SVHN Sequence Detector

Machine Learning Nanodegree Capstone Project

Summer 2016

—

Sergio Gordillo
@Sergio_Gordillo
github.com/archelogos/sequence-detector

Definition

Project Overview

Deep Learning is currently one of the most interesting fields in Machine Learning. The combination of Neural Networks and powerful computation systems is extremely useful to solve complicated Pattern Recognition or Text Classification problems.

The main goal in this project is, thanks to Deep Learning techniques, be able to detect and identify sequences of digits in a random picture. Specifically in this project, the images correspond to houses and the sequences correspond to their house numbers.

In order to simplify the explanation, the desired result is shown in the following picture.

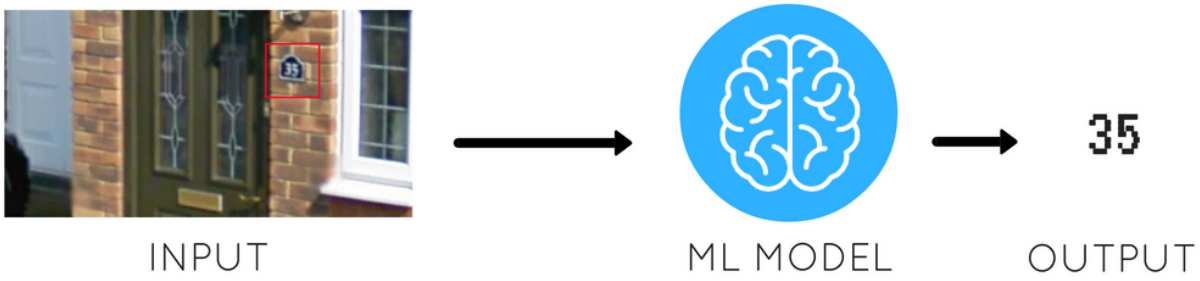

Image 1: Project goal overview

The data used to train our model is part of the SVHN dataset, which is a real-world image dataset for developing machine learning and object recognition and is obtained from house numbers in Google Street View images.

This is a real-world problem studied for many years and it wasn't until the application of neural networks to the images recognition problems when it could be considered as solved. At this moment it can be seen real products and apps that are focused on solve this problem (i.e. http://questvisual.com/).

Problem Statement

As it was said, the main goal is to define, build and train a stable and consistent ML Model which can identify sequences of house numbers in a particular image.

Image 2: Samples of images from the dataset

To reach this goal, it was used the mentioned dataset and it was created a model based on a Neural Network. The final model has been built taking different ideas from investigation papers, tutorials and examples.

The model begins being as basic as it's possible based on a simple Logistic Regression. Along the project, the model is getting more complicated and sophisticated step by step, becoming a Multilayer ConvNet production-ready model at the end of the project.

In the process, different techniques, algorithms and solutions are tried and properly discussed.

The project is divided in 7 steps:

1. Familiarize with TensorFlow, applying from a Logistic Regression to a Multilayer ConvNet model to a **single-digit-MNIST** dataset.
2. Modify the previous model and apply it to a **single-digit-SVHN** dataset.
3. Improve the model and apply it to a **sequence-MNIST** dataset.
4. Modify the previous model and apply it to a **sequence-SVHN** dataset.
5. Define a stable and consistent model.
6. Train the final model in Google Cloud Platform in order to boost its performance.
7. Build and deploy a web app which demonstrates how the model works.

The human ability to correctly identify the numbers from the SVHN dataset is approximately 98% and the most important solutions to this problem given by companies like Google have almost reached that goal.

Convolutional Neural Networks aren't extremely efficient and they are not easy to train properly because they are expensive in terms of computation resources. For that reason the first steps of the project are developed in ipython notebooks, running small training per each session. Once it is clear that the model works fine, it was moved to the Public Cloud (GCP) to train it in a better way (highcpu machines running efficient py scripts).

Anyway it's important to notice that the purpose of this project is not to improve the current best models built for this problem but it's to study and build an able and production-ready model which can perfectly obtain a 96% of accuracy identifying sequences from pictures (which in the field of Neural Networks is a huge difference with the mentioned 98%).

