ML2 Final Project

Violence & Non-Violence Video Detection through neural network

Changhao Ying, Spencer Staub, Hongfei Niu

Introduction

- Dataset
  - RWF-2000 Violence detection database
  - 2000 videos captured from surveillance cameras
  - classified into violent or non-violent videos
- Objective
  - Create a network with binary classification
  - Utilize various MLP, CNN and RNN networks
- Challenges
  - Extremely large data source with limited computing power
  - Preprocessing and deploying networks on video data (multiple frames)

Dataset introduction

- Raw Video Clips from Surveillance videos on Youtube
  - 5 Seconds long
  - 30 Frames Per Second
  - *.avi format
- Downloaded as an ~11GB zip file resulting in ~120GB in video files
  - 2000 Total Videos
  - Separated into Violence/Nonviolence
  - Seperated Into an 80/20 Train/Test Split
- Ming Cheng, Kunjing Cai, and Ming Li. "RWF-2000: An Open Large Scale Video Database for Violence Detection." arXiv preprint arXiv:1911.05913 (2019).

Video Description

- Poor quality:
  - Dark Environments
  - Low Resolution
  - Low Frames Per Second
  - Quick Moving Objects
  - Non-Color Video
  - Blur
  - Image Obstruction
- Format
  - Large Crowds
  - One-on-One Interaction
  - Conflict partially in frame
  - Conflict Far Away

Preprocess

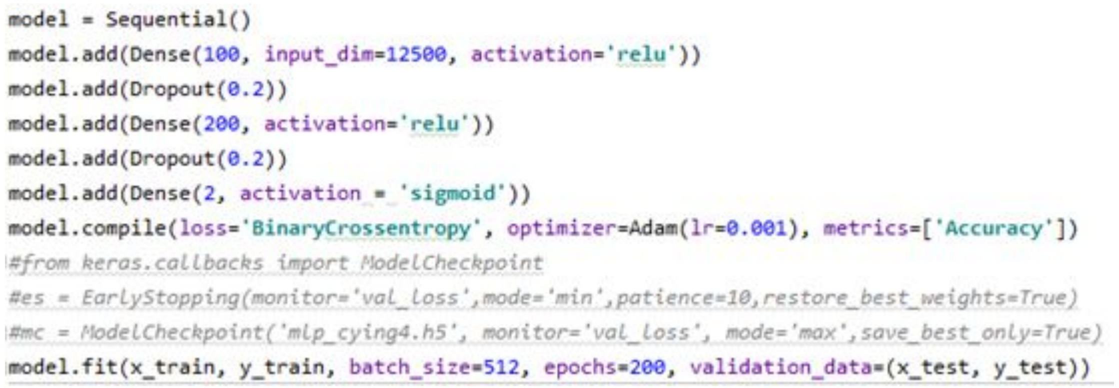
- Transfer all the videos into npy and save them into ubuntu directory
- 149 frames per video
- Use cv2 resize to (224,224), (100,100) or (64,64) depending on the memory requirement of each network
- Create 3RGB and 2 OPT for each frame
- Normalize data using Mean and Standard Deviation
- Downsample 149 frames to 28 frames depending on the memory requirement of each network
- Reshape to work with various network input shapes
- Create categorical labels based off of video file location (1,0)

Models & Analysis

- MLP
- CNN
  - Conv1D
  - Resnet50 (pretrained)
  - EfficientNet-B0 (pretrained)
  - VGG-16 (pretrained
- RNN
  - Time Distributed Conv2D with LSTM

MLP Model

- Setup
  - Input Shape: (Batch*149*50*50*5)
  - Train/Test: 64/16 videos or 9536/2348 Frames
  - Scale Train/Test
- Network



```
model = Sequential()
model.add(Dense(100, input_dim=12500, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(200, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(2, activation = 'sigmoid'))
model.compile(loss='BinaryCrossentropy', optimizer=Adam(lr=0.001), metrics=['Accuracy'])
#from keras.callbacks import ModelCheckpoint
#es = EarlyStopping(monitor='val_loss',mode='min',patience=10,restore_best_weights=True)
#mc = ModelCheckpoint('mlp_cying4.h5', monitor='val_loss', mode='max',save_best_only=True)
model.fit(x_train, y_train, batch_size=512, epochs=200, validation_data=(x_test, y_test))
```

