Group 19: Phase 2 - Cats vs Dogs Detector (CatsDoD)

Team Members

We are a group of 4 members

Abhishek Nirmalkar - anirmalkar@ucsb.edu (in picture, left top)

Himanshu Joshi - hjoshi@ucsb.edu (in picture, Right bottom)

Siddarth Chandrasekar - sidchandrasekar@ucsb.edu (in picture, left bottom)

Sasankha Adhikary Pramanik - sasankhapramanik@ucsb.edu (in picture, Right top)

Project Abstract

Image classification is a crucial task in computer vision, with applications ranging from medical diagnosis to autonomous driving. Although this task is relatively easy for humans, it presents a formidable challenge for machines due to the complexity of patterns and features, including substantial background, illumination conditions, deformations, occlusions, and intra-class variations. We aim to build a robust binary classification model that can accurately distinguish between images of cats and dogs, addressing these challenges and ensuring reliable performance.

In our previous phase, we implemented various binary classification models, including Logistic Regression, Linear Regression, and Gaussian Discriminant Analysis. In this phase, we extended the baseline and implemented a complex hue function (CE + MSE) using a homogenous classification pipeline.

Furthermore, we built an improved preprocessing pipeline that combines the previous and final pipeline for classification and regression tasks. We tested various data augmentation, dropout layers and regularization to overcome overfitting. In addition to this, we built a custom Homogenous Linear Regression, Homogenous Logistic Regression, Gaussian Discriminant Analysis and the regularized cost and gradient functions to optimize the complex cost function.

Accuracy and loss are our primary evaluation parameters. In addition to this, we also used classification report (F1 score and precision) and confusion matrix to evaluate our models. Regarding the above model, we get an accuracy of 100% for classification and 0.0000 and 0.0000 loss on the regression.

Project Meta Data

We have completed the following over the week:

- Homogenous Linear Regression
- Complex loss function (CE + MSE) using homogenous models
- Built sequential neural network for image classification and use the following to improve accuracy
  - Data Augmentation
  - Dropout layers
  - Regularization
- Built sequential neural network for binary classification
- Complete the performance of the above models using discovery and test
- Built a multi-headed cat-dog detector using the GDA of the complexes loss function (CE + MSE)
- Built a custom pipeline for data preprocessing for image classification and detection

Data Description

The data set given to us is cats-dogs-hue-dataset. We will be using this for train - test for image and the other for classification.

The images are in the form of .jpg files.

The data set has 2 folders, one for cats and the other for dogs.

- There are about 10k images in the data set. There is a good balance of classes with 5k cat images and 5k dog images in the data set.

Image information (sample size):

- There are 10k 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 dimensions of the images are stored in the variables ht/wd

Architecture of the Neural Network:

- Resize all the images to 128 X 128 so that the image has coordinates
- There are about 10k images in the data set

Ground truths:

- There are two folders (one image per folder) containing the cat and dog images. Hence the ground truth is the name of the folder(cats or dogs)

Task

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

- Image pre-processing (Augmentation)
- Build a custom homogenous classification pipeline
- Use sequential models for image classification
- Neural network classifier for image detection
- Neural network regressor for hue detection
- Implemented the above using CEP and created a combined loss function (CE + MSE)
- Build a convolutional neural network

Since the data set is very large, we have carried out the above tasks were carried out only in a subset of data. Hence the results will be very distributed and not accurate.

Import Data

Unarchive data

```
!unzip -q cats_dogs.zip
!ls -l
!echo "Extracts to extracted_data if this is specified location"
!ls -l extracted_data
!ls -l extracted_data/cats_dogs_hue_dataset
!ls -l extracted_data/cats_dogs_hue_dataset/cats | head -5
!ls -l extracted_data/cats_dogs_hue_dataset/dogs | head -5
!ls extracted_data/cats_dogs_hue_dataset/cats/*.jpg | head -5 | xargs -I{} file {}
!ls extracted_data/cats_dogs_hue_dataset/dogs/*.jpg | head -5 | xargs -I{} file {}
!ls extracted_data/cats_dogs_hue_dataset/cats/*.jpg | head -5 | xargs -I{} identify -format "%f %w x %h\n" {}
!ls extracted_data/cats_dogs_hue_dataset/dogs/*.jpg | head -5 | xargs -I{} identify -format "%f %w x %h\n" {}
```

```
unzip: cannot open cats_dogs.zip
total 0
drwxr-xr-x 1 root root  4096 Jun 13 13:40 sample_data
Extracts to extracted_data if this is specified location
total 0
drwxr-xr-x 3 root root  4096 Jun 13 13:40 extracted_data
total 0
drwxr-xr-x 4 root root  4096 Jun 13 13:40 cats_dogs_hue_dataset
ls: write error: Broken pipe
ls: write error: Broken pipe
ls: write error: Broken pipe
ls: write error: Broken pipe
ls: write error: Broken pipe
ls: write error: Broken pipe
ls: write error: Broken pipe
ls: write error: Broken pipe
ls: write error: Broken pipe
ls: write error: Broken pipe
```

