Problem Formulation:

We are given an image dataset, on which gray scaling and downsizing are already done. We are supposed to recognize the object in the images. In order to do that, we need to create a classifier that decides
the correct category of a given image.

So, I first tried to implement a simple Logistic Regression model by converting the image data into a NumPy array. It gave very low accuracy, due to which I started searching for other techniques. While exploring, I came across image classification techniques like SVM, Decision Tree, KNN, & CNN. After watching videos of each technique, I found that CNN Model is best suited for the provided dataset since it is mostly used for image recognition and object classification. So, I started exploring CNN over the internet and ways to solve the problem. Due to the time constraints, I could learn from limited resources and tried my best to achieve better accuracy.

Preprocessing:

- **Reading the dataset:** The dataset is in the file of .PKL. So, in order to read that file, we need to import pickle.
- Then, I specified the fraction of items to return to the sample. I set it to 1, to return all the data. Also, checked for null values.
- I also tried to print the images, but it’s giving some error at the last step. It did print images before, but in order to smooth the functioning of the code, I commented on that part.
- Now, to split the dataset into training and testing, I used train_test_split. Also, reshaped the data.

Assumptions:

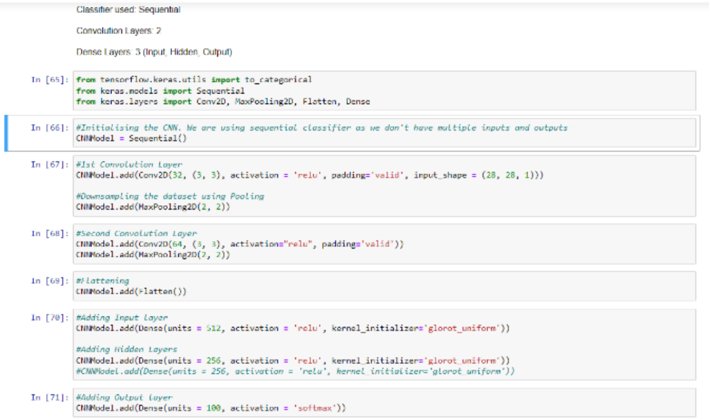
- I didn’t use the test data (unlabelled dataset) as it doesn’t have a target column.
- Pandas version is >= 1.4.1
- All the CNN layers work sequentially, without interfering with each other.

Model Design:

As stated earlier, I’m using CNN to find the solution to this problem.

I’m implementing a sequential classifier in order to achieve the goal. In sequential classifiers, the layers are arranged in a line and have exactly one input and output file.

Attaching the snippet of code for better understanding.



Classifier used: Sequential

Convolution Layers: 2

Dense Layers: 3 (Input, Hidden, Output)

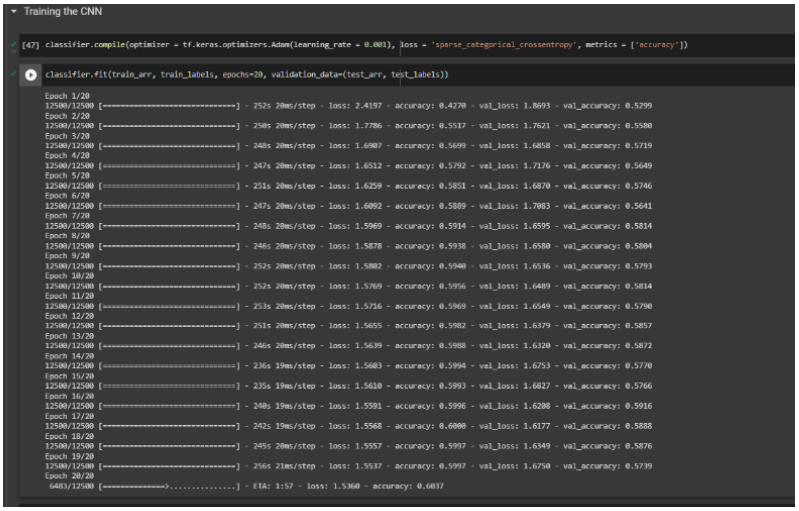
```
In [65]: from tensorflow.keras.utils import to_categorical
         from keras.models import Sequential
         from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

In [66]: #Initialising the CNN. We are using sequential classifier as we don't have multiple inputs and outputs
         CNNModel = Sequential()

In [67]: #1st Convolution Layer
         CNNModel.add(Conv2D(32, (3, 3), activation = 'relu', padding='valid', input_shape = (28, 28, 1)))

         #Downsampling the dataset using Pooling
         CNNModel.add(MaxPooling2D(2, 2))

In [68]: #Second Convolution Layer
         CNNModel.add(Conv2D(64, (3, 3), activation="relu", padding='valid'))
         CNNModel.add(MaxPooling2D(2, 2))

In [69]: #Flattening
         CNNModel.add(Flatten())

In [70]: #Adding Input Layer
         CNNModel.add(Dense(units = 512, activation = 'relu', kernel_initializer='glorot_uniform'))

         #Adding Hidden Layers
         CNNModel.add(Dense(units = 256, activation = 'relu', kernel_initializer='glorot_uniform'))
         #CNNModel.add(Dense(units = 256, activation = 'relu', kernel_initializer='glorot_uniform'))

In [71]: #Adding Output Layer
         CNNModel.add(Dense(units = 100, activation = 'softmax'))
```

The model has 3 Dense Layers: 1 Input layer (with 512 neurons), 1 hidden layer (with 256 neurons), and 1 output layer (with 100 neurons).

I tried various attributes for activation, kernel_initializer before finalizing with the one in the program. I'll explain the results for each attribute in the next point.

