Violence Detection in Video Classification

Utilizing the RWF-2000 Video Database

Hongfei Niu Individual Report

12/07/2020
Machine Learning II
Amir Jafari

1. Introduction

With the increase in surveillance cameras throughout the world, a large amount of video data can now be collected in efforts to classify human action. Like many other sources of data there has become an excess of observations and a lack of human capability to observe them. For this project we will attempt to utilize the RWF-2000 video database which includes 2000 videos split evenly between violent and nonviolent actions to create a binary classification network to detect violence. Hopefully, this classification system will be quick and accurate enough to alert authorities in time to mitigate the damages caused by the violence captured.

Preprocessing was performed separately for each model, however, there were a few steps taken to establish a directory of videos converted to NumPy in tensor format with shape = [nb_frames, img_height, img_width, 5]. Preprocessing began with the cv2 package deriving 3 RGB flows and 2 Optical Flows. Optical flow is defined as the apparent motion of individual pixels on the image plane. In other words, the motion of images and objects between frames of a video. This will assist us in deriving human action between frames.

2. Individual Work

We are unable to utilize all 800 and 200 train/test files due to memory issues. Based on the shared data extraction, 200 train videos and 60 test videos are chosen for my models, which are all resized into 100*100 resolution.

2.1 EfficientNet-B0

Convolutional neural networks are usually developed at a fixed computational resource, and then increase the dimensions when more resources are added to reach higher accuracy. For example, VGG can extend from VGG-16 to VGG-19 by increasing the number of layers. Although this method does improve accuracy, they usually require long manual tuning and still produce secondary performance.

Compared with traditional CNN, which arbitrarily increases the network dimensions, the EfficientNets use a series of fixed scaling factors to uniformly adjust the network dimensions, which reach 10 times efficiency and higher accuracy than SOTA.

