Credit Hours System

HEMN451 – Medical Pattern Recognition

Cairo University

Faculty of Engineering

Project Report

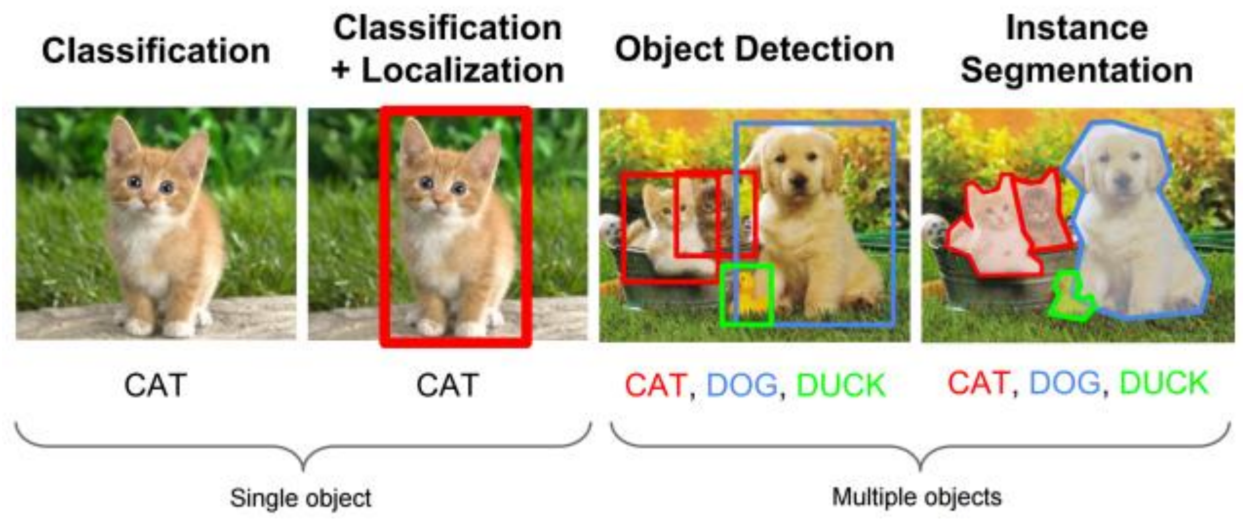
Team Members:

• Hoda Hossam	[1170156]
• Mohammed Nader	[1170366]
• Tasneem Adel	[1162182]
• Youssef Moataz	[1170275]

What is Image Classification:

The intent of the classification process is to categorize all pixels in a digital image into one of several land covers classes, or "themes". This categorized data may then be used to produce thematic maps of the land cover present in an image. Normally, multispectral data are used to perform the classification and, indeed, the spectral pattern present within the data for each pixel is used as the numerical basis for categorization (Lillesand and Kiefer, 1994). The objective of image classification is to identify and portray, as a unique gray level (or color), the features occurring in an image in terms of the object or type of land cover these features actually represent on the ground.

Image classification is perhaps the most important part of digital image analysis. It is very nice to have a "pretty picture" or an image, showing a magnitude of colors illustrating various features of the underlying terrain, but it is quite useless unless to know what the colors mean. (PCI, 1997). Two main classification methods are Supervised Classification and Unsupervised Classification.

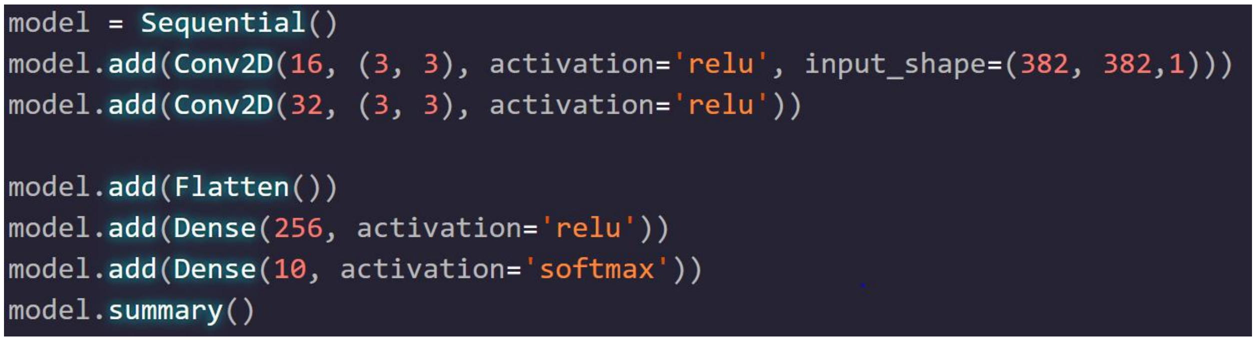


Methods and Algorithms ‘Hela-2D’:

We started working on the Hela2d cells dataset, this dataset contains almost 800 ‘.tif’ black and white images, with 10 different classes, we splitted the data into Train, Test and validation sets with ratios 70,15,15 respectively using the sklearn library.