Results:

- First I implemented CNN without any hidden layers. The model contained only input and output layers, due to which the accuracy barely touched 60%. This model contained 16 steps and 128 units (Screenshot below)



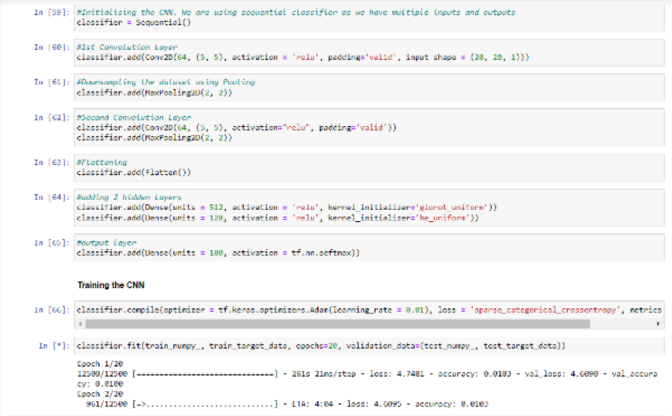
Training the CNN

```
[47] classifier.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001), loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])
```

```
classifier.fit(train_arr, train_labels, epochs=20, validation_data=(test_arr, test_labels))
```

```
Epoch 1/20
12500/12500 [==============================] - 252s 20ms/step - loss: 2.4197 - accuracy: 0.4270 - val_loss: 1.8693 - val_accuracy: 0.5299
Epoch 2/20
12500/12500 [==============================] - 250s 20ms/step - loss: 1.7786 - accuracy: 0.5517 - val_loss: 1.7621 - val_accuracy: 0.5580
Epoch 3/20
12500/12500 [==============================] - 248s 20ms/step - loss: 1.6907 - accuracy: 0.5699 - val_loss: 1.6858 - val_accuracy: 0.5719
Epoch 4/20
12500/12500 [==============================] - 247s 20ms/step - loss: 1.6512 - accuracy: 0.5792 - val_loss: 1.7176 - val_accuracy: 0.5649
Epoch 5/20
12500/12500 [==============================] - 251s 20ms/step - loss: 1.6259 - accuracy: 0.5851 - val_loss: 1.6870 - val_accuracy: 0.5746
Epoch 6/20
12500/12500 [==============================] - 247s 20ms/step - loss: 1.6092 - accuracy: 0.5889 - val_loss: 1.7083 - val_accuracy: 0.5641
Epoch 7/20
12500/12500 [==============================] - 248s 20ms/step - loss: 1.5969 - accuracy: 0.5914 - val_loss: 1.6595 - val_accuracy: 0.5814
Epoch 8/20
12500/12500 [==============================] - 246s 20ms/step - loss: 1.5878 - accuracy: 0.5938 - val_loss: 1.6580 - val_accuracy: 0.5804
Epoch 9/20
12500/12500 [==============================] - 252s 20ms/step - loss: 1.5802 - accuracy: 0.5940 - val_loss: 1.6536 - val_accuracy: 0.5793
Epoch 10/20
12500/12500 [==============================] - 252s 20ms/step - loss: 1.5769 - accuracy: 0.5956 - val_loss: 1.6489 - val_accuracy: 0.5814
Epoch 11/20
12500/12500 [==============================] - 253s 20ms/step - loss: 1.5716 - accuracy: 0.5969 - val_loss: 1.6549 - val_accuracy: 0.5790
Epoch 12/20
12500/12500 [==============================] - 251s 20ms/step - loss: 1.5655 - accuracy: 0.5982 - val_loss: 1.6379 - val_accuracy: 0.5857
Epoch 13/20
12500/12500 [==============================] - 246s 20ms/step - loss: 1.5639 - accuracy: 0.5988 - val_loss: 1.6320 - val_accuracy: 0.5872
Epoch 14/20
12500/12500 [==============================] - 236s 19ms/step - loss: 1.5603 - accuracy: 0.5994 - val_loss: 1.6753 - val_accuracy: 0.5770
Epoch 15/20
12500/12500 [==============================] - 235s 19ms/step - loss: 1.5610 - accuracy: 0.5993 - val_loss: 1.6827 - val_accuracy: 0.5766
Epoch 16/20
12500/12500 [==============================] - 240s 19ms/step - loss: 1.5591 - accuracy: 0.5996 - val_loss: 1.6208 - val_accuracy: 0.5916
Epoch 17/20
12500/12500 [==============================] - 242s 19ms/step - loss: 1.5568 - accuracy: 0.6000 - val_loss: 1.6177 - val_accuracy: 0.5888
Epoch 18/20
12500/12500 [==============================] - 245s 20ms/step - loss: 1.5557 - accuracy: 0.5997 - val_loss: 1.6349 - val_accuracy: 0.5876
Epoch 19/20
12500/12500 [==============================] - 256s 21ms/step - loss: 1.5537 - accuracy: 0.5997 - val_loss: 1.6750 - val_accuracy: 0.5739
Epoch 20/20
 6483/12500 [==============>...............] - ETA: 1:57 - loss: 1.5360 - accuracy: 0.6037
```

- Then, I added a hidden layer, set activation to "relu" and kernel_initializer to both "glorot_uniform" and "he_uniform" in order to achieve good accuracy. But the result was different. The accuracy worsened. It was found to be ~5%.

