W207

Pedestrian Detection

Evaluating the Performance of Various Machine Learning Algorithms

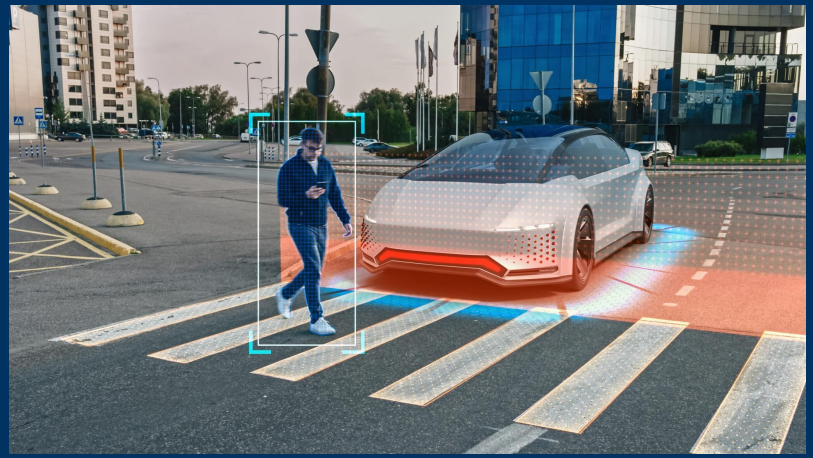
Gautam Sai Yarramreddy, Jacob Lu, Daniel Chung

UC Berkeley

Motivation

Question: How effective are different machine learning models at identifying pedestrians on the street?

Motivation: Pedestrian detection is critical for the development of safe autonomous vehicles



UC Berkeley

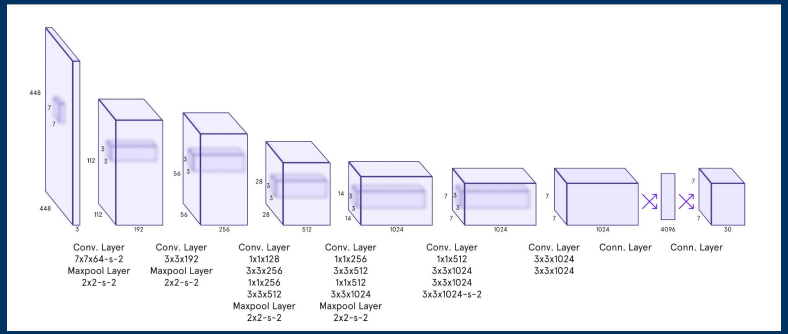
Motivation

Current Advancements:

- Very advanced algorithms such as YOLO and Faster R-CNN with advantages in different scenarios
  - E.g. GR-YOLO for pedestrian in crowded areas, YOLO-GW for fog or low light

Our Approach:

- Test the ability of machine learning algorithms to identify pedestrians and compare results with YOLO algorithm



UC Berkeley

Data

- Collected from Waymo’s autonomous vehicles using 5 cameras and 5 lidars
- Has 11.8 million 2D bounding box annotations on camera images
    - Covers objects like vehicles, pedestrians, cyclists, and traffic signs
- Complements other datasets with:
    - 12.6 million 3D bounding boxes (lidar)
    - 3D semantic segmentation labels
- Good for object detection, tracking, and scene understanding for autonomous driving

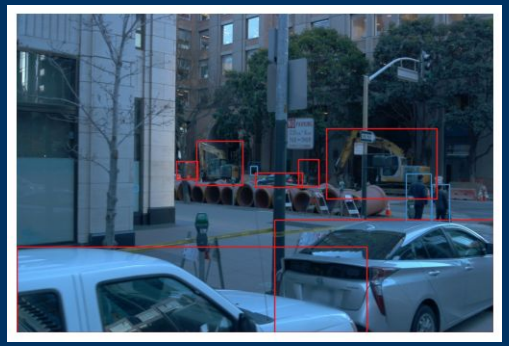
UC Berkeley

Data

- 799 scenes across San Francisco

UC Berkeley

Data

Time	Left	Front	Right
15524401 95462613			
.	.	.	
.	.	.	
.	.	.	

