Group 2 Phase 3 - Cats vs Dogs Detector (CaDoD)

"FrankenNet" convolutional neural network

Team Members

- Ben Perkins
- Lauren Madar
- Mangesh Walimbe
- Samin Barghan

Project Abstract

One of the fundamental tasks in classifying images is object detection within images. Algorithms often employ a 'bounding box' tool. To study bounding boxes, our team first evaluated 3 models with GridSearchCV, to be trained on existing bounding box data for the purpose of predicting bounding boxes. The best model was used for bounding box prediction.

After exploring linear and logistic regression models with SciKit Learn, we created several PyTorch models with classification and bounding box predictions using Cross Entropy and Mean Squared Error loss functions. PyTorch allows for a much simpler modeling, training and prediction process.

Project Description

The purpose of this project is create an end to end process in machine learning to create an object classifier and bounding box predictor for cat and dog images. There are about 13,000 images of varying shapes and aspect ratios. They are all RGB images and have bounding box coordinates stored in a .csv file. In order to create a detector, we will first have to preprocess the images to be all of the same shapes, and flatten them from a 3D array to 2D. Then we will feed this array into a linear localization predictor and a logistic regression model to predict labels and bounding boxes.

Data Description

The image archive `cadod.tar.gz` is a subset Open Images V6. It contains a total of 12,966 images of dogs and cats.

Image bounding boxes are stored in the csv file `cadod.csv`. The following describes whats contained inside the csv.

- ImageID: the image this box lives in.
- Source: indicates how the box was made:
  - xclick are manually drawn boxes using the method presented in [1], were the annotators click on the four extreme points of the object. In V6 we release the actual 4 extreme points for all xclick boxes in train (13M), see below.
  - activemil are boxes produced using an enhanced version of the method [2]. These are human verified to be accurate at IoU>0.7.
- LabelName: the MID of the object class this box belongs to.
- Confidence: a dummy value, always 1.
- XMin, XMax, YMin, YMax: coordinates of the box, in normalized image coordinates. XMin is in [0,1], where 0 is the leftmost pixel, and 1 is the rightmost pixel in the image. Y coordinates go from the top pixel (0) to the bottom pixel (1).
- XClick1X, XClick2X, XClick3X, XClick4X, XClick1Y, XClick2Y, XClick3Y, XClick4Y: normalized image coordinates (as XMin, etc.) of the four extreme points of the object that produced the box using [1] in the case of xclick boxes. Dummy values of -1 in the case of activemil boxes.

The attributes have the following definitions:

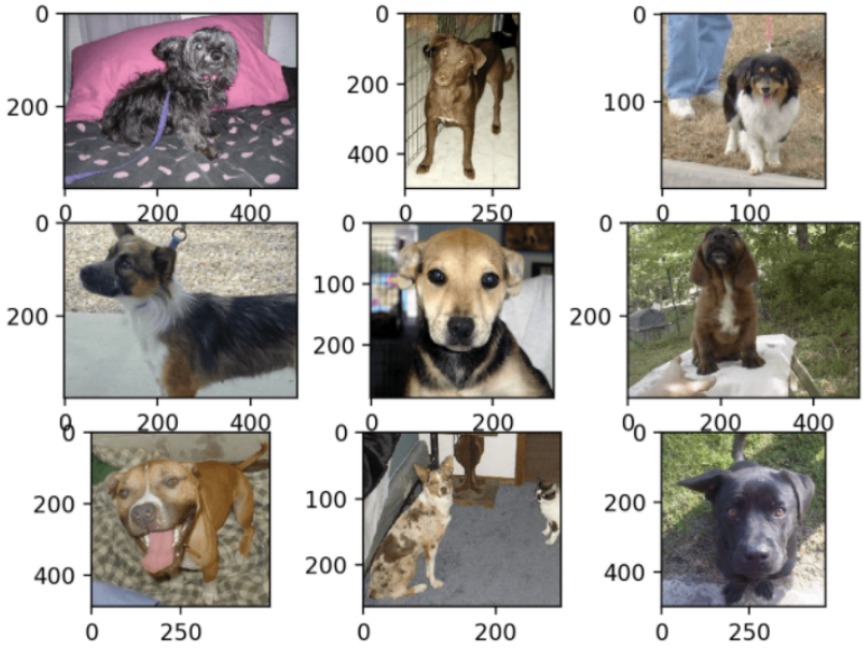
- IsOccluded: Indicates that the object is occluded by another object in the image.
- IsTruncated: Indicates that the object extends beyond the boundary of the image.
- IsGroupOf: Indicates that the box spans a group of objects (e.g., a bed of flowers or a crowd of people). We asked annotators to use this tag for cases with more than 5 instances which are heavily occluding each other and are physically touching.
- IsDepiction: Indicates that the object is a depiction (e.g., a cartoon or drawing of the object, not a real physical instance).
- IsInside: Indicates a picture taken from the inside of the object (e.g., a car interior or inside of a building). For each of them, value 1 indicates present, 0 not present, and -1 unknown.

Identifying columns: ImageID, Source, LabelName, Confidence

Dimensional and positional columns: XMin, XMax, YMin,YMax, XClick1X, XClick2X, XClick3X, XClick4X, XClick1Y, XClick2Y, XClick3Y, XClick4Y

Bounding box and image descriptive columns: IsOccluded, IsTruncated, IsGroupOf, IsDepiction, IsInside

When we look at a few random images, we can see that the photos vary in color and have different shapes and sizes. Also, we can see a photo with both a cat and dog, with the cat being barely visible (bottom row middle) so this shows any classifier fit on this type of photos will have to be robust.



The first step to prepare data must be to standardize the images. Photos will have to be reshaped before modeling so that all images have the same shape and size. One approach we may use would be to load all photos and look at the distribution of the photo widths and heights then determine a new image size that fits the majority of the images. Smaller size allows a model to train more quickly. Another approach would be to start with a fixed size of 200x200 pixels. We can also filter color images to determine where the majority or highest density of each color pixel lies within the image.

The metadata contained in the csv file will need to be matched to each image file, and during Exploratory Data Analysis, we will determine relationships between any of the columns using pandas. For example, how many images contain more than one cat or dog (IsGroupOf)? How many of those images have IsOccluded, IsTruncated, IsInside? Can we determine if the bounding box of one object is larger than the other in order to guess the 'main' object? This will drive creation of additional features.

The code and project files are stored in a GitHub repository: i526Sp21Group2 (SEE PDF). We will impute missing data and document the strategy used, if needed (depending on the results of EDA). NumPy DataFrames embedded in our project Jupyter Notebook will track our exploration and transformation of data and engineering of any features ahead of training and fitting. Other Python libraries may be used for visualizations and will be documented.

```
# Import general modules
import os
import glob
from time import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
import tarfile
from tqdm.notebook import tqdm
from PIL import Image
import warnings

from collections import Counter
import seaborn as sns

# PyTorch Modules
import torch
import torch.nn
import torch.nn.functional as F
from torch.nn import ReLU
from torch import nn, optim
from torch.optim import Adam
from torch.autograd import Variable
import torch.utils
from torch.utils.data import Dataset as torchDataset
from torch.utils.data import DataLoader as torchDataLoader
from torch.utils.data.sampler import SubsetRandomSampler
from torch.utils.tensorboard import SummaryWriter
import torchvision
from torchvision import datasets, transforms, utils
from torchvision.io import read_image
```

```
def extract_tar(file, path):
    """
```

```
    function to extract tar.gz files to specified location

    Args:
        file (str): path where the file is located
        path (str): path where you want to extract
    """
    with tarfile.open(file) as tar:
        files_extracted = 0
        for member in tqdm(tar.getmembers()):
            if os.path.isfile(path + member.name[1:]):
                continue
            else:
                tar.extract(member, path)
                files_extracted += 1
        tar.close()
        if files_extracted < 3:
            print('Files already exist')
```

Import Data

Unarchive data

Place the cadod.tar.gz into the same folder as this notebook. We've already extracted the files into the ./data folder (to prevent committing the large gz file to github).

```
useLocalInJupyter = False # set to False if using Colab, set to True if using locally

# UPDATE PATHS BELOW depending on your GDrive or local setup.

if useLocalInJupyter == False:
    path = '/content/drive/MyDrive/Colab Notebooks/data/cadod/'
    extract_tar('/content/drive/MyDrive/Colab Notebooks/data/cadod.tar.gz', path)

if useLocalInJupyter == True:
    path = './data/cadod/'
    extract_tar('./data/cadod.tar.gz', path)
```

```
100%                                        25936/25936 [00:05<00:00, 5052.80it/s]

Files already exist
```

```
df = pd.read_csv(path + '../cadod.csv')
```

```
df.head()
```

	ImageID	Source	LabelName	Confidence	XMin	XMax	YMin	YMax	IsOccluded	IsTruncated	IsGroupOf	IsDepiction	IsInside	XClickI
0	0000b9fcba019d36	xclick	/m/0bt9lr	1	0.165000	0.903750	0.268333	0.998333	1	1	0	0	0	0.63625
1	0000cb13febe0138	xclick	/m/0bt9lr	1	0.000000	0.651875	0.000000	0.999062	1	1	0	0	0	0.3125(
2	0005a9520eb22c19	xclick	/m/0bt9lr	1	0.094167	0.611667	0.055626	0.998736	1	1	0	0	0	0.4875(
3	0006303f02219b07	xclick	/m/0bt9lr	1	0.000000	0.999219	0.000000	0.998824	1	1	0	0	0	0.50859
4	00064d23bf997652	xclick	/m/0bt9lr	1	0.240938	0.906183	0.000000	0.694286	0	0	0	0	0	0.67803

```
df.LabelName.unique()
```

```
array(['/m/0bt9lr', '/m/01yrx'], dtype=object)
```

