Cats vs Dogs Detector (CaDoD)

Phase 2:

Abstract:

The objective of our project is to classify images as either dogs or cats. Additionally, we also plan to find where the cat/dog is in the image. Although the task is simple to human eyes, computers find it hard to distinguish between images because of a plethora of factors including cluttered background, illumination conditions, deformations, occlusions among several others. We plan to build an end-to-end machine learning model which will help computers differentiate between cat and dog images with better accuracy.

In our previous phase, we finalized using Gradient boosting and linear regression as our baseline for image detection and boundary box detection respectively. In this phase, we extended the baseline and implemented a complex loss function (CXE + MSE) using homegrown linear and logistic regression.

Furthermore, we built multi-layered perceptron and calculated the accuracy and loss per epoch for classification and regression tasks. We have used data augmentation, dropout layers and regularization to overcome overfitting. In addition to this, we built a multi-headed predictor that calculates the combined loss function from classification and the regression tasks and uses it to optimize the weights and bias of the network.

Accuracy and loss are our primary evaluation parameters. In addition to this, we also used classification report (F1 score) and confusion matrix to evaluate the classification model. Using the above model, we got an accuracy of ~60% for classification and MSE close to zero for regression.

Team Members:

We are a group of 4 members:

1. Aishwarya Sinhasane - avsinhas@iu.edu (In picture, Left top)
2. Himanshu Joshi - hsjoshi@iu.edu (In picture, Right bottom)
3. Sreelaxmi Chakkadath - schakkad@iu.edu (In picture, Left bottom)
4. Sumitha Vellinalur Thattai - svtranga@iu.edu (In picture, Right top)

Project Meta Data:

We have completed the following task this week:

- Homegrown linear regression
    - Complex loss function (CXE + MSE) using homegrown models
- Built sequential neural network for image classification and used the following to improve accuracy
    - Data Augmentation
    - Dropout layers
    - Regularization
- Built sequential neural network for boundary detection
    -
- Compare the performance of the above models using Accuracy and loss
    - Compared the performance of the best-chosen neural network model with the baseline model
- Built a multi-headed cat-dog detector using the OOP API that combines loss function: CXE + MSE
- Implemented convolutional neural network for image classification and box detection

Data Description:

The data we plan to use is the Kaggle data set. We will be using two files – one for image and the other for boundary:

- The images are taken from the cadod.tar.gz
- The boundary information of the images is from cadod.csv file

There are about ~13k images in the data set. There is a good balance of classes with ~6.8K cat images and ~6.1K dog images in the data set.

- Image information (cadod.tar.gz):
    - There are ~13K images of various sizes and aspect ratios. Majority of the images are of 512 X 384 size
    - All the images are in RGB scale
    - The boundaries of the images are stored in the cadod.csv file
- Attributes of the Boundary File (cadod.csv):
    - This has information about the image box coordinates
    - There are about 20 features in this data set
        - 15 numerical features
            - This includes the image ID, the coordinates of the boundary boxes, also the normalized coordinates of the boundary boxes
        - 5 categorical features
            - This gives information about the occlusion, depiction, truncation, etc.

