# Project 2 – Digit Recognizer

Ilan Valencius

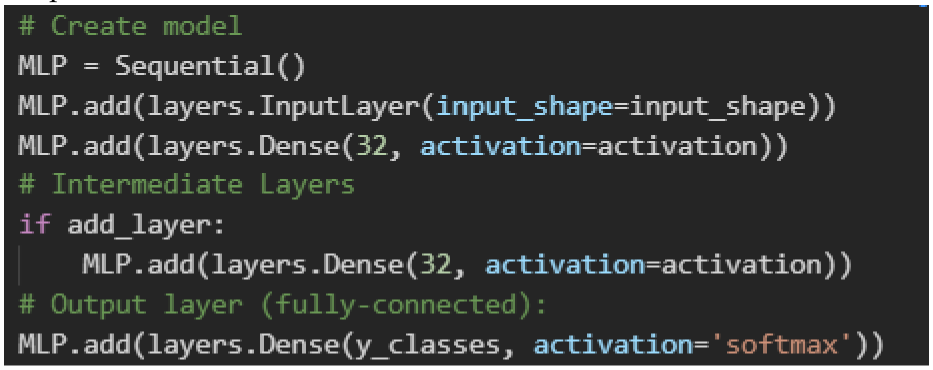
*Summary*

Three model families (CNN, MLP, SVM) were trained then evaluated on the validation set. The model that had the highest validation set accuracy was then determined to be the "best model" and was evaluated on the test set. This ended up being a 7 layer CNN with a 'softsign' activation for its layers. In the real world, this classifier should perform with 98.8% accuracy (determined by evaluating on the test set). This approach is appropriate as it can be extended to test an arbitrarily large number of machine learning models. Hard-coding a classifier to recognize digits would be impractical so machine learning is the appropriate solution.

*Technical Report*

Note: You may notice that the loss and accuracy plots for the CNN and MLP don't look how they are supposed to. I believe this can be a result of the EarlyStopping callback reverting the weights to a previous epoch but I am not sure. These plots aren't really important so they can be ignored if they look funky but they usually plot correctly. Run the code with 'MLP_plotting' and 'CNN_plotting' set to 'True' to see examples of correct plots when training.

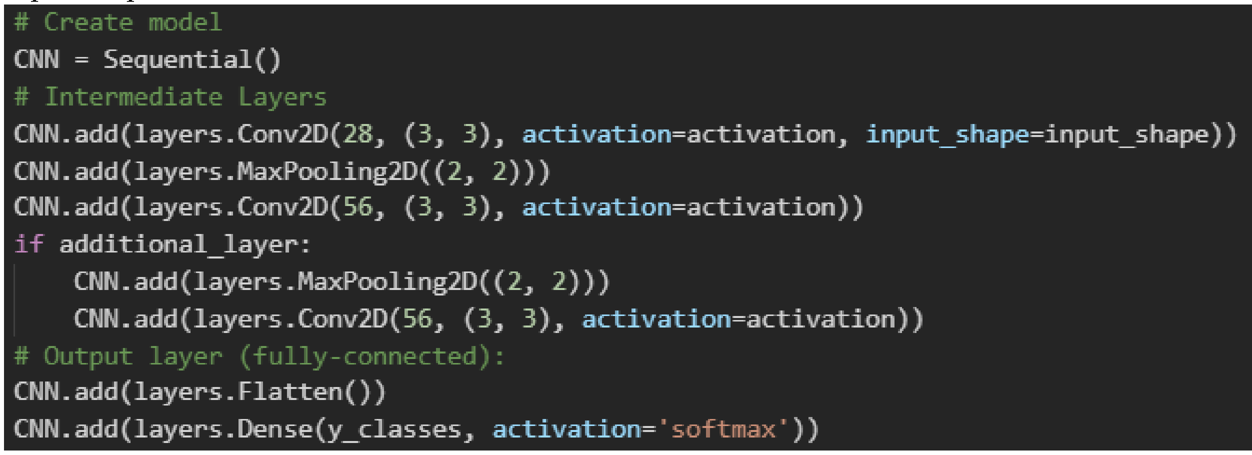
- **Preprocessing steps**:
  - Images and labels split into train – validation – test sets with (0.56 - 0.24 - 0.2) split. Images were then normalized using the *scale()* function. For the CNN and MLP models, labels were converted into one-hot-encoded vectors using the *to_categorical()* function.
- **Candidate algorithms and choice of candidate hyperparameters (and why were the others left out?):**
  - Multilayer perceptron (MLP): Chosen due to simple architecture. With an input shape of 784 features, the architecture is as follows:



```
# Create model
MLP = Sequential()
MLP.add(layers.InputLayer(input_shape=input_shape))
MLP.add(layers.Dense(32, activation=activation))
# Intermediate Layers
if add_layer:
    MLP.add(layers.Dense(32, activation=activation))
# Output layer (fully-connected):
MLP.add(layers.Dense(y_classes, activation='softmax'))
```