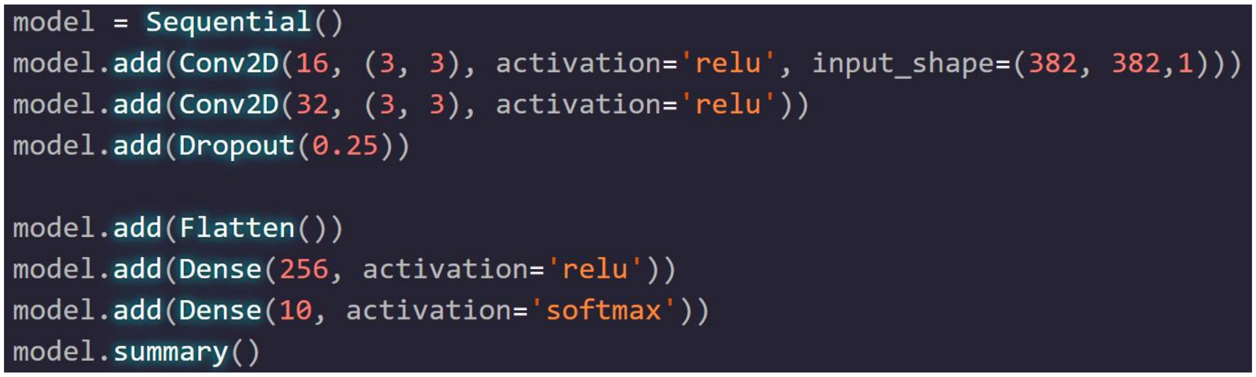
We built many CNN models for this dataset starting with the simplest one:

Two CNN layers, the ultimate purpose of this layer is to receive a feature map, Feature detection is based on ‘scanning’ the input with the filter of a given size and applying matrix computations in order to derive a feature map. we will use ReLU activation function that returns 0 for every negative value in the input image while it returns the same value for every positive value.



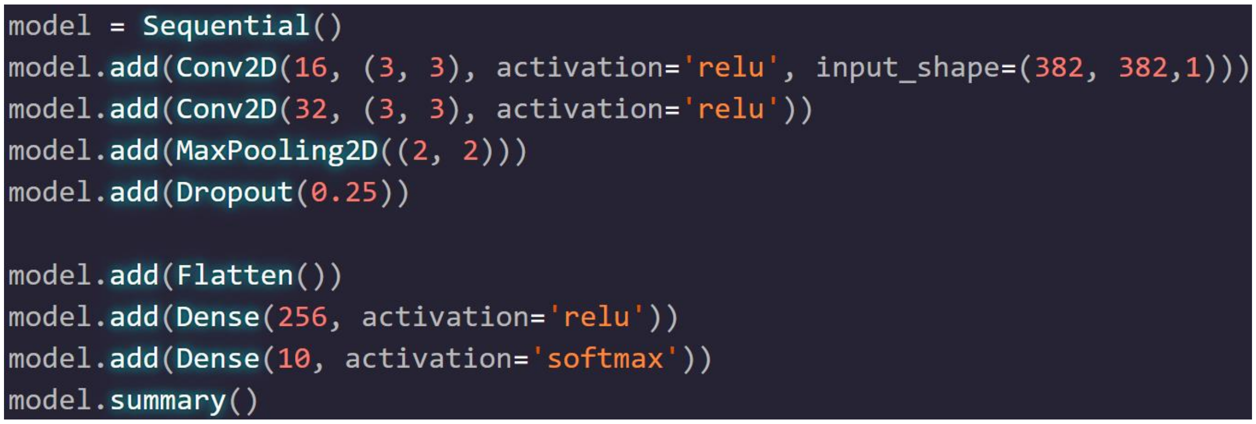
```
model = Sequential()
model.add(Conv2D(16, (3, 3), activation='relu', input_shape=(382, 382,1)))
model.add(Conv2D(32, (3, 3), activation='relu'))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

This model wasn’t very efficient as it had 97% training accuracy but only 52% validation and 57% test accuracy, so it’s so obvious that this model was overfitted to the data and we read online that there is a layer called dropout, that ignore some units (neurons) during the training phase of certain set of neurons which is chosen at random. By “ignoring”, I mean these units are not considered during a particular forward or backward pass to prevent over-fitting.



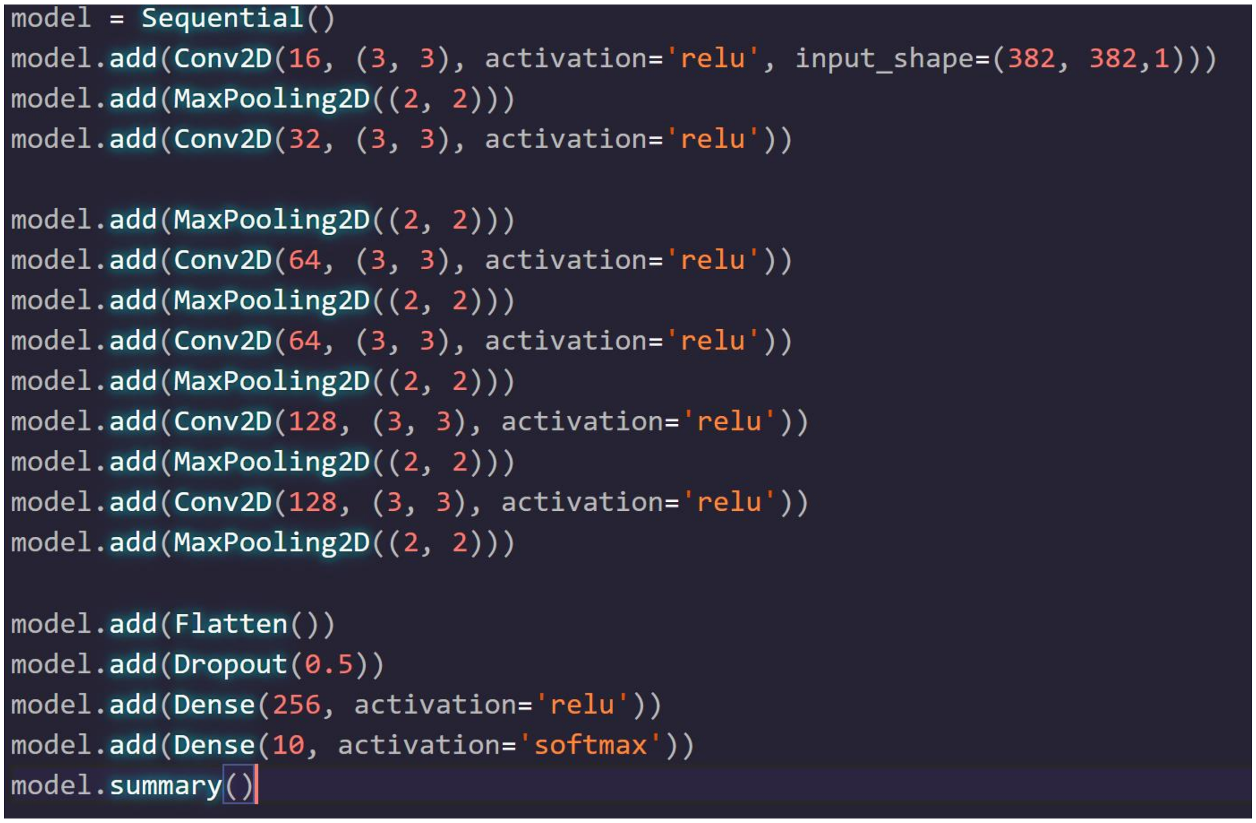
```
model = Sequential()
model.add(Conv2D(16, (3, 3), activation='relu', input_shape=(382, 382,1)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