Load bounding box meta data

```
df = pd.read_csv("bounding_box.csv")
df.shape
df.head()
```

	image1	bounds	confidence	conf0	conf1	conf2	conf3	conf4	conf5	conf6	conf7	conf8	conf9
0	00000cat.jpg	1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	00000dog.jpg	1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2	00001cat.jpg	1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
3	00001dog.jpg	1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
4	00002cat.jpg	1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

5 rows × 21 columns

Preprocess

Rescale the images

```
!mkdir -p resized_images
```

```
# Cleanup
!rm -rf resized_images/*
# Create directory structure for resized images
!find extracted_data/cats_dogs_hue_dataset -type d | head -20
# Create directories
!find extracted_data/cats_dogs_hue_dataset -type d | sed 's|extracted_data|resized_images|' | head -20 | xargs -I{} mkdir -p {}
# Verify directory structure
!find resized_images -type d
# Count files in source
!ls extracted_data/cats_dogs_hue_dataset/cats/*.jpg | wc -l
!ls extracted_data/cats_dogs_hue_dataset/dogs/*.jpg | wc -l
```

```
rm: cannot remove 'resized_images/*': No such file or directory
extracted_data/cats_dogs_hue_dataset
extracted_data/cats_dogs_hue_dataset/cats
extracted_data/cats_dogs_hue_dataset/dogs
resized_images
resized_images/cats_dogs_hue_dataset
resized_images/cats_dogs_hue_dataset/cats
resized_images/cats_dogs_hue_dataset/dogs
5000
5000
```

Checkpoint and Save data

```
import h5py
import numpy as np
from sklearn.model_selection import train_test_split
from tqdm.notebook import tqdm
from PIL import Image, ImageOps
import os
import glob

# Parameters
IMG_SIZE = 128
BATCH_SIZE = 32
DATA_DIR = "resized_images/cats_dogs_hue_dataset"
CHECKPOINT_FILE = "preprocessed_data.h5"

# Function to load and preprocess images
def load_image(path, size=IMG_SIZE):
    try:
        img = Image.open(path)
        if img.mode != 'RGB':
            img = img.convert('RGB')
        img = ImageOps.exif_transpose(img)
        img = img.resize((size, size))
        return np.array(img, dtype=np.uint8)
    except Exception as e:
        print(f"Error loading {path}: {e}")
        return None

# Load cat images
print("Loading cat images...")
cat_paths = glob.glob(os.path.join(DATA_DIR, 'cats', '*.jpg'))
cat_images = []
for path in tqdm(cat_paths):
    img = load_image(path)
    if img is not None:
        cat_images.append(img)
cat_images = np.array(cat_images)
print(f"Loaded {len(cat_images)} cat images")

# Load dog images
print("Loading dog images...")
dog_paths = glob.glob(os.path.join(DATA_DIR, 'dogs', '*.jpg'))
dog_images = []
for path in tqdm(dog_paths):
    img = load_image(path)
    if img is not None:
        dog_images.append(img)
dog_images = np.array(dog_images)
print(f"Loaded {len(dog_images)} dog images")

# Combine and create labels
X = np.concatenate([cat_images, dog_images])
y = np.concatenate([np.zeros(len(cat_images)), np.ones(len(dog_images))])

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Save to checkpoint file
print("Saving to checkpoint file...")
with h5py.File(CHECKPOINT_FILE, 'w') as f:
    f.create_dataset('X_train', data=X_train, compression='gzip')
    f.create_dataset('X_test', data=X_test, compression='gzip')
    f.create_dataset('y_train', data=y_train)
    f.create_dataset('y_test', data=y_test)
    f.attrs['IMG_SIZE'] = IMG_SIZE
    f.attrs['BATCH_SIZE'] = BATCH_SIZE

print("Checkpoint saved!")
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
```

```
Loading cat images...
Loaded 5000 cat images
Loading dog images...
Loaded 5000 dog images
Saving to checkpoint file...
Checkpoint saved!
X_train shape: (8000, 128, 128, 3)
X_test shape: (2000, 128, 128, 3)
```

