Analysis of Detector Simulation Data using Machine Learning

Shreyansh Agrawal

Department of Physics, Indian Institute of Science Education and Research Bhopal, Bhauri, Bhopal, M.P. - 462066.

Ritesh K Singh

Department of Physical Sciences, Indian Institute of Science Education and Research Kolkata, Mohanpur, Nadia, West Bengal - 741246.

Table of Contents

Analysis of Detector Simulation Data using Machine Learning

Shreyansh Agrawal

Department of Physics, Indian Institute of Science Education and Research Bhopal, Bhauri, Bhopal, M.P. - 462066.

Ritesh K Singh

Department of Physical Sciences, Indian Institute of Science Education and Research Kolkata, Mohanpur, Nadia, West Bengal - 741246.

Abstract

Machine learning (ML) is a field of study that uses algorithms to discover meaningful information in the data. Machine Learning has found its application in almost every major field dealing with data analysis, ranging from medical science, stock markets, photography to High energy physics. ML can be subcategorized into supervised learning, where humans tell the algorithm the ‘correct’ answer, unsupervised learning, where the algorithm is just fed the data, and Reinforcement learning, where the only supervision involved is telling the algorithm how good it is doing. The phrase Deep Learning refers to an approach to ML where multiple specialized layers of computations, such as neural networks, are stacked over each other to make a ‘deep’ architechture. Neural networks try to mimick the actual biological neurons in order to create a more efficient model for learning. While the first neural networks (termed as 'perceptron') date back to 1960's, the development in technology leading to availability of high performance computation and big data have resulted in regrown interest in this field. These networks can further be Fully connected, Convoluted or Recurring depending on the task that they are deployed for. Convoluted Neural Networks are used for data with spatial correlation and Recurrent Neural Networks are used for data with temporal correlation. Data from a collider detector can be interpreted as an image and the methods of Convolution Neural Network can be used to analyse such data.

Keywords or phrases: High Energy Physics, Deep Learning, Neural Networks, Convolution, Computer Vision, Particle Identification.

1 INTRODUCTION TO MACHINE LEARNING

1.1 Machine Lerning

Formally machine learning can be defined as: "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E." [Tom Mitchell (1998)].

The algorithm tries to find patterns in the data it is provided and learns from it, then it can apply these patterns to real world data. This is different from the conventional programming which involves the programmer specifying the exact flow chart which the algorithm follows each time identically.

Feature: Set of numeric values that describe the input. For example, in case of house prices, the features could be the area of the house, how old it is, number of rooms, etc.

Parameters: The variables whose values the program will 'learn' during the learning process by trying to minimize the training error. These are the weight values between the layers. The trained model will remember these values and apply them to all the future data it is given to evaluate.

Hyperparameters: Parameters that are defined before training by the user. These control how the algorithm will be trained. For example, how many epochs the algorithm will train on, what will be the step size for each gradient descent, how much momentum to retain during training, etc. A poor set of hyperparameters can result in overall poor performance of the model. So many different values of hyperparameters are tested during the training.

Training error: The error in the output that the model produces with the training data, compared to the correct output for those data. Training error is calculated using a **cost (or loss) function**.

Generalization error: After the training rocess, the model is exposed to general data that the model would see in the real world. Error that the algorithm gets when exposed to a data it hasn't seen before during training is the generalization error.

Representation: How much data complexity can the model learn. If the comlexity in the trends of the data is more than the representational power of the model, it will not be able to extract meaningful information.

1.1.1 Supervised Learning

When the algorithm trains on a set of data that it knows the correct output to, it is said supervised learning. For example, if the task of the program is to predict the housing prices based on some features of a house, then the supervision can be providing the prices of houses that have already been sold with the values of the features. In this case, the dataset that the algorithm would train on contains the features, which are the datapoints, and labels, which are the correct output for each datapoint. The training starts with a random guess and with each cycle of the training, the algorithm will make adjustments to get its output closer to the ground truth, i.e. the provided labels.

Regression

Regression is a problem of predicting an output function which is continuos. The data provided consists of data points and the correct output which then the hypothesis fuction tries to fit upon. For example, given the area, number of bedrooms, age, and prices of some flats, the task of the program could be to predict the prices of other flats given the other details for that flat. This can be done using a linear function:

$$h_{\theta,b}(x) = \sum_i \theta^{(i)} x^{(i)} + b \tag{1}$$

Where $x^{(i)}$ are the features of the flat and b and $\theta^{(i)}$ are the parameters that the program will learn. 'b' here is the bias which ensures that the hypothesis function does not always pass through the origin.

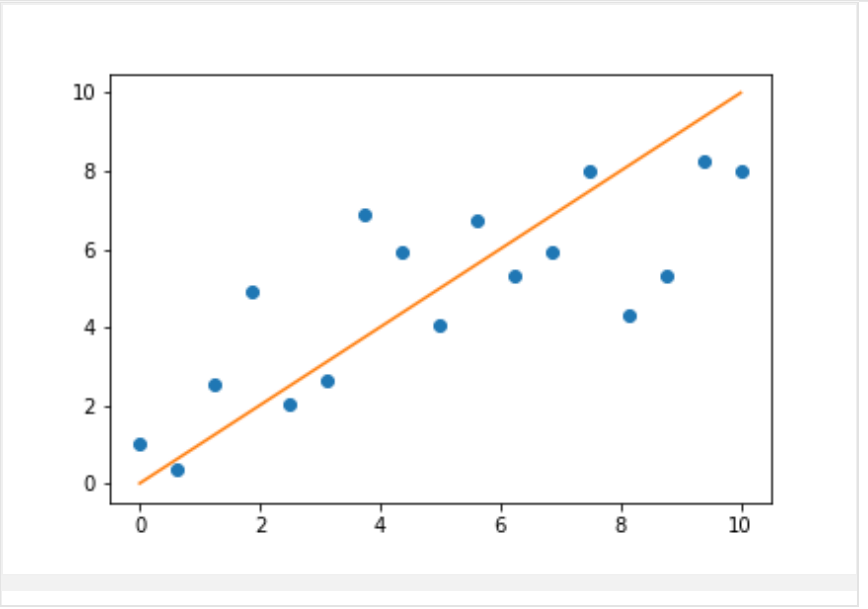

Fig 1 Example of regression. Solid line represents the hypothesis fuction and points are the data points.

Classification

Classification is diffrent from regression as the final output of classification task is discrete and not continuous. An example of classification task using machine learning is to determine the species of lily using different provided features such as petal size. The task of the program is to assign labels to each feature vectors.

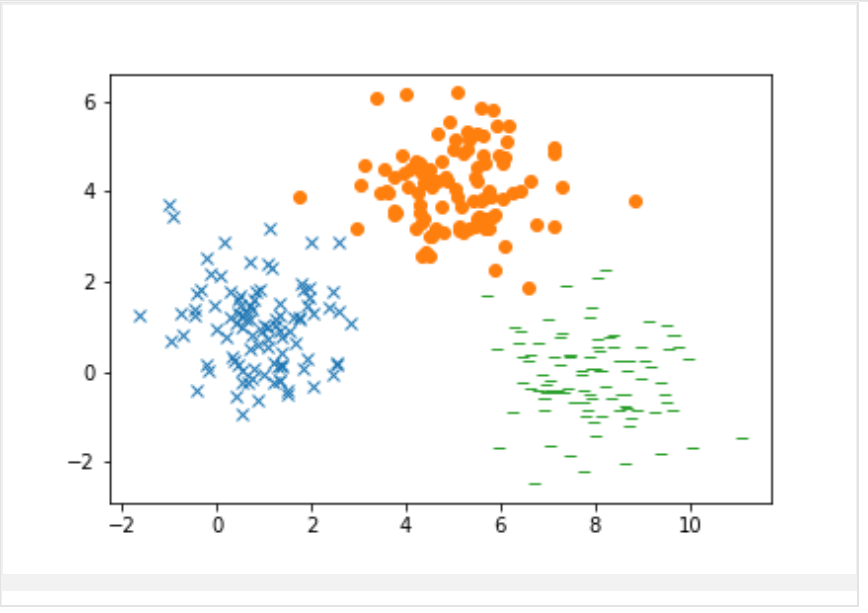

Fig 2 Example of Classification data with the shape of each data point representing its class.

1.1.2 Unsupervised Learning

Clustering

The program distributes the given data into clusters based on the similarity of the features. The data are not pre-labelled so there is no 'correct cluster' that an example woud belong to. An example could be grouping different species based on their genetic similarities.

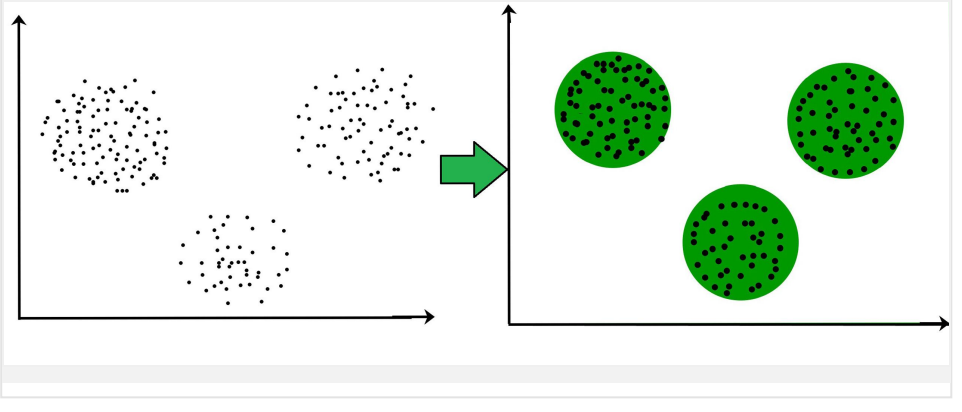

Fig 3 Clustering of unlabelled data into 3 clusters. Source: https://www.geeksforgeeks.org/

clustering-in-machine-learning/

Dinemsionality Reduction

When the data has a lot of features it can slow down the computation. However, some of these features maybe redundant due to multiple measurements or some can even have high correlation. Such features can be reduced from the data using unsupervised learning. So the program reduces the number of features that an algorithm is working on without changing the data by large.

Feature Extraction

If we do not know what can be a set of features for the given dataset, such programs can extract the features that it deems fit for learning. For example, in a bunch of movies rated by different users, the program will decide on different features for the movies, such as a bunch of movies liked by a certain group of people may have more 'action' than other movies. The program would not be told of such features beforehand and therefore it will learn these in an unsupervised fashion.

Noise Reduction

The task of the program is to remove noise from incoming data. Such as, guessing missing pixels in an image. It is another class of unsupervised learning as it only tends to denoise the input and does not know the correct output during training.

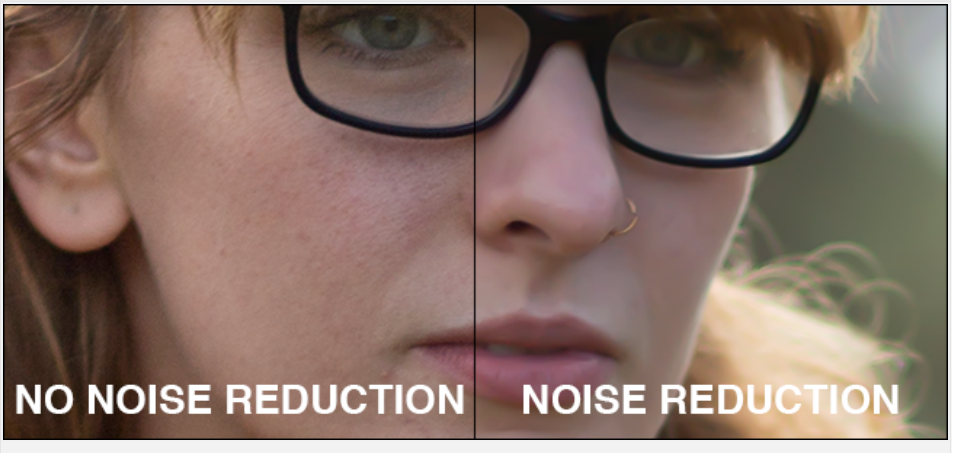

Fig 4 Example of Noise Reduction on an image. Source: https://www.howtogeek.com/368550/what-

is-noise-reduction-in-digital-images/

1.1.3 Other Learning Strategies

Reinforcement Learning

The program, which is the agent here, performs an action in the environment given the state of the environment. The action is chosen based on a policy that depends on the state of the environment. After the action is performed, the environment will give a feedback to the agent. For example, in a game of chess, the state would be the position of pieces on the board, the action would be the next move that the player makes and the feedback would then be how good that move was. Based on the feedback, the agent can then update its policy to take better actions next time. Reinforcement learning helps the program work in a dynamic environment.

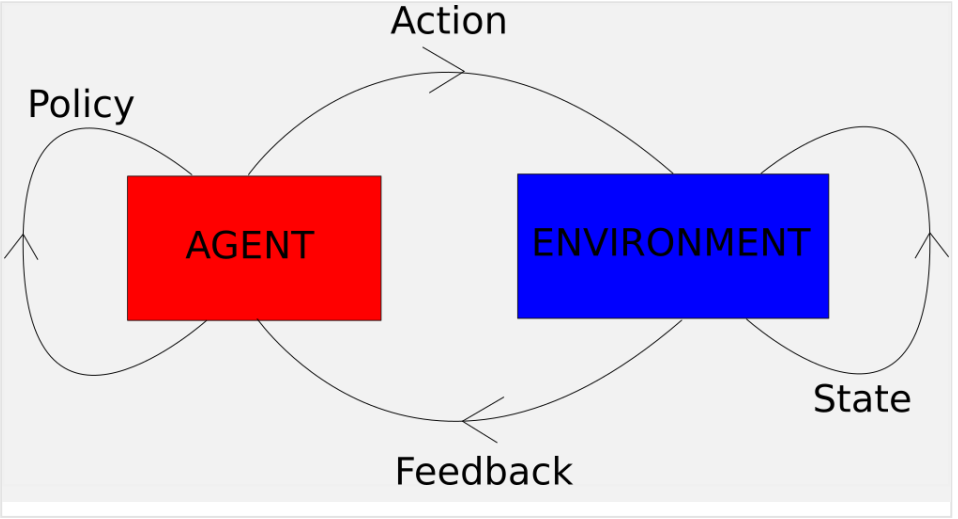

Fig 5 Schematic representation of RL

Generator

Generators are programs that generates new outputs similar to the input provided but not the same. There is no 'correct output', each time a generator is run on the same input it can give different outputs, however, all the outputs are similar to the input, implying that there is some

sense of what output is expected. For example generating patterns similar to the given pattern can be useful many times.