This model was a little bit better, with 96% training accuracy and 66% validation and 69% testing accuracy, but still it’s not that good so we added a maxPoll layer which is a new layer added after the convolutional layer. Specifically, after a nonlinearity (ReLU) has been applied to the feature maps output by a convolutional layer, Max Pooling which Calculates the maximum value for each patch of the feature map.

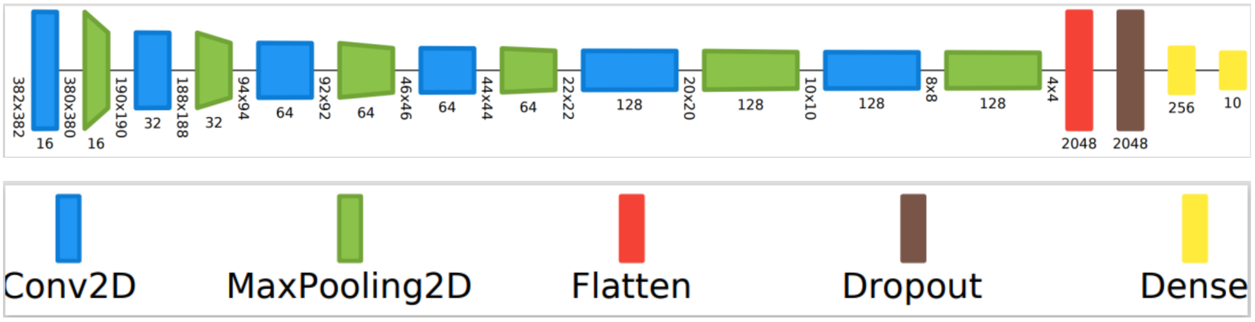


```
model = Sequential()
model.add(Conv2D(16, (3, 3), activation='relu', input_shape=(382, 382,1)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

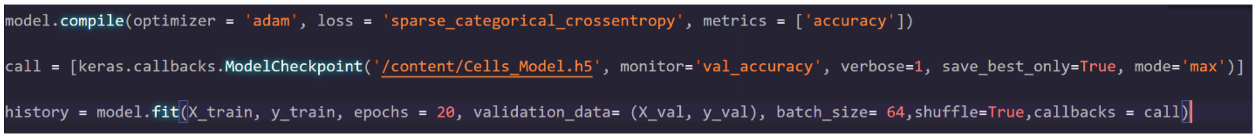
This step didn't show much improvement in the accuracy so we decided to start adding more convolution layers and increase the dropout value, from this point we started adding new layers to the model based on some rules to get the best results, after each convolution layer there is a maxPool layer, and number of filters per each layer is larger than the layer before it to detect high-level features in the images, and finally we settled on this final model.



```
model = Sequential()
model.add(Conv2D(16, (3, 3), activation='relu', input_shape=(382, 382,1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu'))

model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
```



To each model we used ‘adam’ optimizer and sparse_categorical_crossentropy loss function, we also added a callback function to save the best model during the training phase based on



```
model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])

call = [keras.callbacks.ModelCheckpoint('/content/Cells_Model.h5', monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')]

