Machine Learning Engineer Nanodegree

Capstone Project: Football Line of Scrimmage Detection

Pete Martin October 12, 2018

I. Definition

Project Overview

I'm a big fan of American football. I have been thinking of ways that Machine Leaning can be applied to the domain of sports media and in particular sport videos.

When watching videos of sport events viewers often would like to skip forward to the next action or play. This is applicable only to certain sports that have clear start and end of plays such as American football, tennis, and curling. These are called non-continous sports.

Ideally a computer vision algorithm would process a video and through frame analysis would detect at which of the frames contain what looks to be start of a play.

For the purpose of this project I chose American football as the sport and 11 videos around 30 minutes long to base the data set on. These videos or frames inside do not carry any meaningful metadata. I started with basically the raw video files and so manual (human) labeling is involved.

Here is an example of a video frame that would represent a start of a play (also referred to as "the line of scrimmage").

This project is about creating a Machine Learning algorithm that is able to classify individual frames from an American football video as either containing the line of scrimage (positive case) or not (negative case). It is therefor a binary classifier.

Problem Statement

The challenge I tried to solve was to create a binary classfier that would identify frames from American football videos that either contain a line of scrimmage (positive case) or do not (negative case). The only features available to us will be the raw image data (pixels of the frame). Here are few examples of positive cases (frames):



Here are few examples of the negative cases (image does not contain line of scrimmage):



Additionally I decided to include only the 'wide angle' line of scrimmage frames (like positive cases above) due to the nature of the videos I have been working with. These videos also contain 'tight angle' frames like this one but they are not actually the real start of of the play but rather a recap of the action from another camera angle:

Intended Solution

The intended solution I aimed for with this project was to create a binary classifier that given an image frame from a video would be able to accurately classify the image as either containing the wide-angle line of scrimmage (positive case) or not (negative case).

This algorithm could in the future be integrated with a larger video processing system that calls this classifier for every N'th frame (or every T seconds) of the video and consults it to determine if action / play is about to start on that frame. If so that larger system could then simply mark the time on the timeline that the positive case occurs and consider that a place where the user could skip to in order to watch the start of the next action / play. This project however is not concerned with the development of that larger video processing system and is instead focused on just building the image / frame classifier.

Since I was starting the project with just the video files and no other data at all, I will firstly have to manually label data used for model training, validation and testing.

Strategy

It is clear that the ideal candidate technology for the solution would be a form of a **Convolutional Neural Network**. This is in part because it is the overall image that has to be considered. One distinguishing feature of the images that contain the line of scrimmage (positive case) is that they have multiple players lined up in 2 rows. Example:

CNNs are perfect for identifying such features where multiple and neighboring areas of the image are the distingushing charectiristic.

CNN architectures are a big areas of research these days. As was said in the Udacity course on this topic, the CNN architecture design is more of an art than science and getting the network to perform well often comes down to number of trial-and-error runs.

The performance of our model will be measured with target Recall and Accuracy (more on that below.

Our larger strategy is as follows:

1. Build a very simple proof-of-concept network from the ground up, train it and see how it performs.
2. Attempt few tweaks to the intiail network to see if I can make make easy improvements.
3. If manual tweaking via trial-and-error do not yield great results, engage in a **Grid Search** approach to find optimial network architecture.
4. If I am still not getting the desired results, attempt **Transfer Learning** technique.

Tasks

Following is a high-level outline of tasks:

1. Project code base preparation (github repository, Python settings for virtual environment, etc).
2. AWS EC2 compute resource preparation (based on instructions provided during the Udacity course).
3. Acquire 11 video content (mp4 files). Each around 30 minutes long. Each of these represents about half of a college football game so the overall data set is equivalent to about full-length 5 games. I ended up with 11 video files because I selected videos with similar frame size & aspect ratios. In theory I could have selected larger number of videos but it would require larger investment in the manual labeling effort.
4. Write helper scripts to exatract around 200 frames per video (this would be around 1 frame extracted every 10 seconds but is randomized to avoid possible biases such as always picking the first frame of the video as the first frame).
5. Manually lebel the resulting 2200 images as being either positive case (contains line of scrimmage) or negative case (does not contain line of scrimmage).
6. Create harness code to perform certain basic operations such as loading images from the file system, creating tensors from them, computing model performacne and generating metric reports, functions to show False predictions.
7. Go into the iterative approach of trying to design the best performing model (see the Strategy section above)
8. Assuming I get the desired model performacne results, ensure the model is saved with all its computed weights so that it can be reused down the road without retraining the network again.

Metrics

Based on non-scientific observation the ratio of negative class instances to positive instances is about 5:1. Since the data set is not very well balanced I will use **Recall** as our main metric followed closely by the **Accuracy**. I am aiming for both to have fairly high level of performance:

Target Recall: 0.95

Target Accuracy: 0.95

Note: Recall = True Positives / All Positives

Accuracy = True predictions / All predictions

So as example:

Assume test data set contains 100 positive instances and 500 negative instances. If the model predicted 110 of the images to be positive (and rest as negative) but only 90 of those 110 were in fact positive, then the Recall is calculated as:

True Positives = 90

False Positives = 20

True Negatives = 490

False Negatives = 0

All Positives = 100

All predictions = 600

True predictions = 580

Recall = True Positives / All Positives = 90 / 100 = 0.90

Accuracy = True predictions / All predictions = 580 / 600 = 0.96

In addition to Recall and Accuracy I will look at all 4 elements of the Confusion Matrix (TP, FP, TN, FN), Precision and the F1 score.

In our tests I will use 20/80 split of testing vs training data and will also employ the Cross-Validation technique on 20% of the training data.