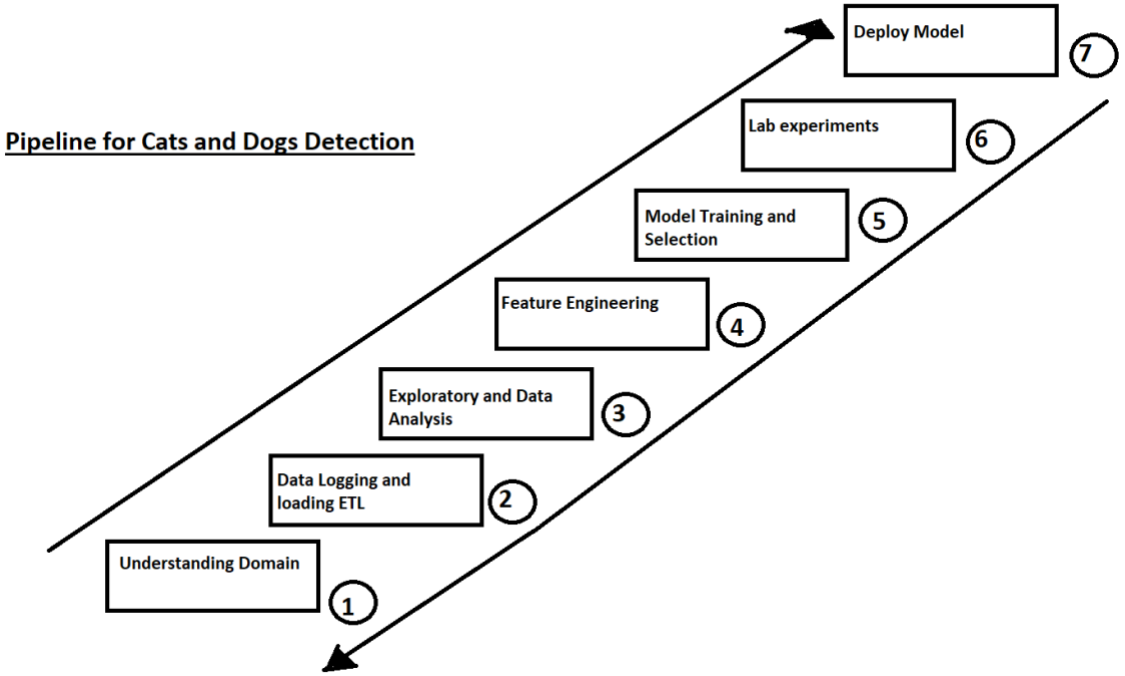
Task:

The following are the end-to-end tasks to achieve the results:

- Image pre-processing (Augmentation)
    - Augmentation involved flipping and rotation
- Use sequential models to build neural networks
    - Neural network classifier for image detection
    - Neural network regressor for box detection
- Implemented the above using OOP API and created a combined loss function (CXE + MSE)
- Build a convolutional neural network

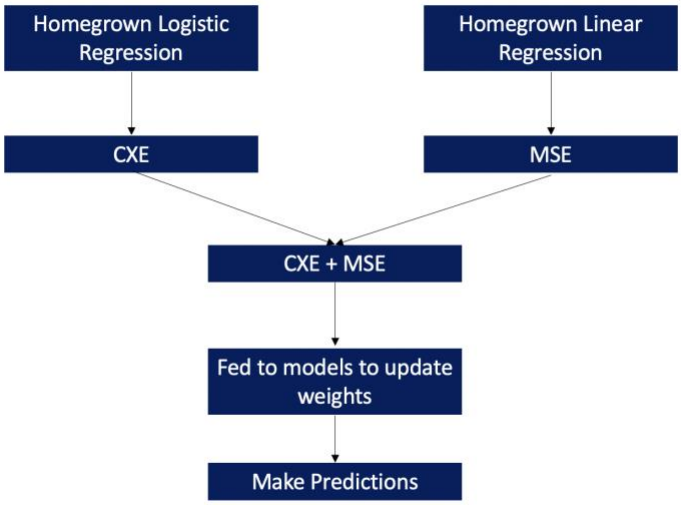
Since the data set is exceptionally large with about 13K images, the above tasks were carried out only in a subset of data. Hence the results will be only directional and not accurate

Workflow:

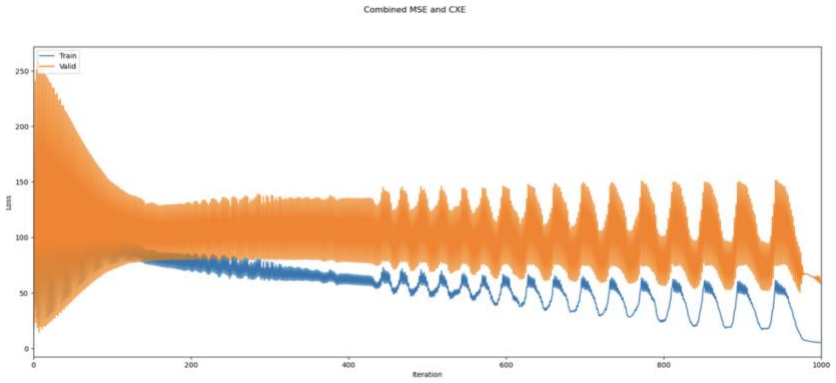


Homegrown linear and logistic regression:

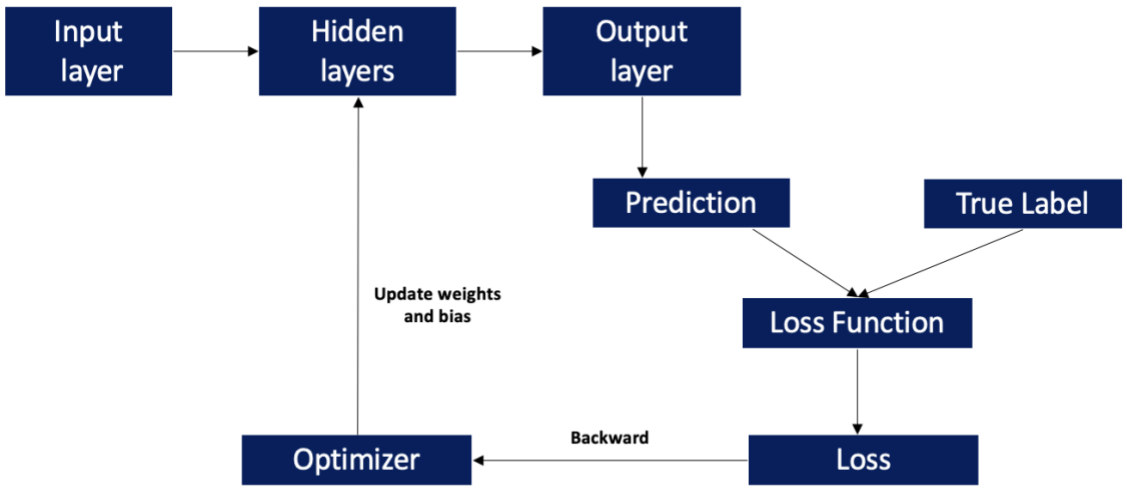
- We have built a homegrown logistic regression for image classification which gives the CXE loss at each epoch. Similarly, we have built a homegrown linear regression for boundary box detection which gives the MSE for each epoch.



- The errors from both models are them combines to give us a complex loss function CXE + MSE which was then used to make predictions.
- **Result:**
    - We ran this for 2000 epoch and found that the loss decreased steadily at each epoch



Neural Network:



We will be using both sequential and class definition (OOP API) to build neural networks.

Sequential neural networks:

In this method, we will pass all our connecting layers in the sequential class constructor itself

Class Definition (OOP API):

To accomplish this in PyTorch you define a Model as a subclasses of torch.nn.Module. This model will include the following methods:

- __init__ method : Initialize the layers we want to use. We have to specify the sizes of our network in this method. We will also add all the layers we want to use in our network in this method
- forward method : We will specify the connections between the layers in this method. We will use the layers you already initialized in the __init__ method, in order to re-use the same layer for each forward pass of data

Classification:

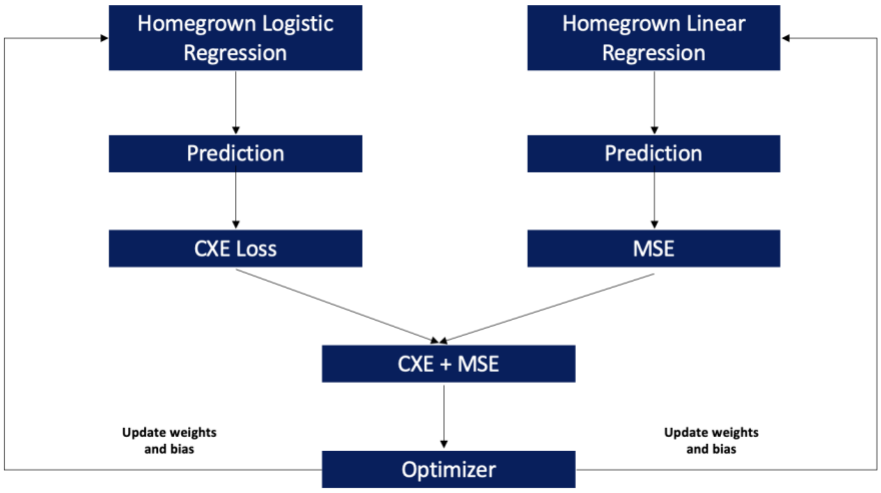
- In both the above cases, we have split the data into train, validation, and test and used 30 epochs to help in the convergence process
- For image detection, we have used 1 input layer, 2 hidden layers and one output layer.
    - ReLU has been used as the activation function and cross-entropy is the loss function.
    - In addition to this, we have experimented with dropout layers.
- We are using Adam optimizer as an optimizer for convergence

Regression:

- We have split the data into train, validation, and tests and used 50 epochs to help in the convergence process
- We have used 1 input layer, 2 hidden layer and 1 output layer to make the prediction for our four boundaries
    - We have use tanh as our activation function
- Mean square is our loss function and we have experimented using stochastic gradient descent and adam as an optimizer for convergence

Multiheaded Neural network:

- **Steps to create the model:**
    - Defined a model for classification and regression separately
    - For each epoch, we calculated CXE and MSE loss from the above models and then used the combined loss for back propagation
    - Adam optimizer was used to take the next step and update the weights and bias



Modeling Pipeline:

- **Preprocessing steps:**
  - Rescaling the images
    - The images are of different size and aspect ratio, we have resized all images to 128 X 128
  - Data augmentation
    - This is done to expand our data set, so it is very diverse.
    - Enables model to learn generalized features instead of overfitting on the dataset
    - Types of augmentation data used:
      - Flip
      - Rotation
  - Normalization
    - The image was normalized so all input parameters have the same distribution. This will, in turn, make our network converge at a faster rate

- **Building a Sequential model:**
  - **a. Image Detection:**
    - i. Sequential neural network classifier
      1. Loss function used – Cross Entropy Loss function
      2. We used a one input layer, 2 hidden layers, and an output layer in our neural network
         a. ReLU was used as the activation function
      3. We also used the following to experiments to avoid overfitting improve accuracy:
         a. Drop out layer

i. With p values of 0.1 and 0.2
b. Batch normalization after output of hidden layer
4. We have used Adam optimizer with a learning rate of .0002
a. We also used a weight decay value of 1e_4 which will provide us with L2 regularization
ii. A similar process was followed for OOP as well

b. Boundary Box Detection

i. Sequential neural network regressor
1. Loss function used – MSE
2. We used a one input layer, 2 hidden layers, and an output layer in our neural network
a. Tanh was used the activation function
3. We also added multiple dropout layers to overcome overfitting
4. We have used Adam as our optimizer with a learning rate of
a. We also used a weight decay value of 1e-3 which will provide us with L2 regularization
ii. A similar process was followed for OOP as well

- **<u>Building a CNN model:</u>**
  - Image Detection:
    - W added a maxpooling2d layer in addition to the linear and activation layer to reduce the number of feature maps
    - We experimented with different # of epochs and learning rate