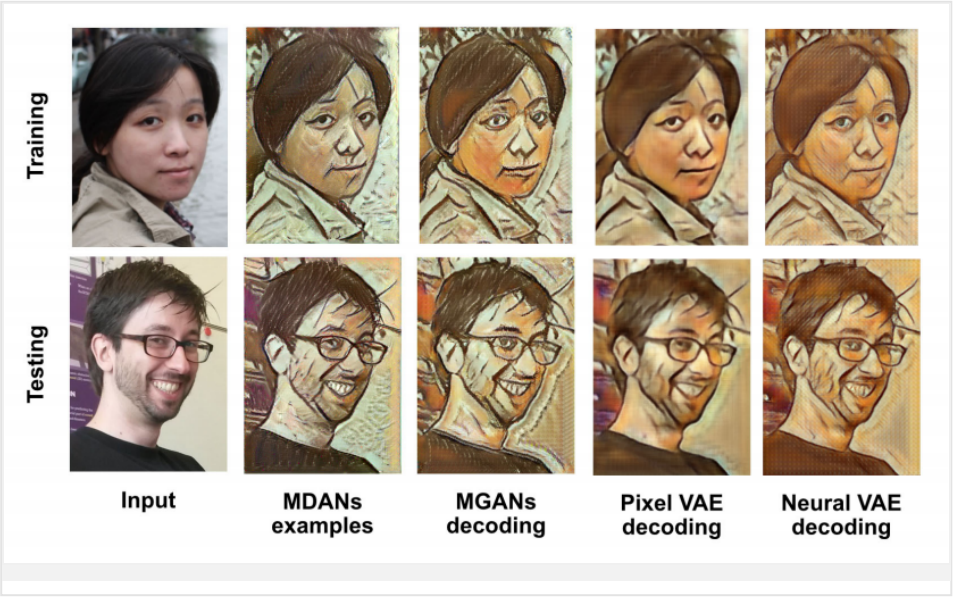

Fig 6 Generating outputs that are similar but not same as the given image using different generators. Source: https://medium.com/@jonathan_hui/gan-some-cool-applications-of-gans-4c9ecca35900

Deep Learning

Many times a single computational operation might not be enough to represent the complexity of the dataset. When multiple layers of computation are added one after the other to build up the model, such a system constitutes Deep Learning.

Evlutionary Algorithms

Instead of creating a single model and improving it each time, evolutionary algorithms create a population of models. Out of this population the models that perform the worst are then killed and the top ones are crossed over to create the population of the next generation. After crossing over, there can be mutation in some of the models of the new generation. In this manner each generation improves a little.

1.2 Mathematical Prelimanaries

1.2.1 Probability and Statistics

For an event A belonging to a sample space, P(A) gives the probability that the event A will occur. For example if a die is rolled, the even will be what number appears on the die. The sample space will contain all the possible events, so in this case the sample space, Ω = {(1), (2), (3), (4), (5), (6)}. Then the probability of a certain event happening, say getting a 4 would be $P(4) = \frac{1}{6}$, i.e., the number of occurences of the event in the sample space devided by the total size of the sample space. The Probability space is then defined by (Ω, A, P(A)).

Random Variable

A Random Vriable (RV) is a function defined on the probability space which associates with every event a real number. For example, when two dice are rolled simultaneously, the total that appears on the two dice is a random variable that associates a real number between 2 and 12 to every event. Then the probability of getting a certain value of random variable would be the sum of probabilities of all events that are mapped to that value. So for a random variable X given on a probability space (Ω, A, P) the probability of getting a certain value of the RV, say x, will be:

$$P(X = x) = \sum_{X(A)=x} P(A) \tag{2}$$

Expectation Value

Expectation value of a RV defines the average value that will be obtained over many trials. It is the weighed mean associated with the RV. It is defined as:

$$\bar{X} = \sum_{x_i \in X(A)} x_i P(X = x_i) \tag{3}$$

Functions on Random Variable

Prbobility distribution funcion:

$$F(a) = P(X < a) \tag{4}$$

For a discrete RV, probability mass function is the probability of getting the respective value of the RV.

For a continuous RV, probability that the value of RV is in an interval [a, b] is given by:

$$P(a < X < b) = \int_a^b f(x)dx \tag{5}$$

Where f(x) is called the density function associated with the RV.

i.i.d

A set of RVs that have the same distribution function and each of their values are independent of the other are called independent identically distributed (i.i.d) RVs.

Covarianve and Correlation

For a RV X, the variance (σ^2) of X is the mean squared deviation from the expectation value of X.

$$\sigma^2 = \langle (x^{(i)} - \bar{X})^2 \rangle \tag{6 a}$$

$$\sigma^2 = \langle X^2 \rangle - \langle X \rangle^2 \tag{6 b}$$

For two RVs X and Y over the same probability space, the variance of the sum of X and Y is:

$$\sigma^2(X + Y) = \sigma^2(X) + \sigma^2(Y) + 2 * \sigma(X) * \sigma(Y) \tag{7}$$

Where, the third term is called the covariance of X and Y (Cov(X, Y)). Covariance determines the degree of dependence of one RV on another.

Correlation is the quantitative measure of dependency of the two random variables and is defined by:

$$\rho(X,Y) = \frac{Cov(X,Y)}{\sigma(X)\sigma(Y)} \tag{8}$$

Conditional Probability

If A and B are two events, then the conditional probability, P(A|B), read as the probability of A occuring given B has already occured is given by,

$$P(A|B) = P(A \cap B)P(B) \tag{9}$$

Joint probability is the probability of events A and B ccuring simultaniously and is simply given by P(A)P(B).

Marginal Probability

Given a RV X and another RV Y, from Eq. (9) we can define the probability of X = x as:

$$P(X = x) = \sum_{y \in Y} P(x|y) \tag{10}$$

This way of definition of probability is called Marginal Probability because of the way it is represented. For example,

Table 1 Marginal Probability of X and Y

Y | X	x_1	x_2	$P_y(Y)$
y_1	1/5	2/5	3/5
y_2	0	2/5	2/5
$P_x(X)$	1/5	4/5	0

Confusion Matrix

Given the result of a hypothesis on a dataset, we can define a confusion matrix for the hypothesis which can measure the performance by comapring actual results and predicted results. For example if the task is to classify whether a given image is of a car or not, then the performance of the hypothesis can be tabulated as:

Table 2 Confusion Matrix

Predicted outcome | Ground Truth	Car	Not Car
Car	True Positive	False Positive
Not Car	False Negative	True Negative

Different performance metrics can then be calculated using the confusion matrix depending on the requirement of the problem.

$$Accuracy = \frac{True\ Positive + True\ Negative}{Total\ Inputs} \tag{11 a}$$

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{11 b}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{11 c}$$

f1 score of a hyothesis defines a metric to compare the precision and recall of algorithms simultaneously. It is defined as:

$$\frac{Precision * Recall}{Precision + Recall} \tag{12}$$

Higher f1 score generally implies a better performing algorithm, but in some cases we only want to optimize one parameter out of precision and recall.

Bootstrapping

A hypothesis is based upon a smaller subset of the population and the performance of the hypothesis might differ when applied to the entire population. Bootstrapping is method of creating many smaller subsets of the sampling set randomly and then using the measure of the performance of hypothesis generated using the smaller subsets on the sampling set, a measure

of performance of hypothesis on the generalized population can be calculated upto the required tolerance.

1.2.2 Gradients

The rate of change of a function along a given axis is called the derivative with respect to that axis. For a surface in n-dimensional space, the direction along which the derivative is maximum gives the direction of the gradient at that point. Gradient on the surface is then defined by:

$$grad(f) = \sum_i \frac{\partial f}{\partial x_i} \hat{i} \tag{13}$$

Gradient Descent

For a given function, we can use the gradients with respect to the parameters to find the minima of that function. For example, if we know the cost function J(θ,x) of the hypothesis function, the cost can be optimized (or minimized) by:

$$\theta_j = \theta_j - \alpha \frac{\partial J(\theta,x)}{\partial \theta_j} \tag{14}$$

Learning Rate

The hyperparameter α in the gradient descent step is called the learning rate. It controls how fast the value of the parameters is changing. A large value can result in overshooting the minima while a small value may lead to very slow convergence.

1.2.3 Information Theory

If the probability of a true statement being correct is P, then the information content of the statement is,

$$Q = -k \, log \, P \tag{15}$$

With k = 1, and base of log as 2, this would give the number of bits of information. A quantitative measure of information is useful for calculating the the performance of various

algorithms, for eaxmple, an algorithm that encodes a data to compress is size can be given a metric using the information it has retained after compression.

Context

The context of the statement being made can change the probability of the statement being correct, e.g., the statement "I see a hippopotamus" carries more information when said in the middle of the city compared to when said at the zoo.

Shanon Entropy

It is the measure of the entropy of a system based on the information it carries. Shanon entropy can be defined as:

$$S = -k \sum_i P_i log P_i \tag{16}$$

If a data has many repeated sequesces, then the transmission can be made efficient by compressing such sequences which reduces the space required for storing data.

Adaptive Code

Codes that are designed to compress data can perform much better if they are specialized based on what data they will be applied to. Specialized codes that differ for the different inputs provided to them are called adaptive code.

Cross Entropy

The performance of a code on a data which it is not specialized for can be calculated using the cross entropy. For two distributions p and q over a given set X, the cross entorpy is defined by

$$H = -k \sum_{x \in X} p(x)\ log\ q(x) \tag{17}$$

This would be the amount of entropy if the compression optimized for distribution q is applied to distribution p.

1.3 Training and Testing

Machine Learning programs are trained over many loops of the training data. Each learning loop is called an **epoch.**

1.3.1 Training Strategy

The model is first randomly intialized, i.e, its parameters are given some random value. The this model is applied to the features in the training set and from there the model's output is obtained. If this output is not the same as the ground truth, then a loss function is used to calculate the error in the output. The aim of the training step is to optimize the loss of the model, i.e., to find the values of the parameters for which the loss is minimum. So, after the loss function is calculated, an optimization is done to the parameters (e.g., gradient descent), to bring the value of parameters closer to optimal. This whole process constitutes of one epoch and the loop is repeated until cnvergence, or a desired performance is reached.

1.3.2 Testing The Model

Test Set

Test set contains the data that the program is not fed during the training. Test set is used to estimate the performance of the program in real world. The test loss gives a measure of how good the program generalizes. From the initial data set, a set fraction is kept aside as the test set while the other set is used to train the data.

Data Leakage

If any of the model's parameters or hypermaters are decided by the best performance on the test data, the test result will not be indicative of generaization then because the model will have a bias towards the test data as well. This data leakage can also occur if during the preprocessing of data, test set and train set were both used together, as again the program will have some knowledge of the test set as well. So to avoid data leakage, the test set is preprocessed seperately and only the final model is exposed to the test set.

Validation

Since the test data can not be used to check the model's performance on unseen dataset, a validation set is also created apart from training and testing set. So the initial set is split in three parts and all of them are processed seperately. Validation data can be useful for fine tuning model's hyperparameters. Since the training data only affects the model's parameters,

the validation loss after training different models can be used to select the model with best hyperparameters. Then the generalization loss of the selected model is evaluated using the test data.

Cross Validation

If the data set is too small to split it into three different sets, we can dynamically split the data. The data is first split into training and cross-validation set, and then the model is trained multipe times with different split sets. The final training and validation loss can then be calculated by averaging over all the loops. Since each time it is trained on a different training and validation sets, it ensures that the program gets trained on all of the data while still keeping a metric for unseen data. For example, in k-fold cross vaidation the data is randomized and split into 'k' folds. Then each model runs over the data 'k' times, each time setting aside one of the folds as validation set and running the training on the rest of the data. Then the model with the best average training and validation loss can be selected as the final model.

1.3.3 Improving the Fit

The training and validation loss during each epoch can be used to say how good the fit of the model to the data was.

Underfitting and Overfitting

Underfitting implies that the model was too simlpe to learn the trends in the data and is therefore a very poor fit on the training set. For an underfit model, training error and validation error are close but they are both very high**.** Overfitting implies that the model got too much specalized on the training data, i.e., it started memorizing the data, this results in a low training loss but a high validation loss. For an overfit model, the training error is very low but the divergence between validation error and training error is high.

Early Stopping

During the training the training loss and the validation loss can be monitered for each epoch. This allows the programmer to stop the training process when the two errors start to diverge, thus preventing overfit.

Learning Curve

The learning curve is a plot of training and validation error against the number of samples. It Can be used in diagnosing overfitting and underfitting. It also gives an insigth if the program

needs more samples to train better. Improvements to the model ca be made by looking at the learning curve.

Bias and Variance

Bias determines how good a curve fits the data points it is fed. A high bias implies underfitting. From the learning curve, high bias is when both training and validation loss meet asymptotically but are still above the desired loss value. So, increasing the sample size will not help in reducing the bias, instead the complexity of the model should be increased. Variance is the diffence in each curve when fitting the data with different sets of random points. High variance implies overfitting which causes the model to be very specific to the data is has been fed. Variance can be checked from the learning curve, when training and validation loss are not close but seem to be approaching each other. Thus, increasing the sample size migth help reduce the variance.