Dense layers were arbitrarily chosen to have an output dimensionality of 32. Hyperparameters are the addition of an extra dense, fully connected layer, as well as every available activation function besides 'selu' and 'elu'. These two are just variations on the relu activation so are ignored so as to decrease computation time.

- Convolutional Neural Network (CNN): Chosen due to intuitive input (28 x 28 images) and the use case for CNNS → to extract information from images. The input shape is 28x28x1 channel and the architecture is as follows.



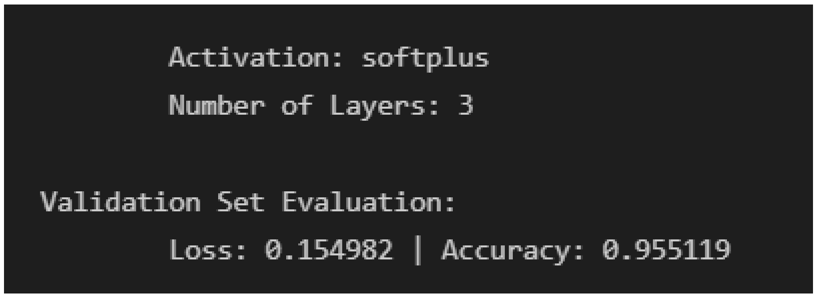
```
# Create model
CNN = Sequential()
# Intermediate Layers
CNN.add(layers.Conv2D(28, (3, 3), activation=activation, input_shape=input_shape))
CNN.add(layers.MaxPooling2D((2, 2)))
CNN.add(layers.Conv2D(56, (3, 3), activation=activation))
if additional_layer:
    CNN.add(layers.MaxPooling2D((2, 2)))
    CNN.add(layers.Conv2D(56, (3, 3), activation=activation))
# Output layer (fully-connected):
CNN.add(layers.Flatten())
CNN.add(layers.Dense(y_classes, activation='softmax'))
```

  The filter size starts at 28 (same dimension as image) then is doubled to 56 for following layers. The hyper parameters and justifications were the same as the MLP as they are both built using the tensorflow framework.

- Both CNN and MLP are compiled using 'adam' optimizer, 'categorical_crossentropy' loss (due to multiclass classification), and 'accuracy' metric.

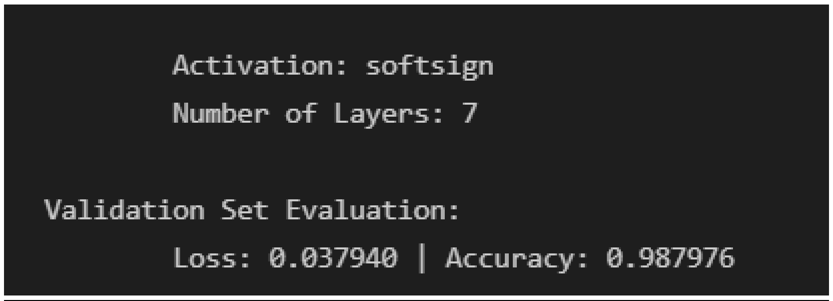
- **SVM:** A support vector machine architecture implemented in sklearn using *SVC()*. Hyper parameters include C, regularization parameter, and the kernel. Every kernel was used except 'linear' as 'rbf' is an optimized version of linear so will always perform better. Values of C used are 0.5, 1, 2. C defaults to 1 so the value of C is set to below, equal to, and above, this default value

- **Chosen Performance Measure/ Model Selection Schemes:** Models were first evaluated based on accuracy on the validation data set. The most accurate model (on validation data) of each type was then selected and compared with the overall most accurate (on validation data) being analyzed on the test set.
  - Best MLP model:

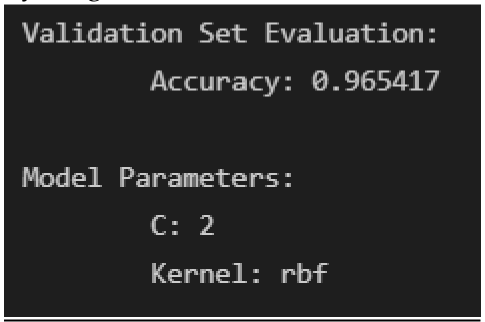


```
        Activation: softplus
        Number of Layers: 3

Validation Set Evaluation:
        Loss: 0.154982 | Accuracy: 0.955119
```

  - Best CNN model:

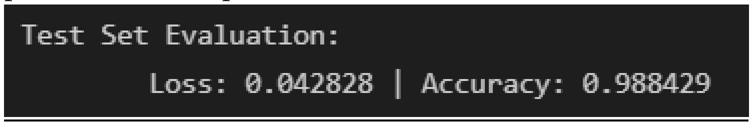
- 

```
        Activation: softsign
        Number of Layers: 7

Validation Set Evaluation:
        Loss: 0.037940 | Accuracy: 0.987976
```

- Best SVM model: As expected, SVM's are not easily scalable and execution takes extremely long.
  - 

```
Validation Set Evaluation:
        Accuracy: 0.965417

Model Parameters:
        C: 2
        Kernel: rbf
```

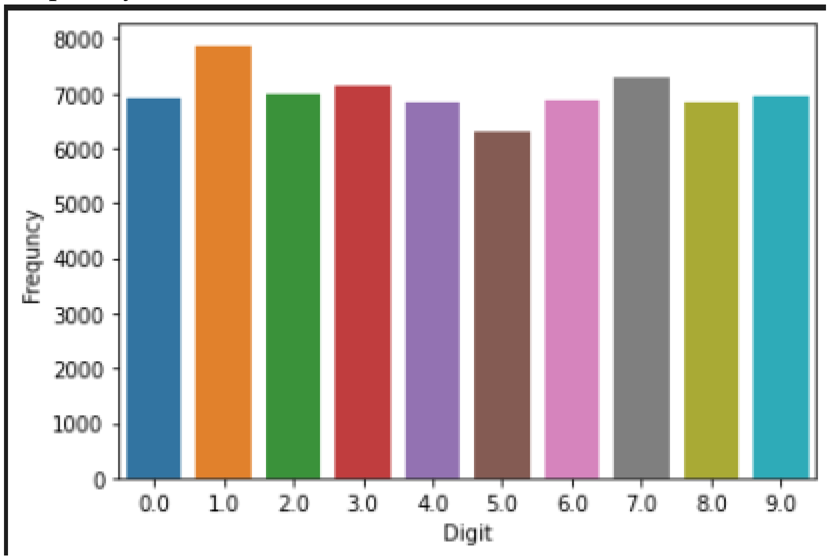
- **Final Classifier/How well it performs on unseen data:**
  - Based on the above validation set accuracies. The CNN model was selected. The full report can be seen at the bottom of the notebook but it has 7 layers, the activation hyper parameter was 'softsign' and there were 47,778 trainable parameters. The performance on the test set/unseen data is as follows:



```
Test Set Evaluation:
        Loss: 0.042828 | Accuracy: 0.988429
```

- **Measures taken to avoid overfitting:**
  - For the MLP and CNN models, I implemented the "EarlyStopping" callback. In my implementation, model training stops if validation loss does not decrease significantly after 3 epochs. 'restore_best_weights' is set to True so if training is halted, the model restores the weights of the epoch with the best validation loss. Validation accuracy was not used as validation accuracy can increase dramatically as the model starts to over fit but this is not optimal for generalizing the model.
- **Given more resources (time or computing resources), how would you improve your solution?**
  - Given more time, more values of C could have been tested for the SVM. In the MLP model, various 'units' could be tried in the dense layers and even more layers could be added. For the CNN, different dropouts and architectures could be tested. I could also use cross-validation. This was infeasible for this project as using 5 folds would scale the running time by 5 and all models/hyper parameters

already took over an hour to run. For all three models, the use of a *ImageDataGenerator()* in tensorflow would allow for the training dataset to be expanded arbitrarily. This would allow for a more extensive training set that would take more time, but would be more accurate. The use of a 'class_weight' dictionary in the MLP and CNN models would have improved accuracy as well due to the fact that the class distributions are not even. This would slightly weight the data, giving more precedence to numbers (such as 5) which occur less frequently. See:


Frequncy
Digit