MLP Model Results

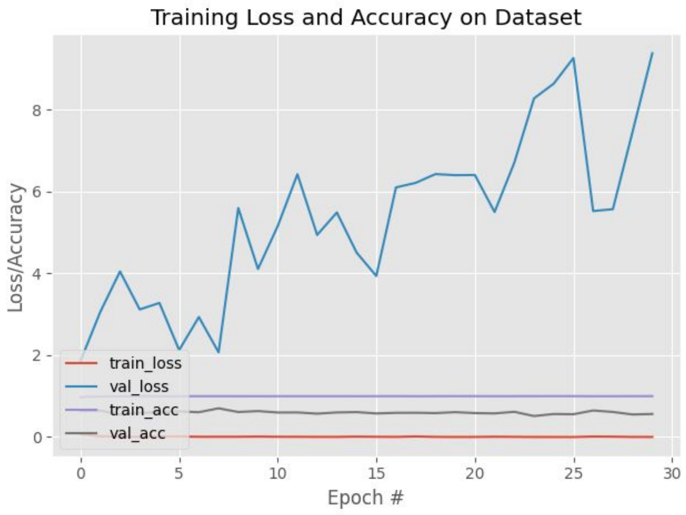
- Validation Accuracy: 56.33%
- F1 Score: 0.56
- Cohen Kappa: 0.12
- Confusion Matrix:

Actual

Predicted

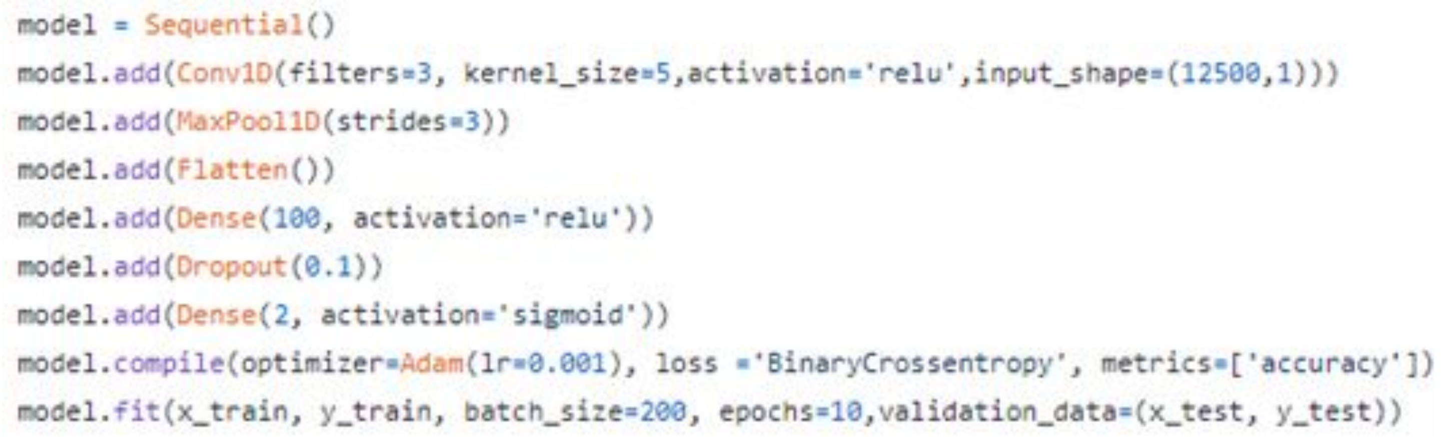
n = 2384	Positive	Negative
Positive	554	638
Negative	403	789

Loss/Accuracy Graph:



Conv1D Model

- Setup
  - Input Shape: (Batch*149*50*50*5*1)
  - Train/Test: 64/16 videos or 9536/2348 Frames
  - Scale Train/Test
- Network



```
model = Sequential()
model.add(Conv1D(filters=3, kernel_size=5,activation='relu',input_shape=(12500,1)))
model.add(MaxPool1D(strides=3))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(2, activation='sigmoid'))
model.compile(optimizer=Adam(lr=0.001), loss ='BinaryCrossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=200, epochs=10,validation_data=(x_test, y_test))
```

Conv1D Model Results

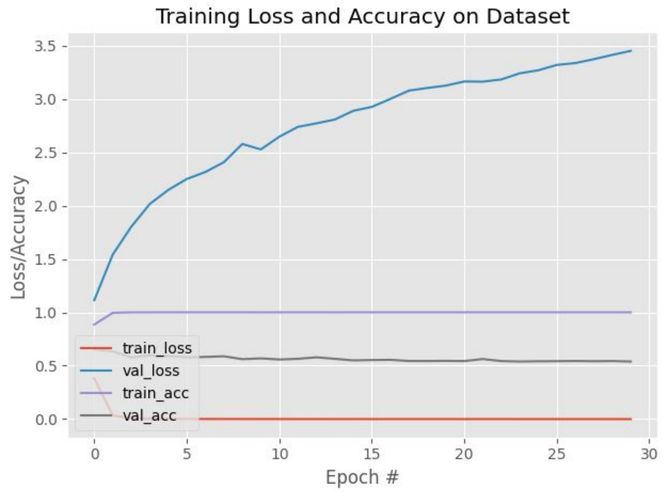
- Validation Accuracy: 53.9 %
- F1 Score: 0.52
- Cohen Kappa: 0.078
- Confusion Matrix:

