

AML Mini Project Report

The mini project is about object classification. Each image contains one object. Likewise, there are 100 different types of images in the dataset.

About the training dataset:

The training dataset contains two columns namely – data i.e the pixels and target (0-99).

```
In [58]: print(train_df)
```

		data	target
0	[[46, 132, 11, 1, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...		0
1	[[0, 0, 3, 0, 116, 78, 0, 1, 1, 0, 0, 0, 0, 0, ...		0
2	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, ...		0
3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...		0
4	[[0, 0, 0, 69, 165, 202, 199, 214, 204, 170, 1...]		0
...	
499995	[[0, 0, 0, 0, 1, 4, 7, 8, 6, 4, 5, 1, 0, 0, 0, ...		99
499996	[[0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...		99
499997	[[0, 0, 0, 0, 2, 0, 25, 26, 0, 1, 0, 0, 0, 0, 0, ...		99
499998	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...		99
499999	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, ...		99

[500000 rows x 2 columns]

After sampling the 50% data we get the training dataset – data attribute(X) as follows:

The test dataset contains only the values of the images i.e pixels, without any labels.

Aim: Our aim is to find the labels in the test dataset by improving the accuracy of the model (KNN). This can be achieved by applying various algorithms.

As the deep learning techniques are highly effective when working with images, I have used CNN model. As it takes inputs of the images and extracts the features of it and classifies based on their attributes

The CNN model consists of 4 stages, namely - Input layer, Convolution layer + Activation function, Pooling layer and Fully Connected Layer.

We have a dataset containing 500000 images. Sampling will be required as the dataset is too large. Hence, we will be using 50% of the training set, around 100000 images. After sampling is performed, we will further reshape our dataset into 28, 28, 1 format for uniformity and in turn normalize it further.

```
In [51]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features, label, test_size = 0.2, random_state = 0)
```

[v	v	j	v	270	150	241	257	162	132	100	101	110	111	157	190	244	241	
	173	72	0	0	8	2	0	0	0	0	0								
[0	0	0	3	0	0	68	172	215	230	246	248	243	243	235	215	166	72	
	0	0	5	3	0	0	0	0	0	0	0]								
[0	0	0	0	1	2	0	0	0	4	58	103	108	93	92	73	12	0	0
	2	2	0	0	0	0	0	0	0	0]									

```
In [55]: #reshaping
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], X_train.shape[2], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], X_test.shape[2], 1))
#after reshaping
print(X_train.shape)
print(X_test.shape)

(20000, 28, 28, 1)
(5000, 28, 28, 1)
```

Normalizing the pixels:

```
In [56]: #normalizing the pixels
X_train=X_train/255
X_test=X_test/255
```

Convolutional layer:

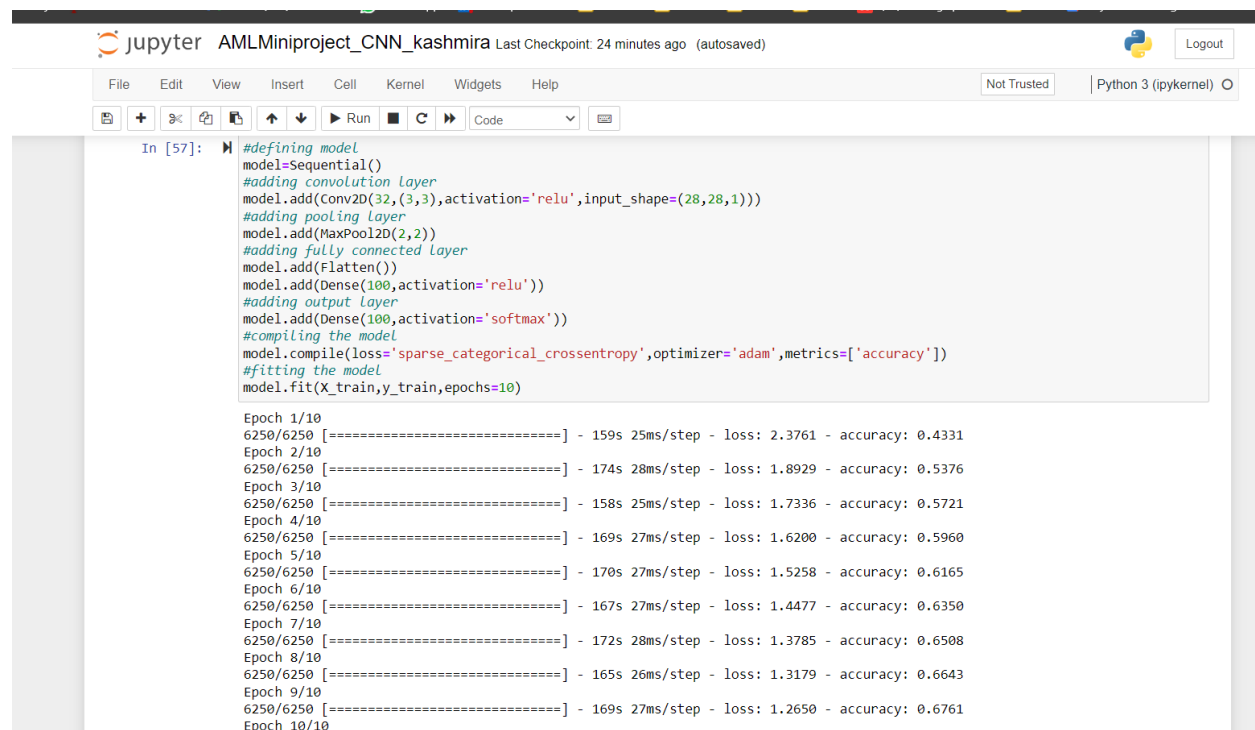
In this layer we will apply filter on our image input. I have applied filter of size 3X3. So a filter of size 3X3 will be applied and multiplied to each pixel in the image and will be shifted by one column in every step. Likewise, a filter will be applied to an image several times. After the filter has completed applying over an entire image, our feature map is ready.