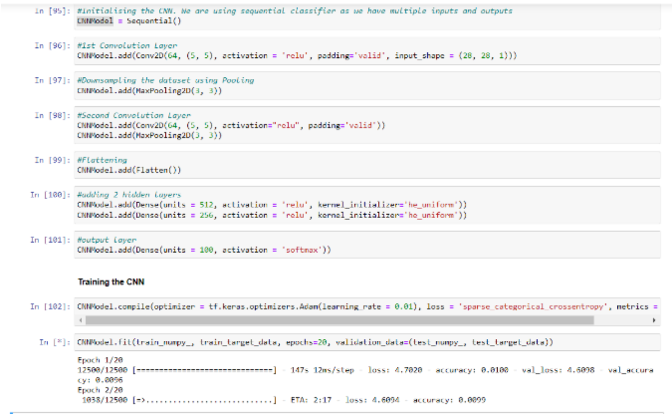


```
In [59]: #Initialising the CNN. We are using sequential classifier as we have multiple inputs and outputs
         classifier = Sequential()

In [60]: #1st Convolution Layer
         classifier.add(Conv2D(64, (5, 5), activation = 'relu', padding='valid', input_shape = (28, 28, 1)))

In [61]: #Downsampling the dataset using Pooling
         classifier.add(MaxPooling2D(2, 2))

In [62]: #Second Convolution Layer
         classifier.add(Conv2D(64, (5, 5), activation="relu", padding='valid'))
         classifier.add(MaxPooling2D(2, 2))

In [63]: #Flattening
         classifier.add(Flatten())

In [64]: #adding 2 hidden Layers
         classifier.add(Dense(units = 512, activation = 'relu', kernel_initializer='glorot_uniform'))
         classifier.add(Dense(units = 128, activation = 'relu', kernel_initializer='he_uniform'))

In [65]: #output layer
         classifier.add(Dense(units = 100, activation = tf.nn.softmax))
```

Training the CNN

```
In [66]: classifier.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.01), loss = 'sparse_categorical_crossentropy', metrics

In [*]: classifier.fit(train_numpy_, train_target_data, epochs=20, validation_data=(test_numpy_, test_target_data))

         Epoch 1/20
         12500/12500 [==============================] - 261s 21ms/step - loss: 4.7481 - accuracy: 0.0103 - val_loss: 4.6090 - val_accura
         cy: 0.0100
         Epoch 2/20
           961/12500 [=>............................] - ETA: 4:04 - loss: 4.6095 - accuracy: 0.0103
```

- The various resources over the internet suggested the implementation of the "he_uniform" kernel. So I used that. But turns out that the combination of "he_uniform" kernel and "relu" activation with specified height and width of image was not good either.

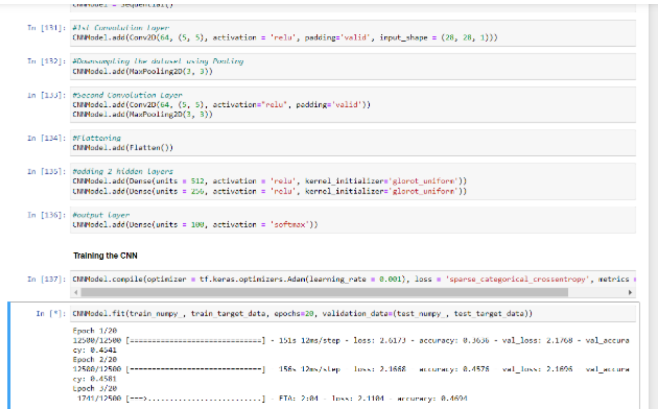


```
In [95]: #initialising the CNN. We are using sequential classifier as we have multiple inputs and outputs
         CNNModel = Sequential()

In [96]: #1st Convolution Layer
         CNNModel.add(Conv2D(64, (5, 5), activation = 'relu', padding='valid', input_shape = (28, 28, 1)))

In [97]: #Downsampling the dataset using Pooling
         CNNModel.add(MaxPooling2D(3, 3))

In [98]: #Second Convolution Layer
         CNNModel.add(Conv2D(64, (5, 5), activation="relu", padding='valid'))
         CNNModel.add(MaxPooling2D(3, 3))

In [99]: #Flattening
         CNNModel.add(Flatten())

In [100]: #adding 2 hidden layers
          CNNModel.add(Dense(units = 512, activation = 'relu', kernel_initializer='he_uniform'))
          CNNModel.add(Dense(units = 256, activation = 'relu', kernel_initializer='he_uniform'))

In [101]: #output layer
          CNNModel.add(Dense(units = 100, activation = 'softmax'))
```

Training the CNN

```
In [102]: CNNModel.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.01), loss = 'sparse_categorical_crossentropy', metrics =

In [*]: CNNModel.fit(train_numpy_, train_target_data, epochs=20, validation_data=(test_numpy_, test_target_data))

        Epoch 1/20
        12500/12500 [==============================] - 147s 12ms/step - loss: 4.7020 - accuracy: 0.0100 - val_loss: 4.6098 - val_accura
        cy: 0.0096
        Epoch 2/20
         1038/12500 [=>............................] - ETA: 2:17 - loss: 4.6094 - accuracy: 0.0099
```

- So, I tried with the “glorot_uniform” kernel, which gave an accuracy of around 70%, best till now.

