Project Report
Team Genome

Introduction.

Problem Being Addressed

Deep learning convolutional neural networks require extensive manual trial and error with expensive resources to find an optimal parameter set. 40% of companies take over 30 days for this process.

Proposed Solution

Implement a genetic algorithm to quickly find optimal parameters & hyperparameters for deep learning convolutional neural networks.
Our team implemented a genetic algorithm and specific techniques for different stages such as cross over, mutation, fitness function, & more and integrated this with Deep Learning to produce optimal CNN models for a given dataset such that the new architecture and parameters (example - filters, activation, optimizer, pool size etc.) will result in optimal accuracy and loss.

Progress Status.

Week 1 - 2: Genetic Algorithm
Week 3 - 4: Deep Learning
Week 5 - 6: Integration of GA with DL
Week 7 - 9: Increasing complexity of Algorithm
Week 10 - 11: Testing on Vermeer Data & Improving Algorithm

Implemented Algorithm.

Overview

The algorithm consists of using a genetic algorithm to optimize the variables of a Deep Learning CNN Model. The genetic algorithm has the following variables: Population Size, Population Retained, and Number of Generations.
Population indicates the number of chromosomes. Each chromosome consists of a gene list. The gene list consists of all the variables with which a Deep Learning CNN model can be made. The Population Retained indicates the percentage of chromosomes that survive each generation (which means that *(1 - Population Retained) * Population Size* chromosomes will need to be reproduced each generation). The Generations indicates the number of generations we run the process of the genetic algorithm.
Since each chromosome represents a Deep Learning CNN model, this means its fitness represents performance of the CNN model. For a given chromosome, we decided to use the validation loss of the CNN it represents as its fitness value.

Chromosomes' Gene List

As mentioned above, the chromosome represents a CNN model, and each chromosome contains a gene list with all the variables detailing the architecture, parameters / hyper parameters of the CNN model it represents.

The Gene List of every chromosome is of the following format: [Number of Pairs, PairList, DenseLayer, Batch Size, Optimizer, Activation].

A Pair is what consists of X number of Convolutional 2D layers followed by 1 Max Pooling Layer.

- Number of Pairs - N number of Pairs in the CNN model.
- PairList - This is a list storing all the Pairs in the Model. It is of the following format - [Pair1, Pair2, Pair3 …..]
  Each Pair has the following format - [Conv2DFilterSize, Conv2DFilterSize ….. MaxPoolingSize]. Thus, each pair can have X number of Convolutional layers (storing the filter size). Every pair always ends with a Max Pooling Layer (containing the pooling size)
  The filter size of each convolutional layer is a number between 16 - 256.
  The Pooling Size is a number between 2-8.
- DenseLayer - The number of nodes in the Dense Layer before the final Output Dense Layer.
- BatchSize - This is a number between 0-3 representing an index. It will be one of the following based on the index : [32,64,128,256]
- Optimizer - This is a number between 0-3 representing an index. It will be one of the following tensorflow optimizers based on the index :
  ['Adam','Nadam','Adadelta','Adamax']
- Activation- This is a number between 0-3 representing an index. It will be one of the following tensorflow activation functions based on the index : ['relu','tanh','selu','elu']

```
Num of Conv2d Sets || ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
'0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '1', '0']2
ConvSet1-ConvLayer-0 || ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
'0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '1']1
ConvSet1-ConvLayer-1 || ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
'0', '0', '0', '0', '0', '0', '0', '0', '1', '1', '1', '0', '1', '0']58
ConvSet1-MaxPoolingLayer || ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
'0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '1', '0']2
ConvSet2-ConvLayer-0 || ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
'0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '1']1
ConvSet2-ConvLayer-1 || ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
'0', '0', '0', '0', '0', '0', '0', '1', '0', '1', '1', '1', '1']47
ConvSet2-MaxPoolingLayer || ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
'0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '1', '0']2
Dense Layer 1 || ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
'0', '0', '0', '0', '0', '0', '0', '1', '0', '0', '0', '1', '0']34
Batch Size || ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
'0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0']0
Optimizer || ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
'0', '0', '0', '0', '0', '0', '0', '0', '0', '1', '1']3
Activation || ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
'0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '1']1
```

