Flooding Damage Detection from Post-Hurricane Satellite Imagery Based on Convolutional Neural Networks

---

Binary Classification Problem

Bixuan Huang & Weining Hu
December 9, 2019
ML2 Final Group Project

Introduction

Source Data:
- Satellite images that covers the Greater Houston area before and after the Hurricane Harvey in 2017 (Kaggle Dataset).

Problem Statement:
- Binary classification problem that classify images into two classes: image with flooding damage or no damage.

Motivation:
- To improve the efficiency of building disaster assessment.

Project Design:
- *PyTorch* framework
- Compared a custom convolutional model to two pre-trained models (VGG-16 & Resnet50)

Dataset

Size: 128 X 128 X 3

Data Distribution:

LABELS / FOLDERS	DAMAGE	NO DAMAGE
TRAINING SET	5,000	5,000
VALIDATION SET	1,000	1,000
TEST SET	8,000	1,000

Example Images:

Damage




No Damage




Baseline

Load data: ImageFolder

Image Augmentation: transforms.compose()

Last Fully Connected Layer: two output classes

Criterion: CrossEntropyLoss()

Optimization Algorithm: Mini-batch gradient descent

```python
traindataFromFolders = datasets.ImageFolder(root='train_another/', transform=train_tfroms)
train_loader = DataLoader(traindataFromFolders, batch_size=50, shuffle=True)
x_train, y_train = iter(train_loader).next()

model = models.vgg16(pretrained=True)
for param in model.parameters():
    param.requires_grad = False
model.classifier[6] = nn.Sequential(
    nn.Linear(model.classifier[6].in_features, 256),
    nn.ReLU(), nn.Dropout(0.2), nn.Linear(256, 2)) #nn.Sigmoid()
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=LR)

BATCH_SIZE = 3
N_EPOCHS = 10
LR = 0.00002

for epoch in range(N_EPOCHS):
    train_loss = 0
    model.train()
    for batch in range(len(x_train)//BATCH_SIZE + 1):
        idx = slice(batch * BATCH_SIZE, (batch+1)*BATCH_SIZE)
        optimizer.zero_grad()
        output = model(x_train[idx])
        loss = criterion(output, y_train[idx])
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
    model.eval()
    with torch.no_grad():
        y_val_pred = model(x_val)
        loss = criterion(y_val_pred, y_val)
        val_loss = loss.item()
```

Pre-trained Model & Fine Tuning Hyperparameters

Pretrained Models:

- VGG-16
- Resnet50

Parameters:

- Batch Size: 50
- Mini-Batch Size: 10 - 20
- Learning Rate: 0.00002 (VGG - 16); 0.001 (Resnet50)
- Optimizer: Adam
- Epochs: 30

```python
#Load pre-trained model
def get_pretrained_model(model_name):
    if model_name == 'vgg16':
        model = models.vgg16(pretrained=True)

        # Freeze early layers
        for param in model.parameters():
            param.requires_grad = False
        n_inputs = model.classifier[6].in_features
        n_classes = 2

        # Add on classifier
        model.classifier[6] = nn.Sequential(
            nn.Linear(n_inputs, 256), nn.ReLU(), nn.Dropout(0.2),
            nn.Linear(256, n_classes)) #, nn.Sigmoid()

    elif model_name == 'resnet50':
        model = models.resnet50(pretrained=True)

        for param in model.parameters():
            param.requires_grad = False

        n_inputs = model.fc.in_features
        n_classes = 2
        model.fc = nn.Sequential(
            nn.Linear(n_inputs, 256), nn.ReLU(), nn.Dropout(0.2),
            nn.Linear(256, n_classes)) #, nn.Sigmoid()

    # Move to GPU
    MODEL = model.to(device)
```

Custom Model & Fine Tuning Hyperparameters

Filter size: (3,3) in first Convolutional Layer and (6,6) in second Convolutional Layer