II. Analysis

Data Exploration

I have selected following college football games to serve as the basis of our data set:

- 5831135399001.mp4 | 2007 | Boston College Eagles (BC) @ Maryland Terrapins (UMD) | OFFENSE
- 5831135989001.mp4 | 2007 | Army Black Knights (Army) @ Boston College Eagles (BC) | OFFENSE
- 5831137888001.mp4 | 2007 | Boston College Eagles (BC) @ Virginia Tech Hokies (VT) | OFFENSE | ACC CHAMPIONSHIP GAME
- 5831138798001.mp4 | 2007 | Bowling Green Falcons @ Boston College Eagles (BC) | OFFENSE
- 5831138807001.mp4 | 2007 | Florida State Seminoles (FSU) @ Boston College Eagles (BC) | OFFENSE
- 5831139883001.mp4 | 2007 | UMass Minutemen (UMASS) @ Boston College Eagles (BC) | OFFENSE
- 5831139886001.mp4 | 2007 | NC State Wolfpack (NCST) @ Boston College Eagles (BC) | OFFENSE
- 5833078851001.mp4 | 2007 | Wake Forest Demon Deacons @ Boston College Eagles (BC) | DEFENSE
- 5833084561001.mp4 | 2007 | Boston College Eagles (BC) @ Michigan State Spartans (MSU) | DEFENSE
- 5833090951001.mp4 | 2007 | Boston College Eagles (BC) @ Clemson Tigers | DEFENSE
- 5833096735001.mp4 | 2007 | Miami Hurricanes @ Boston College Eagles (BC) | DEFENSE

For more information (including how to download these large files) see project file under ./data-videos/metadata.txt

Because these files are fairly large they could not be checked into the public github repo. However the images I extracted to serve as the data for the model were included under the ./data-images/ directory. For example 5831135399001-75.jpg indicates frame extracted at 75 seconds into the video 5831135399001.mp4.

The metadata file containing manual classification for the images is under ./data-labels/images.csv and looks like this:

	A	B	C	D
1	VIDEO ID	IMAGE ID	IMAGE FILENAME	IS POSITIVE
2	5831135399001	3	5831135399001-3.jpg	
3	5831135399001	26	5831135399001-26.jpg	
4	5831135399001	37	5831135399001-37.jpg	
5	5831135399001	41	5831135399001-41.jpg	
6	5831135399001	66	5831135399001-66.jpg	
7	5831135399001	75	5831135399001-75.jpg	
8	5831135399001	79	5831135399001-79.jpg	1
9	5831135399001	88	5831135399001-88.jpg	
10	5831135399001	90	5831135399001-90.jpg	
11	5831135399001	121	5831135399001-121.jpg	
12	5831135399001	128	5831135399001-128.jpg	
13	5831135399001	134	5831135399001-134.jpg	
14	5831135399001	135	5831135399001-135.jpg	
15	5831135399001	140	5831135399001-140.jpg	1
16	5831135399001	142	5831135399001-142.jpg	
17	5831135399001	151	5831135399001-151.jpg	
18	5831135399001	153	5831135399001-153.jpg	
19	5831135399001	155	5831135399001-155.jpg	
20	5831135399001	172	5831135399001-172.jpg	
21	5831135399001	177	5831135399001-177.jpg	
22	5831135399001	187	5831135399001-187.jpg	
23	5831135399001	212	5831135399001-212.jpg	
24	5831135399001	218	5831135399001-218.jpg	1

Stats on the Data Set

I extracted total of 2,200 images from the 11 videos. That was about 1 frame every 10 seconds but the sampling was completely random.

During the manual labeleing process I found the data in this section to be somewhat imbalanced. 21% of images were manuually classified as Positive. This however was not a surprise sicne that was what our estimates were at the start of the project.

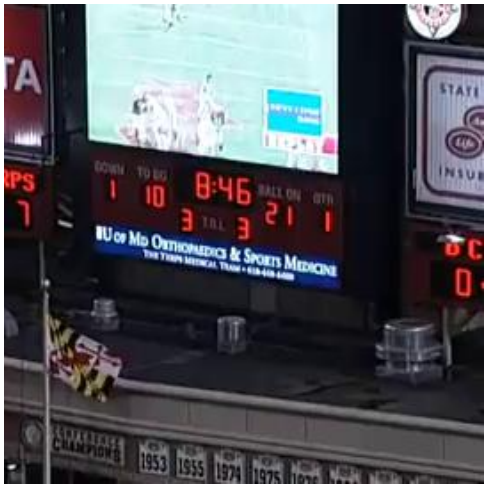
Abnormalities or Interesting Qualities

Here are few interesting examples of the input images:

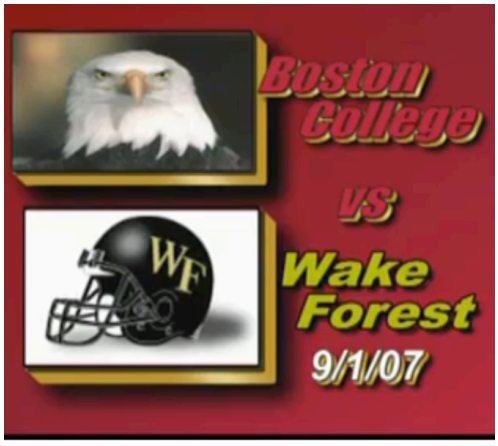
Some of the images show action just fractions of seconds after the play has started (where the line if scrimmage is no longer statatic and distinct). To a human eye this is clearly no longer the line of scrimmage but the model did get confused about these kinds of frames:

Some fo the camera angles & zoom show line of scrimmage (positive case for our model) but look completely different from the rest of the positive images:

Many of the frames contain shots of the scoreboard. Luckily the images displayed on the scoreboard (like the large display showing the action) did not confuse the model:



Videos typically begin with few seconds of frames showing the game title / team names and date:



Some of the images contain close-ups of action:

About 40% of the images contain the tight-angle camera view. This camera angle is is not useful to us.

