Credit Hours System
HEMN451: Medical Pattern Recognition

Cairo University
Faculty of Engineering

Final Project

Submitted to: Eng. Peter Farag

Names	IDs
Doaa Sherif	1170122
Salma Hazem	1170425
Yara Wael	1170431

Problem Definition

The problem is that we have to classify different cell organelles which is important when characterizing newly discovered genes or genes with an unknown function.

Solution

We will build a Convolutional Neural Network (CNN) to build an image classifier.

Dataset

We used the dataset from (https://ome.grc.nia.nih.gov/iicbu2008/hela/index.html).

The images include 10 organelles, we divided them manually to be 70% for training, 15% for validation, and 15% for testing. We classified them as following:

Classes	Labels	Number of Training Images	Number of Testing Images	Number of Validation Images
actin	0	70	15	14
dna	1	62	13	13
endosome	2	64	14	13
er	3	60	13	13
golgia	4	61	13	13
golgpp	5	60	13	12
lysosome	6	60	13	12
microtubules	7	64	14	13
mitochondria	8	52	11	10
nucleolus	9	56	12	12

Methods and Algorithms

Attempted Models

First Model

```
cnn_model = ks.models.Sequential()
cnn_model.add(ks.layers.Conv2D(96, (3,3), activation= 'relu', padding= 'same', input_shape= (32, 32, 3)))
cnn_model.add(ks.layers.Dropout(0.2))

cnn_model.add(ks.layers.Conv2D(96, (3,3), activation= 'relu', padding= 'same'))
cnn_model.add(ks.layers.Conv2D(96, (3,3), activation= 'relu', padding= 'same', strides= 2))
cnn_model.add(ks.layers.Dropout(0.5))

cnn_model.add(ks.layers.Conv2D(192, (3,3), activation= 'relu', padding= 'same'))
cnn_model.add(ks.layers.Conv2D(192, (3,3), activation= 'relu', padding= 'same'))
cnn_model.add(ks.layers.Conv2D(192, (3,3), activation= 'relu', padding= 'same', strides= 2))
cnn_model.add(ks.layers.Dropout(0.5))

cnn_model.add(ks.layers.Conv2D(192, (3,3), padding= 'same'))
cnn_model.add(ks.layers.Activation('relu'))
cnn_model.add(ks.layers.Conv2D(192, (1,1), padding= 'valid'))
cnn_model.add(ks.layers.Activation('relu'))
cnn_model.add(ks.layers.Conv2D(10, (1,1), padding= 'valid'))

cnn_model.add(ks.layers.GlobalAveragePooling2D())
cnn_model.add(ks.layers.Activation('softmax'))
```

Input size of image: 32x32x3

- Layer 1: Convolution with 96 filters 3x3
- Layer 2: Dropout layer with rate 0.2
- Layers 3 and 4: Convolution with 96 filters 3x3
- Layer 5: Dropout layer with rate 0.5
- Layers 6, 7, 8: Convolution with 192 filters 3x3
- Layer 9: Dropout layer with rate 0.5
- Layers 10, 12: Convolution with 192 filters
- Layers 11, 13: Activation relu
- Layer 14: Convolution with 10 filters
- Layer 15: Global Average Pooling
- Layer 16: Activation function Softmax

