AML I526 Spring 2021 Group 2 Phase 2 Update
Cats vs Dogs Classification and Prediction

Team Info:

Left to right:	Ben Perkins	benperki@iu.edu
	Lauren Madar	laurenmadar@gmail.com
	Mangesh Walimbe	mwalimbe@iu.edu
	Samin Barghan	s.barghan@gmail.com

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.

For the second phase, 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, though the Nvidia GPU made a move to Google Colab necessary.

The team will give brief updates on the following sections.

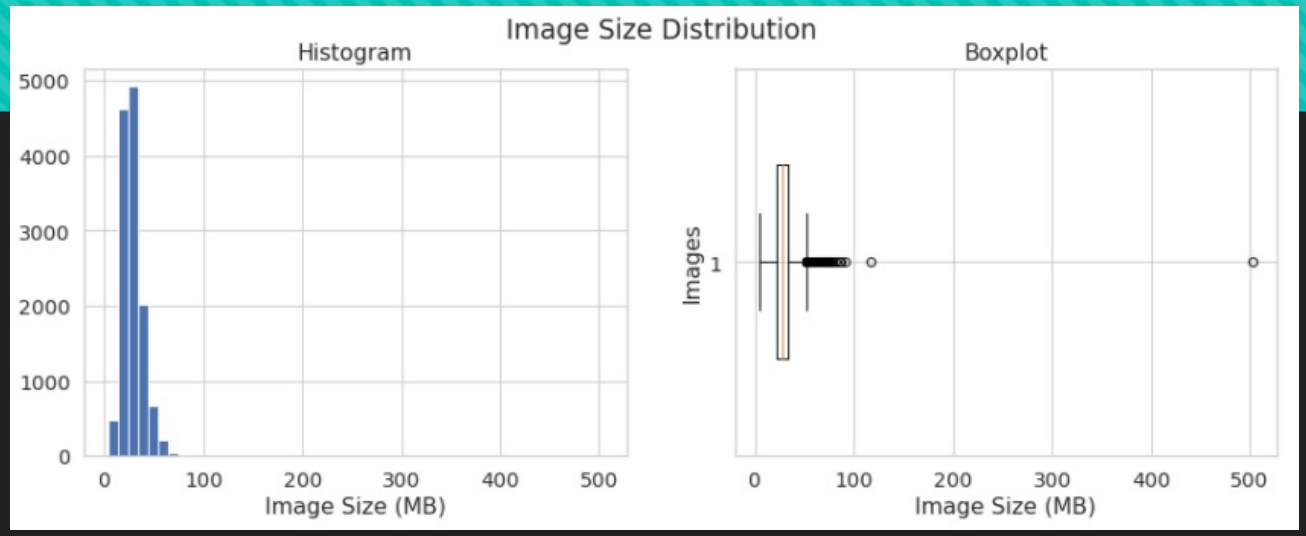
Overview

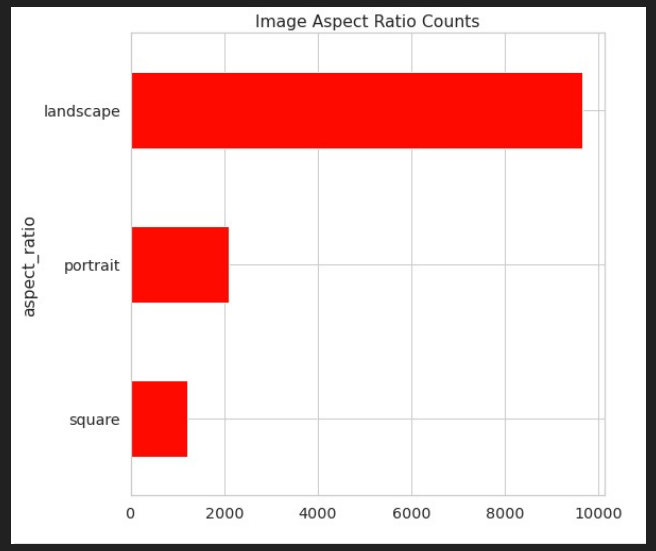
During this update, we will only be talking through the following:

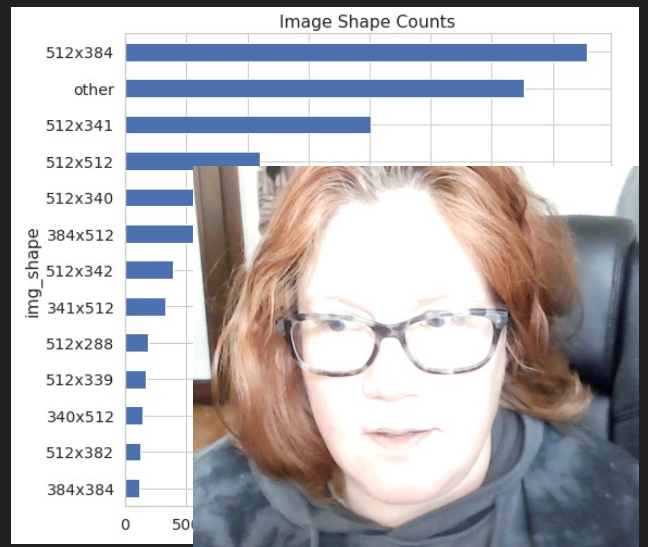
- Changes to workflow for PyTorch - Lauren
- PyTorch Models - Mangesh
- Pytorch Metrics - Ben
- Updates for Results, Discussion & Conclusion - Ben

Data – Description and Statistics – no Phase 2 updates

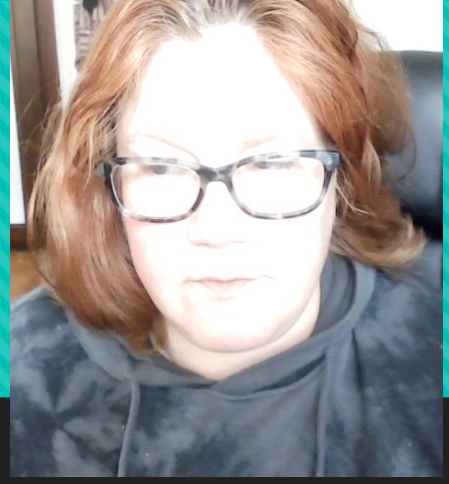
- 12966 images varying in file size from ~ 4mb to ~502 mb
- Variety of width and height in pixels (mix of portrait, landscape and square images and sizes) which **creates normalization problems** (comparing images of different sizes without adjusting for those differences will result in lower performance or inaccurate predictions)
- **Bounding box metadata** to highlight the main object in each image for training and a class label for each image (detailed after the data dictionary)
- **2 total classes** – "cat" and "dog"



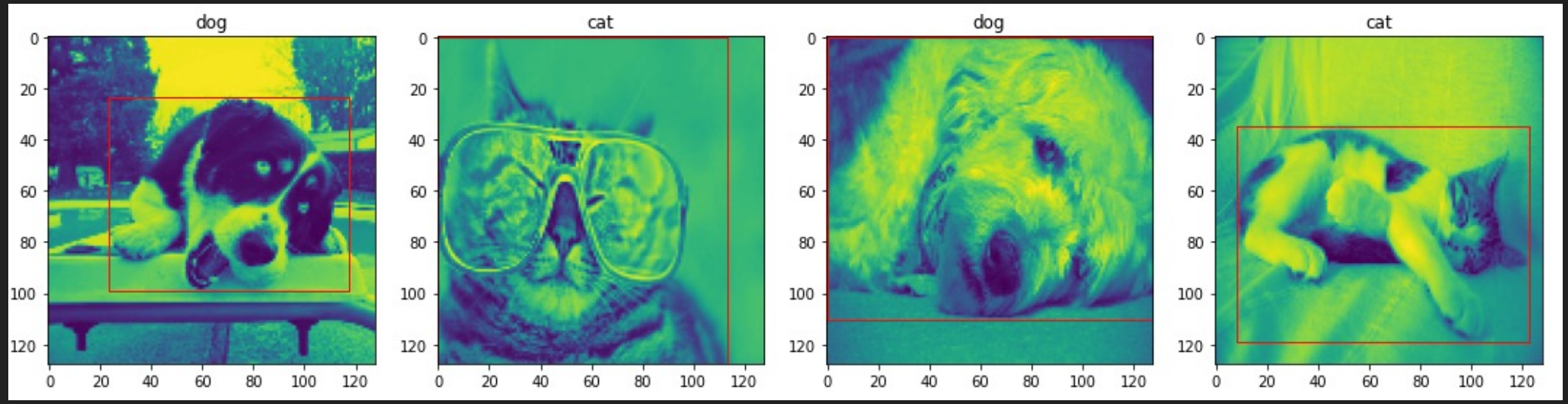




Data Tasks – Phase 2 - PyTorch

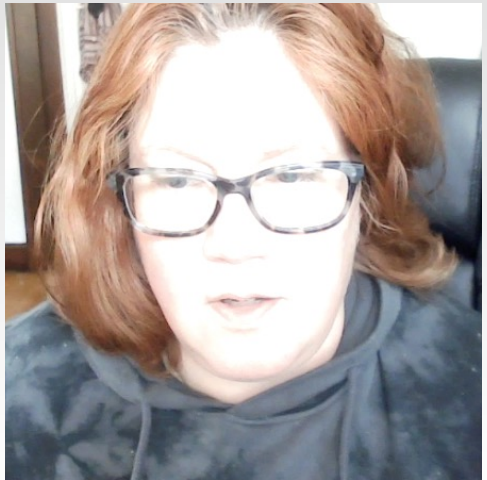


- Import CSV and parse metadata, Train/Test split data indices using SubsetRandomSampler and DataLoader
- Transform with CustomDataset:
  - Convert to Tensor
  - Transform to grayscale to reduce channels
  - Resize images to 128x128 as normalization