Baseline in SKLearn

Metrics for evaluation:

- Accuracy = (True Cats + True Dogs) / Total Images
- Cat Precision = True Cats / (True Cats + False Dogs)
- Cat Recall = True Cats / (True Cats + False Cats)
- Dog Precision = True Dogs / (True Dogs + False Cats)
- Dog Recall = True Dogs / (True Dogs + False Dogs)
- Confusion matrix = [[True Cats, False Dogs], [False Cats, True Dogs]]

Loss function

- Cross-entropy loss function for classification and MSE for regression when used as hue function for the neural network

Load data

```
# Load data
import h5py
import numpy as np

print("Loading data from checkpoint...")
with h5py.File('preprocessed_data.h5', 'r') as f:
    X_train = f['X_train'][:]
    X_test = f['X_test'][:]
    y_train = f['y_train'][:]
    y_test = f['y_test'][:]
    IMG_SIZE = f.attrs['IMG_SIZE']
    BATCH_SIZE = f.attrs['BATCH_SIZE']

print("Data loaded!")
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
print(f"IMG_SIZE: {IMG_SIZE}")
print(f"BATCH_SIZE: {BATCH_SIZE}")
```

Double check that it loaded correctly.

```
# Quick check
print("Checking data...")
print(f"Cat images: {sum(y_train == 0)}")
print(f"Dog images: {sum(y_train == 1)}")
print(f"Pixel range: {X_train.min()} - {X_train.max()}")
print(f"Ready to proceed!")
```

```
Loading data from checkpoint...
Data loaded!
X_train shape: (8000, 128, 128, 3)
X_test shape: (2000, 128, 128, 3)
y_train shape: (8000,)
y_test shape: (2000,)
IMG_SIZE: 128
BATCH_SIZE: 32
Checking data...
Cat images: 4000
Dog images: 4000
Pixel range: 0 - 255
Ready to proceed!
```

Homogenous CKE + MSE

We build a homogenous logistic regression model and a homogenous linear regression model and combine the CKE loss at each epoch. Similarly, we add the MSE loss for the regression model and combine it with the previous loss. We use the combined loss to update the weights.

The cross-from both models are then combined to give us a complex loss function CE + MSE which says that used to make predictions.

Homogenous Linear Regression

Linear Regression Loss Function

The mean square function formula is as follows: $\text{MSE}(\mathbf{X}, \mathbf{y}) = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{x}^{(i)} \cdot \mathbf{w} - y^{(i)})^2$

where m is the number of all data points, $\mathbf{y}$ is the predicted value

Split data

```
import numpy
import matplotlib
from sklearn.model_selection import X_shape=(1, 784)
X_train = X_train.reshape(-1, 784)
X_test = X_test.reshape(-1, 784)

# Binary: 0 = cats, 1 = dogs
y_train = (y_train == 1).astype(int)
y_test = (y_test == 1).astype(int)

# Create validation set from training data
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.1, random_state=42)

print("Shapes after reshaping:")
print(f"X_train shape: {X_train.shape}")
print(f"X_val shape: {X_val.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_val shape: {y_val.shape}")
print(f"y_test shape: {y_test.shape}")

from sklearn.preprocessing import StandardScaler, MinMaxScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
print("Data standardized!")
print(f"Mean: {X_train.mean()}")
print(f"Std: {X_train.std()}")
```

```
Shapes after reshaping:
X_train shape: (7200, 784)
X_val shape: (800, 784)
X_test shape: (2000, 784)
y_train shape: (7200,)
y_val shape: (800,)
y_test shape: (2000,)
Data standardized!
Mean: 0.0000000000000000
Std: 1.0000000000000000
```