This model has accuracy of range 60% and takes long time

Second Model

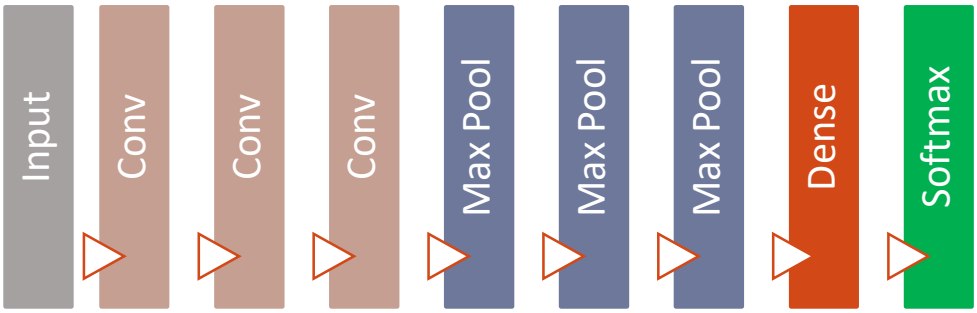
Input size of image: 28x28x1

```
cnn_model = ks.models.Sequential()
cnn_model.add(ks.layers.Conv2D(32, (5,5), activation= 'relu', input_shape= (28, 28, 1), kernel_initializer='he_uniform',
                               name='Convolutional_layer_1', kernel_regularizer=l2(0.0005)))
cnn_model.add(ks.layers.Conv2D(32, (5,5), activation= 'relu',kernel_initializer='he_uniform',
                               name='Convolutional_layer_2', kernel_regularizer=l2(0.0005)))
cnn_model.add(ks.layers.MaxPooling2D((2,2), name= 'Maxpooling_2D_2'))
cnn_model.add(ks.layers.Dropout(0.2))

cnn_model.add(ks.layers.Conv2D(64, (3,3), activation= 'relu', kernel_initializer='he_uniform',
                               name='Convolutional_layer_3',  kernel_regularizer=l2(0.0005)))
cnn_model.add(ks.layers.Conv2D(64, (3,3), activation= 'relu',kernel_initializer='he_uniform',
                               name='Convolutional_layer_4', kernel_regularizer=l2(0.0005)))
cnn_model.add(ks.layers.MaxPooling2D((2,2), name= 'Maxpooling_2D_3'))
cnn_model.add(ks.layers.Dropout(0.3))

cnn_model.add(ks.layers.Flatten(name= 'Flatten'))
cnn_model.add(ks.layers.Dense(128, activation='relu',kernel_initializer='he_uniform', name='Hidden_layer'))
cnn_model.add(ks.layers.Dropout(0.4))
cnn_model.add(ks.layers.Dense(10, activation='softmax', name='Output_layer'))
```

- Layers 1 and 2: Convolution with 32 filters 5x5, updating weights using l2
- Layer 3: Max Pooling with size 2x2
- Layer 4: Dropout layer with rate 0.2
- Layers 5 and 6: Convolution with 64 filters 3x3, updating weights using l2
- Layer 7: Max Pooling with size 2x2
- Layer 8: Dropout layer with rate 0.3
- Layer 9: Flatten
- Layer 10: Dense with activation function relu 128
- Layer 11: Dropout layer with rate 0.4
- Layer 12: Activation function Softmax

At first it was about 76% accuracy with small time period, around 2 seconds per epochs, but then it started to decrease significantly

Final Layers Used



Input size of image: 28x28x1

- Layers 1 and 2: Convolution with 64 filters 5x5
- Layer 3: Convolution with 128 filters 3x3
- Layers 4, 5 and 6: Max Pooling with size 2x2
- Layer 7: Flattening
- Layer 8: Dense with activation function relu 128
- Layer 9: Dense with activation function Softmax with 10 categories

Implementation

1. Imports:
    - PyQt5
    - sys
    - PIL
    - cv2
    - tensorflow
    - os
    - numpy
2. Reading images in grayscale and labelling them
3. Data augmentation: Flipping and Rotation
4. Normalization of images, divided by 255
5. Training the CNN with training dataset
6. Optimizing learning rate using “Adam” algorithm
7. Model Fitting
    - Number of epochs is 40
8. Prediction

Results

Maximum results:

Type	Accuracy
Training	99%
Validation	75%
Testing	78%

```
train_loss, train_accuracy = cnn_model.evaluate(train_images, train_labels)
print('Train Accuracy{} '.format(round(float(train_accuracy), 2)))
```

```
57/57 [==============================] - 1s 26ms/step - loss: 0.0333 - accuracy: 0.9923
Train Accuracy0.99
```

```
validation_loss, validation_accuracy = cnn_model.evaluate(validation_images, validation_labels)
print('Validation Accuracy{} '.format(round(float(validation_accuracy), 2)))
```

```
12/12 [==============================] - 0s 24ms/step - loss: 1.7048 - accuracy: 0.7547
Validation Accuracy0.75
```