Regularization

If the model is overfitting on the data, regularizing the parameters can help improve the fit. Regularization means adding extra weights to parameters so that they don't get too big. e.g., Linear Regression with regularization, the chi-squared cost function can be modified by adding,

$$J_{reg}(\theta, x) = J(\theta, x) + \sum \lambda \theta_n \tag{18}$$

1.4 Classification and Clustering

1.4.1 Binary Classification

The task of a binary classifier is to determine whether a data belongs to a given class or not. So the output from a binary classifier is either 1 or 0. This can be achieved using a sigmoid hypothesis function. So if the feature vector of the given data is $x^{(i)}$, we can define the sigmoid function as:

$$h_\theta(x) = g(z) = \frac{1}{1+e^{-z}} \tag{19 a}$$

$$z(\theta, x) = \sum \theta^{(i)} x^{(i)} + b \tag{19 b}$$

Here the bias is also included in the parameter vector θ.

The model can then be trained by optimizing the logarithmic loss function defined as:

$$J(\theta, x) = -y * log(h_\theta(x)) - (1 - y) * log(1 - h_\theta(x)) \quad (20)$$

Where y is the correct label, i.e., either 0 or 1. Regularization can be added if needed.

To test the model, a threshold can be set on the output (e.g. 0.5), such that if the output is greater than this thershold it is considered 1 and 0 otherwise. Thus the accuracy of the model can be calculated.

Decision Boundary

For a binary classification, the hypothesis function will give a curve when equated to the threshold value. This curve will devide the dataset into two region, with out 1 and output 0. Thus, this curve is called the decision boundary.

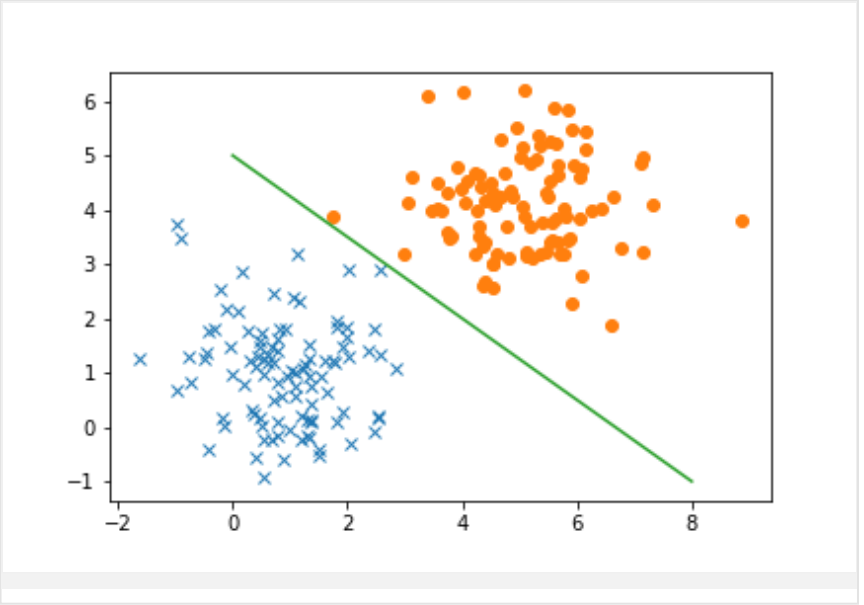

Fig 7 Decision boundary predicted by a binary classifier.

1.4.2 Multi-Class Classification

In most of the cases, the data is needed to be assigned a correct label, for just two labels it can be considered similar to a binary classifier but for more than two, a slightly different strategy has to be used.

One vs All Classification

The probability of the data belonging to each class is calculated independently, i.e., for each class in the output set, a binary classifier is used which returns a value between 0 and 1. Then whichever classifier returns the lasrgest value, is called the label for the given data. So each classifier is trained to tell whether the data belongs to a given class or not.

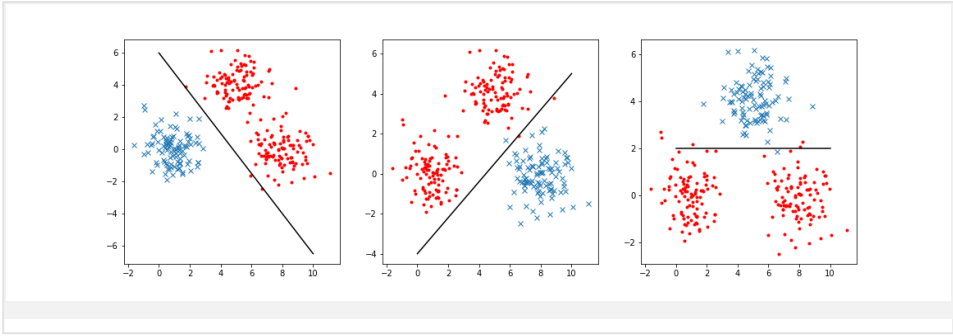

Fig 8 One vs All Classification for data in Fig 2.

One vs One Classification

Instead of using binary classifier on each class, two classes are taken and compared at a time. The best one is then compared with the next class and so on and so forth, until all the classes are exhausted. The assigned label will be the best class from the last comparison. In this case, multiple binary classifiers for each class combination are trained each of which are specialized on segragating two classes at a time.

1.4.3 Other Classifiers

Decision Tree

A decision tree can be used for binary as well as multi class classification. The tree starts at the root which contains all the data. Then based on a selected feature (can be random or can be done according to the variance of each feature or knowledge about data) the node is split (number of split is a hyperparamter here). Thus the first level conatins information about the value at which the split occurs. The split in nodes continue till the number of samples of a class in a node reach above a certain threshold (say 80%), then this leaf (a node with no further split) is then assigned that class. To evaluate a sample, it is traversed along the tree folloing the split values, and is assigned the class of the leaf that it lands into.

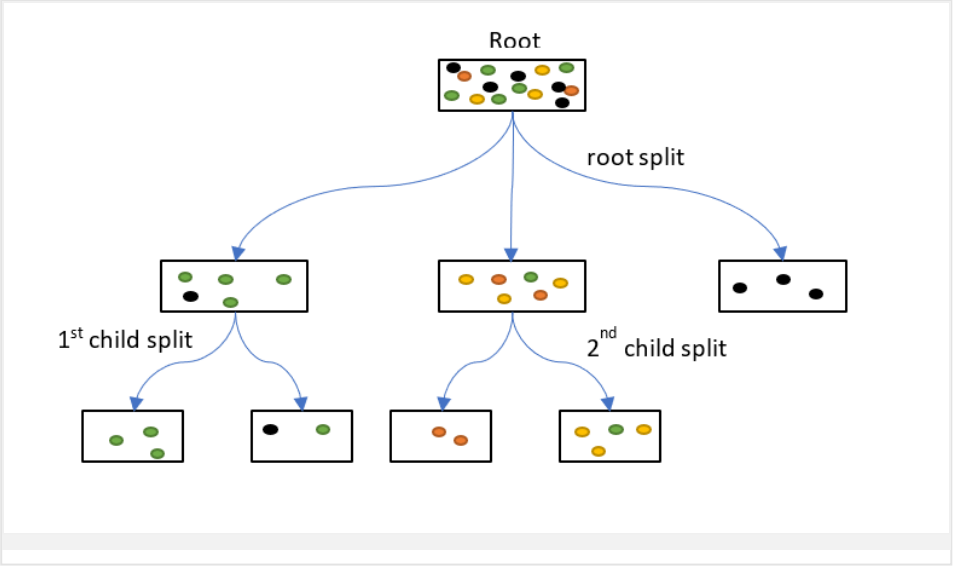

Fig 9 Example of a Decision Tree. Source: https://github.com/benedekrozemberczki/awesome-decision-tree-papers

Random Forest

The performance of the decision tree can be improved by generating a 'Random Forest'. This creates multiple decision trees which each randomly select a feature at each step to split. Therefore, the structure of each tree will be different. For the final output, each tree casts a vote (the label that it predicts) and the class with the most votes is assigned to the data.

k Nearest Neighbour

Instead of training the parameters, the program will simply store all of the training data. When asked to do prediction, the program will check the distance of the given data point from

the stored data point and then consider the class of each of the k nearest neighbours, whichever will be the class of the majority of the neighbours.

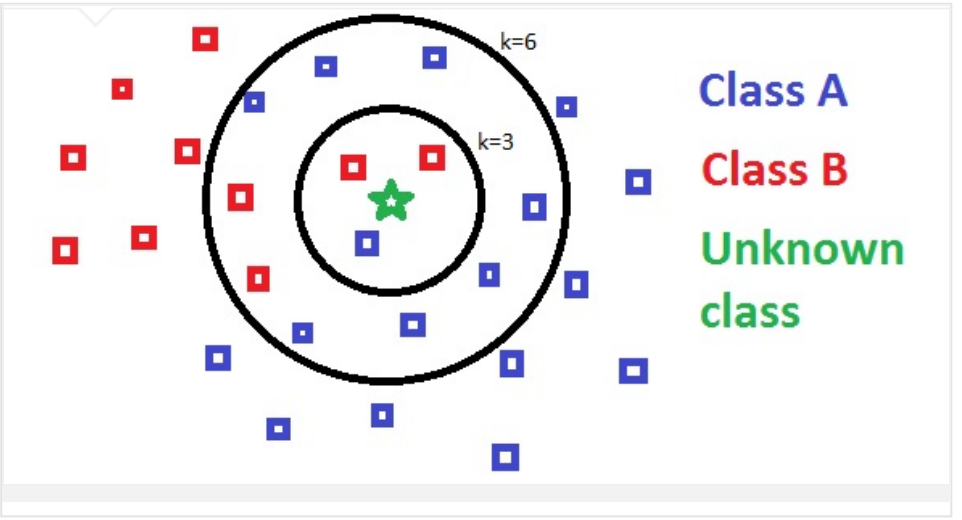

Fig 10 Using k-NN to classifiy the unknown point (star). If k = 3 is taken, then the point is classified as B, however, with k = 6, the point is classified as A. Source: https://towardsdatascience.com/knn-using-scikit-learn-c6bed765be75.

Support Vector Machine

A support vector machine (SVM) excels at assigning decision boundaries which is equidistance from the data clusters. For this purpose, the SVM uses landmark points and a kernel function that gives the similarity between the landmarks and the data point. For example, if the landmark points are labelled l_1, l_2 and l_3, then a gaussian kernel can be used:

$$f_i(x) = e^{-\frac{||x-l_i||^2}{2\sigma^2}} \tag{21}$$

Where number of landmark points and σ are hyperparameters. The f_i's constitute the new features for the data which will be used. These feature vectors can be used in a sigmoid model with,

$$z(\theta, f) = \sum \theta^{(i)} f_i + b \tag{22}$$

The model can then be trained by optimizing the following cost function:

$$J(\theta, f) = -y * c_1(z) - (1 - y) * c_0(z) \tag{23 a}$$

$$c_1(x) = \begin{cases} 0 & \text{if x} > 1 \\ \text{x-1} & \text{if x} < 1 \end{cases} \tag{23 b}$$

$$c_0(x) = \begin{cases} 0 & \text{if x} < \text{-1} \\ \text{x+1} & \text{if x} > \text{-1} \end{cases} \tag{23 c}$$

The boundary cut-off of 1 and -1 makes it so that the decision boundary is placed well.

1.4.4 Clustering

Clustering is useful when the input data has to be segragated but it has no predefined labels. Instead of assigning labels, the task of the program is to simpy group the data that are similar.

k Means Clustering

The program devides the data into 'k' number of clusters, by minimizing the mean of each cluster. First, randomly k points are selected from the dataset. These are called the cluster centroids. Then each data point is assigned to the nearest cluster centraoid. Once this segregation is done, the mean of the newly formed clusters is calculated and cluster means are assigned as the new cluster centroid. This process is repeated until convergence, i.e., till there is no change in the position of all the centroids.

1.5 Data Preparation

The program can not always be exposed to the raw data directly. For example, if the range of one of the features is [0, 1] while another in [-2000, 2000], this can cause inconsistency in training the parameters. Thus this data needs to be scaled down. While some preparation steps are crucial for training the model, other help in a better and faster training.

1.5.1 Preprocessing

One Hot Encoding

Data for multiclass classification needs to be one hot encoded before the program can make sense of the data. This means encoding the labels of data in a vector of size given by the number of labels where a 1 denotes that the data belong to that class, and all other entries of the vector are 0. So if there are 5 classes and a data belong to the 3rd class, the encoded label for this class will be (0, 0, 1, 0, 0). This can then directly be used for multi-class classification by modifying the cost function as:

$$J(\theta, x) = \sum_i (-y_i * log(h_\theta(x)) - (1 - y_i) * log(1 - h_\theta(x))) \quad (24)$$

Where i is the index in the vector.

Data Cleaning

There can be some inconsistency in data resulting from human error during data collection. So such erroneous data has to be manually checked and removed before feeding it to the program.

Normalization

Normalization scales down features with large value to a range of [0, 1]. This can be done either by deviding each value by the total sum of all the samples or by subtracting each value from the mean and deviding by the total range, i.e.,:

$$x_n^{(i)} = \frac{x_n^{(i)} - \bar{x_n}}{max(x_n) - min(x_n)} \quad (25)$$

Where i is the sample number and n is the feature number.

Standardization

Standardization scales the values of the feature to a distribution with mean 0 and standard deviation 1. It applies the following transformation to the data:

$$x_n^{(i)} = \frac{x_n^{(i)} - \bar{x_n}}{\sigma_{x_n}} \quad (26)$$

1.5.2 Transformation

The transformation is done seperately for each set of data. However, the transformation parameters are calculated only for the training set and the same is applied to all sets to avoid data leakage.

Transforming Test Set

The values that were used for transforming the training set must be the same and used as is when testing the algorithm. So the mean and standard deviation of the trainign set is also used for any future data points.

Cross Validation Transformation

The transformation is calculated only on the training set, so in each loop of the cross validation, a new transformation is calculated based on training set and then is applied on the cross validation set. The final transformation parameters are taken from the best performing iteration.

Inverse Transformation

To make sense of the final output of the model, inverse transformation has to be appied to it. It returns the original data back from the transformed data. Thus, the transformation paramaters are always saved.

1.5.3 Slice Processing

Samplewise Processing

Each sample is transformed independent of other samples. For example, in image processing, each image is processed independently of all other images.

Featurewise Processing

A single feature for all the samples is transformed at once. For example, as in standardization.

Elementwise Processing

Each element is independently transformed. For example, deviding every value by some fixed number.

