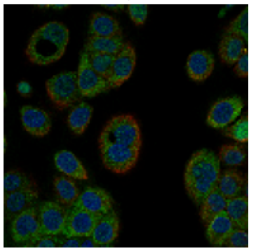

Cytosol, Nucleoplasm

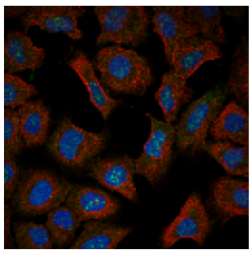

Golgi apparatus

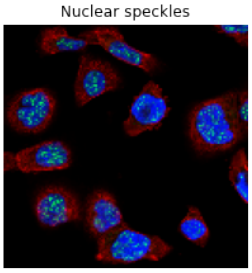

Endoplasmic reticulum, Endosomes, Lysosomes

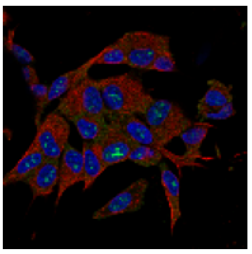

Microtubules

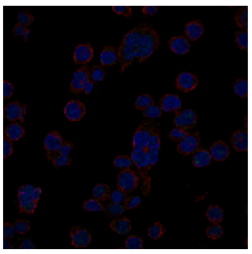

Cytosol, Nuclear membrane, Nucleoplasm

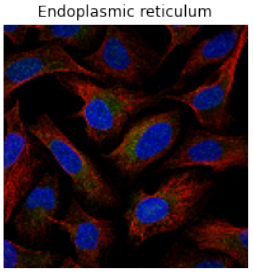

Cytokinetic bridge, Cytosol, Mitotic spindle, Nucleoplasm

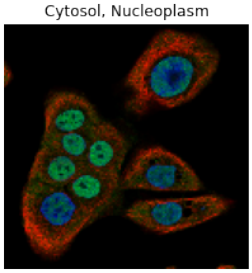

Centrosome, Nucleoli

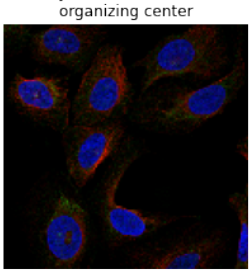

Nuclear speckles

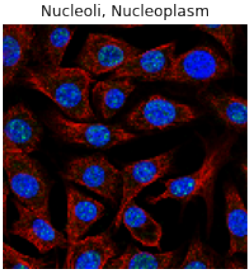

Nucleoli

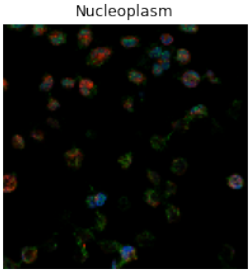

Nuclear bodies

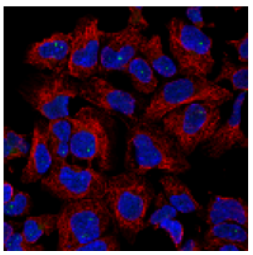

Endoplasmic reticulum


Cytosol, Nucleoplasm

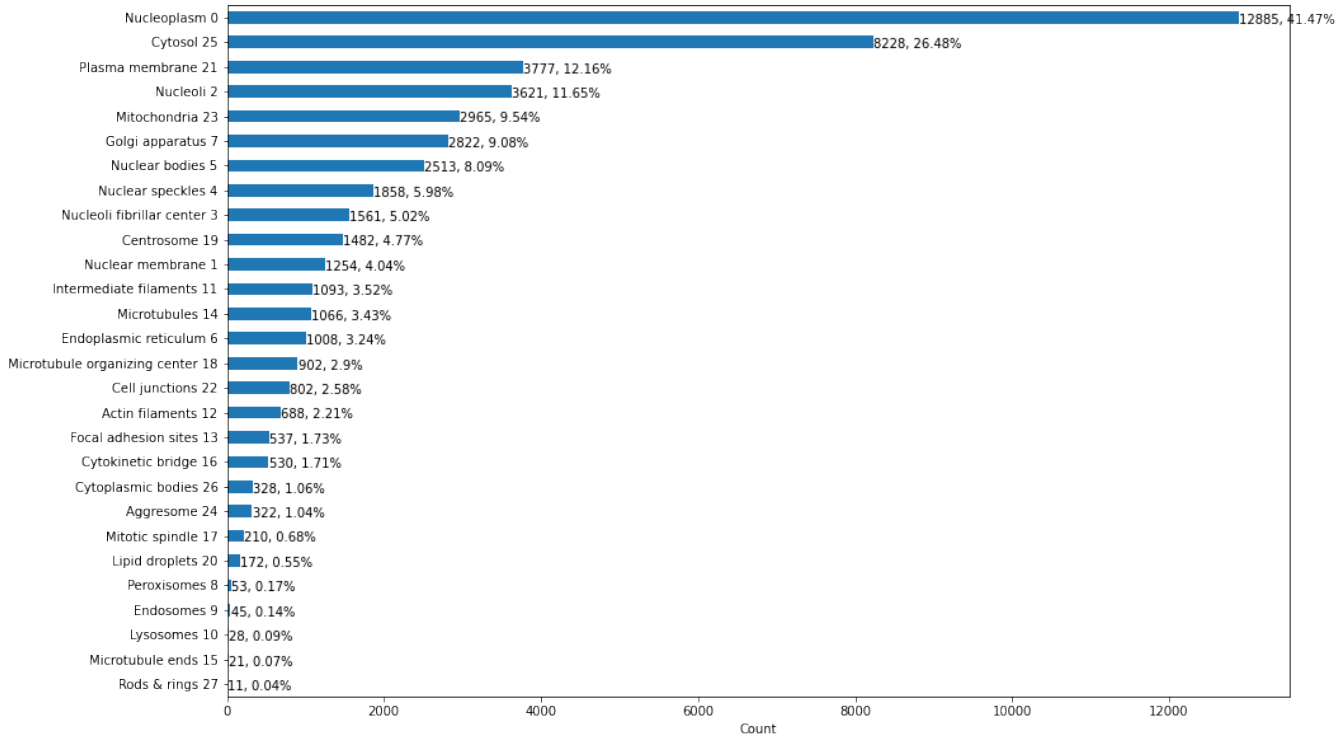

Cytokinetic bridge, Cytosol, Microtubule organizing center

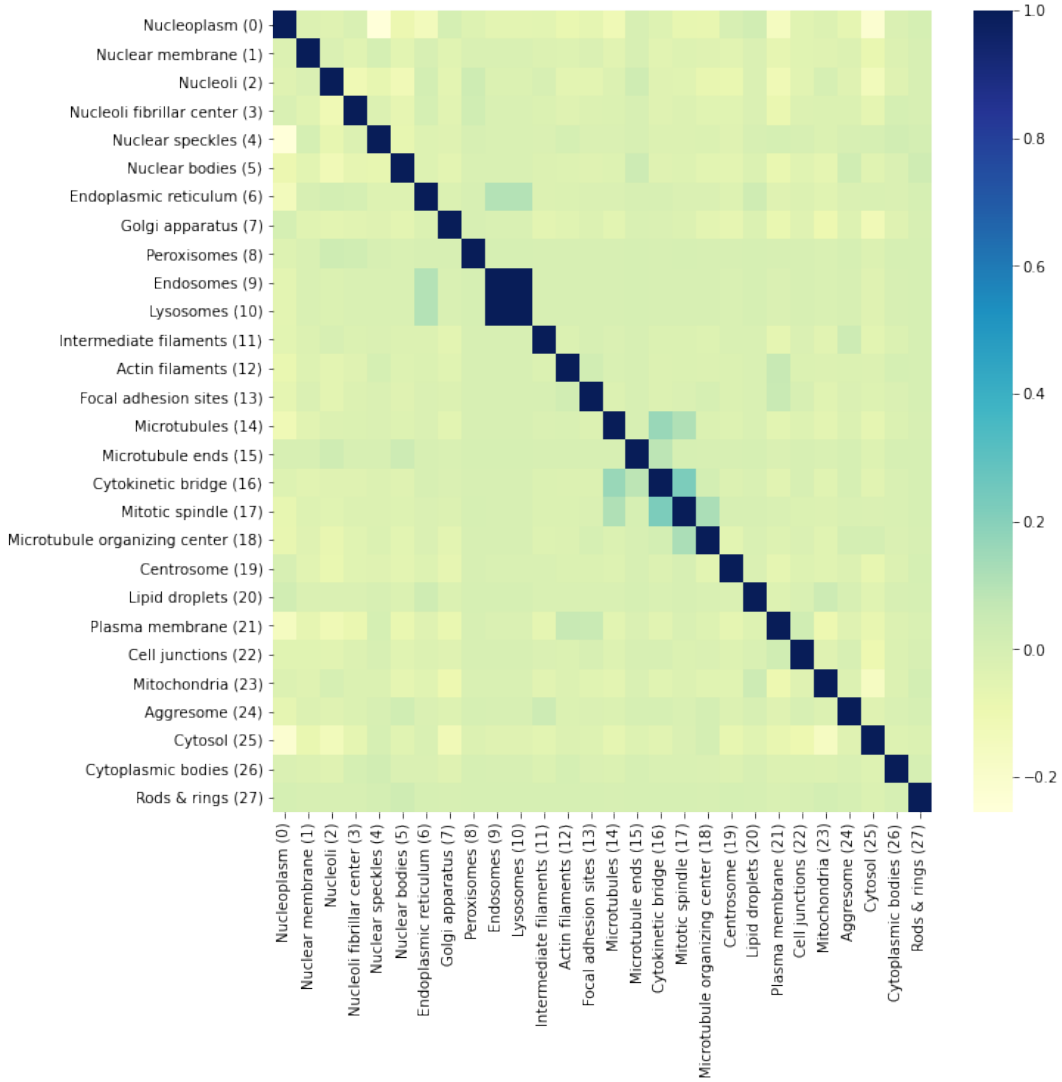

Nucleoli, Nucleoplasm


Nucleoplasm


Nuclear bodies

EDA


Nucleoplasm 0
Cytosol 25
Plasma membrane 21
Nucleoli 2
Mitochondria 23
Golgi apparatus 7
Nuclear bodies 5
Nuclear speckles 4
Nucleoli fibrillar center 3
Centrosome 19
Nuclear membrane 1
Intermediate filaments 11