Some of the images will be difficult to work with as the background colors on the field (like the end-zone) blend with the jerses of the players:

Frames that are somewhat hard to label as being line of scrimamge or not (i.e. not all players are in the bent-forward and ready position):

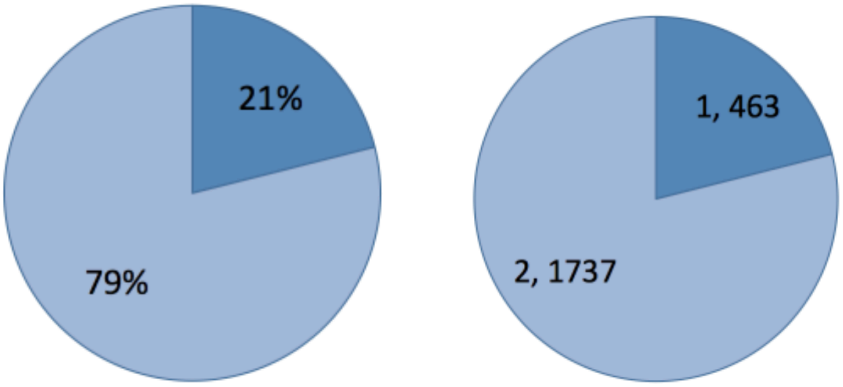
Small number of images contains somewhat blurry action and/or action from the sidelines:

Note: one important part to note here is that our data is naturally augmented as there are many different angles showing the line of scrimmage. There are different colors and zoom involved as well. Additionally the line of scrimage does not always appear in the center of the screen.

This is great for the elasticity / adaptivity of the final model but should be also noted that it required us to obtain a lot more data samples since the variance was fairly high.

Exploratory Visualization

As described above, the split of positive vs negative cases in our data is about 1-to-5:



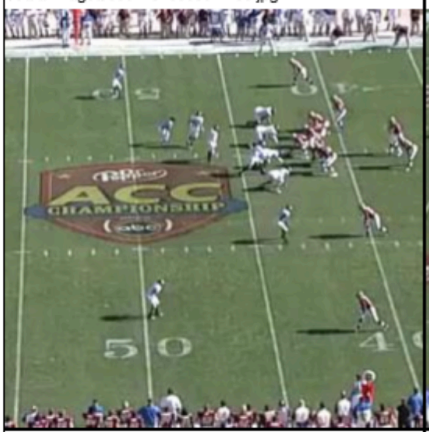
Here are some examples of positive image cases (all of them can be viewed via display_data.ipynb):

./data-images/5831135399001-79.jpg:

./data-images/5831135399001-140.jpg:

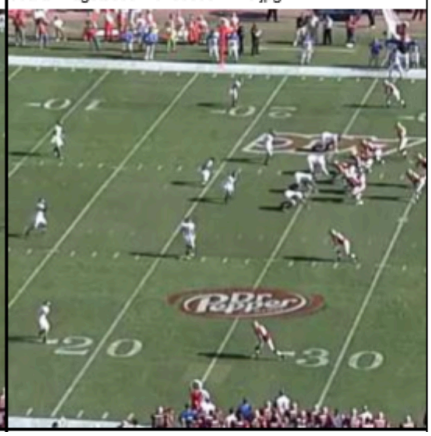
./data-images/5831135399001-218.jpg:

./data-images/5831137888001-138.jpg:



./data-images/5831137888001-195.jpg:

./data-images/5831137888001-246.jpg:

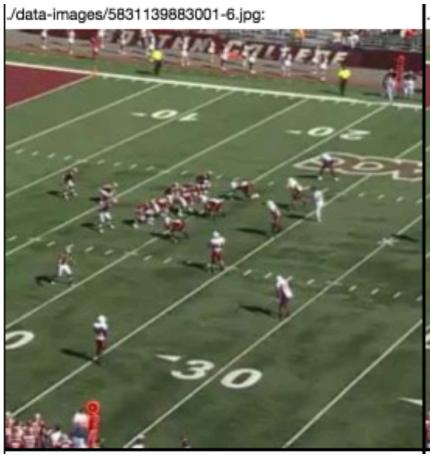


./data-images/5831137888001-268.jpg:

./data-images/5831137888001-269.jpg:

./data-images/5831137888001-481.jpg:

./data-images/5831139883001-6.jpg:



./data-images/5831139883001-9.jpg:

./data-images/5831139883001-34.jpg:

./data-images/5831139883001-624.jpg:

./data-images/5831139883001-644.jpg:

./data-images/5831139883001-692.jpg:

Algorithms and Techniques

As described in the Strategy section above, computer vision and image classification problems are perfect for CNNs.

The challenging part of this project if finding the CNN architecture that accomplishes our overall model performacne objectives.

At the start of the project it was not clear which architecture or methodology would perform well so I decided to pursue different techniques and architectures in the order of difficulty or investment in processing time. Many tests and models have been ran in parallel since some of them took days to run on dedicated AWS hardware:

1. Build a very simple proof-of-concept network from the ground up, train it and see how it performs.
2. Attempt few tweaks to the intiail network to see if I can make make easy improvements.
3. If manual tweaking via trial-and-error do not yield great results, engage in a **Grid Search** approach to find optimial network architecture.
4. If I am still not getting the desired results, attempt **Transfer Learning** technique.

Benchmark

I am not aware of anybody attempting to classify football video frame images into above-mentioned classes. If I used a random binary classifier as our benchmark and I assume the model is evaluated based on the Recall & Accuracy metrics then our data set would calculate to roughly:

Data	
all cases	2,200
positive cases	463
negative cases	1,737
Random classifier's performance	
True Positives:	231.5
True Negatives	868.5
False Positives:	231.5
False Negatives:	868.5
Metrics:	
Recall = True Positives / All Positives	0.50
Accuracy = True predictions / All predictions	0.50

As mentioned above in the Metrics section, the target performance metrics for our model are:

Target Recall: 0.95

Target Accuracy: 0.95

Hence I am attempting to build a classfier that performs about 2x better than just a random binary classfier.