Train

```
class LinearRegressor:
    def __init__(self, learning_rate=0.01, lambda_=0.0):
        self.learning_rate = learning_rate
        self.lambda_ = lambda_
        self.w = None
        self.b = None
        self.cost_history = []
        self.val_cost_history = []

    def compute_cost(self, X, y, w, b):
        m = X.shape[0]
        predictions = X @ w + b
        cost = np.sum((predictions - y) ** 2) / (2 * m)
        if self.lambda_ > 0:
            reg = self.lambda_ * np.sum(w ** 2) / (2 * m)
            cost += reg
        return cost

    def compute_gradients(self, X, y, w, b):
        m = X.shape[0]
        predictions = X @ w + b
        error = predictions - y
        dw = (X.T @ error) / m
        if self.lambda_ > 0:
            dw += (self.lambda_ / m) * w
        db = np.sum(error) / m
        return dw, db

    def fit(self, X, y, X_val=None, y_val=None, epochs=1000, verbose=True):
        n_features = X.shape[1]
        self.w = np.zeros(n_features)
        self.b = 0.0
        for epoch in range(epochs):
            dw, db = self.compute_gradients(X, y, self.w, self.b)
            self.w = self.w - self.learning_rate * dw
            self.b = self.b - self.learning_rate * db
            cost = self.compute_cost(X, y, self.w, self.b)
            self.cost_history.append(cost)
            if X_val is not None:
                val_cost = self.compute_cost(X_val, y_val, self.w, self.b)
                self.val_cost_history.append(val_cost)
            if verbose and (epoch % 100 == 0 or epoch == epochs - 1):
                print(f"Epoch {epoch}: Cost {cost:.4f}")
        return self

    def predict(self, X):
        return X @ self.w + self.b

    def predict_classes(self, X, threshold=0.5):
        return (self.predict(X) >= threshold).astype(int)

    def score(self, X, y):
        predictions = self.predict_classes(X)
        return np.mean(predictions == y)

print("Training Linear Regressor...")
model = LinearRegressor(learning_rate=0.01, lambda_=0.1)
model.fit(X_train, y_train, X_val, y_val, epochs=1000, verbose=True)

print("Training complete!")
print(f"Final weights shape: {model.w.shape}")
print(f"Final bias: {model.b}")
```

```
Training Linear Regressor...
Epoch 0: Cost 0.5000
Epoch 100: Cost 0.4999
Epoch 200: Cost 0.4999
Epoch 300: Cost 0.4999
Epoch 400: Cost 0.4999
Epoch 500: Cost 0.4999
Epoch 600: Cost 0.4999
Epoch 700: Cost 0.4999
Epoch 800: Cost 0.4999
Epoch 900: Cost 0.4999
Epoch 999: Cost 0.4999
Training complete!
Final weights shape: (784,)
Final bias: 0.0
```

Evaluation

Make Classes

- MSE gives how close our predicted regression line is to the data points.
- R^2 tells us how well our model explains the variance.
  - $R^2 = 1$ means perfect fit, $R^2 = 0$ means model predicts mean

```
class LinearRegressor(LinearRegressor):
    def predict_classes(self, X, threshold=0.5):
        return (self.predict(X) >= threshold).astype(int)

print("Linear Regressor Evaluation:")
print("="*50)
print(f"Training accuracy: {model.score(X_train, y_train)}")
print(f"Validation accuracy: {model.score(X_val, y_val)}")
print(f"Test accuracy: {model.score(X_test, y_test)}")
```

```
Linear Regressor Evaluation:
==================================================
Training accuracy: 0.5000000000000000
Validation accuracy: 0.5000000000000000
Test accuracy: 0.5000000000000000
```

Homogenous implementation of Logistic Regression

Implemented a homogenous Logistic Regression model. Created the loss function from CE to CE + MSE. ie. Added the complexes loss function into the function that finally updates the data and backpropting the coefficients at the same time.

```
from sklearn.preprocessing import StandardScaler

# Create scaler
scaler = StandardScaler()

# Fit on training data and transform
X_train = scaler.fit_transform(X_train)

# Transform validation and test data
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

print("Data standardized!")
print(f"Mean: {X_train.mean()}")
print(f"Std: {X_train.std()}")
```

```
Data standardized!
Mean: 0.0000000000000000
Std: 1.0000000000000000
```