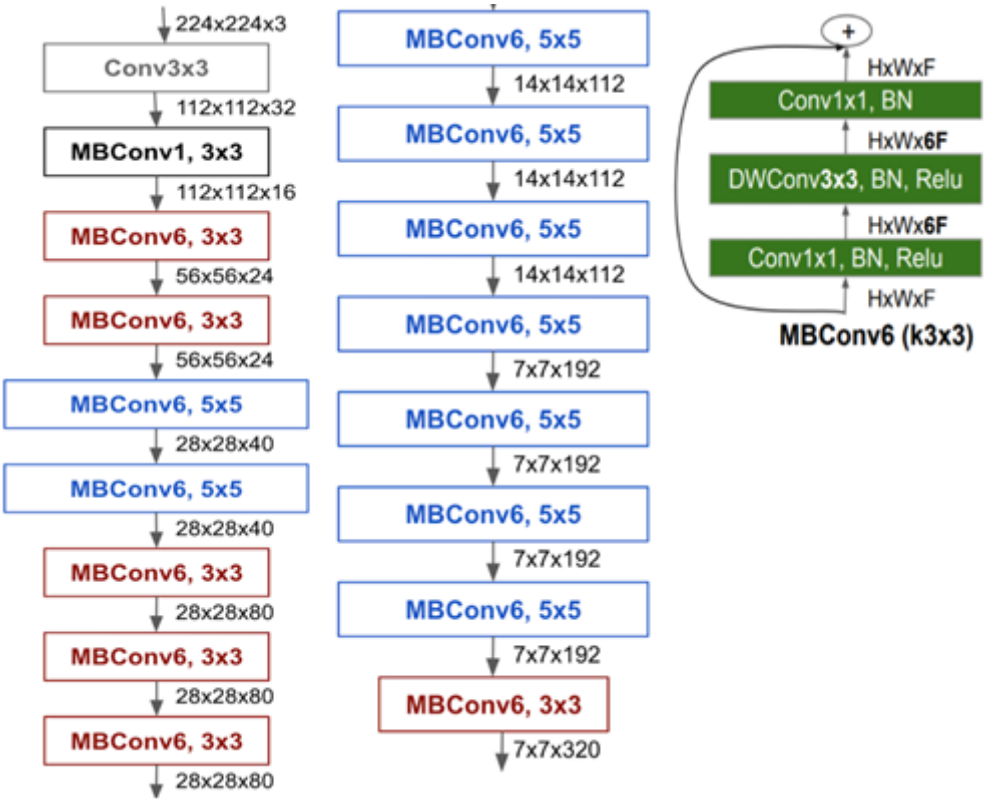


Figure 1 EfficientNet-B0 Architecture

2.2 Training Process

Apart from the resizing, the data are also transformed and reshaped. According to the two-streamed algorithm, the paper adds 2 optical flow channels in the models, trying to contain the action information. For the EfficientNet-B0 model, the videos are cropped again with 28 frames each clip with 5 channels (3 RGB and 2 Optical Flows). And the final input is in shape of (814, 5, 28, 100*100).

```
def get_rgb(input_x):
    rgb = input_x[...,:3]
    return rgb
DATA_DIR = os.getcwd()+"/npy_100/Train200"
DATA_DIR_Test = os.getcwd()+"/npy_100/Test60"
def stack(path,label_names, frame=20):
    x,y=[],[]
    for f1 in label_names:
        path_loop = os.path.join(path, f1)
        for f2 in [f2 for f2 in os.listdir(path_loop) if f2[-4:] == ".npy"]:
            a = np.load(path_loop + "/" + f2)
            # a = get_rgb(a)
            num1 = len(a)//frame
            num2 = ceil(len(a)/frame)
            a_recrop=[]
            if (num1==num2):
                for i in range(num1):
                    a_clip = a[i*frame:(i+1)*frame]
                    a_recrop.append(a_clip)
            if (num1 !=num2):
                for i in range(num1):
                    a_clip = a[i * frame:(i + 1) * frame]
                    a_recrop.append(a_clip)
                a_recrop.append(a[len(a)-frame:])
            a = np.asarray(a_recrop)
            a = a.transpose(0, 4, 1, 2, 3)/255
            a = a.reshape(len(a),5,20,-1)
            x.append(a)
            temp1 = np.ones(len(a))
            temp2 = np.zeros(len(a))
            if f1 == "Violence":
                y_temp = np.dstack((temp1, temp2)).reshape(-1, 2)
            elif f1 == "NoViolence":
                y_temp = np.dstack((temp2, temp1)).reshape(-1, 2)
            y.append(y_temp)
    y = np.vstack(y)
    x = np.vstack(x)
    return x, y

label_names = ["Violence", "NoViolence"]
x_train, x_test, y_train, y_test=[],[],[],[]
x_train, y_train = stack(DATA_DIR, label_names, frame=28)
x_test, y_test = stack(DATA_DIR_Test, label_names, frame=28)
```

The EfficientNet is applied with the pretrained weight “efficientnet-b0” and is added with a con2d layer to change the input channels from 3 to 5. The network has an image input size of (224,224), but our input of (100,100) works well too. The Full-Connection part of our network model contains: an average pooling with shape (2,2), a dense linear layer of 512

neurons with batch normalization and Relu process, a dense function of 256 neurons with batch normalization, Relu process and a 0.2 dropout, finally a dense layer with 2 neurons and sigmoid activation for binary classification.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=5,
                               out_channels=3,
                               kernel_size=3,
                               stride=1,
                               padding=1)
        self.efficientNet = EfficientNet.from_pretrained('efficientnet-b0')
        num_ftrs = self.efficientNet._fc.out_features
        self.linear0_bn = nn.BatchNorm1d(num_ftrs)
        self.linear1 = nn.Linear(num_ftrs, 512)
        self.linear1_bn = nn.BatchNorm1d(512)
        self.drop = nn.Dropout(DROPOUT)
        self.linear2 = nn.Linear(512, 256)
        self.linear2_bn = nn.BatchNorm1d(256)
        self.linear3 = nn.Linear(256, 2)
        self.act = torch.relu


    def forward(self, x):
        x = self.conv1(x)
        x = self.act(self.linear0_bn(self.efficientNet(x)))
        x = self.act(self.linear1_bn(self.linear1(x)))
        x = self.drop(self.act(self.linear2_bn(self.linear2(x))))
        return self.linear3(x)
```

The network is compiled with Binary Cross entropy, Stochastic Gradient Descent, and accuracy metrics. Batch size is 100 and the model does not set a fixed epoch number. The best model is saved when the minimum loss value refreshes.

For the learning rate, the initial value is set as 1e10^-3, and the weight decay is LR. Three decreasing schedulers are also set to adjust the LR during the training, avoiding falling into local optimum. The model chooses SGD optimizer, pursuing both direction accuracy and speed.