III. Methodology

Data Preprocessing

The preprocessing of the data involved mainly extraction of random sample of 200 images frames from each of the 11 videos I downloaded for this project (hence our total data size ended up being 2,200).

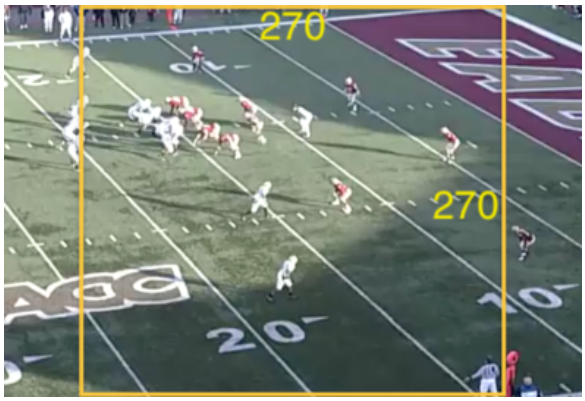
The input videos were treated as just plain raw video media files. I did not include any additional features in the data (such as which teams were playing or what is the color of the end-zone on the field). These kinds of additional features could have possibly helped in training the algorithm but our goal was to be able to take any raw video without any additional metadata and find the starts of plays based purely on image analysis.

Video files came in two sizes (in pixels):

- 406 x 270
- 360 x 270

The quality of the video stream in all these videos is aproximately the same.

Since I anticipated need to engage a pre-trained netural network (ResNet50, VGG## or others) I decided to crop the image to its original height but square shape like so:



The line of scrimmage always was visible inside the crop square.

Data pre-processing was using the ./extract_frames.py script. Please note that in order to run this script you need to have not only the 'av' package installed but also the system **ffmpeg** library installed on your environment.

Dealing with Abnormalities

The somewhat abnormal frames in the videos (from examples above) have not been eliminated from our data set. This is because I wanted the model to be as adaptable to abnormalities as possible and to be able to process any frame from a video and correctly classifly it.

Implementation

To begin I created an empty gihub repository with just the basic standard files like the README, LICENSE, .gitignore, etc. I also checked in the proposal.pdf and .md to keep track of the original Proposal documented submitted.

Next step in the base implementation was to create a Python environment I could work within. For that I gathered list of packages I will require and placed them in the requirements.txt file.

Next I have implemented and tested bunch of helper functions (for things like loading images into tensors, evaluating predictions and generating output reports with the key metrics). Any code that made sense to reuse between notebooks landed here (common.py).

Once the basics were set up it was time to start building models using Jupyter Notebooks.

Since there was fair bit of trial & error involved in coming with the final CNN architecture, below is a somewhat chronological outline of what attempts were made, decisions taken, trial paths retracted from, etc. The Refinement section below has detailed notes on how different models have been tried, evaluated and refiend and how the overall approach changed as I learned more.

Refinement

PART 1: "architecture_01"

(see architecture_01.ipynb)

The first attempt to build the CNN from scratch produced fairly promising results. Almost out of the box. The architecture of the CNN used here was borrowed from the Dog Breed

project and looked like this (with the exception that early attempts used 224x224 image size instead of 270x270 shown below):

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 270, 270, 16)      208
_________________________________________________________________
batch_normalization_1 (Batch (None, 270, 270, 16)      64
_________________________________________________________________
activation_1 (Activation)    (None, 270, 270, 16)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 135, 135, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 135, 135, 32)      2080
_________________________________________________________________
batch_normalization_2 (Batch (None, 135, 135, 32)      128
_________________________________________________________________
activation_2 (Activation)    (None, 135, 135, 32)      0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 67, 67, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 67, 67, 64)        8256
_________________________________________________________________
batch_normalization_3 (Batch (None, 67, 67, 64)        256
_________________________________________________________________
activation_3 (Activation)    (None, 67, 67, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 33, 33, 64)        0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 130
=================================================================
Total params: 11,122
Trainable params: 10,898
Non-trainable params: 224
```

The Recall and Accuracy metrics were around 0.70 – 0.80 (it varried a bit with every training run). As planned I have gone into various attempt to optimize the model. I manually manipulated hyperparameters such as:

- image size (224x224 vs 270x270)
- color vs grayscale image
- the activation function (relu vs sigmoid)
- number of epochs
- batch size (20, 50, 100)
- kernel size (2x2 vs 3x3)
- number of hidden layers (2, 3, 4)

I attempted few of these changes by hand but since re-training of the network took long time on each iteration, I eventually gave up trying to find the optimal architecture for this network by hand.

After few days of trial and error I arrived at the architecture currently outlined above with following performance characteristics (note that I have computed all the metrics not only on the test data set but also on the train and validation data set to see if the model has over-fitted the data):

```
_____________________________________________________________
TIMING SUMMARY:

loading duration: 6.7 seconds
training duration: 31363.9 seconds
_____________________________________________________________
```