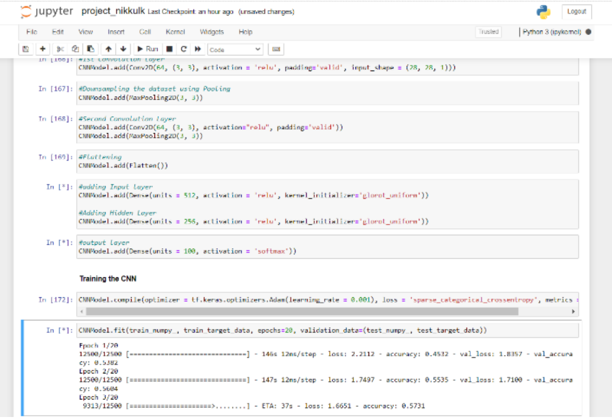


```
         CNNModel = Sequential()

In [131]: #1st Convolution Layer
          CNNModel.add(Conv2D(64, (5, 5), activation = 'relu', padding='valid', input_shape = (28, 28, 1)))

In [132]: #Downsampling the dataset using Pooling
          CNNModel.add(MaxPooling2D(3, 3))

In [133]: #Second Convolution Layer
          CNNModel.add(Conv2D(64, (5, 5), activation="relu", padding='valid'))
          CNNModel.add(MaxPooling2D(3, 3))

In [134]: #Flattening
          CNNModel.add(Flatten())

In [135]: #adding 2 hidden layers
          CNNModel.add(Dense(units = 512, activation = 'relu', kernel_initializer='glorot_uniform'))
          CNNModel.add(Dense(units = 256, activation = 'relu', kernel_initializer='glorot_uniform'))

In [136]: #output layer
          CNNModel.add(Dense(units = 100, activation = 'softmax'))
```

Training the CNN

```
In [137]: CNNModel.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001), loss = 'sparse_categorical_crossentropy', metrics =

In [*]: CNNModel.fit(train_numpy_, train_target_data, epochs=20, validation_data=(test_numpy_, test_target_data))

        Epoch 1/20
        12500/12500 [==============================] - 151s 12ms/step - loss: 2.6173 - accuracy: 0.3636 - val_loss: 2.1768 - val_accura
        cy: 0.4541
        Epoch 2/20
        12500/12500 [==============================] - 156s 12ms/step - loss: 2.1668 - accuracy: 0.4576 - val_loss: 2.1696 - val_accura
        cy: 0.4581
        Epoch 3/20
         1741/12500 [===>..........................] - ETA: 2:04 - loss: 2.1104 - accuracy: 0.4694
```

- Again, to improve the accuracy, I decreased the image size from (5,5) to (3,3), which improved accuracy by around 7%, resulting to 77%



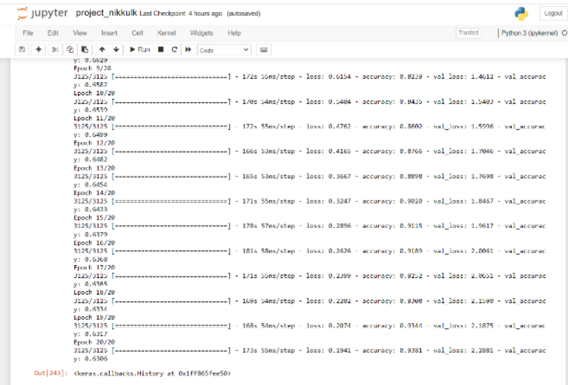
jupyter project_nikkulk Last Checkpoint: an hour ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
In [166]: #1st Convolution Layer
          CNNModel.add(Conv2D(64, (3, 3), activation = 'relu', padding='valid', input_shape = (28, 28, 1)))

In [167]: #Downsampling the dataset using Pooling
          CNNModel.add(MaxPooling2D(3, 3))

In [168]: #Second Convolution Layer
          CNNModel.add(Conv2D(64, (3, 3), activation="relu", padding='valid'))
          CNNModel.add(MaxPooling2D(3, 3))

In [169]: #Flattening
          CNNModel.add(Flatten())

In [*]: #adding Input Layer
        CNNModel.add(Dense(units = 512, activation = 'relu', kernel_initializer='glorot_uniform'))

        #Adding Hidden Layer
        CNNModel.add(Dense(units = 256, activation = 'relu', kernel_initializer='glorot_uniform'))

In [*]: #output layer
        CNNModel.add(Dense(units = 100, activation = 'softmax'))
```

Training the CNN

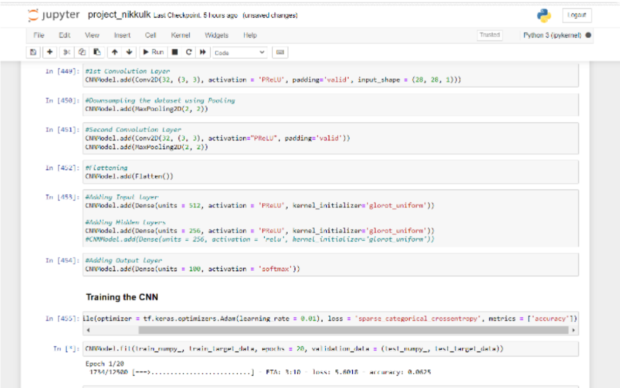
```
In [172]: CNNModel.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001), loss = 'sparse_categorical_crossentropy', metrics =

In [*]: CNNModel.fit(train_numpy_, train_target_data, epochs=20, validation_data=(test_numpy_, test_target_data))