history = model.fit(X_train, y_train, epochs = 20, validation_data= (X_val, y_val), batch_size= 64,shuffle=True,callbacks = call)
```

the validation accuracy and we saved the model history to plot the loss and the accuracy

This model has 98% training accuracy and 84% validation accuracy and 90% testing accuracy.

Finally, we made the model predict each image’s label and compared it with the real labels.

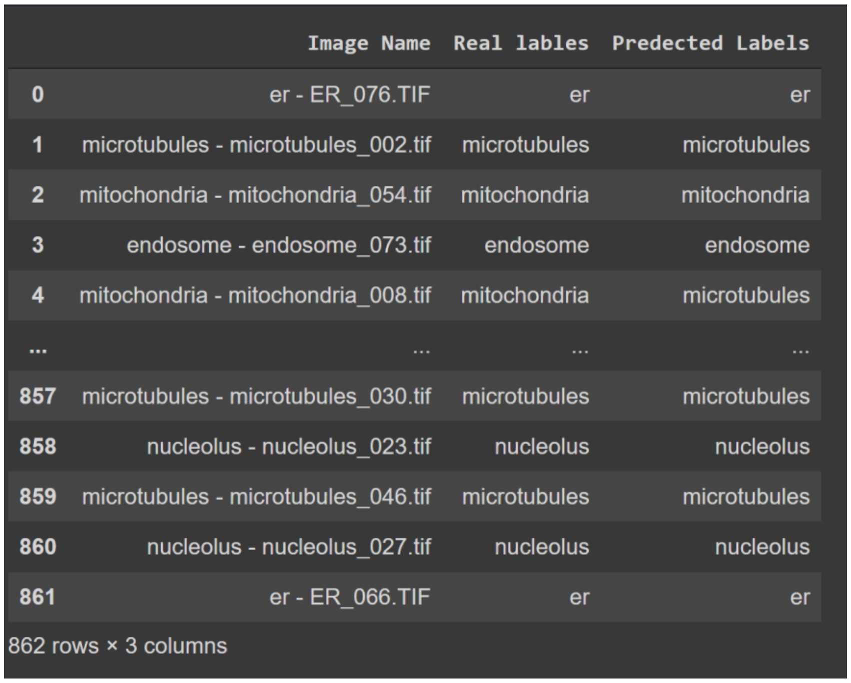
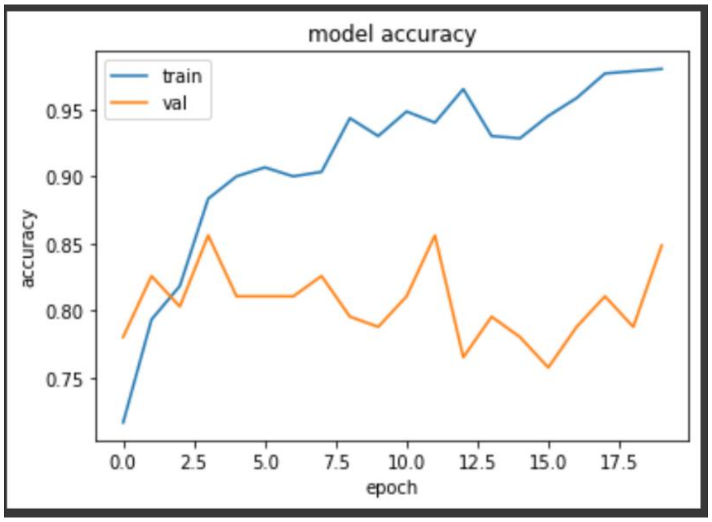
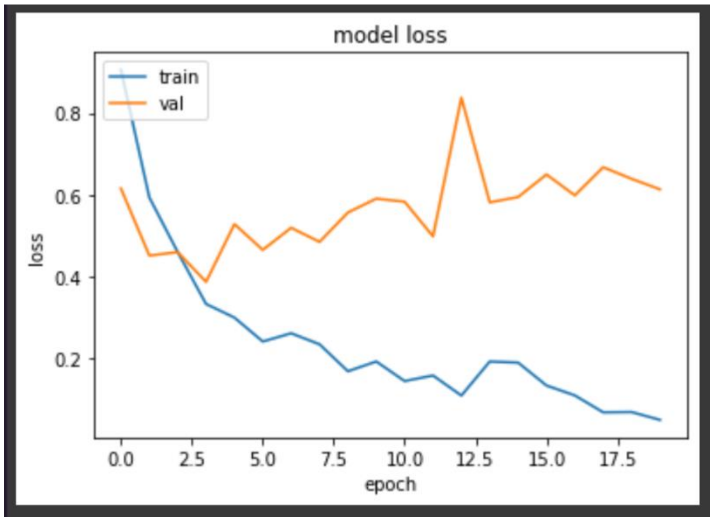


	Image Name	Real lables	Predected Labels
0	er - ER_076.TIF	er	er
1	microtubules - microtubules_002.tif	microtubules	microtubules
2	mitochondria - mitochondria_054.tif	mitochondria	mitochondria
3	endosome - endosome_073.tif	endosome	endosome
4	mitochondria - mitochondria_008.tif	mitochondria	microtubules
...	...	...	...
857	microtubules - microtubules_030.tif	microtubules	microtubules
858	nucleolus - nucleolus_023.tif	nucleolus	nucleolus
859	microtubules - microtubules_046.tif	microtubules	microtubules
860	nucleolus - nucleolus_027.tif	nucleolus	nucleolus
861	er - ER_066.TIF	er	er

862 rows × 3 columns

These plots show the training accuracy and loss vs the validation accuracy and loss for each epoch in the model.





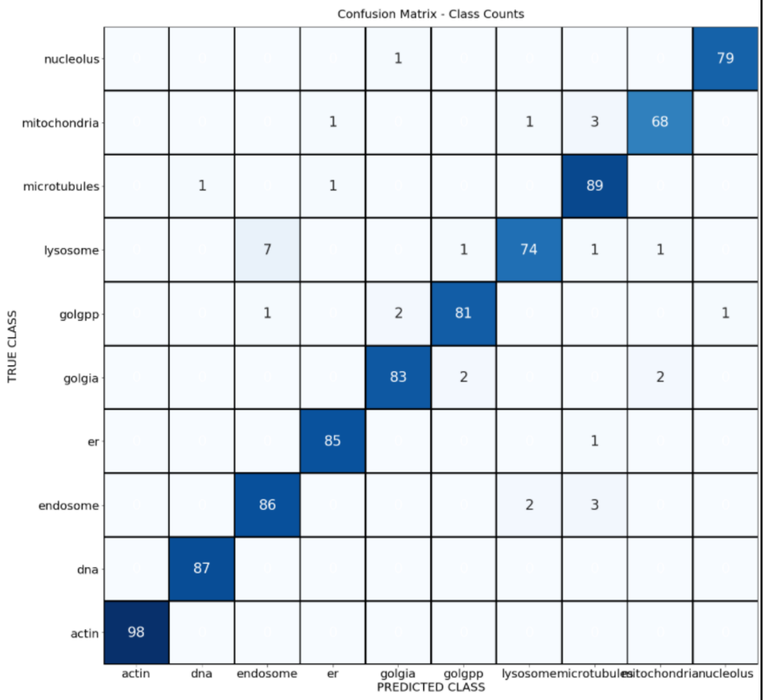
The validation loss is much higher than the training loss for many epochs, and the validation accuracy isn't as high as the training accuracy, this could mean that this model isn't efficient, and we decided to try on the other dataset as this one is hard to improve.

And we also ran the classification report on the data and the confusion matrix:

This shows that the recall for the lysosome images was verry low (88%) compared to other categories as the model couldn't identify and classify it very well, coming after it the mitochondria and golgia images,

The overall accuracy on this dataset was 96%.