Metrics

As it was said, the accuracy is the main metric defined for this problem.

It can be defined as the probability of coincidence between the real number in the picture and the number predicted from the model.

A prediction it's considered correct if all the digits in the sequence and the length of the sequence are exactly the same, in other cases the prediction will be considered not correct.

To check that the model is working properly, and the solution fits the problem, a group of periodic checks were set on the **training** and **validation** dataset along the training.

At the end of each training session it's always estimate the current accuracy of the model on the entire **test** dataset.

```
## Pseudo python
def accuracy(labels, predictions):
    n_accuracy = summation(predictions == labels) / length(labels)
    p_accuracy = 100 * n_accuracy
    return p_accuracy
```

Image 3: Pseudo code of accuracy function used

Analysis

Data Exploration

As it was explained, the project itself and the main objective of it is focused on the SVHN dataset. However, there are other interesting dataset that can be used at the first steps of the project in order to be more practical studying the model and delaying until the end of the project the preprocessing functionality.

One of these dataset is the MNIST one and it consists of 70.000 examples of **handwritten** digits. The size of the representation of each number is 28x28 pixels, and they are normalized. Each digit has 784 features, (28 by 28) and 1 channel-depth corresponding to the grayscale (28x28x1).

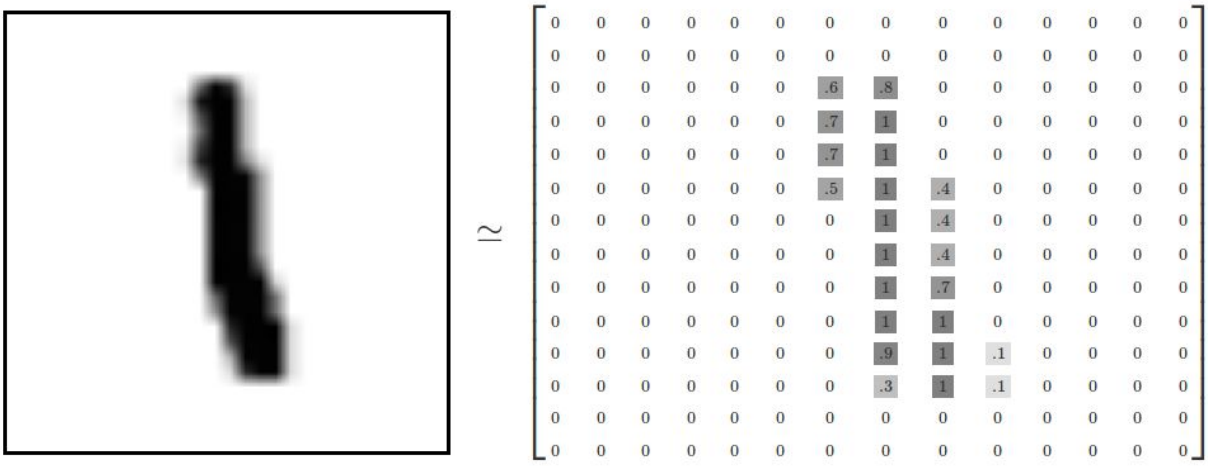

Image 4. MNIST Digit representation

This data is downloaded from the Yann LeCun Website and after that, the files are properly extracted and reformatted creating the different dataset (training, validation and test).

The labels are not downloaded in One-Hot encoding so the label of each digit is the value of the own number and it's not other kind of representation.

Once the model is working for the single-digit-MNIST dataset, the next step is apply it to the SVHN dataset. This dataset has two different formats: the first one is compound by all the real images from Google Street View and the metadata that contains the bounding boxes and the labels for each digit in the picture. This format requires that the images are preprocessed. The details of how they were preprocessed will be explained and

discussed later because for this step it was used the second format from the SVHN dataset.