Pooling layer:

This layer is applied after the previous layer is completed. The main goal of this layer is to reduce the dimensions of the feature map. To help in preserving the important attributes of the image and will also reduce the processing time. There are two types of techniques in pooling – Max and Average Pooling. I have used the max pooling technique with 2X2 strides, which will directly consider the max values of each stride, instead of taking average of the entire stride.

Fully Connected Layer:

This last layer merges the results of all the previous layers and uses them to classify the labels of the input image in the output layer.



The screenshot shows a Jupyter Notebook titled "AMLMiniproject_CNN_kashmira". The interface includes a top bar with the Jupyter logo, the notebook title, and a "Last Checkpoint: 24 minutes ago (autosaved)" message. Below the top bar is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. To the right of the menu bar are buttons for "Not Trusted" and "Python 3 (ipykernel)". Below the menu bar is a toolbar with icons for file operations, cell navigation, and execution. The main area of the notebook displays a code cell with the following Python code:

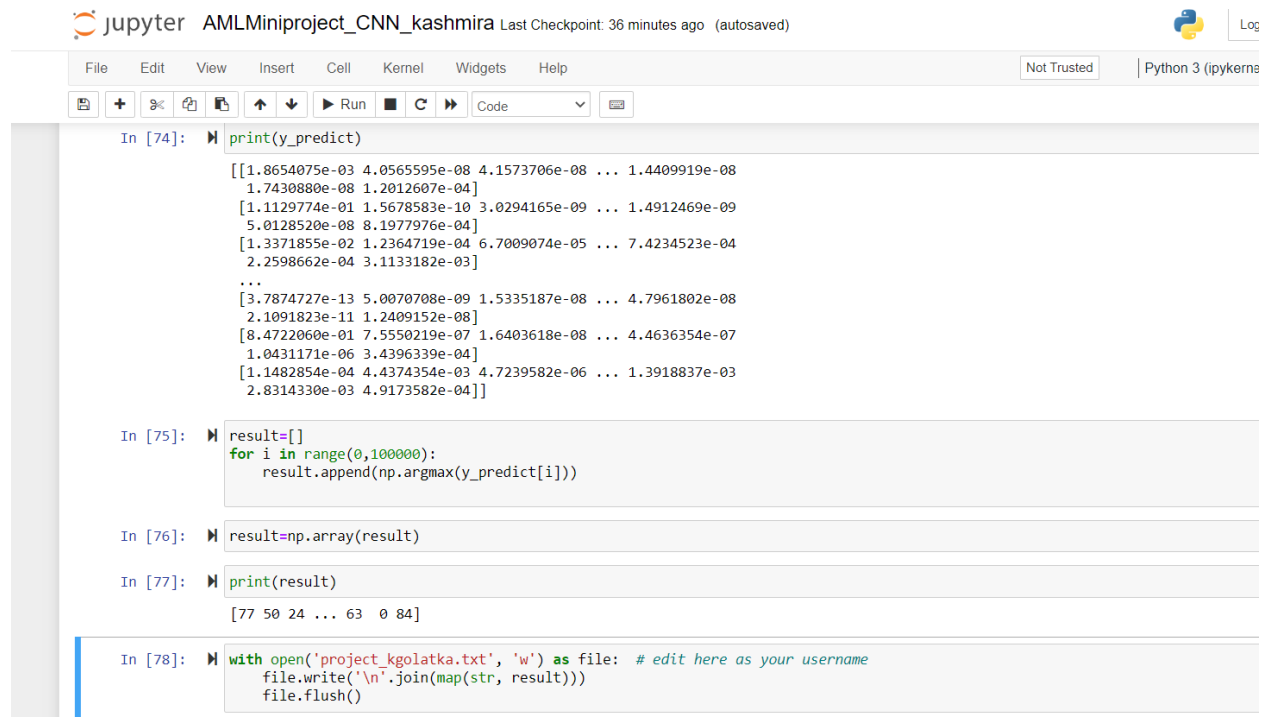
```
In [57]: #defining model
model=Sequential()
#adding convolution layer
model.add(Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)))
#adding pooling layer
model.add(MaxPool2D(2,2))
#adding fully connected layer
model.add(Flatten())
model.add(Dense(100,activation='relu'))
#adding output layer
model.add(Dense(10,activation='softmax'))
#compiling the model
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
#fitting the model
model.fit(X_train,y_train,epochs=10)
```

Below the code cell, the output of the model training is displayed, showing the progress of 10 epochs. Each epoch line includes the current epoch number, the total number of samples (6250/6250), a progress bar, the time taken per step, the loss, and the accuracy.