```
              precision    recall  f1-score   support

       actin       1.00      1.00      1.00        98
         dna       0.99      1.00      0.99        87
    endosome       0.91      0.95      0.93        91
          er       0.98      0.99      0.98        86
      golgia       0.97      0.95      0.96        87
      golgpp       0.96      0.95      0.96        85
    lysosome       0.96      0.88      0.92        84
microtubules       0.92      0.98      0.95        91
mitochondria       0.96      0.93      0.94        73
   nucleolus       0.99      0.99      0.99        80

    accuracy                           0.96       862
   macro avg       0.96      0.96      0.96       862
weighted avg       0.96      0.96      0.96       862
```

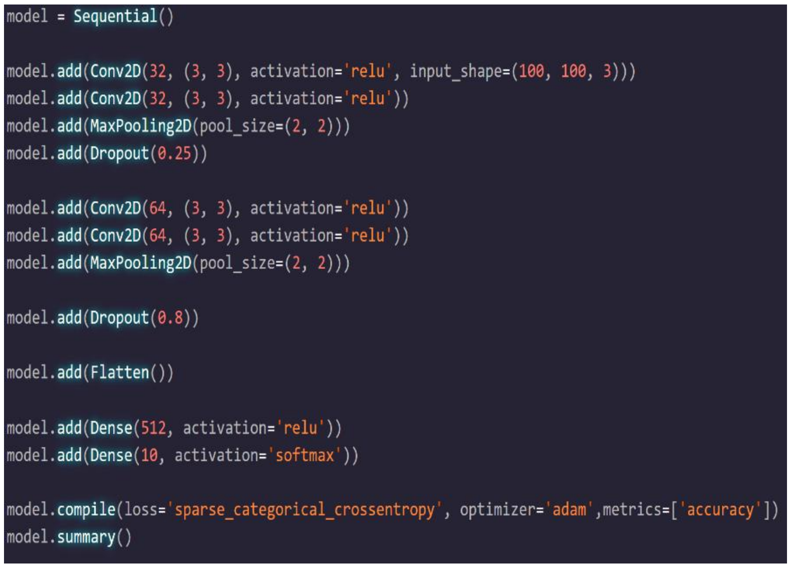


Methods and Algorithms ‘Caltech101’:

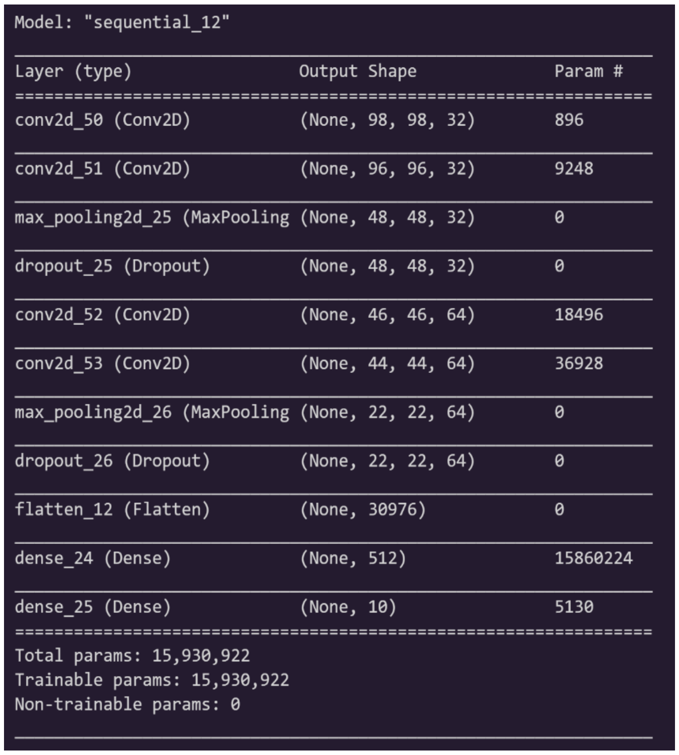
Just like the previous dataset, we preformed the same algorithms on the Caltech101 dataset, using only 10 classes from it [bonsai, Motorbikes, car_side, airplanes, sunflower, laptop, starfish, watch, Leopards, brain], totally 2638 ‘.jpg’ images with RGB format.

We splitted the data into Train, Test and validation sets with ratios 70,15,15 respectively using the sklearn library.

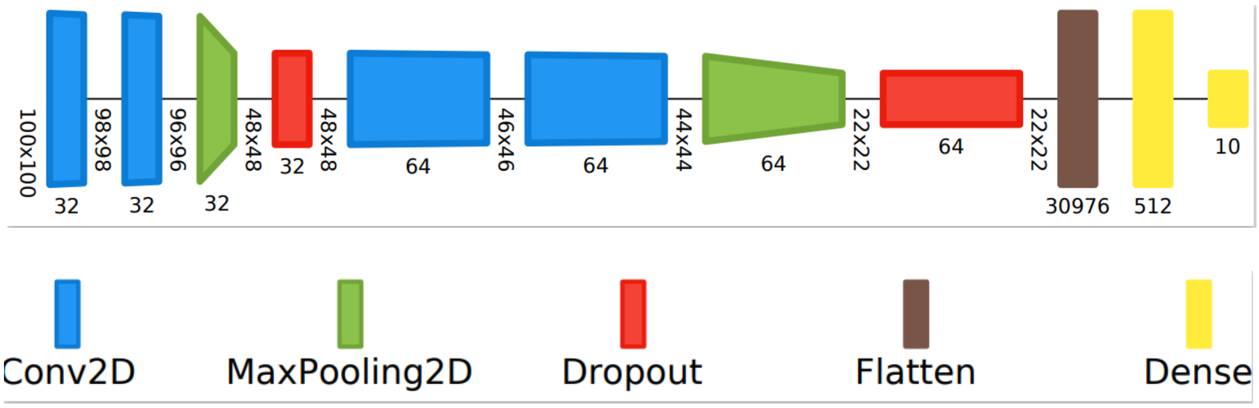
And we basically applied the same rules to build the past model in the ‘Hela-2D’ dataset, after each convolution layer there is a maxPool layer, and number of filters per each layer is larger than the layer before it to detect high-level features in the images.