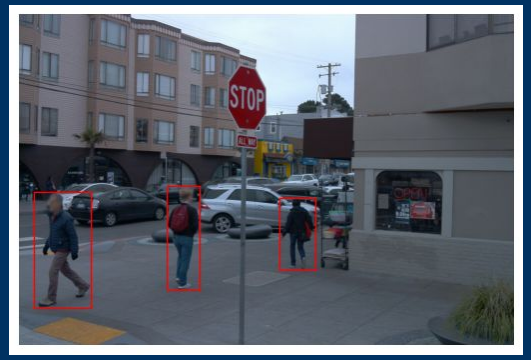
UC Berkeley

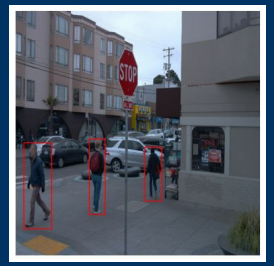
Data

- Use the first 100 scenes in the dataset
- Randomly select 50 images in each scene
- Add image if having > 0.5% of the area with a pedestrian

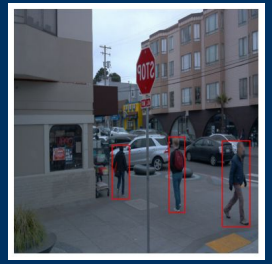
UC Berkeley

Data

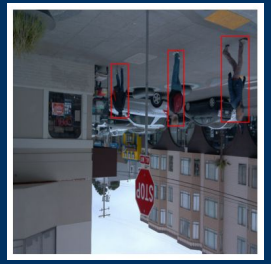




Resize (640x640)



Random Horizontal Flip



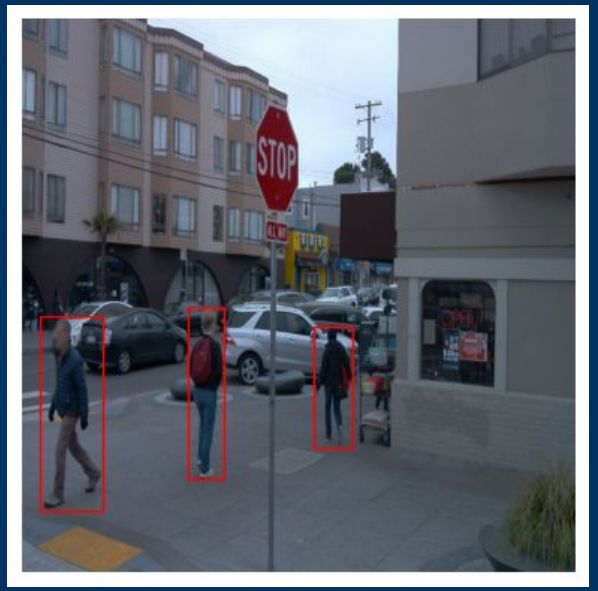
Random Vertical Flip

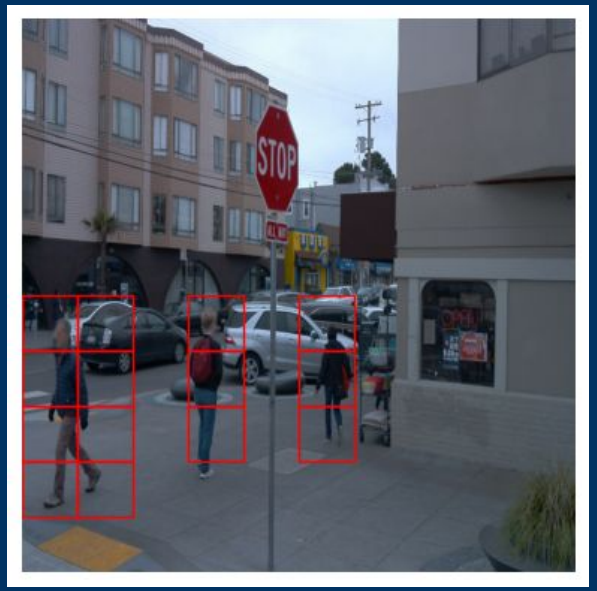
Random 90 ° Rotation

UC Berkeley

Data

- Transform images into grid of subimages (64x64 pixels each)
- Subimage is labeled as having a pedestrian if >25% of the subimage has a pedestrian bounding box





