Active Deep Learning framework with Data programming

Sudhir Kumar Kedarnath Rai (cs17emds11030@iith.ac.in)
Nimesh Jha (cs17emds11014@iith.ac.in)
Santosh Kumar Siripuram (cs17emds11025@iith.ac.in)

Indian Institute of Technology
Hyderabad

Abstract

This project tries to offer a solution to implement existing Deep learning, Active Learning and Data Programming methodologies in medical imaging classification. Deep Learning requires a large labeled set to obtain high-quality results. In this project, we tried to incorporate the active learning and data programming to improve the accuracy in deep learning framework as well as to offer an alternative to combine two powerful methods Active Learning and Data Programming Snorkel.

1. Introduction

Deep learning techniques have recently achieved impressive results in a variety of computer vision problems, raising expectations that they might be applied in other domains, such as medical image analysis. The classification performance (~84%) demonstrated the potential of CNNs in analyzing lung patterns. Data programming enables users to programmatically and cheaply generate large but noisy training sets. Nonconvex analysis techniques then allow us to model and denoise these noisy training datasets.

2. Convolutional Neural Networks

CNNs are feed-forward ANN inspired by biological processes and designed to recognize patterns directly from pixel images (or other signals), by incorporating both feature extraction and classification. A typical CNN involves four types of layers: convolutional,
Activation, pooling and fully-connected (or dense) layers. The training of CNNs is performed similarly to that of other ANNs, by minimizing a loss function using gradient descent-based methods and backpropagation of the error. In this project, we analyze a deep CNN for the classification of lungs X-ray images that exploits the outstanding descriptive capability of deep neural networks

3. Deep Learning Model

Our model, (shown in Figure 1), is a 9-layer convolutional neural network that inputs a chest X-ray image and outputs the probability of pneumonia. The dataset has been collected from Kaggle dataset and has ~1500 X-ray images. For this project we've size normalized the images with 400 x 400 pixels. Detecting pneumonia in chest radiography can be difficult for radiologists. The appearance of pneumonia in X-ray images is often vague, can overlap with other diagnoses, and can mimic many other benign abnormalities [2]. Figure1 shows the CNN model summary which describes the layer and trainable parameters used to achieve ~84% accuracy on a validation set.

```
1 model.summary()
```

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	(None, 400, 400, 1)	0
conv2d_9 (Conv2D)	(None, 399, 399, 32)	160
max_pooling2d_9 (MaxPooling2	(None, 199, 199, 32)	0
dropout_9 (Dropout)	(None, 199, 199, 32)	0
conv2d_10 (Conv2D)	(None, 198, 198, 32)	4128
max_pooling2d_10 (MaxPooling	(None, 99, 99, 32)	0
dropout_10 (Dropout)	(None, 99, 99, 32)	0
conv2d_11 (Conv2D)	(None, 98, 98, 32)	4128
max_pooling2d_11 (MaxPooling	(None, 49, 49, 32)	0
dropout_11 (Dropout)	(None, 49, 49, 32)	0
conv2d_12 (Conv2D)	(None, 48, 48, 32)	4128
max_pooling2d_12 (MaxPooling	(None, 24, 24, 32)	0
dropout_12 (Dropout)	(None, 24, 24, 32)	0
flatten_3 (Flatten)	(None, 18432)	0
dense_5 (Dense)	(None, 32)	589856
dense_6 (Dense)	(None, 2)	66

Total params: 602,466
Trainable params: 602,466
Non-trainable params: 0

Figure 1: CNN Model Summary

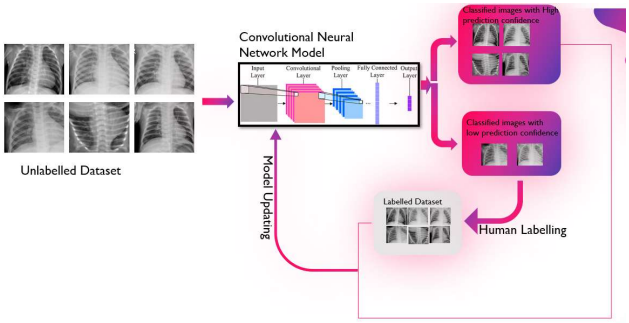


Figure 2: Active learning Methodology

4. Active Learning

This will be used to identify which instances of the dataset need to be labeled. All active learning algorithms follow the next vital steps in an iteratively way: (a) initialization, the starting point before start learning process, pre-training with a starting labeled pool (b)selection new training samples, methodologies to choose new samples to be labeled based on pre-trained model, (c) re-training, training the model again adding the new labels to overall training set, and come back to new labels selection step, (d)finalization, define a methodology to stop the process. Figure2 shows the overall process of Active Learning. Initial labeled data will be used to initialize the convolutional neural network parameters. Once we have the model, we rank all the unlabeled data according to active learning criteria based on the prediction confidence. Now we have two instances, one with a minority sample with low confidence and another with majority sample which has high confidence. Further, the low confidence sample is manually annotated by the human expert, and then we combine both samples and update/retrain the model. The process runs until it reaches a predefined criteria for stopping.