Mini-batch Size: 4

Learning Rate: 0.01

Epoch: 100

Drop Out: 0.5

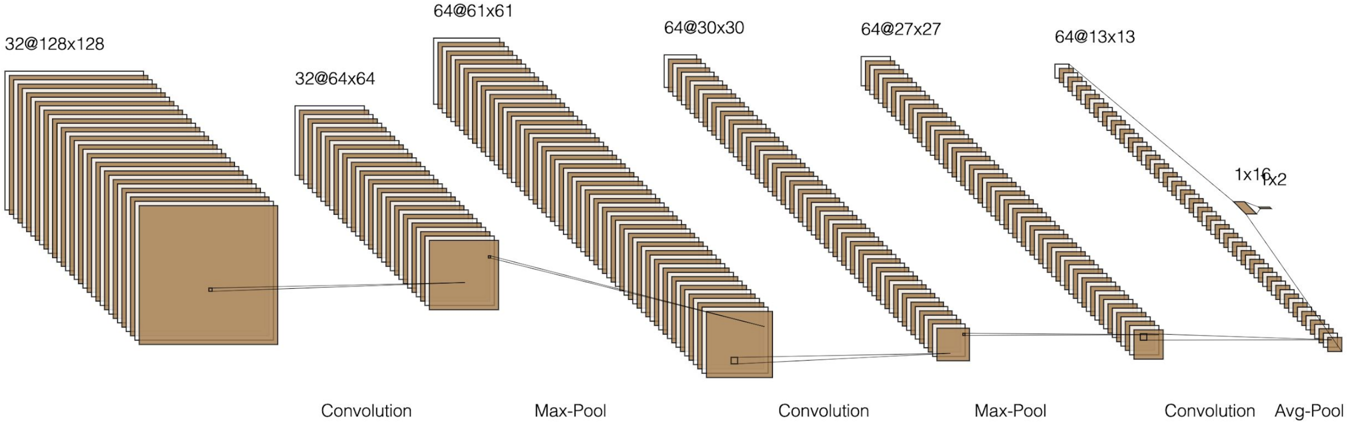
```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size = 3, stride = 1, padding = 1)
        self.convnorm1 = nn.BatchNorm2d(32)
        self.pool1 = nn.MaxPool2d(2, 2)

        self.conv2 = nn.Conv2d(32, 64, kernel_size=6, stride=1, padding=1)
        self.convnorm2 = nn.BatchNorm2d(64)
        self.pool2 = nn.MaxPool2d((2, 2))

        self.conv3 = nn.Conv2d(64, 64, kernel_size = 6, stride = 1, padding = 1)
        self.convnorm3 = nn.BatchNorm2d(64)
        self.pool3 = nn.AvgPool2d((2, 2))

        self.dropout = nn.Dropout(DROPOUT)
        self.linear1 = nn.Linear(64 * 13 * 13, 16)
        self.linear1_bn = nn.BatchNorm1d(16)
        self.linear2 = nn.Linear(16, 2)
        self.linear2_bn = nn.BatchNorm1d(2)
        self.sigmoid = torch.sigmoid
        self.relu = torch.relu

    def forward(self, x):
        x = self.pool1(self.convnorm1(self.relu(self.conv1(x))))
        x = self.pool2(self.convnorm2(self.relu(self.conv2(x))))
        x = self.pool3(self.convnorm3(self.relu(self.conv3(x))))
        # print(x.shape)
        x = self.dropout(self.linear1_bn(self.relu(self.linear1(x.view(-1, 64 * 13 * 13)))))
        x = self.dropout(self.linear2_bn(self.relu(self.linear2(x))))
        x = self.sigmoid(x)
        return x
```



Evaluation

Performance matrix includes: accuracy, confusion matrix, precision and recall

MODELS	ACCURACY RATE	PRECISION	RECALL
VGG-16	0.89	0.90	0.97
RESNET50	0.88	0.91	0.96
CUSTOM MODEL	0.90	0.95	0.94

VGG-16

Resnet50

Custom Model

Conclusion

Project: built three models (custom model, pre-trained VGG-16, pre-trained Resnet)

Results:

- CNNs
- Custom model yielded the highest accuracy

Future Work:

- Increase source image by integrating other region's satellite images
- Apply the model to detect other types of damage, including road damage caused by hurricane..
- Leverage model ensemble techniques

Thank you !