        Epoch 1/20
        12500/12500 [==============================] - 146s 12ms/step - loss: 2.2112 - accuracy: 0.4532 - val_loss: 1.8357 - val_accura
        cy: 0.5382
        Epoch 2/20
        12500/12500 [==============================] - 147s 12ms/step - loss: 1.7497 - accuracy: 0.5535 - val_loss: 1.7100 - val_accura
        cy: 0.5604
        Epoch 3/20
         9313/12500 [=====================>........] - ETA: 37s - loss: 1.6651 - accuracy: 0.5731
```

- I also received an accuracy of 94% at some point, when I set the value of neurons to 1024 for input layer. But I think the model was overfitting, so I didn’t submit that.


jupyter project_nikkulk Last Checkpoint: 4 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
y: 0.6629
Epoch 9/20
3125/3125 [==============================] - 172s 55ms/step - loss: 0.6154 - accuracy: 0.8239 - val_loss: 1.4612 - val_accurac
y: 0.6582
Epoch 10/20
3125/3125 [==============================] - 170s 54ms/step - loss: 0.5404 - accuracy: 0.8435 - val_loss: 1.5403 - val_accurac
y: 0.6539
Epoch 11/20
3125/3125 [==============================] - 172s 55ms/step - loss: 0.4762 - accuracy: 0.8602 - val_loss: 1.5996 - val_accurac
y: 0.6499
Epoch 12/20
3125/3125 [==============================] - 166s 53ms/step - loss: 0.4165 - accuracy: 0.8766 - val_loss: 1.7046 - val_accurac
y: 0.6482
Epoch 13/20
3125/3125 [==============================] - 165s 53ms/step - loss: 0.3667 - accuracy: 0.8898 - val_loss: 1.7698 - val_accurac
y: 0.6454
Epoch 14/20
3125/3125 [==============================] - 171s 55ms/step - loss: 0.3247 - accuracy: 0.9020 - val_loss: 1.8467 - val_accurac
y: 0.6433
Epoch 15/20
3125/3125 [==============================] - 178s 57ms/step - loss: 0.2896 - accuracy: 0.9115 - val_loss: 1.9617 - val_accurac
y: 0.6379
Epoch 16/20
3125/3125 [==============================] - 181s 58ms/step - loss: 0.2626 - accuracy: 0.9189 - val_loss: 2.0061 - val_accurac
y: 0.6368
Epoch 17/20
3125/3125 [==============================] - 171s 55ms/step - loss: 0.2399 - accuracy: 0.9252 - val_loss: 2.0651 - val_accurac
y: 0.6365
Epoch 18/20
3125/3125 [==============================] - 169s 54ms/step - loss: 0.2202 - accuracy: 0.9300 - val_loss: 2.1590 - val_accurac
y: 0.6334
Epoch 19/20
3125/3125 [==============================] - 168s 54ms/step - loss: 0.2074 - accuracy: 0.9344 - val_loss: 2.1875 - val_accurac
y: 0.6317
Epoch 20/20
3125/3125 [==============================] - 173s 55ms/step - loss: 0.1941 - accuracy: 0.9381 - val_loss: 2.2881 - val_accurac
y: 0.6306
Out[243]: <keras.callbacks.History at 0x1ff865fee50>
```

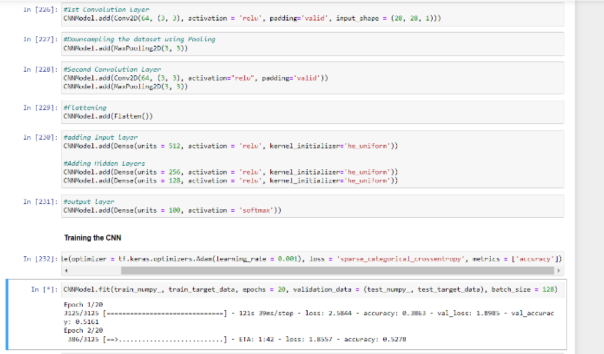
- After this accuracy, I was trying various permutations and combinations of kernel_initializers and activation (relu, selu, sigmoid, tanh, PReLU) in order to compare the accuracies. I found all these parameters over the internet when I was reading about the sequential classifier. The screenshots of the accuracies are attached below.


jupyter project_nikkulk Last Checkpoint: 5 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
In [449]: #1st Convolution Layer
          CNNModel.add(Conv2D(32, (3, 3), activation = 'PReLU', padding='valid', input_shape = (28, 28, 1)))

In [450]: #Downsampling the dataset using Pooling
          CNNModel.add(MaxPooling2D(2, 2))

In [451]: #Second Convolution Layer
          CNNModel.add(Conv2D(32, (3, 3), activation="PReLU", padding='valid'))
          CNNModel.add(MaxPooling2D(2, 2))

In [452]: #Flattening
          CNNModel.add(Flatten())

In [453]: #Adding Input Layer
          CNNModel.add(Dense(units = 512, activation = 'PReLU', kernel_initializer='glorot_uniform'))

          #Adding Hidden Layers
          CNNModel.add(Dense(units = 256, activation = 'PReLU', kernel_initializer='glorot_uniform'))
          #CNNModel.add(Dense(units = 256, activation = 'relu', kernel_initializer='glorot_uniform'))

In [454]: #Adding Output Layer
          CNNModel.add(Dense(units = 100, activation = 'softmax'))
```

Training the CNN

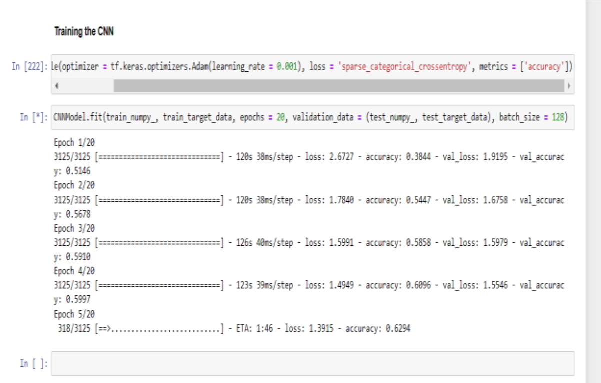
```
In [455]: ile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.01), loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])

In [*]: CNNModel.fit(train_numpy_, train_target_data, epochs = 20, validation_data = (test_numpy_, test_target_data))

          Epoch 1/20
           1734/12500 [--->..........................] - ETA: 3:10 - loss: 5.6018 - accuracy: 0.0625
```



```
In [226]: #1st Convolution Layer
          CNNModel.add(Conv2D(64, (3, 3), activation = 'relu', padding='valid', input_shape = (28, 28, 1)))