5. Data programming - Snorkel

This will be used to automatically label the instances selected for labeling by the active learning process. Large labeled training sets are the critical building blocks of supervised learning methods and are critical enablers of deep learning techniques. For some applications, creating labeled training sets is the most time-consuming and expensive part of applying machine learning. Snorkel has therefore proposed a paradigm for the programmatic creation of training sets called data programming in which users express weak supervision strategies or domain heuristics as labeling functions, which are programs that label subsets of the data, but that are noisy and may conflict. Snorkel shows that by explicitly representing this training set labeling process as a generative model, it can "denoise" the generated training set, and establish theoretically that snorkel can recover the parameters of these generative models in a handful of settings. Initial user studies we observed that data programming might be a more natural way for non-experts to create machine learning models when training. Figure3 shows Human labeling effort is replaced by data programming.Figure4 shows high-level Data Programming workflow provided by Snorkel

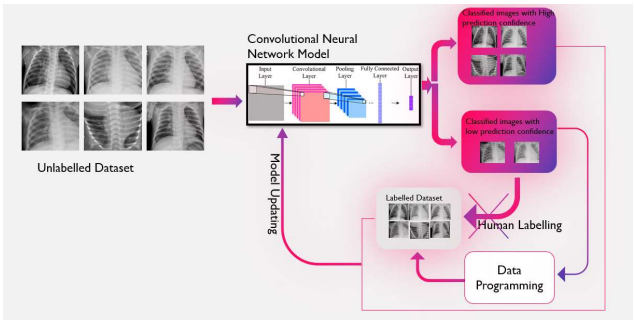


Figure 3

5.1 Data Programming: Current Issue & a possible solution

Current Issue:

Data programming is trying to eliminate the need of domain experts to generate sizeable labeled training dataset. However, in the current setting in the snorkel, it needs domain experts to write labeling functions to generate labels programmatically. This partially defeats the purpose of the entire data programming paradigm.

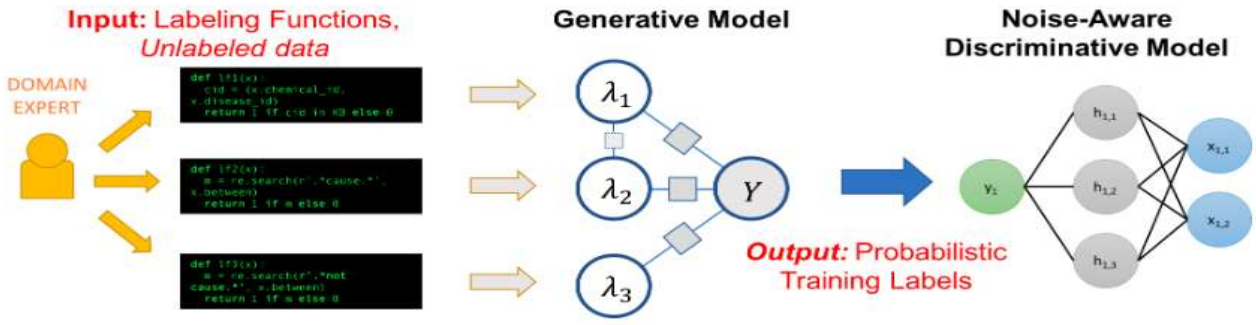


Figure 4: Data programming Model (Snorkel)