UC Berkeley

Data

- Processed images were segmented with a grid pattern into sub-images
  - Only used for Logistic Regression and CNN
- However this created class imbalance
- So, we downsampled the negative class in the training and validation classes, and did not touch the final test

Figure: Positive class examples

	Training	Validation	Test	Test Final
Positive Images	3355	1102	1084	1858
Negative Images	6618	2222	2242	22042
Total	9973	3324	3326	23900

UC Berkeley

Logistic Regression

Sklearn Model

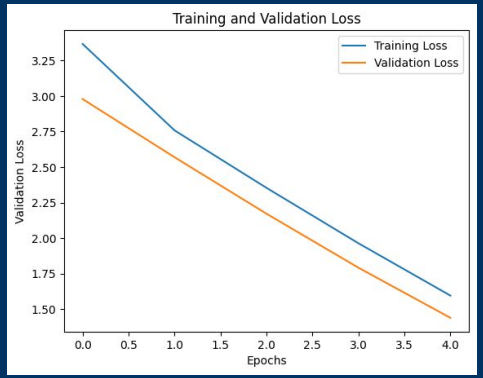
sklearn Logistic Regression Model

```
[17] # Build Logistic Regression Model Using SKlearn
     from sklearn.linear_model import LogisticRegression

     lr_model = LogisticRegression(max_iter=10)
     lr_model.fit(X_train_new_scaled, y_train_new)

     # Check Validation Accuracy
     accuracy_val = lr_model.score(X_val_new_scaled, y_val_new)
     print(f"Validation Accuracy: {accuracy_val}")
```

```
Validation Accuracy: 0.6585439229843562
```

Validation Accuracy = 0.65

Tensorflow Model



Validation Accuracy = 0.51

UC Berkeley

Logistic Regression – Tensorflow

Sklearn Model Hyperparameters

- Batchsize=16
- Epochs=5
- Loss function: Binary Cross Entropy
- Optimizer: SGD

Logistic Regression – Analysis

- No spatial information due to flattening the data into a 1D array
- There are 12288 features which may cause high dimensionality

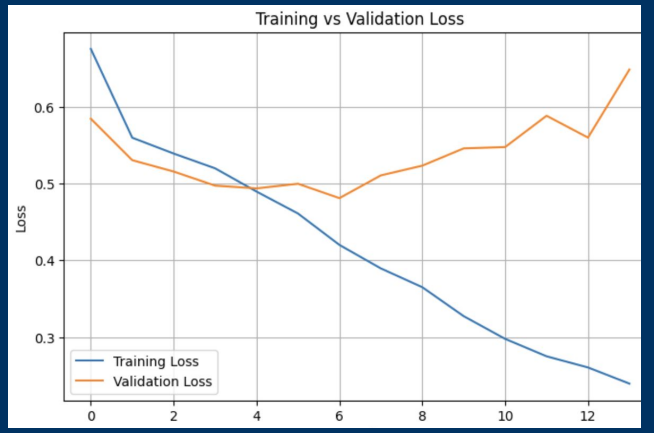
CNN

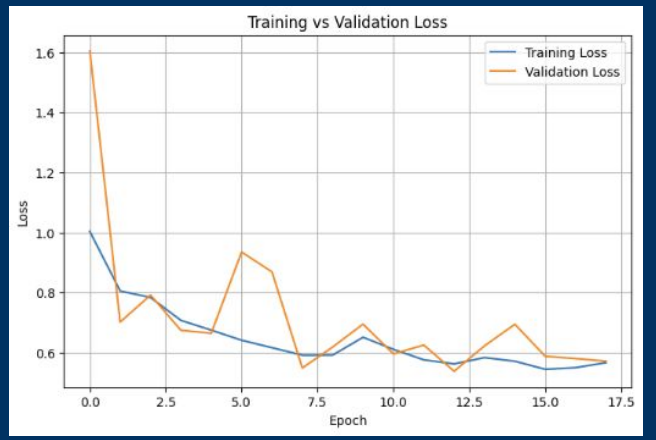
Baseline: Similar CNN architecture from Homework 10