```
optimizer = torch.optim.SGD(model.parameters(),lr=LR, weight_decay=0.5*LR)
scheduler_adam_p1= torch.optim.lr_scheduler.StepLR(optimizer,step_size=5, gamma = 0.1)
scheduler_adam_p2= torch.optim.lr_scheduler.StepLR(optimizer,step_size=8, gamma = 0.5)
scheduler_adam_p3= torch.optim.lr_scheduler.StepLR(optimizer,step_size=5, gamma = 0.8)
```

2.3 Results & Evaluation

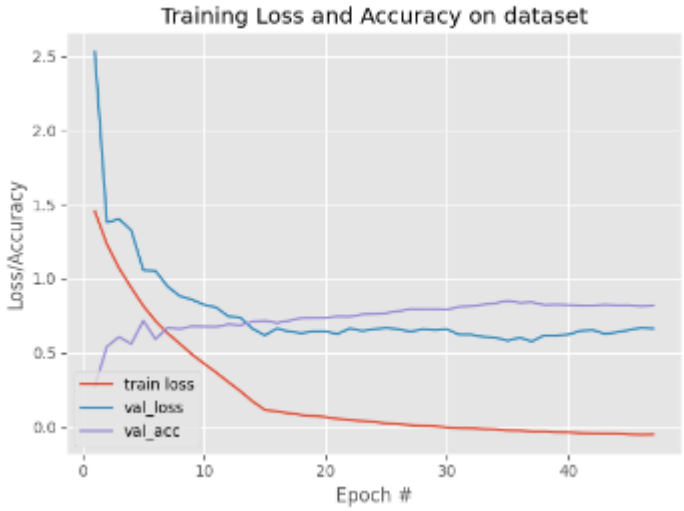
Since this is a balanced dataset accuracy will be our best metric to use, the Accuracy, BCE Loss and Confusion Matrix will be applied to test the result.

Accuracy: 0.791659； BCE Loss: 0.7271

Confusion Matrix:

n = 229	Positive	Negative
Positive	105	18
Negative	32	74

Loss/Accuracy Graph:



Based on the result, the model classifies the Violence video relatively better than the Nonviolence videos, which may be caused by the moving noises. The L2 regularization and LR scheduler make effort because the loss value decreases stable and well. Overall, the model has a relatively good ability of classification, and we will drop the optical flow data in the next model and make a contrast.

2.4 VGG-16

Based on Traditional AlexNet, for the structure of VGG, the three 3x3 convolution kernels are applied instead of 7x7 convolution kernels, and two 3x3 convolution kernels are applied instead of 5*5 convolution kernels. Under the same condition of the perceptual field,

the VGG model increases the depth of the network structure, and the effect of the neural network could be improved to a certain extent.

In this paper, the VGG-16 is applied with the pretrained weight, to increase the training speed. The network has an image input size of (224,224), but our input of (100,100) works well too. The Full-Connection part of our network model contains: an average pooling with shape (2,2), a dense linear layer of 512 neurons with batch normalization and Relu process, a dense function of 256 neurons with batch normalization, Relu process and a 0.2 dropout, finally a dense layer with 2 neurons and sigmoid activation for binary classification.

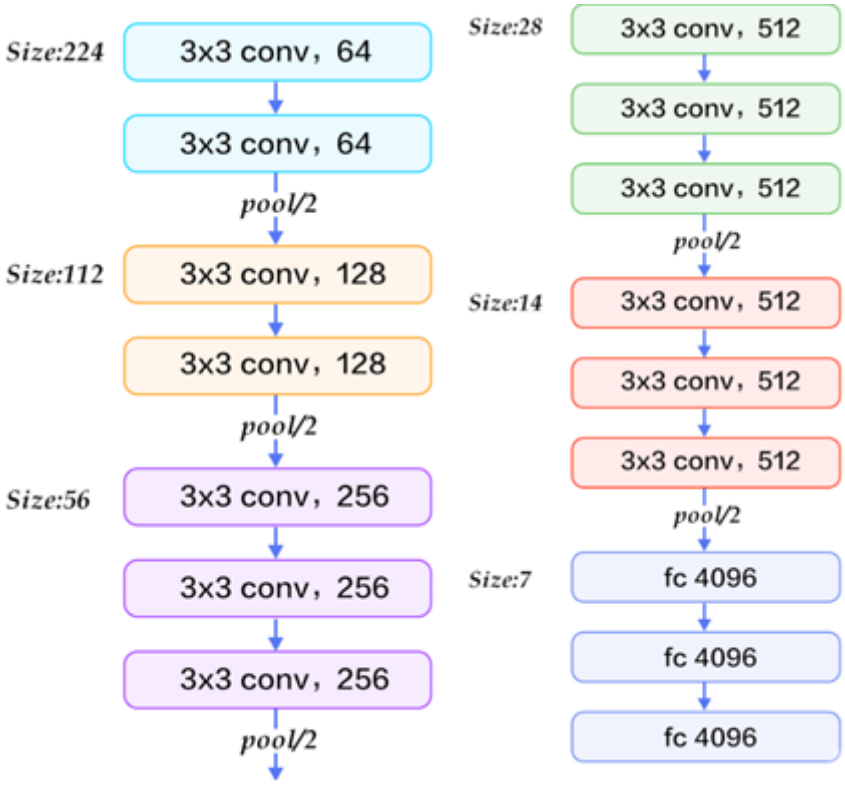


Figure 2 VGG-16 Architecture

2.5 Compile and Fit

The VGG-16 Network starts with 200 training examples and 60 test examples. We are unable to utilize all 800 and 200 train/test files due to memory issues. The network uses 3 flows (3 RGB). Because, it will make a contrast experiment with the EfficientNet-B0 model, and the frame channel already include some action sequence information. For better performance, the videos are cropped again with 28 frames each clip. And the final input is in shape of (814, 3, 28, 100*100).

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.vgg = torchvision.models.vgg16(pretrained=True)
        num_ftrs = self.vgg.classifier[6].out_features
        self.linear0_bn = nn.BatchNorm1d(num_ftrs)
        self.linear1 = nn.Linear(num_ftrs, 512)
        self.linear1_bn = nn.BatchNorm1d(512)
        self.drop = nn.Dropout(DROPOUT)
        self.linear2 = nn.Linear(512, 256)
        self.linear2_bn = nn.BatchNorm1d(256)
        self.linear3 = nn.Linear(256, 2)
        self.act = torch.relu


    def forward(self, x):
        x = self.act(self.linear0_bn(self.vgg(x)))
        x = self.act(self.linear1_bn(self.linear1(x)))
        x = self.drop(self.act(self.linear2_bn(self.linear2(x))))
        return self.linear3(x)
```

