UNIVERSITY OF MARYLAND

BUDT737: Big Data and Artificial Intelligence for Business

Spring 2022

Team Project - Digit Recognizer

Team No : 24

• Bharath Kumar Routhu • Manikanta Koneru

• Rajesh Reddy Bogolu • Tanya Singh

Account Names

Rajesh Reddy Bogolu: ***rajeshreddybogolu***

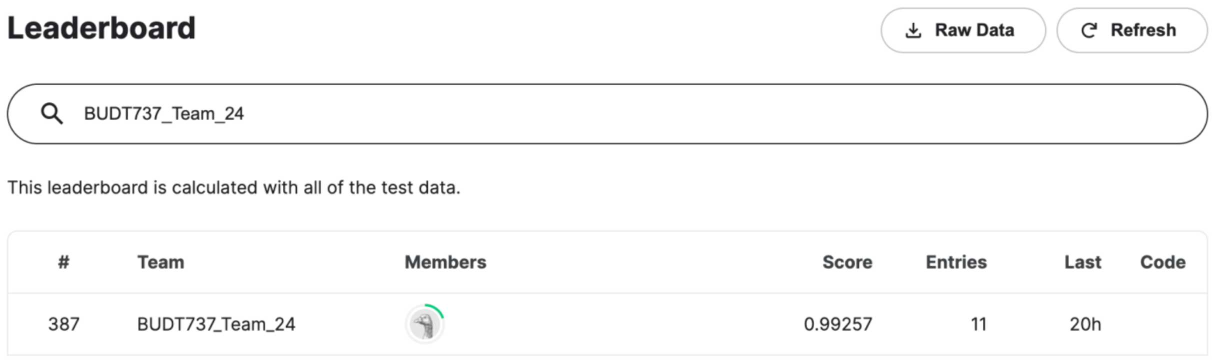
Manikanta koneru: ***manikantakoneru***

Bharath Routhu: ***bharathkumarrouthu***

Tanya Singh: ***tanyasingh01***

Team Name: BUDT737_Team_24

Leaderboard Ranking on Kaggle:



This leaderboard is calculated with all of the test data.

#	Team	Members	Score	Entries	Last	Code
387	BUDT737_Team_24		0.99257	11	20h	

Methods Used:

- **Importing the Data Set**

Imported the Kaggle data directly onto Colab Notebook by leveraging the Kaggle API. Accessing the data from using these API's reduced the dependency of the code on a particular file. This enabled the code to run on any machine with a valid Kaggle API Token in it.

- **Splitting the data for Training, Validation and Testing**

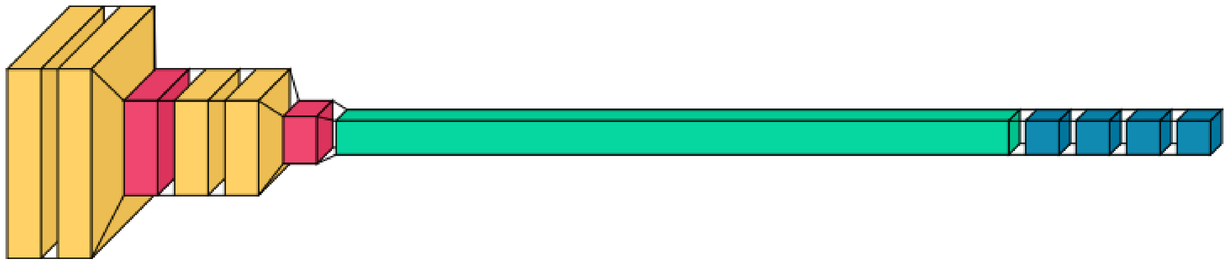
The data in the Excel files has row entries with each column describing the pixel values. This data needs to be transformed to accurately represent a 2 dimensional image. We have reshaped each row to a matrix of dimension [28,28]. This transformed data is used as input for the model.

For Evaluation the data is divided into train and test. The Train data is further split to the validation dataset to avoid generating overfitting models.

- **Model Building**

Developed a Convolutional neural network with 4 2D convolutional network layers with zero padding and kernel window of size(5,5) and filter size of 128. A max pooling layer with poolsize of (2,2) is inserted between every 2 convolutional layers to reduce the size of input to half. There are 4 dense layers with activation function relu present and one output layer with activation function softmax and the batch size of 85 and number of epochs of 25.