In [227]: #Downsampling the dataset using Pooling
          CNNModel.add(MaxPooling2D(3, 3))

In [228]: #Second Convolution Layer
          CNNModel.add(Conv2D(64, (3, 3), activation="relu", padding='valid'))
          CNNModel.add(MaxPooling2D(3, 3))

In [229]: #Flattening
          CNNModel.add(Flatten())

In [230]: #adding Input Layer
          CNNModel.add(Dense(units = 512, activation = 'relu', kernel_initializer='he_uniform'))

          #Adding Hidden Layers
          CNNModel.add(Dense(units = 256, activation = 'relu', kernel_initializer='he_uniform'))
          CNNModel.add(Dense(units = 128, activation = 'relu', kernel_initializer='he_uniform'))

In [231]: #output Layer
          CNNModel.add(Dense(units = 100, activation = 'softmax'))
```

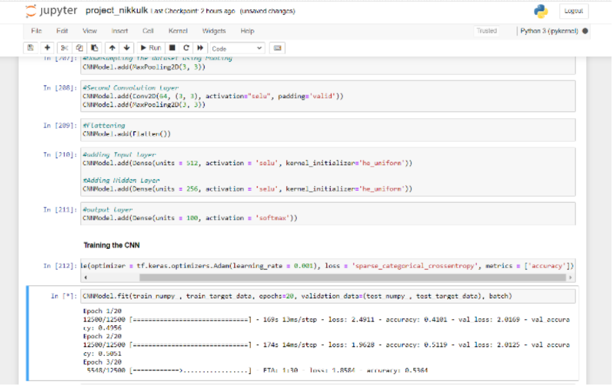
Training the CNN

```
In [232]: le(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001), loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])

In [*]: CNNModel.fit(train_numpy_, train_target_data, epochs = 20, validation_data = (test_numpy_, test_target_data), batch_size = 128)

          Epoch 1/20
          3125/3125 [==============================] - 121s 39ms/step - loss: 2.5844 - accuracy: 0.3863 - val_loss: 1.8985 - val_accurac
          y: 0.5161
          Epoch 2/20
           386/3125 [==>...........................] - ETA: 1:42 - loss: 1.8557 - accuracy: 0.5278
```



Training the CNN

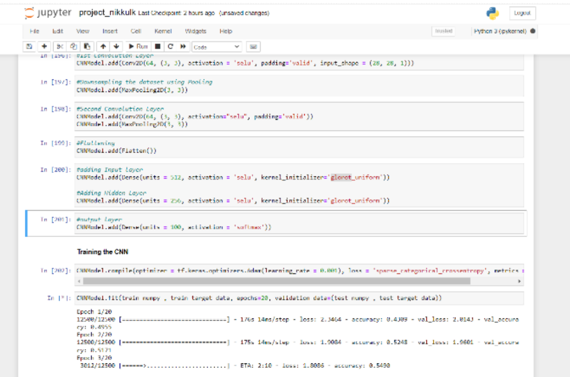
```
In [222]: le(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001), loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])

In [*]: CNNModel.fit(train_numpy_, train_target_data, epochs = 20, validation_data = (test_numpy_, test_target_data), batch_size = 128)

          Epoch 1/20
          3125/3125 [==============================] - 120s 38ms/step - loss: 2.6727 - accuracy: 0.3844 - val_loss: 1.9195 - val_accurac
          y: 0.5146
          Epoch 2/20
          3125/3125 [==============================] - 120s 38ms/step - loss: 1.7840 - accuracy: 0.5447 - val_loss: 1.6758 - val_accurac
          y: 0.5678
          Epoch 3/20
          3125/3125 [==============================] - 126s 40ms/step - loss: 1.5991 - accuracy: 0.5858 - val_loss: 1.5979 - val_accurac
          y: 0.5910
          Epoch 4/20
          3125/3125 [==============================] - 123s 39ms/step - loss: 1.4949 - accuracy: 0.6096 - val_loss: 1.5546 - val_accurac
          y: 0.5997
          Epoch 5/20
           318/3125 [==>...........................] - ETA: 1:46 - loss: 1.3915 - accuracy: 0.6294

In [ ]:
```


jupyter project_nikkulk Last Checkpoint: 2 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
In [207]: #Downsampling the dataset using Pooling
          CNNModel.add(MaxPooling2D(3, 3))

In [208]: #Second Convolution Layer
          CNNModel.add(Conv2D(64, (3, 3), activation="selu", padding='valid'))
          CNNModel.add(MaxPooling2D(3, 3))

In [209]: #Flattening
          CNNModel.add(Flatten())

In [210]: #adding Input Layer
          CNNModel.add(Dense(units = 512, activation = 'selu', kernel_initializer='he_uniform'))

          #Adding Hidden Layer
          CNNModel.add(Dense(units = 256, activation = 'selu', kernel_initializer='he_uniform'))

In [211]: #output Layer
          CNNModel.add(Dense(units = 100, activation = 'softmax'))
```