Microtubules 14
Endoplasmic reticulum 6
Microtubule organizing center 18
Cell junctions 22
Actin filaments 12
Focal adhesion sites 13
Cytokinetic bridge 16
Cytoplasmic bodies 26
Aggresome 24
Mitotic spindle 17
Lipid droplets 20
Peroxisomes 8
Endosomes 9
Lysosomes 10
Microtubule ends 15
Rods & rings 27
12885, 41.47%
8228, 26.48%
3777, 12.16%
3621, 11.65%
2965, 9.54%
2822, 9.08%
2513, 8.09%
1858, 5.98%
1561, 5.02%
1482, 4.77%
1254, 4.04%
1093, 3.52%
1066, 3.43%
1008, 3.24%
902, 2.9%
802, 2.58%
688, 2.21%
537, 1.73%
530, 1.71%
328, 1.06%
322, 1.04%
210, 0.68%
172, 0.55%
53, 0.17%
45, 0.14%
28, 0.09%
21, 0.07%
11, 0.04%
0
2000
4000
6000
8000
10000
12000
Count


Nucleoplasm (0)
Nuclear membrane (1)
Nucleoli (2)
Nucleoli fibrillar center (3)
Nuclear speckles (4)
Nuclear bodies (5)
Endoplasmic reticulum (6)
Golgi apparatus (7)
Peroxisomes (8)
Endosomes (9)
Lysosomes (10)
Intermediate filaments (11)
Actin filaments (12)
Focal adhesion sites (13)
Microtubules (14)
Microtubule ends (15)
Cytokinetic bridge (16)
Mitotic spindle (17)
Microtubule organizing center (18)
Centrosome (19)
Lipid droplets (20)
Plasma membrane (21)
Cell junctions (22)
Mitochondria (23)
Aggresome (24)
Cytosol (25)
Cytoplasmic bodies (26)
Rods & rings (27)
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

```
batch_size = 128
num_classes = len(name_label_dict)
epochs = 60
def preprocess_x(share, train_img_names, test=False):
    x_train = []
    for i in range(0,int(len(train_img_names)*share)):
        x_train.append(open_multilayer_image(train_img_names[i], test))
    x_train = np.array(x_train)
    x_train = x_train.astype('float32')
    x_train /= 255
    return x_train
def preprocess_y(share, train_img_names):
    y_train = []
    for i in range(0,int(len(train_img_names)*share)):
        y_train.append(get_labels(train_img_names[i]))
    y_train = np.array(y_train)
    y_train_formatted = []
    for y in y_train:
        label = np.zeros(num_classes)
        for j in y:
            label[j]=1
        y_train_formatted.append(label)
    y_train = np.array(y_train_formatted)
    return y_train
x_train  = preprocess_x(0.01, train_img_names)
y_train  = preprocess_y(0.01, train_img_names)

x_test = x_train[int(len(x_train)*0.8):]
x_train = x_train[:int(len(x_train)*0.8)]

y_test = y_train[int(len(y_train)*0.8):]
y_train = y_train[:int(len(y_train)*0.8)]
```