Exploratory Data Analysis

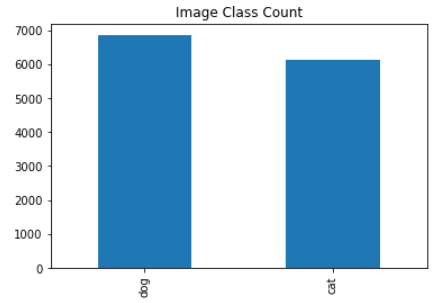
Statistics

```
print(f"There are a total of {len(glob.glob1(path, '*.jpg'))} images")
print(f"The total size is {os.path.getsize(path)/1000} MB")
df.shape
```

```
There are a total of 12966 images
The total size is 4.096 MB
(12966, 21)
```

Replace `LabelName` with human readable labels

```
df.LabelName.replace({'/m/01yrx':'cat', '/m/0bt9lr':'dog'}, inplace=True)
df.LabelName.value_counts()
df.LabelName.value_counts().plot(kind='bar')
plt.title('Image Class Count')
plt.show()
```




```
df.describe()
```

	Confidence	XMin	XMax	YMin	YMax	IsOccluded	IsTruncated	IsGroupOf	IsDepiction	IsInside	XClick1X	XCl
count	12966.0	12966.000000	12966.000000	12966.000000	12966.000000	12966.000000	12966.000000	12966.000000	12966.000000	12966.000000	12966.000000	12966.0
mean	1.0	0.099437	0.901750	0.088877	0.945022	0.464754	0.738470	0.013651	0.045427	0.001157	0.390356	0.4
std	0.0	0.113023	0.111468	0.097345	0.081500	0.499239	0.440011	0.118019	0.209354	0.040229	0.358313	0.4
min	1.0	0.000000	0.408125	0.000000	0.451389	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.0
25%	1.0	0.000000	0.830625	0.000000	0.910000	0.000000	0.000000	0.000000	0.000000	0.000000	0.221293	0.0
50%	1.0	0.061250	0.941682	0.059695	0.996875	0.000000	1.000000	0.000000	0.000000	0.000000	0.435625	0.4
75%	1.0	0.167500	0.998889	0.144853	0.999062	1.000000	1.000000	0.000000	0.000000	0.000000	0.609995	0.8
max	1.0	0.592500	1.000000	0.587088	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.999375	0.9

Sample of Images

By plotting random samples of the images along with the bounding boxes and XClick points, we see that every image has a bounding box but not every image has valid (positive) XClick information. From the descriptions on the CaDoD site, the bounding boxes were either derived from the extreme points clicked (aka XClick) by a human, or provided in some other way (prediction or manually drawn as a box).

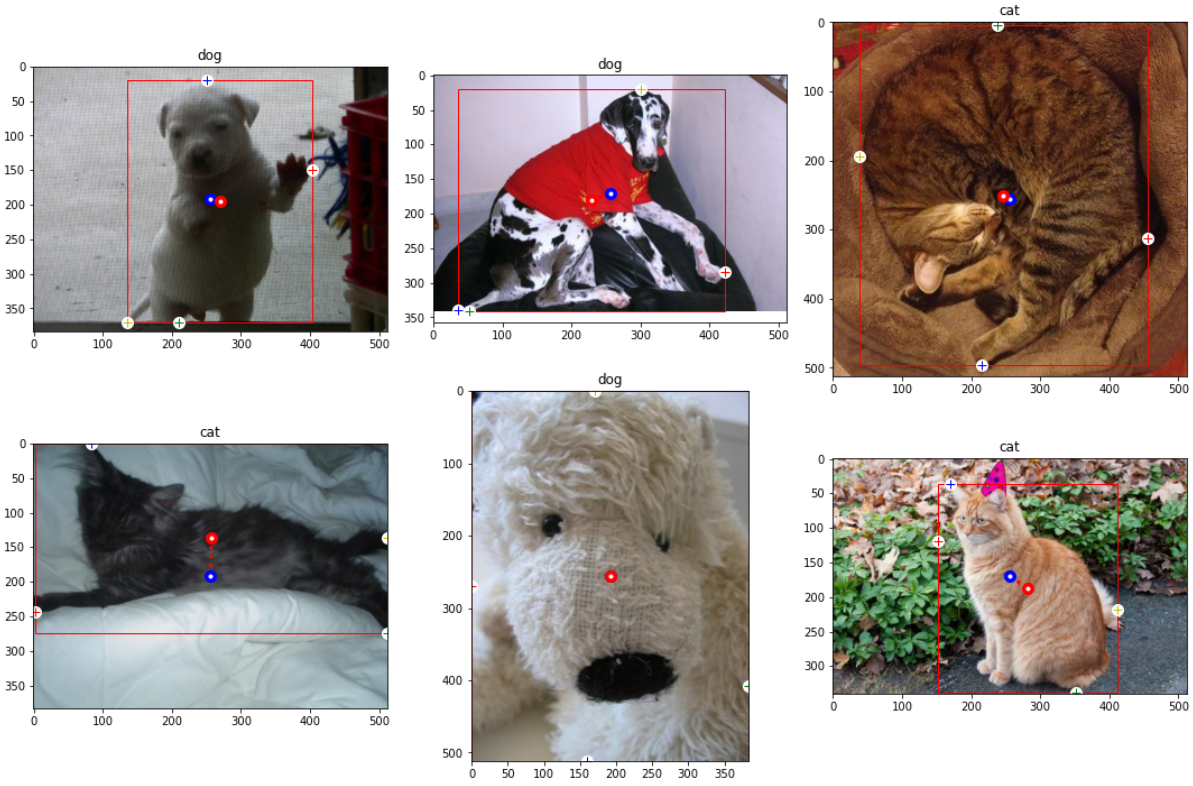
Further, it seems that the XClick items follow no system. For example, they are not in a predictable clockwise or counterclockwise ordering, they do not seem to start on a particular edge (like left side vs right side).

Since the bounding box information is more widely available and, where XClick is present, bounding box is derived from XClick, we can drop the XClick feature later on and just focus more on examining the bounding box attributes.

```
def RandomSixImages(items, orig=True, optSize=""):
    # orig = True means original size.  If False, specify 128 or 32 for resized images.
    fig, ax = plt.subplots(nrows=2, ncols=3, sharex=False, sharey=False,figsize=(15,10))
    ax = ax.flatten()

    impath = path

    if (orig == False):
        impath = path + "../../images/resized"+str(optSize) +"/"

    for i,j in enumerate(np.random.choice(items.shape[0], size=6, replace=False)):
        img = mpimg.imread(impath + items.ImageID.values[j] + '.jpg')
        h, w = img.shape[:2]
        coords = items.iloc[j,4:8]
        xclick_xcoords = items.iloc[j,13:17]*w
        xclick_ycoords = items.iloc[j,17:21]*h

        centerimg = [w/2, h/2]
        centerbox = [(coords[1]*w-coords[0]*w)/2 + coords[0]*w, (coords[3]*h-coords[2]*h)/2 + coords[2]*h]

        ax[i].imshow(img)
        ax[i].set_title(items.LabelName.values[j])

        # Plot the bounding box as a red border
        ax[i].add_patch(plt.Rectangle((coords[0]*w, coords[2]*h),
                                      coords[1]*w-coords[0]*w, coords[3]*h-coords[2]*h,
                                      edgecolor='red', facecolor='none'))

        ax[i].plot([centerimg[0], centerbox[0]],[centerimg[1], centerbox[1]],linestyle='dotted',linewidth=3,color='red')
        ax[i].plot(centerimg[0], centerimg[1],'bo', markersize=10)
        ax[i].plot(centerimg[0], centerimg[1],'w.')
        ax[i].plot(centerbox[0], centerbox[1],'ro', markersize=10)
        ax[i].plot(centerbox[0], centerbox[1],'w.')

        # If this image has valid positive XClick coords, plot those points
        if xclick_xcoords.any() < 0 or xclick_ycoords.any() < 0 :
            print("cannot plot, invalid (negative) xclick!")
        else :
            # plot each XClick point in scale on the image, 1 = blue, 2 = green, 3 = yellow, 4 = red
            ax[i].plot(xclick_xcoords[0], xclick_ycoords[0],'w.', markersize=20, label="1")
            ax[i].plot(xclick_xcoords[0], xclick_ycoords[0],'b+', markersize=8)
```

```
            ax[i].plot(xclick_xcoords[1], xclick_ycoords[1],'w.', markersize=20, label="2")
            ax[i].plot(xclick_xcoords[1], xclick_ycoords[1],'g+', markersize=8)
            ax[i].plot(xclick_xcoords[2], xclick_ycoords[2],'w.', markersize=20, label="3")
            ax[i].plot(xclick_xcoords[2], xclick_ycoords[2],'y+', markersize=8)
            ax[i].plot(xclick_xcoords[3], xclick_ycoords[3],'w.', markersize=20, label="4")
            ax[i].plot(xclick_xcoords[3], xclick_ycoords[3],'r+', markersize=8)
    plt.tight_layout()
    plt.show()
```

```
# plot random 6 images (original size)
RandomSixImages(df)
```



Let's look at the IsDepiction column. This seems to indicate if the image is a depiction (drawing, painting, not a 'real' animal photo).

```
depictioncount = Counter(df.IsDepiction)
depictioncount
```

```
Counter({0: 12371, 1: 592, -1: 3})
```

There are 3 items with -1 values. Let's look at those images to see what they're like.