Training the CNN

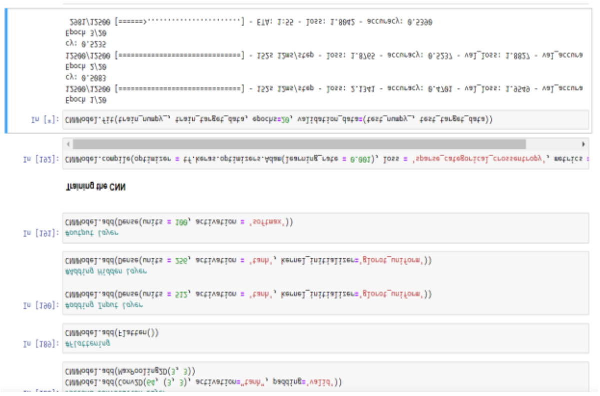
```
In [212]: le(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001), loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])

In [*]: CNNModel.fit(train_numpy_, train_target_data, epochs=20, validation_data=(test_numpy_, test_target_data), batch)

          Epoch 1/20
          12500/12500 [==============================] - 169s 13ms/step - loss: 2.4911 - accuracy: 0.4101 - val_loss: 2.0169 - val_accura
          cy: 0.4956
          Epoch 2/20
          12500/12500 [==============================] - 174s 14ms/step - loss: 1.9628 - accuracy: 0.5119 - val_loss: 2.0125 - val_accura
          cy: 0.5051
          Epoch 3/20
           5548/12500 [============>.................] - ETA: 1:30 - loss: 1.8584 - accuracy: 0.5364
```


jupyter project_nikkulk Last Checkpoint: 2 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
In [196]: #1st Convolution Layer
          CNNModel.add(Conv2D(64, (3, 3), activation = 'selu', padding='valid', input_shape = (28, 28, 1)))

In [197]: #Downsampling the dataset using Pooling
          CNNModel.add(MaxPooling2D(3, 3))

In [198]: #Second Convolution Layer
          CNNModel.add(Conv2D(64, (3, 3), activation="selu", padding='valid'))
          CNNModel.add(MaxPooling2D(3, 3))

In [199]: #Flattening
          CNNModel.add(Flatten())

In [200]: #adding Input Layer
          CNNModel.add(Dense(units = 512, activation = 'selu', kernel_initializer='glorot_uniform'))

          #Adding Hidden Layer
          CNNModel.add(Dense(units = 256, activation = 'selu', kernel_initializer='glorot_uniform'))

In [201]: #output Layer
          CNNModel.add(Dense(units = 100, activation = 'softmax'))
```

Training the CNN

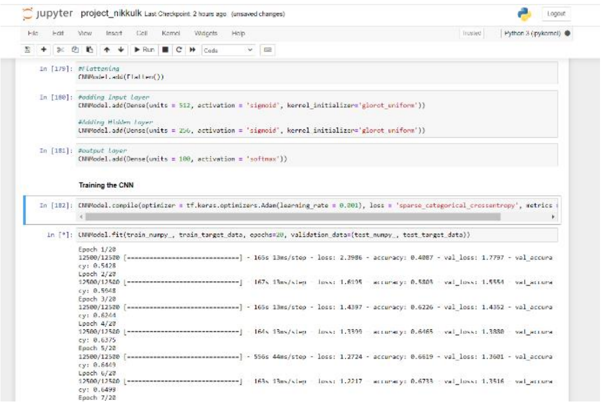
```
In [202]: CNNModel.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001), loss = 'sparse_categorical_crossentropy', metrics =

In [*]: CNNModel.fit(train_numpy_, train_target_data, epochs=20, validation_data=(test_numpy_, test_target_data))

          Epoch 1/20
          12500/12500 [==============================] - 176s 14ms/step - loss: 2.3464 - accuracy: 0.4309 - val_loss: 2.0143 - val_accura
          cy: 0.4955
          Epoch 2/20
          12500/12500 [==============================] - 175s 14ms/step - loss: 1.9084 - accuracy: 0.5248 - val_loss: 1.9601 - val_accura
          cy: 0.5171
          Epoch 3/20
           3012/12500 [======>.......................] - ETA: 2:10 - loss: 1.8086 - accuracy: 0.5490
```



```
CNNModel.add(Conv2D(64, (3, 3), activation="tanh", padding='valid'))
CNNModel.add(MaxPooling2D(3, 3))

In [189]: #Flattening
CNNModel.add(Flatten())

In [190]: #adding Input layer
CNNModel.add(Dense(units = 512, activation = 'tanh', kernel_initializer='glorot_uniform'))

#Adding Hidden layer
CNNModel.add(Dense(units = 256, activation = 'tanh', kernel_initializer='glorot_uniform'))

In [191]: #output layer
CNNModel.add(Dense(units = 100, activation = 'softmax'))
```

Training the CNN

```
In [192]: CNNModel.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001), loss = 'sparse_categorical_crossentropy', metrics :

In [*]: CNNModel.fit(train_numpy_, train_target_data, epochs=20, validation_data=(test_numpy_, test_target_data))
Epoch 1/20
12500/12500 [==============================] - 152s 12ms/step - loss: 2.1341 - accuracy: 0.4701 - val_loss: 1.9549 - val_accura
cy: 0.5083
Epoch 2/20
12500/12500 [==============================] - 152s 12ms/step - loss: 1.8765 - accuracy: 0.5237 - val_loss: 1.8827 - val_accura
cy: 0.5235
Epoch 3/20
 2081/12500 [=====>........................] - ETA: 1:55 - loss: 1.8042 - accuracy: 0.5300
```



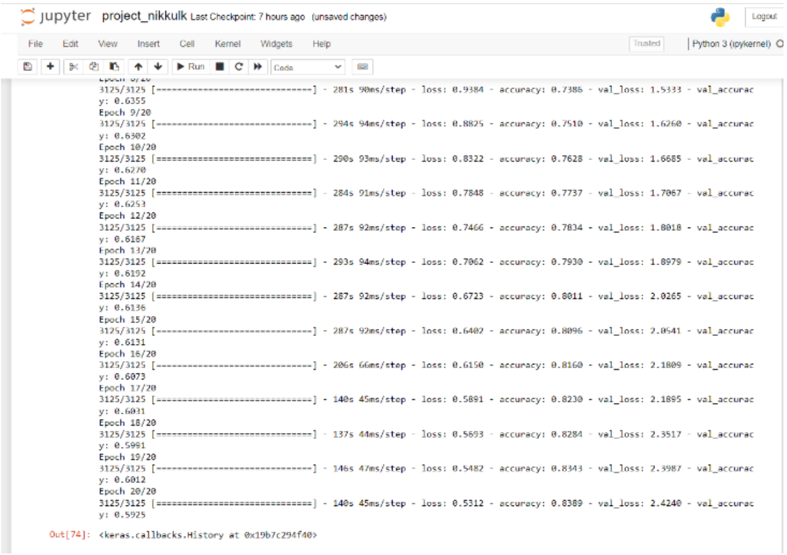
jupyter project_nikkulk Last Checkpoint: 2 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
In [179]: #Flattening
CNNModel.add(Flatten())

In [180]: #adding Input layer
CNNModel.add(Dense(units = 512, activation = 'sigmoid', kernel_initializer='glorot_uniform'))

#Adding Hidden layer
CNNModel.add(Dense(units = 256, activation = 'sigmoid', kernel_initializer='glorot_uniform'))

In [181]: #output layer
CNNModel.add(Dense(units = 100, activation = 'softmax'))
```