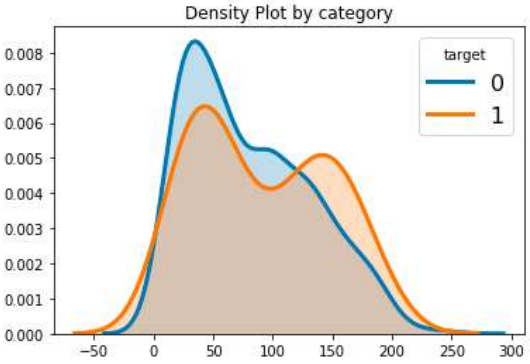


Figure 5
Case 1: Distribution is almost similar by class

A Possible Solution:
We are proposing automatic labeling using features of the data itself. The idea here is drawn from fundamental exploratory data analysis. In the classification setting when we explore
various features of the data and analyze their relationship with the target variable, we see the following typical scenarios:

a. Various charts like box plot, density plot show that the distribution of data is not different by class. In turn, we can say that a feature does not have discriminative power.
b. Box plots and density plots show that the distribution of feature is quite different for each class. In turn, we can say that a feature has significant discriminative power. This means it has a higher predictive power.
c. Box plots and density plots show that in general, it is difficult to find a difference in distribution by class, but there are individual pockets in the distribution where minority class is present. Also, there can be more than
one such pocket in the entire distribution.

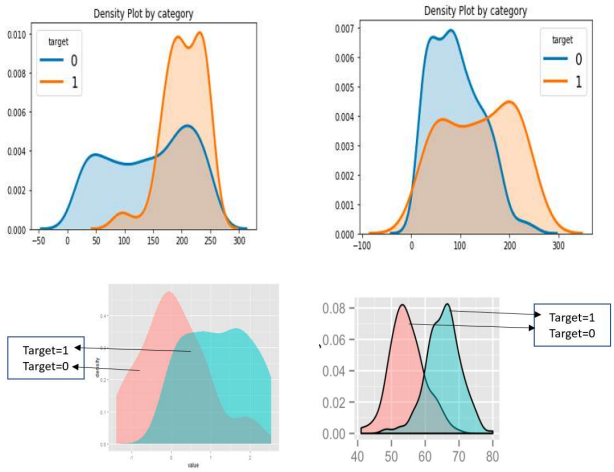


Figure 6
Case 2: Distribution is visibly different for each class

5.2 Properties of the distributions:

We can analyze the properties of the distributions in case 2 and case 3 as there they differ for class 0 and class 1:
Properties of distributions for Case 2: It is observed that the distribution of the feature is different for class 1 and 0 in the following aspect:
Class 1: high mean (median), the high standard deviation for one feature, high mean(median), low standard deviation another feature
Class 0: low mean (median), the low standard deviation for one feature, low mean (median), high standard deviation another feature
Properties of distributions for Case 3: We can see that there are some pockets in the distribution of the feature where target=1. This is a useful pattern to develop a meaningful logic to create a labeling function.

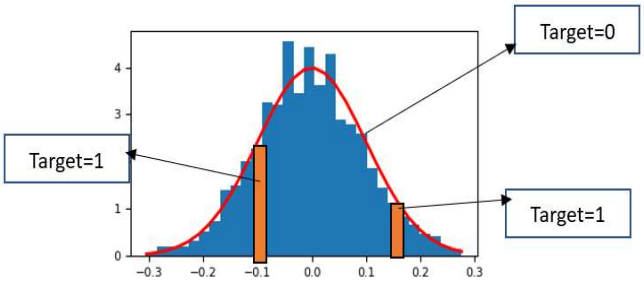


Figure 7
Case 3: Distribution is visibly different for each class

5.3 Data Labeling methods

Case 1 is not very useful. However, case 2 and case 3 kind of distribution has significant discriminative power. The key idea here is to generate these distributions from the actual distribution of the feature and use the probability values from distributions to assign labels based different cut-offs.
Current work is restricted to creating four algorithms from 4 distributions shown in Figure 6. The algorithm based on case 3 will be future research work.

Case2.1: Certain features might have distributions as shown in figure 8.
Characteristic of this kind of distribution are:

1. Class 1 has a high median and low standard deviation
2. Class 0 has a low median and high standard deviation

These characteristics will be used to assign labels. The key idea is to generate two distributions from the actual Median and Standard Deviation of the feature. These two

distributions will have characteristics as above. The iterate over the feature and calculated the probability of each element of the feature coming from these two distributions. Then generate the labels accordingly.