- Early stopping (this time monitoring validation AUC instead of accuracy)
- Metrics: accuracy and AUC

Improved

- Incorporated BatchNorm layers
- More Conv2D + BatchNorm + Max Pooling layers
- More Dropout layers
- L2 Regularization





UC Berkeley

CNN

Hyperparameters

- Batchsize=32
- Epochs=25
  - Early stopping restored weights from epoch 13
- Loss function: Binary Cross Entropy
- Optimizer: Adam

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 32)	896
batch_normalization (BatchNormalization)	(None, 64, 64, 32)	128
conv2d_1 (Conv2D)	(None, 64, 64, 32)	9,248
batch_normalization_1 (BatchNormalization)	(None, 64, 64, 32)	128
max_pooling2d (MaxPooling2D)	(None, 32, 32, 32)	0
dropout (Dropout)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 32, 32, 64)	18,496
batch_normalization_2 (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_3 (Conv2D)	(None, 32, 32, 64)	36,928
batch_normalization_3 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_1 (Dropout)	(None, 16, 16, 64)	0
flatten (Flatten)	(None, 16384)	0
dense (Dense)	(None, 128)	2,097,280
dropout_2 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129

Total params: 2,163,745 (8.25 MB)
Trainable params: 2,163,361 (8.25 MB)
Non-trainable params: 384 (1.50 KB)

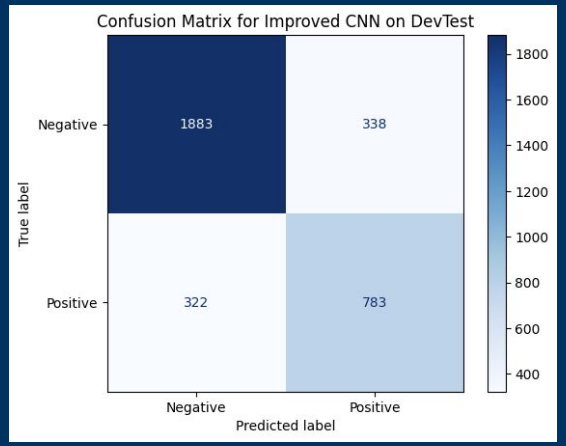
UC Berkeley

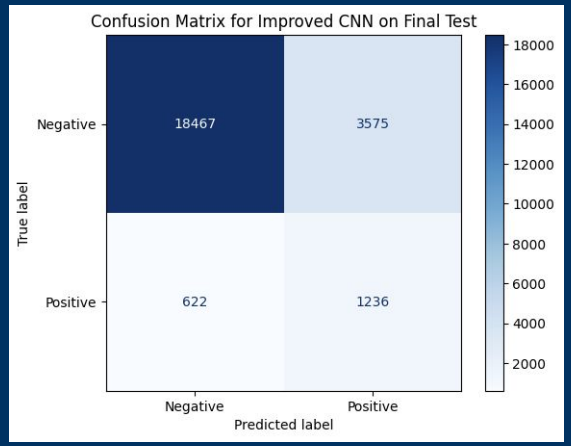
CNN Evaluation

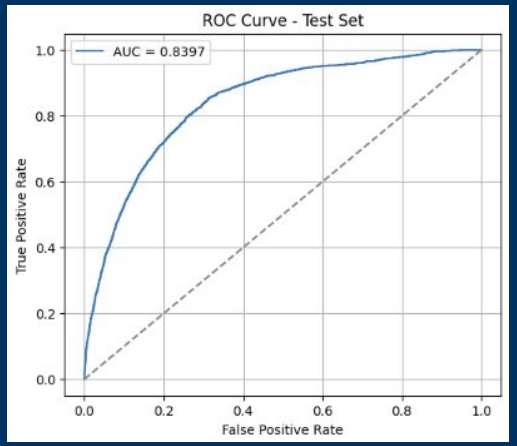
Improved model showed better generalizability

- Evaluation was similar in both test sets
- Great AUC value of 0.84
- FP rate was higher with final test set
- FN rate is biggest weakness in this application

```
Test Loss: 0.4434
Test Accuracy: 0.8244
Test AUC: 0.8396
```



