BT4012 Kaggle Report

By: Prawira Satya Darma (A00200684L)

# Data Preparation

In this section, I will discuss about data preparation techniques which I applied to the data to make the data compatible for training. In machine learning, data preparation is an important step to create a robust predictive model. The data preparation in this project is divided into 2 sections, data preprocessing and feature engineering.

## Data Preprocessing

Data should be processed first before it is used to build a predictive model. In this project, our data an array of 401 columns. The columns are “label” which indicates the labelling of the row, whether it is 0 or 1, and rxcy which indicates the pixel intensity value for position (x,y) where r stands for row and c stands for column and x,y has values from 1 to 20. As a result, after dividing into X and y data, I need to convert X as an array and reshape X into [20,20,1]. Moreover, each element in the array has a value between 0-255. Since several activation functions, including Sigmoid function, work better with a data range of 0.0 – 1.0, I normalize each element by dividing each element by 255.

## Feature Engineering

Feature engineering is the next stage in improving the performance of machine learning models after ensuring data is compatible with machine learning algorithms through data preparation. One feature engineering method in deep learning is batch normalization, a technique for standardizing the input to a layer for each mini batch. Batch normalization accelerates training and provides some regularization which reduces generalization error. Throughout the project, I experimented more with CNN and found that batch normalization has improved my model performance significantly.

# Model

In this section, I will discuss the final model I used and its layers which was used to predict the test dataset. Moreover, I will discuss the model validation method which was used to compare the results of several models.

## Model Building

Text

Description automatically generated

Text

Description automatically generated

Figure 1: CNN Model

Figure 2 shows the function that will create our CNN model.

The first layer is Conv2D layer with 96 filters, 3x3 kernel size, and input shape of (20, 20, 1) which is the same as what the train data has been reshaped into since the input will be in 2 dimensions. This layer creates a 3x3 convolution kernel which creates a feature map that summarizes the features of the input. This Conv2D layer is the first layer which will extract the features from our data. I specified the padding parameter to be “same” not the default value because I will create another Conv2D layer, so I need the result of this layer to have the same spatial dimensions as its input. Specifying the padding parameter would achieve this desired result.

After the convolution layer, I add a non-linear activation function which is ReLU (rectified linear unit). This non-linear activation function will introduce non-linearity and help the model to learn complex pattern in the data. As a result, this layer will improve the predictive power of the model.

Furthermore, in order to reduce input parameter, I use maximum pooling by adding MaxPooling2D(pool\_size = (2,2)) which gets the maximum value on each patch of 2 x 2 matrix. I tried to implement average pooling and minimum pooling other than maximum pooling. I found that maximum pooling gives better results in general than average pooling and minimum pooling. This could be due to the fact that most of the pixel intensity value is 0 (black). Applying maximum pooling would select the higher intensity pixels and accentuate important features in the data.

Moreover, I also added a batch normalization which standardizes the outputs and inputs to a layer for each mini batch. It also stabilizes and accelerates the learning process of the model and reduce the number of training epochs required to train the model. Moreover, it also provides some regularization which reduces generalization error.

After those 4 layers, I added the same 4 layers in the architecture of the model. However, I changed the number of layers in the second Conv2D layer to be 128 for variation. The parameters for the other layers were the same as the 4 layers before this.

I added a dropout layer after the 8 layers with Dropout(0.8). I experimented with adding a dropout layer before the dense layers and turns out that it improved the model performance. In addition, I tried changing the value of the dropout rate and found that 0.8 would give the best results.

The next layer is the flatten layer. It is used to flatten the matrix and reshape it so it can be passed to the dense layers. After the flatten layer, I added two dense layers with 128 neurons. Dense layers perform a matrix-vector calculations and are used for classification based on the feature data that has gone through feature extraction layers explained before. Each dense layer require an activation function and I used ReLU activation function. I experimented with other activation function such as tanh, but I found that ReLU produced better results in general. I added a dropout layer with Dropout(0.2) after each dense layer to implement regularization.