1.5.4 Dimensionality

It might be tempting to use a lot of features to describe the data in a hope of getting better results. But, higher number of features imply higher dimensionality which can slow the algorithms and produce poor results as the number of data points required in higher dimensions are also high. This is called the curse of dimensionality.

Feature Selection

There can be many redundant features resulting from data collected from different sources. So, the dimensionality of the data can be decreased by manually eliminating some features which might not be of importance. For example, different features for the same data collected in different units.

Principal Componenet Analysis (PCA)

PCA can decrease the dimensionality of the data by reducing the features wih high correlation. It calculates the correlation matrix as,

$$U = \frac{1}{m} \sum_{i=1}^{m} \left(x^{(i)}\right)\left(x^{(i)}\right)^T \tag{27}$$

Where m is the number of samples. Then it reduces the features by selecting the first k eigen vectors of U and multiplying with the sample vectors:

$$z^{(i)} = U_k^T x^{(i)} \tag{28}$$

Where U_k is a matrix with its colums being the eigen vectors. So, z is the new set of data that has only k features.

1.6 Neural Networks

Neural Network is a computer system modelled on human brain and nervous system. Each neuron is represented by a node and the data flows through the connection between these nodes. Each connection has a weight and each node has a bias, and all such weights form the parameters for the the entire network. The weights are initialzed randomly to ensure each neuron gets specialized in a different task, this breaks the symmetry. Each layer of the network can consist of multiple neurons. The number of features constitute the neurons in the input layer.

1.6.1 Feed Forward Network

The program uses the pre-initialized weigths to calcutate the output of the neural network by calculating each layer's out put one after other. The data flows from one neuron to another in a forward direction (i.e. input to output). Each neuron layer can be represented as a weight matrix multipied to the previous layer output, i.e.,

$$a^{(l)} = Wa^{(l-1)} + b \tag{29}$$

Where $a^{(l-1)}$ is the vector output from previuous layer, W is a matrix of shape (number of nodes in layer (l-1), number of nodes in layer l) and b are biases for each node in layer l.

Then each layer can simply be called a function that acts on a vector from previous layer to give a new vector. For example, for a three layer structure, if each layer is denoted by a function, f, g, and h, where f and g are hidden layers and h the output, then,

$$h(x) = g(f(x)) \tag{30}$$

This equation represents a feed forward operation where x is the input vector.

Network Error

After a feed forward step, the performance of the network can be estimated. This is done by calculating the error between the final output of the model and the ground truth. The program will train by changing weights and biases to minimize the network error. Different cost functions can be used to calculate the network error based on the atatement of problem. For

example, the cross entropy can be used to calculate the network error in case of multi-class classification.

1.6.2 Activation Function

If each layer is denoted simply by Eq (25), then the final output of the network is as good as a single layered network. For example,

Thus an activation function is added to each layer which acts on the output of the layer before it is fed to the next layer. Activation function adds non-linearity to the output of a neuron layer, thus increasing the representational power of our model. The sum of the previous layer and the weights is fed into the activation function to get the final output of the layer. Thus, the layer operation can now be represented as:

$$g(x) = f(Wx + b) \tag{31}$$

Where f is the activation function. Following are sum examples of an activation function given that $z(x) = Wx + b$.

Linear

Similar to applying no activation function,

$$f(z) = z \tag{32}$$

Rectified Linear Unit (ReLU)

$$ReLU(z) = \begin{cases} \mathrm{z} & \text{if } \mathrm{z} > 0 \\ 0 & \text{if } \mathrm{z} < 0 \end{cases} \tag{33}$$

Step Function

$$\theta(z) = \begin{cases} 1 & \text{if z} > 0 \\ 0 & \text{if z} < 0 \end{cases} \tag{34}$$

Sigmoid Activation

$$g(z) = \frac{1}{1+e^{-z}} \tag{35}$$

Softmax

Normalizes and returns the output of a layer as a probability of each node,

$$Softmax(z) = \frac{e^{z_i}}{\sum_i e^{z_i}} \tag{36}$$

Where the denominator is summed over all the nodes in the layer.

$$g_2(x) = W_2 x + b_2 \tag{37 a}$$

$$\begin{aligned} h(x) &= g_2(g_1(x)) \\ &= W_2(W_1 x + b_1) + b_2 \\ &= (W_2 W_1)x + (W_2 b_1 + b_2) \\ &:= W_3 x + b_3 \end{aligned} \tag{37 b}$$

1.6.3 Backpropogation

After calculating the output via feed forward step and estimating the error, the gradient of the error with respect to the weights needs to be calculated to optimize the error.
Backpropogation is one such algorithm that speeds up the gradient calculation. Applying chain rule to the feed forward formula, we get,

$$h'(x) = g'(f(x))f'(x) \tag{38}$$

The loss function is a function of the final layer, i.e., h(x). Therefore, if we know how the network error changes with respect to the weights of the final layer, we can work our way backward to calculate gradient of each layer's weights. This can be achieved by the following steps:

$$\delta^{(l)} = \frac{\partial J}{\partial z^{(l)}} = \frac{\partial J}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial z^{(l)}} = \delta^{(l)} f'(z^{(l)}) \tag{39 a}$$

$$\Delta^{(l)} = \frac{\partial J}{\partial W_l} = \sum_i \frac{\partial J}{\partial z_i^{(l+1)}} \frac{\partial z_i^{(l+1)}}{\partial W_l}$$
$$\Delta^{(l)} = \delta^{(l+1)} \left(z^{(l)}\right)^T \tag{39 b}$$

Where l denotes the layer, δ is the gradient with respect to layer and Δ is the gradient with respect to the weights, and f is the activation function of (l+1)th layer.

1.6.4 Optimization

After calculating the gradients, the final step of the learning loop is to apply the gradients to optimize the error. Gradient descent can be used for this purpose which will give a minima of the function. The problem is that it does not guarentee a convergence to the global minima and can get stuck at a local minima. To tackle this issue, many imrovements over the native algorithm are used.

Adaptive Learning Rate

As said earlier, a bad choice of learning rate can resut in poor learning. Ideally, we would want to start with a big learning rate so we move fast towrds the minima and jump out of any small pits, but as the training continues and we near the global minima, a smaller learning rate is desirable so we do not overshoot. So the learning rate can be adjusted depending on the number of epochs. Generally, the learning rate will decrease with increase in epoch, either by deviding the epoch number or by setting a constant decay factor. So, the learning rate η in the n^{th} eposh can be calculated by either of:

$$\eta_n = \frac{\eta_0}{n} \tag{40 a}$$

$$\eta_n = \eta_0 e^{-n\beta} \tag{40 b}$$

Stochastic Gradient Descent (SGD)

The error value calculation, gradient calculation and weight updates are all done done one sample at a time. This adds a lot of randomness in the training process compared to doing all at once which smoothens out the process. This randomness can help give the error value a push out when stuck in a local minima, however this also means that the final value will always oscillate around the otimal value rather than converging to it.

Batch Gradient Descent

Instead of all sample at once or one sample at a time, the training can also be done in mini-batches of sample. Small batches are made out of the initial set and fed into the network for each update step. This captures the randomness from SGD while also keeping the process fairly smooth, thus helping the process to converge.

Gradient Descent with Momentum

Information from the descent from previous step can also be used to update the variable in current step. This hepls the function move faster on plateaus and also provides an extra push to get out of stucking in local minimas. The momentum idea is similar to a ball rolling down a hill. So the update step now looks like,

$$W_i = W_i - \alpha \frac{\partial J}{\partial W_i} - \gamma m \tag{41}$$

Where m, momentum, is the grdient of the previous step and γ is a hyperparameter.

Nesterov Momentum

Instead of looking at the past position, we can also use the gradient from the position that the function would goto with the current momentum, to update the current value. This can be written as,

$$W_i = W_i - \alpha \frac{\partial J}{\partial (W_i - \gamma m)} \tag{42}$$

Adagrad

The prefix 'Ada' stands for adaptive learning rate and 'grad' for gradient. The program uses the values of the current gradients to decay the learning rate. The current learning rate is the current gradient divided by the running sum of square of gradients after each step.

Adadelta

Adadelta is similar to Adagrad, but instead of summing up the squared gradients, adadelta keeps a decaying sum of the gradients to update the learning rate.

RMSProp

The rms values of the gradient is added to the running sum which is used to determine the current learning rate.

Adam

Two lists are kept to derive the learning rate, one is the squared sum and the other is the simpe sum of the gradients. This way, the information about the sign of the gradients is also preserved.

1.7 Deep Learning

Deep Learning refers to an approach to ML where multiple specialized layers of computations, such as neural networks, are stacked over each other to make a ‘deep’ architechture. Each addition layer makes the representationl power of the model better. However, too many layers can also result in overfitting of the data. Diagramatically, the flow of data through the layers can be shown with arrows.

1.7.1 Tensors

Tensors in the context of ML are n-dimensional arrays. For example, for a one dimensional feature vector and m number of samples, the entire data can be represented as a m by n matrix where n is the number of feature. This then simply makes each operation of a deep network as a simple matrix operation. Thus, depending on the dimension of the input features a tensor of one additional dimension (number of samples) is fed into the network.

Input Layer

This forms the first layer of the network. The input features are fed to the input layer. So naturally, the size of the input layer is same as the size of input. The tensor then flows forward from this layer.

Output Layer

The layer than generates the final output of the deep network. This is the last layer of the network .

Hidden Layers

All the layers in between the input and output are hidden. They constitute a black box as the user only gives the input and can only see the output.

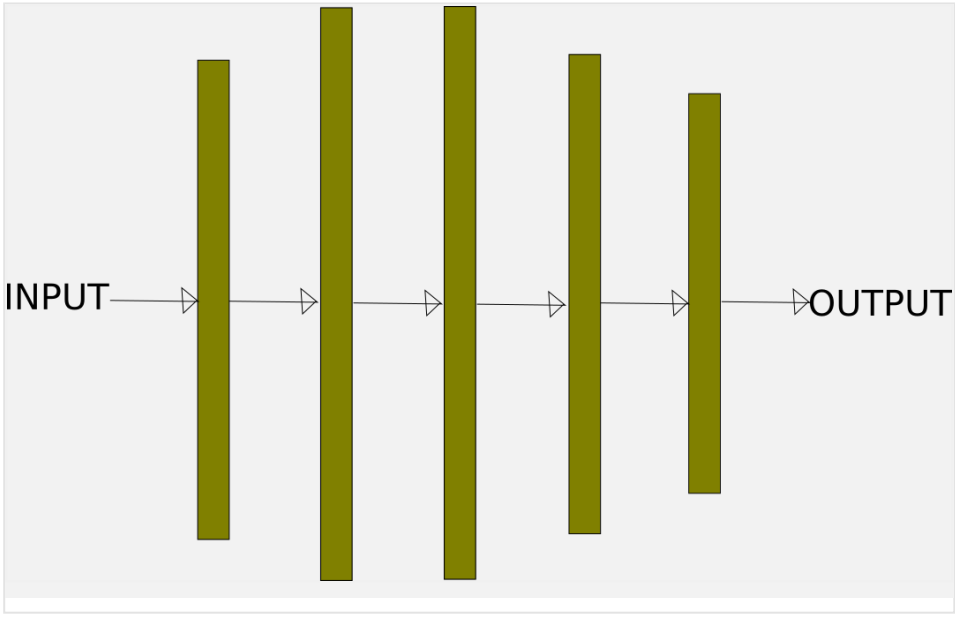

Fig 11 A Deep Network with 5 hidden layers. Each layer performs an independent computation. Arrows show the flow of Tensor.

1.7.2 Core Layers

Fully Connected Layer

Every node in a fully connected layer is connected to every node in the previous layer. This is the simplest layer whose example was taken in the neural networks. A network contituted majorly by fully connected layers is often called a Dense Neural Network

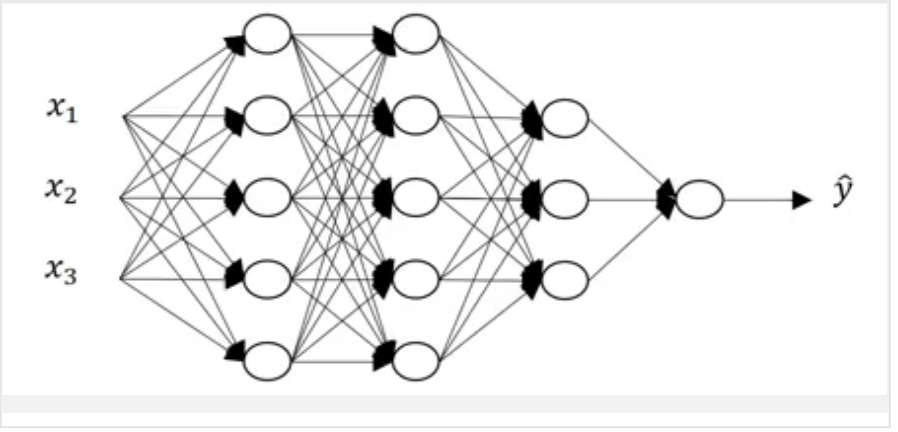

Fig 12 Deep Network consisting only of Fully Connected Layers. Source: https://www.analyticsvidhya.com/blog/2018/10/introduction-neural-networks-deep-learning/

Convolution Layer

Convolution exploits the saptial correlation in data to improve the performance of a network. Instead of connecting the entire layer, a convolution layer processes small collections of nodes at a time. It is somewhat similar to scanning an image with a filter to exract the features from that image. A network with mostly convolution layers is called a Convolutional Neural Network (CNN). CNN's are very good at extracting features from images for analysis.

Recurrent layer

In some cases, the sequence in which the data is coming also matters. For example, while translating a sentence, the order of the words is important. So network needs to have some sort of memory of the previous data. For this purpose, in a recurrent cell, the output of the network is fed back into the network, so the model will have a memory of the previous data. This layer is used for data with temporal correlation.

e.g., A simple recurrent cell without an activation function can be expressed as,

$$h_t = x_t W_x + h_{t-1} W_h + b \tag{43}$$

Where t is the time stamp if the sample, i.e., the order in which it arrived.

