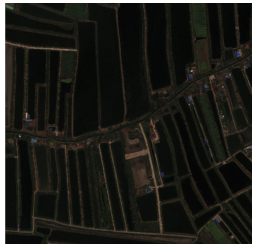
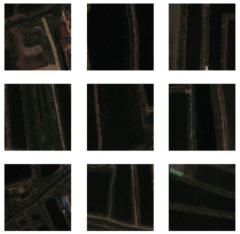
# xView Final Project

Machine Learning II DATS 6203 - 11

Group 2: Jiarong Che, Diana Holcomb, Jiajun Wu

# xView Challenge





xView contains images from complex scenes around the world. It is the largest publicly available sets of overhead imagery to date.

- Over 1 million objects across 60 classes in over 1,400 square km of imagery.

Compared to other overhead imagery datasets, xView images are high-resolution, multi-spectral, and labeled with a greater variety of objects. The solution of xView data is 0.3 meters per pixel, that means this is the highest resolution that we can get from satellites currently.

The DIUx xView Challenge occurred from March to September in 2018, and its goal was to apply computer vision to the growing amount of available imagery from space so that we can understand the visual world in new ways and address a range of important applications. It focused on four computer vision frontiers:

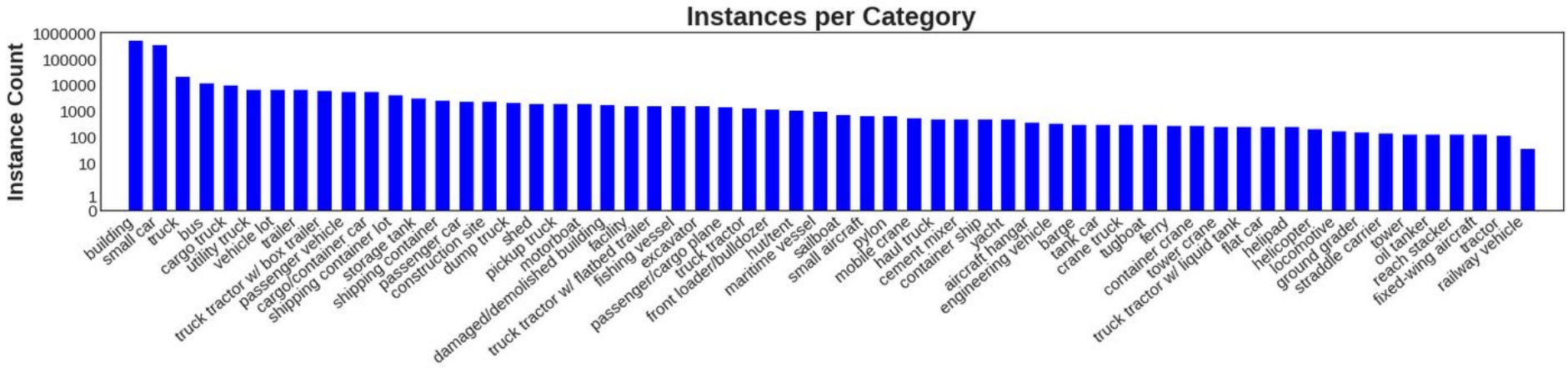
- Reduce minimum resolution for detection;
- Improve learning efficiency;
- Enable discovery of more object classes;
- Improve detection of fine-grained classes

Website:
https://challenge.xviewdataset.org/challenge-description

# Dataset

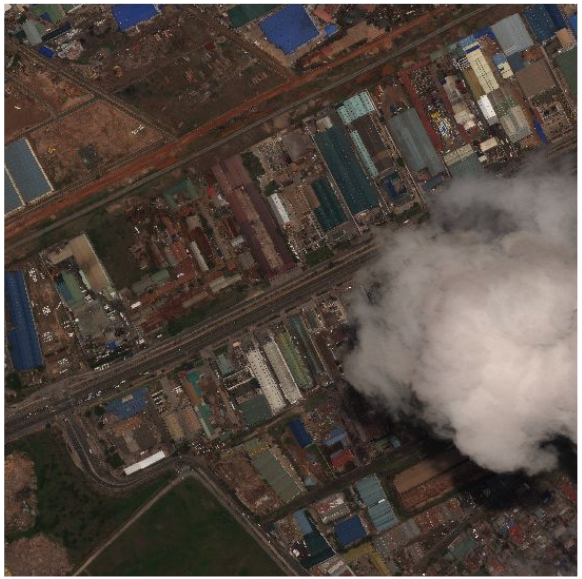
The xView dataset is 1400 high-resolution satellite images and a geojson file of ~1 million ground truth annotations

- The 1 million bounding boxes represent a diverse range of 60 object classes.
- These classes include different types of vehicles, boats, aircrafts, buildings and so on.