Figure: Example of a chromosome gene list

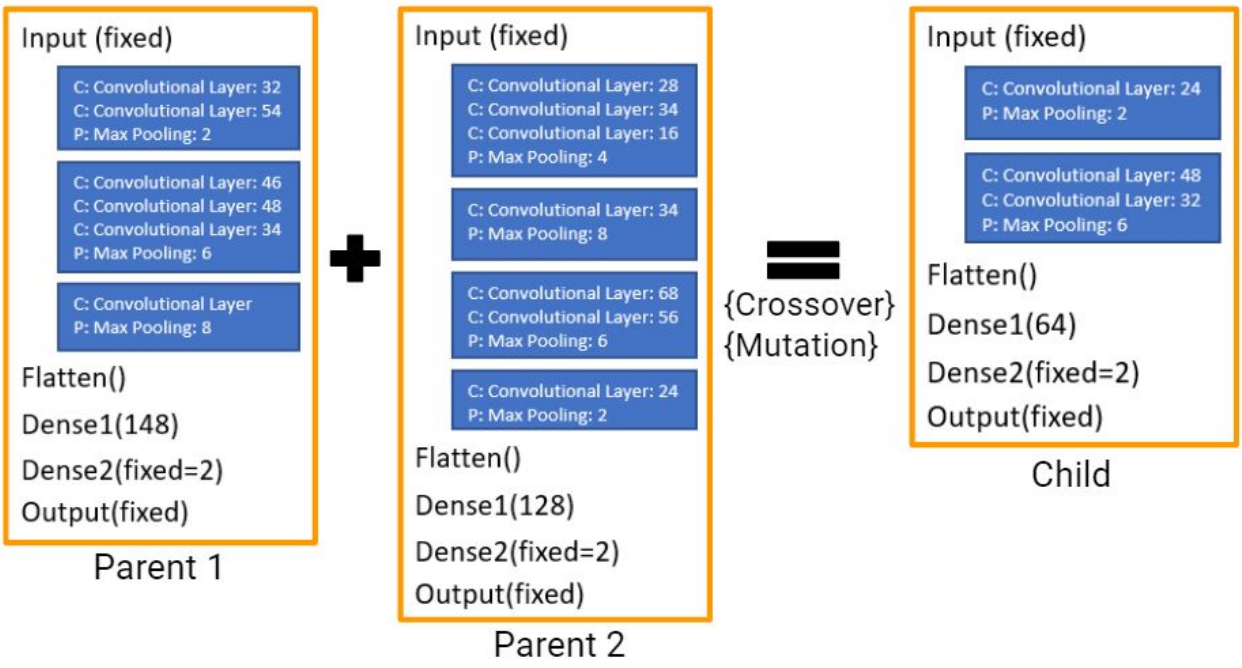
With this, a given chromosome represents a CNN model, and has a gene list containing all parameters and hyper parameters of the CNN it represents.

Every variable/gene within the gene list is stored as a 32-Bit Binary String. This was done to help with the crossover and mutation steps of the genetic algorithm.

Integration with Genetic Algorithm

The implemented Genetic Algorithm will perform the tasks of selection, reproduction (crossover, mutation), and evaluation. With these steps the Genetic Algorithm's population will move towards a more fit (with less validation loss) population.
This means that the chromosomes' gene list will contain the suitable parameters and hyper parameters that are needed to give rise to optimal results.

The algorithm goes through the Initial Population Creation and in this step randomized numbers within ranges are chosen as variables/genes and added to the gene list of each chromosome. After Initial Population Creation, the next step is Selection where the top percent (population retained %) chromosomes are chosen based on the fitness (lowest validation loss).
Next, the Reproduction step (consisting of Crossover, Mutation, and Evaluation) takes place. The figure below shows an example of this step. From the most fit population (obtained from Selection), 2 parents are chosen and crossed over. Under a chance of 0.5 mutation takes place after the cross over.



The Crossover and Mutation is performed on every gene within the gene list. Thus, we see that when two parents are cross overed, the resultant child can have a very different architecture and other hyper parameters / parameters. The method of crossover used is known as the Two-Point Crossover, and the Mutation uses a flip digit technique.