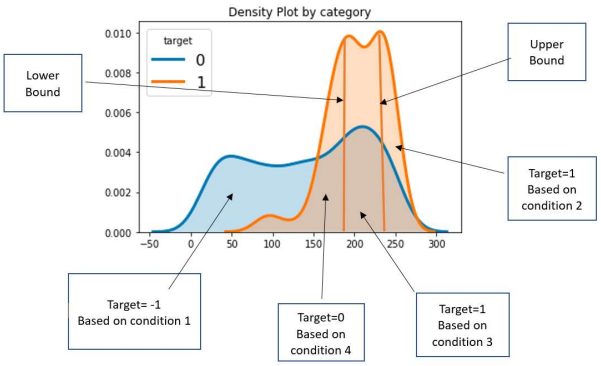


Figure 8 - Case 2.1

```python
# Case 2.1 - Algorithm as follows:
def Generate_Labels(Feature, absCutOff, pctCutoff):
    Feature_Median = Median(Feature)
    Feature_SD= StandardDev(Feature)
    High_Median =  Feature_Median + ε
    Low_Median =  Feature_Median - ε
    High_SD = Feature_SD + ε
    Low_SD = Feature_SD - ε
    lower_bound= High_Median - Low_SD
    upper_bound= High_Median + Low_SD
    for element in feature:
        P1=NormProbability(High_Median, Low_SD, element)
        P2=NormProbability(Low_Median, High_SD, element)
        Diff=P1-P2
        absDiff=abs(Diff)
        pctDiff= absDiff/P1
        if((diff<0) & (absDiff > pctCutOff) & (element < High_Median-Low_SD):
            label=-1
        elif(((diff>0) & (absDiff > pctCutOff) & (element > High_Median+Low_SD):
            label=1
        elif((diff>0)&(pctDiff>pctCutoff)&(element>=lower_bound)&(element<=upper_bound):
            label=1
        else: label=0
```

Case 2.1 algorithm

Case 2.2 Certain features might have distributions as shown in figure 9. Characteristic of this kind of distribution are:

1. Class 1 has a high median and high standard deviation
2. Class 0 has a low median and low standard deviation

Similar to in case 2.1, these distribution characteristics will be used for label generation.

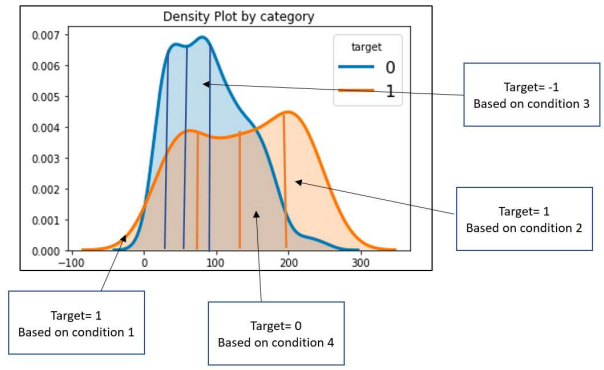


Figure 9 - Case 2.2

```python
# Case 2.2 - Algorithm as follows:
def Generate_Labels(Feature,absCutOff, pctCutoff):
    Feature_Median = Median(Feature)
    Feature_SD= StandardDev(Feature)
    High_Median =  Feature_Median + ε
    Low_Median =  Feature_Median - ε
    High_SD = Feature_SD + ε
    Low_SD = Feature_SD - ε
    lower_bound= Low_Median - Low_SD
    upper_bound= Low_Median + Low_SD
    for element in feature:
        P1=NormProbability(High_Median, High_SD, element)
        P2=NormProbability(Low_Median, Low_SD, element)
        Diff=P1-P2
        absDiff=abs(Diff)
        pctDiff= absDiff/P1
        if((diff>0) & (absDiff > pctCutOff) & (element < High_Median-1.5*High_SD):
            label=1
        elif(((diff>0) & (absDiff > pctCutOff) & (element > High_Median+ 1.5 * High_SD):
            label=1
        elif((diff<0)&(pctDiff>pctCutoff)&(element>=lower_bound)&(element<=upper_bound):
            label=-1
        else: label=0
```

Case 2.2 algorithm

Case 2.3 Certain features might have distributions as shown in figure 10. Characteristic of this kind of distribution are:

1. Class 1 has a high median and high standard deviation
2. Class 0 has a low median and high standard deviation

Similar to in case 2.1, these distribution characteristics will be used for label generation.

