Machine Learning Final Exam Report
Haekal Rizky Yulianto - 17/415900/PA/18169

Architecture of MLP model

Input layer with 76800 neurons and using sigmoid activation function

Hidden layer with 64 neurons and using sigmoid activation function

Output layer with 3 neurons (3 class) and using softmax activation function

Architecture of CNN model

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 240, 320, 32)      2432
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 120, 160, 32)      0
_________________________________________________________________
flatten (Flatten)            (None, 614400)            0
_________________________________________________________________
dense (Dense)                (None, 32)                19660832
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 99
=================================================================
Total params: 19,663,363
Trainable params: 19,663,363
Non-trainable params: 0
_________________________________________________________________
```

Steps for MLP model from scratch

1. Prepare the data and determine daisy, rose, and tulip path

Data preparation

Check directory

In [2]:
```
import os
print(os.listdir("/Users/Haekal/Documents/ML/input"))
print(os.listdir("/Users/Haekal/Documents/ML/input/flowers/flowers"))
```

```
['.DS_Store', 'flowers']
['.DS_Store', 'daisy', 'rose', 'tulip', 'dandelion', 'sunflower']
```

Read Image

In [3]:
```
trainLabels = [] # Image label array
data = [] # Image data array

size = 320, 240 # Resize image

def readImages(flowerPath, folder):

    imagePaths = []
    for file in os.listdir(flowerPath):
        if file.endswith("jpg"):
            imagePaths.append(flowerPath + file)
            trainLabels.append(folder)
            img = cv2.imread((flowerPath + file), 0)
            im = cv2.resize(img, size)
            data.append(im)

    return imagePaths
```

In [4]:
```
daisy_path = "/Users/Haekal/Documents/ML/input/flowers/flowers/daisy/"
rose_path = "/Users/Haekal/Documents/ML/input/flowers/flowers/rose/"
tulip_path = "/Users/Haekal/Documents/ML/input/flowers/flowers/tulip/"
```

2. Preprocess the data

- Normalize image data

Data preprocessing

Normalize image data

In [7]:
```
X = np.array(data)
X = X.astype('float32') / 255.0
```

- One hot encoding for target/label variable

In [10]:
```
def to_numeric(x):
    if x == 'daisy':
        return '0'
    if x == 'rose':
        return '1'
    if x == 'tulip':
        return '2'
```

In [11]:
```
df['cat'] = df['category'].apply(to_numeric)
```

In [12]:
```
y = pd.get_dummies(df.cat, prefix='C')
y
```

Out[12]:

	C_0	C_1	C_2
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
...	...	...	...
295	0	0	1
296	0	0	1
297	0	0	1
298	0	0	1
299	0	0	1

300 rows × 3 columns

- Transform image data into 2D shape

Transform into 2D shape

In [13]:
```
print(X.shape)
print(y.shape)
```

```
(300, 240, 320)
(300, 3)
```

In [14]:
```
y = np.array(y)
```

In [15]:
```
X = X.reshape(X.shape[0], X.shape[1] * X.shape[2])
Y = y
```

In [16]:
```
print(X.shape)
print(Y.shape)
```

```
(300, 76800)
(300, 3)
```

- Split into training and testing, and transpose the data

Split the data for training and testing

In [17]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
numberOfTrain = X_train.shape[0]
numberOfTest = X_test.shape[0]

print('Train size : ', X_train.shape[0], 'Test size : ', X_test.shape[0])
```

```
Train size :  240 Test size :  60
```

Transpose the data

In [18]:
```
X_train = X_train.T
X_test = X_test.T
Y_train = Y_train.T
Y_test = Y_test.T
X = X.T
Y = Y.T
```

In [19]:
```
print(X_train.shape)
print(Y_train.shape)
```

```
(76800, 240)
(3, 240)
```