Loss Graph:

Actual

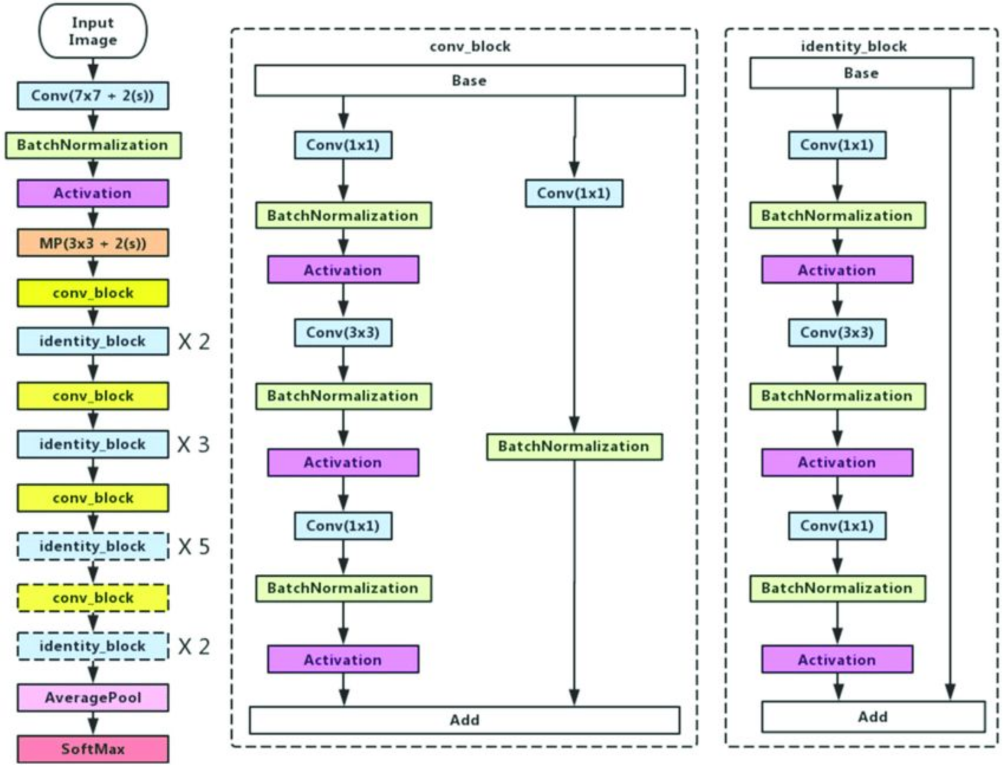
Predicted

n = 2384	Positive	Negative
Positive	464	728
Negative	371	821



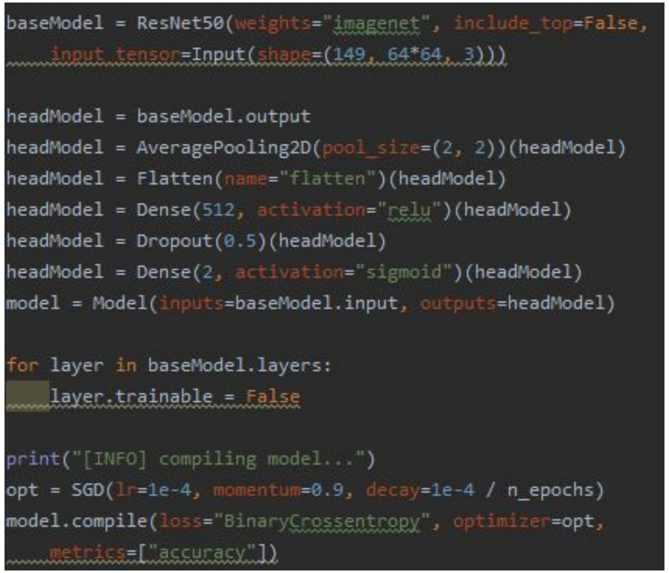
ResNet50 Model Description

- Convolutional Neural Network
- 50 Layers
- Pretrained on Imagenet Dataset
- Input shape of (224,224), but our input of (64,64) works well too