Epoch	6250/6250	Time	Loss	Accuracy
Epoch 1/10	[=====]	159s 25ms/step	2.3761	0.4331
Epoch 2/10	[=====]	174s 28ms/step	1.8929	0.5376
Epoch 3/10	[=====]	158s 25ms/step	1.7336	0.5721
Epoch 4/10	[=====]	169s 27ms/step	1.6200	0.5960
Epoch 5/10	[=====]	170s 27ms/step	1.5258	0.6165
Epoch 6/10	[=====]	167s 27ms/step	1.4477	0.6350
Epoch 7/10	[=====]	172s 28ms/step	1.3785	0.6508
Epoch 8/10	[=====]	165s 26ms/step	1.3179	0.6643
Epoch 9/10	[=====]	169s 27ms/step	1.2650	0.6761
Epoch 10/10	[=====]			

Applying the model on test the data:

As we can see the predicted output is in exponential form, we need to convert it to a numerical form by using `np.argmax` from the possible 100 outcomes. After the result is acquired in the numerical format, append the results in the text file



The image shows a Jupyter Notebook interface with the title 'AMLMiniproject_CNN_kashmira'. The top bar indicates 'Last Checkpoint: 36 minutes ago (autosaved)' and 'Python 3 (ipykernel)'. The notebook contains several code cells:

```
In [74]: print(y_predict)

[[1.8654075e-03 4.0565595e-08 4.1573706e-08 ... 1.4409919e-08
 1.7430880e-08 1.2012607e-04]
 [1.1129774e-01 1.5678583e-10 3.0294165e-09 ... 1.4912469e-09
 5.0128520e-08 8.1977976e-04]
 [1.3371855e-02 1.2364719e-04 6.7009074e-05 ... 7.4234523e-04
 2.2598662e-04 3.1133182e-03]
 ...
 [3.7874727e-13 5.0070708e-09 1.5335187e-08 ... 4.7961802e-08
 2.1091823e-11 1.2409152e-08]
 [8.4722060e-01 7.550219e-07 1.6403618e-08 ... 4.4636354e-07
 1.0431171e-06 3.4396339e-04]
 [1.1482854e-04 4.4374354e-03 4.7239582e-06 ... 1.3918837e-03
 2.8314330e-03 4.9173582e-04]]

In [75]: result=[]
for i in range(0,100000):
    result.append(np.argmax(y_predict[i]))

In [76]: result=np.array(result)

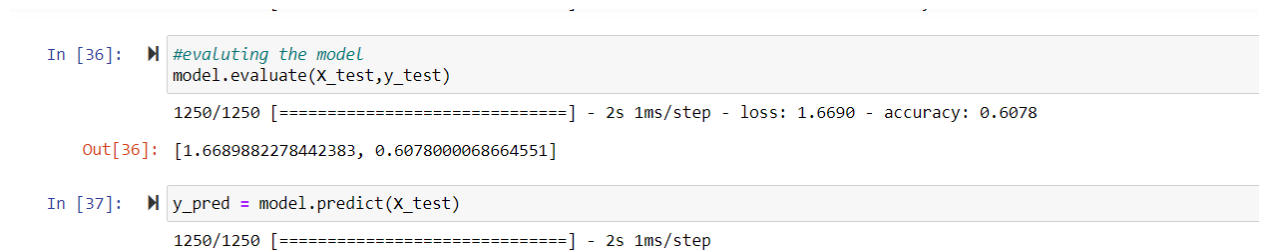
In [77]: print(result)

[77 50 24 ... 63  0 84]

In [78]: with open('project_kgolatka.txt', 'w') as file: # edit here as your username
    file.write('\n'.join(map(str, result)))
    file.flush()
```

Results:

After applying the CNN model on 50% sampled training data, the accuracy is 60%.



The image shows a Jupyter Notebook interface with the following code cells:

```
In [36]: #evaluting the model
model.evaluate(X_test,y_test)

1250/1250 [=====] - 2s 1ms/step - loss: 1.6690 - accuracy: 0.6078

Out[36]: [1.6689882278442383, 0.6078000068664551]

In [37]: y_pred = model.predict(X_test)

1250/1250 [=====] - 2s 1ms/step
```

Ways to improve accuracy:

The accuracy can be improved by training the data on a larger training set i.e. more than 50% of the training data.

It can also be improved by applying other different machine learning models like SVM and Random Forest.