Train images after greyscale/resize with overlay ground truth bounding boxes and labels

PyTorch Workflow – Phase 2



Models - PyTorch

Our PyTorch models are neural networks, and produce predictions for classification and bounding boxes in the same 'pipeline'

Conv2d Model

Intended to use 2d convolutions (2) and pooling before linear classification and regression.

Issues: could not format parameters for Conv2d from our image, bbox and label tensors successfully.

Did not run due to bugs.

SimpleLinearNet Model

3 linear layers and 2 dropout layers (for regularization).

Issues: trains, but with max 53% accuracy.

Run several times with train sizes on GPU but lost output due to usage limits, and ran on CPU: 4/2, 10/2, 30/6 train/test splits

FrankenNet Model

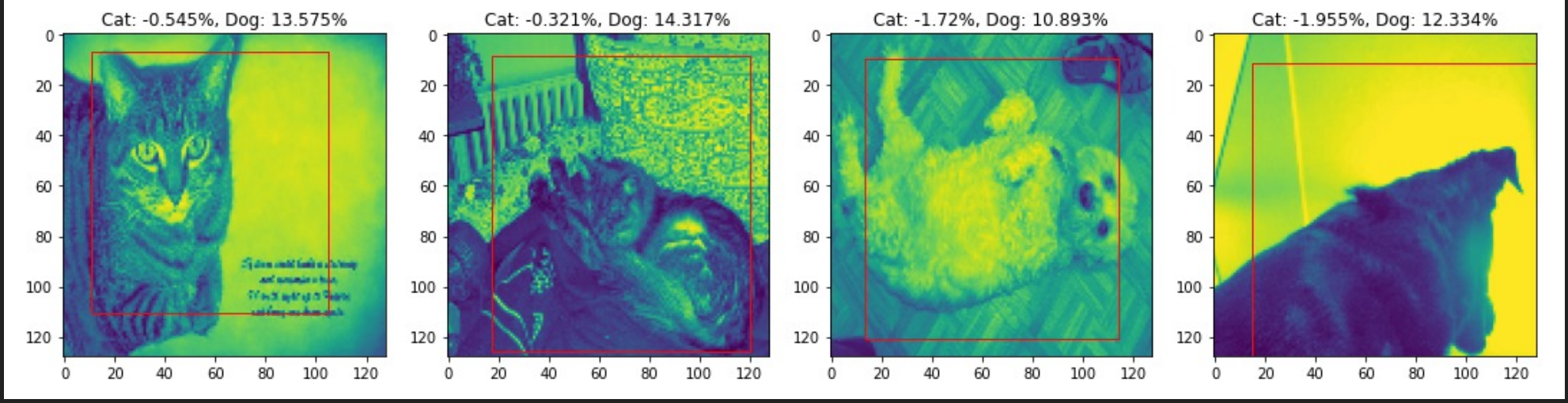
Promising many-layered Conv2d, BatchNorm2D, ReLU and Linear model.

Issues: got this running late on the Phase 2 due date, needs further refinement and analysis, is overfitting, needs bounding box work

Training accuracy: 95% CXE 11%, Test accuracy: 72%

Metrics - PyTorch

Both CXE and MSE are loss functions for SimpleLinearNet (not weighted)



CXE alone is the loss function for FrankenNet

```
Checkpoint saved
Epoch 34, Train Accuracy: 0.9788854718208313 , TrainLoss: 0.05941368266940117 , Test Accuracy: 0.7423833608627319
```

nn.CrossEntropyLoss is the PyTorch implementation of CXE, with added features

Results & Discussion

	epoch	train_accuracy	test_accuracy	train_loss	test_loss
0	1.0	51.402680	49.980717	1885.654139	913.702380
1	2.0	51.749735	46.934053	1828.743909	915.688790
2	3.0	52.270317	53.335904	1825.661987	910.474112
3	1.0	51.682252	53.220208	761.783327	912.429865
4	2.0	51.528005	53.335904	731.602085	911.987281
5	3.0	51.923262	53.335904	731.413786	911.017639
6	1.0	52.414923	54.300039	271.848532	308.819530
7	2.0	53.629615	54.030081	245.075522	303.998979
8	3.0	52.723417	54.030081	243.520348	303.816998