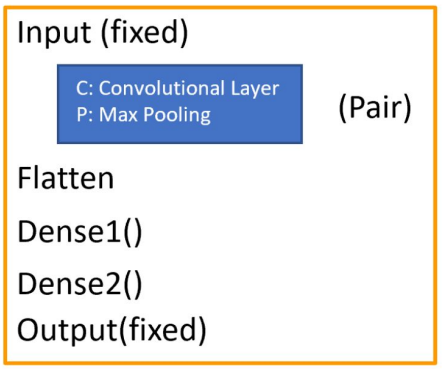
Next is the Evaluation step where out of the two children produced from two parents during the Reproduction, the one with a better fitness (lower validation), is added to the Population. Chromosomes are evaluated using the Fitness Function which returns their fitness, ie - Validation Loss. This will be used to repopulate the lost population during the Selection step.

This means, the Reproduction step takes place for *(1 - Population Retained) * Population Size* times.

These steps consisting of Selection, Reproduction (Crossover, Mutation, Evaluation), are performed for how many ever generations were specified in the program.
Gradually, the population begins to have more fit chromosomes over generations.
Thus, the goal of the Genetic Algorithm is to minimize the fitness value (validation loss).

Fitness Function

Everytime a new chromosome is created during the reproduction stage of crossover and mutation, a new genelist is passed to the Fitness Function as parameters for its Evaluation (calculating fitness).
The parameters are parsed and converted from binary format to their numerical values, and are used to build the model's architecture and hyperparameters.

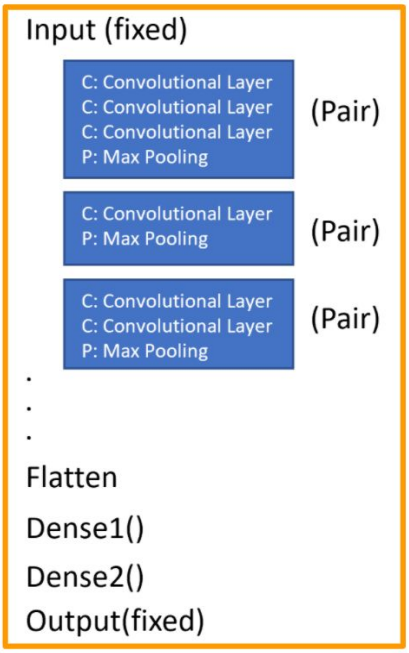


Baseline model

Using the parameters, the CNN models are generated. The minimal model that can be constructed is shown in the Baseline model figure.

A general example CNN model constructed is shown in the Generated Model figure below. In addition to using the parameters to create the model, it also uses the training data shape for the input data shape in the first Conv 2D layer, and uses the number of classes in the data for the number of nodes in the final Dense layer.

Once the CNN model is created, it is trained using model.fit() with the training data for 50 epochs using early stopping with a patience of 1. After training, the model is then evaluated with the validation data to obtain the validation accuracy and the validation loss. As mentioned earlier, the validation loss is returned as the result of the Fitness Function since it is the fitness value for all chromosomes.

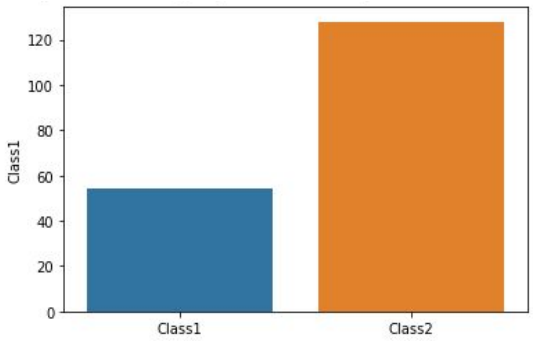


Generated model

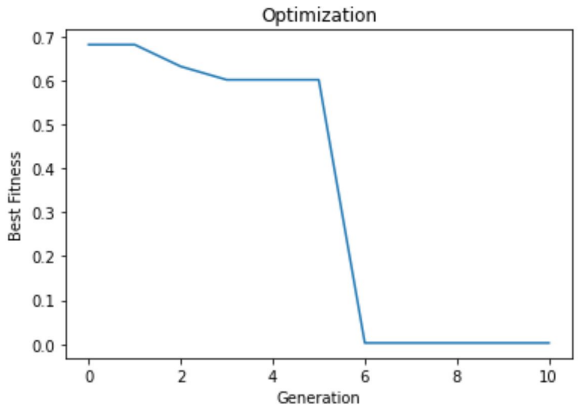
Results.