```
MODEL PERFORMANCE ON TEST DATA:

predict duration: 100.6 seconds
all:  440
all_positives:  81
all_negatives:  359
true_positives:  70
true_negatives:  341
false_positives:  18
false_negatives:  11
RECALL: 0.86
SPECIFICITY: 0.95
ACCURACY: 0.93
PRECISION: 0.80
F1 SCORE: 0.83
FP RATE / ERROR I: 0.05
FN RATE / ERROR II: 0.14
_____________________________________________________________
MODEL PERFORMANCE ON TRAIN DATA:

predict duration: 324.3 seconds
all:  1408
all_positives:  316
all_negatives:  1092
true_positives:  255
true_negatives:  1026
false_positives:  66
false_negatives:  61
RECALL: 0.81
SPECIFICITY: 0.94
ACCURACY: 0.91
PRECISION: 0.79
F1 SCORE: 0.80
FP RATE / ERROR I: 0.06
FN RATE / ERROR II: 0.19
_____________________________________________________________
MODEL PERFORMANCE ON VALIDATION DATA:

predict duration: 79.8 seconds
all:  352
all_positives:  66
all_negatives:  286
true_positives:  53
true_negatives:  271
false_positives:  15
false_negatives:  13
RECALL: 0.80
SPECIFICITY: 0.95
ACCURACY: 0.92
PRECISION: 0.78
F1 SCORE: 0.79
FP RATE / ERROR I: 0.05
FN RATE / ERROR II: 0.20
```

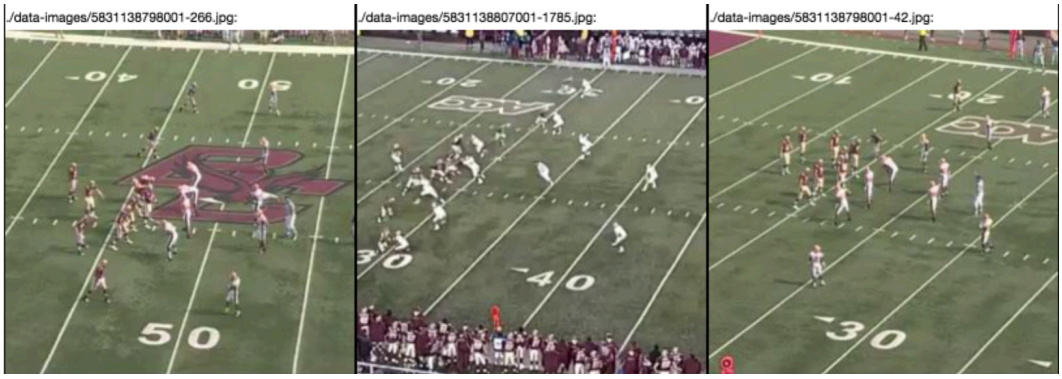
Notice that training process with 50 epochs took almost 9 hours for each run.

The architecture_01.ipynb represents the final state of the network architecture with most optimal and seemingly stable performance & results. The best model showing results above was saved to ./saved-models/achitecture-01.hdf5

It was very interesting to analyze the specific instances of False classifications. For example, following are examples of False positives.

Since the network likely looks for lines in the image, it is easy to see how to got confused on the images below. Both the images on the left and right below would have been labeled by a human as positive if only the players on the field bent forward. They have already formed the lines that the algorithm might be looking for but clearly were still a second or two away from bending forward to get ready for the whistle.

The image in the center is an example of a frame just after the play begun but clearly too late to be classified as containing steady line of scrimmage:



At the same time the model clearly made mistakes classifying some of the images as negative while they clearly contained all the clear features of line of scrimmage (hence we can't begin to guess why the model made these mistakes or how to begin to address them):

This manually composed model did not reach our target goal for performance (Recall and Accuracy both at 0.95) so it was time to move on to another optimization methodology.

PART 2: Grid Search

(see grid_search.ipynb)

Since the attempt to optimize the slow-training model above did not accomplish our performance goal, I decided to leverage grid search techniques to possibly find the most optimal architecture (while still making some assumptions about that certain parameters would look like).

The permutations below produced around 48 different models that were searched through using sklearn's GridSearchCV:

```
optimizer=['rmsprop', 'adam'],

activation=['relu', 'sigmoid'],

network_layers=[2, 3, 4],

kernel_size=[2, 3],

batch_size=[10, 50],
```

Even with only 3 epochs per network, the computation required over 30 hours of compute time on AWS's server.

In the end the Grid Search came up with following compbination of optimal parameters:

```
'activation': 'relu',

'batch_size': 10,

'kernel_size': 2,

'network_layers': 4,

'optimizer': 'rmsprop'
```

However, even this configuration of parameters showed Grid Search output as performing somewhat poorly (Accuracy of 0.85).

Since the Grid Search used only 3 epochs to train the optimal model, I decided to create another model with those same hyperparameters that were identified by the Grid Search but running on 20 epochs insteaed of just 3. See architecture 02 section below.

PART 3: "architecture_02"

(see architecture_02.ipynb)

Using the paramters found during the Grid Search I built another network and trained it using 20 epochs:

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 270, 270, 16)	208
batch_normalization_5 (Batch	(None, 270, 270, 16)	64
activation_5 (Activation)	(None, 270, 270, 16)	0
max_pooling2d_5 (MaxPooling2	(None, 135, 135, 16)	0
conv2d_6 (Conv2D)	(None, 135, 135, 32)	2080
batch_normalization_6 (Batch	(None, 135, 135, 32)	128
activation_6 (Activation)	(None, 135, 135, 32)	0
max_pooling2d_6 (MaxPooling2	(None, 67, 67, 32)	0
conv2d_7 (Conv2D)	(None, 67, 67, 64)	8256
batch_normalization_7 (Batch	(None, 67, 67, 64)	256
activation_7 (Activation)	(None, 67, 67, 64)	0
max_pooling2d_7 (MaxPooling2	(None, 33, 33, 64)	0
conv2d_8 (Conv2D)	(None, 33, 33, 128)	32896
batch_normalization_8 (Batch	(None, 33, 33, 128)	512
activation_8 (Activation)	(None, 33, 33, 128)	0
max_pooling2d_8 (MaxPooling2	(None, 16, 16, 128)	0
global_average_pooling2d_2 (	(None, 128)	0
dense_2 (Dense)	(None, 2)	258

Total params: 44,658
Trainable params: 44,178
Non-trainable params: 480

Results were clearly better here than what I got with my initial “architecuture 01” (which was largely a shot in the dark).

However, this new “architecture 02” still didn’t attain the levels of performances I have set as the target for the solution:

```
_________________________________________________________________
TIMING SUMMARY:

loading duration: 26.1 seconds
training duration: 8913.1 seconds

_________________________________________________________________
MODEL PERFORMANCE ON TEST DATA:

predict duration: 40.5 seconds
all:  440
all_positives:  81
all_negatives:  359
true_positives:  71
true_negatives:  343
false_positives:  16
false_negatives:  10
RECALL: 0.88
SPECIFICITY: 0.96
ACCURACY: 0.94
PRECISION: 0.82
```

```
F1 SCORE: 0.85
FP RATE / ERROR I: 0.04
FN RATE / ERROR II: 0.12
_________________________________________________________________
MODEL PERFORMANCE ON TRAIN DATA:

predict duration: 124.7 seconds
all:  1408
all_positives:  316
all_negatives:  1092
true_positives:  255
true_negatives:  1047
false_positives:  45
false_negatives:  61
RECALL: 0.81
SPECIFICITY: 0.96
ACCURACY: 0.92
PRECISION: 0.85
F1 SCORE: 0.83
FP RATE / ERROR I: 0.04
FN RATE / ERROR II: 0.19
_________________________________________________________________
MODEL PERFORMANCE ON VALIDATION DATA:

predict duration: 31.2 seconds
all:  352
all_positives:  66
all_negatives:  286
true_positives:  49
true_negatives:  274
false_positives:  12
false_negatives:  17
RECALL: 0.74
SPECIFICITY: 0.96
ACCURACY: 0.92
PRECISION: 0.80
F1 SCORE: 0.77
FP RATE / ERROR I: 0.04
FN RATE / ERROR II: 0.26
```

Above model has been saved to ./saved-models/achitecture-02.hdf5

PART 4: Transfer Learning

(see transfer_learning.ipynb)

During the Dog Breed classification project we learned that using parts of pre-trained networks can vastly improve performance of our own models. There are many different networks that have competed in the ImageNet initiative and I had a wide variety of them to choose from. I read through some meterials that compare the different types of networks. Example resource:

https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

https://openreview.net/pdf?id=Bygq-H9eg

In the end any of the models seemed like a good place to start so I begun with VGG16, thinking that if I don't attain the desired results I would try others.

Once again selecting combined transfer learning architecture was a bit of an educated guess work but in the end I ended up removing all but the first 6 layers (4 convolutional layers and 2 max-pooling). Initially I tried to simply add a fully connected layer on top of that but found the performance to be horrible (there weren't perhaps enough parmeters to train and the problem domain is fairly specific). So I added another convolutional layer on top of the 6 frozen VGG layers and arrived at following architecture:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 56, 56, 128)       65664
_________________________________________________________________
batch_normalization_1 (Batch (None, 56, 56, 128)       512
_________________________________________________________________
activation_1 (Activation)    (None, 56, 56, 128)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 28, 28, 128)       0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 258
=================================================================
Total params: 326,594
Trainable params: 66,178
Non-trainable params: 260,416
```

This arechitecture required us to rescale the images to 224x224 which was done by code added for this purpos to our common.py library.

The model was trained with:

- 50 epochs
- batch sizes of 20

The new combined model attained the target performance. Our targets were initially set as:

Target Recall: 0.95

Target Accuacy: 0.95

The model produced:

Recall: 0.98

Accuacy: 0.95

Below is a full report with other metrics included:

```
_________________________________________________________________
TIMING SUMMARY:

loading duration: 8.5 seconds
training duration: 43418.8 seconds
_________________________________________________________________
MODEL PERFORMANCE ON TEST DATA:

predict duration: 188.8 seconds
all:  440
all_positives:  81
all_negatives:  359
true_positives:  79
true_negatives:  340
false_positives:  19
false_negatives:  2
RECALL: 0.98
SPECIFICITY: 0.95
ACCURACY: 0.95
PRECISION: 0.81
F1 SCORE: 0.88
FP RATE / ERROR I: 0.05
FN RATE / ERROR II: 0.02
_________________________________________________________________
MODEL PERFORMANCE ON TRAIN DATA:

predict duration: 546.3 seconds
all:  1408
all_positives:  316
all_negatives:  1092
true_positives:  298
true_negatives:  1018
false_positives:  74
false_negatives:  18
RECALL: 0.94
SPECIFICITY: 0.93
ACCURACY: 0.93
PRECISION: 0.80
F1 SCORE: 0.87
FP RATE / ERROR I: 0.07
FN RATE / ERROR II: 0.06
_________________________________________________________________
MODEL PERFORMANCE ON VALIDATION DATA:
```

```
predict duration: 139.3 seconds
all:  352
all_positives:  66
all_negatives:  286
true_positives:  62
true_negatives:  266
false_positives:  20
false_negatives:  4
RECALL: 0.94
SPECIFICITY: 0.93
ACCURACY: 0.93
PRECISION: 0.76
F1 SCORE: 0.84
FP RATE / ERROR I: 0.07
FN RATE / ERROR II: 0.06
```

Additionally it does not appear that the model over-fit the data.

This model took about 12 hours to train on AWS compute resources. The model snapshot has been saved to ./saved-models/transfer_learning.hdf5

Below are some interesting examples of False classifications:

Fraction of a second after the play has started (line of scrimmage is still some visible and players look bent over but they are actually already in motion):

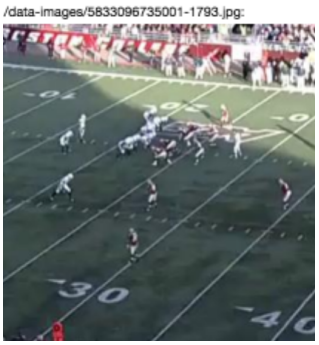
./data-images/5831139883001-314.jpg:

./data-images/5833078851001-2231.jpg:

Here some of the players have already bent over and so one could say this frame is sort of in-between case.

