ANN

preprocessing images:

First I started trying reading data

by function “create_training_data_gray”

and started normalization steps

by covert the images to gray scale and

resize it (64,64)

then I shuffled that data to make it mixed

x is features and Y is the final result

and did more of preprocessing by converting

X and Y to float32

and in X I divided each by 255

Splitting data:

reshape X splitting data to 80:20

and covert both y train and test to binary class matrices

Calling models:

and did two models with different numbers of layers and nodes

and then applied K fold in both models

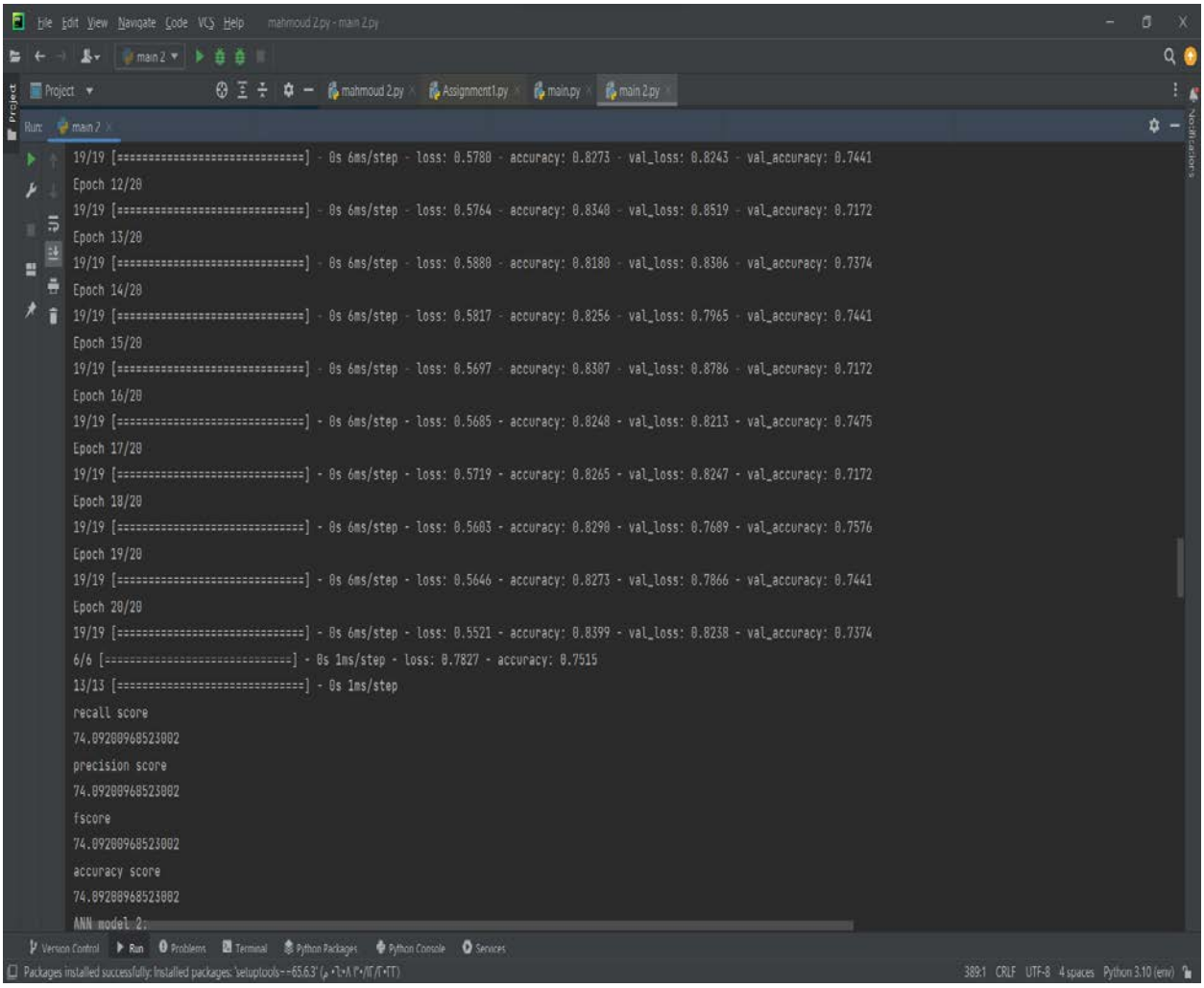
1- first model results which is more accurate and more complex mainly goes from 71:75%

recall score=74.09200968523002

precision score = 74.09200968523002

fscore = 74.09200968523002

accuracy score = 74.09200968523002



2- second model results which is less accurate but still in a good range which is mainly goes from 61:68 but it is still