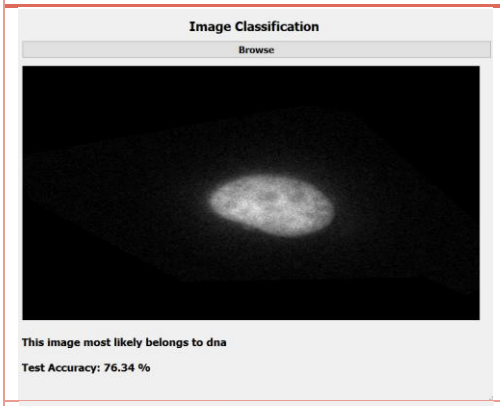
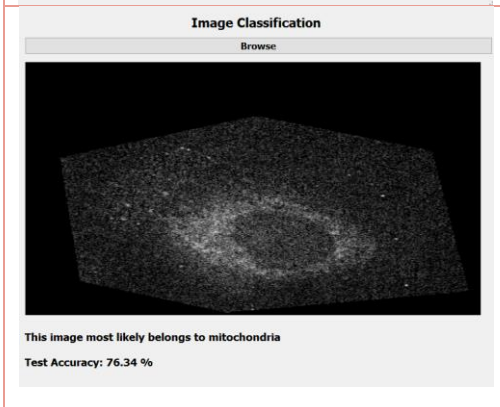
```
test_loss, test_accuracy = cnn_model.evaluate(test_images, test_labels)
print('Test Accuracy {} '.format(round(float(test_accuracy), 2)))
```

```
5/5 [==============================] - 0s 18ms/step - loss: 1.4708 - accuracy: 0.7786
Test Accuracy 0.78
```

Random Results

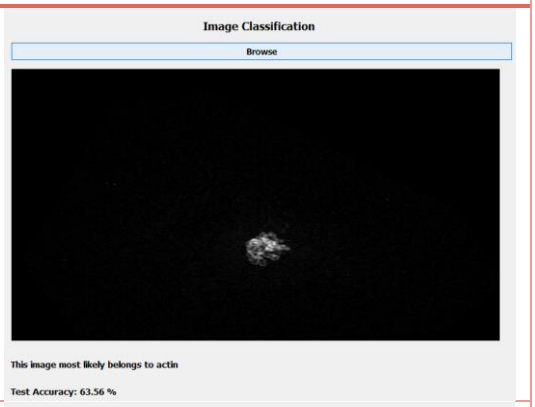
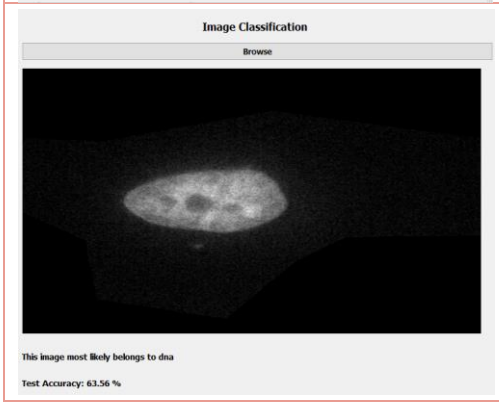
Result 1

```
In [89]: train_loss, train_accuracy = cnn_model.evaluate(train_images, train_labels)
         print('Train Accuracy{} '.format(round(float(train_accuracy), 2)))

         57/57 [==============================] - 1s 14ms/step - loss: 3.8088e-04 - accuracy: 1.0000
         Train Accuracy1.0

In [90]: validation_loss, validation_accuracy = cnn_model.evaluate(validation_images, validation_labels)
         print('Validation Accuracy{} '.format(round(float(validation_accuracy), 2)))

         12/12 [==============================] - 0s 12ms/step - loss: 2.5908 - accuracy: 0.7360
         Validation Accuracy0.74

In [91]: test_loss, test_accuracy = cnn_model.evaluate(test_images, test_labels)
         print('Test Accuracy {} '.format(round(float(test_accuracy), 2)))

         5/5 [==============================] - 0s 10ms/step - loss: 2.0543 - accuracy: 0.7634
         Test Accuracy 0.76
```

