Genetic Algorithm for Transfer Learning in a deep CNN

Mohammad Safiuddin

B170765ME

S6 ME-B

Done as an assignment for the course ME4126D – Optimization Methods in Engineering

Abstract

Convolutional Neural Networks (CNN) are extremely effective for image recognition tasks, and they have become popular in recent years. Most state-of-the-art CNN architectures are designed manually with domain expertise. Designing the architecture for a CNN is a cumbersome task because of the numerous parameters to configure, including activation functions, layer types and hyperparameters. Current CNN architectures complex and require a lot of time to train on large image data sets. Transfer Learning and Fine-tuning can reduce the training time significantly, but they require a lot of manual experimentation to find the best architecture.

This assignment aims to state a method which utilizes a Genetic Algorithm to find the best architecture for Transfer Learning and fine-tuning without much manual intervention. A group of hyperparameters is constructed (chromosome) and these are again grouped to form a population. This population goes through a Genetic Algorithm and after a few generations, the algorithm will properly find better hyperparameters for the CNN.

Keywords**:** Genetic Algorithm, Convolutional Neural Networks, Hyper-parameters,

Transfer Learning, Fine-tuning

Introduction

What is a Neural Network?

The basic concept behind artificial Neural Networks was built upon hypotheses and models of how the human brain works to solve complex problem tasks. A neural network is a combination of many small units called neurons. Each neuron takes in some inputs and a weighted sum of these inputs is calculated which is then passed through an activation function. A simple neuron classifier Adaline **(B.Widrow, 1960)** is shown below in Fig.1

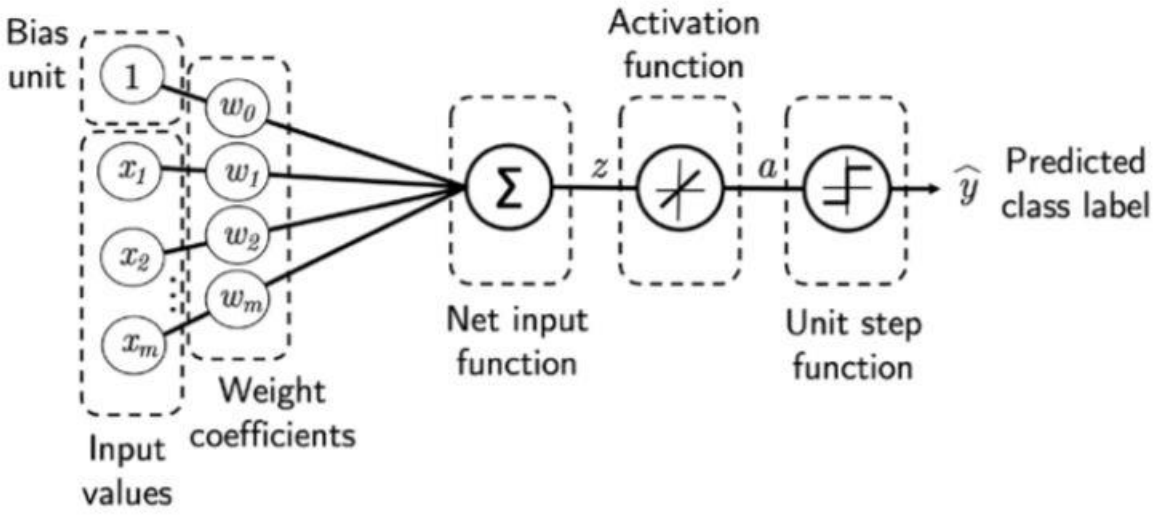

Fig.1 An AdaLine Classfier

The unit step function in Fig.1 is required only for classification tasks and neurons in the hidden layer exclude it.

A network or circuit of neurons is called a Neural Network. The connections between neurons do not form a loop therefore these Neural Networks are referred to as Feed Forward Neural Networks.