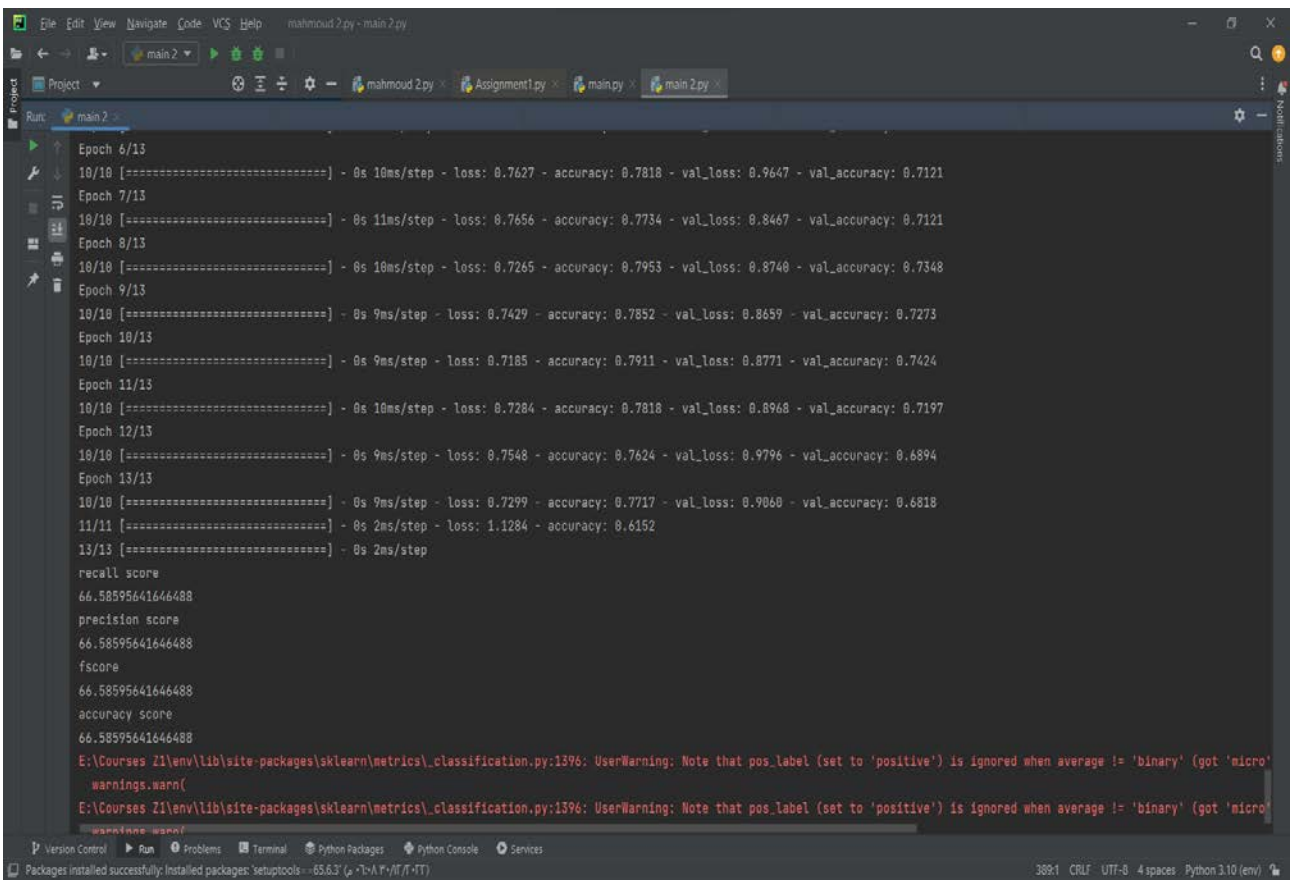
more simpler than the first one

recall score = 66.58595641646488

precision score = 66.58595641646488

fscore = 66.58595641646488

accuracy score = 66.58595641646488



CNN & SVM

CNN model:

preprocessing images:

We preprocessing images by calculate average for each pixel in each image (using stat fun)then convert it to numpy array and subtract average then divide it by 255 add image after preprocessing to list with it's class

Splitting data:

Split data to X and Y then using K-fold concept split data to train and test and validation and call the model