The the data is passed into the first convolutional network layer with padding, filter size 128 window size(5,5) and activation function of relu the output of this layer of size(28,28) is passed to the next similar convolutional network layer with the same parameters and the output from the second layer of size(28,28) is then passed to a max pooling layer of pooling size(2,2) and the output of this layer of size(14,14) is passed on to the similar network of 2 convolutional layers and 1 max pooling layer the resulting output from the final max pooling layer is of size(7,7) is flattened and passed to a neural network of 3 dense layers with node size of 128, 64, 64 and with activation function of "relu" for all and the output layer has a node size of 10 with activation function as "softmax".



Hyper parameter Tuning using GridSearchCV :

Leveraged the *GridSearchCV* library to optimize the model with hyper parameter tuning. We have experimented with multiple combinations of epoch and Batch size.

The model produced best results when compiled with following parameters:

optimizer : "adam"

loss function: "sparse_categorical_crossentrop"

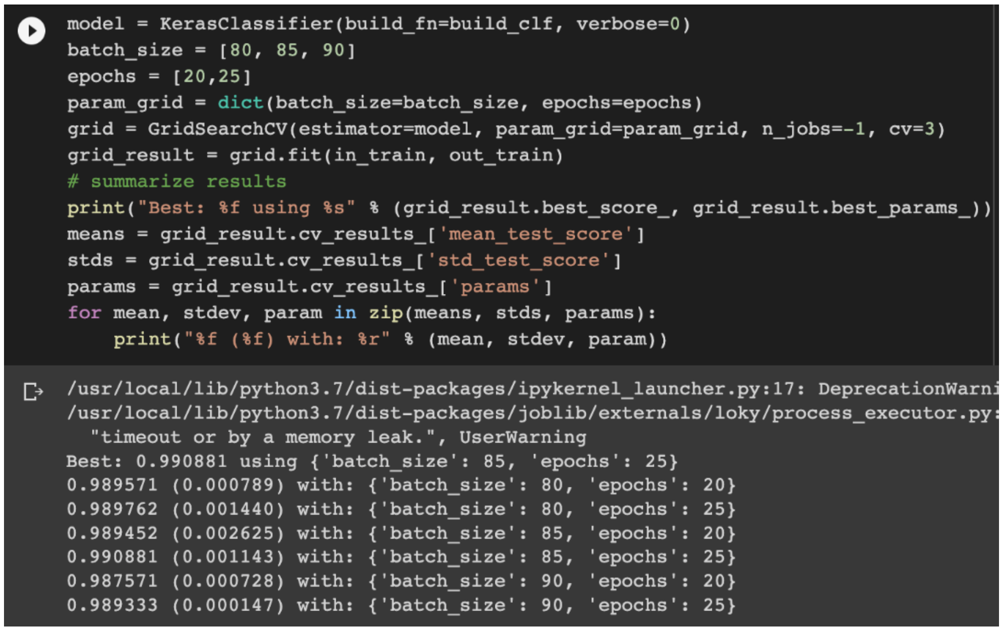
performance metrics : ["accuracy " ,"categorical accuracy"]

Batch Size : **"85"**

epochs : **"25"**

validation split : **"0.2"**.

The Same can be inferred from below GridSearchCV result:



```
model = KerasClassifier(build_fn=build_clf, verbose=0)
batch_size = [80, 85, 90]
epochs = [20,25]
param_grid = dict(batch_size=batch_size, epochs=epochs)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(in_train, out_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:17: DeprecationWarni
/usr/local/lib/python3.7/dist-packages/joblib/externals/loky/process_executor.py:
  "timeout or by a memory leak.", UserWarning
Best: 0.990881 using {'batch_size': 85, 'epochs': 25}
0.989571 (0.000789) with: {'batch_size': 80, 'epochs': 20}
0.989762 (0.001440) with: {'batch_size': 80, 'epochs': 25}
0.989452 (0.002625) with: {'batch_size': 85, 'epochs': 20}
0.990881 (0.001143) with: {'batch_size': 85, 'epochs': 25}
0.987571 (0.000728) with: {'batch_size': 90, 'epochs': 20}
0.989333 (0.000147) with: {'batch_size': 90, 'epochs': 25}
```