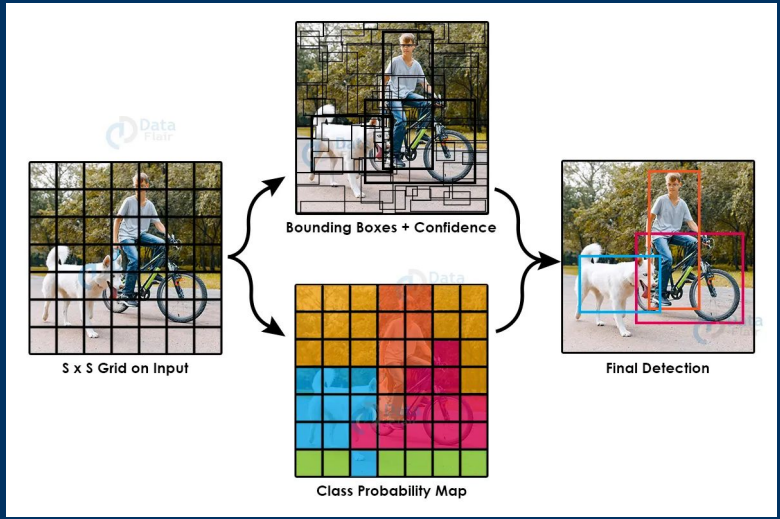




UC Berkeley

YOLO

- You Only Look Once (YOLO)
- Designed for object detection and classification using bounding boxes
- Pretrained on the PASCAL VOC dataset
- Compound Loss Function
    - box_loss: how far the box is from ground truth (both size and location)
    - cls_loss: The class prediction
    - obj_loss: The prediction confidence



UC Berkeley

YOLO

Ground Truth

Predictions

box_loss	obj_loss	cls_loss	total
1.29	1.09	0.278	2.658

UC Berkeley

YOLO

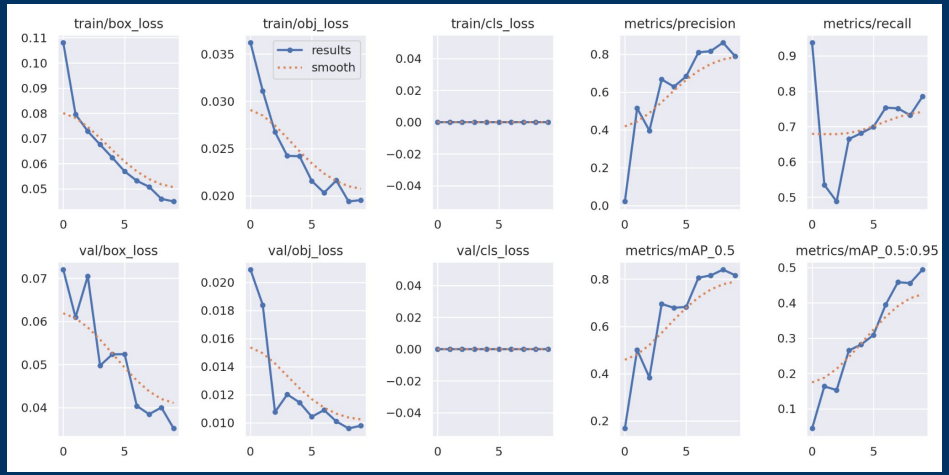
Batch Size	Epochs	Optimizer	Train Loss	Val Loss	Recall
8	5	SGD	0.083981	0.062203	0.67763
8	5	AdamW	0.096923	0.080926	0.41941
8	10	SGD	0.064414	0.046944	0.77632
8	10	AdamW	0.088209	0.066129	0.48054
16	5	SGD	0.085662	0.061643	0.66244
16	5	AdamW	0.093176	0.194419	0.52813
16	10	SGD	0.064542	0.044955	0.78456
16	10	AdamW	0.085590	0.064685	0.49178

- SGD better for finetuning than AdamW
- Larger batch size is generally better
- More epochs is marginally better

UC Berkeley

YOLO

- Loss consistently decreases over epochs
  - cls_loss is always 0 because we only have one class
- mAP (mean average precision) consistently increasing
- Recall was initially very high, decreased, then started improving again