The second format provides preprocessed images with a size of 32 by 32 pixels. The labels don't correspond with the labels of the entire sequence but they represent the central digit on the image. The data is given in a 4D Tensor so the previous build model can be applied just converting the images to grayscale and modifying the variables of our model that are related with the image size (from 28x28 to 32x32).

Technically the model works at the same way in both cases, using MNIST or SVHN data.

The next logical step was to transform the MNIST single digits into sequences, and these were just built concatenating randomly digits and creating sequences of variable length. After that, the new image was reformatted to a size of 28x28. Obviously, the model had to change to be able to detect and identify more than one digit.

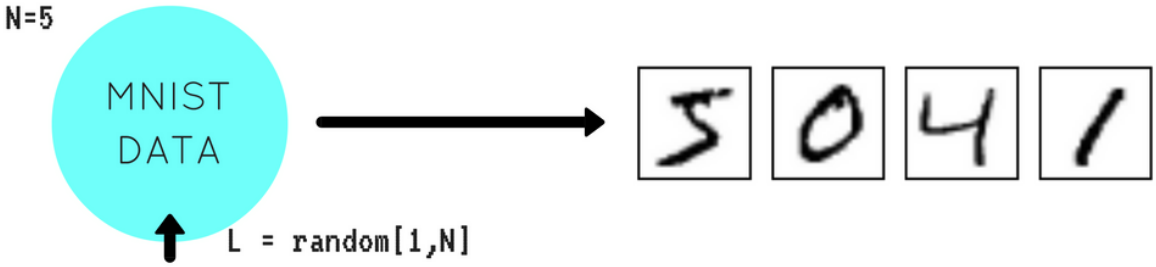

Image 5: Creating MNIST Sequences

Finally, once the model was built and tested in the MNIST sequences, it was necessary to think in how to preprocess the SVHN images. Thanks to some Python libraries this step wasn't so hard as it could be. Each file was downloaded and extracted and and using the available metadata it was possible to open each image and crop each digit from it. Knowing perfectly how many digits were in each image and the bounding boxes of each of them, the new images were created concatenating the digits according to the length of the sequence and properly resized to 32 by 32 pixels.

Labels were also extracted and all the data was converted in Tensors. After that it was properly saved in a pickle file which gets easier to access the data.

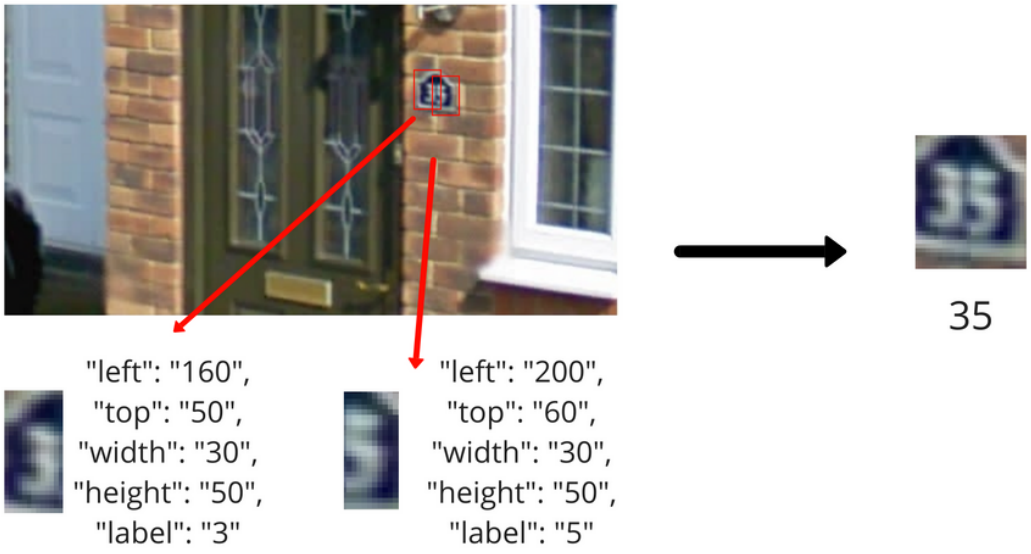

Image 6: SVHN Preprocessing

Exploratory Visualization

The most important point to cover in the data visualization was to be secure and confident about any step done in the preprocessing functions.