Split was training 80, test 20 and validation 20% of training data

Calling CNN model:

from train and test model in data we see that when we add more conv layers

and pooling layers accuracy increased more than we just add more hidden layers

as before increasing conv and pooling layers accuracy was in range between 83% and 88%

this run accuracies were:

recall score = = 88.61985472154964

precision score = 88.61985472154964

fscore = 88.61985472154964

accuracy score = 88.61985472154964

```
11/11 [==============================] - 0s 33ms/step - loss: 0.7029 - accuracy: 0.7848
At k_fold 2:
Epoch 1/3
17/17 [==============================] - 6s 336ms/step - loss: 0.2917 - accuracy: 0.9128 - val_loss: 0.6081 - val_accuracy: 0.8258
Epoch 2/3
17/17 [==============================] - 6s 383ms/step - loss: 0.1471 - accuracy: 0.9630 - val_loss: 0.6370 - val_accuracy: 0.8182
Epoch 3/3
17/17 [==============================] - 7s 390ms/step - loss: 0.0971 - accuracy: 0.9754 - val_loss: 0.8542 - val_accuracy: 0.7841
11/11 [==============================] - 0s 42ms/step - loss: 0.9018 - accuracy: 0.7788
At k_fold 3:
Epoch 1/3
17/17 [==============================] - 6s 369ms/step - loss: 0.0557 - accuracy: 0.9886 - val_loss: 0.6068 - val_accuracy: 0.8523
Epoch 2/3
17/17 [==============================] - 6s 347ms/step - loss: 0.0231 - accuracy: 0.9962 - val_loss: 0.8185 - val_accuracy: 0.8106
Epoch 3/3
17/17 [==============================] - 6s 335ms/step - loss: 0.0295 - accuracy: 0.9905 - val_loss: 0.7933 - val_accuracy: 0.8523
11/11 [==============================] - 0s 36ms/step - loss: 1.0179 - accuracy: 0.7879
At k_fold 4:
Epoch 1/3
17/17 [==============================] - 6s 347ms/step - loss: 0.1512 - accuracy: 0.9545 - val_loss: 0.8554 - val_accuracy: 0.7803
Epoch 2/3
17/17 [==============================] - 6s 339ms/step - loss: 0.0813 - accuracy: 0.9810 - val_loss: 0.7244 - val_accuracy: 0.8106
Epoch 3/3
17/17 [==============================] - 6s 341ms/step - loss: 0.0235 - accuracy: 0.9972 - val_loss: 0.7669 - val_accuracy: 0.8371
11/11 [==============================] - 0s 36ms/step - loss: 0.8588 - accuracy: 0.8242
At k_fold 5:
Epoch 1/3
17/17 [==============================] - 6s 341ms/step - loss: 0.0082 - accuracy: 1.0000 - val_loss: 0.8406 - val_accuracy: 0.8371
Epoch 2/3
```

```
Epoch 3/3
17/17 [==============================] - 6s 341ms/step - loss: 0.0235 - accuracy: 0.9972 - val_loss: 0.7669 - val_accuracy: 0.8371
11/11 [==============================] - 0s 36ms/step - loss: 0.8588 - accuracy: 0.8242
At k_fold 5:
Epoch 1/3
17/17 [==============================] - 6s 341ms/step - loss: 0.0082 - accuracy: 1.0000 - val_loss: 0.8406 - val_accuracy: 0.8371
Epoch 2/3
17/17 [==============================] - 6s 339ms/step - loss: 0.0025 - accuracy: 1.0000 - val_loss: 0.8201 - val_accuracy: 0.8485
Epoch 3/3
17/17 [==============================] - 6s 361ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 0.8466 - val_accuracy: 0.8523
11/11 [==============================] - 0s 40ms/step - loss: 0.9724 - accuracy: 0.8242
13/13 [==============================] - 1s 42ms/step
E:\Courses Z1\env\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarning: Note that pos_label (set to 'positive') is ignored when average != 'binary' (got 'micro'
  warnings.warn(
E:\Courses Z1\env\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarning: Note that pos_label (set to 'positive') is ignored when average != 'binary' (got 'micro'
  warnings.warn(
E:\Courses Z1\env\lib\site-packages\sklearn\metrics\_classification.py:1396: UserWarning: Note that pos_label (set to 'positive') is ignored when average != 'binary' (got 'micro'
  warnings.warn(
recall score
88.61985472154964
precision score
88.61985472154964
fscore
88.61985472154964
accuracy score
88.61985472154964

Process finished with exit code 0
```

but after increasing them accuracy became in range between 93% and 97%

SVM model:

Using all steps of preprocessing in CNN then calling SVM model

when we apply SVM model we see that accuracy of SVM is very low according to CNN

as accuracy of SVM in range between 44% and 53% and accuracy of CNN is higher than 90%

this run accuracies were:

CNN recall score = 94.91525423728814

CNN precision score = 94.91525423728814

CNN fscore = 94.91525423728814

CNN accuracy score = 94.91525423728814

SVM accuracy score = 45.036319612590795