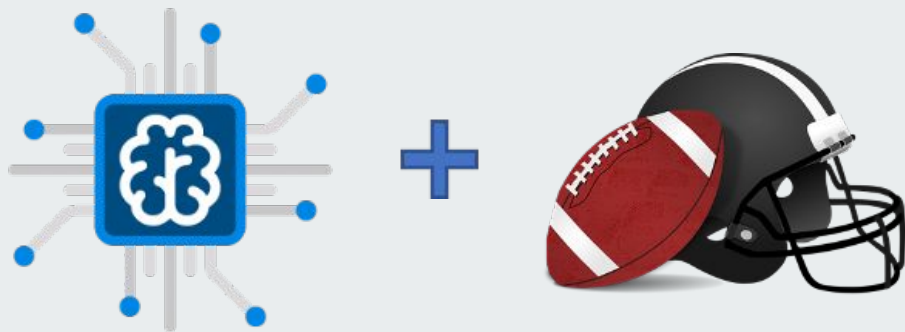




Automatic Classification of American Football Plays Final Report

Group 12:

Artur Bessa Cabral
Joel Corporan
V. Vinh Huynh
Pedro H.R. Pinto



Outline



- ❖ Motivation and Objectives
- ❖ System Design
- ❖ Classification Model:
 - Proposed Work
 - Data Description
 - Data Preprocessing
 - CNN-RNN Model
- ❖ Use Case and Prototype Demo
- ❖ Code Walkthrough
- ❖ Training Models and Results
- ❖ Future work

Motivation & Objectives

American Football is in high demand for new and innovative strategies to better predict outcomes and augment the viewers' experience.

The “Yellow Line” or 1st & Ten developed by Sportvision offers an instant appreciation to the audience of where the offense would take the ball to make another first-down.

There is always a place for more innovation and growth.



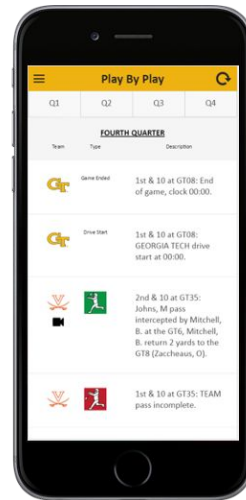
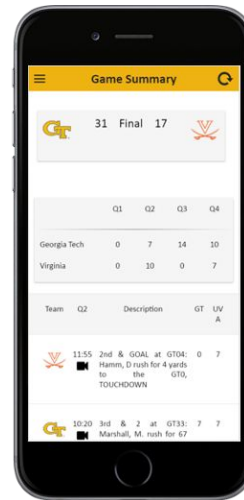
Source: Sport Illustrated

Motivation & Objectives

Stadium-IoPT VIP Research Team

- For years, the Stadium-IoPT team developed and operated a web application found at <http://estadium.gatech.edu/>
- Features:
 - Videos of every play with their official NCAA descriptions
 - Team Statistics
 - Drive Tracker
- Rich and well-labeled video clips of football plays.

However, these clips are manually sliced and labeled by students without any automation to match the videos with the proper description.



Motivation & Objectives



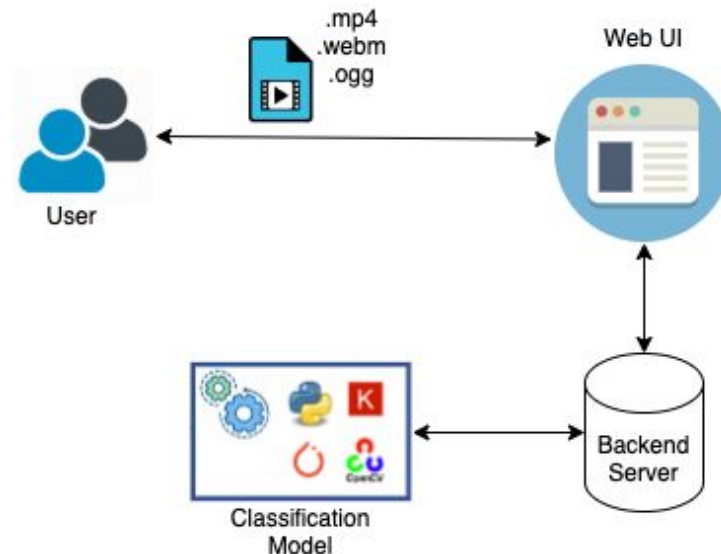
Our Solution: Developing a computer vision model to classify videos of football plays by type. This model will be incorporated in a web application where users can upload new videos and receive the prediction. This service aims to reduce the need for human intervention when classification is needed.

Potential use cases:

- Provisioning labeled videos to direct-to-consumer applications
- Quick retrieval of classified videos for coaches and players
- Creating a better analysis to predict play outcomes that the offense team might perform
- Integration with media services

