ANOMALY DETECTION using AI

```
del(Model):
init__(self):
per().__init__()
lf.res1 = layers.Rescaling(1./255, input_shape=(img_height, img_wi
lf.conv1 = layers.Conv2D(16, 3, padding='same', activation='relu')
lf.max = layers.MaxPooling2D()
lf.conv2 = layers.Conv2D(32, 3, padding='same', activation='relu')
lf.conv3 = layers.Conv2D(64, 3, padding='same', activation='relu')
lf.drop = layers.Dropout(0.05)
lf.flat = layers.Flatten()
lf.d1 = layers.Dense(128, activation='relu')
lf.d2 = layers.Dense(60)

ll(self, x):
= self.res1(x)
= self.conv1(a)
= self.max(b)
= self.conv2(c)
= self.max(d)
= self.conv3(e)
= self.max(f)
= self.drop(g)
= self.flat(h)
= self.d1(i)

turn self.d2(j), j
```

AGENDA

Project overview

Data processing

Creating our classification model

Anomaly detection

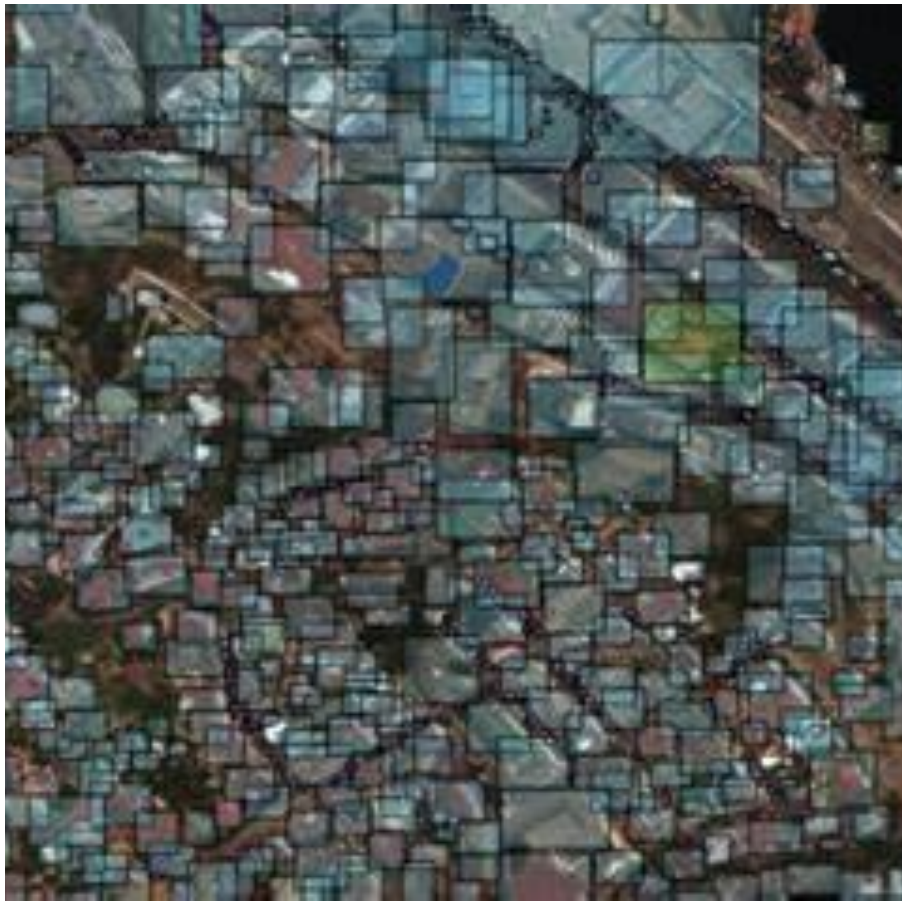
Results

Improvements

Project Overview

- Customer trust in AI models can be challenging to establish and maintain. As an example of this, models that classify objects in imagery typically produce moderately high class confidences for images of classes that were not available in the training, calling their self-reported confidence values into question.
- Use the xView dataset of overhead imagery and associated labeled bounding boxes to train an object classification model. Use this model to predict whether new images are of a known class or unknown (“anomalous”).