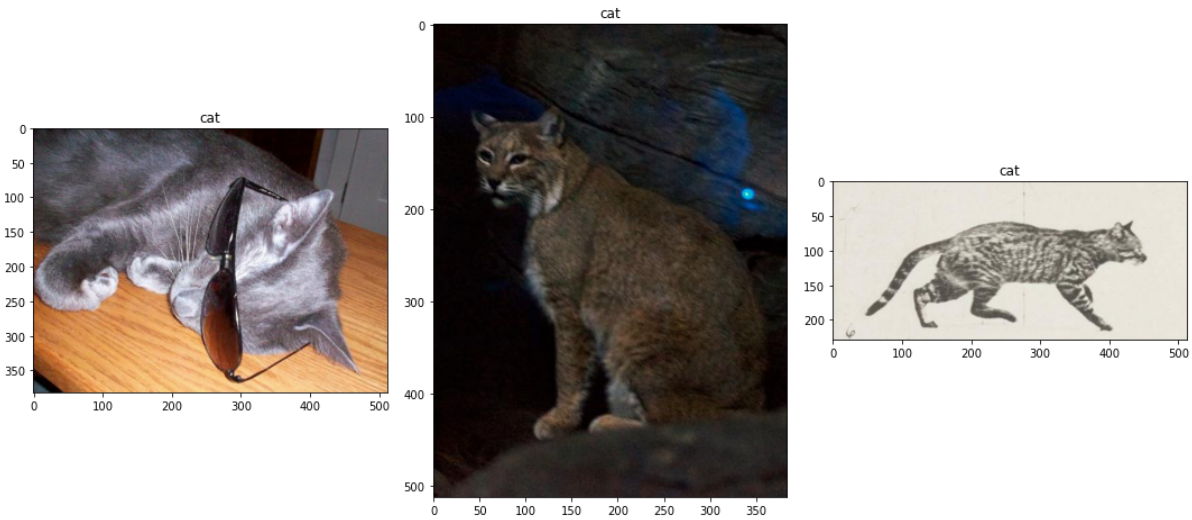
```
df[df.IsDepiction < 0]
```

	ImageID	Source	LabelName	Confidence	XMin	XMax	YMin	YMax	IsOccluded	IsTruncated	...	IsDepiction	IsInside	XClick1X
6878	004b6bf1ef1414b6	activemil	cat	1	0.002500	0.919375	0.002506	0.950710	-1	-1	...	-1	-1	-1.0
7084	021fd3fec9e6a438	activemil	cat	1	0.107500	0.957500	0.165625	0.884375	-1	-1	...	-1	-1	-1.0
7170	03c7118d468d94aa	activemil	cat	1	0.075548	0.874898	0.219601	0.900181	-1	-1	...	-1	-1	-1.0

3 rows × 21 columns

Let's look at the 3 images with IsDepiction = -1

```
# plot 3 "IsDepiction" = -1 images.
fig, ax = plt.subplots(nrows=1, ncols=3, sharex=False, sharey=False,figsize=(15,10))
ax = ax.flatten()
depictionimgs = df[df.IsDepiction < 0].to_numpy()
iterax = 0
for item in depictionimgs:
    img = mpimg.imread(path + item[0] + '.jpg')
    ax[iterax].imshow(img)
    ax[iterax].set_title(item[2])
    iterax += 1
plt.tight_layout()
plt.show()
```



Well, it looks from the dataframe above that these 3 images from Source = "activemil" don't have any XClick info or other info, though they do have bounding boxes. These are NOT depictions/drawings though, so we can look for -1 values to clean up in our Feature Engineering stage.

What else can we tell from this? Let's see if there is any other -1 data to deal with.

```
occludecount = Counter(df.IsOccluded)
truncatecount = Counter(df.IsTruncated)
groupcount = Counter(df.IsGroupOf)
insidecount = Counter(df.IsInside)
print (occludecount, truncatecount, groupcount, insidecount)
```

```
Counter({0: 6934, 1: 6029, -1: 3}) Counter({1: 9578, 0: 3385, -1: 3}) Counter({0: 12783, 1: 180, -1: 3}) Counter({0: 12945, 1: 18, -1: 3})
```

Gladly, this tells us that the same 3 images have -1 data, but others are ok (using either 0 or 1 for the IsOccluded, IsTruncated, IsGroupOf or IsInside columns.

Image shapes and sizes

Go through all images and record the shape of the image in pixels and the memory size

```
img_shape = []
img_size = np.zeros((df.shape[0], 1))
img_format = []
for i,f in enumerate(tqdm(glob.glob1(path, '*.jpg'))):
    file = path+'/'+f
    img = Image.open(file)
    img_shape.append(f"{img.size[0]}x{img.size[1]}")
    imgratio = img.size[0]/img.size[1]
    if imgratio > 1 :
        img_format.append('landscape')
    elif imgratio < 1:
        img_format.append('portrait')
    else:
        img_format.append('square')
    img_size[i] += os.path.getsize(file)
    img.close() # cleanup!
```

```
  0%|          | 0/12966 [00:00<?, ?it/s]
```

Count all the different image shapes

```
img_shape_count = Counter(img_shape)
img_format_count = Counter(img_format)
img_format_count
```

```
Counter({'landscape': 9657, 'square': 1216, 'portrait': 2093})
```

```
# create a dataframe for image shapes
img_df = pd.DataFrame(set(img_shape_count.items()), columns=['img_shape','img_count'])
img_df.shape
```

```
(594, 2)
```

```
# create a dataframe for simple image aspect ratios
img_ratio_df = pd.DataFrame(set(img_format_count.items()), columns=['aspect_ratio','img_count'])
img_ratio_df.head()
```

	aspect_ratio	img_count
0	landscape	9657
1	portrait	2093
2	square	1216

There are a ton of different image shapes. Let's narrow this down by getting a sum of any image shape that has a count less than 100 and put that in a category called `other`

```
img_df = img_df.append({'img_shape': 'other','img_count': img_df[img_df.img_count < 100].img_count.sum()},
                       ignore_index=True)
```

Drop all image shapes

```
img_df = img_df[img_df.img_count >= 100]
```

Check if the count sum matches the number of images

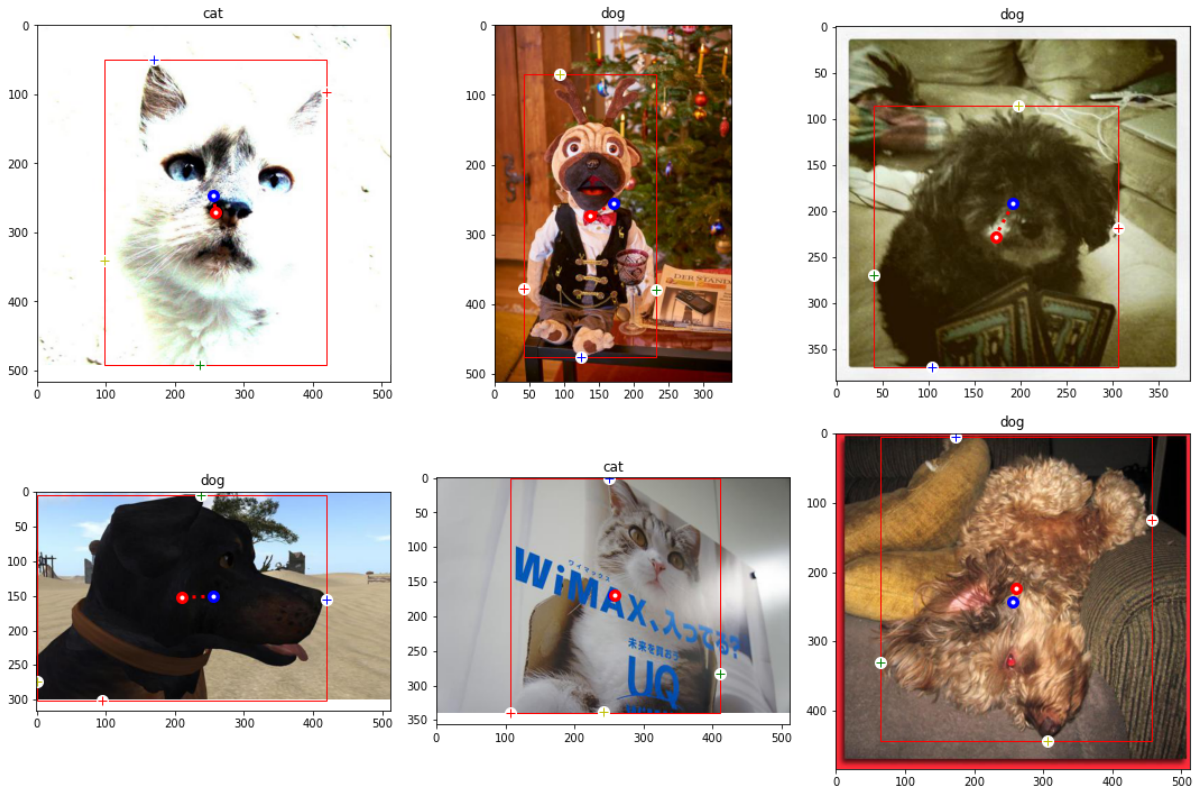
```
img_df.img_count.sum() == df.shape[0]
```

```
True
```

Depiction Images

Now, let's look at a random sample of images marked as "Depictions" (IsDepiction = 1). This would indicate a drawing or other nonstandard image representation of a dog or cat and we need to verify this assumption.