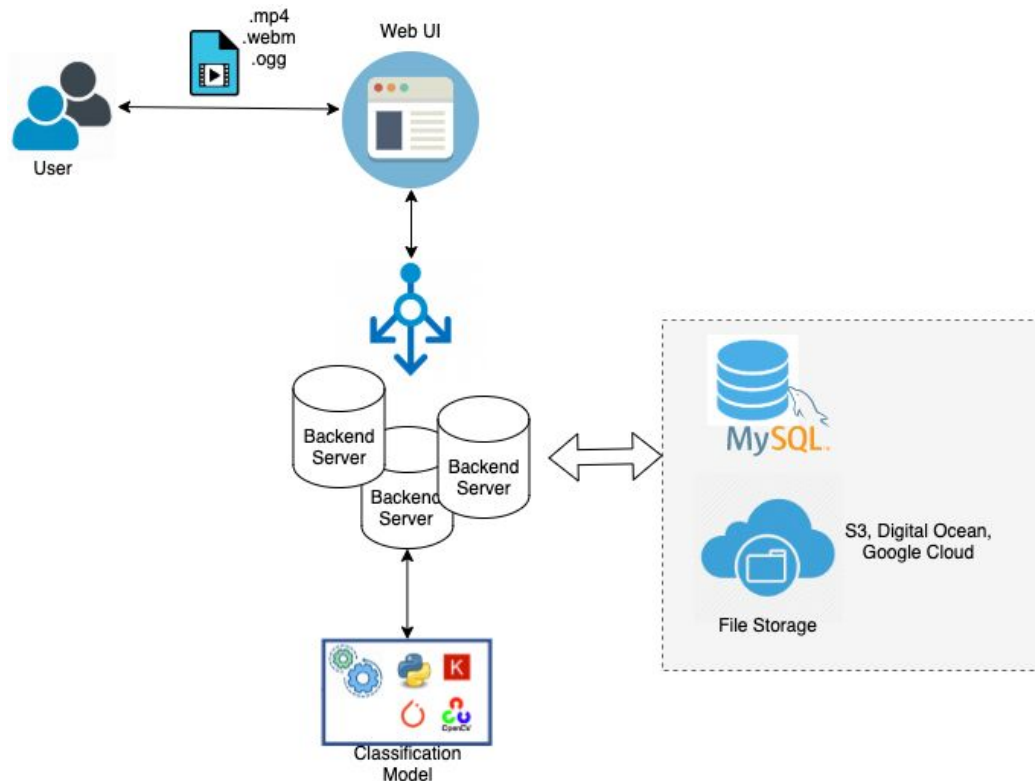
System Design—Workflow

- ❖ User uploads video as input file to our web page.
- ❖ Web server sends video file to web backend server.
- ❖ Backend server uses our pre-trained classification model to determine the likelihood of different plays based on input video.
- ❖ Backend server returns and displays results on Web UI.



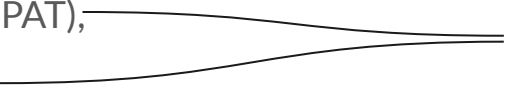
System Design—Scaling Up

- ❖ Adding more servers.
- ❖ Make use of MySQL and Cloud File Storage.
- ❖ Archive user's uploaded videos to enrich our datasets for maintaining and improving our models over time.



Classification Task



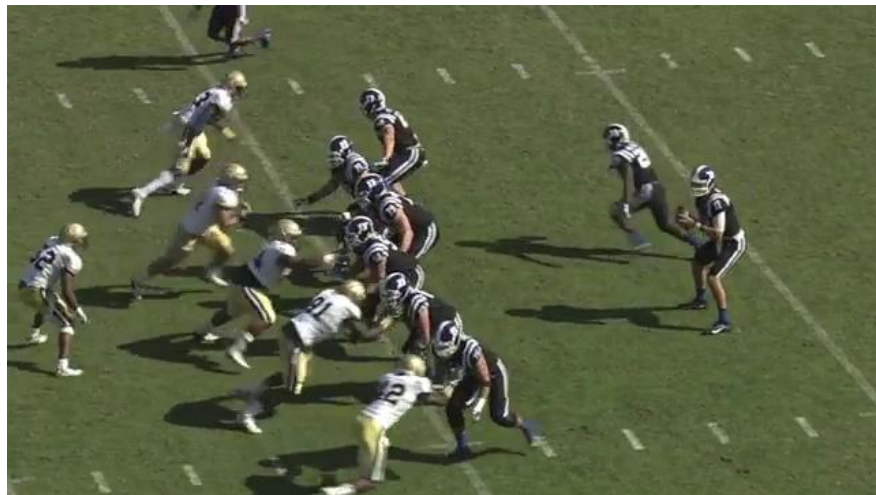
- The main task of this project was to build a model to automatically classify videos of football plays initially into 6 categories, but now 5:
 - 'K': Kickoff,
 - 'R': Rushing Play,
 - 'P': Passing Play,
 - 'X': Extra Point (PAT),
 - 'F': Field Goal,
 - 'U': Punt.
- 
- 'K': Kickoff,
 - 'R': Rushing Play,
 - 'P': Passing Play,
 - 'X/F': Extra Point/ Field Goal,
 - 'U': Punt.

Previous

Current

*changes made to increase model performance, since extra-points are essentially a Field Goal from the 3-yard line