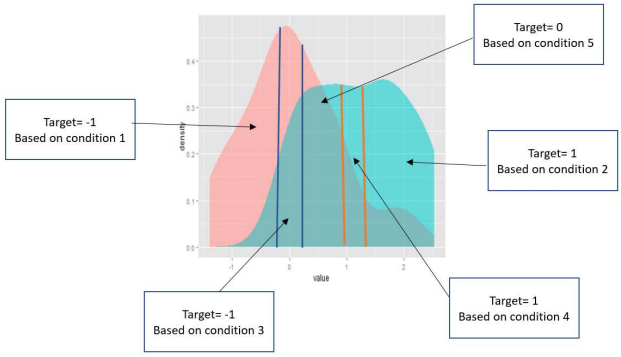


Figure 10 - Case 2.3

```
# Case 2.3 - Algorithm as follows:
def Generate_Labels(Feature,absCutOff, pctCutoff):
    Feature_Median = Median(Feature)
    Feature_SD= StandardDev(Feature)
    High_Median =  Feature_Median + ε
    Low_Median =  Feature_Median – ε
    High_SD = Feature_SD + ε
    Low_SD = Feature_SD – ε
    lower_bound= High_Median - High_SD
    upper_bound= Low_Median + High_SD
    for element in feature:
        P1=NormProbability(High_Median, High_SD, element)
        P2=NormProbability(Low_Median, High_SD, element)
        Diff=P1-P2
        absDiff=abs(Diff)
        pctDiff= absDiff/P1
        if((diff<0) &  (element < Low_Median):
            label=-1
        elif(((diff>0) & (element > High_Median):
            label=1
        elif((diff<0)&(pctDiff>pctCutoff)&(element>=Low_Median)&(element<=upper_bound):
            label=-1
        elif((diff>0)&(pctDiff>pctCutoff)&(element< High_Median)&(element>=lower_bound):
            label=1
        else: label=0
```

Case 2.3 algorithm

Case 2.4 Certain features might have distributions as shown in figure 11. Characteristic of this kind of distribution are:

1. Class 1 has a high median and low standard deviation
2. Class 0 has a low median and low standard deviation

Similar to in case 2.1, these distribution characteristics will be used for label generation.

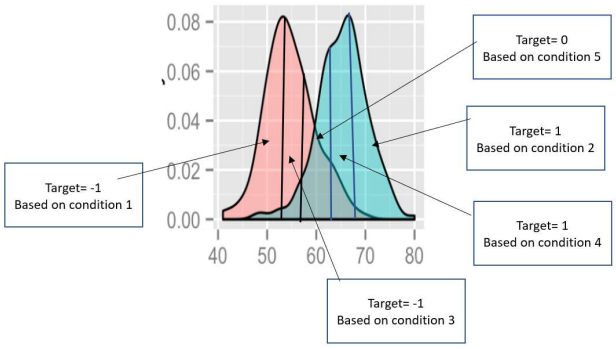


Figure 11 - Case 2.4

```
# Case 2.4 - Algorithm as follows:
def Generate_Labels(Feature,absCutOff, pctCutoff):
    Feature_Median = Median(Feature)
    Feature_SD= StandardDev(Feature)
    High_Median =  Feature_Median + ε
    Low_Median =  Feature_Median – ε
    High_SD = Feature_SD + ε
    Low_SD = Feature_SD – ε
    lower_bound= High_Median -  1.5 *Low_SD
    upper_bound= Low_Median + 1.5 * Low_SD
    for element in feature:
        P1=NormProbability(High_Median, Low_SD, element)
        P2=NormProbability(Low_Median, Low_SD, element)
        Diff=P1-P2
        absDiff=abs(Diff)
        pctDiff= absDiff/P1
        if((diff<0) &  (element < Low_Median):
            label=-1
        elif(((diff>0) & (element > High_Median):
            label=1
        elif((diff<0)&(pctDiff>pctCutoff)&(element>=Low_Median)&(element<=upper_bound):
            label=-1
        elif((diff>0)&(pctDiff>pctCutoff)&(element<= High_Median)&(element>=lower_bound):
            label=1
        else: label=0
```

Case 2.4 algorithm

5.4 Experiment and Results:

All label generating functions accept two parameters defined as follows:
absCutoff: Threshold for probability difference between probabilities of an element from two different distributions. This is generally kept low due to low probability values.
pctCutoff: Threshold for the ratio between absolute difference into probability values and probability of perceived class 1. This is generally kept high. The idea here is to mark class labels for cases with high percentage difference in probability in the central part of the distribution. If this is not done then many instances will not have been marked as they belong to overlapping area of the two distributions.

Two experiments were run for two different values of pctCutoff.
Experiment Procedure:

1. A select number of features to be used to generate labels.
2. Generate labels
3. Select only those labels which have more than 60% coverage
4. Pass these labels to snorkel generative model
5. Get class labels from snorkel given probability by using different cutoffs.
6. Evaluate the accuracy of the labels with actual labels.