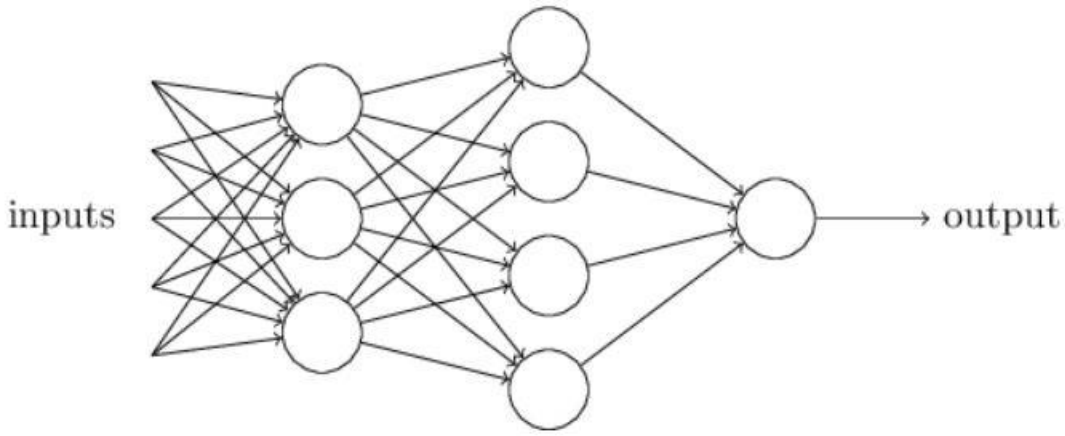

Fig.2 A Feed Forward Neural Network

These Neural Networks are trained by minimizing a loss function which is defined based on the task at hand. This loss function is minimized or maximized using gradient-based optimization algorithms like Stochastic Gradient Descent (SGD). The ability of neural networks to approximate any function is the reason why NNs gained traction in the past few years. This ability of Neural Networks is described by Universal Approximation Theorem. A single hidden layer feed-forward neural network with one neuron in the hidden layer can approximate any univariate function **(Guliyev and Ismailov, 2016)**

What is a Convolutional Neural Network?

If the standard Neural Networks are to be applied to image data, every single pixel of the image must be given as input to the Feed Forward NN. This is highly inefficient as images have a very large number of pixels and training this NN will require lots of computational resources. Also, images have a lot of shared features within them and taking advantage of these common localized features will not only make the model more efficient but also will increase its ability to generalize better to new images as adjacent pixels together make more sense as far as image semantics are concerned.

The idea Convolutional Neural Networks (CNNs) was inspired by how the visual cortex of our brain works when recognizing objects. Visual Cortex uses a hierarchical system which detects various layers of abstraction to recognize objects. **(Hubel and Wiesel, 1962)**

The Convolution Operator

The convolutional operator is denoted by $*$ and is defined as follows for univariate functions.

$$s(t) = (x * w)(t) = \sum_{a=\infty}^{-\infty} x(a)\, w(t-a)$$

In most machine learning and deep learning applications the inputs are multidimensional and also the kernel is multidimensional. If or example an image *I* with two dimensions and a kernel *K* with two dimensions are considered, then the convolution between them can be written as,

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)\, K(i-m, j-n)$$

As convolution is commutative, we can write,

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)\, K(m,n)$$

The commutative property is due to the flipping of the kernel *K* with respect to the input image *I*. This is not useful for the neural net implementation. Therefore we can write it as

$$\boldsymbol{S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)\, K(m,n)}$$

In the above implementation, there is no kernel flipping. This is called cross-correlation in mathematics. In machine learning, however, this is still referred to as the Convolution operator

To put it in simple terms the convolution operator $(K * I)$ takes a kernel *K* and overlaps it over the top left of the image *I* and does an element-wise multiplication

and sums them up then, takes one more step and repeats the same process until it has swiped over the entire image. This process is shown in Fig.3 and Fig.4