Used Tensorflow with Keras, pandas, PIL, numpy



Data Processing

- Data sources:
  - Zip files containing TIFF (.tif) images of overhead imagery
  - Geojson (.geojson) file containing labels for objects within TIFF images

Goal:

- Obtain cropped images with associated labels for training our model

{"crs": {"properties": {"name": "urn:ogc:def:crs:OGC:1.3:CRS84"}, "type": "name"}, "type": "FeatureCollection", "features": [{"type": "Feature", "properties": {"bounds_imcoords": "2712,1145,2746,1177", "edited_by": "wwoscarbecerril", "cat_id": "1040010028371A00", "type_id": 73, "ingest_time": "2017/07/24 12:49:09.118+00", "index_right": 2356, "image_id": "2355.tif", "point_geom": "0101000020E6100000616E4E6406A256C03BE6ADA0D6212D40", "feature_id": 374410, "grid_file": "Grid2.shp"}, "geometry": {"type": "Polygon", "coordinates": [[[-90.53169885094464, 14.56603647302396], [-90.53169885094464, 14.56614473506768], [-90.53158140073565, 14.56614473506768], [-90.53158140073565, 14.56603647302396], [-90.53169885094464, 14.56603647302396]]]}}, {"type": "Feature", "properties": {"bounds_imcoords": "2720,2233,2760,2288", "edited_by": "wwoscarbecerril", "cat_id": "1040010028371A00", "type_id": 73, "ingest_time": "2017/07/24 17:26:05.701+00", "index_right": 2356, "image_id": "2355.tif", "point_geom": "0101000020E6100000042D0CC705A256C0004F7071E71F2D40", "feature_id": 394393, "grid_file": "Grid2.shp"}, "geometry": {"type": "Polygon", "coordinates": [[[-90.53167232380382, 14.562217332510999], [-90.53167232380382, 14.562407959236182], [-90.53153294103244, 14.562407959236182], [-90.53153294103244, 14.562217332510999], [-90.53167232380382, 14.562217332510999]]]}}, {"type": "Feature", "properties": {"bounds_imcoords": "2687,1338,2740,1399", "edited_by": "wwoscarbecerril", "cat_id": "1040010028371A00", "type_id": 73, "ingest_time": "2017/07/24 12:45:09.081+00", "index_right": 2356, "image_id": "2355.tif", "point_geom": "0101000020E6100000B29F7E4707A256C0000AE65379212D40", "feature_id": 374031, "grid_file": "Grid2.shp"}, "geometry": {"type": "Polygon", "coordinates": [[[-90.53178519354792, 14.565273205700436], [-90.53178519354792, 14.565484358521783], [-90.53160338993305, 14.565484358521783], [-90.53160338993305, 14.565273205700436], [-90.53178519354792, 14.565273205700436]]]}}, {"type": "Feature", "properties": {"bounds_imcoords": "2691,1201,2730,1268", "edited_by": "wwoscarbecerril", "cat_id": "1040010028371A00", "type_id": 73, "ingest_time": "2017/07/24 12:49:09.118+00", "index_right": 2356, "image_id": "2355.tif", "point_geom": "0101000020E6100000CE53137207A256C00EBF7084B5212D40", "feature_id": 374409, "grid_file": "Grid2.shp"}, "geometry": {"type": "Polygon", "coordinates": [[[-90.53177155821692, 14.56572251279647], [-90.53177155821692, 14.565953472087747], [-90.53163732940662,

Data Processing

Our approach:

- Read geojson file contents into a pd dataframe containing the img_ids, coordinates, and classes for all objects.
- Load overhead images and use dataframe to determine coordinates for images and crop accordingly (padding based on image size, keep within image bounds).
- Assign cropped data in directory structure based on class label

