San Francisco Crime Prediction

MLDM Lab 11

Introduction

The San Francisco Crime Prediction challenge is a competition to forecast criminal activity in the city using past crime data and machine learning. The goal is to help prevent and reduce crime. This project was implemented by

- Jon Defilla (defiljon)
- Armando Shala (shalaar3)
- Omer Asipi (asipiome)

Details of the model

To begin, we transformed the "Category" column into a one-hot encoded array. This is a widely-used method for representing categorical data. We then removed any unnecessary columns from the resulting dataframe.

```
# convert the category column to a one hot encoded array
y_train = pd.get_dummies(train["Category"])
class_names = y_train.columns
x_train = train.drop(["Category", "Descript", "Resolution"], axis=1)
x_train
```

	Dates	DayOfWeek	PdDistrict	Address	X	Y
0	2015-05-13 23:53:00	Wednesday	NORTHERN	OAK ST / LAGUNA ST	-122.425892	37.774599
1	2015-05-13 23:53:00	Wednesday	NORTHERN	OAK ST / LAGUNA ST	-122.425892	37.774599
2	2015-05-13 23:33:00	Wednesday	NORTHERN	VANNESS AV / GREENWICH ST	-122.424363	37.800414
3	2015-05-13 23:30:00	Wednesday	NORTHERN	1500 Block of LOMBARD ST	-122.426995	37.800873
4	2015-05-13 23:30:00	Wednesday	PARK	100 Block of BRODERICK ST	-122.438738	37.771541
...	...	...	...	...	...	...
878044	2003-01-06 00:15:00	Monday	TARAVAL	FARALLONES ST / CAPITOL AV	-122.459033	37.714056
878045	2003-01-06 00:01:00	Monday	INGLESIDE	600 Block of EDNA ST	-122.447364	37.731948
878046	2003-01-06 00:01:00	Monday	SOUTHERN	5TH ST / FOLSOM ST	-122.403390	37.780266
878047	2003-01-06 00:01:00	Monday	SOUTHERN	TOWNSEND ST / 2ND ST	-122.390531	37.780607
878048	2003-01-06 00:01:00	Monday	BAYVIEW	1800 Block of NEWCOMB AV	-122.394926	37.738212

878049 rows × 6 columns

The resulting dataframe will contain one column for each possible category in the original "Category" column. Each row will have a value of 1 in the column corresponding to its category, and 0 in all other columns.