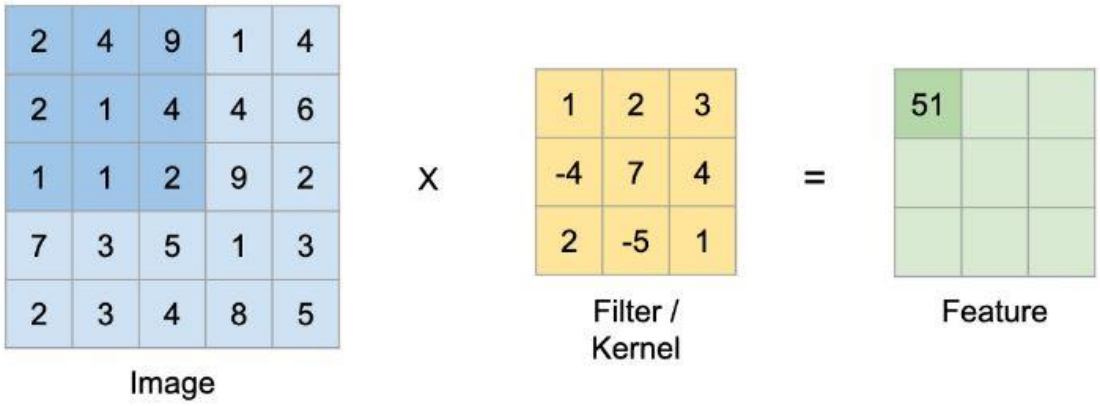

Fig.3 Convolution operation in action-1

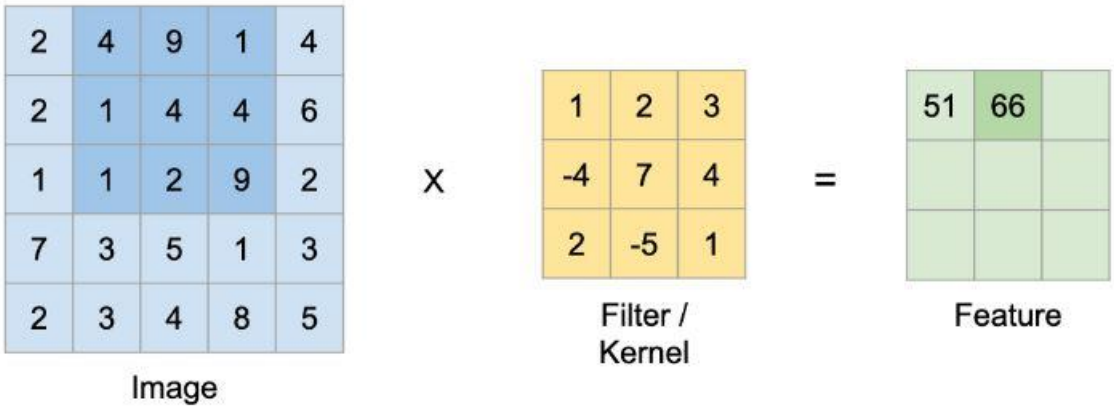

Fig.4 Convolution operation in action-2

The process shown in the above figures is repeated until the entire image is swiped over by the kernel.

Images usually tend to have 3 input channels (RGB) which makes the image 3 dimensional and kernel for this case has 3 dimensions. All the outputs along the third dimension are summed up to give a 2-dimensional output. The convolution for the 3-dimensional image is shown below