UC Berkeley

YOLO

Ground Truth

Predictions

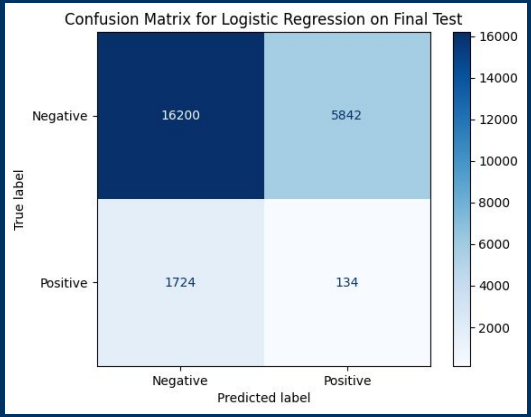
UC Berkeley

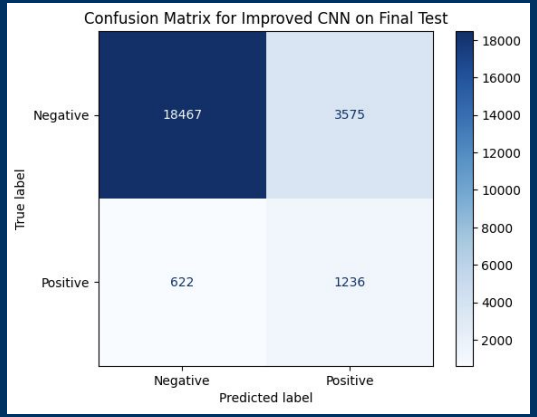
Conclusion

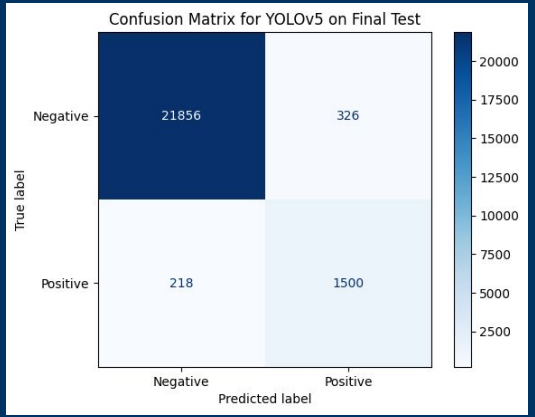
Logistic Regression:
Test Loss: 1.02
Test Accuracy: 0.683

CNN:
Test Loss: 0.4434
Test Accuracy: 0.8244
Test AUC: 0.8396

YOLO:
Test Loss*: 0.049308
Test Accuracy**: 0.977
Test AUC**: 0.9292







* YOLO uses a different loss function than LR and CNN methods
** Predictions from YOLO are bounding boxes which are then converted to similar format as the LR and CNN data for comparison purposes

UC Berkeley

Future Directions

Model Improvements:

- Fine-tuning YOLO directly with Waymo Data
- Apply vision transformers, which have recently shown strong results in object detection tasks

Data and Preprocessing

- Incorporate the 3D lidar point clouds for richer feature sets and improved model accuracies, especially in occlusion-heavy or low-light scenarios

Other Object Detection Models

- Test other models such as Faster R-CNN, RetinaNet, and SSD

Thank you

Any questions?

UC Berkeley

Contributions

Github:

Daniel Chung: I worked on the motivation slides, two of the data slides, future directions, and the CNN portion of coding and slides. I also used the functions Gautam provided to create the subimages for the Logistic Regression and CNN parts of the project.

Jacob Lu: I worked on the evaluation slide for the baseline and logistic regression slides of the final presentation. Additionally, I did additional preprocessing of the logistic regression and creating both the sklearn and TF logistic regression modeling/fitting.

Gautam Sai Yarramreddy: I worked on creating the dataset used in the LR, CNN, and YOLO models from the raw waymo data (both the bounding boxes and patches datasets). I also worked on creating the proper structure and datasets files needed for training the YOLOv5 model. I worked on finetuning and optimizing the hyperparameters on the YOLO model.