Play Type Examples



This is an example of a ***Passing*** play



This is an example of a ***Field Goal***

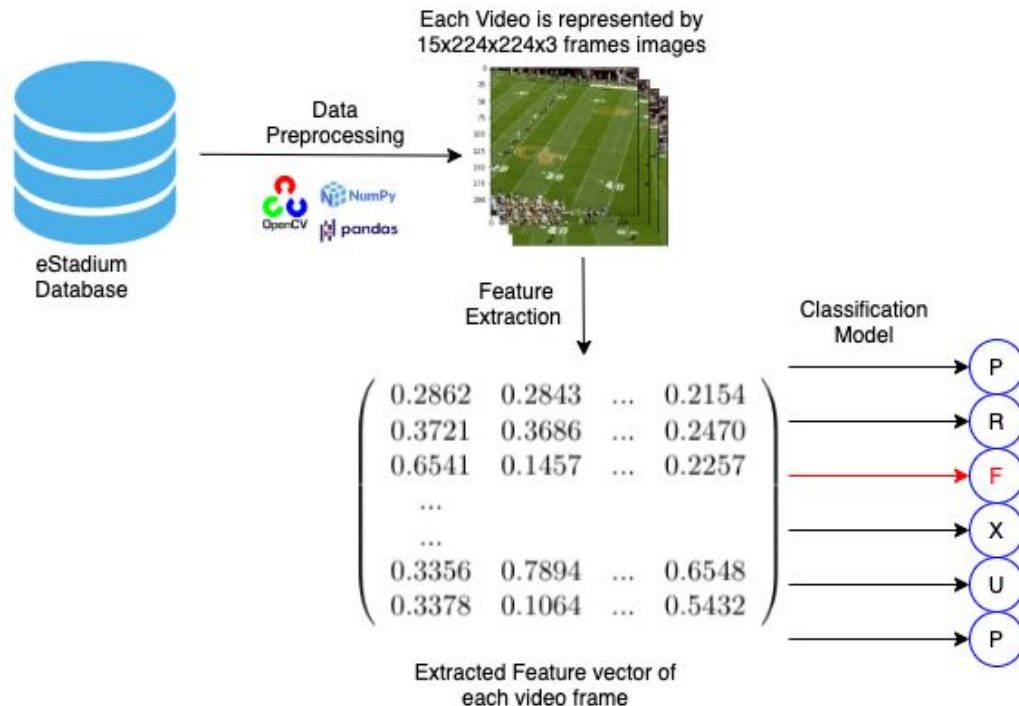
Proposed work



- Data Preprocessing
 - Split videos into static frames
 - Represent each play with “numpy arrays”
 - Labels represented in a one-hot encoding format
- Create and train a CNN-RNN model for image classification automation
 - Use of transfer learning with ResNet50 and VGG16
 - Remove the fully connected layers to output feature vectors
 - Feed plays as sequence of feature vectors into an LSTM network
 - Fully connected output layer classes match the number of play types

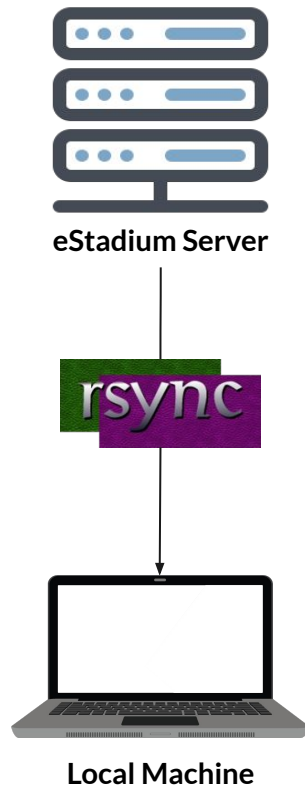
ML Classification Models

- ❖ Obtain Data from the eStadium database
- ❖ Preprocess the Data using OpenCV, Numpy
- ❖ Input preprocessed data into feature extraction models
- ❖ Input extracted feature vector into classification models
- ❖ Output: type of play



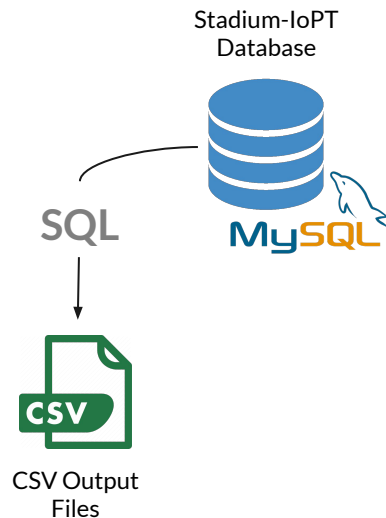
Model Training Process—Video Dataset

- **Description:**
 - 8522 videos
 - Size: ~32 GBs
 - Format: mp4
 - Length: most videos range from ~7 - 15 seconds
- **Acquisition Method:**
 - The videos were imported from the Stadium-loPT server using the *rsync* file synchronization utility.



Model Training Process—Play Description Dataset

- This dataset contains:
- Relevant Columns:
 - Video Path/File Name
 - Has Ball (Home or Visitor)
 - Spot on the Field (in Yards)
 - **Play Type:** Target Variable for Classification
- Acquisition Method:
 - This dataset was queried from the Stadium-IoPT *MySQL* database and exported as *csv* files.

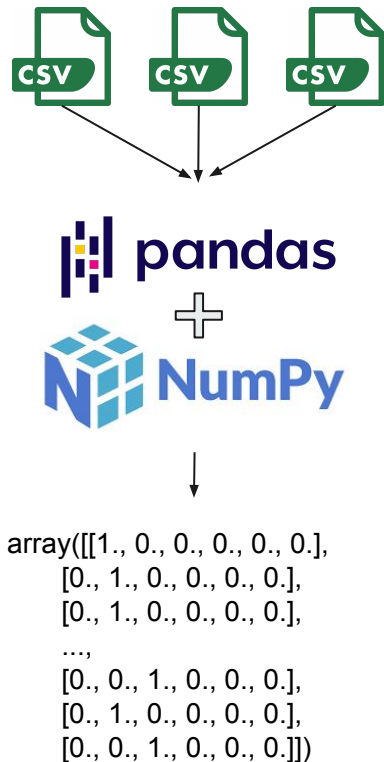


	ID	Down	ToGo	Spot	Text	Quarter	HasBall	Type	VideoFileName	VideoPath	PlayID	GameID
0	11	2.0	3	H24	2nd & 3 at OSU24: Saine, Brandon rush for 11 y...	1	V	R	000	2009/G3/Q1/	11	3
1	11	2.0	3	H24	2nd & 3 at OSU24: Saine, Brandon rush for 11 y...	1	V	R	003	2009/G3/Q1/	11	3

