Exam Report

Using Keras, three Multi-Layer Perceptron (MLP) models were built and ensembled to classify blood cells into 4 different categories: “red blood cell”, “ring”, “schizont” and “trophozoite”. PIL (Python Image Library) was used to preprocess the image. The hyperparameter optimization libraries – Hyperopt and Optuna were tried and compared in training the MLP model and tuning the hyperparameter.

1. Load data

```
os.chdir("/home/ubuntu/train/")
mypath = "/home/ubuntu/train/"
images = [f for f in listdir(mypath) if f.endswith(".png")]
images.sort()
txts = [f for f in listdir(mypath) if f.endswith(".txt")]
txts.sort()

y = []
for f in txts:
    with open(f, 'r') as file:
        y.append(file.read())
```

2. Preprocess the images

    1) Oversampling unbalanced classes by rotating images

    Before oversampling, there are 7000 images of ‘red blood cell’, 365 images of ‘ring’, 133 images of ‘schizont’, and 1109 images of ‘trophozoite’. By rotating the images, the number of each class is brought to the same, 7000 here. This process made the score get higher from 0.41 to 0.53.

```
index0 = [i for i, x in enumerate(y) if x =='red blood cell']
index1 = [i for i, x in enumerate(y) if x =='ring']
index2 = [i for i, x in enumerate(y) if x == 'schizont']
index3 = [i for i, x in enumerate(y) if x =='trophozoite']

image_list = []
for i in index0:
    image_list.append(images[i])
```

```
def rotate_image(images, index, max):
    image_list = []
    angle = 360 /(max/len(index))
    for i in index:
        image = Image.open(images[i])
        for j in range(int(max/len(index))):
            path = str(j) + images[i]
            rotated = image.rotate(angle * (j + 1))
            rotated.save(path)
            image_list.append(path)
        image.close()
    return image_list

image_list = image_list + rotate_image(images, index1, len(index0))
image_list = image_list + rotate_image(images, index2, len(index0))
image_list = image_list + rotate_image(images, index3, len(index0))
```

2) Reformat images before fed to the model

Images are resized with the aspect ratio preserving. I first chose the largest images in the dataset to be the standard size, however this resulted in great memory allocation. Therefore, the median of the image size which is 128*128 was chosen as the standard size to resize all the images.

Further, a white background of size 128*128 was created and the manipulated image was pasted to the center of the background to make all the images same size before feeding to the model. In the end, the raveled np array of the image was returned.

Model1 and model2 were fed with the images manipulated with the stated process.

Additionally, model 3 was fed with the images processed with gamma correction, which corrects images' luminance. Gamma values < 1 will shift the image towards the darker end of the spectrum while gamma values > 1 will make the image appear lighter. Here, I chose gamma = 1.2 to make images slightly brighter.

```
def Reformat_Image(image_path, widest, highest):
    image = Image.open(image_path)
    image.thumbnail((widest, highest))
    image_size = image.size
    width = image_size[0]
    height = image_size[1]
    gamma = 1.2
    result_img = Image.new("RGB", (width, height))
    dict = {}
    for i in range(256):
        value = round(pow(i / 255, (1 / gamma)) * 255, 0)
        if value >= 255:
            value = 255
        dict[i] = int(value)
    for x in range(width):
        for y in range(height):
            value1 = dict[image.getpixel((x, y))[0]]
            value2 = dict[image.getpixel((x, y))[1]]
            value3 = dict[image.getpixel((x, y))[2]]
            result_img.putpixel((x, y), (int(value1), int(value2), int(value3)))
    background = Image.new('RGB', (widest, highest), (255, 255, 255))
    offset = (int(round(((widest - width) / 2), 0)), int(round(((highest - height) / 2), 0)))
    background.paste(image, offset)
    pix = np.ravel(np.array(background.getdata()))
    image.close()
    os.remove(image_path)
```

3. Build the model

   The hyperparameter optimization libraries – Hyperopt and Optuna were tried and compared in training the MLP model and tuning the hyperparameter. Optuna was used in the end, as it does a bit better job with simpler, more user-friendly interface. An objective function, which return negative cohen_kappa_score, was defined and optimized in the minimizing direction.