ResNet50 Model

- Setup
  - Input Shape: (149,64*64,3)
  - Train/Test: 256/64 videos
- Network

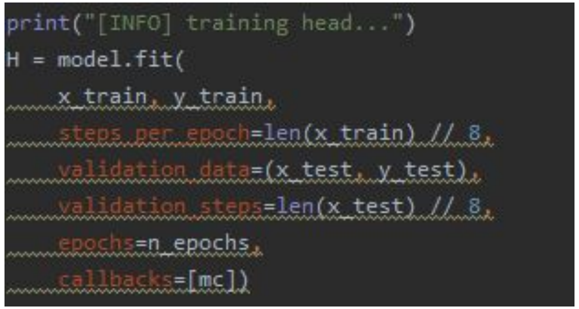


```
baseModel = ResNet50(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(149, 64*64, 3)))

headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(2, 2))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(512, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="sigmoid")(headModel)
model = Model(inputs=baseModel.input, outputs=headModel)

for layer in baseModel.layers:
    layer.trainable = False

print("[INFO] compiling model...")
opt = SGD(lr=1e-4, momentum=0.9, decay=1e-4 / n_epochs)
model.compile(loss="BinaryCrossentropy", optimizer=opt,
    metrics=["accuracy"])
```



```
print("[INFO] training head...")
H = model.fit(
    x_train, y_train,
    steps_per_epoch=len(x_train) // 8,
    validation_data=(x_test, y_test),
    validation_steps=len(x_test) // 8,
    epochs=n_epochs,
    callbacks=[mc])
```

ResNet50 Model Results

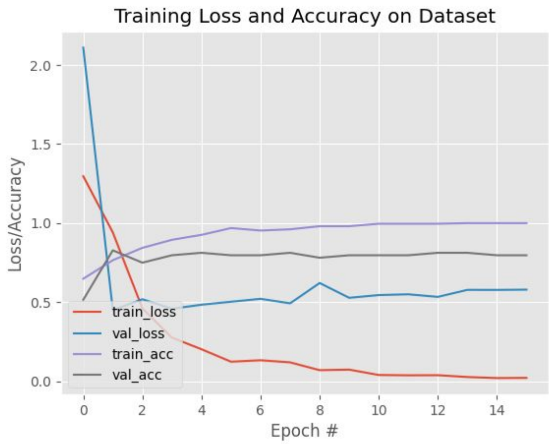
- Validation Accuracy: **84.375 %**
- F1 Score: 0.84
- Cohen Kappa: 0.6875
- Confusion Matrix:

Actual

Predicted

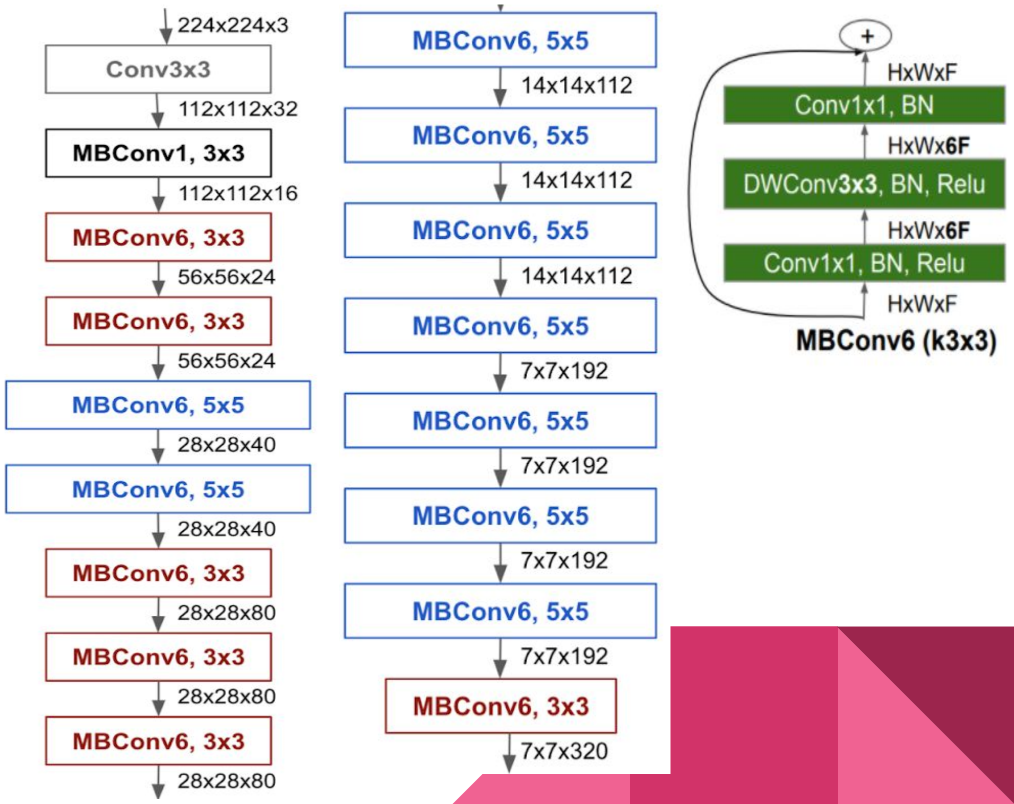
n = 64	Positive	Negative
Positive	26	6
Negative	7	25