Training the CNN

```
In [182]: CNNModel.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001), loss = 'sparse_categorical_crossentropy', metrics :

In [*]: CNNModel.fit(train_numpy_, train_target_data, epochs=20, validation_data=(test_numpy_, test_target_data))
Epoch 1/20
12500/12500 [==============================] - 165s 13ms/step - loss: 2.3986 - accuracy: 0.4087 - val_loss: 1.7797 - val_accura
cy: 0.5428
Epoch 2/20
12500/12500 [==============================] - 167s 13ms/step - loss: 1.6195 - accuracy: 0.5803 - val_loss: 1.5554 - val_accura
cy: 0.5948
Epoch 3/20
12500/12500 [==============================] - 165s 13ms/step - loss: 1.4397 - accuracy: 0.6226 - val_loss: 1.4352 - val_accura
cy: 0.6244
Epoch 4/20
12500/12500 [==============================] - 164s 13ms/step - loss: 1.3399 - accuracy: 0.6465 - val_loss: 1.3880 - val_accura
cy: 0.6375
Epoch 5/20
12500/12500 [==============================] - 556s 44ms/step - loss: 1.2724 - accuracy: 0.6619 - val_loss: 1.3601 - val_accura
cy: 0.6449
Epoch 6/20
12500/12500 [==============================] - 163s 13ms/step - loss: 1.2217 - accuracy: 0.6733 - val_loss: 1.3516 - val_accura
cy: 0.6499
Epoch 7/20
```

- I stopped at an accuracy of 83.89%.



jupyter project_nikkulk Last Checkpoint: 7 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code

```
Epoch 8/20
3125/3125 [==============================] - 281s 90ms/step - loss: 0.9384 - accuracy: 0.7386 - val_loss: 1.5333 - val_accurac
y: 0.6355
Epoch 9/20
3125/3125 [==============================] - 294s 94ms/step - loss: 0.8825 - accuracy: 0.7510 - val_loss: 1.6260 - val_accurac
y: 0.6302
Epoch 10/20
3125/3125 [==============================] - 290s 93ms/step - loss: 0.8322 - accuracy: 0.7628 - val_loss: 1.6685 - val_accurac
y: 0.6270
Epoch 11/20
3125/3125 [==============================] - 284s 91ms/step - loss: 0.7848 - accuracy: 0.7737 - val_loss: 1.7067 - val_accurac
y: 0.6253
Epoch 12/20
3125/3125 [==============================] - 287s 92ms/step - loss: 0.7466 - accuracy: 0.7834 - val_loss: 1.8018 - val_accurac
y: 0.6167
Epoch 13/20
3125/3125 [==============================] - 293s 94ms/step - loss: 0.7062 - accuracy: 0.7930 - val_loss: 1.8979 - val_accurac
y: 0.6192
Epoch 14/20
3125/3125 [==============================] - 287s 92ms/step - loss: 0.6723 - accuracy: 0.8011 - val_loss: 2.0265 - val_accurac
y: 0.6136
Epoch 15/20
3125/3125 [==============================] - 287s 92ms/step - loss: 0.6402 - accuracy: 0.8096 - val_loss: 2.0541 - val_accurac
y: 0.6131
Epoch 16/20
3125/3125 [==============================] - 206s 66ms/step - loss: 0.6150 - accuracy: 0.8160 - val_loss: 2.1809 - val_accurac
y: 0.6073
Epoch 17/20
3125/3125 [==============================] - 140s 45ms/step - loss: 0.5891 - accuracy: 0.8230 - val_loss: 2.1895 - val_accurac
y: 0.6031
Epoch 18/20
3125/3125 [==============================] - 137s 44ms/step - loss: 0.5693 - accuracy: 0.8284 - val_loss: 2.3517 - val_accurac
y: 0.5991
Epoch 19/20
3125/3125 [==============================] - 146s 47ms/step - loss: 0.5482 - accuracy: 0.8343 - val_loss: 2.3987 - val_accurac
y: 0.6012
Epoch 20/20
3125/3125 [==============================] - 140s 45ms/step - loss: 0.5312 - accuracy: 0.8389 - val_loss: 2.4240 - val_accurac
y: 0.5925

Out[74]: <keras.callbacks.History at 0x19b7c294f40>
```

Methods to improve the accuracy:

- If we don't have a time limit, we can add additional hidden layers which can improve the accuracy significantly.
- We can also increase the epochs. (Currently set to 20)
- We can increase the image size, which can add more details to the image, resulting in more data to train.
- We can lower the learning rate, resulting in more details while training. (Currently set to 0.01)