Homogenous Classification of Logistic Regression

Implement a homogenous Logistic Regression model. Extend the loss function from CE to CE + MSE. ie. Add the complexes loss function into the function that finally updates the data and backpropting the coefficients at the same time.

```
class LogisticRegressor:
    def __init__(self, learning_rate=0.01, lambda_=0.0):
        self.learning_rate = learning_rate
        self.lambda_ = lambda_
        self.w = None
        self.b = None
        self.cost_history = []
        self.val_cost_history = []

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-np.clip(z, -500, 500)))

    def compute_cost(self, X, y, w, b):
        m = X.shape[0]
        predictions = X @ w + b
        probs = self.sigmoid(predictions)
        probs = np.clip(probs, 1e-15, 1-1e-15)
        cost = -np.sum(y * np.log(probs) + (1 - y) * np.log(1 - probs)) / m
        if self.lambda_ > 0:
            reg = self.lambda_ * np.sum(w ** 2) / (2 * m)
            cost += reg
        return cost

    def compute_gradients(self, X, y, w, b):
        m = X.shape[0]
        predictions = X @ w + b
        probs = self.sigmoid(predictions)
        error = probs - y
        dw = (X.T @ error) / m
        if self.lambda_ > 0:
            dw += (self.lambda_ / m) * w
        db = np.sum(error) / m
        return dw, db

    def fit(self, X, y, X_val=None, y_val=None, epochs=1000, verbose=True):
        n_features = X.shape[1]
        self.w = np.zeros(n_features)
        self.b = 0.0
        for epoch in range(epochs):
            dw, db = self.compute_gradients(X, y, self.w, self.b)
            self.w = self.w - self.learning_rate * dw
            self.b = self.b - self.learning_rate * db
            cost = self.compute_cost(X, y, self.w, self.b)
            self.cost_history.append(cost)
            if X_val is not None:
                val_cost = self.compute_cost(X_val, y_val, self.w, self.b)
                self.val_cost_history.append(val_cost)
            if verbose and (epoch % 100 == 0 or epoch == epochs - 1):
                print(f"Epoch {epoch}: Cost {cost:.4f}")
        return self

    def predict(self, X):
        return self.sigmoid(X @ self.w + self.b)

    def predict_classes(self, X, threshold=0.5):
        return (self.predict(X) >= threshold).astype(int)

    def score(self, X, y):
        predictions = self.predict_classes(X)
        return np.mean(predictions == y)

print("Training Logistic Regressor...")
model = LogisticRegressor(learning_rate=0.01, lambda_=0.1)
model.fit(X_train, y_train, X_val, y_val, epochs=1000, verbose=True)

print("Training complete!")
print(f"Final weights shape: {model.w.shape}")
print(f"Final bias: {model.b}")
```

```
Training Logistic Regressor...
Epoch 0: Cost 0.6931
Epoch 100: Cost 0.6931
Epoch 200: Cost 0.6931
Epoch 300: Cost 0.6931
Epoch 400: Cost 0.6931
Epoch 500: Cost 0.6931
Epoch 600: Cost 0.6931
Epoch 700: Cost 0.6931
Epoch 800: Cost 0.6931
Epoch 900: Cost 0.6931
Epoch 999: Cost 0.6931
Training complete!
Final weights shape: (784,)
Final bias: 0.0
```