3. Model training

- Declare required function for MLP model

In [20]:
```
def sigmoid(z):
    s = 1. / (1. + np.exp(-z))
    return s

def compute_loss(Y, Y_hat):

    L_sum = np.sum(np.multiply(Y, np.log(Y_hat)))
    m = Y.shape[1]
    L = -(1./m) * L_sum

    return L

def feed_forward(X, params):

    cache = {}

    cache["Z1"] = np.matmul(params["W1"], X) + params["b1"]
    cache["A1"] = sigmoid(cache["Z1"]) # Sigmoid activation function
    cache["Z2"] = np.matmul(params["W2"], cache["A1"]) + params["b2"]
    cache["A2"] = np.exp(cache["Z2"]) / np.sum(np.exp(cache["Z2"]), axis=0) # Softmax activation function

    return cache

def back_propagate(X, Y, params, cache):

    dZ2 = cache["A2"] - Y
    dW2 = (1./m_batch) * np.matmul(dZ2, cache["A1"].T)
    db2 = (1./m_batch) * np.sum(dZ2, axis=1, keepdims=True)

    dA1 = np.matmul(params["W2"].T, dZ2)
    dZ1 = dA1 * sigmoid(cache["Z1"]) * (1 - sigmoid(cache["Z1"]))
    dW1 = (1./m_batch) * np.matmul(dZ1, X.T)
    db1 = (1./m_batch) * np.sum(dZ1, axis=1, keepdims=True)

    grads = {"dW1": dW1, "db1": db1, "dW2": dW2, "db2": db2}

    return grads
```

- Declare function to calculate accuracy

Declare function to calculate accuracy

In [21]:
```
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0
```

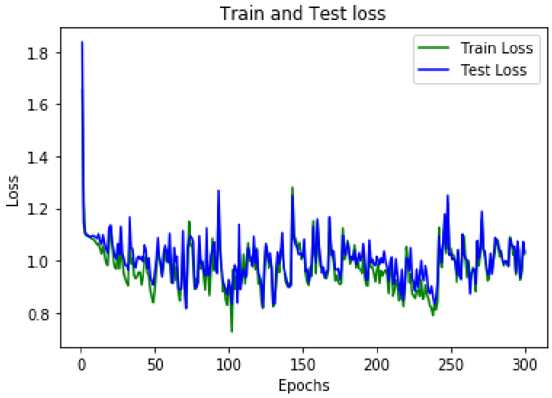
- Weight and hyperparameter initialization (Uses 128 batch size for fast computing)

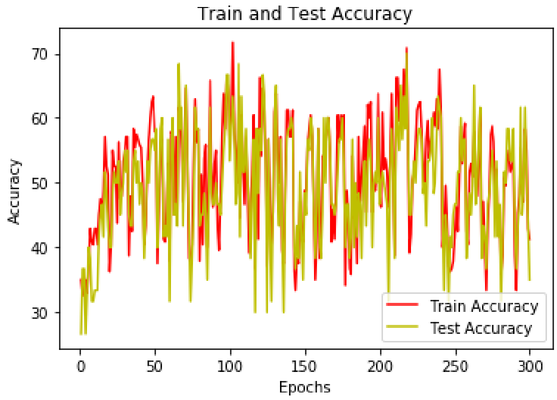
In [22]:
```
np.random.seed(138) # Random seed for reproducibility
m = numberOfTrain # Number of training data

# Hyperparameters
n_x = X_train.shape[0] # Number of neuron in input layer = 320 X 240 = 76800
n_h = 64 # Number of neuron in hidden layer
num_class = 3 # Number of class
learning_rate = 0.1
beta = 0.5
batch_size = 128
batches = -(-m // batch_size)

# Initialization
params = { "W1": np.random.randn(n_h, n_x) * np.sqrt(1. / n_x),
           "b1": np.zeros((n_h, 1)) * np.sqrt(1. / n_x),
           "W2": np.random.randn(num_class, n_h) * np.sqrt(1. / n_h),
           "b2": np.zeros((num_class, 1)) * np.sqrt(1. / n_h) }

V_dW1 = np.zeros(params["W1"].shape)
V_db1 = np.zeros(params["b1"].shape)
V_dW2 = np.zeros(params["W2"].shape)
V_db2 = np.zeros(params["b2"].shape)

ep = 300 # Number of epoch
epoch = list(range(1,ep+1))
train_history = []
test_history = []
train_accuracy = []
test_accuracy = []
```