Note : This result is not included in the notebook due to Run time limitation

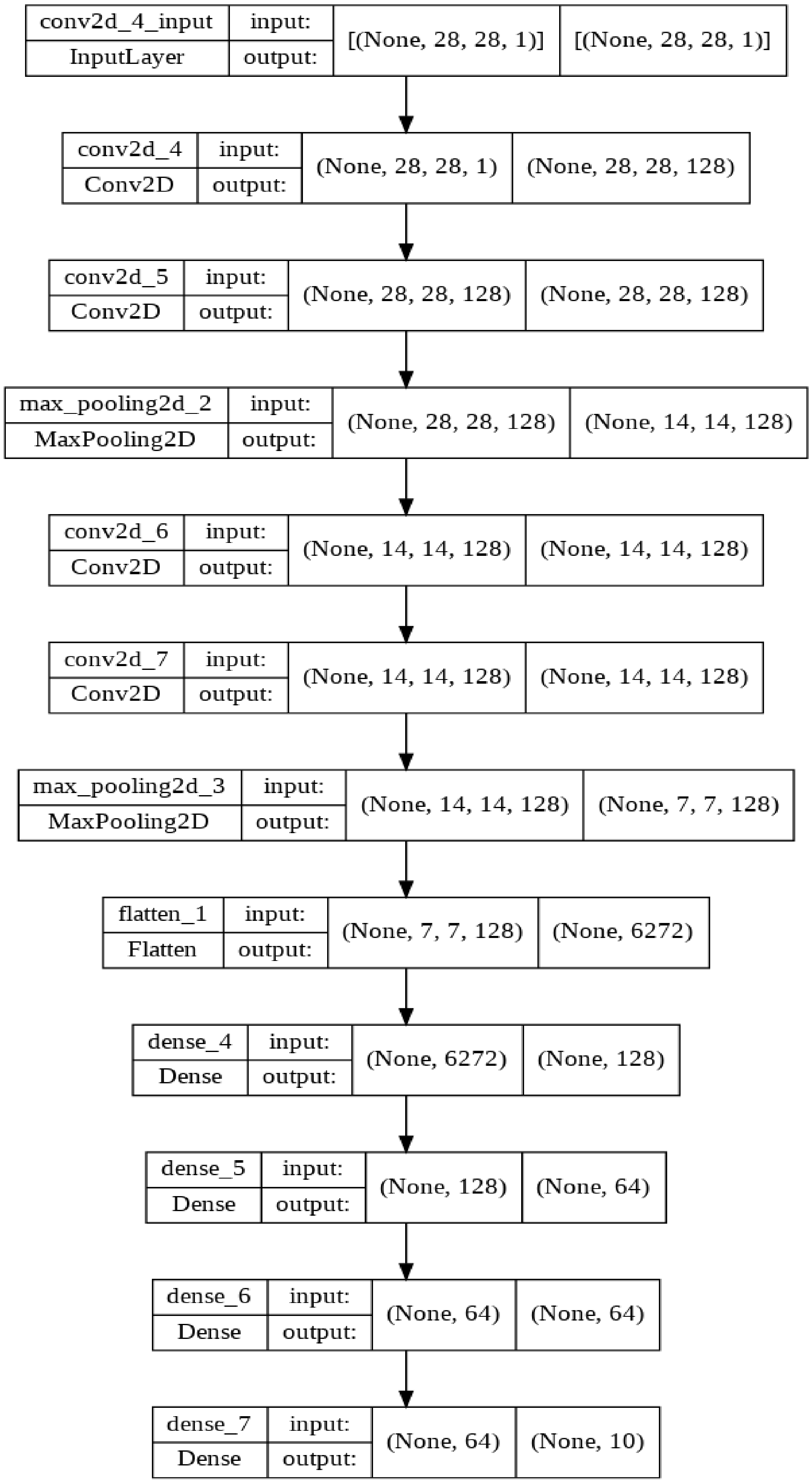
Dimensionality Reduction is one key feature to consider in neural networks and this can be incorporated but using suitable layers and activation methods in neural networks:

ReLu : The algorithm is used to filter information as it travels through the network.ReLU helps to prevent the exponential growth in the computation required to operate the neural network. It does not activate all the neurons at the same time hence is great for CNN layer activations.

Max Pooling : The main idea of a Pooling layer is to reduce the dimensions input. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network.