	DayOfWeek	Year	Month	Day	Hour	Minute	WeekOfYear	X	Y	PdDistrict_BAYVIEW	...	Address_SF	Address_EX	Address_MT	Address_ST	Address_AL
186723	0.666667	0.750000	0.909091	0.500000	0.956522	0.000000	0.88	0.159725	0.709756	0.0	...	0.0	0.0	0.0	0.0	0.0
678529	0.000000	0.166667	0.818182	0.066667	0.826087	0.000000	0.76	0.159872	0.709597	0.0	...	0.0	0.0	0.0	0.0	0.0
725414	1.000000	0.166667	0.000000	0.966667	0.000000	0.254237	0.04	0.159974	0.710002	0.0	...	0.0	0.0	0.0	0.0	0.0
786787	1.000000	0.083333	0.181818	0.666667	0.521739	0.762712	0.2	0.159933	0.709596	0.0	...	0.0	0.0	0.0	0.0	0.0
345677	0.500000	0.583333	0.545455	0.933333	0.913043	0.440678	0.56	0.159939	0.710028	0.0	...	0.0	0.0	0.0	0.0	0.0
295482	0.833333	0.666667	0.272727	0.733333	0.434783	0.508475	0.28	0.159989	0.709874	0.0	...	0.0	0.0	0.0	0.0	0.0
126556	0.666667	0.833333	0.727273	0.166667	0.304348	0.084746	0.68	0.159918	0.709784	0.0	...	0.0	0.0	0.0	0.0	0.0
301116	0.666667	0.666667	0.181818	0.800000	0.391304	0.000000	0.2	0.159979	0.709888	0.0	...	0.0	0.0	0.0	0.0	0.0
513867	0.000000	0.416667	0.090909	0.566667	0.869565	0.084746	0.12	0.159977	0.709977	0.0	...	0.0	0.0	0.0	0.0	0.0
873128	0.666667	0.000000	0.000000	0.766667	0.826087	0.000000	0.04	0.160012	0.709789	1.0	...	0.0	0.0	0.0	0.0	0.0
590685	0.833333	0.250000	1.000000	0.966667	0.347826	0.508475	1.0	0.159934	0.709843	0.0	...	0.0	0.0	0.0	0.0	0.0
78080	0.500000	0.916667	0.363636	0.000000	0.043478	0.440678	0.32	0.159889	0.709532	0.0	...	0.0	0.0	0.0	0.0	0.0
329565	0.500000	0.583333	0.818182	0.666667	0.826087	0.254237	0.8	0.160002	0.709994	0.0	...	0.0	0.0	0.0	0.0	0.0
447291	0.833333	0.500000	0.000000	0.766667	0.000000	0.508475	0.04	0.159854	0.709829	0.0	...	0.0	0.0	0.0	0.0	0.0
254222	0.000000	0.666667	1.000000	0.366667	0.695652	0.000000	0.96	0.159941	0.709805	0.0	...	0.0	0.0	0.0	0.0	0.0
708038	0.166667	0.166667	0.363636	0.066667	1.000000	0.508475	0.32	0.159947	0.709761	0.0	...	0.0	0.0	0.0	0.0	0.0
490349	1.000000	0.416667	0.454545	0.466667	0.000000	0.728814	0.44	0.159937	0.709689	0.0	...	0.0	0.0	0.0	0.0	0.0
313874	0.666667	0.666667	0.000000	0.433333	0.000000	0.016949	0.0	0.159800	0.709705	0.0	...	0.0	0.0	0.0	0.0	0.0
752598	0.666667	0.083333	0.727273	0.066667	0.695652	0.593220	0.68	0.159791	0.709663	0.0	...	0.0	0.0	0.0	0.0	0.0
524429	0.833333	0.333333	1.000000	0.466667	0.478261	0.000000	0.96	0.159766	0.709858	0.0	...	0.0	0.0	0.0	0.0	0.0

20 rows × 49 columns

Our data preprocessing begins by transforming the dates into a datetime object and generating new columns for the year, month, day, hour, minute, day of the week, and week of the year. We also utilize one-hot encoding for both the "PdDistrict" and "Address" columns, and drop the original "Dates" and "PdDistrict" columns. To ensure optimal performance, we normalize the data using min-max normalization. To facilitate efficient processing, we convert the data into a numpy array. To properly evaluate our model's performance, we split the data into a training and validation set. For even further optimization, we randomly sample a specified percentage of the training data. To better understand the influence of time and location on our data, we separate these factors and analyze them individually. Once we have gained insight from this analysis, we combine the time and location data and split it into training and validation sets once again.

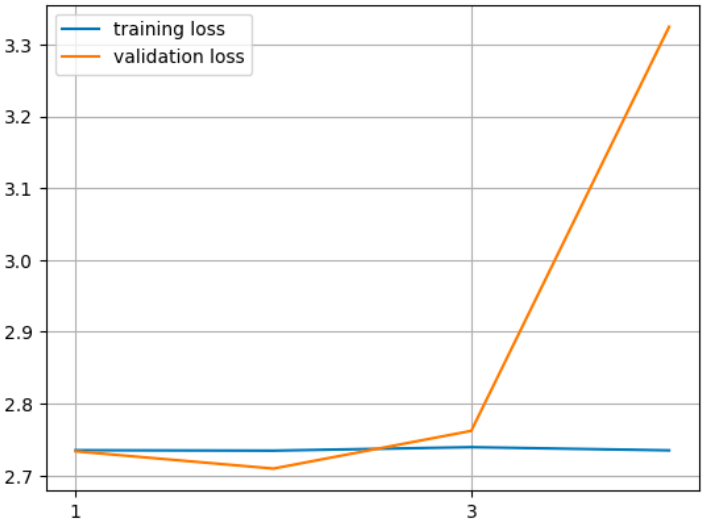
Our code includes functions for visualizing the training process, creating neural network layers, and building residual blocks to improve the performance of our machine learning model. The plot_learning_curves function allows us to see how well the model is doing by plotting the training and validation loss over time. The DenseReluBatchNorm function creates a simple neural network layer with a dense layer, ReLU activation, and batch normalization. The residual_block functions build residual blocks, which improve the flow of information through the model and reduce the risk of vanishing gradients. The create_model function brings all of these together to build the final machine learning model.