The image data and labels from Vermeer were loaded using numpy, and had the following shape and class ratio.

```
Image Data:  (182, 1000, 91, 1)
Labels:  (182, 2)
```

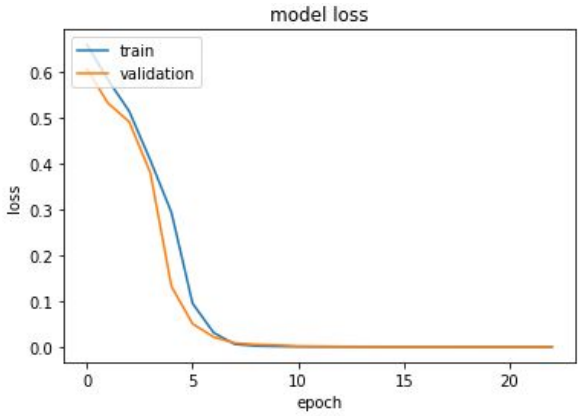


Despite unbalanced data, the algorithm worked very well. First, the dataset is split into training, validation, and testing datasets, by splitting in a 0.9 and 0.1 ratio twice. The Genetic Algorithm was run for 10 generations, with a Population Size of 2, and Population Retained value of 0.5 (indicating 50%).
As we see in the graph on the right, the validation loss reaches near 0.0 validation loss over 10 generations.



We are able to notice that Genetic Algorithm by nature gradually reaches the optimal solution.

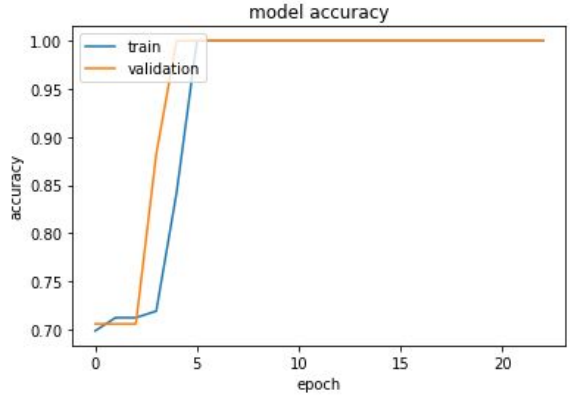
As the algorithm runs, the best model is saved as a .hdf5 file with checkpoints & a .txt file is used to store all the values of the gene list to save the parameters and hyper parameters.



The best model was loaded and evaluated on the test data, and showed excellent results.

The algorithm successfully works on Vermeer Data:

- Converges very well
- Validation Loss is brought to minimum
- Near 100% accuracy and 0.0% loss on test data



Limitations.

Based on the dataset the algorithm is run on, it will use lots of memory. This is because models are created and trained & validated on data during each generation for each chromosome. This sometimes leads to OOM (out of memory errors) on GCP VMs and Google Colab Notebooks. The algorithm needs lots of compute, hence requires GPUs to be able to run it on a dataset. More variables need to be added such as Learning Rate, Dropout Layer, Patience for Early Stopping to create more options for the Genetic Algorithm as these values might need to be different for various datasets. Another limitation is that, so far, the algorithm has been tested on only a few datasets for convergence.

Future Works.

Improvement will be focused on adding more hyper parameter variables and tweaking the algorithm for better convergence on any dataset.
Testing will be focused on new datasets to check performance. Also, testing will need to be conducted with different population size, generations, other parameters & ranges for filters / nodes.
Lastly, the algorithm needs compared with results from other Research Algorithms. Once, all this is completed, we can write a paper on the algorithm and techniques used.

Team Members.

Karthik Subbarao
Jichuan Zhang
Brad Akin
Sam Henkes
William Doughty

Client.

Vermeer Corporation

Project Owner.

Dr. Reza Morsali

Advisor.

Dr. Adisak Sukul