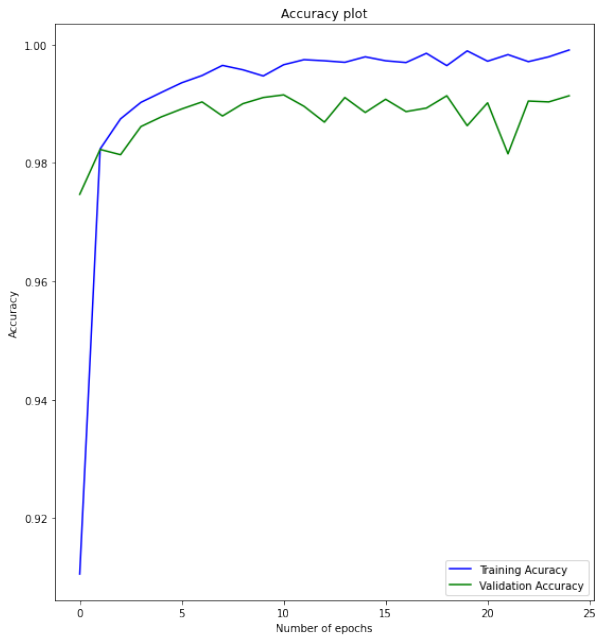
Kernel: The height and width of the window size for the convolution layer should be specified as a 2 integer list.It is using this window the convolution happens based on the stride length given.We tried multiple kernel sizes [3,3], [5,5], [7,7] etc and of all those we got the best accuracy for [5,5].

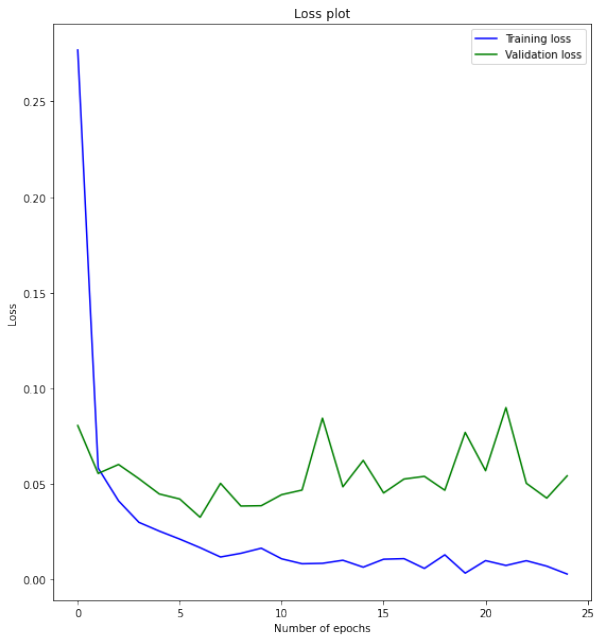
Padding: Padding adds additional rows and columns to the outer dimensions of the image based on the window size and stride length. Padding is done to even preserve the information of the image at the edges. We used padding='same' to consider the edges and make inferences from them as well. Since the stride length is 1 the size of the output and input remains unchanged.


conv2d_4_input | InputLayer | input: | output: | [(None, 28, 28, 1)] | [(None, 28, 28, 1)]
conv2d_4 | Conv2D | input: | output: | (None, 28, 28, 1) | (None, 28, 28, 128)
conv2d_5 | Conv2D | input: | output: | (None, 28, 28, 128) | (None, 28, 28, 128)
max_pooling2d_2 | MaxPooling2D | input: | output: | (None, 28, 28, 128) | (None, 14, 14, 128)
conv2d_6 | Conv2D | input: | output: | (None, 14, 14, 128) | (None, 14, 14, 128)
conv2d_7 | Conv2D | input: | output: | (None, 14, 14, 128) | (None, 14, 14, 128)
max_pooling2d_3 | MaxPooling2D | input: | output: | (None, 14, 14, 128) | (None, 7, 7, 128)
flatten_1 | Flatten | input: | output: | (None, 7, 7, 128) | (None, 6272)
dense_4 | Dense | input: | output: | (None, 6272) | (None, 128)
dense_5 | Dense | input: | output: | (None, 128) | (None, 64)
dense_6 | Dense | input: | output: | (None, 64) | (None, 64)
dense_7 | Dense | input: | output: | (None, 64) | (None, 10)

- **Predicting Metrics and Performance Measures**

  Loss function used is ‘`sparse_categorical_crossentropy`’ and ‘adam’ as optimizer. The performance measures used are `‘accuracy’,’categorical_accuracy’` .





For the model we built we got an accuracy of **0.99257** for the **Kaggle submission file.**

- **Summary and Learning Outcome**

  Predictors are the most important aspect of an Explanatory or Predictive Model. However data doesn’t also come with well defined predictors that help us make a logical deduction on their importance or impact on the dependent variable. Neural networks offer a solution to tackle such problems. Where the model tries to assign weights to each predictor based on the prediction metrics and update it on every iteration.

  Considering the dataset we currently possess. It is highly unlikely to come up with a model or relation that would best define the number. Whereas by implementing a neural network model we could develop multiple solutions with accuracies as high as 99.2%. We can further tune the model to produce results aimed at a certain criteria of interest such as execution time, accuracy, simplicity, etc.