# Dataset

Example Images



The xView images and ground truth can be obtained here: https://challenge.xviewdataset.org/download-links
You must create an account first, and then log in. Logging in always happens from the “Register” page, which is not the most intuitive.

# Data Analysis - Preprocessing

The images of xView are so large that even a small training set (379 images) will not fit in memory on GCP. We decided to break the large images up into chips, and to run our code in minibatches.

- With a smaller chip size, like 28 in MNIST or FashionMNIST, the amount of chipped images became enormous.
- With a bigger chip size, like 800x800, we ran out of memory processing the chip.
- We decided to chip each image into 300x300 chips, which still resulted in 35858 chipped images.

# Data Analysis - Tensorflow Baseline Models

The xView dataset comes with three pretrained Tensorflow models, available here: https://github.com/DIUx-xView/baseline/releases

- Vanilla - the original out-of-the box model, all image chips 300x300
- MultiRes - model with variable image chip sizes (300x300, 400x400, 500x500)
- MultiRes_Aug - the MultiRes model, with additional augmentation in the form of shifting, rotation, noise, and blurring

Vanilla

MultiRes Aug

# Data Analysis - PyTorch CNN

**Challenge 1**: DataLoader
There is not out-of-the-box Torchvision DataSet like our in-class examples (torchvision.datasets.MNIST, for example). Creating our own xView-specific DataSet was time consuming and difficult. Most online PyTorch examples used datasets that were already included in Torchvision, so examples of how to create our own were sparse and very problem-specific.

**Challenge 2**: Imagery Formatting
NumPy and image-specific libraries such as PIL parse an image as (height, width, channels). PyTorch needs images to be formatted as (channels, height, width).

**Challenge 3:** Target Array Formatting
All of the PyTorch examples we did in class had one annotation per image.
But for xView dataset, one image has many annotations, and different images have different numbers of annotations

# Data Analysis - PyTorch CNN

Based on in-class PyTorch CNN example with MNIST and convolutional layers

```
# CNN Model (2 conv layer)
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(in_channels, conv_size, kernel_size=kernel_size, padding=2),
            nn.BatchNorm2d(conv_size),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(conv_size, conv_size*2, kernel_size=kernel_size, padding=2),
            nn.BatchNorm2d(conv_size*2),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.fc = nn.Linear(3375000, num_classes) #conv_size**2 * in_channels * (co

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        return out
```

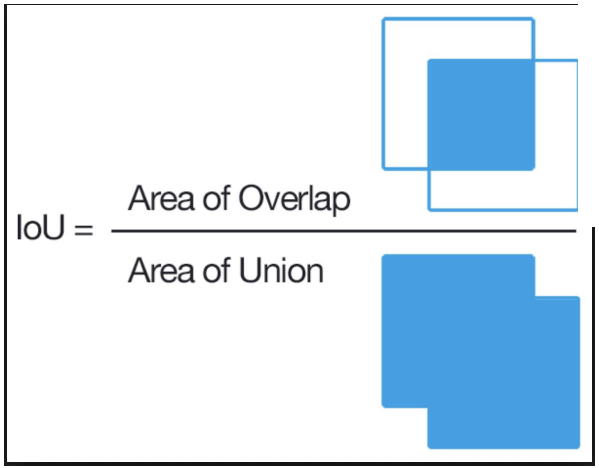
# Data Analysis - Evaluation

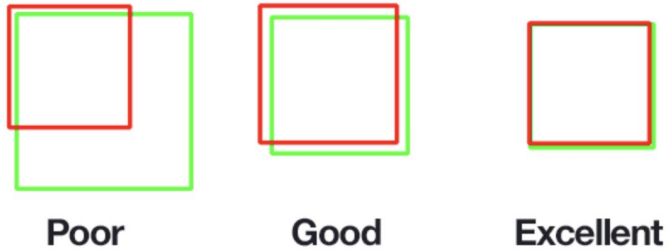
The xView team evaluated their pretrained Tensorflow models with a method known as mAP, or mean average precision. This method is used in Computer Vision problems where both the classification *and* localization need to be judged.

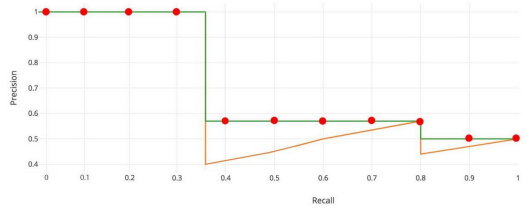
$$\text{MAP} = \frac{\sum_{q=1}^{Q} \text{AveP(q)}}{Q}$$

where *Q* is the number of class in the set, and AveP(q) is the average precision (AP) for a given class, q.

Interpolated mAP metric used for primary evaluation to measure the overlap between the true bounding boxes and predicted bounding boxes.