True Classification	False Classifications
	
	

Result 2

```
In [89]: train_loss, train_accuracy = cnn_model.evaluate(train_images, train_labels)
         print('Train Accuracy{} '.format(round(float(train_accuracy), 2)))

         57/57 [==============================] - 1s 14ms/step - loss: 3.8088e-04 - accuracy: 1.0000
         Train Accuracy1.0

In [90]: validation_loss, validation_accuracy = cnn_model.evaluate(validation_images, validation_labels)
         print('Validation Accuracy{} '.format(round(float(validation_accuracy), 2)))

         12/12 [==============================] - 0s 12ms/step - loss: 2.5908 - accuracy: 0.7360
         Validation Accuracy0.74

In [91]: test_loss, test_accuracy = cnn_model.evaluate(test_images, test_labels)
         print('Test Accuracy {} '.format(round(float(test_accuracy), 2)))

         5/5 [==============================] - 0s 10ms/step - loss: 2.0543 - accuracy: 0.7634
         Test Accuracy 0.76
```

True Classifications	False Classification
	
	

Result 3

```
train_loss, train_accuracy = cnn_model.evaluate(train_images, train_labels)
print('Train Accuracy{} '.format(round(float(train_accuracy), 2)))
```

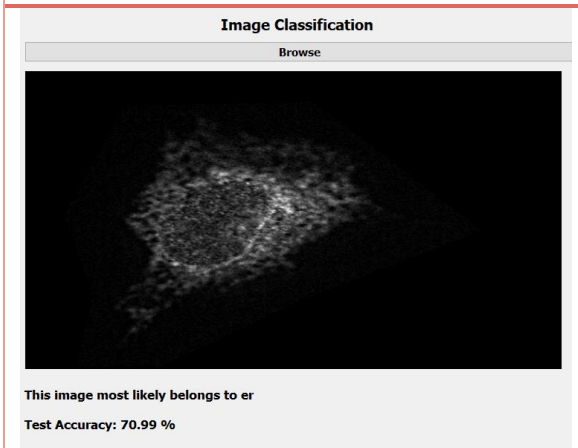
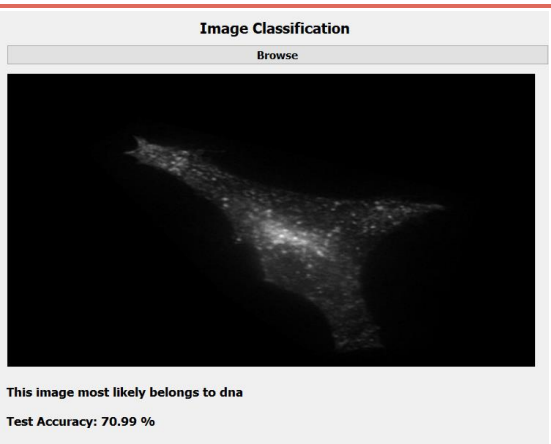
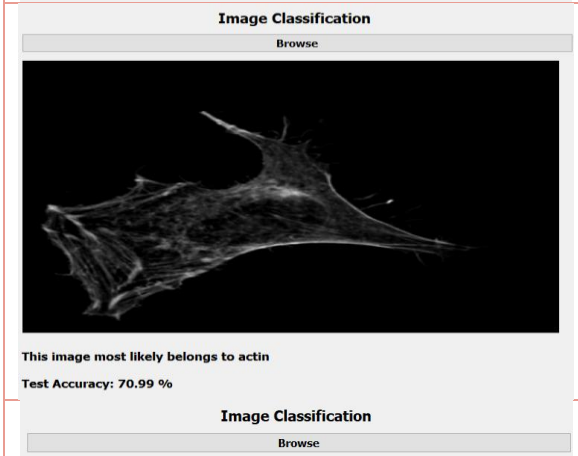
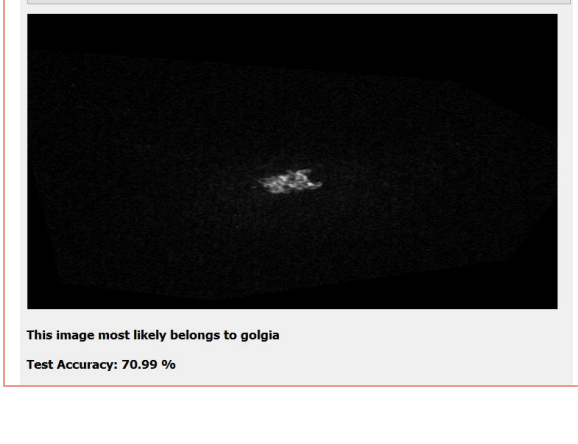
```
57/57 [==============================] - 1s 25ms/step - loss: 0.0377 - accuracy: 0.9917
Train Accuracy0.99
```

```
validation_loss, validation_accuracy = cnn_model.evaluate(validation_images, validation_labels)
print('Validation Accuracy{} '.format(round(float(validation_accuracy), 2)))
```

```
12/12 [==============================] - 0s 24ms/step - loss: 1.4314 - accuracy: 0.7280
Validation Accuracy0.73
```

```
test_loss, test_accuracy = cnn_model.evaluate(test_images, test_labels)
print('Test Accuracy {} '.format(round(float(test_accuracy), 2)))
```

```
5/5 [==============================] - 0s 17ms/step - loss: 1.6092 - accuracy: 0.7099
Test Accuracy 0.71
```