Loss Graph:



EfficientNet-B0 Description

- Convolutional Neural Network
- Pretrained on Imagenet Dataset
- Input shape of (224,224), but our input of (100,100) works well too



EfficientNet-B0

1. Grid search

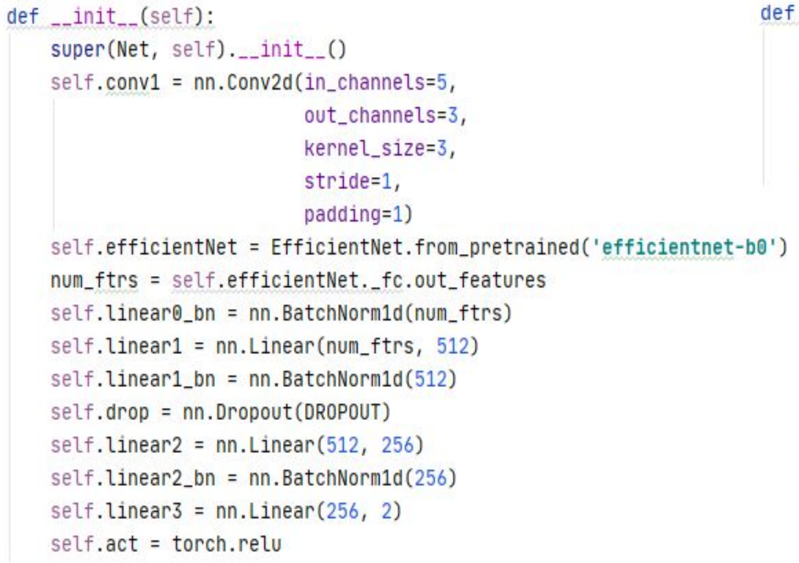
   In order to find the relationship between the different scaling dimensions of the baseline network, such as depth, width and resolutions, the appropriate scaling factor for each dimension are searched to expand the model.

2. New baseline model

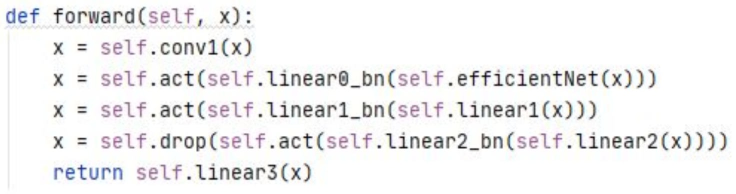
   In order to further improve performance, the author use the AutoML MNAS framework to perform neural structure search. The final architecture uses mobile reverse bottleneck convolution (MBConv).

EfficientNet-B0 Model

- Setup
  - Input Shape: (28,100*100,5)
  - Train/Test: 200/60 videos
- Network



```python
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
```



```python
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)
```

EfficientNet-B0 Model Results

- Validation Accuracy: 79.165 %

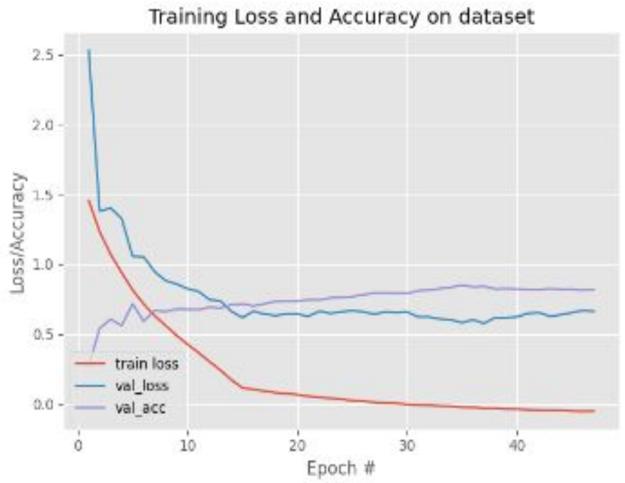
- Confusion Matrix:

Actual

Predicted

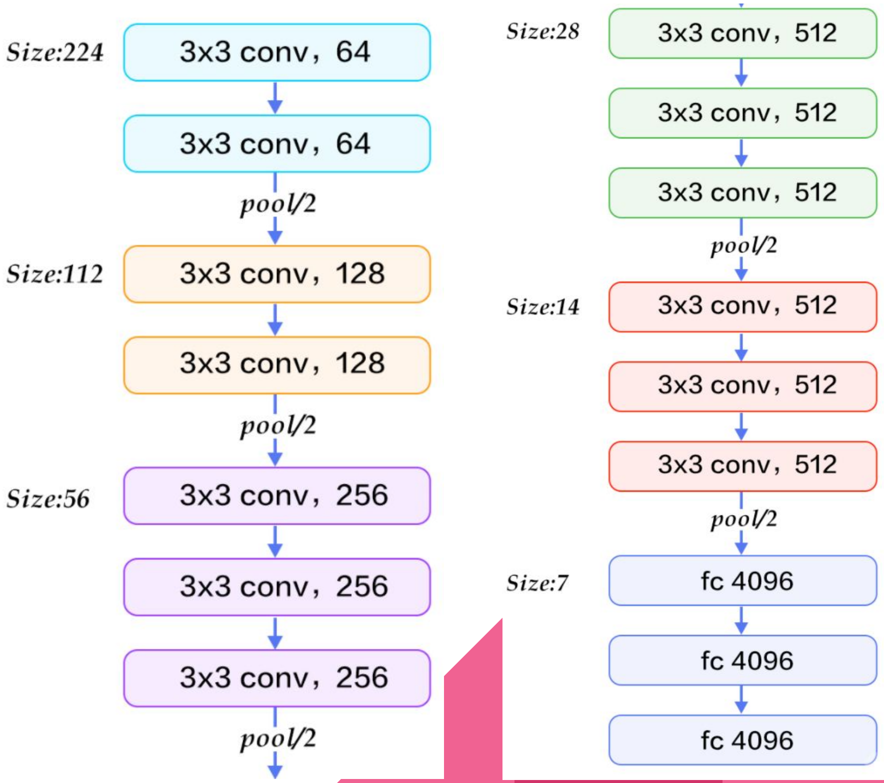
n = 229	Positive	Negative
Positive	105	18
Negative	32	74

Loss Graph:



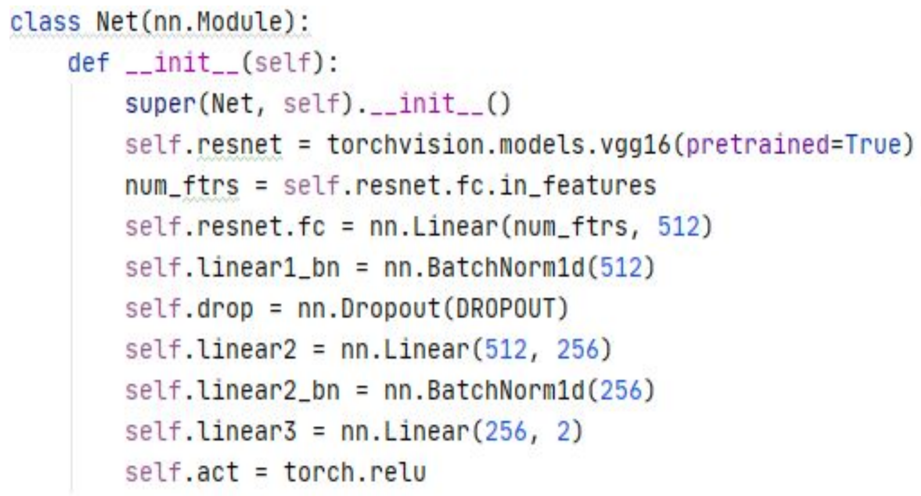
VGG-16 Description

- Convolutional Neural Network
- Pretrained on Imagenet Dataset
- Input shape of (224,224), but our input of (100,100) works well too

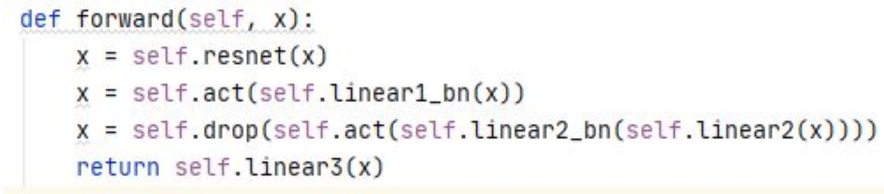


VGG-16 Model

- Setup
  - Input Shape: (28,100*100,5)
  - Train/Test: 200/60 videos
- Network



```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.resnet = torchvision.models.vgg16(pretrained=True)
        num_ftrs = self.resnet.fc.in_features
        self.resnet.fc = 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
```



```python
def forward(self, x):
    x = self.resnet(x)
    x = self.act(self.linear1_bn(x))
    x = self.drop(self.act(self.linear2_bn(self.linear2(x))))
    return self.linear3(x)
```