- Train for ep = 300 and test the model

```
for i in range(ep):

    for j in range (int(batches)):

        begin = j * batch_size
        end = min(begin + batch_size, X_train.shape[1] - 1)
        m_batch = end - begin

        cache = feed_forward(X, params)
        grads = back_propagate(X, Y, params, cache)

        # Update parameters
        V_dW1 = (beta * V_dW1 + (1. - beta) * grads["dW1"])
        V_db1 = (beta * V_db1 + (1. - beta) * grads["db1"])
        V_dW2 = (beta * V_dW2 + (1. - beta) * grads["dW2"])
        V_db2 = (beta * V_db2 + (1. - beta) * grads["db2"])

        params["W1"] = params["W1"] - learning_rate * V_dW1
        params["b1"] = params["b1"] - learning_rate * V_db1
        params["W2"] = params["W2"] - learning_rate * V_dW2
        params["b2"] = params["b2"] - learning_rate * V_db2

    # Train accuracy
    cache = feed_forward(X_train, params)
    trainpredictions = np.argmax(cache["A2"], axis=0)
    trainlabels = np.argmax(Y_train, axis=0)
    train_acc = accuracy_metric(trainlabels, trainpredictions)
    train_accuracy.append(train_acc)

    # Train loss
    train_cost = compute_loss(Y_train, cache["A2"])
    train_history.append(train_cost)

    # Test accuracy
    cache = feed_forward(X_test, params)
    testpredictions = np.argmax(cache["A2"], axis=0)
    testlabels = np.argmax(Y_test, axis=0)
    test_acc = accuracy_metric(testlabels, testpredictions)
```

4. Model Evaluation





Average train accuracy = 50.79
Average test accuracy = 49.29

Average train loss = 0.98
Average test loss = 1.00

Steps for CNN model

1. Prepare the data and only include daisy, rose tulip path

In [29]:
```
# Set the path of the input folder
data_path = "/Users/Haekal/Documents/ML/input/flowers/flowers/"

# List out the directories inside the main input folder
folders = os.listdir(data_path)
folders.remove('.DS_Store')
folders.remove('dandelion')
folders.remove('sunflower')

print(folders)
```

```
['daisy', 'rose', 'tulip']
```

In [30]:
```
# Resize image
size1 = 320
size2 = 240

# lists to store data
data = []
label = []

for folder in folders:
    for file in os.listdir(os.path.join(data_path,folder)):
        if file.endswith("jpg"):
            img = cv2.imread(os.path.join(data_path,folder,file))
            img = cv2.resize(img, (size1,size2))
            data.append(img)
            label.append(folder)
        else:
            continue
```

2. Preprocess the data

- Split, one hot encoding, rescale, and assign train and validation datagen

Split the data for training and testing

In [31]:
```
# Transforming into numpy array
data = np.array(data)
label = np.array(label)

# Split dataset into train and test sets
train_data, test_data, train_label, test_label = train_test_split(data, label, test_size=0.2, random_state=25)

# Get the categories/classes
label_categories = np.unique(label)
test_label_names = test_label
```

One hot encoding for target/label variable

In [32]:
```
# Transforming object categories into numerical
encoder = LabelEncoder()

train_label = encoder.fit_transform(train_label).astype(int)
test_label = encoder.fit_transform(test_label).astype(int)
```

Normalize image data

In [33]:
```
datagen_train = ImageDataGenerator(rescale=1./255)
datagen_valid = ImageDataGenerator(rescale=1./255)

datagen_train.fit(train_data)
datagen_valid.fit(test_data)
```

3. Model training

- Build model

Build model

In [36]:
```
# CNN architecture
classifier = Sequential([
    Conv2D(32, (5,5), padding='same', activation='relu', input_shape=(size2, size1,3)),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(32, activation='relu'),
    Dense(3, activation='softmax')
])

# Visualize model summary
classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 240, 320, 32)      2432
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 120, 160, 32)      0
_________________________________________________________________
flatten (Flatten)            (None, 614400)            0
_________________________________________________________________
dense (Dense)                (None, 32)                19660832
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 99
=================================================================
Total params: 19,663,363
Trainable params: 19,663,363
Non-trainable params: 0
_________________________________________________________________
```

- Compile, train, and validate model (Uses 128 batch size for fast computing)

Compile model

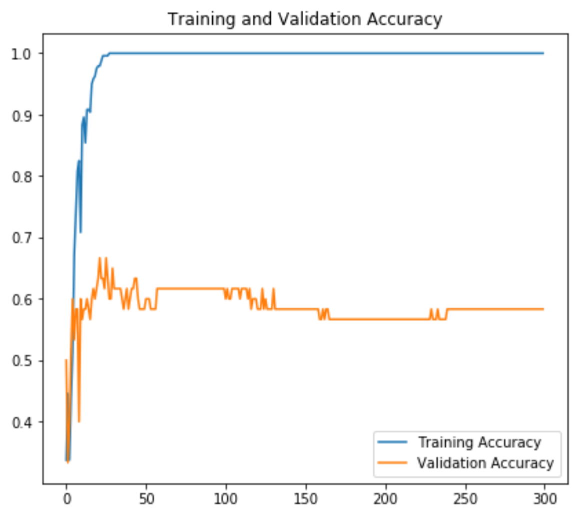
```
In [37]: classifier.compile(optimizer='Nadam', loss='sparse_categorical_crossentropy', metrics=["accuracy"])
```

