DATS 6203
Machine Learning II
Exam 1 Final Report
Instructor: Amir Jafari, PhD
By: Madhuri Yadav
11/02/2020

Table of Contents

Introduction:

In exam 1 we work on the image classification using MLP with Keras Package. For this exam we used the dataset consisting of the images of red blood cells infected with malaria. The data set consists of 4 types of images ‘‘red blood cell’’, ‘‘ring’’, ‘‘schizont’’, ‘‘trophozoite’’. PyCharm, Keras and AWS services were used to implement the model.

Day wise progress and final model:

Day 2:

Summary: On day 2 I worked on setting up the project connecting it with AWS instance, Understanding the problem and the dataset and the example script provided at Link.

The data set provided for this exam could be found at the location Link. The dataset consists of images of 4 types ‘‘red blood cell’’, ‘‘ring’’, ‘‘schizont’’, ‘‘trophozoite’’. The dataset was downloaded to the cloud for further use. The dataset consists of total 8430 number of images of different sizes. The images are colored i.e. (R, G, B).

To implement the solution for the exam in the same fashion as mention in example script I implemented train.py file to generate the MLP model. This day’s work included the reading of all the images from the train folder and resizing the images to the mean of width and height of all the images i.e. (115,115). The types of the images was read in the similar fashion and stored as list of strings and encoded for further use. Some of the parameters used were initialized to random values using the example script provided. Seed value is initialized.

Before we begin the data preprocessing the data it split into the train and test with 80:20 ratio. The split is stratified split because the data could be unbalanced. The train features are then converted to NumPy ndarray. The shape of the data set is changed to a vector form as MLP accepts input in vector form. Since the RGB values range between 0 and 255 the data is scaled dividing it by 255. The encoded labels are converted to categorical type. The inp_dim is then set to the length of each vector generated.

The first model consisted of only 2 layers with Relu as activation function for first layer and second layer consisted of Softmax as transfer function with 4 neurons as the images needs to be classified into 4 types. Then the model was fit with train and test data(for validation) with N_NEURONS = 300, N_EPOCHS = 10 and BATCH_SIZE = 512.

Results:

- Final accuracy on validations set: 83.73%
- Cohen Kappa 0.31
- F1 score 0.33

Day 3 and day 4:

Summary: worked on the predict function and the improvement of the model by data augmentation.

Using the example script as a reference I generated predict script in the and tested it using check predict function.

Since the data was highly imbalanced with following values I tried to augment the dataset and balance the dataset across 4 features. Basic augmentation techniques like rotate, flip, blur are used

	red blood cell	ring	schizont	trophozoite
Original set	7000	365	133	1109
Test set	1400	73	27	222
Train set	5600	292	106	887

Since **Red blood cells** are more in number, only few of the images are augmented by using vertical and horizontal flips.

Trophozoite images are augmented by using rotations by 90, 180, 270 degrees and also by blurring the image. This augmentation gives 7 more images for each image.

Since **Ring and Schizont** are very less in number rotation of 90, 180, 270 degrees, also flip and blur is used on these images. For each of the images in this type we get 23 augmented images.

The number of images after Augmentation:

	red blood cell	ring	schizont	trophozoite
Original set	9600	7008	2544	1776

Further the number of epochs increased from 10 to 40 since the model was taking more time to learn. Added 4 additional intermediate layers to improve the performance of the model. The activation functions tanh, sigmoid and Relu. Were used for these layers.

The results obtained by this model are:

- Final accuracy on validations set: 93%
- Cohen Kappa 0.82
- F1 score 0.78

Day 5:

Made changes in the predict file to resolve the error in the previous submission.

To further improve the model performance tried following implementations.

1. Tried further augmentation using brightness changes, image cropping etc.
2. Tried to use gray scale images instead of RGB.
3. Tried to improve and optimize the model using Dropout and Batch Normalization
4. Tried to vary parameters like number of neurons, batch size (32, 64, 256 in powers of 2)
5. Tried different number of epochs observed overfitting.
6. Tried for different combinations of activation functions
7. Finally, tried to add early stopping.

None of the above variations seem to improve the model performance than the previous day's model.

Day 6:

Summary: worked on cleaning the code and improvement on the model performance.