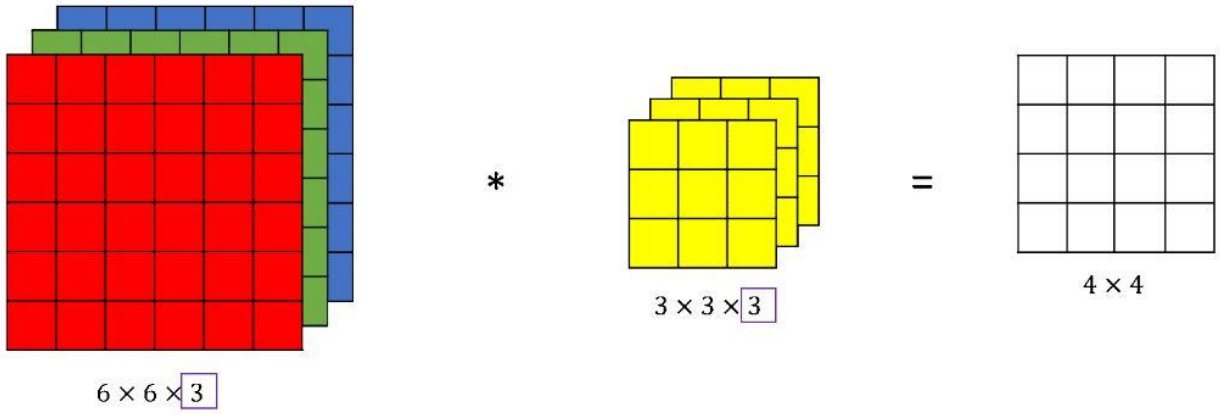

Fig.5 Convolution on an RGB Image with a 3d kernel

CNN Architectures

CNN is composed of different types of layers. These layers are the building blocks of the CNN architecture

- **Convolutional Layer:** This layer generates feature maps using convolutions on inputs. The kernels are randomly initialized to have a particular mean and variance.
- **Pooling Layer:** Down samples the amount of information from the convolutional layer

Convolutional and Pooling layers together are used many times within the architecture and their output is passed through an activation function (ReLU activation is common)

- **Fully connected layers:** A general Feed Forward NN which are usually placed at the end of the architecture. It takes the output from the previous layers and flattens it to a single vector and then uses it as an input. ReLU is a common activation for these layers
- **Output layers:** These are fully connected layers which are placed after the Fully connected layers with ReLU activation. These layers have the same number of neurons as the number of classes in the given classification problem. They use SoftMax activation to predict the class probabilities

LeNet-5 was the first CNN architecture to be built and had a total of 5 layers **(Lecun *et al.*, 1998)**. Modern CNN architectures are much deeper and perform better.

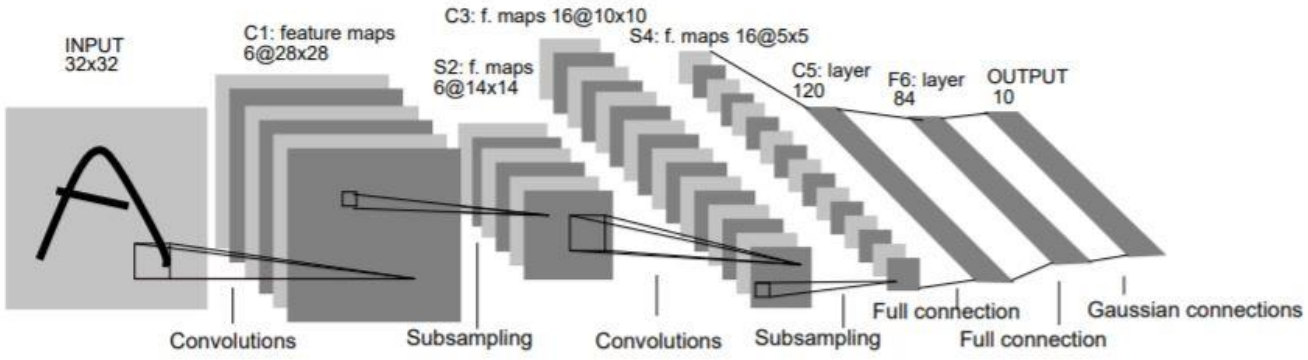

Fig.6 LeNet-5 for digit recognition

Transfer Learning Methodology using Genetic Algorithm

DenseNet Architecture

For this assignment, DenseNet Architecture **(Huang *et al.*, 2017)** was chosen. DenseNet concatenates the feature maps such that each layer's inputs contains feature maps from all previous layers' outputs. This arrangement is called Dense Connection and layer which are densely connected are referred to as a Dense Block. A single dense block consists of