1.7.3 Utility Layers

Dropout

The dropout layer is active only during the training time. It helps to prevent over-fitting due to specialization of neurons. In each run of the network, it randomly disconnects a given percentage of nodes from the previous laye, so that no nodes overspecializes.

Batchnorm

Batchnorm or Batch Normalization is also used only during the trainig to prevent the layer weigths from exploding (or becoming too large). It is similar to iniatially normalizing the data, but instead the previous layer, which might not be an input layer, gets normalized with respect to the entire batch that was fed. This ensure that the value of nodes during training never becomes too large, thus regularizing the weights.

Pooling Layer

Moving forwad in the network, decreasing the size of the tensor might help increase the speed. Thus, the pooling layer decreases the number of outputs from the previous layer either by using the mean (average pooling) or maximum (max pooling) of a group of values at a time. Pooling layers are used generally after Convolution layers to reduce the image size.

Noise Layer

Adds random noise to the output of a layer. This gain, is helpful during the training as it adds randomness to gradient descent, helping it avoid local minimas.

Reshaping Layer

Reshapes the output of previous layer. It is useful when the next layer expects a input in different shape that the output of previous layer.

Cropping Layer

Crops the output of the previous layer, i.e., it reduces the size of tensor by removing fixed number of elements from the ends in each dimension of the tensor, except in which the samples are stored.

Zero-Padding Layer

Takes the output from the previous layer and puts zeros all around it. So, padding increases the dimension of the tensor, this can be useful for a convolution layer.

Flatten

Flattening is same as reshaping each sample into a single dimension. It is useful to use flattening between a convolution layer and a dense layer.

1.7.4 Model Explainability

With multiple layers in the model, each layer might be getting trained to perform a different set of tasks. However the final output does not conatin much information about how the network analysed the data. To examine what each layer is doing, the weights of that layer ne checked. Then based on the weight, what value of the input 'activates' a particular node in the layer can be calculated to give an idea on what aspect of the input is that layer working on.

The explainability of a model tells how good each layer of a deep network can be examined and their outputs be explained. In all cases it is not possible to explain the working of a network. Such hidden layers constitute a black box.

1.8 Convolutional Neural Networks

Convolution Networks are used to extract features from images. Imgaes can be simpe 2D or even 3D.

1.8.1 Image Features

In the computer, images are stored as a matrix of numbers, or pixels. In this two dimensional grid, the value of each cell denotes the weight of a particular color to draw the image on screen. This is also how a program will percive the image. One way of handling and processing images could be to just flatten out the entire image, treat this as the new feature vector and feed it to a Dense Neural Network. But, this would be slow and contain absurd

amount of parameters making it computationally expensive. But a thing about images is that the nearby pixels are more strongly correlated that pixels far apart. So instead of connecting all the pixels to the next layer, convolution provides a better alternative to look at the images and extract features for further analysis.

Image Depth

The depth of a image is the number of features each pixel in the image has. The depth constitutes the number of **channels**. For example, for a colored input image coded in RGB, each pixel will have three values of Red, Green and Blue weights. Thus, the number of channel in this image would be three. While it is easier to mke sense of the channels in an input image, for an intermediate output of a convolution, the meaning of a channel might not be so trivial, and it simply constitutes another layer that defines the image. The depth of the image is different from an additional dimension of the image, as there will be no convolution along the depth of the image.

Image Tensor

Since all the data fed to a network has to be a tensor, the images are also converted to a tensor first. Depending on the convention, the structure of the image can either be channel first or channel last. Whatever be the convention, it has to be followed in all layers of the network. The first dimension of the tensor is always the number of samples. In a channels first structure, the next dimensions will be the number of chennels followed by the coordinates of the image (2 for 2D image, 3 for 3D, so on and so forth), while in the channels last structure, the coordinates are placed first followd by the number of channels as the last dimension. Even for a single channel image (e.g. grayscale), the dimension for the channels need to be specified with a value 1, to maintain consistency.

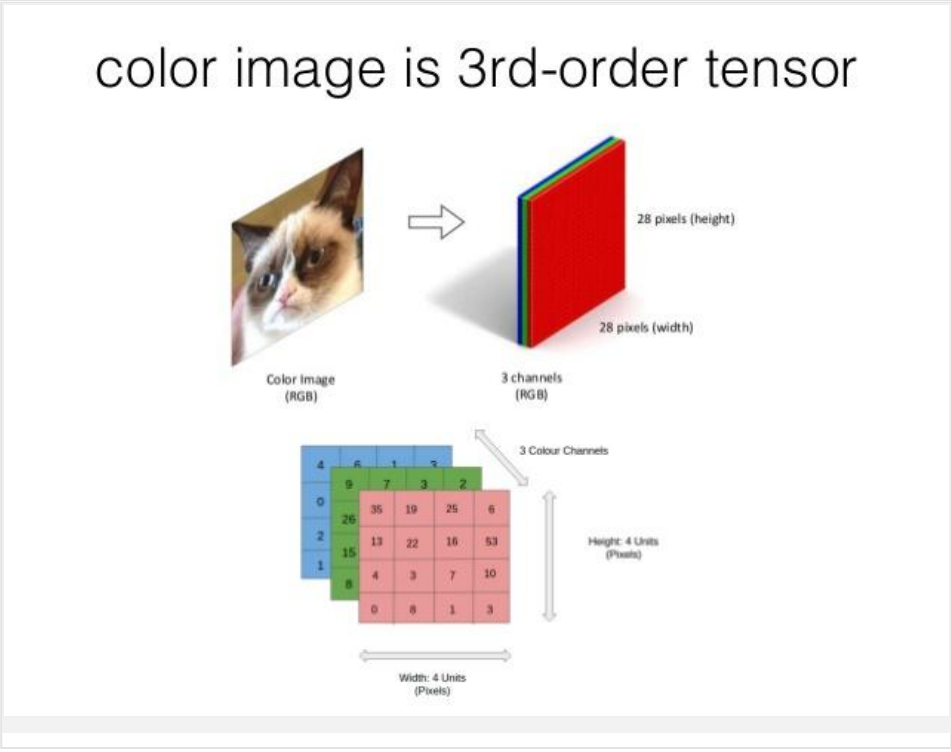

Fig 13 Example of an image represented as a Tensor. Source: https://www.slideshare.net/BertonEarnshaw/a-brief-survey-of-tensors

1.8.2 Convolution

Now that the image tensor has been provided to the convolution layer, we need to convolve over it. Convolution is done using various filters of fixed sizes.

Filters

A filter or a kernel is a matrix with weights which scans over a given image generating a new image. For example, if the size of a filter is 3 X 3 for a 2D image, then the filter is basically a matrix with numbers of size (3, 3). The region of image from which this filter is reading the value is called **local receptive field.**

So the (3, 3) matrix is placed over some region in the image, the the value of the pixels below the filter is multiplied by the respective weights and all are added together. This gives a single value using which the pixel of the output image is contructed at the position specified by the

center of the local receptive field, or the **focus pixel**. The center of the filter is called the **anchor** that moves over the focus pixels.

If the topmost corner of the image is denoted by (0 ,0), then the 3 X 3 filter will initially be anchored at (1, 1) pixel, because there is nothing beyond the boundaries. So the output of a convolution of a 28 X 28 image by a filter of size 3 X 3 will be of size 26 X 26. To avoid reduction in size, zero padding can be added to the image. Then the 28 X 28 image will first be padded to 30 X 30, then after convolution the size of output will be 28 X 28.

Convolution with Multiple Channels

If the input image has multiple channels, the filter will be anchored at the same position in each channel and the output value of each channel will be calculated. The final value of the output pixel will be the sum of all the channels. So if then number of channels in input image is 5, after a convolution by a single filter, the channel in output will be one.

Different number of filter can be used in a single convolution step. Each filter scans the image, behaving as it is looking for a particular feature in the image. For example, a certain filter can specialize in detecting parallel lines during training, or other can be looking for gradients. Such feature detectors can be estimated by investigating the weights of the filter. The output of each filter can then be stacked to create different channels in the output image. So if the number of channels in put image is 5, after a convolution by 12 filters, the channel in the output will be 12.

Stride

The stride defines the movement of the filter over the image. For a 5 X 5 filter, it will initially be anchored at (2, 2) because this is the first valid cell so the filter fits inside the boundary of image. After calculating the output from this local receptive field, the filter has to move to a new anchor position. With simple striding, the filter will move next to (2, 3) then (2, 4) till (2, n - 2), where n X n is the size of the image. After the first row row done, the filter will move on to the next, i.e., (3, 2) and continue this till it reaches (n-2, n-2) which will be the last pixel in the output image.

The stride is the amount by which the anchor shifts. In the above example, the stride of the filter was (1, 1). Changing the stride of the filters will change the size of the out put image. For eaxmple, with a stride of (2, 2), the size of the output image is halved. This is called down sampling which can also be done using pooling layers. The stride size can be asymmetrycal as well, in which case the output will also be asymmetrical.

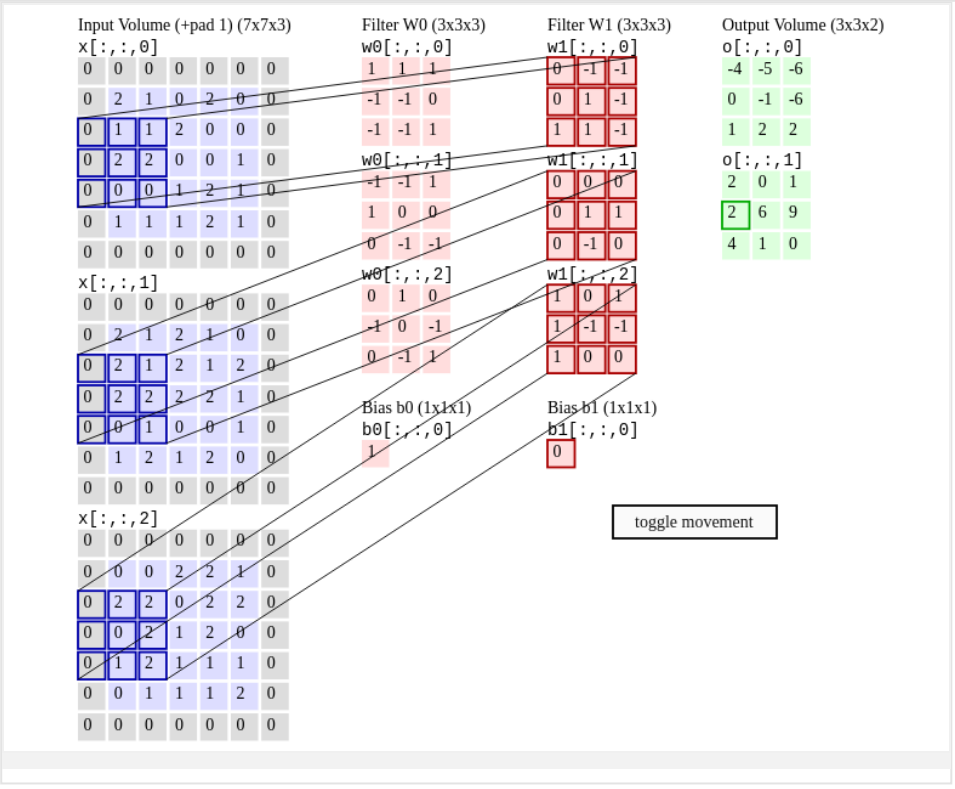

Fig 14 Convolving with 2 filters on an input image with 3 channels. Output volume will have 2 channels.Highlighted boxes show how a single cell in the output was generated. Source: http://cs231n.github.io/convolutional-networks/

Weight Initialization

The weights of each filter are initialized randomly so that there is no symmetery between two filters, that is from the start itself they are looking for different features. The initialization can also be randomized based on the information of the number of inputs to the layer. This is called **fan-in**.

Image Classifier

The task of a classifier is to put labels on an image, for example, which digit is in the image, or is there a person in the image or not. There are many basic datasets like the MNIST and CIFAR on which a network can be trained. For this purpose a Convolutional Network can be used for

feature extraction (the final output of multiple convolutions flattened out), followed by dense networks to classify the features into labels. An example of such a network is the VGG-16.

Adversary

It is possible to fool a trained program by using slightly perturbated images. Small perturbations, called adversarial perturbations are calculated using the filter weights and are added to the image, so the image remains unchanged but the CNN fails in recognizing such perturbed image. If an adversarial perturbation works for every image given to a classifier it is called an **universal perturbation**.

1.8.3 Upsampling

While downsampling reduces the size of tensor, there might be cases where increasing the size instead required. This is done by upsampling convolution layers. An example for a model that implements the upsampling layer is the U-Net.

Transposed Convolution

The idea here is that if we use a single filter of size 3 X 3 with padding, then the output will be of the same size. However, if we transpose the filters, we can use 3 filters of size 1 X 3 and the output of each filter giving a different pixel of the output image, thus increasing its size.

Dialated Convolution

The image before convolution is dialated with zeros, i.e., zeros are added in between each row and column of the image, then simple convolution can be done over the image. For example, after padding and dialating a 28 X 28 image for a 3 X 3 filter, its size will be 58 X 58, and after convolution the final output size will be 56 X 56

1.8.4 Autoencoders

Autoencoders are used to store data in a smaller space, i.e., they can compress data.

Encoder-Decoder Pair

An encoder and a decoder forms a bottleneck kind of a model. The encoder reduces the size of the input and the decoder converts the reduced input back to the original. Thus, the converted output should be same as the original output, this forma a basis for trining the model. The encoder can consist of downsampling steps while the decoder has upsampling steps. After

training is complete, the decoder part can be removed and the encoder is then used to compress the data.

Latent Variables

After the encoding step, the variables that store the compressed output are called latent variables. Latent variables are also fed to the decoder to decode. For example, if a 28 X 28 image is passed through multiple downsampling steps to get to 2 X 2 and then flattened, the image can be stored in just 4 latent variables. Number of latent variables can decide the representation power of the encoder.

Variational Autoencoder (VAE)

Variational Autoencoders are used to generate new images which are similar to the input. VAE also consist of an encoder - decoder pair. After the encoder gives the latent variables, small perturbations can be added to these before feeding to the decoder. However if the network is traied without any perturbations, these might give non-sense result.