For the MNIST dataset wasn't necessary any kind of preprocessing as it was mentioned, but in the case of SVHN, with the objective of simplify the final model without losing information, the images were transformed into grayscale. Furthermore it was applied a Global Contrast Normalization to get better shapes of the numbers.

It was important to notice that all the different transformations didn't alter the original information.

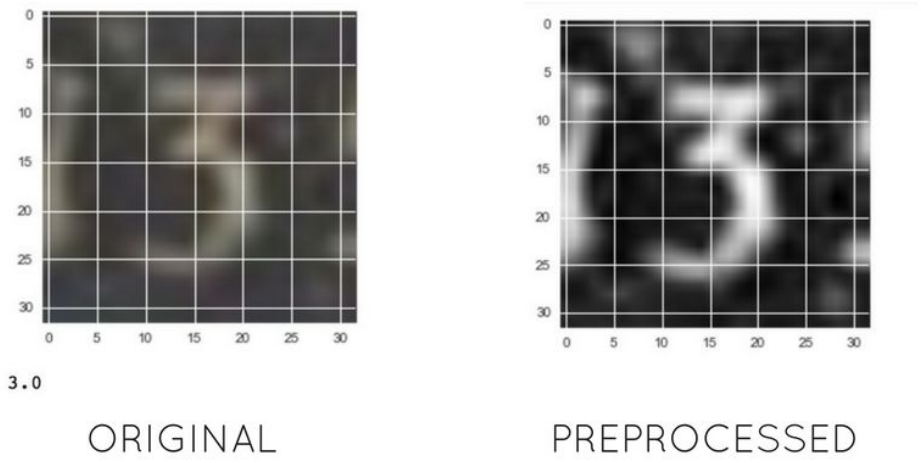

Image 7: Preprocessing SVHN single digits

In the case of the MNIST **sequences**, the images were resized and the digits were a little bit distorted, but it didn't alter strength of the CNN model and the digits were properly identified. This fact was analyzed in terms of information loss but, as it was just said, the model worked very fine detecting the shapes of the numbers in any case. Besides that, it's important to point out that as soon as the digits were horizontally concatenated, the proportion of the image was lost in one dimension.

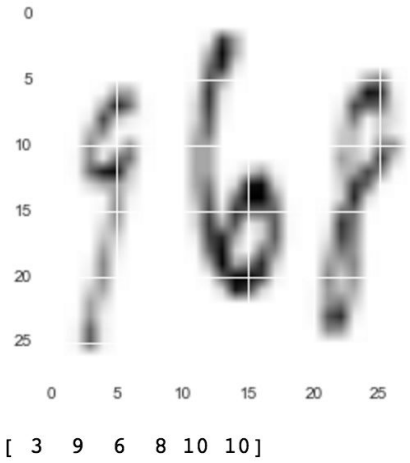

Image 8: Artificial MNIST sequences

Finally for the SVHN images, the required preprocessing was a bit harder than the previous one.

Once the files was downloaded and extracted, the metadata was obtained from the .mat file and each digit on each image were preprocessed (taking the bounding box from the metadata) and concatenated with the numbers of the same sequence.

In a parallel process the labels were correctly extracted and processed, forming at the end a 4D Tensor for the images and a 2D Tensor for the labels.

This process will be explained in detail in the next section.