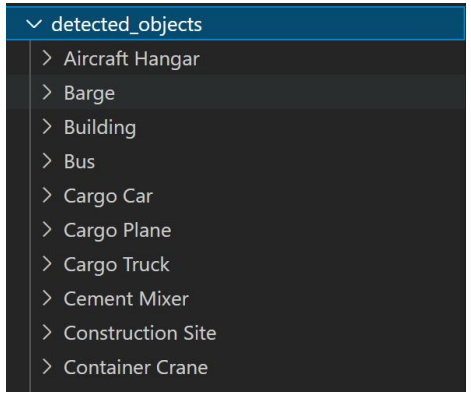
```
def read_geojson():
    with open('/domino/datasets/local/train_labels/xView_train.geojson') as f:
        data = json.load(f)

    num_features = len(data['features'])

    img_ids = np.zeros(num_features, dtype="object")
    coords = np.zeros((num_features, 4))
    classes = np.zeros(num_features)
    ids = np.zeros(num_features)

    # Process every feature obj
    for i in range(num_features):
        if data['features'][i]['properties']['bounds_imcoords'] != []:
            img_ids[i] = data['features'][i]['properties']['image_id']
            coords[i] = np.array([int(num) for num in data['features']
                                  [i]['properties']['bounds_imcoords'].split(",")])
            classes[i] = data['features'][i]['properties']['type_id']
            ids[i] = i


    dataset = pd.DataFrame({'ids': list(ids), 'img_id': list(img_ids), 'coordinates': list(coords), '
    # can use if you want to make a csv to see the data easily
    # dataset.to_csv('data.csv')
    return dataset
```



Creating the Initial Classification Model

Used a CNN-based model consisting of three convolution blocks. Dropout was added to help reduce overfitting. The model was written as a class with layers added one at a time to make it easier to grab the latent vectors at any point within the model.

```
class MyModel(Model):
    def __init__(self):
        super().__init__()
        self.res1 = layers.Rescaling(1./255, input_shape=(img_height, img_width, 3))
        self.conv1 = layers.Conv2D(16, 3, padding='same', activation='relu')
        self.max = layers.MaxPooling2D()
        self.conv2 = layers.Conv2D(32, 3, padding='same', activation='relu')
        self.conv3 = layers.Conv2D(64, 3, padding='same', activation='relu')
        self.drop = layers.Dropout(0.05)
        self.flat = layers.Flatten()
        self.d1 = layers.Dense(128, activation='relu')
        self.d2 = layers.Dense(60)

    def call(self, x):
        a = self.res1(x)
        b = self.conv1(a)
        c = self.max(b)
        d = self.conv2(c)
        e = self.max(d)
        f = self.conv3(e)
        g = self.max(f)
        h = self.drop(g)
        i = self.flat(h)
        j = self.d1(i)

        return self.d2(j), j
```

Creating the Initial Classification Model

We used 20 epochs and Adam as the optimizer with Sparse Categorical Crossentropy loss and Sparse Categorical accuracy.

```
train_ds, val_ds, class_names = split_images()
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

model = MyModel()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
optimizer = tf.keras.optimizers.Adam()

train_loss = tf.keras.metrics.Mean(name='train_loss')
train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='train_accuracy')

test_loss = tf.keras.metrics.Mean(name='test_loss'8)
test_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='test_accuracy')
```

```
EPOCHS = 20

for epoch in range(EPOCHS):
    train_loss.reset_states()
    train_accuracy.reset_states()
    test_loss.reset_states()
    test_accuracy.reset_states()

    for images, labels in train_ds:
        train_step(images, labels)

    labels = []
    latent_vectors = []

    for test_images, test_labels in val_ds:
        latent = test_step(test_images, test_labels)
        labels.append(test_labels)
        latent_vectors.append(latent)

    label_to_vector = {}

    for i in range(len(labels)):
        for x in range(len(labels[i].numpy())):
            if labels[i].numpy()[x] in label_to_vector.keys():
                label_to_vector[labels[i].numpy()[x]].append(latent_vectors[i].numpy()[x])
            else:
                label_to_vector[labels[i].numpy()[x]] = [latent_vectors[i].numpy()[x]]

    medians = calculate_medians(label_to_vector)

    print(
        f'Epoch {epoch + 1}, '
        f'Loss: {train_loss.result()}, '
        f'Accuracy: {train_accuracy.result() * 100}, '
        f'Test Loss: {test_loss.result()}, '
        f'Test Accuracy: {test_accuracy.result() * 100}'
    )
```

Anomaly Detection

Once the model is trained, grab the latent vectors before the output layer of the model during the test step.