1. Batch Normalization **(Ioffe and Szegedy, 2015)**
2. ReLU activation
3. Convolution with a kernel size 3×3

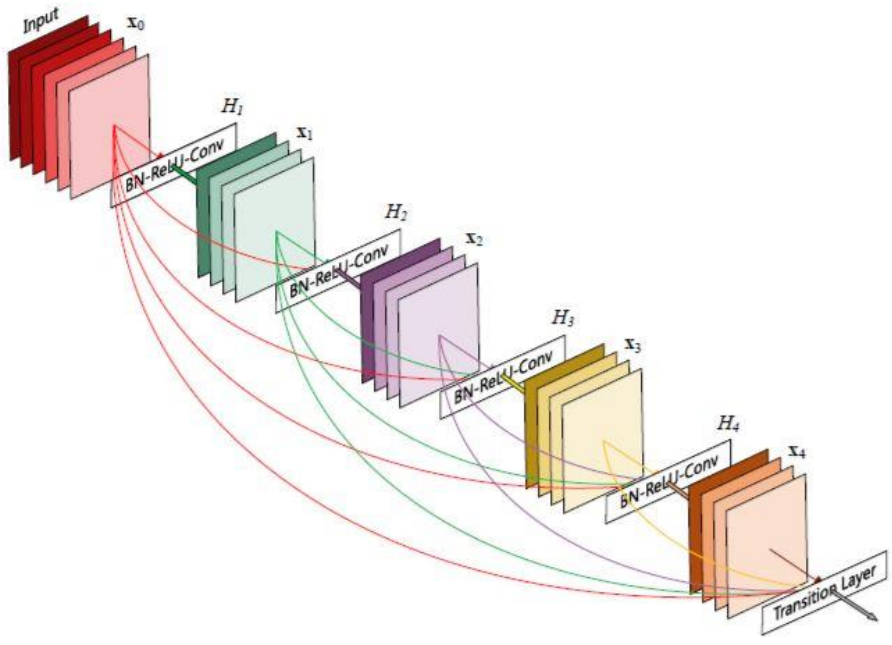

Fig.7 Structure of a Dense Block

This Dense Block structure helps the model with the flow of gradient during backpropagation and also prevents it from learning redundant feature maps. There is a Transition layer between two Dense blocks for down-sampling. For more implementational details refer to the DenseNet paper **(Huang *et al.*, 2017)**