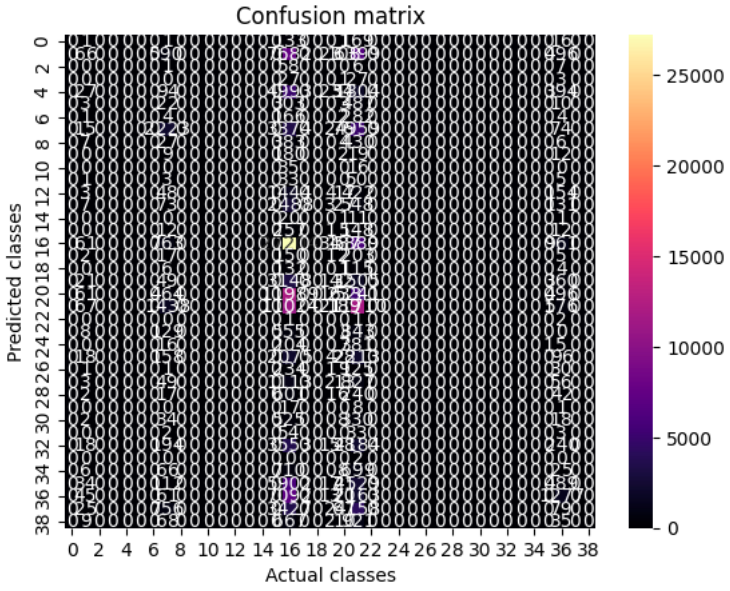
This machine learning model is built using two types of residual blocks - a parallel connection block and a shortcut connection block. Both of these blocks consist of multiple dense layers with ReLU activation and batch normalization. The first block has 64 and 32 units in its dense layers, while the second block has 128 and 64 units in its dense layers. The model also has an output layer, which is a dense layer with softmax activation and 39 units. The softmax activation is used because we are doing multi-class classification, where the model needs to predict one of several possible classes. To train this model, we use the Adam optimizer and the categorical crossentropy loss function. We also track the accuracy of the model as a metric. Our model is designed to train quickly and efficiently by using a batch size of 32 and only 10% of the available training data. We're using the Adam optimizer and the categorical crossentropy loss function to ensure that our model is as accurate as possible. We'll be training for a max of 30 epochs, with the model automatically stopping if the validation loss doesn't decrease for 7 epochs - this is to prevent overfitting and ensure that our model is as generalizable as possible.

We didn't need to rely on any external data sources - our team had all the information we needed to succeed. We kept things in-house and used only internal resources to power our project. We didn't need to look beyond our own walls for data - we had everything we needed right here