The network is compiled with Binary Cross entropy, Stochastic Gradient Descent, and accuracy metrics. Batch size is 100 and the model does not set a fixed epoch number. The best model is saved when the minimum loss value refreshes.

For the learning rate, the initial value is set as 1e10^-3, and the weight decay is LR. The decreasing schedulers are also set to adjust the result, avoiding falling into local optimum. The model chooses SGD optimizer, pursuing both direction accuracy and speed.

2.6 Results & Evaluation

Since this is a balanced dataset accuracy will be our best metric to use, the Accuracy, BCE Loss and Confusion Matrix will be applied to test the result.

Accuracy: 0.8340611 BCE Loss: 0.6896

Confusion Matrix:

n = 229	Positive	Negative
Positive	112	11
Negative	22	79

Loss/Accuracy Graph:

Based on the result, the model classifies the Violence video still slightly better than the Nonviolence videos, which may be also caused by the moving noises. And for the training process, it seems that the schedulers do not make good effort in the beginning, because the Loss line fluctuates much. Maybe, the initial value of the LR could be smaller and we will get a better decreasing trend. Overall, the VGG-16 model has relatively better results than the EfficientNet-B0 model. It means that the optical flow data have some noises, such as the moving of the background or camera, which will negatively influence the training. However, if we could classify the background and target people to calculate useful optical flow data, then make a classification, I guess the result will improve a lot.

3. **Summary and Improvement**

For this project, the VGG-16 model has relatively better results than the EfficientNet-B0 model. It means that the optical flow data have some noises, such as the moving of the background or camera, which will negatively influence the training. However, if we could

classify the background and target people to calculate useful optical flow data, then make a classification, I guess the result will improve a lot. Hence, I search some articles and find the iDT algorithm. I will try it after this project.

In this project, I learn a lot about video processing, such as the information of the optical flow. Also, after practicing, I could change the structure and the form of the data flexibly in order to reach different needs/

Percentage: (427-113)/(427+203) = 49.84127%

4. Reference

https://github.com/mchengny/RWF2000-Video-Database-for-Violence-Detection

https://hellozhaozheng.github.io/z_post/%E8%AE%A1%E7%AE%97%E6%9C%BA%E8%A7%86%E8%A7%89-EfficientNet-ICML2019/

https://github.com/cydonia999/VGGFace2-pytorch

https://zhuanlan.zhihu.com/p/91854877