	epoch	train_accuracy	test_accuracy	train_loss	test_loss
count	9.000000	9.000000	9.000000	9.000000	9.000000
mean	2.000000	52.147134	52.500321	947.255959	710.215064
std	0.866025	0.705198	2.446014	707.211090	303.509477
min	1.000000	51.402680	46.934053	243.520348	303.816998
25%	1.000000	51.682252	53.220208	271.848532	308.819530
50%	2.000000	51.923262	53.335904	731.602085	911.017639
75%	3.000000	52.414923	54.030081	1825.661987	912.429865
max	3.000000	53.629615	54.300039	1885.654139	915.688790

```
train(net0, train_dataloader0, test_dataloader0,3) # 3 epochs, 4 train images, 2 test images, CPU
```

```
  0%|          | 0/3 [00:00<?, ?it/s]
Accuracy after epoch 1 : 51.40268003470548
Loss after epoch 1 : 0.18178483940338452
Test Accuracy after epoch 1 : 49.98071731585036
Test Loss after epoch 1 : 0.3523726878228723
Accuracy after epoch 2 : 51.74973488865324
Loss after epoch 2 : 0.17629845844771885
Test Accuracy after epoch 2 : 46.934053220208256
Test Loss after epoch 2 : 0.3531387543016265
Accuracy after epoch 3 : 52.27031716957486
Loss after epoch 3 : 0.1760013484051542
Test Accuracy after epoch 3 : 53.33590435788662
Test Loss after epoch 3 : 0.3511276946672821
```

```
train(net1, train_dataloader1, test_dataloader1,3) # 3 epochs, 10 train images, 2 test images, CPU
```

```
  0%|          | 0/3 [00:00<?, ?it/s]
Accuracy after epoch 1 : 51.68225200038562
Loss after epoch 1 : 0.07343905591279783
Test Accuracy after epoch 1 : 53.22020825298882
Test Loss after epoch 1 : 0.3518819378418119
Accuracy after epoch 2 : 51.52800539863106
Loss after epoch 2 : 0.0705294596255946
Test Accuracy after epoch 2 : 53.33590435788662
Test Loss after epoch 2 : 0.3517112536086477
Accuracy after epoch 3 : 51.923262315627106
Loss after epoch 3 : 0.07051130689398298
Test Accuracy after epoch 3 : 53.33590435788662
Test Loss after epoch 3 : 0.3513373078823641
```

```
train(net2, train_dataloader2, test_dataloader2,3)  # 3 epochs, 20 train images, CPU
```

```
  0%|          | 0/3 [00:00<?, ?it/s]
Accuracy after epoch 1 : 52.414923358719754
Loss after epoch 1 : 0.02620732016587779
Test Accuracy after epoch 1 : 54.3000385653683
Test Loss after epoch 1 : 0.11909738896760157
Accuracy after epoch 2 : 53.62961534753688
Loss after epoch 2 : 0.023626291520867
Test Accuracy after epoch 2 : 54.03008098727343
Test Loss after epoch 2 : 0.11723832605062559
Accuracy after epoch 3 : 52.72341656222886
Loss after epoch 3 : 0.023476366290910076
Test Accuracy after epoch 3 : 54.03008098727343
Test Loss after epoch 3 : 0.11716814439727954
```

Results & Discussion

Project challenges/difficulties in Phase 2

- Cuda GPU requirement for PyTorch required use of Google Colab (team working from Mac laptops that did not have applicable GPUs)
- Issues with linking Google Drive (image set and data) reliably due to known issue/race condition (would freeze/hang on image open, fail to find image file)
- Confusion on how to format parameters for Conv2d
- Low performance on linear neural network ~ 53% indicates a need for other transformation steps and different layer options in neural net setup
- Can't use Tensorboard for SummaryWriter nice output on Google Colab
- FrankenNet seems to be overfitting

Conclusion

We aimed to train a model to predict bounding boxes based on provided images and then predict whether each image was a dog or cat as a classification step, using PyTorch and neural networks. Image classification is a complex machine learning problem. Focusing on a subset of data allowed a short-term project to be approachable. Class prediction based on bounding-boxes alone doesn't seem to indicate a high probability of success. We achieved about 53% accuracy on SimpleLinear and intended to refine FrankenNet transformations to reduce overfitting, and time limitations affected our outcome. Two of our three model training processes function properly, but the more promising requires more work on bounding box predictions. We hope to refine FrankenNet for our final submission.

Next Steps:

Tune FrankenNet model and image transformations to improve prediction performance, accuracy, and fix overfitting problem.