Model Training Process—Data Pre-processing

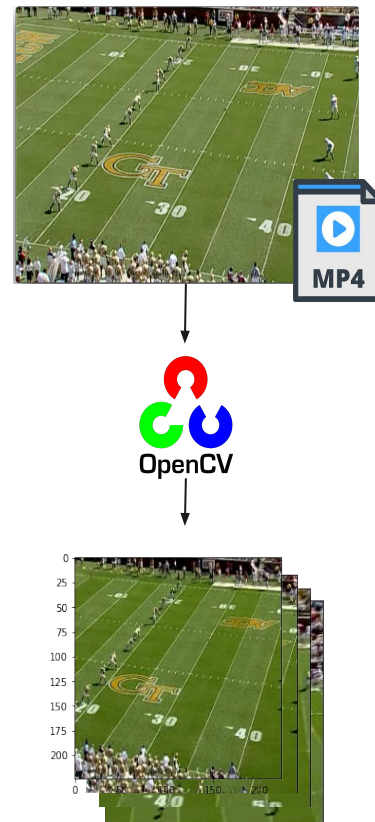
- Training Labels:

- Loaded the csv files as *Pandas DataFrames*.
- Merged the different DataFrames.
- Dropped the rows containing plays of undesired types (Penalties, Start of Game/Quarter, etc).
- Converted the label vector to **one-hot encoding** format.

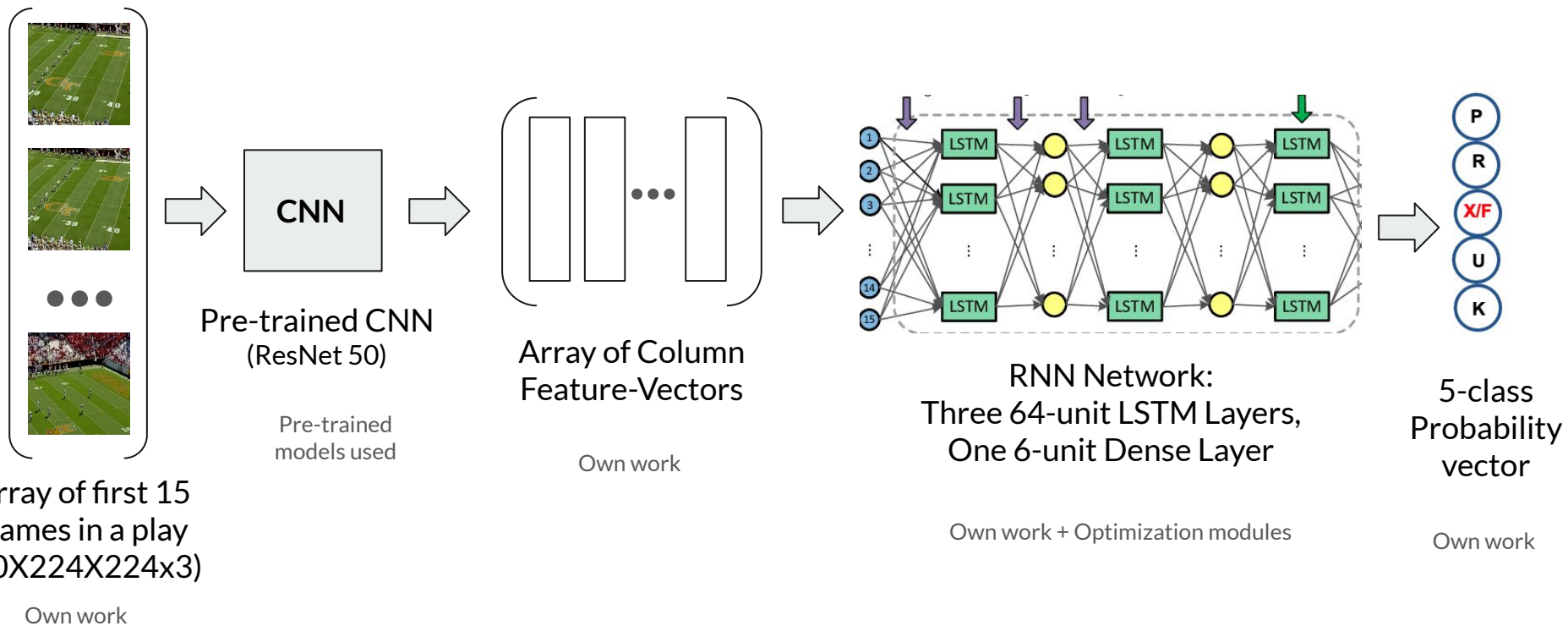


Model Training Process—Data Pre-processing

- Training Samples:
 - Used **OpenCV** to split videos into static frames
 - Frame size: 224x224
 - 2 frames per second
 - Final Dataset:
 - 5,667 Plays
 - 134,310 Frames - 3.4 GBs
 - Saved the entire dataset as a *numpy array*

