SCS3253 Project Report - Street View House Numbers Image Recognition

- Pei Yao (Celine) Li and Kevin Liu

Table of Contents

1.0 Background

The purpose of this project is to employ different machine learning methods to extract house numbers from street view images containing house numbers. The dataset used for this project will be a publicly available dataset from the Stanford website; it is a set of real-world images of Street View House Numbers [1]. This project will focus on exploring the effectiveness of using support vector classification radial basis function (SVC RBF) , Logistic Regression, K-nearest neighbours (KNN) classification, Random Forest, Ensemble Stacking, and convolution neural networks (CNN), for number recognition in the images.

The images will be various types and colours of street numbers on various housing backgrounds. The entire dataset consists of "73257 images for training, 26032 images for testing. The format of the dataset is "MNIST-like 32-by-32 images centered around a single character (many of the images do contain some distractors at the sides) [1]." Although the quality and clarity of the images vary a bit; overall, the numbers on the images are largely recognizable to the human eye. Thus, it is expected that machine learning models can be trained on this dataset to yield a satisfactory model in the number recognition.

There are two separate workbooks created for this project, one for ensemble method named "Street-View-House-Numbers-Project- Ensemble Methods", and the second one is for CNN named "Street-View-House-Numbers-ProjectCNN Notebook". Both of these workbooks along with the training and testing datasets were submitted along with this report.

1.1 Preprocessing

To make fair comparisons between the model performances, the same preprocessing code was used at the beginning of both notebooks. The original data was split into training and testing with a ratio of 7.3:2.7. For each image in the original dataset, we converted it from red, green, and blue (RGB) to grayscale, using a weight of 0.3, 0.6 and 0.1 on the RGB value respectively. Then we scaled each pixel value in the image to be in the range of 0 to 1. The original pixel values are from 0 to 255, which may be harder for the neural network to deal with. We applied a simple linear scale by dividing by the max value of 255.

2.0 Methodology and Models Explanation

The sections below explain how ensemble methods and CNN were trained and what parameters were used, and their test performance in terms of accuracy.

2.1 Ensemble Methods

An ensemble method is the utilization of two or more different machine learning models together to create one model. This model typically aims to improve predictive accuracy or decrease variance/bias. It is essentially combining multiple good models to create a better model through the power of voting. This method will be beneficial to further improve the performance of our base models (SVC/Logistic Regression/KNN).

SVC (RBF): This is a classification model that employs supervised learning, and constructs hyperplane(s) to try to effectively separate the classes. Some benefits of using SVC to classify images are: they tend to perform fairly well on image datasets, have a decreased risk of over-fitting, and can scale upwards to data of increased dimensions [2]. The 'RBF' SVC outperforms the linear kernel, due to the data not being linearly separable.

Logistic Regression: Although primarily a binary classifier, the logistic regression model can be converted to perform multi-classification, by using 'one-vs-rest' regression for each class. The result is *n* logistic regression models corresponding to *n* total classes.

K-Nearest Neighbours: This classifier determines its predictions by finding k-nearest training points relative to each instance, and averages the nearest k points. There is only one hyperparameter for this method (# of neighbors) and explicitly uses past data to classify new data.
Random Forest: This is an ensemble method, and operates by training the data on multiple decision trees with a specified depth, returning the class that occurs most frequently. Overfitting is not as prevalent, and tend to garner a lower variance compared to a single decision tree. This model tends to perform well by simply tweaking the tree count hyperparameter, and is a good base model to begin with.
Ensemble Stacking: This is an ensemble method that aims to further increase model performance in terms of accuracy. It uses the prediction(s) of combined classifiers, and employs another classifier to predict based on the results of the combined classifiers. We will use this to boost the accuracy of our base model(s) after fine-tuning their hyperparameters.

2.11 Code Explanation:

The notebook "Street-View-House-Numbers-Project- Ensemble Methods" is split up into various sections. The first section applies pre-processing as discussed in report section 1.1, with the addition of undersampling. This is due to the training dataset being imbalanced, as a majority of samples belong to classes 1, 2, and 3. As a result, the classes were undersampled to match the number of samples in class 10, achieving an even distribution.