MODULE 2 - Image preprocessing

```
def _load_and_preprocess(self, image_path):
    fullpath = image_path
    red = plt.imread(fullpath+"_red.png")
    red = cv2.resize(red, SIZE)
    green = plt.imread(fullpath+"_green.png")
    green = cv2.resize(green, SIZE)
    blue = plt.imread(fullpath+"_blue.png")
    blue = cv2.resize(blue, SIZE)
    yellow = plt.imread(fullpath+"_yellow.png")
    yellow = cv2.resize(yellow, SIZE)
    ni = np.zeros((SIZE[0],SIZE[1],4), 'uint8')
    ni[..., 0] = red*255
    ni[..., 1] = green*255
    ni[..., 2] = blue*255
    ni[..., 3] = yellow*255
    ni = ni.astype('float32')
    ni /= 255
    return ni
```

MODULE 3 - The model definition is provided in model definition section in previous

MODULE 4 - The created model has been called here along with the pre-defined data-generator and finally the the model is trained for 60 epochs.

```
model = create_model()

image_path = root_train_directory
training_generator = DataGenerator(train_img_names[:int(len(train_img_names)*0.8)], image_path, dim=SIZE,
                n_channels=4)
validation_generator = DataGenerator(train_img_names[int(len(train_img_names)*0.8):], image_path, dim=SIZE,
                n_channels=4)

H = model.fit_generator(generator=training_generator, validation_data=validation_generator,
          epochs=epochs,
          verbose=1,
         callbacks = [EarlyStopping(monitor='val_loss', patience=6, verbose=1), ModelCheckpoint
                      (filepath='/tmp/weights.hdf5', verbose=1, save_best_only=True)])
model = create_model()
model.load_weights('/tmp/weights.hdf5')
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Epoch 1/60
776/776 [==============================] - 1753s 2s/step - loss: 0.1871 - accuracy: 0.9399 - f1: 0.0990 - val_loss:
0.1584 - val_accuracy: 0.9473 - val_f1: 0.1088

Epoch 00001: val_loss improved from inf to 0.15836, saving model to /tmp/weights.hdf5
Epoch 2/60
776/776 [==============================] - 1672s 2s/step - loss: 0.1640 - accuracy: 0.9472 - f1: 0.1133 - val_loss:
0.1471 - val_accuracy: 0.9478 - val_f1: 0.1219
```

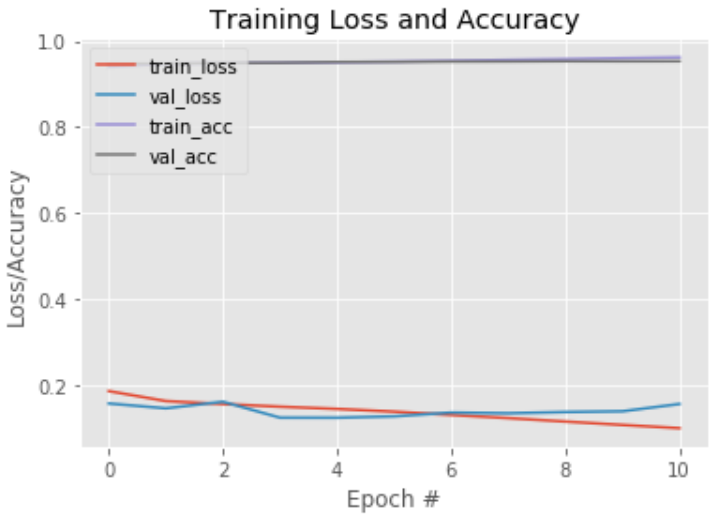
Evaluating the trained model. The code produces accuracy and loss curves which are provided in the Results section.