The last layer is Dense(1) since the label is scalar and the activation function is sigmoid. Moreover, I compiled my model with Adam optimizer and learning rate equals to 0.0001. I started with the default learning rate (0.001) and divided the learning rate by 10. I found that 0.0001 would be the best learning rate for my architecture. I also added early stopping to ensure that the model is not overfitting and could restore the best weights that the model had found. My early stopping callback would monitor the validation loss and would stop if the validation loss did not improve 0.0001 in 10 epochs. I specified the batch size to be 50. I found that smaller batch sizes would give better performance as I started at 4000 and as I keep decreasing the batch size, the performance would increase.

## Model Validation

I split the train dataset into 2 parts, train and validation sets to provide an unbiased evaluation of a model. The train set is used to fit the model and the validation set is used to evaluate the performance of the fitted model. I used train\_test\_split to split the data into train and validation sets with ratio 0.2 for the validation set. The shuffle parameter is specified to be True to ensure that the function will randomly sample from the data to split into train and validation sets. Moreover, I specified the random\_state parameter to be 1 so the results could be reproducible.

# Finalization

In this section, I will discuss all the models that I explored and the reason why I choose the final model.

## Explored Models

There are several models other than CNN I have explored. These models include simple models such as logistic regression, k-NN, and XGBoost and deep learning models such as RNN. I wanted to explore those models as they use different algorithms, and I would like to see which algorithm would give the best performance. Logistic regression applies regression, k-NN applies nearest neighbor, and XGBoost applies tree-based boosting method. I tried to implement RNN since RNN provides different type of deep learning model than CNN. I compared each model’s performance to see which model is the best. Table 1 below shows the validation dataset’s accuracy and AUC score of each model.

|  |  |  |
| --- | --- | --- |
| Model | Validation Accuracy | AUC Score |
| Logistic Regression | 0.9242 | 0.8372 |
| k-NN | 0.9164 | 0.6291 |
| XG Boost | 0.9359 | 0.8471 |
| RNN | 0.9624 | 0.9492 |
| CNN | 0.9836 | 0.9923 |

Table 1: Models and their Respective Validation Accuracy

The results indicate that logistic regression and k-NN, which are simple models, performed worse that the other models based on the validation accuracy and AUC score. XGBoost has a higher validation accuracy and AUC score than those models but still performed worse compared to the deep learning models, CNN and RNN. Between the 2 deep learning models I tried, CNN performed better. As a result, I chose CNN as my final model.

## Chosen Model

As mentioned in the previous subsection, I decided to use CNN after evaluating the results of other models I explored. CNN has the advantage in the convolution layer which automatically detects the important features without human supervision. Moreover, since the dataset has 400 columns, CNN can be computationally efficient since it uses convolution and pooling operations. Moreover, as I mentioned in section 2.1, I experimented with various hyperparameters and architectures to find the best model based on the validation loss and accuracy.

# Conclusion and Recommendation

In this section, I will provide a brief conclusion of this report and some suggestions on how to improve the model further given more time.

## Conclusion

In conclusion, this project starts from reshaping the data to a 20x20 array since each row has 20 rows and 20 columns and normalizing it by dividing each value by 255. After reshaping, I divided the data into train set and validation set. Then, I explored several models including Logistic Regression, k-NN, XGBoost, RNN, and CNN to use on the dataset. I decided to use CNN by evaluating the performance results of the models when tested on the validation dataset. In creating the CNN model, I explored different architectures, hyperparameter values, and applied batch normalization, a feature engineering technique. I found that some ways to improve the model’s performance, most notably by applying batch normalization and reducing the batch size.

## Recommendation

1. **Hyperparameter Tuning**

The parameters used for my model may not be the optimal parameters to use. Since we may only use open-source non-auto ml tools, I had to manually experiment with different hyperparameter values. In order to have a better model performance, auto hyperparameter tuning can be done to obtain the best parameter to use. From hyperparameter tuning, we could get the best combination of hyperparameter values including the learning rate, batch size, and number of epochs. Different architectures may produce different optimal hyperparameter values so we could compare the best performances of different architectures. Some tools that can be used for hyperparameter tuning include GridSearchCV and RandomizedSearchCV.

1. **Creating More Complex Models**

More complex CNN models could have been made to increase the performance of the model on the dataset. More complex architecture may have better performance as more complex model may be able to better understand the patterns of data. However, we need to be wary of overfitting and apply regularization methods including those that have been implemented in my model (dropout, early stopping, etc.)