This will give an n-dimensional (based on Dense layer size) space representation of each class since we will have a n-dimensional labeled point for each test image.

```
@tf.function
def train_step(images, labels):
  with tf.GradientTape() as tape:
    # training=True is only needed if there are layers with different
    # behavior during training versus inference (e.g. Dropout).
    predictions, _ = model(images, training=True)
    loss = loss_object(labels, predictions)
  gradients = tape.gradient(loss, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))

  train_loss(loss)
  train_accuracy(labels, predictions)


@tf.function
def test_step(images, labels):
  # training=False is only needed if there are layers with different
  # behavior during training versus inference (e.g. Dropout).
    predictions, latent = model(images, training=False)

    t_loss = loss_object(labels, predictions)

    test_loss(t_loss)
    test_accuracy(labels, predictions)

    return latent
```

Anomaly Detection

For each class, calculate max/min (or X percentile) and median values in each of the n-dimensions.

Run new images through the model to get their latent representation. Compare that with the max/min/medians for each class in each dimension to predict whether the new image belongs to a known class or is an anomaly.

```python
def calculate_medians(labels_to_vectors):
    label_to_median = {}
    for label in labels_to_vectors.keys():
        multiple_lists = labels_to_vectors[label]
        arrays = [np.array(x) for x in multiple_lists]
        medians = [np.percentile(k, 50) for k in zip(*arrays)]
        fifths = [np.percentile(k, 0) for k in zip(*arrays)]
        ninetyfifths = [np.percentile(k, 100) for k in zip(*arrays)]
        all_values = []
        for i in range(len(medians)):
            all_values.append([fifths[i],medians[i],ninetyfifths[i]])

        label_to_median[label] = all_values
    return label_to_median
```

```python
def predict(vector, medians):
    options = []

    for item in medians.items():
        new_medians = []
        count = 0
        for i in range(len(item[1])):
            if item[1][i][0] > vector[i] or item[1][i][2] < vector[i]:
                count = 0
                new_medians = []
                break

            else:
                count += 1
                new_medians.append(item[1][i][1])
                if count == len(vector):
                    options.append((str(item[0]), distance.euclidean(vector, new_medians)))

    if len(options) == 0:
        return [str(key), 'anomaly']
    else:
        sortedoptions = sorted(options, key=lambda x:x[1])
        return [str(key), str(sortedoptions[0][0])]
```

Results - Model

We created a pretty successful classification model.