Experiment 1 input and results (pctCutoff = 0.15):

No_Feature_Used	Total_No_Label_Generated	No_Label_To_GenerativeModel
5	36	26
10	70	64
15	104	84
20	142	106
25	172	128
30	212	154
35	242	200
40	282	210
45	308	224
50	338	254
55	382	278
100	670	546

Table 1 - Count of labels generated and used in the generative model:

Points to be noted here is that the accuracy statistics is only calculated based on the instances for which label generator has assigned a label. This is means accuracy statistics is generated only based on coverage of the labels generated.
It is observed that there is a little increasing trend in coverage, precision and F1 score as the number of features increase.

No_Feature_Used	Coverage	Precision	Recall	F1	Accuracy
5	0.666667	0.580178	0.5	0.516568	0.5
10	0.832137	0.609203	0.5	0.537883	0.5
15	0.777015	0.601551	0.5	0.541149	0.5
20	0.742423	0.603607	0.5	0.518300	0.5
25	0.718288	0.602134	0.5	0.518817	0.5
30	0.705842	0.580238	0.5	0.531172	0.5
35	0.774215	0.587374	0.5	0.535845	0.5
40	0.721782	0.594480	0.5	0.535408	0.5
45	0.715242	0.599864	0.5	0.532165	0.5
50	0.730414	0.595366	0.5	0.528754	0.5
55	0.717420	0.589422	0.5	0.537838	0.5
100	0.769371	0.606942	0.5	0.537896	0.5

Table 2 - Average accuracy of labels generated by the label generator

No_Feature_Used	Precision	Accuracy	F1	Recall
5	0.690476	0.5040	0.483333	0.371795
10	0.559871	0.4352	0.494993	0.443590
15	0.559603	0.4336	0.488439	0.433333
20	0.566265	0.4464	0.520776	0.482051
25	0.610000	0.4112	0.248980	0.156410
30	0.684543	0.5632	0.613861	0.556410
35	0.595745	0.4480	0.448000	0.358974
40	0.646310	0.5600	0.648787	0.651282
45	0.667598	0.5680	0.639037	0.612821
50	0.636364	0.4576	0.412478	0.305128
55	0.627848	0.5376	0.631847	0.635897
100	0.642487	0.5520	0.639175	0.635897

Table 3 - Accuracy of labels generated by Snorkel Generative Model

90% probability cut-off is used to create labels from generative model probability. Here it is observed that the highest values of different accuracy measurement are coming for labels generated by 100 features. Most importantly F1 score is highest in this case. This is the most important accuracy metric for the experiment because we want to know how well the system can label the primary class.
For 30 features case precision is highest among all cases but F1 is lower than 100 feature cases.

Label_Name	Coverage	Precision	Recall	F1	Accuracy
ProbDist_3_Column_113593	0.9952	0.741935	0.592784	0.659026	0.617363
ProbDist_7_Column_113593	0.9952	0.741935	0.592784	0.659026	0.617363
ProbDist_4_ColumnR_113593	0.9952	0.506410	0.407216	0.451429	0.382637
ProbDist_8_ColumnR_113593	0.9952	0.506410	0.407216	0.451429	0.382637
ProbDist_8_ColumnR_40464	0.9792	0.796721	0.641161	0.710526	0.676471
ProbDist_7_Column_103510	0.9792	0.776316	0.617801	0.688047	0.650327
ProbDist_8_ColumnR_79662	0.9792	0.724919	0.584856	0.647399	0.601307
ProbDist_7_Column_110728	0.9792	0.675410	0.543536	0.602339	0.555556
ProbDist_8_ColumnR_110728	0.9792	0.563518	0.456464	0.504373	0.444444
ProbDist_7_Column_79662	0.9792	0.524752	0.415144	0.463557	0.398693

Table 4 - Accuracy of top 10 labels generated by 100 features before the Generative Model

We can observe that all these labels have high coverage and high precision.

Experiment 2 input and results (pctCutoff = 0.08):

No_Feature_Used	Total_No_Label_Generated	No_Label_To_GenerativeModel
5	34	30
10	68	54
15	110	70
20	134	108
25	182	116
30	210	144
35	246	190
40	280	200
45	304	248
50	346	260
55	388	286
100	696	482

Table 5 - Count of labels generated and used in the generative model:

No_Feature_Used	Coverage	Precision	Recall	F1	Accuracy
5	0.860047	0.592384	0.5	0.568782	0.5
10	0.725553	0.588080	0.5	0.528095	0.5
15	0.672000	0.596885	0.5	0.529309	0.5
20	0.790209	0.607225	0.5	0.534816	0.5
25	0.674338	0.591140	0.5	0.526903	0.5
30	0.690575	0.596399	0.5	0.536493	0.5
35	0.762433	0.595895	0.5	0.529482	0.5
40	0.716743	0.590553	0.5	0.536529	0.5
45	0.773242	0.608682	0.5	0.531026	0.5
50	0.741179	0.600036	0.5	0.531266	0.5
55	0.728965	0.589801	0.5	0.530108	0.5
100	0.706349	0.596504	0.5	0.531064	0.5

Table 6 - Average accuracy of labels generated by the label generator

It is observed here is that the increasing trend in F1 score with an increase in the number of features seen in experiment 1 is not seen here. However, this is average statistics; hence some outlier effect may not be ruled out.

No_Feature_Used	Precision	Accuracy	F1	Recall
5	0.623693	0.4896	0.528804	0.458974
10	0.685131	0.5792	0.641201	0.602564
15	0.630380	0.5408	0.634395	0.638462
20	0.625323	0.5312	0.622909	0.620513
25	0.694444	0.5776	0.630252	0.576923
30	0.667553	0.5776	0.655352	0.643590
35	0.542857	0.4144	0.453731	0.389744
40	0.555160	0.4256	0.464978	0.400000
45	0.701863	0.5840	0.634831	0.579487
50	0.623501	0.5408	0.644362	0.666667
55	0.676630	0.5840	0.656992	0.638462
100	0.631004	0.5680	0.681604	0.741026

Table 7 - Accuracy of labels generated by Snorkel Generative Model

Here it is observed that the highest values of different accuracy measurement are coming for labels generated by 100 features. Most importantly F1 score and Recall is highest in this case. These results are better than the results in experiment 1. This is suggesting that pctCutoff is an important parameter to get better labeling accuracy for class 1. For 25 features case precision is highest among all cases but F1 and Recall is lower than 100 feature cases.

Label_Name	Coverage	Precision	Recall	F1	Accuracy
ProbDist_7_Column_43044	0.9952	0.755627	0.607235	0.673352	0.633441
ProbDist_7_Column_89107	0.9952	0.739550	0.591260	0.657143	0.614148
ProbDist_8_ColumnR_89107	0.9952	0.511254	0.408740	0.454286	0.385852
ProbDist_8_ColumnR_43044	0.9952	0.488746	0.392765	0.435530	0.366559
ProbDist_3_Column_144790	0.9936	0.730769	0.589147	0.652361	0.608696
ProbDist_7_Column_144790	0.9936	0.730769	0.589147	0.652361	0.608696
ProbDist_4_ColumnR_144790	0.9936	0.514563	0.410853	0.456897	0.391304
ProbDist_8_ColumnR_144790	0.9936	0.514563	0.410853	0.456897	0.391304
ProbDist_8_ColumnR_56863	0.9920	0.782468	0.619537	0.691535	0.653226
ProbDist_8_ColumnR_21529	0.9920	0.766990	0.613990	0.682014	0.643548

Table 8 - Accuracy of top 10 labels generated by 100 features before the Generative Model

We can observe that all these labels have high coverage and high precision. Based on the results of these experiments we can see that the initial logic of automatic labeling is working up to a certain extent. By using up to 100 features and by using different parameter values we can get up to 68% F1 score. However, to come up with a more concrete framework for automatic labeling, this idea needs to be further explored. More detailed experiments need to be run using more number of features and by varying the values of labeling function parameter values in a wider range than explored in this project.

6. Integration of Components

Table 9 summarizes the results obtained after integrating data programming labeled data with original labeled data with just one iteration on updating the model.

	Train loss	Train acc	Val loss	Val acc
CNN (Train on 1481 samples, validate on 635 samples with epoch = 10)	0.0987	0.9608	0.4381	0.8898
CNN with Exp1(Train on 2106 samples, validate on 635 samples with epoch =10)	0.3066	0.8642	0.8686	0.7496
CNN with Exp2 (Train on 2106 samples, validate on 635 samples with epoch =10)	0.2990	0.8761	0.4564	0.7496

Table 9

The results indicate that the data programming model needs improvement in accuracy. However, these initial results confirm that the overall pipeline and framework to have data programming integration with CNN model works. We understand that multiple iterations to update the model would result in better accuracy.