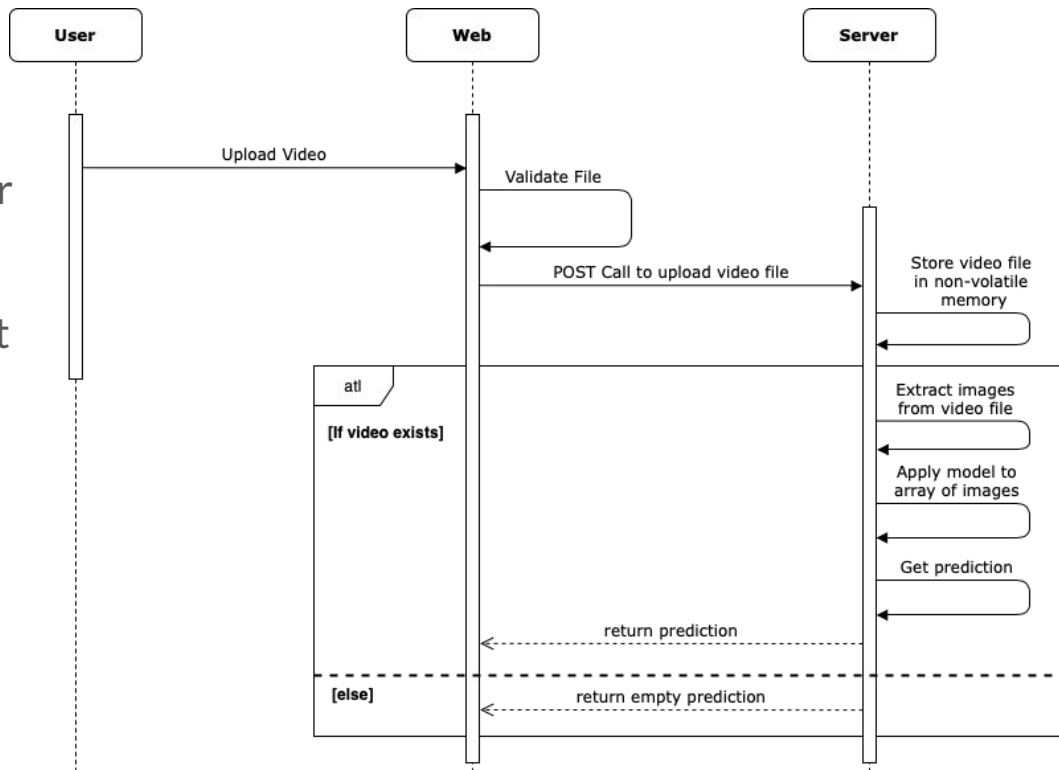


CNN-RNN Model Architecture



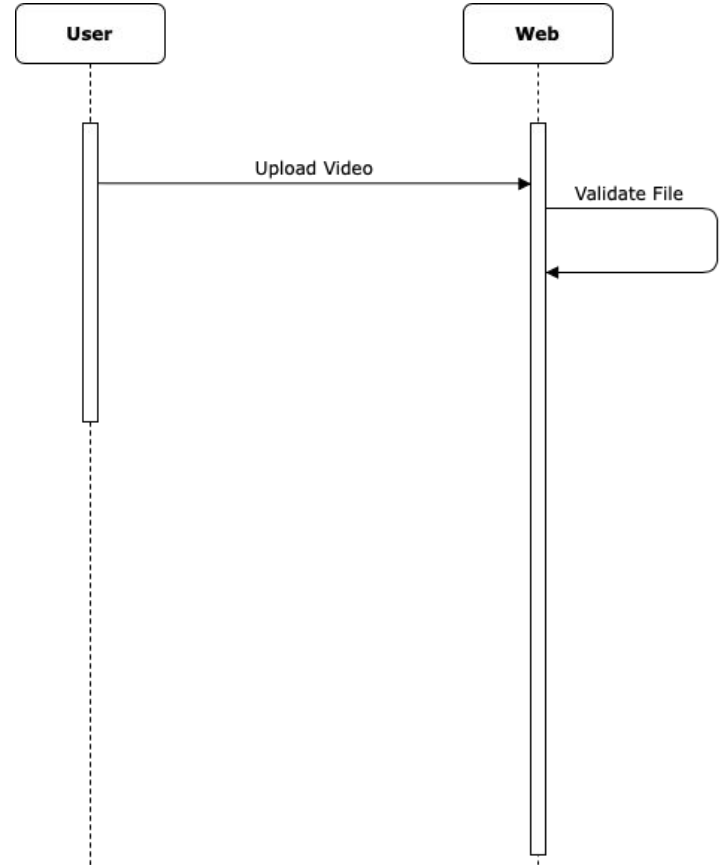
Use Case of Prototype—Using System-Level Sequence Diagram

- A representation of the interaction between the user and the model
- Only one endpoint use to get the prediction needed after the video is uploaded