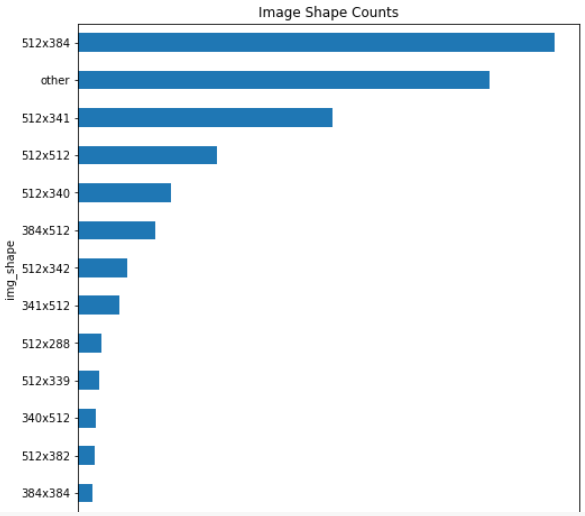
```
# plot random 6 depiction images (original size)
RandomSixImages(df[df.IsDepiction > 0])
```



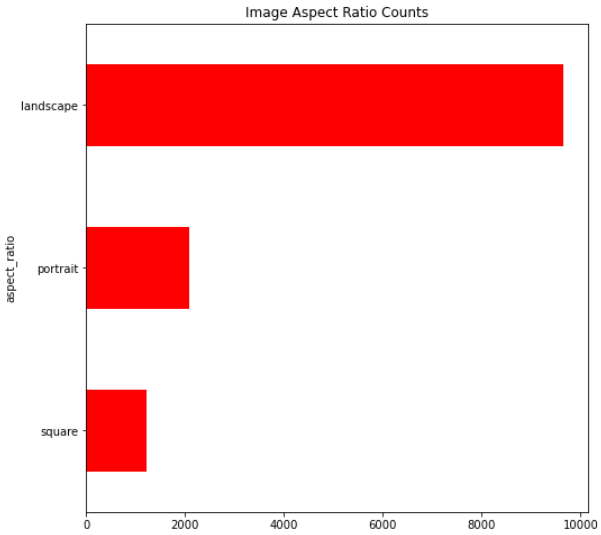
Well, it doesn't seem like IsDepiction tells us a whole lot. Sometimes, IsDepiction is set to 1 for paintings, statues, or just 'heavily filtered' or artistically distorted photos, but also photos that have low light or are black and white are tagged with this feature.

Plot aspect ratio

```
img_df.sort_values('img_count', inplace=True)
img_df.plot(x='img_shape', y='img_count', kind='barh', figsize=(8,8), legend=False)
plt.title('Image Shape Counts')
plt.show()
```




```
img_ratio_df.sort_values('img_count', inplace=True)
img_ratio_df.plot(x='aspect_ratio', y='img_count', kind='barh', color='red', figsize=(8,8), legend=False)
plt.title('Image Aspect Ratio Counts')
plt.show()
```

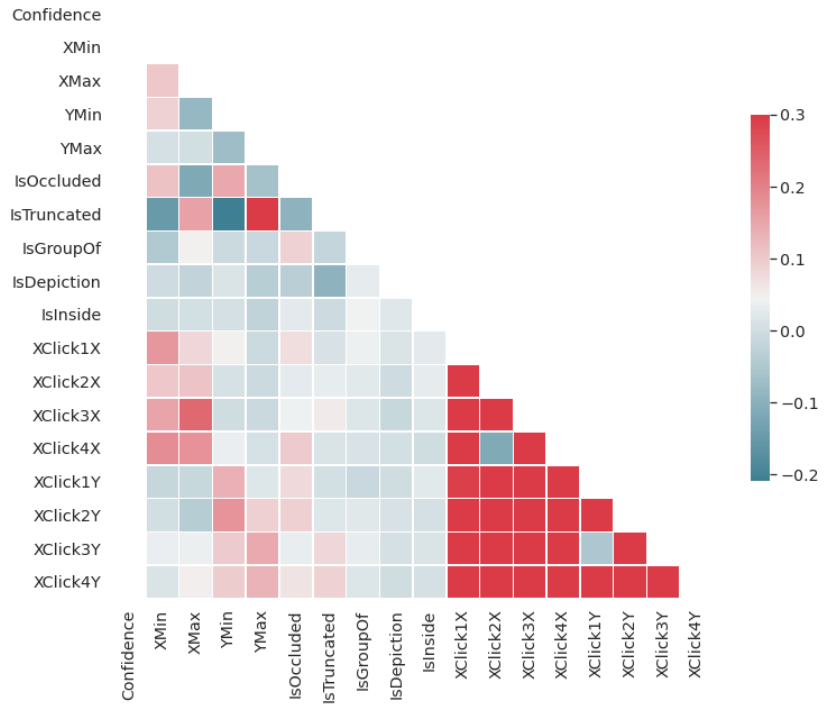



```
np.min(img_size), np.max(img_size)

# convert to megabytes
img_size = img_size / 1000
np.min(img_size), np.max(img_size)
```

```
(4.471, 502.8)
```

```
fig, ax = plt.subplots(1, 2, figsize=(15,5))
fig.suptitle('Image Size Distribution')
ax[0].hist(img_size, bins=50)
ax[0].set_title('Histogram')
ax[0].set_xlabel('Image Size (MB)')
ax[1].boxplot(img_size, vert=False, widths=0.5)
ax[1].set_title('Boxplot')
ax[1].set_xlabel('Image Size (MB)')
ax[1].set_ylabel('Images')
plt.show()
```


```
# data correlation LLM
import seaborn as sns

sns.set(style="whitegrid", font_scale=1.3)

class_labels = ["cat", "dog"]

# compute the correlation matrix
corr = df.corr()

# generate a mask for the lower triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# set up the matplotlib figure
f, ax = plt.subplots(figsize=(12, 12))

# generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3,
            square=True,
            linewidths=.5, cbar_kws={"shrink": .5}, ax=ax);
```



Preprocess

Rescale the images - 128x128

This rescaling will resize (and deform) all images to the same dimensions, 128 wide by 128 high. The bounding box information in the CSV file are based on percentages and not pixel locations, so bounding box info is already normalized. The rescaling of images to 128x128 in effect normalizes the image data so that we can better compare bounding boxes and images on the same dimensions without having to do further transformations.

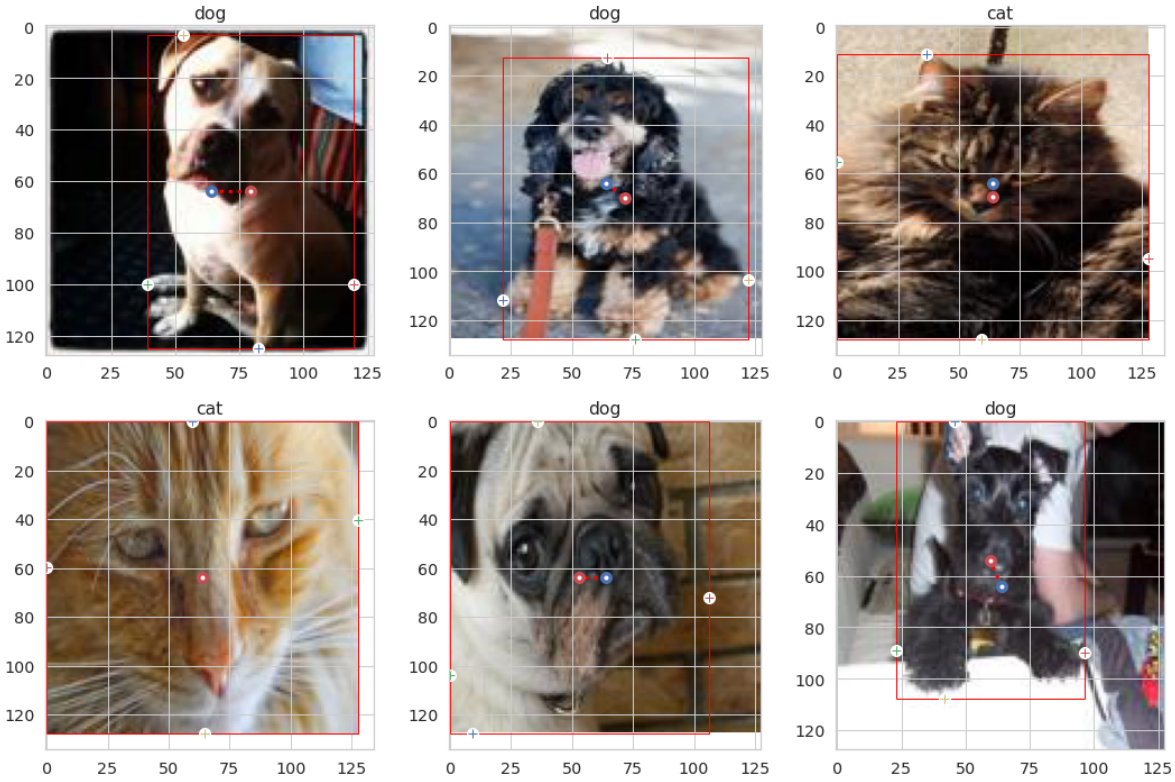
```
%%time

!mkdir -p images/resized128


# resize image and save, convert to numpy
def saveImageAndResize(w,h,ch):
    # w = 128, h = 128, ch = 3
    arr = np.zeros((df.shape[0],w*h*ch))

    for i, f in enumerate(tqdm(df.ImageID)):
        img = Image.open(path+f+'.jpg')
        img_resized = img.resize((w,h))
        img_resized.save("images/resized"+str(w)+"/"+f+'.jpg', "JPEG", optimize=True)
```

```
        img_resized.save('images/resized '+str(w)+' / '+i+'.jpg', 'JPEG', optimize=True)
        arr[i] = np.asarray(img_resized, dtype=np.uint8).flatten()
        img.close() # cleanup!
    return arr

img_arr = saveImageAndResize(128,128,3)
```

```
  0%|          | 0/12966 [00:00<?, ?it/s]
CPU times: user 1min 13s, sys: 9.53 s, total: 1min 23s
Wall time: 5min 9s
```