7. Future Work and Conclusion

7.1 Future Work:

Data Programming: Future work in this area is to come up with a labeling function based on case 3 like distribution. There are a few more ideas to be explored to enhance performance under the current framework. For example:

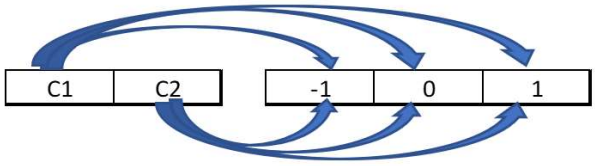
1. Bootstrapped Sample of Labels: The idea here is that before sending feature generated labels to "Snorkel Generative Model," take a bootstrapped sample of labels and then take the mode of labels for each sample in each row. This will give a majority voted labels generated from originally generated labels. Now supply these samples to "Snorkel Generative Model." The intuition is that the mode will give a higher confidence in the labels.

2. Localizing using Clustering: The idea here is to cluster data before using label generation. Then for data in each cluster run the label generating process. Then consolidate at the end. The intuition is that clustering will help localization of patterns and hence for each cluster different function parameters can be used to gain maximum performance.

In this framework, we have tried to integrate data programming to replace human effort in active learning.

3. Label Generation using Categorical Features: So far in this process, we have only used continuous features for label generation. In future we would like to use categorical features for label generation. High level idea is as follows:

For simplicity let's assume that a categorical feature has two categories. For example Cat_Feature = [C1,C2]. So, the problem is to assign three labels to 2 categories. Three labels are [-1,0,1]



We will use following logic to assign labels:

```
Cat1_Theta_1 = Beta(1,1)
Cat1_Theta_0 = Beta(1,1)
Cat1_Theta_-1 = Beta(1,1)
Cat2_Theta_1 = Beta(1,1)
Cat2_Theta_0 = Beta(1,1)
Cat2_Theta_-1 = Beta(1,1)

for element in Cat_Feature:
    if element=cat1:
        Label=class(argmax(Cat1_Theta_1,Cat1_Theta_0, Cat1_Theta_-1)
        if Label==1:
            Cat1_Theta_1=Beta(1,1+1)
        elif Label==0:
            Cat1_Theta_0=Beta(1,1+1)
        elif Label==-1:
            Cat1_Theta_-1=Beta(1,1+1)
    elif element=cat2:
        Label=class(argmax(Cat2_Theta_1, Cat2_Theta_0, Cat2_Theta_-1)
        if Label==1:
            Cat2_Theta_1=Beta(1,1+1)
        elif Label==0:
            Cat2_Theta_0=Beta(1,1+1)
        elif Label==-1:
            Cat2_Theta_-1=Beta(1,1+1)
```

Further work required to improve the model

- CNN Model Tuning
- Active learning loop Improvement
- Data Programming Validation

7.2 Conclusion

The main objective of this project was to propose a framework able to manage active deep
Learning for medical imaging segmentation. The results obtained are satisfactory but not enough to be compared to the classical training model. We were able to successfully integrate the data programming in the CNN model and able to remove human in the loop. The results obtained do open the door for further work and research on the framework.

As of last work on this project, we may suggest using transfer learning by using pre-trained networks to initialize the weights safely which would certainly give a reasonable boost to the methodology.

The code used in this project can be accessed publicly at *https://github.com/NimJ/IITH_Project*

The dataset can be accessed at *https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia*

References

[1] M. Anthimopoulos, S. Christodoulidis, L. Ebner, A. Christe, and S. Mougiakakou, "Lung Pattern Classification for Interstitial Lung Diseases Using a Deep Convolutional Neural Network," in IEEE Transactions on Medical Imaging, vol. 35, no. 5, pp. 1207-1216, May 2016.

[2] CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning https://arxiv.org/abs/1711.05225v3 .

[3] Alexander Ratner, Christopher De Sa, Sen Wu, Daniel Selsam, Christopher Ré. Data Programming Creating Large Training Sets Quickly. https://arxiv.org/pdf/1605.07723.pdf

[4] Górriz Blanch, Marc: Active deep learning for medical imagingsegmentation https://upcommons.upc.edu/handle/2117/109304

[5] Alexander Ratner Stephen H. Bach Henry EhrenbergJason Fries Sen Wu Christopher R´e. Snorkel: Rapid Training Data Creation with Weak Supervision. https://arxiv.org/pdf/1711.10160.pdf

[6] https://hazyresearch.github.io/snorkel/blog/snorkel_programming_training_data.htm

[7] https://hazyresearch.github.io/snorkel/

[8] http://sujitpal.blogspot.com/2018/02/using-snorkel-probabilistic-labels-for.html

[9] https://www.frontiersin.org/articles/10.3389/fpsyg.2017.01745/full