True Classifications	False Classification
	
	
	

Time

Each epochs takes from 6 to 7 seconds. The whole training takes about 4 minutes

Open Source Model

Used tensorflow_hub

Input size of image: 224x224x3

```
feature_extractor_model = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4"

pretrained_model_without_top_layer = hub.KerasLayer(
    feature_extractor_model, input_shape=(224, 224, 3), trainable=False)
```

```
model = tf.keras.Sequential([pretrained_model_without_top_layer, tf.keras.layers.Dense(10)])

model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer_5 (KerasLayer)   (None, 1280)              2257984
_________________________________________________________________
dense_7 (Dense)              (None, 10)                12810
=================================================================
Total params: 2,270,794
Trainable params: 12,810
Non-trainable params: 2,257,984
_________________________________________________________________
```

```
model.compile(optimizer= "adam", loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['acc'])

model.fit(train_images, train_labels, epochs= 5)
```

Random Results

```
model.evaluate(train_images, train_labels)
```

```
57/57 [==============================] - 29s 511ms/step - loss: 0.1212 - acc: 0.9692

[0.12119679898023605, 0.9691969156265259]
```

```
model.evaluate(test_images, test_labels)
```

```
5/5 [==============================] - 2s 320ms/step - loss: 0.6530 - acc: 0.8244

[0.6529501676559448, 0.8244274854660034]
```

```
model.evaluate(train_images, train_labels)
```

```
57/57 [==============================] - 52s 914ms/step - loss: 0.1212 - acc: 0.9736

[0.1211928129196167, 0.9735973477363586]
```

```
model.evaluate(test_images, test_labels)
```

```
5/5 [==============================] - 2s 309ms/step - loss: 0.6889 - acc: 0.7863

[0.688930094242096, 0.7862595319747925]
```

Time

Each epochs takes about 30 seconds. Using 5 epochs, takes about 2.5 minutes.

Comparison

The maximum accuracy of the open source model is 84%, which is higher than ours, 78%. However, the time of each epoch in open source model is higher than ours.

The open source reaches higher accuracy using less number of epochs.