# Evaluation - Tensorflow Baseline Models

| | Vanilla | Multires | Aug |
|---|---|---|---|
| Aircraft Hangar | 0.1698 | 0.5270 | 0.3247 |
| Barge | 0.1829 | 0.3738 | 0.2210 |
| Building | 0.4718 | 0.5534 | 0.4451 |
| Bus | 0.2949 | 0.3773 | 0.2609 |
| Cargo Truck | 0.0493 | 0.0972 | 0.0445 |
| Cargo/container car | 0.3659 | 0.4737 | 0.1676 |
| Cement mixer | 0.0863 | 0.1441 | 0.1220 |
| Construction site | 0.0172 | 0.1711 | 0.0032 |
| Container crane | 0.0663 | 0.2879 | 0.1648 |
| Container ship | 0.2269 | 0.4660 | 0.3400 |
| Crane Truck | 0.0946 | 0.0838 | 0.0894 |
| Damaged/demolished building | 0.0269 | 0.0785 | 0.0366 |
| Dump truck | 0.1468 | 0.2275 | 0.0858 |
| Engineering vehicle | 0.0020 | 0.1234 | 0.0357 |
| Excavator | 0.3535 | 0.4691 | 0.2064 |
| Facility | 0.0777 | 0.3750 | 0.1201 |
| Ferry | 0.0532 | 0.3771 | 0.2197 |
| Fishing vessel | 0.1768 | 0.1839 | 0.0968 |
| Fixed-wing aircraft | 0.0888 | 0.1218 | 0.1042 |
| Flat car | 0.0000 | 0.0000 | 0.0000 |
| Front loader/Bulldozer | 0.1644 | 0.3220 | 0.1959 |
| Ground grader | 0.1590 | 0.1910 | 0.0289 |
| Haul truck | 0.3542 | 0.2109 | 0.6875 |
| Helicopter | 0.3788 | 0.5800 | 0.2965 |
| Helipad | 0.2459 | 0.4500 | 0.1889 |
| Hut/Tent | 0.0004 | 0.0006 | 0.0000 |
| Locomotive | 0.0760 | 0.1929 | 0.1124 |
| Maritime vessel | 0.1947 | 0.4040 | 0.2884 |
| Mobile crane | 0.0248 | 0.1375 | 0.0945 |
| Motorboat | 0.0811 | 0.2488 | 0.1110 |
| Oil Tanker | 0.0958 | 0.3677 | 0.1193 |
| Passenger Vehicle | 0.4765 | 0.5569 | 0.2980 |
| Passenger car | 0.0305 | 0.0471 | 0.0000 |
| Passenger/cargo plane | 0.6508 | 0.6691 | 0.6104 |

The xView team evaluated their pretrained Tensorflow models with a method known as mAP, or mean average precision.

MultiRes performed the best, with a mAP of 0.2590. Vanilla performed the worst with a mAP of 0.1456. MultiRes_Aug has a mAP of .1549.

Best detected classes per model:

- Vanilla: passenger/cargo plane, small aircraft, building, passenger car, cargo/container car
- MultiRes: passenger/cargo plane, helicopter, shipping container lot, passenger car, building
- MultiRes_Aug: haul truck, passenger/cargo plane, small aircraft, building, tugboat

# Future Work


- Create the equivalent of the MultiRes (varied image chip sizes) and MultiRes_Aug (shifting, rotation, noise, and blurring) in PyTorch and compare the results against the Tensorflow model.
- Try various optimizers in PyTorch to see how accuracy and convergence time change. Currently we use Adam, but it would be interesting to compare with some of the more successful optimizers from our FashionMNIST project.
- Find a better way to deal with the variable target array lengths than simple padding.
- Try oversampling and undersampling.  The distribution of the classes is not even, with many more building, cars, trucks, and buses than fixed-wing aircraft, tractors, and railway vehicles.

# Conclusion

Processing xView data in PyTorch proved to be much more challenging than our FashionMNIST assignment.

- There are a lot of multi-class per image and xView-specific examples in TensorFlow and Keras, but not a lot in PyTorch.
- PyTorch doesn't easily accommodate variable length in the target arrays

We had hoped to do a comparison of Tensorflow and PyTorch over a large overhead imagery dataset, but got too bogged down in the engineering aspects.

Our current challenge is properly calculating loss from a dataset with a variable length of target arrays.

PyTorch

```
RuntimeError: multi-target not supported at /pytorch/aten/src/THCUNN/generic/ClassNLLCriterio
n.cu:15
```

# Questions???

# References

- xView Competition website: https://challenge.xviewdataset.org
- xView dataset and ground truth geojson: https://challenge.xviewdataset.org/download-links
- Pretrained Tensorflow models: https://github.com/DIUx-xView/baseline/releases
- xView Pre-Processing code: https://github.com/DIUx-xView/data_utilities
- xView Baseline TensorFlow code: https://github.com/DIUx-xView/baseline
- xView Baseline Docker container: https://hub.docker.com/r/xview2018/baseline/tags