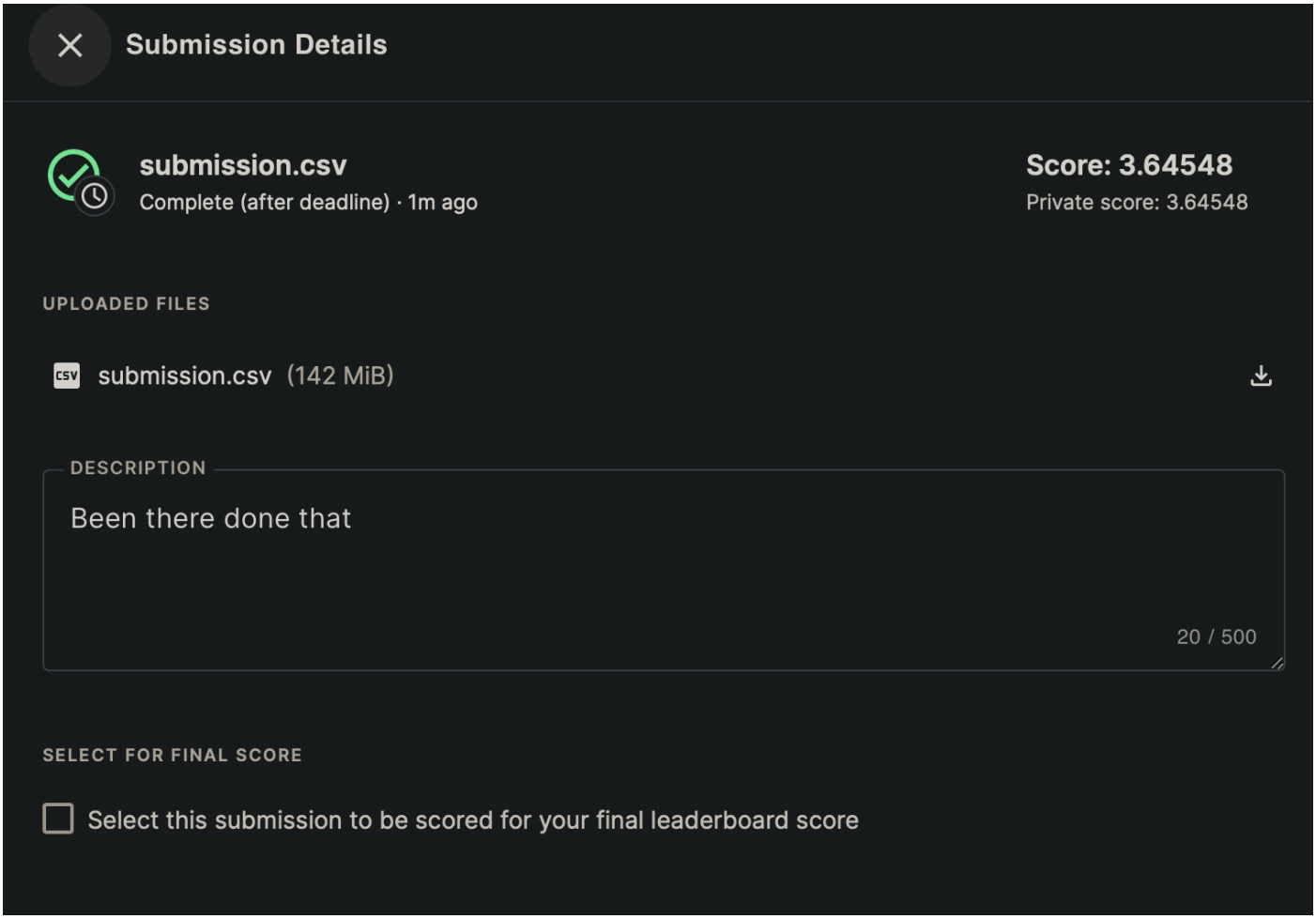
Please see at the end for a visual representation generated by Keras of our model.