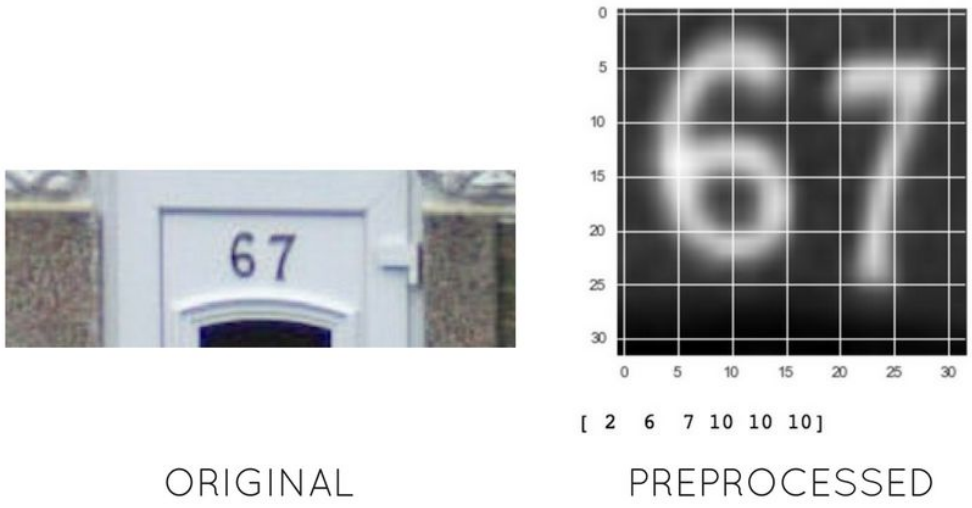

Image 9: Preprocessing SVHN images

Algorithms and Techniques

A the beginning it was built a single-layer softmax regression model in order to create the foundations. Later, the model was extended to the case of softmax regression with a multilayer convolutional network.

Focusing on the SVHN-sequences, that is the main objective of the project, the result of the preprocessing step was 2 Tensors per each set as it was pointed out.

Training set: (160755, 32, 32, 1) (160755, 6)
Test set: (58068, 32, 32, 1) (58068, 6)
Validation set: (30000, 32, 32, 1) (30000, 6)

Analyzing the Training set, the first Tensor (4D) represents the data converted from the images:

- 160755 samples
- Width: 32 pixel
- Height: 32 pixel
- Depth: 1 channel (grayscale)

The second Tensor (2D) contains the information of the labels:

- 160755 labels, matching with the number of samples
- 6 labels (5+1) **N=5 is the maximum number of digits in a sequence**. The first label of each sequence is always the length of it. The absence of a digit it's represented by a **10**.

An example of this is shown in the Image 9:

- Real number: 67
- Labels: [2, 6, 7, 10, 10, 10] (length of the array 6, first element 2 corresponding with the length of the sequence, the digits, and 3 numbers 10 filling the array. According to this, the number of different classes is 11 [0-10]

The same explanation applied to the validation and test set.

The final architecture consists of **three convolutional locally connected hidden layers**. The connections of the convolutions layers are feedforward and go from one layer to the next. The number of units at each spatial location in each layer is [16, 32, 64] respectively. All convolution kernels are of size 5 × 5.

Each convolutional layer uses a stride of one and includes max pooling and subtractive normalization. The max pooling window size is 2 × 2. The second convolution uses zero padding on the input to preserve representation size, the first and third convolution use VALID padding in order to reduce the output dimensionality.

The result of the convolutions (the feature vector **H**) is **fully connected to a N+1 Linear Regressions**. Each regression extracts the features corresponding to each digit in the sequence. The regression has to compute **256** values according to the size of the feature vector and of course the size of each Regression output matches with the number of classes for each digit [0-10] (**11**)

This model could be easily improved adding a fourth convolutional layer or increasing the number of units at each spatial location in each layer to looking for a better feature extraction. However this modifications can increase drastically the computation cost of the training.

After that, the logits $z_{s_i} = w_{s_i}H + b_{s_i}$ are computed applying the **Softmax** algorithm and obtaining the probabilities that are assigned to each digit.