On day six formatted the code and added functions data preprocessing and data augmentation.

```
def pre_processing(x_raw, y_raw):
    features = []
    for i in x_raw:
        features.append(cv2.resize(i, (RESIZE_TO, RESIZE_TO)))

    # # Pre-processing the dataset
    le = LabelEncoder()
    le.fit(["red blood cell", "ring", "schizont", "trophozoite"])
    label = le.transform(y_raw)
    print(label)

    # Calculating the number of images of each type
    unique, counts = np.unique(label, return_counts=True)
    p = dict(zip(unique, counts))
    print(p)

    # converting dataset to numpy
    features, label = np.array(features), np.array(label)
    return features, label
```

```
# data augmentation and data preprocessing functions
def data_augment(x_train, y_train):
    # train data Augmentation
    x_raw, y_raw = [], []
    cnt = 0
    for i in range(len(x_train)):
        l = y_train[i]
        x = [x_train[i]]
        mx = []
        alpha = 1
        beta = 40
        ksize = (5, 5)
        if l == "schizont" or l == "ring":
            # rotating original image by 90,180 and 270 degree
            for m in x:
                mx.append(m)
                mx.append(cv2.rotate(m, cv2.cv2.ROTATE_90_CLOCKWISE))
                mx.append(cv2.rotate(m, cv2.ROTATE_180))
                mx.append(cv2.rotate(m, cv2.ROTATE_90_COUNTERCLOCKWISE))
            x = mx
            mx = []
            # flipping each of the rotated images vertically and horizontally
            for m in x:
                mx.append(m)
                mx.append(cv2.flip(m, 0))
                mx.append(cv2.flip(m, 1))
            x = mx
            mx = []
            # blurring all teh above images
            for n in range(4):
                mx.extend(x[(n * 3):(n * 3)+3])
```

```
            mx.append(cv2.blur(x[n * 3], ksize))
            mx.append(cv2.blur(x[(n * 3) + 1], ksize))
            mx.append(cv2.blur(x[(n * 3) + 2], ksize))
        x = mx
        mx = []
    elif l == "trophozoite":
    # rotating original image by 90,180 and 270 degree
        for m in x:
            mx.append(m)
            mx.append(cv2.rotate(m, cv2.cv2.ROTATE_90_CLOCKWISE))
            mx.append(cv2.rotate(m, cv2.ROTATE_180))
            mx.append(cv2.rotate(m, cv2.ROTATE_90_COUNTERCLOCKWISE))
        x = mx
        mx = []
        # blurring all teh above images
        for m in x:
            mx.append(m)
            mx.append(cv2.blur(m, ksize))
            # mx.append(cv2.cvtColor(m, cv2.COLOR_BGR2HSV))
        x = mx
        mx = []

    elif l == "red blood cell" and cnt <= 2000:
        # flipping each of the rotated images vertically and horizontally
        cnt = cnt + 1
        for m in x:
            mx.append(m)
            mx.append(cv2.flip(m, 0))
            mx.append(cv2.flip(m, 1))
        x = mx

    x_raw.extend(x)
    y_raw.extend([l] * len(x))

return x_raw, y_raw
```

Day 7:

As mention in the towards data science website mentioned in link 3 I tried to replace Relu with Selu Observed some amount of improvement in the model. Finally, I tried to change the layer structure as follows.

```
model = Sequential()
model.add(Dense(N_NEURONS, input_dim=inp_dim, activation="selu"))
model.add(Dense(1024, activation="tanh"))
model.add(Dense(512, activation="selu"))
model.add(Dense(256, activation="tanh"))
model.add(Dense(64, activation="selu"))
model.add(Dense(4, activation="softmax"))
model.compile(optimizer=Adam(lr=LR), loss="categorical_crossentropy", metrics=["accuracy"])
```

This was the best model out of all the above models tried with following results on the test dataset.

- Final accuracy: 94.94%

- Cohen Kappa: 0.83
- F1 score 0.79

Final Model: Hence after multiple iterations final model is as follows.

- SEED = 42
- LR = 1e-4
- N_NEURONS = 300
- N_EPOCHS = 40
- BATCH_SIZE = 512
- RESIZE_TO = 50

There were no changes made in the data augmentation after day 4. The number of images is as follows

	red blood cell	**ring**	**schizont**	**trophozoite**
Original set	7000	365	133	1109
Test set	1400	73	27	222
Train set	5600	292	106	887
Augmented Train set	9602	7008	2544	7096

Model creation:

```
model = Sequential()
model.add(Dense(N_NEURONS, input_dim=inp_dim, activation="selu"))
model.add(Dense(1024, activation="tanh"))
model.add(Dense(512, activation="selu"))
model.add(Dense(256, activation="tanh"))
model.add(Dense(64, activation="selu"))
model.add(Dense(4, activation="softmax"))
model.compile(optimizer=Adam(lr=LR), loss="categorical_crossentropy", metrics=["accuracy"])
```

References:

1. https://docs.opencv.org/master/d4/d13/tutorial_py_filtering.html
2. https://www.geeksforgeeks.org/python-opencv-cv2-cvtcolor-method/
3. https://towardsdatascience.com/gentle-introduction-to-selus-b19943068cd9
4. https://github.com/amir-jafari/Deep-Learning/tree/master/Exam_MiniProjects/3-Keras_Exam1_Sample_Codes_F19