The only two False Negatives are interesting as well:

Perhaps the shadows on the field have overpowered the signal for the “lines” of scrimmage:



The color of the endzone might have confused the model with the player in the line of scrimmage:

Complications Found Along the Way

1. Installing the ‘av’ library was a bit more complicated than initially expected. The system requirements (installation of ffmpeg) to support this library ended up being poorly documented.
2. Found that in somewhat low resolution of the frames (height of only 270 pixels) certain features of the image blend even to a human eye. For example, this image

has the players blending with the background of the field endzone. This in many cases created False Positives:

These kinds of issues pushed me to incrase resolution of the images from the initial 224x224 to 270x270. These kinds of issues (which led to large number of False predictions) also pushed me to include more videos and labeled frames in the data set (I increased the number of images from 1,400 to 2,200 at somepoint during this project)

3. Higher number of data samples and larger resolution took significantly longer for the models to train. Some of the models required as much as 12+ hours to train on AWS compute resources.

4. Running long Notebook sessions has a significant problem: if the browser disconnects from the server it is unable to reconnect and continue displaying results as the notebook continues to execute. There isn't a good work around for this. I was forced to leave my laptop plugged in and stay on the same Wi-Fi for days at a time.

5. The video files were too big to store on GitHub so I had to find alternative location to store them (my personal dropbox account).

Code Complexity & Documentation

For the most part code & notebooks included in this project are fairly straightforward. I have included fair bit of documentation in-line within the code. The challenge of correctly installing the 'av' library has been documented in the main README.md file.

IV. Results

Model Evaluation and Validation

Much of what led to the final model selection has been already described above in the “Refinement” section. Here is a final summary of the chosen model:

The final model (see transfer_learning.ipynb) was selected purely based on the fact that it was the only model that satisfied our target metrics of performance.

Model robustness and performance on unseen data

I believe the model I selected has naturally been tested for sensitivity as the input data varied significantly. Here are some of the variances that were built into the testing data (in addition to player formations which are the key to correct classification):

- Varying camera zooms
- Varying playing field colors and colors of player jerseys
- Varying light conditions (shadows, etc)
- Varying quality of images (pixalation, blur, etc)

Despite all of these variances the model still managed to attain the target metrics.

Model’s Approrietness

Since the model was trained on variety of videos. I believe that the model would be approriate to use on any other football game that contains the type of wide-camera angle we presented as positive cases in this project. This essentially covers all American football videos that came from the broadcast television.

It is not clear how the model would perform on user-generated content (think shaky footage captured on somebody’s cell phone from the seats in the stadium).

Trusting the Model’s Results

The model’s final Recall and Accuracy on the test data set has proven to meet our target criteria (Recall and Accuracy both at 0.95). Reruning predictions on both the training and the validation set have yield similar although slightly lower. In both cases they were:

Training & Validation set Recall:	0.94
Training & Validation set Accuracy:	0.93

One one hand it is reassuring to see that the model did not over-fit on the training data but at the same time it is a bit strange that its performance lowered by ~1% compared to the test set.

None the less the story the final numbers tell is that the model produces reliable accuracy at around 95% for all data we were able to label.

Justification

The benchmark model (random binary classifier) that was proposed above would have produced following performance metrics (see the worksheet above as well):

Recall: 0.50

Accuracy: 0.50

The final model in this project produced significantly stronger results:

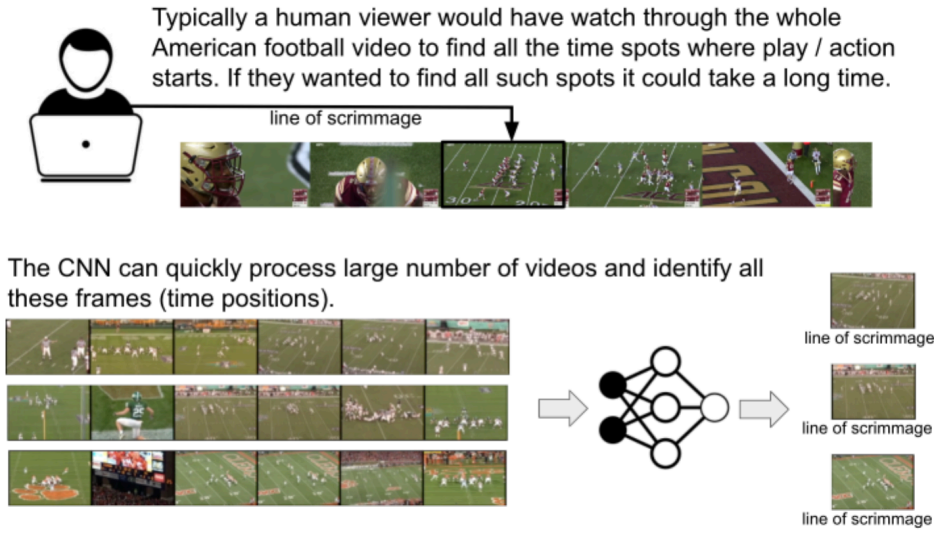
Recall: 0.98

Accuracy: 0.95

Since the performance of the final model met our target metrics (both metrics at min of 0.95) the proposed model does solve the problem that we tackled in this project.

V. Conclusion

Free-Form Visualization



Reflection

In this project we created a CNN model to detect video frames containing lines of scrimmage in American football (broadcast media).

The project started off by setting fairly high expectations on the model performance in terms of Recall and Accuracy.

Data for the project was extracted from raw video files and pre-processed to uniform image dimensionality.

Manual labels were created for all 2,200 data instances (image frames).

Multiple network architectures and neural network building techniques were experimented with as the performance of the model improved up to the target levels.