Thus, a **reparametrization trick** is used to train VAEs. The random noise that is added to the encoded data is taken from a gaussian distributin with mean μ and variance σ. The mean and variance for each latent variable is a different parameter that the model will learn during the training. So, the model learns to create similar images if random noises are added to encoded data.

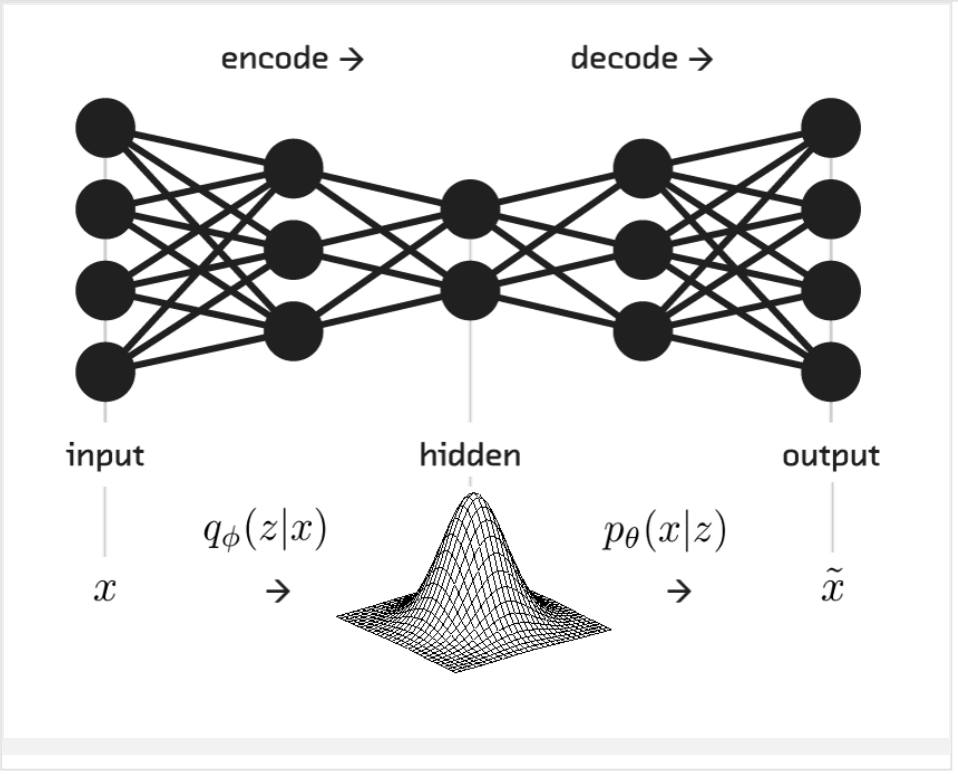

Fig 15 Random numbers from a normal distribution are added to the latent variables (middle layer) in an encoder(q)-decoder(p) pair to produce similar outputs. Source: https://towardsdatascience.com/what-the-heck-are-vae-gans-17b86023588a

1.8.5 Computer Vision

Computer Vision is a field which deals with real time object detection and localization.

Object Detection

In a given image, the task of a detector is to say whether the image contains a said object or not. For this purpose, the identifier is trained on tigthly cropped images of the object of interest. All the images are of the same size during training. Images are skewed, rotated and scled by random amount during training to create new images that the program can train on, this is called **augmentation**.

After the model is trained, when it is exposed to a real image, a sliding window is used to crop out small portions of the image one at a time to the size that the model expects, and runs the

classifier on the cropped image. Then if in any portion, the object of interest if found, the model will detect it. For this puprpose, window is slid over the image many times, each time with a different scaling of the actual image.

Object Localization

Image localization means, that over saying whether an object is present in the image or not, also determing where in the image is it found. This is also called as a bounding box problem, as a box that bounds the object has to be drawn. One is to do this is by using bouding box regression, which trains the model's output by comparing the intersection over union with the ground truth, thus ensuring a tight bounding box.

Image Segmentation

Image segmentation means drawing borders around the objects in the image to seperate them from the entire image. Mask-RCNN is one such algorithm trained for image segmentation.

YOLO Implementation

You Only Look Once (YOLO) is a program that does object identification and localization in a single step. It does not need sliding windows and is thus much faster in real time applications. The idea here is that the final output of the CNN is reshaped to a 3D array. The first and second dimension of the array give the segments in which the original image is divided. For example, a 200 X 200 can be divided into 5 X 5 blocks, each block having a size of 40 X 40. The third dimension gives the details of the object whose center lies in the block. The first five coordinates of the last dimension, define a bounding box, followed by a one hot encoding for the labels. Out of the first five coordinates, the first one is the confidence that the program has that an object is present. The next two are the x and y coordinates of the center of the box, with respect to the block, followed by the height and width of the bounding box normalized to the size of the image. Instead of just one box per block, even more can be used and some anchor boxes and be predefined. Each anchorbox will have a different shape, thus each of the box in output will also specialize with a different shape.

For training the model, the following cost function is used:

$$(\bar{C} - C)^2 + \lambda_{obj}((\bar{x} - x)^2 + (\bar{y} - y)^2 + (\sqrt{\bar{w}} - \sqrt{w})^2 + (\sqrt{\bar{h}} - \sqrt{h})^2) + \sum_{i=1}^{number\ of\ classes} (\bar{z}_i - z_i)^2 \quad (44\ a)$$

$$\lambda_{noobj}(\bar{C} - C)^2 \qquad (44\ b)$$

Where values with bar are the ground truth. When the object is present in the grid and a box is responsible for it, then eq (a) is used otherwise equation (b) is used. The final value is the mean over all the values of the grid.

Fig 1 shows a simple model implementing the YOLO training process. The data was generated by using the MNIST dataset. Random number were selected and scaled to sizes between 30 X 30 and 60 X 60, keeping the aspect ratio. Then these images were placed on a canvas of size 200 X 200. To train the network, the ground truth was a 5 X 5 X 15 dimensional tensor, i.e., the number of grids was 25 and one box in each grid with a 10 class classification.

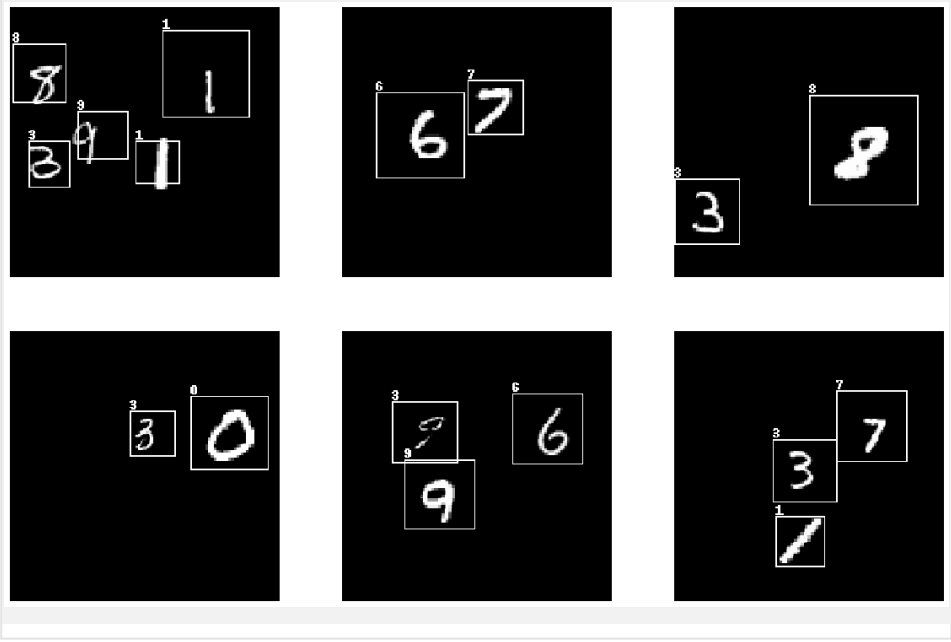

Fig 16 Ouputs from a simple implementation of YOLO run on MNIST dataset.

1.9 Recurrent Neural Networks

In a recurrent network, the data does not flow only forward, but is also feeded back to a layer. Thus it forms a loop which helps the network remember what data came before the one it is processing. The input tensor conatins data segrated into different time stams. The number number samples that constitute one time window must be the same. So now, the first

dimension of the tensor is the number of sample and the next is different time stamps followed by the features for each time stamp.

1.9.1 Recurrent Cell

A basic recurrent cell can be defined as:

$$h_t = f(Wh_{t-1} + b) \tag{45}$$

Where f is any activation function and t is the time stamp of the data. This is similar to a dense layer but insted of taking input from the previous layer, it is taking the input that the same layer produced for the previous sample.

State

The state of a recurrent cell is how the cell will remember the data. Equation (37) is what defines the state of a naive recurrent cell. With each new data from one set of time window, the state gets updated. But before the state gets updated, it is copied to be used with the input data to generate the output for the cell. Then this output is used to get the new state of the cell which will be used for the next data.

Output of Recurrent Cell

Each time step to the recurrent cell generates an output. Therefore the number of inputs and outputs are the same. However, not all the outputs might be necessary, so only the last output from the cell can be retained. If, there is another recurrent cell after the current cell, then all the outputs are fed to the following cell in the same sequence.

Bi-directional RNN

With a single Recurrent cell, it analyses the data in one direction only, and it has no idea of what if coming after the given input. If two cells are used, where first one analyses the data from front to back and the second one from back to front, and the final output is the sum of both these cells, then the network is called a Bi-directional RNN. Such networks are useful for translation where the sructure of sentence is crucial.

1.9.2 Backprop Through Time

To calculate the gradient of the loss with respect to the weights of the recurrent cell, the recurrent cell is first flattened to look similar to a dense neural network. The connection between each layer is the state that gets to the next layer. The difference being, in this case each layer is also getting another input which is the time seperated data. With this in hand, backpropogation can be performed through this network, only difference being the weights in all the layers of a single recurrent cell will remain the same, and are therefor summed over. This is called Back propogation through time (BPTT).

There is a mojor issue which arises during BPTT in a naive recurrent cell. The activation function used generally in the cell is hyperbolic tan, and with multiple applications of tanh, the gradients can explode, i.e., they take on either extremely large value or extremely small value. Thus, gradient descent becomes difficult. another issue with the recurrent cell is that it can not remember very long time sequences since it has a single loop. For this purpose Long Short Term Memory (LSTM) cells are used over recurrent cells.

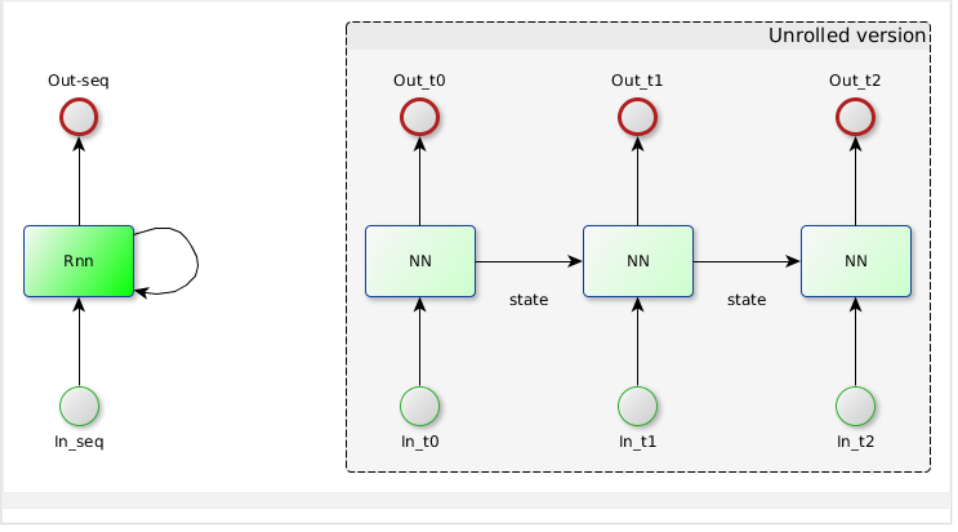

Fig 17 Left: recurrence happening in a single step. Right: unrolling the rnn cell to apply backpropogation. Source: https://leonardoaraujosantos.gitbooks.io

1.9.3 LSTM Cell

A Long Short Term Memory cell is similar to different from a recurrent cell in it decides how much of the previous state it needs to forget and how much of the new information it needs to remember. Thus, it has a longer short term memory.

Gate

A gate is a logical operation that decides how much of the input information passes through. It can be number in the range [0, 1] which is multiplied to the input, where 0 means all the information is lost and 1 means all is kept. LSTM cells use the information from the inputs to decide the values on various gate.

As an inut LSTM Cell takes the next data from time sequence, the output from previous time step and the previou state of the cell. The previous output and next data is used to calculate the values for forget gate, remember gate and select gate. Neural networks are used to calculate these gate values. Then first the forget gate is applied to the previous state to forget some information, then the remember gate is applied to the current data to decide how much infrormation to remember. This information is then added to the current state. Finally the select gate is applied to the current state to determine how much of the current state has to be gievn as the output.