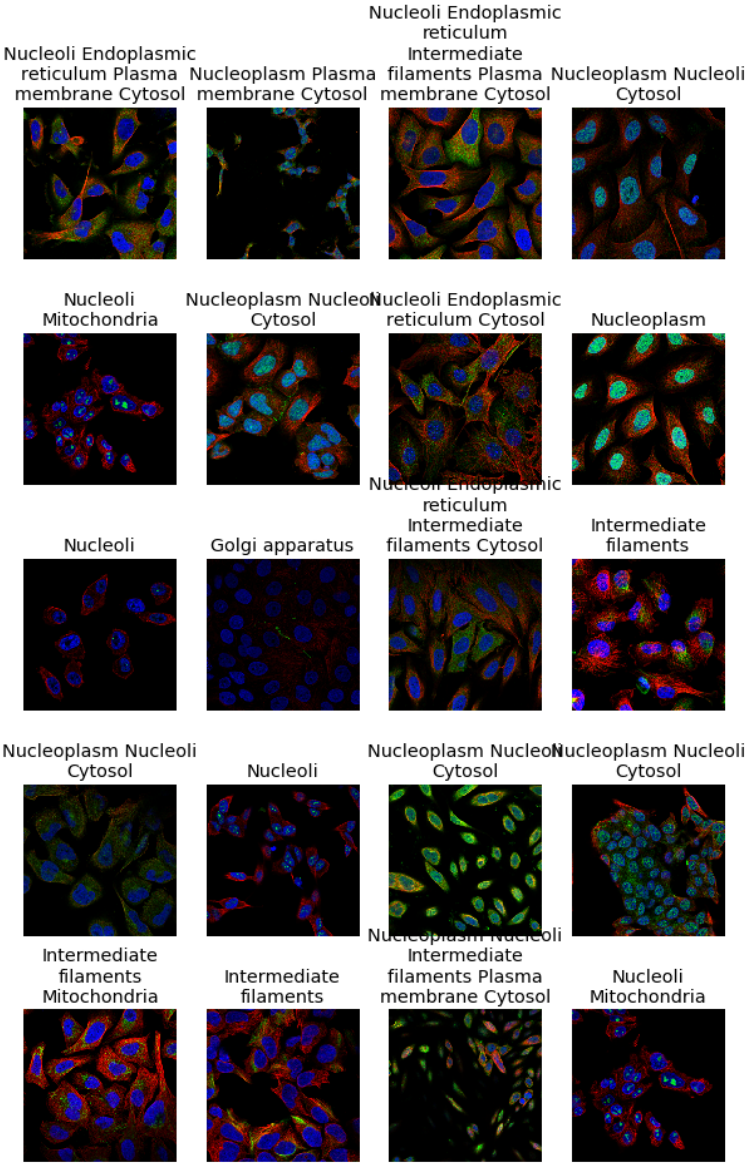
```
model = create_model()
model.load_weights('/tmp/weights.hdf5')
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

# plot the training loss and accuracy
plt.style.use("ggplot")
plt.figure()
N = len(H.history['loss'])
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="upper left")
plt.show()
```

MODULE 5 - Making sample predictions. The well performing model is now provided with random inputs and observed. The input image along with its predicted

```
fig=plt.figure(figsize=(10, 16))

columns = 4
rows = 5
for i in range(1, columns*rows +1):
    num = randrange(len(train_img_names))
    num = randrange(1200)
    img_name = train_img_names[num]
    img = open_multilayer_image(img_name)
    img_to_predict = cv2.resize(img, SIZE)
    pred = model.predict(np.array([img_to_predict]))
    title = pred_to_textlabels(pred)

    real_label = ''
    for label in get_labels(img_name):
        real_label+=name_label_dict[label]+', '
    print('\'',real_label, '\'was predicted as\'',title,'\'' )
    sub = fig.add_subplot(rows, columns, i)
    # make title
    sub.set_title("\n".join(wrap(title,20)))
    plt.axis('off')
    plt.imshow(img[...,:-1])
plt.show();
```

Results


Training Loss and Accuracy
train_loss
val_loss
train_acc
val_acc
Loss/Accuracy
Epoch #
1.0
0.8
0.6
0.4
0.2
0
2
4
6
8
10


Nucleoli Endoplasmic reticulum Plasma membrane Cytosol
Nucleoplasm Plasma membrane Cytosol
Nucleoli Endoplasmic reticulum Intermediate filaments Plasma membrane Cytosol
Nucleoplasm Nucleoli Cytosol
Nucleoli Mitochondria
Nucleoplasm Nucleoli Cytosol
Nucleoli Endoplasmic reticulum Cytosol
Nucleoplasm
Nucleoli
Golgi apparatus
Nucleoli Endoplasmic reticulum Intermediate filaments Cytosol
Intermediate filaments
Nucleoplasm Nucleoli Cytosol
Nucleoli
Nucleoplasm Nucleoli Cytosol
Nucleoplasm Nucleoli Cytosol
Intermediate filaments Mitochondria
Intermediate filaments
Nucleoplasm Nucleoli Intermediate filaments Plasma membrane Cytosol
Nucleoli Mitochondria

Conclusion

The human protein images are well analyzed, understood and a CNN model was developed to make required predictions. Initially, the images were plotted along with the labels i.e., the ground truth labels. The images were then preprocessed i.e., resized and normalized. Then datagenerator was defined and develeoped after which the planned simple CNN model was defined. This was then followed by deciding all the required hyper-parameters and then the model was trained for 60 epochs. The trained model was evaluated and found to be 95% accurate. Having analyzed the model, sample files are provided and the results are plotted along with the considered image. Hence, the components of the human protein cells are well predicted and analyzed.

References

- https://pubmed.ncbi.nlm.nih.gov/28940711/
- https://pubmed.ncbi.nlm.nih.gov/18853439/