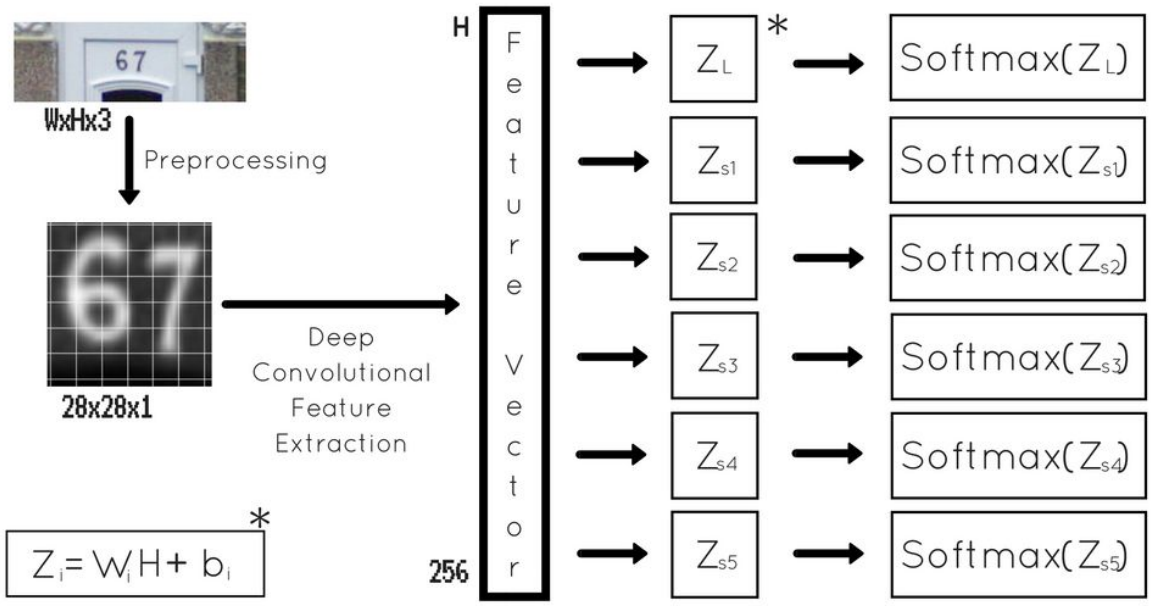

Image 10: Model Overview

The result of this softmax function can be considered the prediction of our model. It could be defined as the probability of be a particular value. These logits are computed with the labels applying the **Cross Entropy Function**.

Each digit has its own label and, in consequence, its own loss function, the total loss of the entire model is just the arithmetic summation of all individual losses.

In order to train the model, it was applied the **AdagradOptimizer** with an initial value of 0.01 for the learning rate.

Summing up the hyperparameters initial setup is:

IMAGE_SIZE = **32** (32 by 32 pixels each image)

NUM_CHANNELS = **1** (grayscale images, it would be 3 if the images were in color)

NUM_LABELS = **11** (0-10 both included)

N = **5** (maximum number of digits in a sequence)

BATCH_SIZE = **64** (number of samples of each batch on the training step)

PATCH_SIZE = **5** (5 x 5 convolution kernels)

DEPTH_1 = **16** (the first convolution compute 16 features for each 5x5 patch)

DEPTH_2 = **32** (the second convolution compute 32 features for each 5x5 patch)

DEPTH_3 = **64** (the third convolution compute 64 features for each 5x5 patch)

REGRESSION = **256** (the Regression compute the 256 features of the Feature Vector)

Benchmark

In this section, you will need to provide a clearly defined benchmark result or threshold for comparing across performances obtained by your solution. The reasoning behind the benchmark (in the case where it is not an established result) should be discussed. Questions to ask yourself when writing this section:

- *Has some result or value been provided that acts as a benchmark for measuring performance?*
- *Is it clear how this result or value was obtained (whether by data or by hypothesis)?*

Methodology

Data Preprocessing

In this section, all of your preprocessing steps will need to be clearly documented, if any were necessary. From the previous section, any of the abnormalities or characteristics that you identified about the dataset will be addressed and corrected here. Questions to ask yourself when writing this section:

- *If the algorithms chosen require preprocessing steps like feature selection or feature transformations, have they been properly documented?*
- *Based on the Data Exploration section, if there were abnormalities or characteristics that needed to be addressed, have they been properly corrected?*
- *If no preprocessing is needed, has it been made clear why?*