```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.8))

model.add(Flatten())

model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',metrics=['accuracy'])
model.summary()
```



```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_50 (Conv2D)           (None, 98, 98, 32)        896
_________________________________________________________________
conv2d_51 (Conv2D)           (None, 96, 96, 32)        9248
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 48, 48, 32)        0
_________________________________________________________________
dropout_25 (Dropout)         (None, 48, 48, 32)        0
_________________________________________________________________
conv2d_52 (Conv2D)           (None, 46, 46, 64)        18496
_________________________________________________________________
conv2d_53 (Conv2D)           (None, 44, 44, 64)        36928
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 22, 22, 64)        0
_________________________________________________________________
dropout_26 (Dropout)         (None, 22, 22, 64)        0
_________________________________________________________________
flatten_12 (Flatten)         (None, 30976)             0
_________________________________________________________________
dense_24 (Dense)             (None, 512)               15860224
_________________________________________________________________
dense_25 (Dense)             (None, 10)                5130
=================================================================
Total params: 15,930,922
Trainable params: 15,930,922
Non-trainable params: 0
_________________________________________________________________
```



This model has 98% training accuracy with 93% validation accuracy and 94% testing accuracy, so we saved this model and it's history and we made it predict the whole dataset's labels and compared it to the real labels.

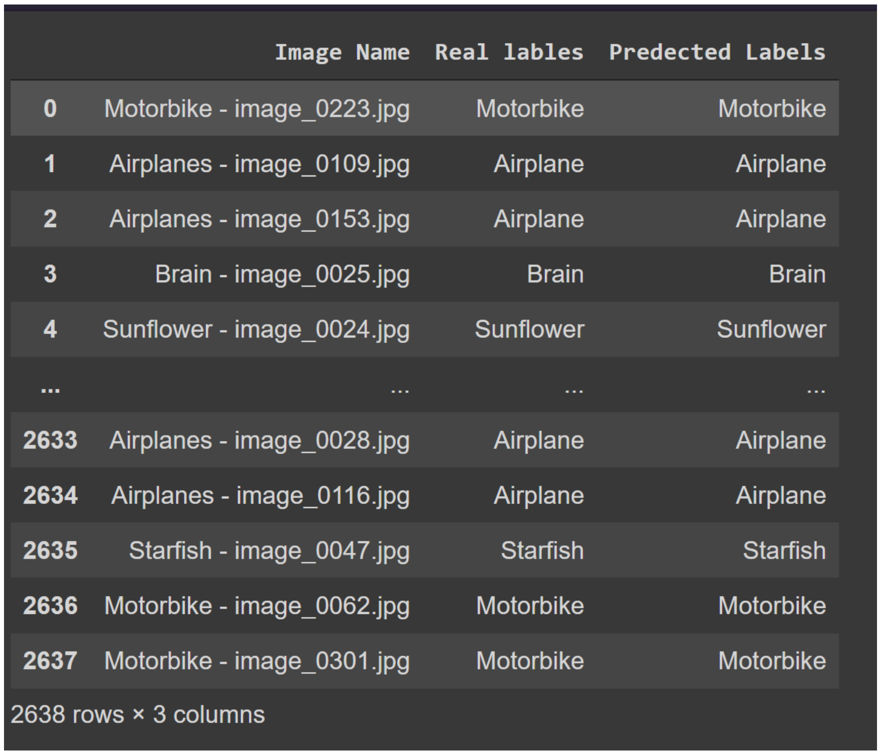


	Image Name	Real lables	Predected Labels
0	Motorbike - image_0223.jpg	Motorbike	Motorbike
1	Airplanes - image_0109.jpg	Airplane	Airplane
2	Airplanes - image_0153.jpg	Airplane	Airplane
3	Brain - image_0025.jpg	Brain	Brain
4	Sunflower - image_0024.jpg	Sunflower	Sunflower
...	...	...	...
2633	Airplanes - image_0028.jpg	Airplane	Airplane
2634	Airplanes - image_0116.jpg	Airplane	Airplane
2635	Starfish - image_0047.jpg	Starfish	Starfish
2636	Motorbike - image_0062.jpg	Motorbike	Motorbike
2637	Motorbike - image_0301.jpg	Motorbike	Motorbike

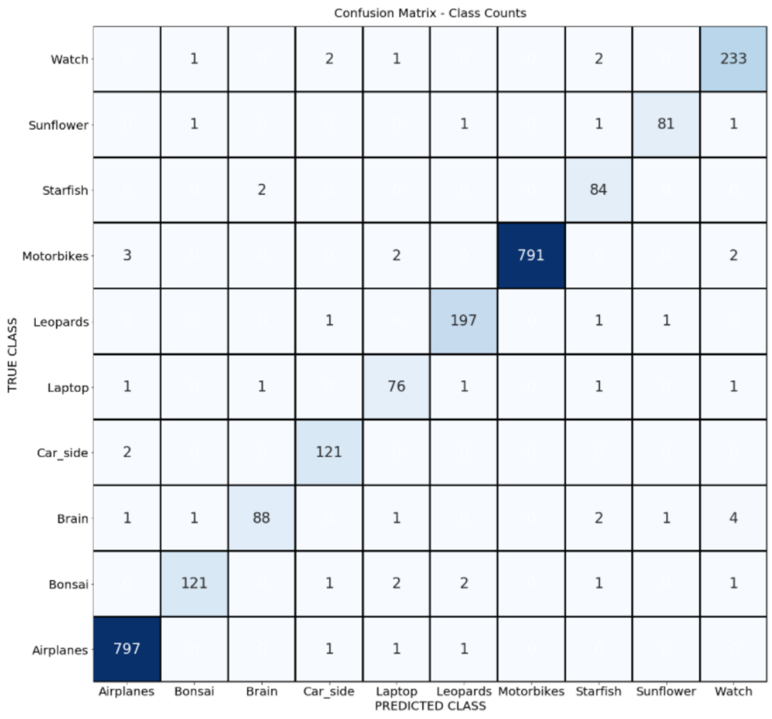
2638 rows × 3 columns

And we also ran the classification report on the data and the confusion matrix::

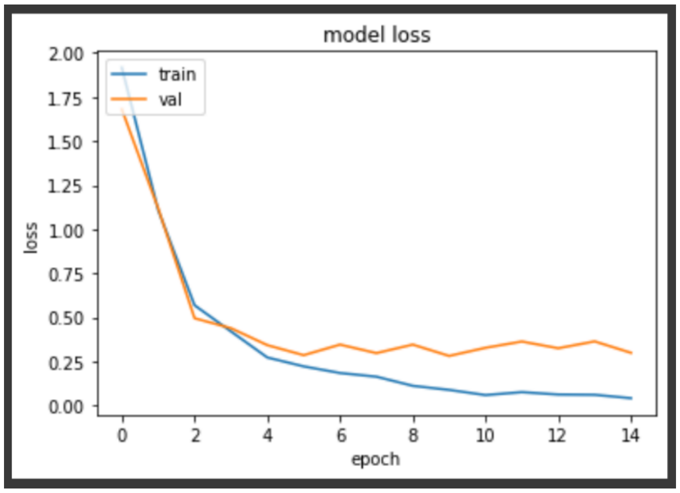
This shows that the recall for the Brain images was verry low (90%) compared to other categories as the model couldn't identify and classify it very well, coming after it the Laptop and Bonsai images,

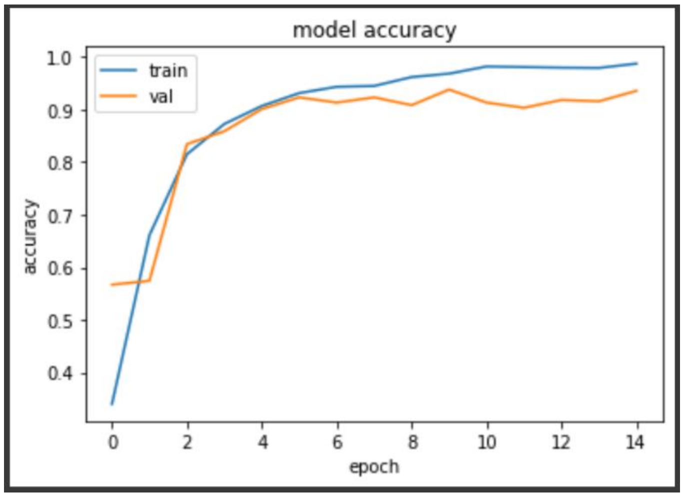
The overall accuracy on this dataset was 98%.