   An integer value was suggested for number of neurons from 200 to 500, which was first tried in range (200, 1000) and then narrowed down. A float value of dropout is suggested from 0.2 to 0.5. An integer value was suggested for number of layers from 3 to 7.

```
study = optuna.create_study(study_name = 'cell', storage = 'sqlite:///cell.db', load_if_exists = True, direction='minimize')
filepath = "mlp_xinyu_yao.hdf5"
```

```
def objective(trial):
    global X_train_std, y_train, y_test
    model = Sequential([
        Dense(trial.suggest_int('neurons', 200, 500), input_dim=128*128*3, activation='relu', kernel_initializer=glorot_uniform
        Dropout(trial.suggest_uniform('dropout', 0.2, 0.5), seed=0),
        BatchNormalization()
    ])
    n_neurons = trial.suggest_int('n_neurons', 3, 7)
    for i in range(2, n_neurons):
        model.add(Dense(trial.suggest_int('neurons', 200, 500), activation='relu', kernel_initializer=glorot_uniform(0)))
        model.add(Dropout(trial.suggest_uniform('dropout', 0.2, 0.5), seed=0))
        model.add(BatchNormalization())
    model.add(Dense(4, activation='softmax', kernel_initializer=glorot_uniform(0)))
    model.compile(optimizer=Adam(lr=trial.suggest_loguniform('lr', 1e-4, 0.1)), loss="categorical_crossentropy", metrics=["acc
    model.fit(x_train, y_train, batch_size=trial.suggest_int('batch_size', 512, 10000), shuffle=True, epochs=200,
              validation_data=(x_test, y_test), callbacks= [ModelCheckpoint(filepath, monitor="val_loss", save_best_only=True),
                                                            EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbos
    return -cohen_kappa_score(np.argmax(model.predict(x_test),axis=1),np.argmax(y_test,axis=1))

study.optimize(objective, n_trials = 100, timeout=1800)
print('Number of finished trials: {}'.format(len(study.trials)))
print('Best trial:')
trial = study.best_trial
```

In the end, three models were chosen:

	Model1	Model2	Model3
Batch size	8180	5863	2911
Dropout	0.29	0.40	0.24
Learning rate	0.001	0.003	0.013
Number of layers	4	3	4
Number of neurons	284	403	338

4. Ensemble models

1) Flip
Addition to the original images fed to the model1, the flipped version of the images was fed to the model2 for prediction.

2) Gamma correction
Model3 was trained on the images after gamma correction and was used for prediction on the images after gamma correction

3) Ensemble
Probability of the 4 classes predicted by the 3 models are added together, and the class with the highest probability was returned as prediction.

```python
def Reformat_Image(ImageFilePath, widest, highest, flip=False, correction=False):
    image = Image.open(ImageFilePath, 'r')
    image.thumbnail((widest, highest))
    image_size = image.size
    width = image_size[0]
    height = image_size[1]
    if flip:
        image = image.transpose(Image.FLIP_LEFT_RIGHT)
    if correction:
        gamma = 1.2
        result_img = Image.new("RGB", (width, height))
        dict = {}
        for i in range(256):
            value = round(pow(i / 255, (1 / gamma)) * 255, 0)
            if value >= 255:
                value = 255
            dict[i] = int(value)
        for x in range(width):
            for y in range(height):
                value1 = dict[image.getpixel((x, y))[0]]
                value2 = dict[image.getpixel((x, y))[1]]
                value3 = dict[image.getpixel((x, y))[2]]
                result_img.putpixel((x, y), (int(value1), int(value2), int(value3)))
    background = Image.new('RGB', (widest, highest), (255, 255, 255))
    offset = (int(round(((widest - width) / 2), 0)), int(round(((highest - height) / 2), 0)))
    background.paste(image, offset)
    pix = np.array(background.getdata()).reshape((widest, highest, 3))
    image.close()
    return pix
x_test = []
for im_path in images:
    x_test.append(Reformat_Image(im_path, 128, 128))
x_test = np.asarray(x_test).reshape((-1, 128*128*3))
model1 = load_model('mlp_xinyu_yao1.hdf5')
y_pred1 = model1.predict(x_test)
x_test = []
for im_path in images:
    x_test.append(Reformat_Image(im_path, 128, 128, flip=True))
x_test = np.asarray(x_test).reshape((-1, 128*128*3))
model2 = load_model('mlp_xinyu_yao2.hdf5')
y_pred2 = model2.predict(x_test)
x_test = []
for im_path in images:
    x_test.append(Reformat_Image(im_path, 128, 128, flip=True, correction=True))
x_test = np.asarray(x_test).reshape((-1, 128*128*3))
model3 = load_model('mlp_xinyu_yao3.hdf5')
y_pred3 = model3.predict(x_test)
y_pred = np.argmax(np.add(np.add(y_pred1, y_pred2), y_pred3), axis=1)
return y_pred, model1, model2, model3
```