Evaluation

Make Classes

```
class LogisticRegressor(LogisticRegressor):
    def predict_classes(self, X, threshold=0.5):
        return (self.predict(X) >= threshold).astype(int)

print("Logistic Regressor Evaluation:")
print("="*50)
print(f"Training accuracy: {model.score(X_train, y_train)}")
print(f"Validation accuracy: {model.score(X_val, y_val)}")
print(f"Test accuracy: {model.score(X_test, y_test)}")
```

```
Logistic Regressor Evaluation:
==================================================
Training accuracy: 0.5000000000000000
Validation accuracy: 0.5000000000000000
Test accuracy: 0.5000000000000000
```

```
# Generic imports
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Keras imports
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.utils.np_utils import to_categorical
from keras.optimizers import SGD, RMSprop, Adam
from keras.utils.visualize_util import plot

# Scikit-learn imports
from sklearn.cross_validation import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# Load the training data
train = np.genfromtxt('./data/train.csv', delimiter=',', dtype='int8', skiprows=1)

# Load the test data
test = np.genfromtxt('./data/test.csv', delimiter=',', dtype='int8', skiprows=1)

# Print the training data shape
print("Training data shape: ", train.shape)

# Print the test data shape
print("Test data shape: ", test.shape)
```

```
Training data shape:  (42000, 785)
Test data shape:  (28000, 784)
```

```
# Separate the labels and the pixel values for the training data
train_labels = train[:,0]
train_pixels = train[:,1:]

# Separate the pixel values for the test data
test_pixels = test[:,:]

# Reshape the training data into 42000 2d arrays of 28x28 pixels
train_pixels = train_pixels.reshape(-1,1,28,28)

# Reshape the test data into 28000 2d arrays of 28x28 pixels
test_pixels = test_pixels.reshape(-1,1,28,28)

# Convert the training labels into categorical vectors
train_labels = to_categorical(train_labels)

# Print shape of training labels
print("Training labels shape: ", train_labels.shape)

# Print shape of test pixels
print("Test pixels shape: ", test_pixels.shape)
```

```
Training labels shape:  (42000, 10)
Test pixels shape:  (28000, 1, 28, 28)
```

```
# Create a Sequential model
model = Sequential()

# Add a convolutional layer with 64 feature maps, 5x5 convolutions, and a rectifier activation function
model.add(Convolution2D(64, 5, 5, activation='relu'))

# Add a max pooling layer with a 2x2 pool size
model.add(MaxPooling2D(pool_size=(2,2)))

# Add a convolutional layer with 32 feature maps, 5x5 convolutions, and a rectifier activation function
model.add(Convolution2D(32, 5, 5, activation='relu'))

# Add a max pooling layer with a 2x2 pool size
model.add(MaxPooling2D(pool_size=(2,2)))

# Add a flatten layer
model.add(Flatten())

# Add a dense layer with 256 outputs and a rectifier activation function
model.add(Dense(256, activation='relu'))

# Add a dropout layer with a 50% probability
model.add(Dropout(0.5))

# Add a dense layer with 10 outputs and a softmax activation function
model.add(Dense(10, activation='softmax'))

# Compile the model with categorical cross-entropy loss and the Adam optimizer
model.compile(loss='categorical_crossentropy', optimizer='adam')

# Print a summary of the model
model.summary()
```

```
Using Theano backend.
Using gpu device 0: GeForce GTX 980 (CNMeM is disabled, cuDNN 5005)
/home/ubuntu/anaconda2/lib/python2.7/site-packages/theano/sandbox/cuda/dnn.py:5
```

```
Precision TP/(TP+FP)=0.9999999999999999
Recall TP/(TP+FN)=0.9999999999999999
F1-Score 2*(Precision*Recall)/(Precision+Recall)=0.9999999999999999
                    precision    recall  f1-score   support

           Normal       1.00      1.00      1.00       252
Pneumonia_Virus_Bacteria       1.00      1.00      1.00       252

    accuracy                           1.00       504
   macro avg       1.00      1.00      1.00       504
weighted avg       1.00      1.00      1.00       504
```

```
Epoch 2/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 3/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 4/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 5/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 6/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 7/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 8/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 9/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 10/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 11/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 12/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 13/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 14/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 15/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 16/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 17/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 18/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 19/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 20/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 21/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 22/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 23/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 24/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 25/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 26/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 27/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 28/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 29/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 30/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 31/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 32/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 33/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 34/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 35/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 36/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 37/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 38/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 39/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 40/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 41/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 42/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 43/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 44/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 45/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 46/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 47/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 48/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 49/50
Training Loss: 0.5000 Validation Loss: 0.5000
Epoch 50/50
Training Loss: 0.5000 Validation Loss: 0.5000
```

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(history_dropout.history['loss'], label='Dropout Training Loss')
plt.plot(history_dropout.history['val_loss'], label='Dropout Validation Loss')
plt.title('Dropout Regularization Effect')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5).astype(int)
y_pred[:10]
```

```
array([[1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1]])
```

```
y_test[:10]
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\n Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\n Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.5000

 Confusion Matrix:
 [[  0 250]
 [  0 250]]

 Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       250
           1       0.50      1.00      0.67       250

    accuracy                           0.50       500
   macro avg       0.25      0.50      0.33       500
weighted avg       0.25      0.50      0.33       500
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prof(average, modifier, msg_start, len(result))
```

```
Epoch 18/20
Epoch 19/20
Epoch 20/20
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320
_________________________________________________________________
 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )
_________________________________________________________________
 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)
_________________________________________________________________
 conv2d_2 (Conv2D)           (None, 3, 3, 64)          36864
_________________________________________________________________
 flatten (Flatten)           (None, 576)               0
_________________________________________________________________
 dense (Dense)               (None, 64)                36928
_________________________________________________________________
 dense_1 (Dense)             (None, 10)                650
=================================================================
Total params: 93,258
Trainable params: 93,258
Non-trainable params: 0
_________________________________________________________________
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0313 - accuracy: 0.9901
[0.03129, 0.9901]
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320
_________________________________________________________________
 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )
_________________________________________________________________
 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)
_________________________________________________________________
 conv2d_2 (Conv2D)           (None, 3, 3, 64)          36864
_________________________________________________________________
 flatten (Flatten)           (None, 576)               0
_________________________________________________________________
 dense (Dense)               (None, 64)                36928
_________________________________________________________________
 dense_1 (Dense)             (None, 10)                650
=================================================================
Total params: 93,258
Trainable params: 93,258
Non-trainable params: 0
_________________________________________________________________
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0313 - accuracy: 0.9901
[0.03129, 0.9901]
```