Along the way many lessons were learned from looking at False prediction examples.

Interesting Aspects

There were many interesting aspects of this project nd lessons learned but here are a few that I thought were particularly intriguing:

1. Humans can be wrong and corrected by ML:

   During early phases of labeling I made a mistake labeling following frame as a Positive. The model however predicted it to be Negative (which is the truth consideirng the lines of scrimmage have collapsed and players are already in motion).

   It was very interesting see the network correct the human label in this instance.

2. Amazing results considering the training data:

Frankly I was amazed that desipite the large variances in Positive data instances in the training set and relatively small number of them, the final network used in this project was able to attain the high levels of performance.

I am also very fascinated by the idea of transfer learning since it reminds me of the sci-fi themes of freezing or storing away one's mind and restoring it down the road. In essence this transfer learning does just that. As the public-domain and research neural networks become more and more complex, we will be able to build even smarter machines by leveraging what others have althought taught them. Maybe one day we will be able to start transfer learning from a base 'simple-human-mind' model and teach it what we want like you would a human child.

Difficult Aspects

Here are some of the difficult aspects I found about the project:

1. Neural Network architecture design process:

   As mentioned in the course materials, CNN architecture design is still more of a trial-and-error process than a scientitific process. It is hard to guess all of the parameters that the network should have to obtain optimal performance. There are no golden rules about how many layers to use for a given problem, how many filters or how to best transform the image ahead of time to make it more suitable for the network. There is a lot of research out there right now that pertains to visualizing what the network is doing as it learns (e.g. filter map visualizations, etc). There are tools out there one can try to get some help in analysing and refining performance of network layers but it is all still very experimental. I wish the course itself had gone a bit deeper into these kinds of tools and processes.
   So the challenging part in the project here was just doing a lot of guesswork and trying to find resources online that can help.

2. Neural Network training is an expensive process:

   Even with just 2200 data samples the networks that have been trained during this project sometimes took 12+ hours to train. This created two problems:

   - I had to engage compute resources from AWS but that created created another complication that I had to leave my laptop's browser open on the Notebook and plugged into the network so that it doesn't go to sleep, loose connection, etc. Notebook platform is not very robust as far as preserving state and allowing users to reconnect to previously started sessions.
   - Iterating over different architectures took very long time. It would sometimes take hours for me to even begin to guess if architecture was going to perform well or not. That slowed down the whole project. Once I used Grid Search I got slightly better results but that process itself ran for over 30 hours on AWS.

Practical Application

I believe the model that was built, as part of this project, could be leverage in a real production system to help humans identify begininings of video clips where the football action begins:

1. Model's accuracy and recall metric performed at a fairly high level of 0.95 and 0.98 respectively. This means the model will make some mistakes but that overall it would be a great aid and identify vast majority of action sub-clips in a video. There are also secondary models that could be employed on top of this model to improve confidence levels (see below).
2. In the tests we performed using the final CNN model we have observed speed performance of about 2.3 frames processed per second. This was done on a fairly small EC2 compute instance (not much faster than a personal laptop). If this algorithm was to be employed at larger scale compute cluster it would certainly outperform the 'real-time' of the video's playback and therefore produce much faster than a human could.
3. The model employs fairly standard set of Python and ML libraries and could be easily deployed onto a cluster of servers. The project structure is a fairly typical one (with requirements.txt for dependencies, etc). It took me only minutes to configure the project on a AWS compute resource. The only complication is ensuring the ffmpeg library is installed but that is a very simple system command on most Linux operating systems and distributions.

Improvement

This project attained fairly high levels of our target performance metrics and I believe it would be a great new benchmark for other and future projects to compare to. The original data set is included in this project to others can leverage it and attempt to build an model that outpeforms what was built here.

There are many improvements that could be leveraged within this project or in future projects that attempt to solve this problem:

1. **Train on more data**: I believe the more data we could feed to the model, the better. Considering large variances in the training data samples (colors, zoom, orientation, etc) it is wonder the network learned it all. Still, I feel better performance could be attained by adding perhaps 2x or 5x the data. This would of course require human time to manually label this data.
2. **Try different base transfer networks**: In this project we only (successfully) tried the VGG16 network. The ResNet was attempted as well but I ran into compatibility and dimensionallity issues. If I had more time on this project, I would love to try many other base networks as well (includig Inception, Xception and others). I would love to see how each of them performs in terms of accuracy / recall metrics but also compute time.

3. **Secondary model / logic on top of this model:** To leverage the model from this project in production I would probably create secondary model or methodology that takes a statistical approach before issuing the final recommendations as to where the line-of-scrimmage plays start in the video. If the model could be employed to say run on every frame of the video (or maybe every 0.5 seconds) there would likely be clusters of Positive predictions like:

   0,0,0,0,0,1,1,0,1,1,0,0,0,0,0,0,0,0

   Even thought the highlighted prediction was perhaps a False Negative, the wrapping model or processing could recognize that it is an abnormality and ignore it. Also if the model predicted just a signle positive frame like this:

   0,0,0,0,0,0,1,0,0,0,0,0,0

   The wrapping model could assume that is is very unlikely that line of scrimmage formed for just 0.5 seconds in the game and could ignore this prediction and assume it was a False Positive.

   On other words, the model that was built in this project could be leveraged as input into other statistical or clustering models that look for certain patterns in predictions on the timeline.

4. **Visualizations for the learning process:** If I had a bit more time on this project I would love to try out some of the visualization libraries and methodologies that are out there that help network designers understand how their models are training and behaving (and also structured). Here are few interesting links I found on this topic:

   https://jacobgil.github.io/deeplearning/filter-visualizations

   https://github.com/raghakot/keras-vis

   https://towardsdatascience.com/visualizing-artificial-neural-networks-anns-with-just-one-line-of-code-b4233607209e