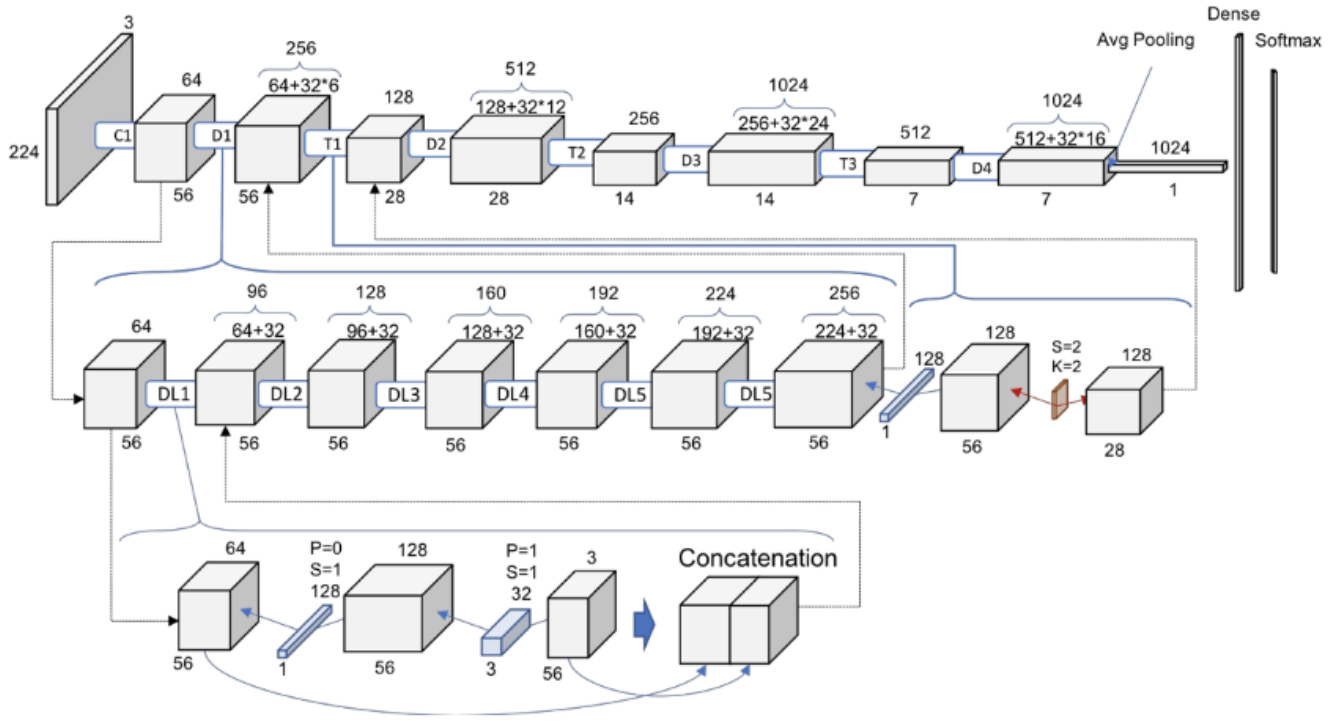

Fig.8 Schematic representation of densenet-121 model.

Transfer learning using a pre-trained model

A model which was already trained on a standard dataset is called a pre-trained model. Most researchers train their models on standard datasets for benchmarking. For Computer Vision ImageNet **(Deng *et al.*, 2009)** is a standard

dataset for training models. This model will have already learned to recognize low-level features well and thus can be used on other image datasets. This technique of utilizing pre-trained models on new datasets is called Transfer Learning. Transfer Learning reduces training time significantly and makes training models more efficient. Deciding the Number of layers to include, Number of layers to freeze, assigning learning rates etc. in a model requires a lot of manual intervention. We can reduce this experimentation using heuristics like Genetic Algorithm.

Genetic Algorithm

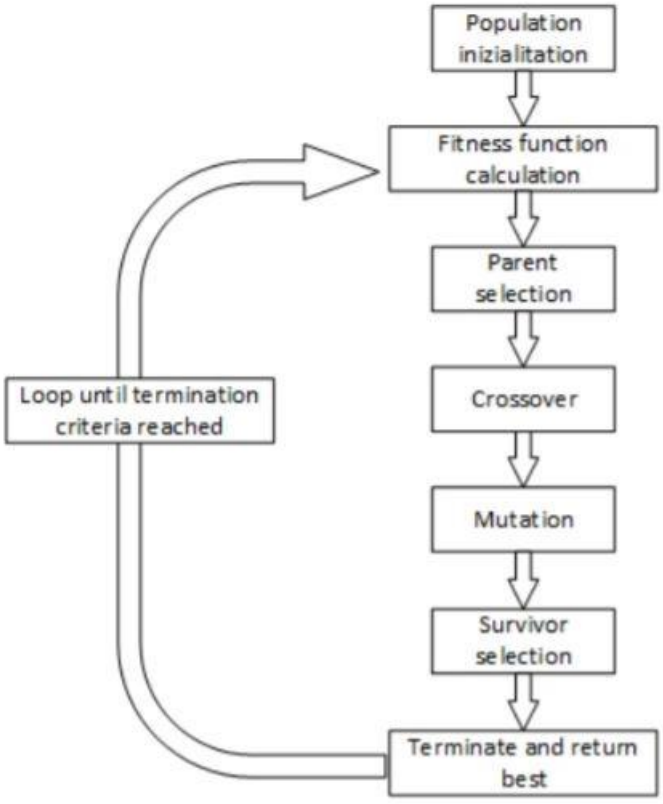

Fig.9 Genetic Algorithm

A chromosome which contains a few cells is generated. A group of chromosomes is generated and it is called a population. This population goes through crossover and mutation.

Random No. 1-58	Random No. 0-18	Random No. 0.1-0.000001	Random No. 0.1-0.9
Included layers	Freeze layers	Learning Rates	Dropout rate

Chromosome initialization scheme

Implementational details

DenseNet implementation

Pytorch framework was used to implement DenseNet. Most of the code used was borrowed from the official Pytorch repository on Github. Code for the SE (Squeeze-Excitation) layers **(Hu, Shen and Sun, 2018)** was added to the standard DenseNet implementation and also a few changes were made to accommodate for the changing hyperparameter values due to Genetic Algorithm.