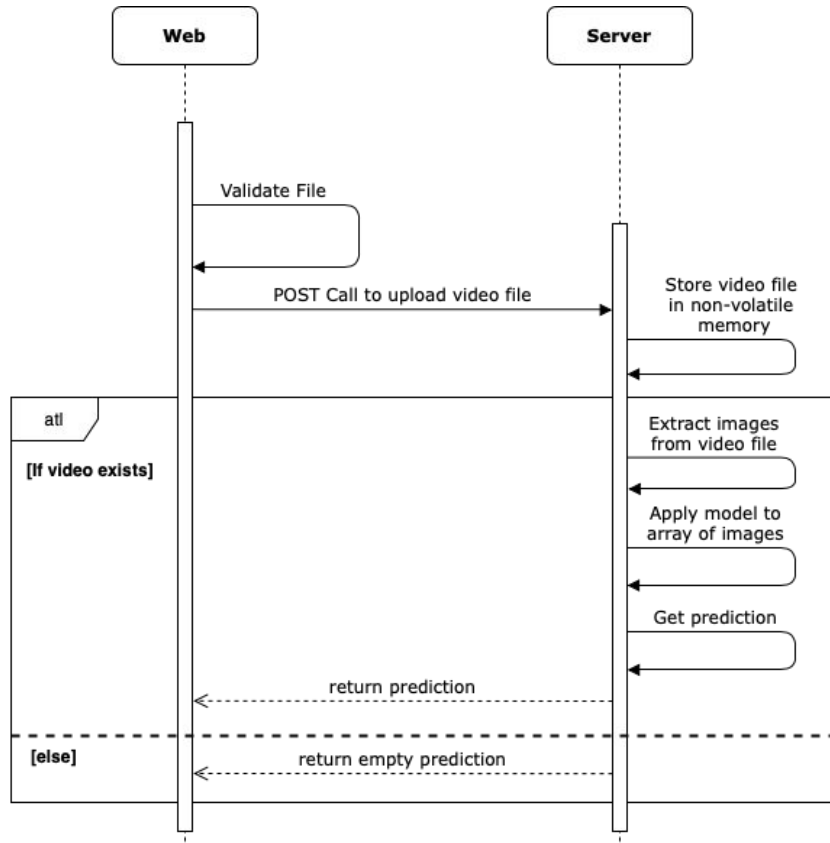
Use Case of Prototype—User and Web

- The user simply uploads the video play to get the prediction.
- The web will determine, internally, if the video file is correct.



Use Case of Prototype—Web and Server

- After the validation is completed, the web will make a POST call to the server to deliver the video file.
- The video file is then stored in a non-volatile memory.
- If the video file exists, an image array from the frame is extracted from the video.
- After generating the array of images, the model is applied to this array.
- Once a prediction is generated from the model, this prediction is sent to the web browser.



Prototype—Deployment



- Project is deployed on <http://54.221.66.111:5000/>
- Amazon Elastic Compute Cloud (EC2)
 - T2.Medium Instance Type (Real-time video processing with Open-CV)
 - 4MB RAM
 - Low-to-moderate Network Performance
 - Ubuntu 18.4 - amd64 server
 - 20GB SSD
- Running a flask server on *tmux* session
 - `flask run --host=0.0.0.0 --port=5000`
- Measurements on live host
 - Request/Response Latency: **10.4s on 8MB high-resolution mp4 video**
 - Request Sent with Video Upload: 6.5s
 - Server side: 3.7s
 - Video input pre-processing: 0.9s
 - Getting prediction: 2.4s

Prototype—Demo

The screenshot shows a web browser window with the title "Football Video Classification". The address bar shows "Not Secure | 54.221.66.111:5000/uploader". The page content is divided into two main sections: "Automatic Classification of American Football Plays" and "Your Video Prediction".

Automatic Classification of American Football Plays

Upload your Video

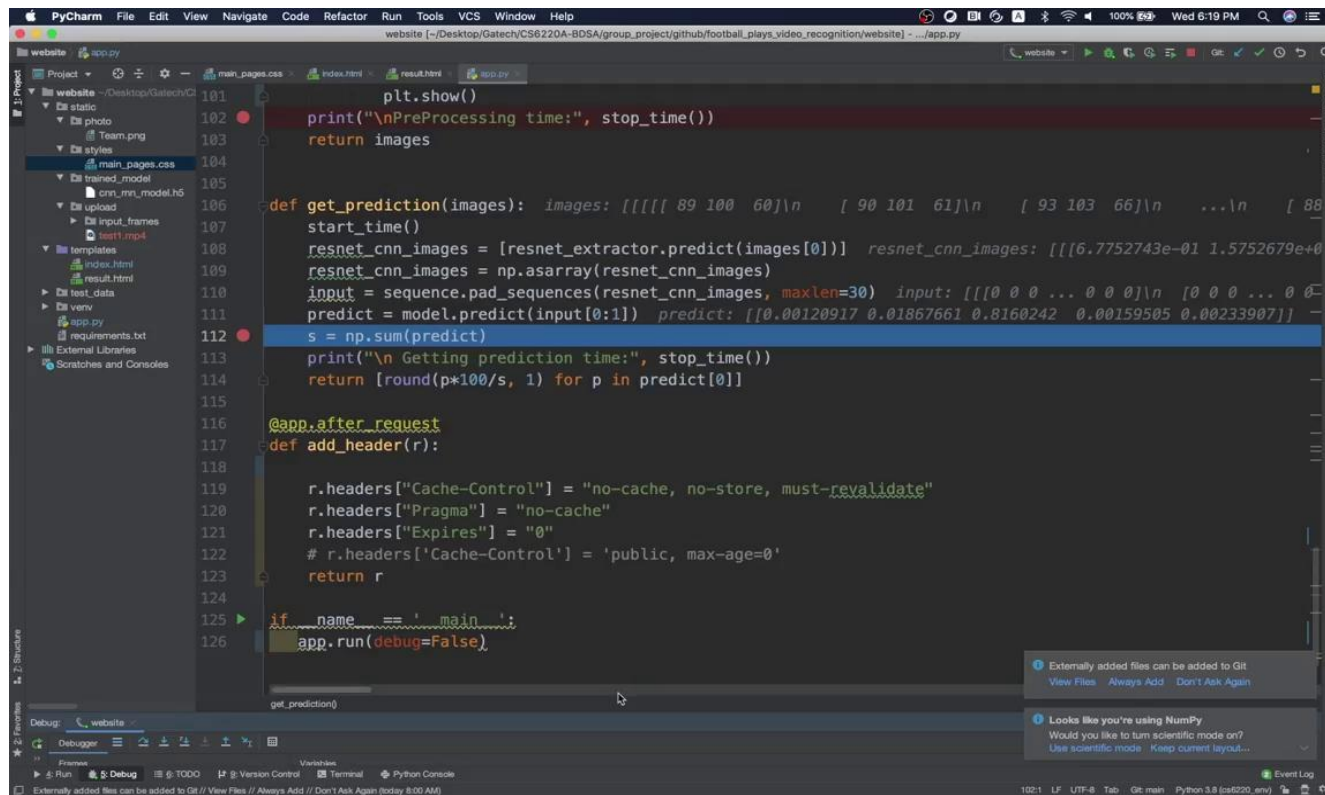
No file chosen

Your Video Prediction

A video player shows a football play. Below the video is a table with the following data:

Types of Play	Prediction
Kickoff	0.1%
Rushing Play	2.2%
Passing Play	97.2%
Extra Point	0.2%
Punt	0.3%

Code Walkthrough—Website



The image shows a PyCharm IDE window with a project named 'website'. The code is a Flask application that uses a pre-trained ResNet model for image classification. The code is as follows:

```
101 plt.show()
102 print("\nPreProcessing time:", stop_time())
103 return images
104
105
106 def get_prediction(images): images: [[[[[ 89 100 60]\n [ 90 101 61]\n [ 93 103 66]\n ... \n [ 88
107 start_time()
108 resnet_cnn_images = [resnet_extractor.predict(images[0])] resnet_cnn_images: [[[6.7752743e-01 1.5752679e+0
109 resnet_cnn_images = np.asarray(resnet_cnn_images)
110 input = sequence.pad_sequences(resnet_cnn_images, maxlen=30) input: [[0 0 0 ... 0 0 0]\n [0 0 0 ... 0 0
111 predict = model.predict(input[0:1]) predict: [[0.00120917 0.01867661 0.8160242 0.00159505 0.00233907]]
112 s = np.sum(predict)
113 print("\n Getting prediction time:", stop_time())
114 return [round(p*100/s, 1) for p in predict[0]]
115
116 @app.after_request
117 def add_header(r):
118
119     r.headers["Cache-Control"] = "no-cache, no-store, must-revalidate"
120     r.headers["Pragma"] = "no-cache"
121     r.headers["Expires"] = "0"
122     # r.headers['Cache-Control'] = 'public, max-age=0'
123     return r
124
125 if __name__ == '__main__':
126     app.run(debug=False)
```

The code is a Flask application that uses a pre-trained ResNet model for image classification. The code is as follows:

```
plt.show()
print("\nPreProcessing time:", stop_time())
return images

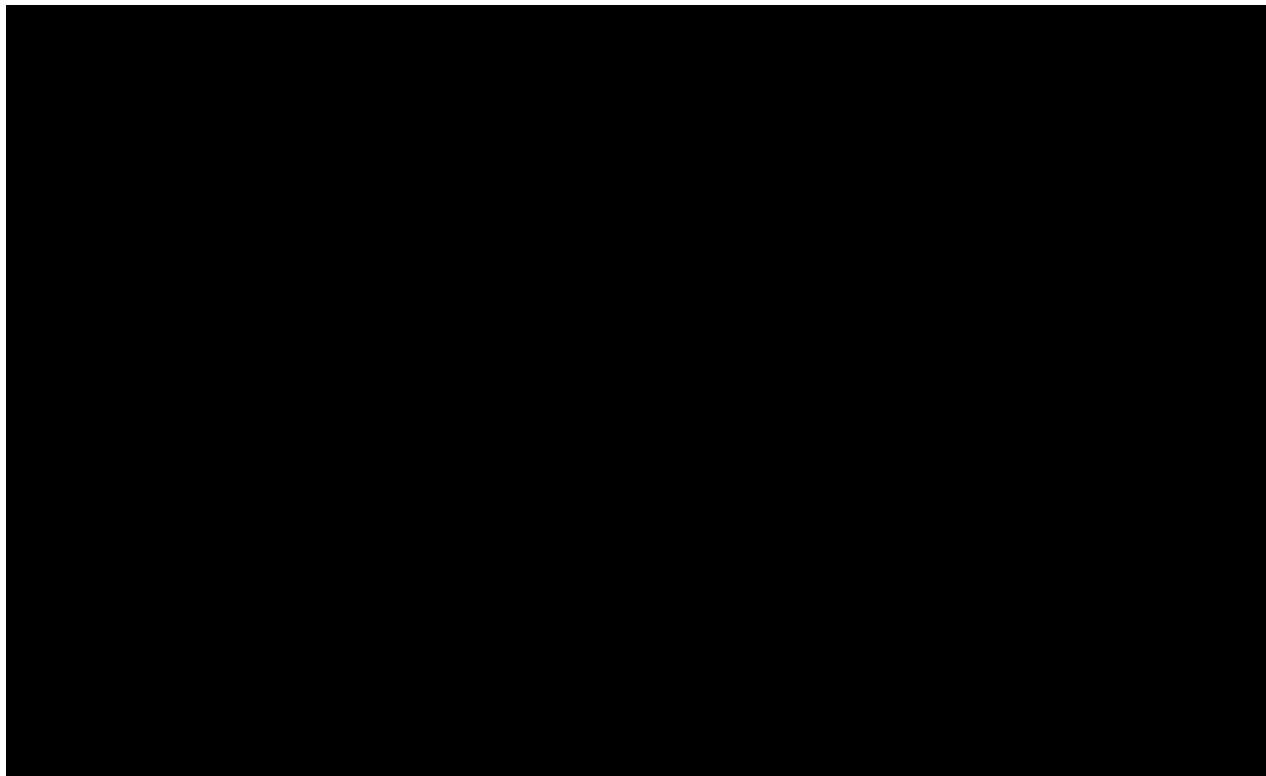
def get_prediction(images):
    images: [[[[[ 89 100 60]\n [ 90 101 61]\n [ 93 103 66]\n ... \n [ 88
    start_time()
    resnet_cnn_images = [resnet_extractor.predict(images[0])]
    resnet_cnn_images: [[[6.7752743e-01 1.5752679e+0
    resnet_cnn_images = np.asarray(resnet_cnn_images)
    input = sequence.pad_sequences(resnet_cnn_images, maxlen=30)
    input: [[0 0 0 ... 0 0 0]\n [0 0 0 ... 0 0
    predict = model.predict(input[0:1])
    predict: [[0.00120917 0.01867661 0.8160242 0.00159505 0.00233907]]
    s = np.sum(predict)
    print("\n Getting prediction time:", stop_time())
    return [round(p*100/s, 1) for p in predict[0]]

@app.after_request
def add_header(r):

    r.headers["Cache-Control"] = "no-cache, no-store, must-revalidate"
    r.headers["Pragma"] = "no-cache"
    r.headers["Expires"] = "0"
    # r.headers['Cache-Control'] = 'public, max-age=0'
    return r

if __name__ == '__main__':
    app.run(debug=False)
```