Section two of the notebook consists of grid searches and random searches for optimal parameters, specifically for the random forest algorithm and SVC using RBF kernel. Predictions on the test data are then made for each of the models. A stacking ensemble model using SVC and random forest (voting classifier) was included in the list of tested classifiers. The hyperparameters that the search returned for random forest achieved a test accuracy of 64%. Ultimately, these hyperparameters were not used in favour of simply increasing the number of estimators, and keeping the max depth and minimum samples split at the default value. Ideally, we would search for more hyperparameters. However, runtimes increased significantly as more hyperparameters were included in the search. The grid search for appropriate gamma and C values for the SVC gave a test accuracy of approximately 68% when gamma = 0.00001 and C = 10. However, we found that we could further increase test accuracy by having an even smaller gamma of 0.000001. Lowering the gamma value more than this value resulted in diminishing returns and a decrease in test accuracy. Refer to Table 2-11 below for the hyperparameters searched for the corresponding models, as well as the associated code. Hyperparameter searching was not done for KNN and logistic regression due to significant runtimes, although principal component analysis was applied to reduce dimensionality. We also output a confusion matrix for our predictions to see the details of the misclassified digits.

The third section in the notebook is where we aim to improve prediction accuracy on the test data by applying a stacking ensemble method using both the SVC and random forest classifiers. We utilize a voting classifier to determine the final prediction.

Table 2-11. List of hyperparameters used for each model

Model	Hyperparameters Searched	Final Hyperparameters
Random Forest	N_estimators = [500, 700, 900], max_features = "auto", max_depth = [1,5,10], min_samples_split = [3, 5], min_samples_leaf = [1, 2, 4], bootstrap = [True, False]	N_estimators = 1500, max_features = "auto", min_samples_leaf = 1
Random Forest w/ Rand Search		N_estimarors = 900, max_features = "auto", min_samples_leaf = 1, max_depth = 10, min_samples_split = 5
K-Nearest Neighbors	None	N_neighbors = 2
SVC RBF	Gamma = [0.00001, 0.0001, 0.1], C = [1, 10]	Gamma = 0.000001, C = 10
Logistic Regression	None	Multi_class = "ovr", solver = "lbfgs"
Stacking SVC with Random Forest	None	Voting = "hard"

2.12 Model Performance Comparison:

Table 2-12 . Comparison of Test Accuracy for Various Models

Model	Test Accuracy	Comments for future recommendation
Random Forest	71%	Highest accuracy out of tested classifiers, slightly better than SVC
Random Forest w/ Rand Search	64%	Likely need to add more parameters
K-Nearest Neighbors	46%	May have overlap between classes, also a bit noisy due to images including overlapping digits
SVC RBF	71%	Try testing different C-values
Logistic Regression	17%	Try an increasing number of features (image size)
Stacking SVC with Random Forest	71%	Could improve performance by adding another decent-performing model, with weight(s)

We expected SVC RBF to perform pretty well in comparison to the other base models, as a radial basis function with a very small gamma value tends to draw lower variance, with the tradeoff of higher bias. The SVC can separate the classes with hyperplanes, and depending on the choice of C value, is able to eliminate samples that fall too close to the margin between two classes [3]. The biggest challenge with this dataset is the presence of multiple images where there is more than one digit, many of which are cut-off parts of different numbers. This is more difficult to train due to many training samples containing other image's pixels, compared to a simpler dataset such as MNIST. This might be the reason why logistic regression did not perform well on this dataset. One possible solution would be to include the extra set of images to increase the number of samples available for training, and to be able to oversample the training data rather than undersample.

We expected random forest and ensemble stacking to perform better than the rest. However, it was a bit surprising that they all have the same test accuracy of 71%, same as the SVC RBF.

2.13 Future Improvements on Ensemble Stacking:

Due to long runtimes, performing grid search / random search on all models was not feasible. Thus, there is still room for improvement in various models, but we did not achieve those due to limited computation capacity of home laptops. The optimal scenario would be to test a multitude of different hyperparameters for all models to achieve the best performances for them. Inclusion of the extra set provided, combined with the current training set likely would have increased accuracy for the models as well. Accuracy could be potentially increased even further if we performed stacking SVC w/ random forest, and convolutional neural network prediction on top of the ensemble.