VGG-16 Model Results

- Validation Accuracy: **83.406 %**

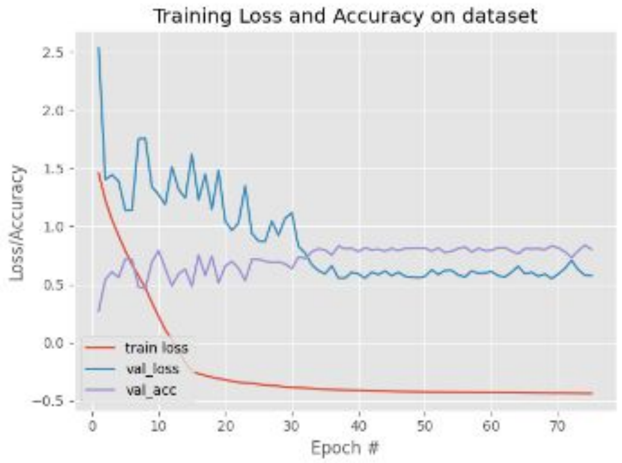
- Confusion Matrix:

Actual

Predicted

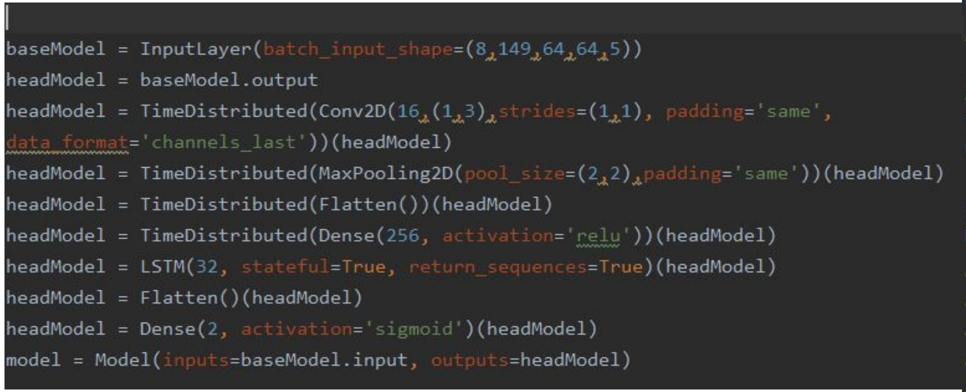
n = 229	Positive	Negative
Positive	112	11
Negative	22	79

Loss Graph:

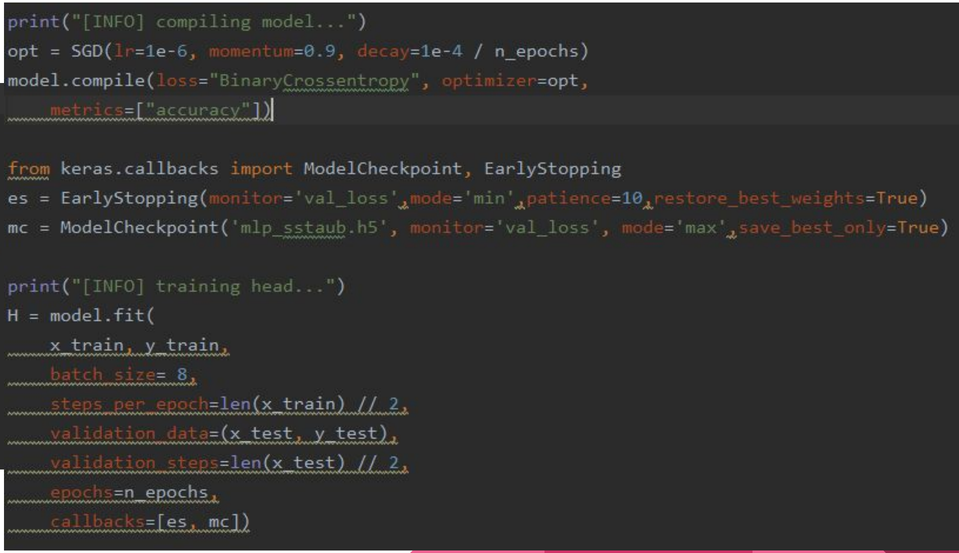


Time Distributed Conv2D with LSTM Model

- Setup
  - Input Shape: (8,149,64,64,5)
  - Train/Test: 256/64 videos
- Network



```
baseModel = InputLayer(batch_input_shape=(8,149,64,64,5))
headModel = baseModel.output
headModel = TimeDistributed(Conv2D(16,(1,3),strides=(1,1), padding='same',
data_format='channels_last'))(headModel)
headModel = TimeDistributed(MaxPooling2D(pool_size=(2,2),padding='same'))(headModel)
headModel = TimeDistributed(Flatten())(headModel)
headModel = TimeDistributed(Dense(256, activation='relu'))(headModel)
headModel = LSTM(32, stateful=True, return_sequences=True)(headModel)
headModel = Flatten()(headModel)
headModel = Dense(2, activation='sigmoid')(headModel)
model = Model(inputs=baseModel.input, outputs=headModel)
```



```
print("[INFO] compiling model...")
opt = SGD(lr=1e-6, momentum=0.9, decay=1e-4 / n_epochs)
model.compile(loss="BinaryCrossentropy", optimizer=opt,
    metrics=["accuracy"])

from keras.callbacks import ModelCheckpoint, EarlyStopping
es = EarlyStopping(monitor='val_loss',mode='min',patience=10,restore_best_weights=True)
mc = ModelCheckpoint('mlp_sstaub.h5', monitor='val_loss', mode='max',save_best_only=True)

print("[INFO] training head...")
H = model.fit(
    x_train, y_train,
    batch_size= 8,
    steps_per_epoch=len(x_train) // 2,
    validation_data=(x_test, y_test),
    validation_steps=len(x_test) // 2,
    epochs=n_epochs,
    callbacks=[es, mc])
```

Time Distributed Conv2D with LSTM Model Results

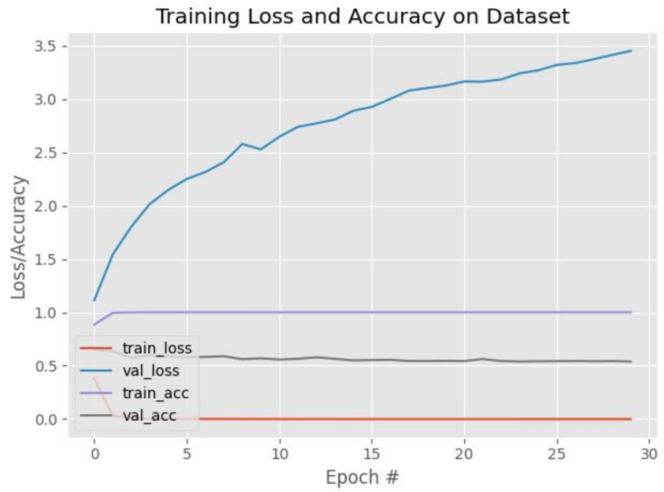
- Validation Accuracy: 81.5 %
- F1 Score: 0.74
- Cohen Kappa: 0.5
- Confusion Matrix:

Actual

Predicted

n = 16	Positive	Negative
Positive	7	1
Negative	3	5

Loss Graph:



Summary

- Conclusion
- Limitation
- Improvement

Conclusion

- Multilayer Perceptron and Conv1D are not accurate given its 2-dimension input
  - Trains on every frame
- Conv2D is more accurate given its 3-dimension input
  - Trains on every frame per video at a time
- Pretrained ImageNet Convolutional Neural Networks have better accuracy
- Time Distributed Convolution Neural Networks could be most accurate given its 4-dimension input
  - Trains every image per layer per video at a time
- Time Distributed Convolution Neural Networks with Long Short-Term Memory
  - Gives a relationship between frames of a video.

Limitations

- Enormous size of the database ~120GB
  - Must reduce size and take a random sample of videos
  - Only used one GPU
  - Limited memory and computing power
  - Reduces accuracy and creates unstable networks (varying accuracies and loss)
- Video Quality
  - Dark Environments
  - Low Resolution
  - Low Frames Per Second
  - Quick Moving Objects
- Classifying frames vs classifying videos

Improvement

- Video Augmentation and preprocessing
- Addition GPUs and processing power for the Virtual Machine
- Additional exploration of RNN/LSTM
- Classification output by frame

Research

- Raw dataset and some sample code
  - https://github.com/mchengny/RWF2000-Video-Database-for-Violence-Detection
- Time Distributed and LSTM
  - https://medium.com/smileinnovation/how-to-work-with-time-distributed-data-in-a-neural-network-b8b39aa4ce00
- Video Classification with Keras
  - https://www.pyimagesearch.com/2019/07/15/video-classification-with-keras-and-deep-learning/
- Violence Detection with LSTM
  - https://www.worldscientific.com/doi/pdf/10.1142/S2196888820500013
- 5 Video Classification Methods
  - https://blog.coast.ai/five-video-classification-methods-implemente [illegible] as-and-tenso 9cad29cc0b5
  -

Reference

- Ming Cheng, Kunjing Cai, and Ming Li. "RWF-2000: An Open Large Scale Video Database for Violence Detection." arXiv preprint arXiv:1911.05913 (2019).