```
313/313 [==============================] - 1s 2ms/step
```

```
array([[0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ,
        1.        , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 1.        , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.9999999 , 0.0000000 , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.9999999 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.9999999 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 0.0000000 , 0.9999999 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.9999999 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.9999999 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.9999999 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.9999999 ]]
      dtype=float32)
```

```
array([5, 2, 1, 0, 4, 6, 7, 8, 9, 3])
```

```
array([5, 2, 1, 0, 4, 6, 7, 8, 9, 3])
```

```
array([[ 0,  0,  0,  0,  0,  5,  0,  0,  0,  0],
       [ 0,  0,  2,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  1,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 1,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  1,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  1,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  1,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  1,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  1,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  1]]
```

With Regularization and Without Dropout

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320
_________________________________________________________________
 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )
_________________________________________________________________
 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)
_________________________________________________________________
 conv2d_2 (Conv2D)           (None, 3, 3, 64)          36864
_________________________________________________________________
 flatten (Flatten)           (None, 576)               0
_________________________________________________________________
 dense (Dense)               (None, 64)                36928
_________________________________________________________________
 dense_1 (Dense)             (None, 10)                650
=================================================================
Total params: 93,258
Trainable params: 93,258
Non-trainable params: 0
_________________________________________________________________
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0313 - accuracy: 0.9901
[0.03129, 0.9901]
```

```
313/313 [==============================] - 1s 2ms/step
```

```
array([[0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ,
        1.        , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 1.        , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.9999999 , 0.0000000 , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.9999999 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.9999999 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 0.0000000 , 0.9999999 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.9999999 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.9999999 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.9999999 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.9999999 ]]
      dtype=float32)
```

```
array([5, 2, 1, 0, 4, 6, 7, 8, 9, 3])
```

```
array([5, 2, 1, 0, 4, 6, 7, 8, 9, 3])
```

```
array([[ 0,  0,  0,  0,  0,  5,  0,  0,  0,  0],
       [ 0,  0,  2,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  1,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 1,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  1,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  1,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  1,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  1,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  1,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  1]]
```

With Dropout

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320
_________________________________________________________________
 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )
_________________________________________________________________
 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)
_________________________________________________________________
 conv2d_2 (Conv2D)           (None, 3, 3, 64)          36864
_________________________________________________________________
 flatten (Flatten)           (None, 576)               0
_________________________________________________________________
 dense (Dense)               (None, 64)                36928
_________________________________________________________________
 dense_1 (Dense)             (None, 10)                650
=================================================================
Total params: 93,258
Trainable params: 93,258
Non-trainable params: 0
_________________________________________________________________
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0313 - accuracy: 0.9901
[0.03129, 0.9901]
```

```
313/313 [==============================] - 1s 2ms/step
```

```
array([[0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ,
        1.        , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 1.        , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.9999999 , 0.0000000 , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.9999999 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.9999999 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 0.0000000 , 0.9999999 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.9999999 , 0.0000000 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.9999999 , 0.0000000 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.9999999 , 0.0000000 ],
       [0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 ,
        0.0000000 , 0.0000000 , 0.0000000 , 0.0000000 , 0.9999999 ]]
      dtype=float32)
```

```
array([5, 2, 1, 0, 4, 6, 7, 8, 9, 3])
```

```
array([5, 2, 1, 0, 4, 6, 7, 8, 9, 3])
```

```
array([[ 0,  0,  0,  0,  0,  5,  0,  0,  0,  0],
       [ 0,  0,  2,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  1,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 1,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  1,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  1,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  1,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  1,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  1,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  1]]
```

```
Recording: 1.0 [00:00:00]
Recording: 1.0 [00:00:00]
Recording: 1.0 [00:00:00]
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Unzipping corpus/stopwords.zip.
[nltk_data] Downloading package punkt_tokenizer to /usr/share/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tokenizer.zip.
[nltk_data] Downloading package wordnet to /usr/share/nltk_data...
[nltk_data]   Unzipping corpus/wordnet.zip.
[nltk_data] Downloading package averaged_perceptron_tagger_eng to /usr/share/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
```

	clean_name	Train Acc	Valid Acc	Test Acc
0	Sequential Neural Network without Regularization	0.9	0.8889	0.8889
1	Sequential Neural Network with Dropout (p = 0.2)	0.8667	0.8667	0.7778
2	Sequential Neural Network with Dropout (p = 0.5)	0.7667	0.7667	0.8889
3	Sequential Neural Network with Dropout (p = 0.8)	0.7667	0.7667	0.8889
4	Neural Network with Dropout (p = 0.2)	0.8667	0.8667	0.7778
5	Neural Network with Dropout (p = 0.5)	0.8667	0.8667	0.8889

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Unzipping corpus/stopwords.zip.
[nltk_data] Downloading package punkt_tokenizer to /usr/share/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tokenizer.zip.
[nltk_data] Downloading package wordnet to /usr/share/nltk_data...
[nltk_data]   Unzipping corpus/wordnet.zip.
[nltk_data] Downloading package averaged_perceptron_tagger_eng to /usr/share/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
```