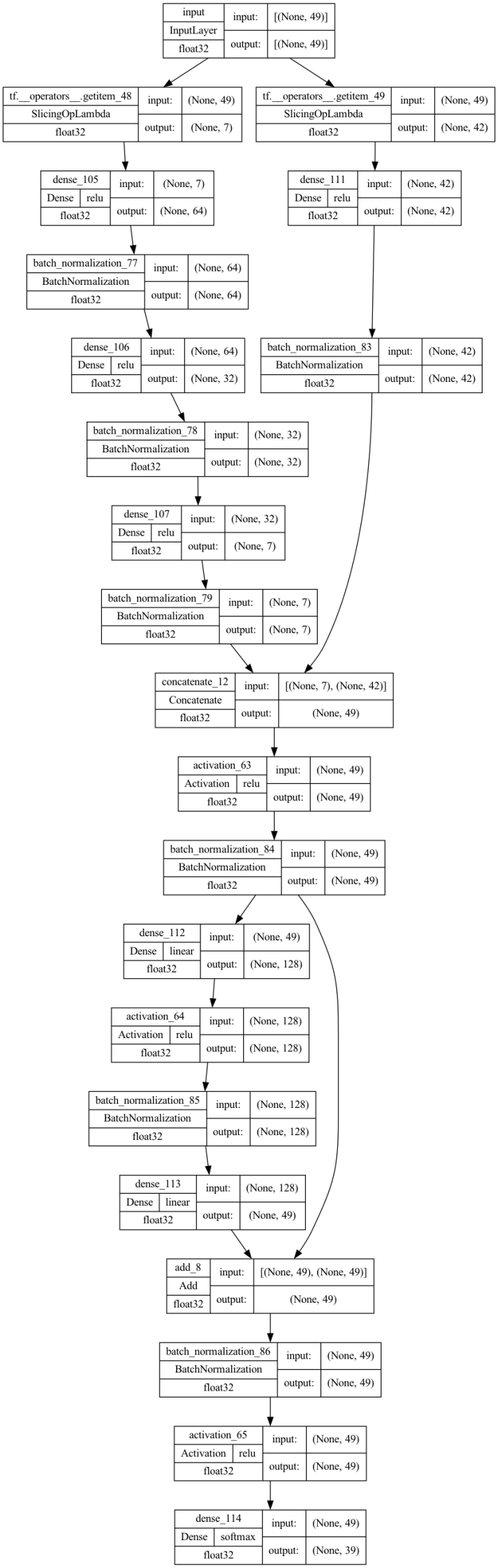
The result is the following plot of the history and the confusion matrix ():





Whit this model we achieved a grand total of:




input
InputLayer
float32
input: [(None, 49)]
output: [(None, 49)]
tf.__operators__.getitem_48
SlicingOpLambda
float32
input: (None, 49)
output: (None, 7)
tf.__operators__.getitem_49
SlicingOpLambda
float32
input: (None, 49)
output: (None, 42)
dense_105
Dense relu
float32
input: (None, 7)
output: (None, 64)
dense_111
Dense relu
float32
input: (None, 42)
output: (None, 42)
batch_normalization_77
BatchNormalization
float32
input: (None, 64)
output: (None, 64)
dense_106
Dense relu
float32
input: (None, 64)
output: (None, 32)
batch_normalization_83
BatchNormalization
float32
input: (None, 42)
output: (None, 42)
batch_normalization_78
BatchNormalization
float32
input: (None, 32)
output: (None, 32)
dense_107
Dense relu
float32
input: (None, 32)
output: (None, 7)
batch_normalization_79
BatchNormalization
float32
input: (None, 7)
output: (None, 7)
concatenate_12
Concatenate
float32
input: [(None, 7), (None, 42)]
output: (None, 49)
activation_63
Activation relu
float32
input: (None, 49)
output: (None, 49)
batch_normalization_84
BatchNormalization
float32
input: (None, 49)
output: (None, 49)
dense_112
Dense linear
float32
input: (None, 49)
output: (None, 128)
activation_64
Activation relu
float32
input: (None, 128)
output: (None, 128)
batch_normalization_85
BatchNormalization
float32
input: (None, 128)
output: (None, 128)
dense_113
Dense linear
float32
input: (None, 128)
output: (None, 49)
add_8
Add
float32
input: [(None, 49), (None, 49)]
output: (None, 49)
batch_normalization_86
BatchNormalization
float32
input: (None, 49)
output: (None, 49)
activation_65
Activation relu
float32
input: (None, 49)
output: (None, 49)
dense_114
Dense softmax
float32
input: (None, 49)
output: (None, 39)