- **<u>Metrics for evaluation:</u>**
  - Accuracy - the ratio of correction predictions to the total predictions
    - Accuracy = (True Cats + True Dogs) / (True Cats + False Cats + True Dogs + False Dogs)
  - Confusion matrix – Gives us F1 score = 2*precision * recall / (precision + recall)
    - Recall is the ratio of true positives to all actual positives
      - Recall = True positives / (True positives + False negatives)
    - Precision is the ratio of true positives to all predicted positives
      - Precision = True positives / (True positives + False positives)
    - F1 score metric weights both recall and precision equally and thereby a higher value of recall and precision will ensure superior performance of the model
  - Cross entropy loss function for classification and MSE for regression were used as loss function for the neural network

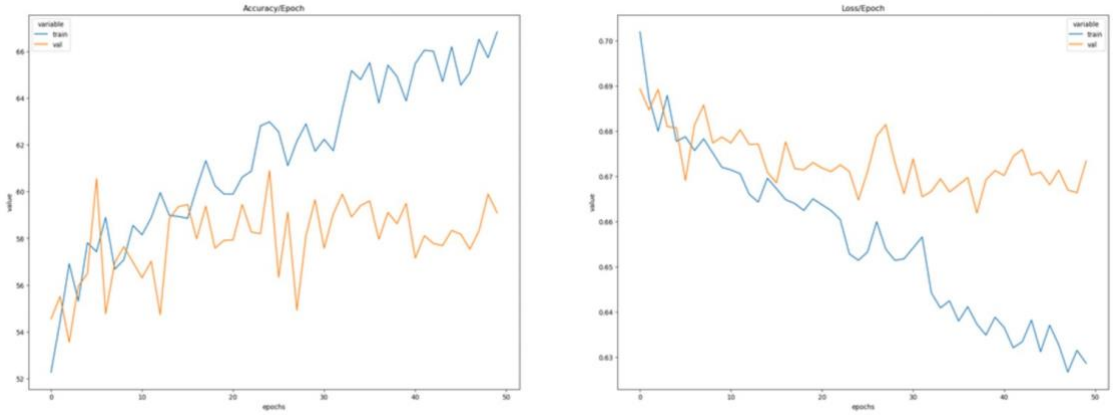
Results and Discussions:

- **Classification:**

	exp_name	Train Acc	Valid Acc	Test Acc
0	Baseline Neural Network without Augmentation, ...	97.9	56.022	56.978
1	Neural Network without Augmentation and with R...	96.325	56.467	57.106
2	Neural Network without Regularization and Dropout	70.702	57.511	59.923
3	Neural Network without Dropout	69.702	58.556	57.875
4	Neural Network with Dropout (p = 0.2)	63.83	57.644	58.259
5	Neural Network with Dropout (p = 0.1)	66.83	59.089	59.283

  - Without augmentation, we see that though we got a training accuracy of more than 90%, the validation accuracy of only 56%, which clearly indicates that the model is overfitting the data
    - Adding regularization did drop the test accuracy but not to a larger extent
  - We have overcome this by data augmentation, adding drop out layers and regularization
  - Our best model with a dropout layer of 0.1 yielded us a training accuracy of 67% and validation and test accuracy of ~60%

Below is the graph of our best model



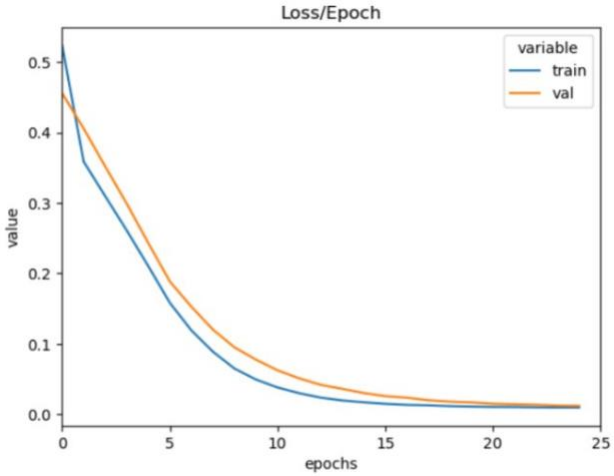
  - We also see that we also got a f1 score of ~60% for this model, with close to no difference in precision and recall values

	precision	recall	f1-score	support
0	0.61	0.57	0.59	402
1	0.57	0.62	0.60	379
accuracy			0.59	781
macro avg	0.59	0.59	0.59	781
weighted avg	0.59	0.59	0.59	781