For the weight initialization, the Pytorch default, Kaiming initialization scheme **(He *et al.*, 2015)** was used. Adam optimization **(Kingma, Ba and Diederik P., 2015)** was used for optimizing the CNN.

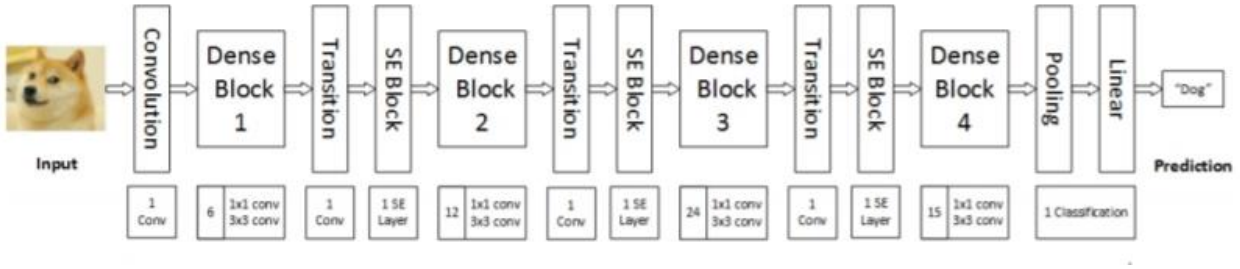

Fig.10 Densenet-121 architecture with SE layers

Genetic Algorithm

After the population was initialized with generated chromosomes the population for the next generation was formed using selection, cross-over and mutation. The type of selection method used was tournament selection (k=2) which is described in the figure below.

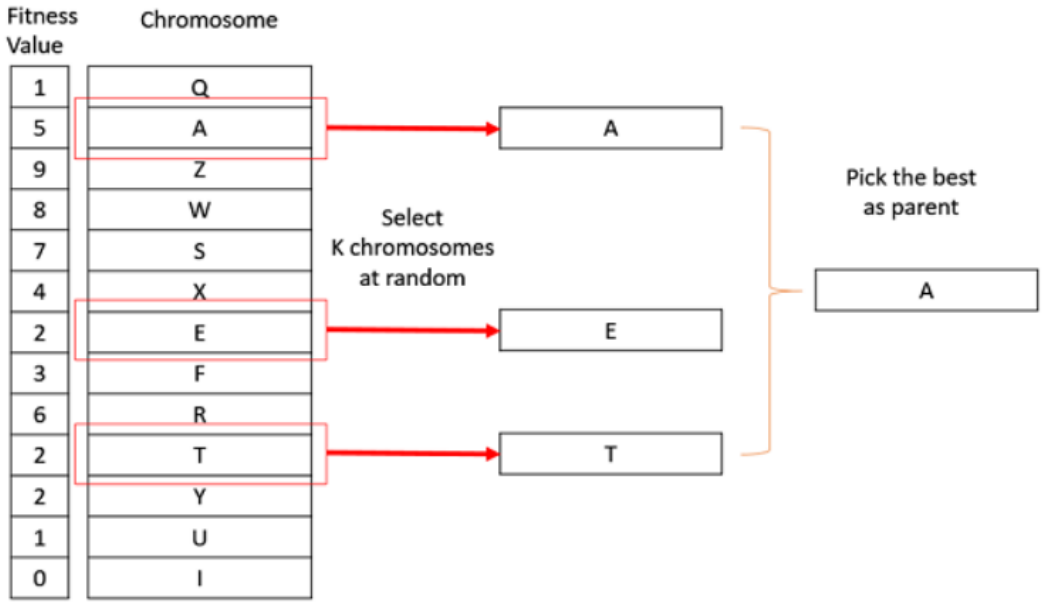

Fig.11 Tournament selection

The selected chromosomes are referred to as parents. The parent chromosomes are crossed over using two-point crossover as shown in the figure below.