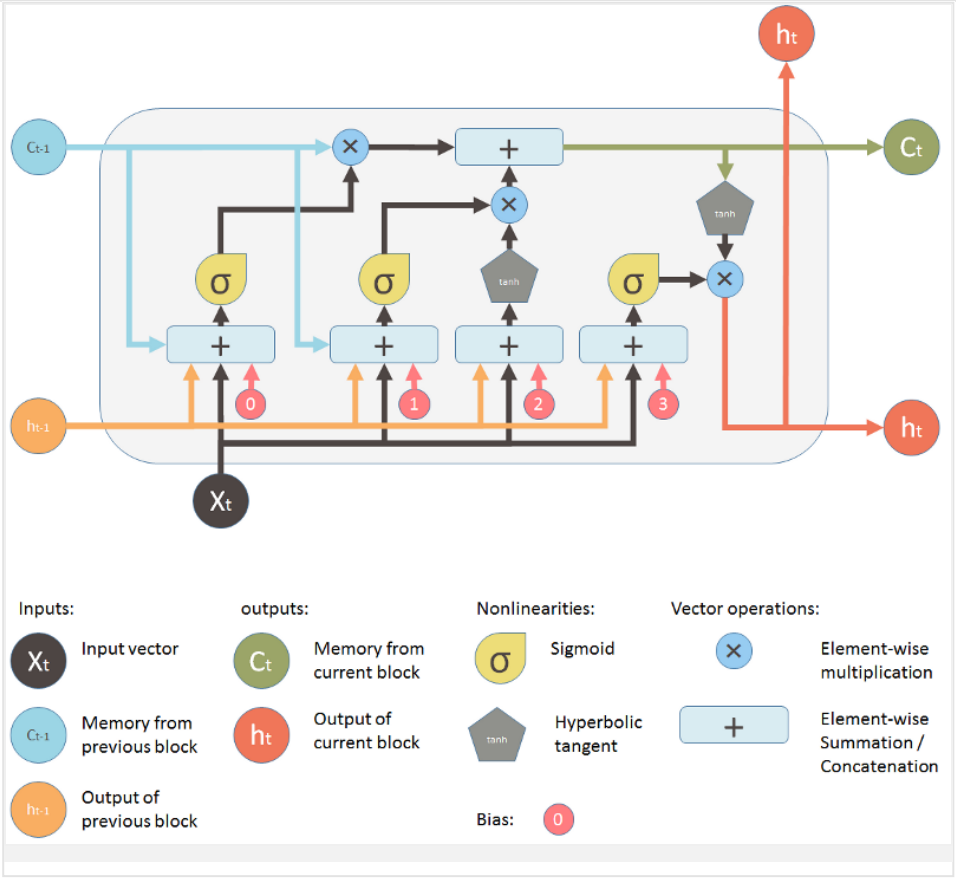

Fig 18 Scematic of a LSTM cell. Source: https://leonardoaraujosantos.gitbooks.io

2 3-D SIMULATION OF DETECTOR

In a particle collider, the detectors are lare grids of crystal each of which store an energy value due to physical processes by the incoming beam of particles. Computer Vision is used generally to detect objects in an image, but the data from a detector is very similar to an image in that it is also a matrix of numbers. The objctive of this project is to ascertain whether a Convolutional Neural Network can be trained to identify different particle beams from a detector data.

For this project, data from only a prototype of an ECal (Electromagnetic Calorimeter) was used. The analysis was done over different types of input data. The basic prototype had 11 X 11 crystals in the xy plane of $PbWO_4$ of dimension 1 cm X 1 cm X 2.5 cm. The number of

layers in z direction was variable and all particles were shot along the z-axis at the center of the structure.

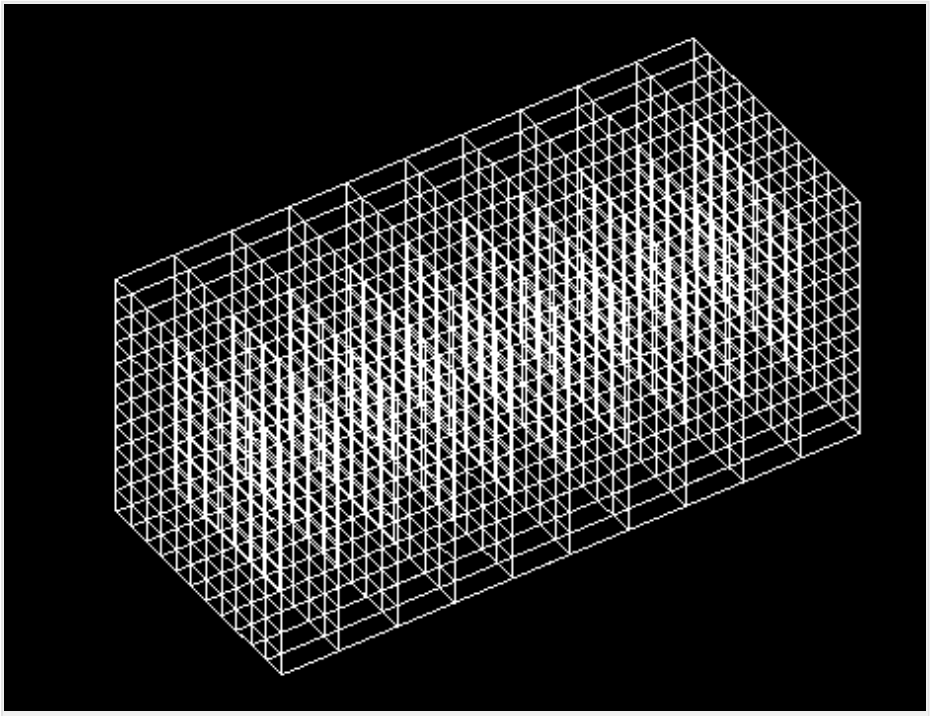

Fig 19 10 layered Detector Simulated using Geant4

The following configurations were studied:

1. ECal with 10 layers, No magnetic field, particles shot with energy 10 GeV starting from the surface of the ECal.
2. ECal with 10 layers, magnetic field of 4 Tesla, particles shot with energy 10 GeV starting from the surface of the ECal.
3. ECal with 10 layers, No magnetic field, particles shot with energy 10 GeV starting from 1 m before the surface of the ECal.
4. ECal with 40 layers, No magnetic field, particles shot with energy 10 GeV starting from 1 m before the surface of the ECal.

The data for following particles was analysed: e_+ (positron), γ (photon), κ_0 (kaon0), κ_+ (kaon+), μ_+ (muon), n_0 (neutron), π_0 (pion0), π_+ (pion+), p_+ (proton).

The Geant4 simulations were done by S. Dansana and the data generated from those simulations were used for this project.

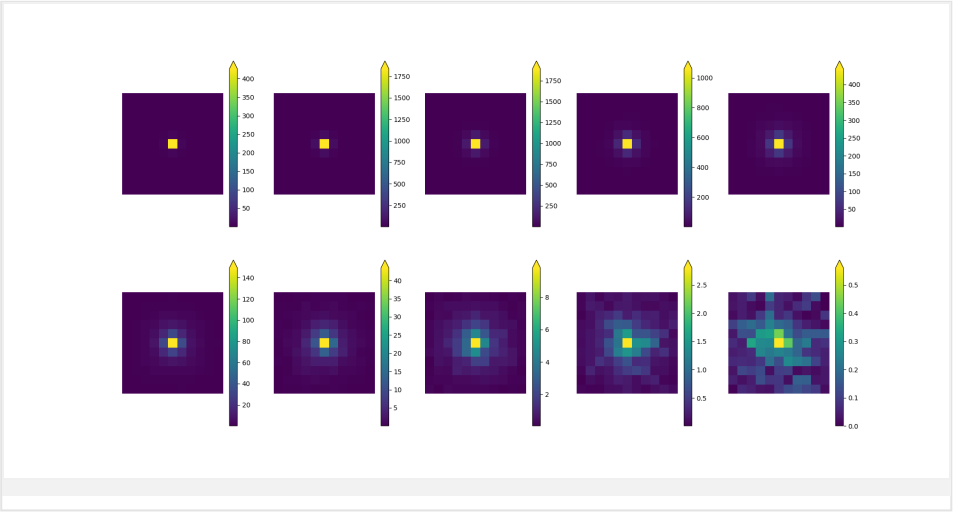

Fig 20 Positron beam averaged over 100 events for configuration 1. Energy values in MeV

3 ANALYSING THE DATA

For analysing the given data, Python libraries Numpy, Tensor Flow 2.0, Keras, MatpltLib and PyROOT were used. PyROOT was used to read data from a ROOT file into a numpy array so the analysis can be done on that.

3.1 Preprocessing

The data preprocessing was done using Tensor Flow to get a TF Tensor that can be feeded to the model. The initial data was three dimentional (z, x, y). Each data was first flattened to a single feature veactor (u), then each vector was standardized independently using the following formula:

$$u = \frac{u - \langle u \rangle}{\sigma'_u} \tag{46 a}$$

$$\sigma'_u = max[\sigma_u, \frac{1}{p}] \quad (46\ b)$$

Where mean and deviation are calculated over each sample and p is the number of pixels in one sample. The second equation was added to avoid division by 0.

The data was then again reshaped into a tensor of size (z, x, y, c), where c is the number of channels, which in this case was 1. All the data were concatenated along the first dimension, therefore the final size of the tensor was (n, z, x, y, c), where n is the number of samples.

The labels were converted to one-hot encodings so as to make it trainable for the model. Therefore the final output of the model has 9 values, one for each particle.

3.2 Model

Since the data is in three-dimension, 3D Convolution was used to analyse the data. Instead of 2D filters, 3D convolution has blocks that slide through the spatial region. Two different CNNs were used to study the data, here on referred to as ConvNet1 and ConvNet2. The models were coded and trained using the Keras API.

3.2.1 ConvNet1

The first model has two convolution layers followed by a fully connected layer. The size of the final layer is 9. The first convolution layer has 8 filters each of size 5 X 5 X 5 convolving with a stride of (2, 2, 2) and zero padding to maintain the size. Thus the output from this layer has a size (5, 6, 6, 16). The next convolution layer has 16 filters of size 3 X 3 X 3 convolving with a stride of (2, 2, 2) and zero padding. So the output has a size (3, 3, 3, 32). Both these layers have ReLU activation and are followed by a BatchNorm layer and the kernels were regularized using L2 regularization with a factor of 0.1. The convolution output is then flettened and followed by a fully connected layer of size 9 with softmax activation. There is a dropout layer before the final layer with a factor of 0.2.

The first three configurations were analysed using ConvNet1, however, for the forth configuration, ConvNet2 produced better results as this configuration as more layers.

3.2.2 ConvNet2

The second model has four convolution layers followed by a fully connected layer. The size of the final layer is 9. The first convolution layer has 32 filters each of size 5 X 5 X 5 convolving with a stride of (2, 1, 1) and zero padding to maintain the size. The next convolution layer has 64 filters of size 5 X 5 X 5 convolving with a stride of (2, 2, 2) and zero padding. Next two

layers have similar number of filters and strides as the previous two layers, but with a filter of of 5 X 3 X 3. All these layers have ReLU activation and are followed by a BatchNorm layer and the kernels were regularized using L2 regularization with a factor of 0.1. The convolution output is then flettened and followed by a fully connected layer of size 9 with softmax activation. There is a dropout layer before the final layer with a factor of 0.2.

3.3 Training

The models were trained using categorical cross entropy loss function. For training purpose, 1000 events of each particle were taken and then randomized. So a total of 9000 data. Out of these 900 were used for validation and the model was trained on the other 8100 data. The epochs were monitered during the training and the process was stopped when validation loss started to grow. Adam optimizer was used in all the cases with a learning rate of 0.001. Along with the loss, the accuracy was monitered during the training. The training was done on an Nvidia GeForce 940 MX GPU.

4 RESULTS AND CONCLUSION

4.1 Results

For each configuration, the model was tested on a different set of 1000 data for each particle. The number of times the model predicted a particular particle (i.e., the particle with highest value in the final output), was plotted for each particle.

4.1.1 Configuration 1

ConvNet1 was trained for configuration 1. The training was stopped at 20th epoch and it ended with a training accuracy of 48.68% and validation accuracy of 45.20%. The performance on the test data is shown in figure 21.

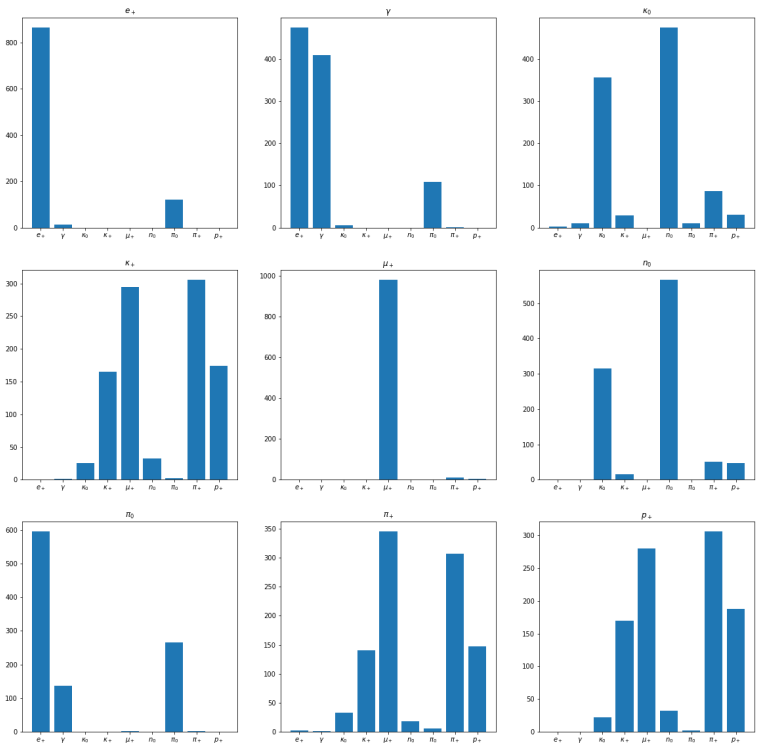

Fig 21 Predictions of ConvNet1 trained on Configuration 1. Test accuracy: 45.63%.

4.1.2 Configuration 2

ConvNet1 was trained for configuration 2. The training was stopped at 20th epoch and it ended with a training accuracy of 58.38% and validation accuracy of 46.88%. The performance on the test data is shown in figure 22.

