Malaria Cell Classification using Multi-Layer Perceptron

Anwesha Tomar
MS Data Science
George Washington University
atomar19@gwu.edu

1. **Introduction:**

   We use the Multilayer perceptron(MLP) to classify 4 different cells, that are: “red blood cell”, “ring”, “schizont” and “trophozoite”. Over the 7 days of the competition an average score of 7.55747E-01.

2. **Thought process for training the models:**

   Day 1- Basic model, I did not make a lot of changes, I wanted to make sure that the model runs and that it correctly inputs the image path without errors.

```
def predict(dir):
   RESIZE_TO = 50
   x=[]
   # %% ---------------------------------------------- Data Prep
-------------------------------------------------------
   for i in dir:
       x.append(cv2.resize(cv2.imread(i), (RESIZE_TO, RESIZE_TO)))
   x=np.array(x)
   x = x.reshape(len(x), -1)
   x = x / 255

   # %% ---------------------------------------------- Predict
-------------------------------------------------------
   model = load_model('mlp_day5_e.hdf5')
   y_pred = np.argmax(model.predict(x), axis=1)
   return y_pred, model
```

   Day 2- Corrupt model

   Day 3- Corrupt model

   Day 4- Manually running multiple iterations with sets of learning rate, number of neurons, number of epochs, batch size and dropout, this gave me a very small improvement on my machine, but the mean score dropped on the held out set.

I was only focussing on changing the 4 hyperparameters:
LR, N_EPOCHS, BATCH_SIZE, DROPOUT.

Day 5- Next I ran a Grid Search CV, which did not work, it gave an error for the sequential model. I attempted to emulate the process by using multiple for loops, but the process never ended and went into a forever loop.

```
LR = [0.1,0.01,0.001,0.0001,0.00001]
N_NEURONS = [100,200,300,400,500]
NN_I=[128,32,32]
N_EPOCHS =  [100,150,250,300]
BATCH_SIZE = [54,68,81,102,108,324,512,612]
DROPOUT = [0.1,0.2,0.3,0.4,0.5]
KI= ['uniform','normal','lecun_normal']
ACT=['relu', 'elu','tanh','softmax','selu']

# %% ------------------------------------ Data Prep
-----------------------------------------------------------------
x, y = np.load("x_train.npy"), np.load("y_train.npy")
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=SEED,
test_size=0.2, stratify=y)
x_train, x_test = x_train.reshape(len(x_train), -1),
x_test.reshape(len(x_test), -1)
x_train, x_test = x_train/255, x_test/255
y_train, y_test = to_categorical(y_train, num_classes=4),
to_categorical(y_test, num_classes=4)

# %% ------------------------------------ Training Prep
-------------------------------------------------------
#iterate:
df=pd.DataFrame()
l_a=[]
nn_a=[]
nn_i_a=[]
ne_a=[]
b_a=[]
d_a=[]
acc=[]
ck=[]
f1=[]
k_a=[]
act_a=[]
for l in LR:
   for nn in N_NEURONS:
       for ne in N_EPOCHS:
           for b in BATCH_SIZE:
               for d in DROPOUT:
```

```
                    for k in KI:
                        for activ in ACT:
                            for nn_i in NN_I:

                                model = Sequential([
                                    Dense(nn, input_dim=7500,
activation=activ,kernel_initializer=k),
                                    Dense(nn_i, activation=activ,
kernel_initializer=k),
                                    Dense(4,
activation=activ,kernel_initializer=k),
                                    BatchNormalization()
                                ])
                                model.compile(optimizer=Adam(lr=l),
loss="categorical_crossentropy", metrics=["accuracy"])
                                # %% -------------------------------------
Training Loop ------------------------------------------------------------
                                 model.fit(x_train, y_train, batch_size=b,
epochs=ne, validation_data=(x_test, y_test),

callbacks=[ModelCheckpoint("mlp_grid.hdf5", monitor="val_loss",
save_best_only=True)])
                                l_a.append(l)
                                nn_a.append(nn)
                                nn_i_a.append(nn_i)
                                ne_a.append(ne)
                                b_a.append(b)
                                d_a.append(d)
                                k_a.append(k)
                                act_a.append(activ)
                                acc.append(100 * model.evaluate(x_test,
y_test)[1])

ck.append(cohen_kappa_score(np.argmax(model.predict(x_test), axis=1),
np.argmax(y_test, axis=1)))

f1.append(f1_score(np.argmax(model.predict(x_test), axis=1), np.argmax(y_test,
axis=1), average='macro'))


df['learning rate']=l_a
df['n_neurons']=nn_a
df['inner neuron']=nn_i_a
df['epochs']=ne_a
df['batch']=b_a
df['dropout']=d_a
df['accuracy']=acc
df['cohens kappa']=ck
df['f1 score']=f1
```

```
df['Kernel activation']=k_a
df['activation']=act_a
print(df)
df.to_csv(index=False)
```