2.2 Convolutional Neural Network:

CNN method is a deep learning algorithm that can take an input image and assign importance to various aspects in the image in order to differentiate one from another [4]. The benefit of the CNN method is that is efficient in recognizing patterns and key features in images as it is not a fully connected neural network, but still contains multiple convolution layers (the kernels) which are used to "filter" the images. Because it is not a fully connected neural network, but small filters, the location of the number on the image does not impact the testing performance. These are reasons why CNN is a standard algorithm used for image recognition, and it is why we decide to use it in our project. The challenges with applying the CNN method are potential overfitting, and also it requires a large amount of training data to perform well.

2.21 CNN Model Code Explanation:

The first part of the "Street-View-House-Numbers-ProjectCNN Notebook" is data exploration and pre-processing of the data. The same preprocessing as ensemble methods was used in order to standardize the comparison.

The second part of the code defines the architectural layout and the specific parameters for each layer. At the core of the model are 3 convolutional layers, each followed by a Max-Pooling and Leaky ReLu activation. The filters in the convolutional layers are the core feature extractors for our model, while the max pooling layers reduce feature size and help prevent overfitting. For the activation function, Leaky ReLu has recently become more popular than ReLu since it can improve the accuracy of some architectures.

Following the convolutional layers is a fully connected dense layer, allowing information extracted by the convolutional layers to be used all together. At the end of the network, there is the Softmax layer that outputs the probability for each class.

We used the Adam Optimizer since it includes many optimizations compared to the original Momentum optimizer, and is a standard optimizer that generally performs well. For the loss function, since our class labels are discrete (1,2,3...) instead of being embedded through other means (embedding or one-hot code), we used the sparse categorical cross-entropy [5].

2.22 Model Performance vs Parameters Comparison:

Table 2-22. Performance of CNN Model with Various Parameters

Epochs	**Depths and Activation of each CNN layer**	**Dense Layer # Neurons**	**Test Accuracy**	**Result & Comments**
5	(32, 64, 64; ReLu)	64	90.32%	Good initial performance
5	(32, 64, 64; Leaky ReLu α=0.1)	64	90.73%	Leaky ReLu performs slightly better
5	(64, 128, 128; Leaky ReLu α=0.1)	64	91.03%	Increased depths performs better
5	(64, 128, 128; ReLu)	64	89.67%	Leaky ReLu performs slightly better
5	(64, 128, 128; Leaky ReLu α=0.1)	128	90.81%	Overfit from increased dense layer neuron
5	(32, 64, 64; Leaky ReLu α=0.1)	128	90.83%	Still overfit from increased dense layer neuron, will stick to 64
10	(32, 64, 64; Leaky ReLu α=0.1)	64	90.68%	Epoch =10 lead to overfitting
10	(64, 128, 128; Leaky ReLu α=0.1)	64	90.97%	Epoch =10 lead to overfitting
5	(64, 128, 128; Leaky ReLu α=0.2)	64	91.01%	No significant difference than α=0.1
5	(32, 64, 64, 64; Leaky ReLu α=0.1)	64	90.07%	4 layers led to overfitting, so will stick with 3 layers

In order to achieve the best performance, various parameters were tried out and their accuracies are recorded in Table 2-22. The three highlighted rows were the top three performance without employing dropout. The initial trial run model consisted of 3 CNN layers at depths of 32, 64, and 64 respectively with an activation function of ReLu. From the table above, we can see that using the Leaky ReLu activation function in the CNN layers indeed performs better than ReLu. The α value of the Leaky ReLu function is to be the standard 0.1 as we did not find any significant difference between performance and α value. However, the activation function of the dense layer is ReLu as we found it performs better than Leaky ReLu. Therefore, it is not conclusive that Leaky ReLu always performs better than ReLu.