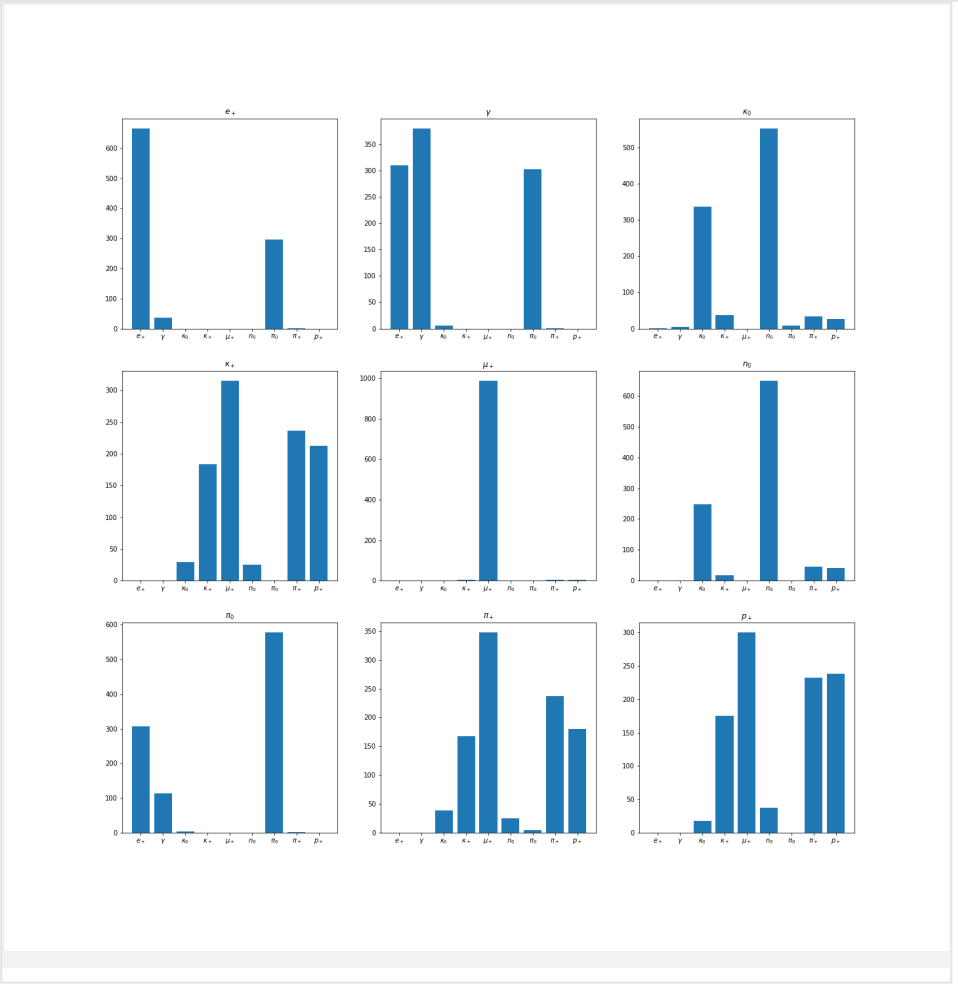

Fig 22 Predictions of ConvNet1 trained on Configuration 2. Test accuracy: 47.26%.

4.1.3 Configuration 3

ConvNet1 was trained for configuration 2. The training was stopped at 20th epoch and it ended with a training accuracy of 60.57% and validation accuracy of 52.11%. The performance on the test data is shown in figure 23.

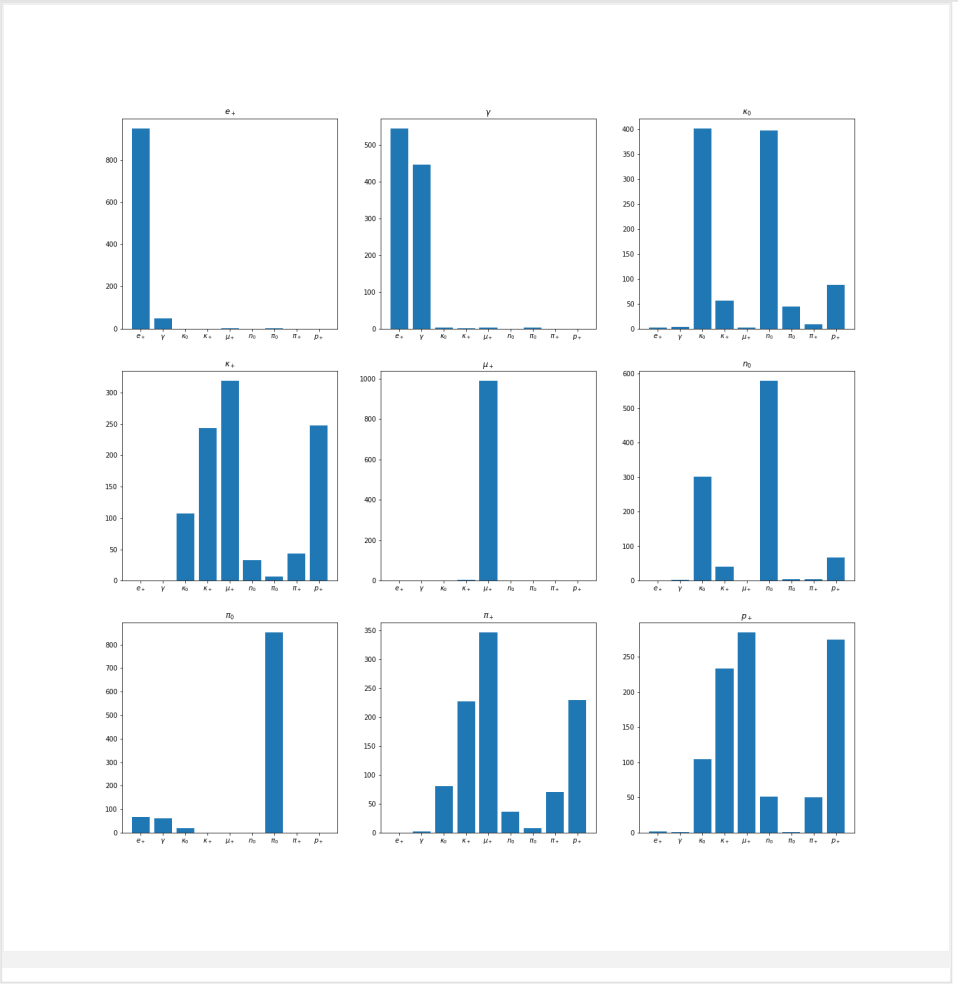

Fig 23 Predictions of ConvNet1 trained on Configuration 3. Test accuracy: 53.4%.

4.1.4 Configuration 4

ConvNet2 was trained for configuration 4. The training was stopped at the 4th epoch and it ended with a training accuracy of 64.31% and validation accuracy of 61.22%. The performance on the test data is shown in figure 24.

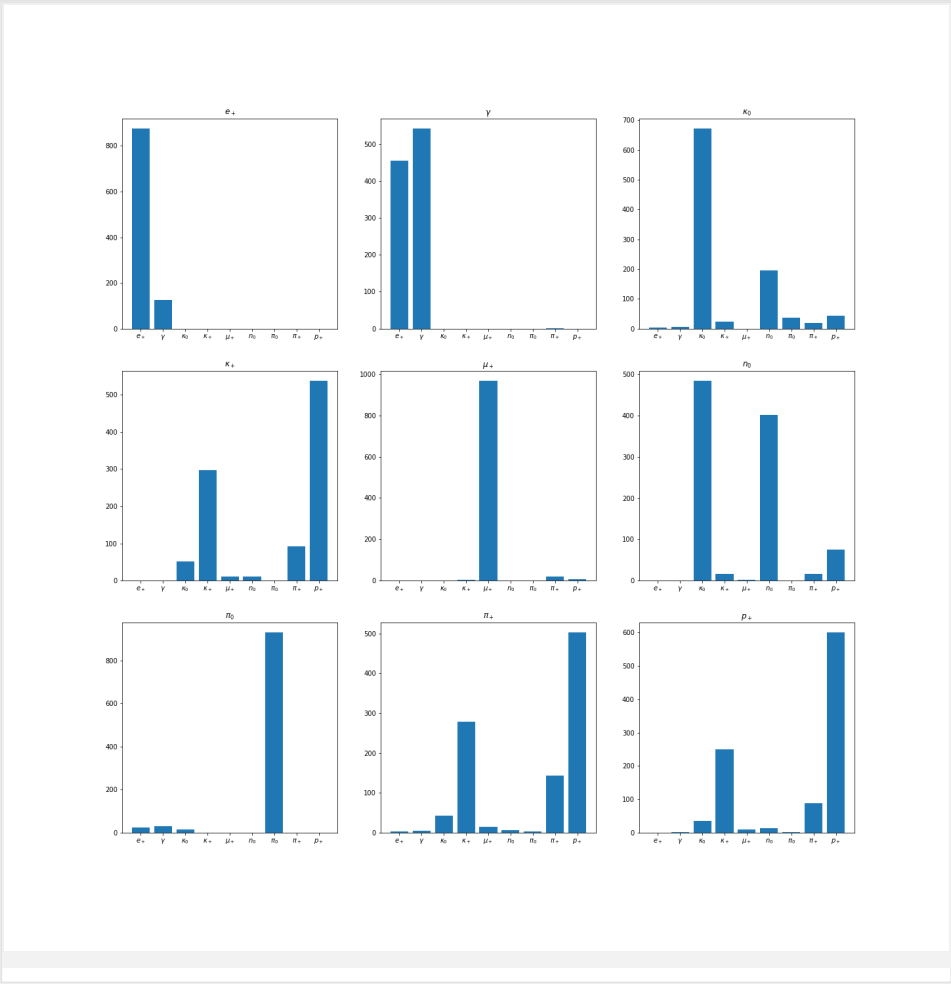

Fig 24 Predictions of ConvNet2 trained on Configuration 4. Test accuracy: 60.34%.

4.2 Conclusion

Largely, the program devided the data into four classes, positrons and photons as one, pion0, muon, positive hadrons, and neutral hadrons. Some improvements in each configuration were:

- The difference from configuration 1 to configuration 2 is very minimal. Thus a magnetic field of 4 Tesla did not affect the results by large.
- The first major improvement from first two configuration to third configuration is the distimguishing pion0 from photon and positron. This is because pion0 has a very short half life and decays to give two photons. When shot from he surface of ECal two photons and a single photon looks similar, thus creating the confusion. However, when shot from a distance, two photons can be

- easily distinguished from a single photon, as they get seperated enough by the time it reaches the ECal surface. This is shown in fig 25.
- Next major improvement in the final configuration was that it was able to tell apart muon from other particles. Initial configurations were getting confused here, because with just 10 layers, even hadrons can sometimes pass through the ECal without creating any shower. Hadrons create late shower and thus by increasing the depth we increase the chance of observing a hadronic shower. However, muon always passes through without any interactions.
- The distinction between charged particles and neutral particles could have been improved by adding a thin layer of Silicon, which either registers a tick (if the partice is charged), or it does not. Thus, improving the segragation between particles such as neutron and proton.

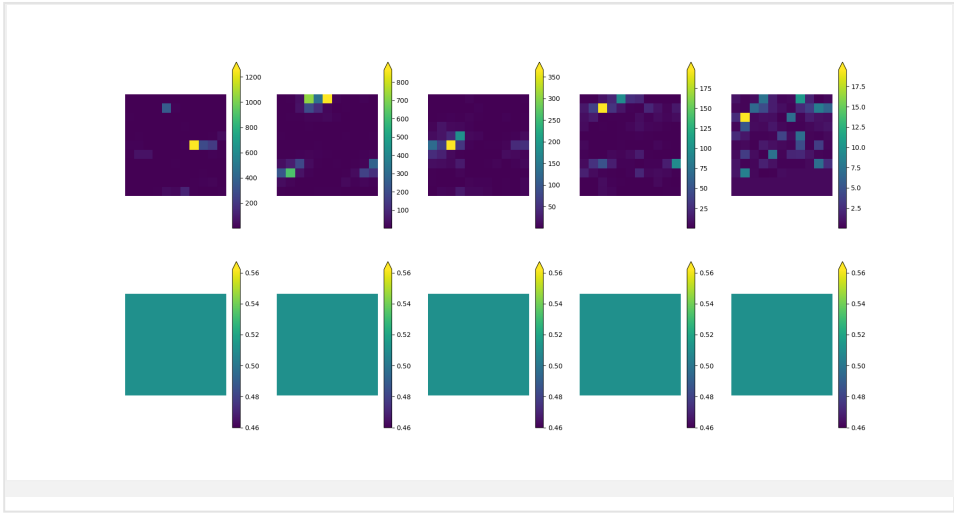

Fig 25 π_0 beam reaches offset from the center of detector due to the added distance. Energy values are in MeV.

4.3 Future Prospect

The next step in this analysis would be to train a regression model to predict the incident energy of the particles and also the incident angle. Also, another configuration with 40 layers has to be tried where the magnetic field is active inside the crystal. Then moving on to higher energies, the models will be trained with beams of 100 GeV in hopes of improving the performance. Once a model with acceptable classification accuracy is obtained, a YOLO kind of implementation will be tried on crystals with larger xy plane, to detect and localize the particle beams and jet beams.

REFERENCES

1. Andrew Glassner (2018). Deep Learning: From Basics to Practice, Vol. 1 and Vol. 2. ASIN:B079XSQNRX.
2. Nishant Shukla & Kenneth Fricklas (2018). Machine Learning with Tensor Flow. ISBN:9781617293870.
3. Joseph Redmon et al. (2016). You Only Look Once: Unified, Real-Time Object Detection. arXive:1506.02640v5.
4. Sepp Hochreiter & Jurgen Schmidhuber (1997). Long Short-Term Memory. Neural Computation 9(8):1735-1780.
5. Olaf Ronneberger et al. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. arXive:1505.04597v1
6. Andrew Ng (2018). Machine Learning Yearning: Technical strategy for AI engineers in the era of deep learning. ASIN: B07SSJCBXF.
7. Ka-Chun Wong (2015). Evolutionary Algorithms: Concepts, Designs, and Applications in Bioinformatics: Evolutionary Algorithms for Bioinformatics. arXiv:1508.00468.

ACKNOWLEDGEMENTS

I would like to thank DST INSPIRE and Science Academy's Summer Research Fellowship programme for funding the project. I am grateful to IAS-INSA-NASI, IISER Kolkata and Dr. Ritesh K Singh for providing me with the opportunity to work on the project. Also, the majority of data generation in the project were done by Soumya Dansana of IISER Kolkata, a huge thanks to him as well.