Plot the resized and filtered images

```
# plot random 6 of the 128x128 images
RandomSixImages(df, False, 128)
```



Encode Labels

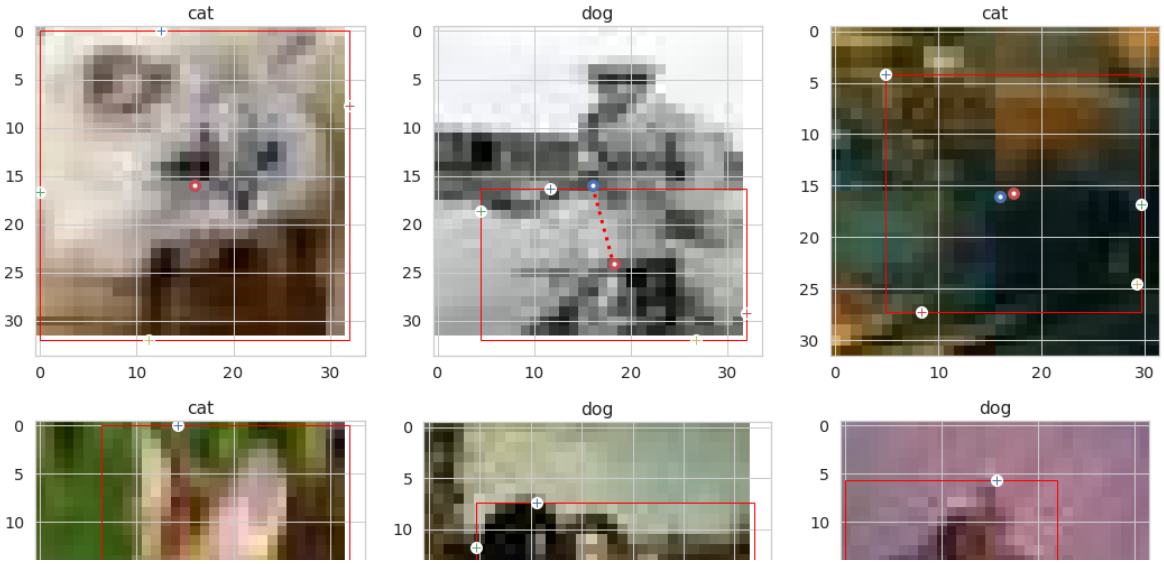
```
# encode labels
df['Label'] = (df.LabelName == 'dog').astype(np.uint8)
```

Resize images (32x32)

```
%%time

!mkdir -p images/resized32

img_arr32 = saveImageAndResize(32,32,3)
```

```
  0%|          | 0/12966 [00:00<?, ?it/s]
CPU times: user 53.9 s, sys: 5.29 s, total: 59.2 s
Wall time: 3min 50s
```

```
# plot random 6 of the 32x32 images
RandomSixImages(df, False, 32)
```



Split Data

Checkpoint and Save data

```
!mkdir -p data

# Skip/comment out the following steps if you've already exported the data as it takes a while.
np.save(path + '../img128.npy', img_arr.astype(np.uint8), allow_pickle=True)
np.save(path + '../img32.npy', img_arr32.astype(np.uint8), allow_pickle=True)
np.save(path + '../y_label.npy', df.Label.values, allow_pickle=True)
np.save(path + '../y_bbox.npy', df[['XMin', 'YMin', 'XMax', 'YMax']].values.astype(np.float32), allow_pickle=True)
```

Load data

```
#X128 = np.load(path + '../img128.npy', allow_pickle=True)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)

X32 = np.load(path + '../img32.npy', allow_pickle=True)
y_label = np.load(path + '../y_label.npy', allow_pickle=True)
y_bbox = np.load(path + '../y_bbox.npy', allow_pickle=True)
```

```
cuda:0
```

Train and Test Split

Numpy to Tensor & Normalize

```
X = torch.FloatTensor(X32)
y_label = torch.FloatTensor(y_label)
y_bbox = torch.FloatTensor(y_bbox)

X_train, X_test, y_train_label, y_test_label, y_train_box, y_test_box  = train_test_split(X, y_label,y_bbox, test_size=0.2, random_state=27)
```

```
# Normalize
X_train.shape
```

```
torch.Size([10372, 3072])
```

```
X_train = X_train / 255
X_test = X_test / 255
y_train_label = y_train_label / 255
y_test_label = y_test_label / 255
y_train_box = y_train_box / 255
y_test_box = y_test_box / 255

# Need to reshape from flat (what was stored in numpy) out to 3 channels by 32 by 32.
X_train = X_train.reshape(-1,3,32,32)
X_test = X_test.reshape(-1,3,32,32)

X_train.shape[0]
```

```
10372
```

PyTorch Implementation

```
# Configure device for gpu or cpu depending on what's available
#device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
#print(device)
```

```
!nvidia-smi
```

```
Tue May  4 01:38:22 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 465.19.01    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   72C    P0    32W /  70W |   1150MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

Create the Model class

```
class ConvNet(nn.Module):
    # Conv2d, pool, Conv2d, pool layering scheme adapted from Group 4's Phase 2 notebook share.

    def __init__(self):
        super(ConvNet, self).__init__()

        self.conv1 = nn.Conv2d(in_channels=3, out_channels=8, kernel_size=(3,3),stride=(1,1),padding=(1,1))

        # Reduce dimensions by half - POOL
        self.pool1 = nn.MaxPool2d(kernel_size=(2,2),stride=(2,2))

        self.conv2 = nn.Conv2d(in_channels=8, out_channels=16, kernel_size=(3,3),stride=(1,1),padding=(1,1))

        # Reduce dimensions by half again - POOL
        self.fc_label = nn.Linear(16 * 8 * 8, 120)
        self.out_label = nn.Linear( in_features=120, out_features=2)

    def forward(self, x):
        x = F.relu(self.conv1(x)) # first layer
        x = self.pool1(x) # pool
        x = F.relu(self.conv2(x)) # second layer
        x = self.pool1(x) # pool
        x = x.reshape(x.shape[0],-1)

        # Classify
        x1 = F.relu(self.fc_label(x))
        # Classify output
        x1 = F.log_softmax(self.out_label(x1), dim=1)
        return x1


    def num_flat_features(self, x):
        size = x.size()[1:]
        num_features = 1
        for s in size:
            num_features *= s
        return num_features


conv_net = ConvNet()
print(conv_net)
```

```
ConvNet(
  (conv1): Conv2d(3, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool1): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc_label): Linear(in_features=1024, out_features=120, bias=True)
  (out_label): Linear(in_features=120, out_features=2, bias=True)
)
```

Train the model

```
def convTrain(model,maxepoch = 10,eta = 0.01):
    loss_train = []
    loss_test = []
    acc_train = []
    acc_test = []
    stepnums = []
    steps = 0
    crit1 = nn.CrossEntropyLoss()
    params = model.parameters()
    optimiz = Adam(params, lr=eta, weight_decay=0)
    traintotal = X_train.shape[0]

    for epoch in range(maxepoch):
        y_pred = model(X_train)
        _, pred_labels = torch.max(y_pred.data, 1)
        correct = (pred_labels == y_train_label).sum().item()
        train_loss = crit1(y_pred, y_train_label.type(torch.LongTensor))
        loss_train.append(train_loss)

        y_pred_test = model(X_test)
        test_loss = crit1(y_pred_test, y_test_label.type(torch.LongTensor))
        loss_test.append(test_loss)
        print('epoch: ', epoch+1,' loss: ', train_loss.item())
        optimiz.zero_grad()
        train_loss.backward()
        optimiz.step()

        acc_test.append( 100 * correct / traintotal)
        stepnums.append(steps)
        steps+=1

    return {"epoch": stepnums, "train_loss": loss_train, "test_loss": loss_test, "train_accuracy": acc_train, "test_accuracy": acc_test }
```