From the table above, we can also see that when the depths of the 3 layers was increased to 64, 128, 128 respectively, the model generally performed better, which is not surprising. However, when we increased the number of layers from 3 to 4, the model overfitted and performed worse; thus, the final model parameter was kept at 3 layers with depths of 64, 128, 128.

Finally, we found that although an increasing number of epochs increased training accuracy significantly, the testing accuracy was generally the same or slightly worse than fewer epochs. This is due to overfitting behaviour as shown and explained in Appendix A. Thus, in order to mitigate the effect of overfitting, a grid search was performed over different drop out values. With the addition of dropout parameter, it is clear that the model stopped overfitting as much. The comparison of model accuracy with various dropout values is shown in Appendix B. It is determined that dropout=0.4 and epoch =10 performs the best; thus, these are the final parameters employed the final CNN model.

2.23 Future Improvements on CNN:

Due to limitations in computational power, not a lot of changes in parameter was explored when tuning the CNN model. If computational power allows, perform more grid searches on the number of epochs as well as the number of neurons for each layer to see if test performance can be further improved.

3.0 Final Conclusion:

The CNN model with 92% test accuracy on the Street View House Numbers Image dataset performs significantly better than Ensemble Method with 71% test accuracy. Thus, the final model proposed for this project is the CNN model. Although the challenge of CNN model overfitting has impacted its performance, with the use of dropout parameter, we have mitigated overfitting and produced a satisfactory model.

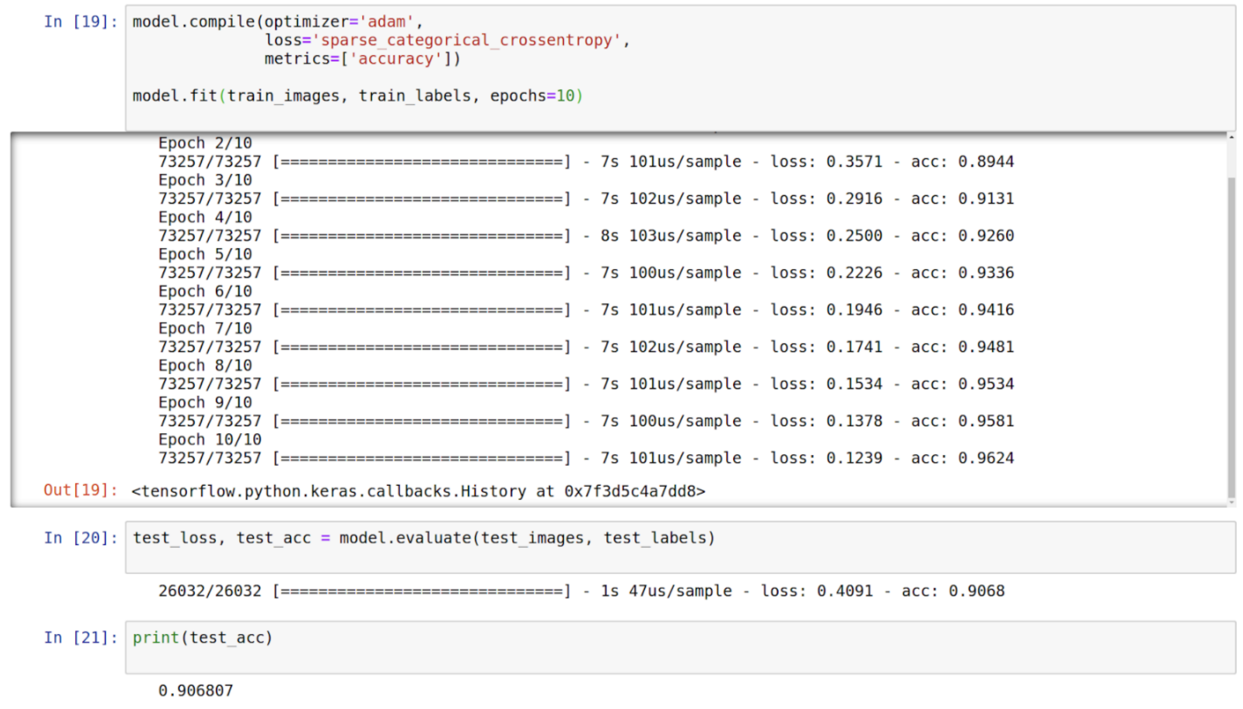
4.0 References:

[1] Y. Netzer, T.Wang, A.Coates, A.Bissacco, B.Wu, A.Y. Ng "Reading Digits in Natural Images with Unsupervised Feature Learning" *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*. (PDF) [Online]. Available: http://ufldl.stanford.edu/housenumbers
[2] A. Yadav, Towards Data Science. *Support Vector Machines (SVC)*. [Online]. Available: https://towardsdatascience.com/support-vector-machines-SVC-c9ef22815589
[3] Stack Overflow, *Support Vector Machine : What are C & Gamma?*, [Online]. Available: https://stackoverflow.com/questions/35848210/support-vector-machine-what-are-c-gamma?fbclid=IwAR0L0jxLr94rCwYwBsjKPwSwJCGtiSDPhgS5AJoncYFhCpzPjqFY36esVVA
[4] S. Saha, Towards Data Science. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
[5] Tensorflow API Doc, *Sparse Categorical Cross Entropy*. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/backend/sparse_categorical_crossentropy

5.0 Appendices

Appendix A: Overfitting Behaviour with Too Many Epochs

This screenshot of the jupyter notebook output below shows the accuracy of each of the 10 epochs and the final test accuracy. Notice that the final test accuracy is only 0.9068 which is quite lower than the latter epochs in the training reaching above 0.95; which means the training dataset was starting to overfit.



```
In [19]: model.compile(optimizer='adam',
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])

         model.fit(train_images, train_labels, epochs=10)

            Epoch 2/10
            73257/73257 [==============================] - 7s 101us/sample - loss: 0.3571 - acc: 0.8944
            Epoch 3/10
            73257/73257 [==============================] - 7s 102us/sample - loss: 0.2916 - acc: 0.9131
            Epoch 4/10
            73257/73257 [==============================] - 8s 103us/sample - loss: 0.2500 - acc: 0.9260
            Epoch 5/10
            73257/73257 [==============================] - 7s 100us/sample - loss: 0.2226 - acc: 0.9336
            Epoch 6/10
            73257/73257 [==============================] - 7s 101us/sample - loss: 0.1946 - acc: 0.9416
            Epoch 7/10
            73257/73257 [==============================] - 7s 102us/sample - loss: 0.1741 - acc: 0.9481
            Epoch 8/10
            73257/73257 [==============================] - 7s 101us/sample - loss: 0.1534 - acc: 0.9534
            Epoch 9/10
            73257/73257 [==============================] - 7s 100us/sample - loss: 0.1378 - acc: 0.9581
            Epoch 10/10
            73257/73257 [==============================] - 7s 101us/sample - loss: 0.1239 - acc: 0.9624

Out[19]: <tensorflow.python.keras.callbacks.History at 0x7f3d5c4a7dd8>

In [20]: test_loss, test_acc = model.evaluate(test_images, test_labels)

            26032/26032 [==============================] - 1s 47us/sample - loss: 0.4091 - acc: 0.9068

In [21]: print(test_acc)

            0.906807
```

Appendix B: Addition of Dropout Reduced Overfitting

The table below shows the performance comparison between the same CNN model except with different dropout for epoch =5 and 10. The model uses 3 CNN layers at depths of 64, 128, 128 respectively and leaky ReLu of α=0.1 for activation type, as well as dense layer neuron number of 64.

Dropout rate	**Epoch = 5**		**Epoch = 10**	
	Training Accuracy	**Testing Accuracy**	Training Accuracy	**Testing Accuracy**
None	94.57%	91.03%	96.24%	90.97%
0.2	92.25%	90.41%	94.75%	91.03%
0.3	91.81%	91.16%	94.15%	91.70%
0.4	91.12%	91.25%	93.60%	**91.88%**
0.5	90.36%	90.77%	92.96%	91.63%

It is clear that the dropout of 0.4 and 10 epochs performs best. Its accuracy of 91.88% exceeds the performs of the same model but without any dropout with an accuracy of 90.97%.