```
              precision    recall  f1-score   support

   Airplanes       0.99      1.00      0.99       800
      Bonsai       0.98      0.95      0.96       128
       Brain       0.97      0.90      0.93        98
    Car_side       0.96      0.98      0.97       123
      Laptop       0.92      0.94      0.93        81
    Leopards       0.98      0.98      0.98       200
  Motorbikes       1.00      0.99      1.00       798
    Starfish       0.91      0.98      0.94        86
   Sunflower       0.98      0.95      0.96        85
       Watch       0.96      0.97      0.97       239

    accuracy                           0.98      2638
   macro avg       0.96      0.96      0.96      2638
weighted avg       0.98      0.98      0.98      2638
```



	Airplanes	Bonsai	Brain	Car_side	Laptop	Leopards	Motorbikes	Starfish	Sunflower	Watch
Watch	0	1	0	2	1	0	0	2	0	233
Sunflower	0	1	0	0	0	1	0	1	81	1
Starfish	0	0	2	0	0	0	0	84	0	0
Motorbikes	3	0	0	0	2	0	791	0	0	2
Leopards	0	0	0	1	0	197	0	1	1	0
Laptop	1	0	1	0	76	1	0	1	0	1
Car_side	2	0	0	121	0	0	0	0	0	0
Brain	1	1	88	0	1	0	0	2	1	4
Bonsai	0	121	0	1	2	2	0	1	0	1
Airplanes	797	0	0	1	1	1	0	0	0	0



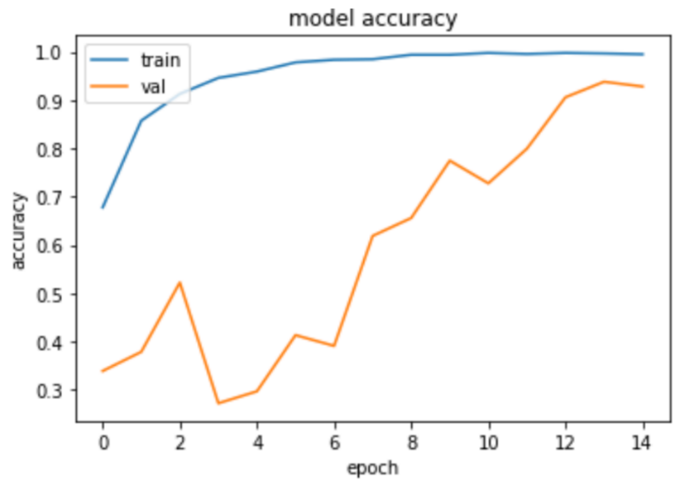


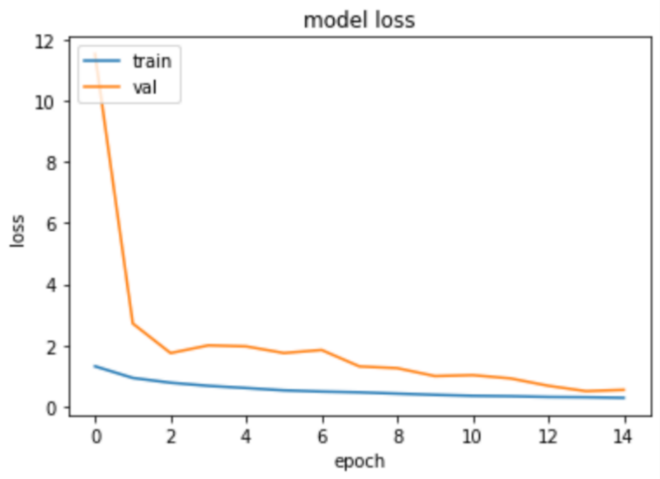
Training accuracy and loss were very similar to validation loss and accuracy for each epoch, which shows that this model is very efficient.

we used AlexNet model to compare it with our CNN implementation:

this model has 99% training accuracy and 92% validation accuracy and 94% testing accuracy which is very similar to out implemented CNN model this model was trained with the exact same training set and number of epochs and bach size.

We also plotted the history of the model ‘loss and accuracy’:





```
# Initialize model
alexnet = Sequential()

# Layer 1
alexnet.add(Conv2D(96, (11, 11), input_shape=img_shape,
  padding='same', kernel_regularizer=l2(l2_reg)))
alexnet.add(BatchNormalization())
alexnet.add(Activation('relu'))
alexnet.add(MaxPooling2D(pool_size=(2, 2)))

# Layer 2
alexnet.add(Conv2D(256, (5, 5), padding='same'))
alexnet.add(BatchNormalization())
alexnet.add(Activation('relu'))
alexnet.add(MaxPooling2D(pool_size=(2, 2)))

# Layer 3
alexnet.add(ZeroPadding2D((1, 1)))
alexnet.add(Conv2D(512, (3, 3), padding='same'))
alexnet.add(BatchNormalization())
alexnet.add(Activation('relu'))
alexnet.add(MaxPooling2D(pool_size=(2, 2)))

# Layer 4
alexnet.add(ZeroPadding2D((1, 1)))
alexnet.add(Conv2D(1024, (3, 3), padding='same'))
alexnet.add(BatchNormalization())
alexnet.add(Activation('relu'))

# Layer 5
alexnet.add(ZeroPadding2D((1, 1)))
alexnet.add(Conv2D(1024, (3, 3), padding='same'))
alexnet.add(BatchNormalization())
alexnet.add(Activation('relu'))
alexnet.add(MaxPooling2D(pool_size=(2, 2)))

# Layer 6
alexnet.add(Flatten())
alexnet.add(Dense(3072))
alexnet.add(BatchNormalization())
alexnet.add(Activation('relu'))
alexnet.add(Dropout(0.5))

# Layer 7
alexnet.add(Dense(4096))
alexnet.add(BatchNormalization())
alexnet.add(Activation('relu'))
alexnet.add(Dropout(0.5))

# Layer 8
alexnet.add(Dense(n_classes))
alexnet.add(BatchNormalization())
alexnet.add(Activation('softmax'))
```

Timing:

We used Google collab for running our code, to make use of the GPU to gain more computation power, the device specifications are as following:

https://colab.research.google.com/drive/151805XTDg--dgHb3-AXJCpnWaqRhop_2#scrollTo=7q3wBeTmb4I9	
CPU	single core hyper threaded intel Xeon Processors @2.3Ghz i.e. (1 core, 2 threads)
GPU	Nvidia Tesla K80, compute 3.7, having 2496 CUDA cores, 12GB GDDR5 VRAM
RAM	12.6 GB Available
DISK	33 GB HDD Available

For the first model on dataset (Hela-2D) with 15 epochs, the average time taken to run a full epoch was 4 seconds so it took almost 60 seconds (1 min) in the training phase, and 33 seconds for the prediction phase.

For the first model on dataset (Caltech-101) with 15 epochs, the average time taken to run a full epoch was 6 seconds so it took almost 90 seconds (01:30 mins) in the training phase, and 64 seconds (01:04 mins) for the prediction phase.

Using the AlexNet model with 15 epochs, the average time taken to run a full epoch was 11 seconds so it took almost 165 seconds (03:15 min) in the training phase.