```
results = []
```

```
results.append(convTrain(conv_net))
```

```
epoch:  1  loss:  0.7229611277580261
epoch:  2  loss:  0.005692302715033293
epoch:  3  loss:  7.27551878298982e-07
epoch:  4  loss:  3.529097292176431e-10
epoch:  5  loss:  0.0
epoch:  6  loss:  0.0
epoch:  7  loss:  0.0
epoch:  8  loss:  0.0
epoch:  9  loss:  0.0
epoch:  10  loss:  0.0
```

```
print(results)
```

Add CXE + MSE and adjust Model for bounding boxes

```
class ConvNetLabelBbox(nn.Module):
    # Conv2d, pool, Conv2d, pool layering scheme adapted from Group 4's Phase 2 notebook share.
    # For some reason, can't get as good of accuracy

    def __init__(self):
        super(ConvNetLabelBbox, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=8, kernel_size=(3,3),stride=(1,1),padding=(1,1))

        # Reduce dimensions by half - POOL
        self.pool1 = nn.MaxPool2d(kernel_size=(2,2),stride=(2,2))

        self.conv2 = nn.Conv2d(in_channels=8, out_channels=16, kernel_size=(3,3),stride=(1,1),padding=(1,1))

        # Reduce dimensions by half again - POOL
        self.fc_label = nn.Linear( 16 * 8 * 8, 120).to(device)
        self.out_label = nn.Linear( in_features=120, out_features=2).to(device)

        self.fc_bbox = nn.Linear( in_features=16 * 8 * 8,  out_features=120).to(device)
        self.out_bbox = nn.Linear( in_features=120, out_features=4).to(device) # One out feature for each BBOX coordinate

    def forward(self, x):
        x = x.to(device)
        x = F.relu(self.conv1(x)) # first layer
        x = self.pool1(x) # pool
        x = F.relu(self.conv2(x)) # second layer
        x = self.pool1(x) # pool
        x = x.reshape(x.shape[0],-1)

        # All data goes through the layers, then we split for the final predictions.

        xlabel = F.relu(self.fc_label(x))# cat/dog?
        xlabel = F.log_softmax(self.out_label(xlabel), dim=1) # prediction output for label

        xbbox = F.relu(self.fc_bbox(x)) # regression
```

```
        xbbox = F.relu(self.fc_bbox(x)) # regression
        xbbox = self.out_bbox(xbbox) # regression output Layer
        return xlabel, xbbox

    def num_flat_features(self, x):
        size = x.size()[1:]
        num_features = 1
        for s in size:
            num_features *= s
        return num_features


conv_netLB = ConvNetLabelBbox().to(device)

print(conv_netLB)
```

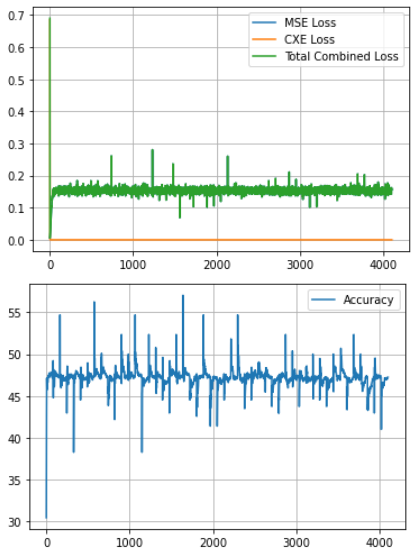
```
ConvNetLabelBbox(
  (conv1): Conv2d(3, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool1): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc_label): Linear(in_features=1024, out_features=120, bias=True)
  (out_label): Linear(in_features=120, out_features=2, bias=True)
  (fc_bbox): Linear(in_features=1024, out_features=120, bias=True)
  (out_bbox): Linear(in_features=120, out_features=4, bias=True)
)
```

```
def convLBTrain(model,maxepoch = 7,eta = 0.001, batchsize=128):
    model = model.to(device)
    train_data = torch.utils.data.TensorDataset(X_train, y_train_label, y_train_box)
    test_data = torch.utils.data.TensorDataset(X_test, y_test_label, y_test_box)
    train_dataloader = torch.utils.data.DataLoader(train_data, batch_size=batchsize, shuffle=True)
    test_dataloader = torch.utils.data.DataLoader(test_data, batch_size=batchsize, shuffle=False)

    loss_train = []
    loss_MSE_train = []
    loss_CXE_train = []
    loss_total = []

    loss_test = []
    acc_train = []
    acc_test = []
    acc_total = []
    stepnums = []
    steps = 0
    crit1 = nn.CrossEntropyLoss(reduction='mean') # add mean reduction
    crit2 = nn.MSELoss() # add mse loss

    params = model.parameters()
    optimiz = Adam(params, lr=eta, weight_decay=0)

    acc_train = [0] * maxepoch
    acc_test = [0] * maxepoch
    loss_train = [0] * maxepoch
    loss_test = [0] * maxepoch

    for epoch in tqdm(range(maxepoch)):

        correct = 0
        total = 0
        running_loss =0
        correct_val=0
        total_val = 0
        running_loss_val = 0


        for batch, data in enumerate(train_dataloader):
            inputs, predlabels, predbox = data[0].to(device), data[1].to(device), data[2].to(device)
            optimiz.zero_grad()

            outputs1, outputs2 = model(inputs) #forward pass
            _,predicted = torch.max(outputs1.data, 1)
            predicted.to(device)
            total += predlabels.size(0)
            correct += (predicted == predlabels).sum().item()
            acc_total.append( 100 * correct / total )


            loss1 = crit1(outputs1, predlabels.type(torch.LongTensor).to(device))
            loss2 = crit2(outputs2.data, predbox.float().to(device))
            loss_CXE_train.append(loss1)
            loss_MSE_train.append(loss2)


            loss_train = loss1 + loss2 #combined loss
            #print('CXE Loss',loss1.item(),'MSE Loss',loss2.item(),'Total Loss:',loss_train.item())
            loss_total.append(loss_train)
            loss_train.backward() #backward pass
            optimiz.step()
            running_loss += loss_train.item()
            steps +=1
            stepnums.append(steps)

        print('End of Epoch ' + str(epoch + 1) + ", Running loss " + str(running_loss))
```

```
    plt.plot(stepnums, loss_MSE_train, label='MSE Loss')
    plt.plot(stepnums, loss_CXE_train, label='CXE Loss')
    plt.plot(stepnums, loss_total, label='Total Combined Loss')
    plt.legend()
    plt.grid()
    plt.show()
    plt.plot(stepnums, acc_total, label='Accuracy')
    plt.legend()
    plt.grid()
    plt.show()
    return stepnums, loss_MSE_train, loss_CXE_train, loss_total, acc_total
```

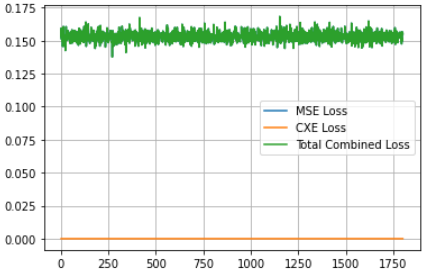
```
results_01_fresh = convLBTrain(conv_netLB,50,0.01,128)
```

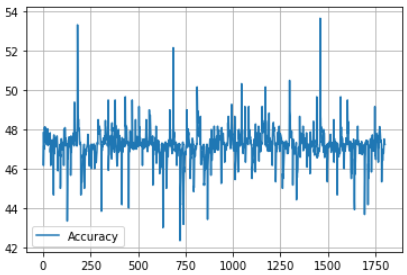
```
100%                                                50/50 [00:19<00:00, 2.61it/s]
End of Epoch 1, Running loss 10.490718557033688
End of Epoch 2, Running loss 12.56389006972313
End of Epoch 3, Running loss 12.582363933324814
End of Epoch 4, Running loss 12.590561509132385
End of Epoch 5, Running loss 12.555502504110336
End of Epoch 6, Running loss 12.590005844831467
End of Epoch 7, Running loss 12.589588448405266
End of Epoch 8, Running loss 12.552010849118233
End of Epoch 9, Running loss 12.665040865540504
End of Epoch 10, Running loss 12.542620778083801
End of Epoch 11, Running loss 12.579917699098587
End of Epoch 12, Running loss 12.538521558046341
End of Epoch 13, Running loss 12.534482523798943
End of Epoch 14, Running loss 12.58026510477066
End of Epoch 15, Running loss 12.683552980422974
End of Epoch 16, Running loss 12.530542597174644
End of Epoch 17, Running loss 12.531489133834839
End of Epoch 18, Running loss 12.640820235013962
End of Epoch 19, Running loss 12.478399254381657
End of Epoch 20, Running loss 12.548613488674164
End of Epoch 21, Running loss 12.511766470968723
End of Epoch 22, Running loss 12.572217836976051
End of Epoch 23, Running loss 12.509953558444977
End of Epoch 24, Running loss 12.513827733695507
End of Epoch 25, Running loss 12.527319997549057
End of Epoch 26, Running loss 12.663912773132324
End of Epoch 27, Running loss 12.581460922956467
End of Epoch 28, Running loss 12.535957977175713
End of Epoch 29, Running loss 12.56496275961399
End of Epoch 30, Running loss 12.564749166369438
End of Epoch 31, Running loss 12.540914043784142
End of Epoch 32, Running loss 12.590051352977753
End of Epoch 33, Running loss 12.597077488899231
End of Epoch 34, Running loss 12.552403450012207
End of Epoch 35, Running loss 12.616141840815544
End of Epoch 36, Running loss 12.594156309962273
End of Epoch 37, Running loss 12.533605873584747
End of Epoch 38, Running loss 12.509444877505302
End of Epoch 39, Running loss 12.51125480234623
End of Epoch 40, Running loss 12.551057279109955
End of Epoch 41, Running loss 12.54937270283699
End of Epoch 42, Running loss 12.530248738825321
End of Epoch 43, Running loss 12.568877428770065
End of Epoch 44, Running loss 12.583553209900856
End of Epoch 45, Running loss 12.610554590821266
End of Epoch 46, Running loss 12.608046531677246
End of Epoch 47, Running loss 12.539220824837685
End of Epoch 48, Running loss 12.585761085152626
End of Epoch 49, Running loss 12.534449830651283
End of Epoch 50, Running loss 12.562974452972412
```