Implementation

In this section, the process for which metrics, algorithms, and techniques that you implemented for the given data will need to be clearly documented. It should be abundantly clear how the implementation was carried out, and discussion should be made regarding any complications that occurred during this process. Questions to ask yourself when writing this section:

- *Is it made clear how the algorithms and techniques were implemented with the given datasets or input data?*
- *Were there any complications with the original metrics or techniques that required changing prior to acquiring a solution?*
- *Was there any part of the coding process (e.g., writing complicated functions) that should be documented?*

Refinement

In this section, you will need to discuss the process of improvement you made upon the algorithms and techniques you used in your implementation. For example, adjusting parameters for certain models to acquire improved solutions would fall under the refinement category. Your initial and final solutions should be reported, as well as any significant intermediate results as necessary. Questions to ask yourself when writing this section:

- *Has an initial solution been found and clearly reported?*
- *Is the process of improvement clearly documented, such as what techniques were used?*
- *Are intermediate and final solutions clearly reported as the process is improved?*

Results

Model Evaluation and Validation

In this section, the final model and any supporting qualities should be evaluated in detail. It should be clear how the final model was derived and why this model was chosen. In addition, some type of analysis should be used to validate the robustness of this model and its solution, such as manipulating the input data or environment to see how the model's solution is affected (this is called sensitivity analysis). Questions to ask yourself when writing this section:

- *Is the final model reasonable and aligning with solution expectations? Are the final parameters of the model appropriate?*
- *Has the final model been tested with various inputs to evaluate whether the model generalizes well to unseen data?*
- *Is the model robust enough for the problem? Do small perturbations (changes) in training data or the input space greatly affect the results?*
- *Can results found from the model be trusted?*

Justification

In this section, your model's final solution and its results should be compared to the benchmark you established earlier in the project using some type of statistical analysis. You should also justify whether these results and the solution are significant enough to have solved the problem posed in the project. Questions to ask yourself when writing this section:

- *Are the final results found stronger than the benchmark result reported earlier?*
- *Have you thoroughly analyzed and discussed the final solution?*
- *Is the final solution significant enough to have solved the problem?*

Conclusion

Free-Form Visualization

In this section, you will need to provide some form of visualization that emphasizes an important quality about the project. It is much more free-form, but should reasonably support a significant result or characteristic about the problem that you want to discuss. Questions to ask yourself when writing this section:

- *Have you visualized a relevant or important quality about the problem, dataset, input data, or results?*
- *Is the visualization thoroughly analyzed and discussed?*
- *If a plot is provided, are the axes, title, and datum clearly defined?*

Reflection

In this section, you will summarize the entire end-to-end problem solution and discuss one or two particular aspects of the project you found interesting or difficult. You are expected to reflect on the project as a whole to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section:

- *Have you thoroughly summarized the entire process you used for this project?*
- *Were there any interesting aspects of the project?*
- *Were there any difficult aspects of the project?*
- *Does the final model and solution fit your expectations for the problem, and should it be used in a general setting to solve these types of problems?*

Improvement

In this section, you will need to provide discussion as to how one aspect of the implementation you designed could be improved. As an example, consider ways your implementation can be made more general, and what would need to be modified. You do not need to make this improvement, but the potential solutions resulting from these changes are considered and compared/contrasted to your current solution. Questions to ask yourself when writing this section:

- *Are there further improvements that could be made on the algorithms or techniques you used in this project?*
- *Were there algorithms or techniques you researched that you did not know how to implement, but would consider using if you knew how?*
- *If you used your final solution as the new benchmark, do you think an even better solution exists?*

Before submitting your report, ask yourself…

Does the project report you've written follow a well-organized structure similar to that of the project template?

Is each section (particularly Analysis and Methodology) written in a clear, concise and specific fashion? Are there any ambiguous terms or phrases that need clarification?

Would the intended audience of your project be able to understand your analysis, methods, and results?

Have you properly proof-read your project report to assure there are minimal grammatical and spelling mistakes?

Are all the resources used for this project correctly cited and referenced?

Is the code that implements your solution easily readable and properly commented?

Does the code execute without error and produce results similar to those reported?