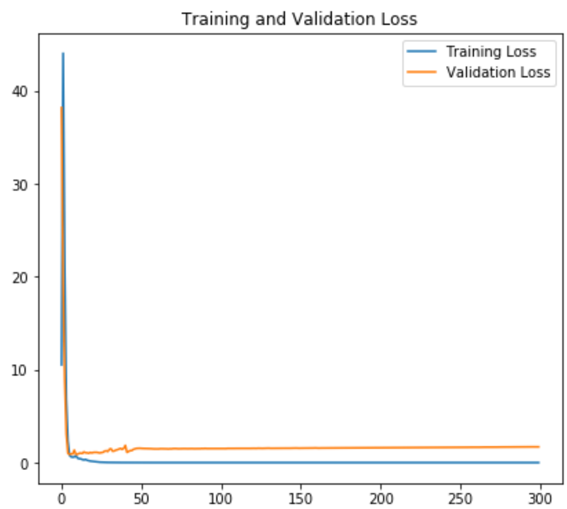
Train and validate model

```
In [38]: history = classifier.fit(datagen_train.flow(train_data, train_label, batch_size = 128),
                                  validation_data = datagen_valid.flow(test_data, test_label, batch_size = 128),
                                  epochs=300)
```

```
curacy: 0.5833
Epoch 287/300
2/2 [==============================] - 9s 4s/step - loss: 4.2836e-05 - accuracy: 1.0000 - val_loss: 1.6803 - val_ac
curacy: 0.5833
Epoch 288/300
2/2 [==============================] - 9s 4s/step - loss: 4.2546e-05 - accuracy: 1.0000 - val_loss: 1.6882 - val_ac
curacy: 0.5833
Epoch 289/300
2/2 [==============================] - 9s 4s/step - loss: 4.2242e-05 - accuracy: 1.0000 - val_loss: 1.6899 - val_ac
curacy: 0.5833
Epoch 290/300
2/2 [==============================] - 9s 5s/step - loss: 4.1975e-05 - accuracy: 1.0000 - val_loss: 1.6879 - val_ac
curacy: 0.5833
Epoch 291/300
2/2 [==============================] - 9s 4s/step - loss: 4.1630e-05 - accuracy: 1.0000 - val_loss: 1.6870 - val_ac
curacy: 0.5833
Epoch 292/300
2/2 [==============================] - 9s 4s/step - loss: 4.1348e-05 - accuracy: 1.0000 - val_loss: 1.6930 - val_ac
curacy: 0.5833
Epoch 293/300
```

4. Model Evaluation





Average training accuracy : 0.9831806
Average validation accuracy : 0.58683336

Average training loss : 0.30622097028021217
Average validation loss : 1.7382829558849335

Conclusion

A Multilayer Perceptron (MLP) model with 1 hidden layer architecture produces a faster classification result with ~50 % accuracy and 1.0 loss for both training and testing.

Convolutional Neural Network (CNN) model with 1 pair of Conv-Pool layers and 1 hidden layer architecture produces a longer classification result with ~98% accuracy for training and ~59% accuracy for validation, ~0.3 loss for training and ~1.7 loss for validation.

This MLP architecture gives a reasonable accuracy and loss because it has no convolution layer(s) like CNN model can have to distinguish features, however loss and accuracy are fluctuating in this MLP architecture as the model seems to be confused on what makes which images belongs to which flower class, since MLP doesn't have feature extraction layer(s).

This CNN architecture successfully distinguishes the training images well (too well), but cannot generalize well in the validation images. Overfitting happens in the CNN model because there is no dropout layer(s) and regularization(L1/L2) applied in this CNN architecture model.

My feedback and evaluation for better accuracy and loss for this data is by using a pre-trained model that has achieved good results for this data and applied it to our model with transfer learning, then by tuning CNN’s hyperparameter and configuring the best layer architecture that is suitable for this data. Feeding huge data for training with variative position and angle for each class will also produce higher accuracy for this classification problem.

Notes

For taking 100 images from each daisy, tulip, and rose folder, I manually select the first 100 images for each folder of daisy, tulip, and rose.