```
results_01_b = convLBTrain(conv_netLB,100,0.001,600)
```

```
100%                                  100/100 [00:47<00:00, 2.11it/s]
End of Epoch 1, Running loss 2.7517277896404266
End of Epoch 2, Running loss 2.7603221982717514
End of Epoch 3, Running loss 2.759460598230362
End of Epoch 4, Running loss 2.754369407892227
End of Epoch 5, Running loss 2.7597551941871643
End of Epoch 6, Running loss 2.758982166647911
End of Epoch 7, Running loss 2.7545095682144165
End of Epoch 8, Running loss 2.76386658847332
End of Epoch 9, Running loss 2.7560833990573883
End of Epoch 10, Running loss 2.7506542652845383
End of Epoch 11, Running loss 2.7624558955430984
End of Epoch 12, Running loss 2.759729638695717
End of Epoch 13, Running loss 2.7513801902532578
End of Epoch 14, Running loss 2.7585514187812805
End of Epoch 15, Running loss 2.746001675724983
End of Epoch 16, Running loss 2.7578049302101135
End of Epoch 17, Running loss 2.7621534168720245
End of Epoch 18, Running loss 2.7566923052072525
End of Epoch 19, Running loss 2.748416528105736
End of Epoch 20, Running loss 2.763768747448921
End of Epoch 21, Running loss 2.7577979117631912
End of Epoch 22, Running loss 2.755906507372856
End of Epoch 23, Running loss 2.767270341515541
End of Epoch 24, Running loss 2.7590008676052094
End of Epoch 25, Running loss 2.7533900141716003
End of Epoch 26, Running loss 2.753100648522377
End of Epoch 27, Running loss 2.7548489272595445
End of Epoch 28, Running loss 2.7585076093673706
End of Epoch 29, Running loss 2.757560670375824
End of Epoch 30, Running loss 2.7577087581157684
End of Epoch 31, Running loss 2.7606718987226486
End of Epoch 32, Running loss 2.7587674856185913
End of Epoch 33, Running loss 2.759646251797676
End of Epoch 34, Running loss 2.7620539367198944
End of Epoch 35, Running loss 2.752718538045883
End of Epoch 36, Running loss 2.7586718052625656
End of Epoch 37, Running loss 2.76017926633358
End of Epoch 38, Running loss 2.7574908435344696
End of Epoch 39, Running loss 2.76192869246006
End of Epoch 40, Running loss 2.7602422684431076
End of Epoch 41, Running loss 2.7574294954538345
End of Epoch 42, Running loss 2.764261931180954
End of Epoch 43, Running loss 2.758371502161026
End of Epoch 44, Running loss 2.75602388381958
End of Epoch 45, Running loss 2.757798582315445
End of Epoch 46, Running loss 2.7547053396701813
End of Epoch 47, Running loss 2.7556531727313995
End of Epoch 48, Running loss 2.764784425497055
End of Epoch 49, Running loss 2.755275458097458
End of Epoch 50, Running loss 2.757424294948578
End of Epoch 51, Running loss 2.762610226869583
End of Epoch 52, Running loss 2.7576088160276413
End of Epoch 53, Running loss 2.760297805070877
End of Epoch 54, Running loss 2.757363274693489
End of Epoch 55, Running loss 2.7576982975006104
End of Epoch 56, Running loss 2.7548659443855286
End of Epoch 57, Running loss 2.757068619132042
End of Epoch 58, Running loss 2.7601994574069977
End of Epoch 59, Running loss 2.7569881081581116
End of Epoch 60, Running loss 2.758719578385353
End of Epoch 61, Running loss 2.7568608224391937
End of Epoch 62, Running loss 2.7525739520788193
End of Epoch 63, Running loss 2.760172039270401
End of Epoch 64, Running loss 2.767883062362671
End of Epoch 65, Running loss 2.7608528584241867
End of Epoch 66, Running loss 2.76300011575222
End of Epoch 67, Running loss 2.7636103332042694
End of Epoch 68, Running loss 2.7507272511720657
End of Epoch 69, Running loss 2.7624413818120956
End of Epoch 70, Running loss 2.7646234184503555
End of Epoch 71, Running loss 2.759262427687645
End of Epoch 72, Running loss 2.753308266401291
End of Epoch 73, Running loss 2.7565923780202866
End of Epoch 74, Running loss 2.7546677589416504
End of Epoch 75, Running loss 2.7636872977018356
End of Epoch 76, Running loss 2.758094161748886
End of Epoch 77, Running loss 2.754141479730606
End of Epoch 78, Running loss 2.7543038725852966
End of Epoch 79, Running loss 2.7609037309885025
End of Epoch 80, Running loss 2.755443960428238
End of Epoch 81, Running loss 2.7614660263061523
End of Epoch 82, Running loss 2.761630117893219
End of Epoch 83, Running loss 2.7608061879873276
End of Epoch 84, Running loss 2.753798559308052
End of Epoch 85, Running loss 2.764297768473625
End of Epoch 86, Running loss 2.7515719383955
End of Epoch 87, Running loss 2.7511641085147858
End of Epoch 88, Running loss 2.750826597213745
End of Epoch 89, Running loss 2.7569742500782013
End of Epoch 90, Running loss 2.7654207944869995
End of Epoch 91, Running loss 2.751123398542404
End of Epoch 92, Running loss 2.7595983743667603
End of Epoch 93, Running loss 2.7550555765628815
End of Epoch 94, Running loss 2.751991346478462
End of Epoch 95, Running loss 2.7565902918577194
End of Epoch 96, Running loss 2.7541757971048355
End of Epoch 97, Running loss 2.7578192055225372
End of Epoch 98, Running loss 2.7602310329675674
End of Epoch 99, Running loss 2.759196639060974
End of Epoch 100, Running loss 2.7584883868694305
```






```
results_01_b = convLBTrain(conv_netLB,50,0.01,128)
```

```
100%                                    50/50 [07:41<00:00, 9.23s/it]
End of Epoch 1, Running loss 12.610188990831375
End of Epoch 2, Running loss 12.566380321979523
End of Epoch 3, Running loss 12.539105832576752
End of Epoch 4, Running loss 12.526105999946594
End of Epoch 5, Running loss 12.51476502418518
End of Epoch 6, Running loss 12.55795268714428
End of Epoch 7, Running loss 12.552002727985382
End of Epoch 8, Running loss 12.52805045992136
End of Epoch 9, Running loss 12.58572182059288
End of Epoch 10, Running loss 12.52332715690136
```

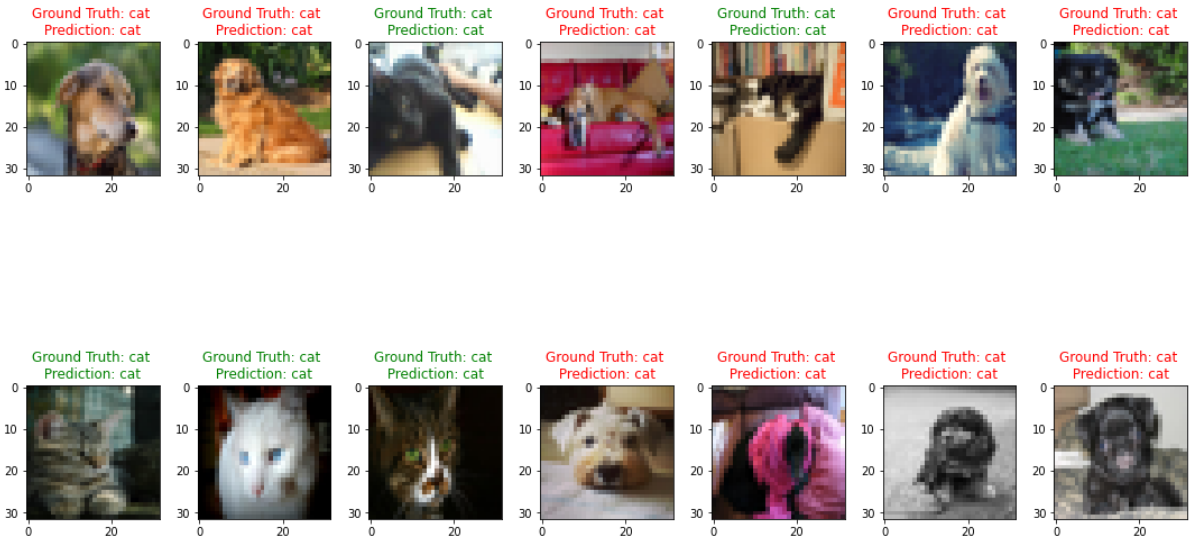
```
results_01_c = convLBTrain(conv_netLB,80,0.01,128)
```