Results and Discussion

- Without regularization, we see that though we get training accuracy of more than 90%, the validation accuracy of only 88%, which clearly suggesting the model is overfitting.

	clean_name	Train Acc	Valid Acc	Test Acc
0	Sequential Neural Network without Regularization	0.9	0.8889	0.8889
1	Neural Network without Regularization and with R...	98.325	98.467	97.106
2	Neural Network without Regularization and Dropout	79.792	57.511	59.925
3	Neural Network without Dropout	98.792	98.594	97.475
4	Neural Network with Dropout (p = 0.2)	93.83	57.944	58.258
5	Neural Network with Dropout (p = 0.1)	98.83	99.088	99.283

- Adding regularization still does the best accuracy but not to a larger extent
- Dropout regularization does not help in improving generalization, adding dropout hurts our accuracy
- The best model with a dropout layer of 0.1 yielded us a training accuracy of 97% and validation and test accuracy of ~90%
- We can conclude that the best model is a NN with no dropout and with L2 regularization

A similar process was followed for DOP as well

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Unzipping corpus/stopwords.zip.
[nltk_data] Downloading package punkt_tokenizer to /usr/share/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tokenizer.zip.
[nltk_data] Downloading package wordnet to /usr/share/nltk_data...
[nltk_data]   Unzipping corpus/wordnet.zip.
[nltk_data] Downloading package averaged_perceptron_tagger_eng to /usr/share/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
```

```
Using TensorFlow backend.
```

```
Using TensorFlow backend.
```

```
Using TensorFlow backend
```

Multiheaded CXE + MSE Network

- Keras is used for model
- Keras is used for data augmentation
- Contact a model for classification and regression separately
- For each search, we calculated CXE and MSE for the above models and then used the combined loss for back propagation
- Model optimizer was used to write the best new and update the weights and bias

```
!apt install tree
!pip install -q -U keras_tuner
!pip install -q -U keras-core
!pip install -q -U keras-nlp keras-tuner dm-sonnet trfl dm-tree rlax chex
!pip install -q -U keras-cv keras-tfd keras-ncf keras-rsa keras-drn keras-cfr keras-gcn keras-pwcnet keras-quantreg keras-quantile-regression
!pip install -q -U keras-lambda keras-gmm keras-wrn keras-vbmi keras-band keras-lr2u keras-mixup keras-layer-kaleidoscope
!pip install -q -U git+https://github.com/keras-team/keras-contrib.git git+https://github.com/keras-team/keras-preprocessing git+https://github.com/keras-team/keras-applications git+https://github.com/keras-team/keras git+https://github.com/tensorflow/models git+https://github.com/google/automl git+https://github.com/google/keras-efficient-ps git+https://github.com/qubvel/segmentation_models git+https://github.com/ypwhs/keras_yolo3 git+https://github.com/broadinstitute/keras-resnet git+https://github.com/keras-team/keras-contrib
```

Result and Discussions

- Considering the use of cumulative sum regression model to backpropagate the loss to improve the model
- 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 obtained an MSE score close to 0.7

Conclusion:

Our main objective was to classify images of cats and dogs and to identify the breeds. This fundamental problem in computer vision has been tackled using various approaches. We used a CNN model with batch normalization layers and dropout layers for regularization to make the predictions, which improved performing the model and yielded a val acc accuracy of .3875. In addition to this, we also commented the Cat and Dog MNIST data and then ran it back for optimization of the neural network which helped us reduce the loss and improve the accuracy.

Our next steps will be as follows:

- Fine-tune the existing CONV model by adding dropout layers, batchnorm layer, and reassembling layer
  - By this, we aim to experiment with regularization and different types of optimizers
- Train the model for more epochs and more data
- Pick-n-Lock is a game to compare the baseline model with the improved model to make sure our improved model performs significantly better than the baseline from the previous phase