Next, I attempted to look at various code changes that could improve the model.

I first attempted to modify the Adam optimizer. After looking into the documentation I realized that there were three hyper parameters, beta_1, beta_2, and epsilon. The model did not improve on changing the beta_1, beta_2 values, the default values(0.9, and 0.99) seemed to be the best values. Next I experimented on the epsilon values, according to an article(1), the value of epsilon should be set close to 1 for computer vision problems. This led to an extreme drop in the accuracy and therefore I discarded the idea.

Next, I attempted to increase the number of “steps per epoch”, I quickly realized that as the batch size changes the value of steps per epoch is adjusted, therefore I discarded this idea too.

Then I started adding a kernel initializer, after experimenting with three initializers namely, glorot_uniform, he_uniform and lecun_uniform. I realized that the best kernel initializer was glorot_uniform and improved the performance of the model.

After adding a few more layers to the sequential model, I began to modify the activation functions, tanh, sigmoid and relu were the activation functions that I was experimenting with initially. After reading the kears layer activation function documentation(2), I started experimenting selu, this activation function when added to the hidden layers caused a small increase in the mead score. So I used the selu activation function in the hidden layers.

Lastly I experimented with Batch Normalization, for most image classifications batch normalisation was used and after the activation function and the after adding the dropout rate, this significantly dropped the accuracy and so I discarded it.

Day 6- Did not submit a model as I did not create any model with an improvement

Day 7- Final Model

The final model includes,

```
LR = 0.0001
N_EPOCHS =  150
```

```
BATCH_SIZE = 68
DROPOUT = 0.1
```

I used a loop function to find batch size values, as we want there to be no remainders.

```
for i in range(1,5508):
   if (5508%i)==0:
       print(i)
```

Next I used a 4 layer sequential function,

```
model = Sequential()
model.add(Dense(128,
input_dim=7500,activation="relu",kernel_initializer=weight_init))
model.add(Dropout(DROPOUT, seed=SEED))
model.add(Dense(32, activation="selu",kernel_initializer=weight_init))
model.add(Dropout(DROPOUT, seed=SEED))
model.add(Dense(32, activation="selu",kernel_initializer=weight_init))
model.add(Dropout(DROPOUT, seed=SEED))
model.add(Dense(4,activation="softmax",kernel_initializer=weight_init))
```

In some of my initial models I set the last layer to 10,

```
model.add(Dense(10,activation="softmax",kernel_initializer=weight_init))
```

But the output of the MLP should only be 4 classes and therefore I changed the value in both number of classes and dense layer to 4.

```
y_train, y_test = to_categorical(y_train, num_classes=4),
to_categorical(y_test, num_classes=4)
```

Although setting `num_classes=10` and having 10 dense layers as the output layer does give a higher mean score, I do not believe it is the right approach and therefore I chose to change the values to 4.

Note: As I have only submitted 4 fully working models, I will add only those 4 folders, day 2,3 and 6 folders will be missing.

References:

1. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
2. https://keras.io/api/layers/activations/