```
100%                                  80/80 [00:33<00:00, 2.37it/s]
End of Epoch 1, Running loss 0.1487987118307501
End of Epoch 2, Running loss 0.14878770906943828
End of Epoch 3, Running loss 0.14870843361131847
End of Epoch 4, Running loss 0.14869893970899284
End of Epoch 5, Running loss 0.14864880952518433
End of Epoch 6, Running loss 0.14862454251851887
End of Epoch 7, Running loss 0.14874280209187418
End of Epoch 8, Running loss 0.14875924517400563
End of Epoch 9, Running loss 0.1486717585939914
End of Epoch 10, Running loss 0.14857113955076784
End of Epoch 11, Running loss 0.14858647377695888
End of Epoch 12, Running loss 0.1486640617949888
End of Epoch 13, Running loss 0.14860021183267236
End of Epoch 14, Running loss 0.14863313001114875
End of Epoch 15, Running loss 0.14861080050468445
End of Epoch 16, Running loss 0.14856920205056667
End of Epoch 17, Running loss 0.14845248719211668
End of Epoch 18, Running loss 0.14852923981379718
End of Epoch 19, Running loss 0.14848020754288882
End of Epoch 20, Running loss 0.1485614035045728
End of Epoch 21, Running loss 0.14855455607175827
End of Epoch 22, Running loss 0.14843470905907452
End of Epoch 23, Running loss 0.1484722044551745
```

```
# predict
y_pred_label,y_pred_bbox =  conv_netLB(X_test)
_,predicted = torch.max (y_pred_label.data, 1)
predicted.to(device)

y_pred_label =predicted.cpu().detach().numpy()
y_pred_bbox = y_pred_bbox.data.cpu().detach().numpy()
```

```
End of Epoch 36, Running loss 0.14847890031524003
```

```
#test
idx_to_label = {1:'dog', 0:'cat'} # encoder

fig, ax = plt.subplots(nrows=2, ncols=7, sharex=False, sharey=False,figsize=(15,10))
ax = ax.flatten()


for i,j in enumerate(np.random.choice(X_test.shape[0], size=14, replace=False)):

    img = X_test[j].reshape(32,32,3)
    coords = y_pred_bbox[j] *32
    #print(coords) # why are so many negative?
    ax[i].imshow(img)
    ax[i].set_title("Ground Truth: {0} \n Prediction: {1} ".format(idx_to_label[int(y_test_label[j])],idx_to_label[int(y_pred_label[j])]),color=("green" if

    ax[i].add_patch(plt.Rectangle((coords[0], coords[1]),
                                  coords[2]-coords[0], coords[3]-coords[1],
                                  edgecolor='red', facecolor='none'))

plt.tight_layout()
plt.show()
```



Ben's - only look at training on class cat/dog - Create train_csv.csv

```
def label_img(row):
    if row['LabelName'] == '/m/0bt9lr':
        return 1
```

```
    if row['LabelName'] == '/m/01yrx':
        return 0
```

```
df2 = df
df2.head()
```

```
df2['cdlabel'] = df2.apply(lambda row: label_img(row), axis=1)
df2['cdlabel'].unique()
```

```
df3 = pd.concat([df2['ImageID'], df2['cdlabel']], axis=1, keys=['ImageID', 'label'])
```

```
df3.head()
```

```
df3.to_csv(r'train_csv.csv', index=False, header=True)
```

Create (New) Custom Dataset

```
class CustomDataload(Dataset):
    def __init__(self, root_dir, annotation_file, transform=None):
        self.root_dir = root_dir
        self.annotations = pd.read_csv(annotation_file)
        self.transform = transform

    def __len__(self):
        return len(self.annotations)

    def __getitem__(self, index):
        img_id = self.annotations.iloc[index, 0]
        img = Image.open(os.path.join(self.root_dir, img_id+'.jpg')).convert("RGB")
        y_label = torch.tensor(self.annotations.iloc[index, 1])

        if self.transform is not None:
            img = self.transform(img)

        return (img, y_label)
```

```
#Set training set transform process on PIL Image item from custom dataset
train_transform = transforms.Compose([
    # transforms.Grayscale(num_output_channels=1),
    transforms.Resize((32,32)),
    transforms.ToTensor(),
    # transforms.Normalize((0.5,0.5,0.5), (0.5,0.5,0.5))
])

# Set test set transformations:
test_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,0.5,0.5), (0.5,0.5,0.5))
])


train_file = 'train_csv.csv'
img_dir = '/content/drive/MyDrive/Colab Notebooks/CatsNDogs/data/cadod/'

all_data = CustomDataload(root_dir=img_dir, annotation_file=train_file, transform=train_transform)
```

```
batch_size = 64
# shuffle = True
# test_split = .2

# dataset_size = len(all_data)
# indices = list(range(dataset_size))

# split = int(np.floor(test_split * dataset_size))
# np.random.seed(27)
# np.random.shuffle(indices)

# train_indices, test_indices = indices[split:], indices[:split]

# #call the subset sampler to sample our data
# train_sampler = SubsetRandomSampler(train_indices) #random sample for the indices for training data and test data
# test_sampler = SubsetRandomSampler(test_indices)

num_train = int(len(all_data) * 0.8)
train_set, test_set = torch.utils.data.random_split(all_data, [num_train, len(all_data) - num_train])

# Make Train Loader:
train_dataloader = DataLoader(train_set, shuffle=True, batch_size=batch_size)

# Make test loader:
test_dataloader = DataLoader(test_set, shuffle=False, batch_size=batch_size)
```

Define the Model

```
class Unit(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(Unit, self).__init__()

        self.conv = nn.Conv2d(in_channels=in_channels, kernel_size=3,
                              out_channels=out_channels, stride=1, padding=1)
        self.bn = nn.BatchNorm2d(num_features=out_channels)
        self.relu = nn.ReLU()

    def forward(self, input):
        output = self.conv(input)
        output = self.bn(output)
        output = self.relu(output)

        return output

class ImgNet(nn.Module):
    def __init__(self, num_classes):
        super(ImgNet, self).__init__()

         #Create 14 layers of the unit with max pooling in between
        self.unit1 = Unit(in_channels=3,out_channels=32)
        self.unit2 = Unit(in_channels=32, out_channels=32)
        self.unit3 = Unit(in_channels=32, out_channels=32)

        self.pool1 = nn.MaxPool2d(kernel_size=2)

        self.unit4 = Unit(in_channels=32, out_channels=64)
        self.unit5 = Unit(in_channels=64, out_channels=64)
        self.unit6 = Unit(in_channels=64, out_channels=64)
        self.unit7 = Unit(in_channels=64, out_channels=64)

        self.pool2 = nn.MaxPool2d(kernel_size=2)

        self.unit8 = Unit(in_channels=64, out_channels=128)
        self.unit9 = Unit(in_channels=128, out_channels=128)
        self.unit10 = Unit(in_channels=128, out_channels=128)
        self.unit11 = Unit(in_channels=128, out_channels=128)

        self.pool3 = nn.MaxPool2d(kernel_size=2)

        self.unit12 = Unit(in_channels=128, out_channels=128)
        self.unit13 = Unit(in_channels=128, out_channels=128)
        self.unit14 = Unit(in_channels=128, out_channels=128)

        self.avgpool = nn.AvgPool2d(kernel_size=4)

        #Add all the units into the Sequential layer in exact order
        self.net = nn.Sequential(self.unit1, self.unit2, self.unit3, self.pool1,
                                 self.unit4, self.unit5, self.unit6
                                 ,self.unit7, self.pool2, self.unit8,
                                 self.unit9, self.unit10, self.unit11, self.pool3,
                                 self.unit12, self.unit13, self.unit14, self.avgpool)

        self.fc = nn.Linear(in_features=128, out_features=num_classes)

    def forward(self, input):
        output = self.net(input)
        output = output.view(-1,128)
        output = self.fc(output)
        return output
```

Train Model

```
from torch.optim import Adam

cuda_avail = torch.cuda.is_available()

model = ImgNet(num_classes=2)

if cuda_avail:
    model.cuda()

optimizer = Adam(model.parameters(), lr=0.001, weight_decay=0.0001)
loss_fn = nn.CrossEntropyLoss()
```

```
for name, param in model.named_parameters():
  print(name, '\t', param.shape)
```

```
# Check for available GPU for modelNN:
if torch.cuda.is_available():
    model.cuda()
```