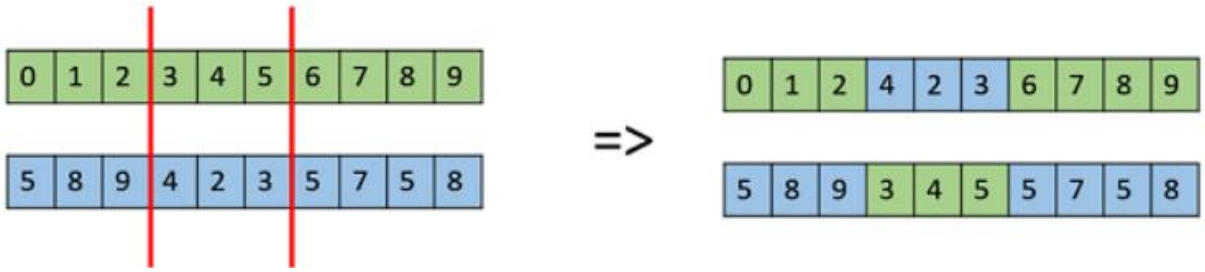

Fig.13 Two-point cross-over

The daughter chromosomes generated are mutated. The mutation, in this case, involves randomly altering the value of cells in each chromosome in a particular range.

Fitness is just the negative loss function of the CNN, therefore, higher the fitness, better the chromosome.

If this Genetic Algorithm is implemented then after a few generations the chromosomes corresponding to a better set of hyperparameters are retained in the population. Thus, we can now use these parameters and train a model on a new dataset to get higher accuracy quickly.

Further techniques to be explored

- Other heuristics like Simulated techniques can be utilized and can be compared to the Genetic Algorithm in this scenario.
- Other CNN architectures can be used instead of DenseNet.

Another Application of Genetic Algorithm in CNNs

A CNN architecture called AmoebaNet **(Real *et al.*, 2019)** was developed using evolution algorithms by effectively searching the architecture space. The downside of this was the high number of parameters used which increased the computational costs significantly.

.

References

- B. Widrow,1960. **An Adaptive "Adaline" Neuron Using Chemical "Memistors,"**

- Deng, J., Dong, W., Socher, R., Li, L.-J., Kai Li, Li Fei-Fei, 2009. **ImageNet: A large-scale hierarchical image database,** in 2009 IEEE Conference on Computer Vision and Pattern Recognition. Presented at the 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255. https://doi.org/10.1109/CVPR.2009.5206848

- Guliyev, N.J., Ismailov, V.E., 2016. **A single hidden layer feedforward network with only one neuron in the hidden layer can approximate any**

univariate function. Neural Comput. 28, 1289–1304. https://doi.org/10.1162/NECO_a_00849

- He, K., Zhang, X., Ren, S., Sun, J., 2015. **Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,** in 2015 IEEE International Conference on Computer Vision (ICCV). Presented at the 2015 IEEE International Conference on Computer Vision (ICCV), pp. 1026–1034. https://doi.org/10.1109/ICCV.2015.123

- Hu, J., Shen, L., Sun, G., 2018. **Squeeze-and-Excitation Networks,** in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. Presented at the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 7132–7141. https://doi.org/10.1109/CVPR.2018.00745

- Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q., 2017. **Densely Connected Convolutional Networks,** in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Presented at the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2261–2269. https://doi.org/10.1109/CVPR.2017.243

- Hubel, D.H., Wiesel, T.N., 1962. **Receptive fields, binocular interaction and functional architecture in the cat's visual cortex.** J. Physiol. 160, 106-154.2.

- Ioffe, S., Szegedy, C., 2015. **Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.** ArXiv150203167 Cs.

- Kingma, Ba, Diederik P., 2015. **Adam: A method for stochastic optimization.** 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings 1–5.

- Lecun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. **Gradient-based learning applied to document recognition.** Proc. IEEE 86, 2278–2324. https://doi.org/10.1109/5.726791

- Real, E., Aggarwal, A., Huang, Y., Le, Q.V., 2019. **Regularized Evolution for Image Classifier Architecture Search.** ArXiv180201548 Cs.