With 20 epochs, we reached 99.08% train accuracy and 86.34% test accuracy.

```
Epoch 1, Loss: 0.8763441443443298, Accuracy: 80.59548950195312, Test Loss: 0.6624085307121277, Test Accuracy: 84.48999786376953
Epoch 2, Loss: 0.5920321345329285, Accuracy: 85.61035919189453, Test Loss: 0.5528571009635925, Test Accuracy: 86.93000030517578
Epoch 3, Loss: 0.4670892059803009, Accuracy: 87.99030303955078, Test Loss: 0.5710568428039551, Test Accuracy: 86.62999725341797
Epoch 4, Loss: 0.35996222496032715, Accuracy: 90.250244140625, Test Loss: 0.6024032235145569, Test Accuracy: 86.3699951171875
Epoch 5, Loss: 0.26884254813194275, Accuracy: 92.43018341064453, Test Loss: 0.6635093092918396, Test Accuracy: 85.88999938964844
Epoch 6, Loss: 0.20503033697605133, Accuracy: 94.0051498413086, Test Loss: 0.7997263669967651, Test Accuracy: 86.29000091552734
Epoch 7, Loss: 0.1526491641998291, Accuracy: 95.47261810302734, Test Loss: 0.8743302822113037, Test Accuracy: 85.29000091552734
Epoch 8, Loss: 0.11377345025539398, Accuracy: 96.45758819580078, Test Loss: 0.9693673849105835, Test Accuracy: 85.37999725341797
Epoch 9, Loss: 0.09126788377761841, Accuracy: 97.132568359375, Test Loss: 1.121343731880188, Test Accuracy: 86.44999694824219
Epoch 10, Loss: 0.07744744420051575, Accuracy: 97.54255676269531, Test Loss: 1.3255126476287842, Test Accuracy: 85.50999450683594
Epoch 11, Loss: 0.06361482292413712, Accuracy: 98.04254913330078, Test Loss: 1.307328224182129, Test Accuracy: 85.30999755859375
Epoch 12, Loss: 0.053126174956560135, Accuracy: 98.37004089355469, Test Loss: 1.4949190616607666, Test Accuracy: 85.65999603271484
Epoch 13, Loss: 0.04980340227484703, Accuracy: 98.5600357055664, Test Loss: 1.5803717374801636, Test Accuracy: 86.56999969482422
Epoch 14, Loss: 0.03627829998731613, Accuracy: 98.91503143310547, Test Loss: 1.637528419494629, Test Accuracy: 86.4000015258789
Epoch 15, Loss: 0.04092969745397568, Accuracy: 98.84503173828125, Test Loss: 1.7007077932357788, Test Accuracy: 86.54999542236328
Epoch 16, Loss: 0.03814692795276642, Accuracy: 98.85002899169922, Test Loss: 1.5850194692611694, Test Accuracy: 85.22999572753906
Epoch 17, Loss: 0.03316102921962738, Accuracy: 98.97752380371094, Test Loss: 1.6998156309127808, Test Accuracy: 86.13999938964844
Epoch 18, Loss: 0.035765886306762695, Accuracy: 98.8850326538086, Test Loss: 1.7951807975769043, Test Accuracy: 85.68999481201172
Epoch 19, Loss: 0.030668308958411217, Accuracy: 99.0875244140625, Test Loss: 1.9048504829406738, Test Accuracy: 85.86000061035156
Epoch 20, Loss: 0.028253726661205292, Accuracy: 99.08252716064453, Test Loss: 1.9930514097213745, Test Accuracy: 86.3499984741211
```

Results – Anomaly Detection

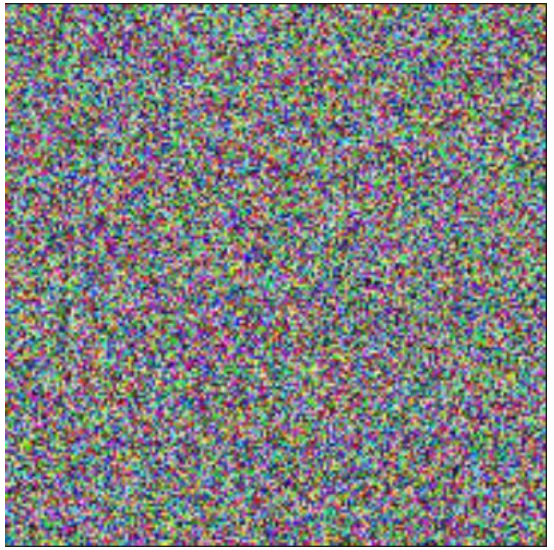
Anomaly detection worked better for classes with more examples. Similar classes were confused for each other. Classes that were trained on were sometimes mislabeled as anomalous. We successfully could detect complete anomalies.

Num, Actual Class, Prediction

```
37,Building,Building
38,Building,Building
39,Building,Building
40,Building,Building
41,Building,Building
42,Building,anomaly
43,Building,Building
44,Building,anomaly
45,Building,Building
46,Building,Building
```

```
334,Dump Truck,Dump Truck
335,Dump Truck,Cargo Truck
336,Dump Truck,Dump Truck
337,Dump Truck,Cargo Truck
338,Dump Truck,anomaly
339,Dump Truck,Dump Truck
340,Dump Truck,anomaly
341,Dump Truck,anomaly
342,Dump Truck,Truck with Flatbed
343,Dump Truck,Dump Truck
344,Dump Truck,anomaly
```

Complete Anomaly



Further Optimizations

Things we tried:

- Deeper CNN with more layers, but training took a long time so limited results were available
- Playing with dropout percentage, number of epochs, min/max thresholds to compare new latent vectors to for anomaly predictions

Future improvements:

- Better handle class imbalance
- Issues with “Killed” when using very large dataset, so had to narrow
- Resnet 18/50