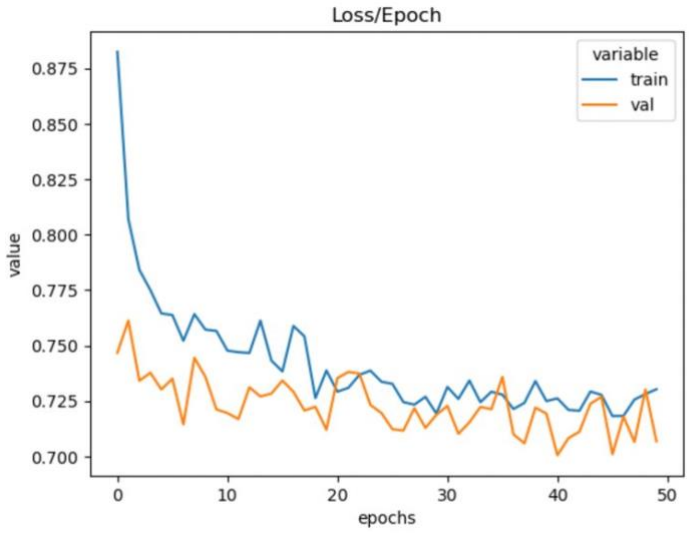
- **Regression:**

	exp_name	Train MSE	Valid MSE	Test MSE
2	Sequential Regression NN Dropout (p = 0.5)	0.009	0.011	0.011
0	Sequential Regression NN Dropout (p = 0.1)	0.009	0.011	0.011
1	Sequential Regression NN Dropout (p = 0.2)	0.01	0.012	0.012
3	Sequential Regression NN Dropout (p = 0.3)	0.01	0.013	0.013
4	Sequential Regression NN Dropout (p = 0.4)	0.01	0.012	0.012
5	Sequential Regression NN without dropout	0.01	0.012	0.012
6	Sequential Regression NN without dropout for w...	0.009	0.011	0.01
7	Sequential Regression NN without dropout for w...	0.01	0.022	0.022
8	Sequential Regression NN without dropout for w...	0.011	0.013	0.013

  - We have experimented for models with and without dropouts and also different p values in dropout layer
  - Regularization and dropout value of 0.1 worked well for regression as well. Our best model yielded a close to zero MSE for training, validation and test data set



- **Multi headed NN:**
    - Combining the loss of classification and regression model to backpropagate to optimize the model further
        - We trained the model for 50 epochs and saw that both train and validation loss decreased steadily after each epoch
            - At the end of all the epochs, we ended up with a loss close to 0.7



Conclusion:

Our main objective was to classify images of cats and dogs and to identify the location. This fundamental problem in computer vision is the basis of many other computer vision tasks. In the phase, we used neural networks with data augmentation, drop out layers and regularization to make the prediction, which reduced overfitting the model and yielded us with an accuracy of ~60%. In addition to this, we also combined the CXE and MSE loss and then sent it back for optimization of the neural network which helped us reduce the loss steadily after each epoch.

Our next steps will be as follows:

1. Fine tune the existing CNN model by adding dropout layers, batchnorm layer, and maxpooling layer
    a. Also, we plan to experiment with regularization and different types of optimizers
2. Build a transfer learning network using efficient det
3. Perform a t-test to compare the baseline model with the improved model to make sure our improved model performs significantly better than the models from the previous phases