Code Walkthrough—Model



CNN-RNN Model

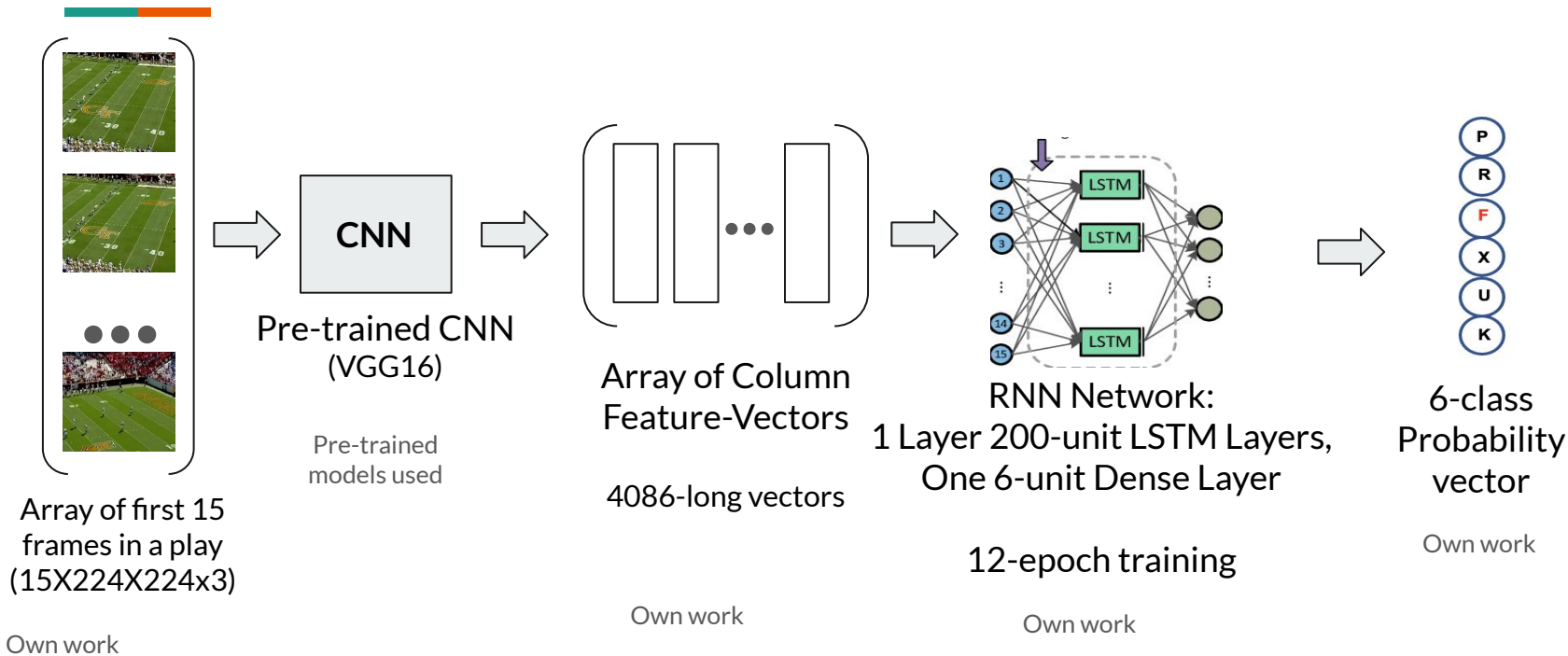
The idea behind this model was to use a pre-trained Convolutional Neural Network (CNN) to extract features from the static frames, and build our own Recurrent Neural Network (RNN) to learn patterns from sequences of feature-vectors output by the CNN grouped as individual plays.

Technology stack:

- **Programming Language:** *Python 3.6.10*
- **Numerical Computing Library:** *NumPy 1.17.0*
- **Data Manipulation Library:** *Pandas 1.0.5*
- **Video processing library:** *OpenCV 4.4.0*
- **Deep Learning API:** *Keras 2.3.1*
 - **Backend Library:** *TensorFlow 1.14.0*

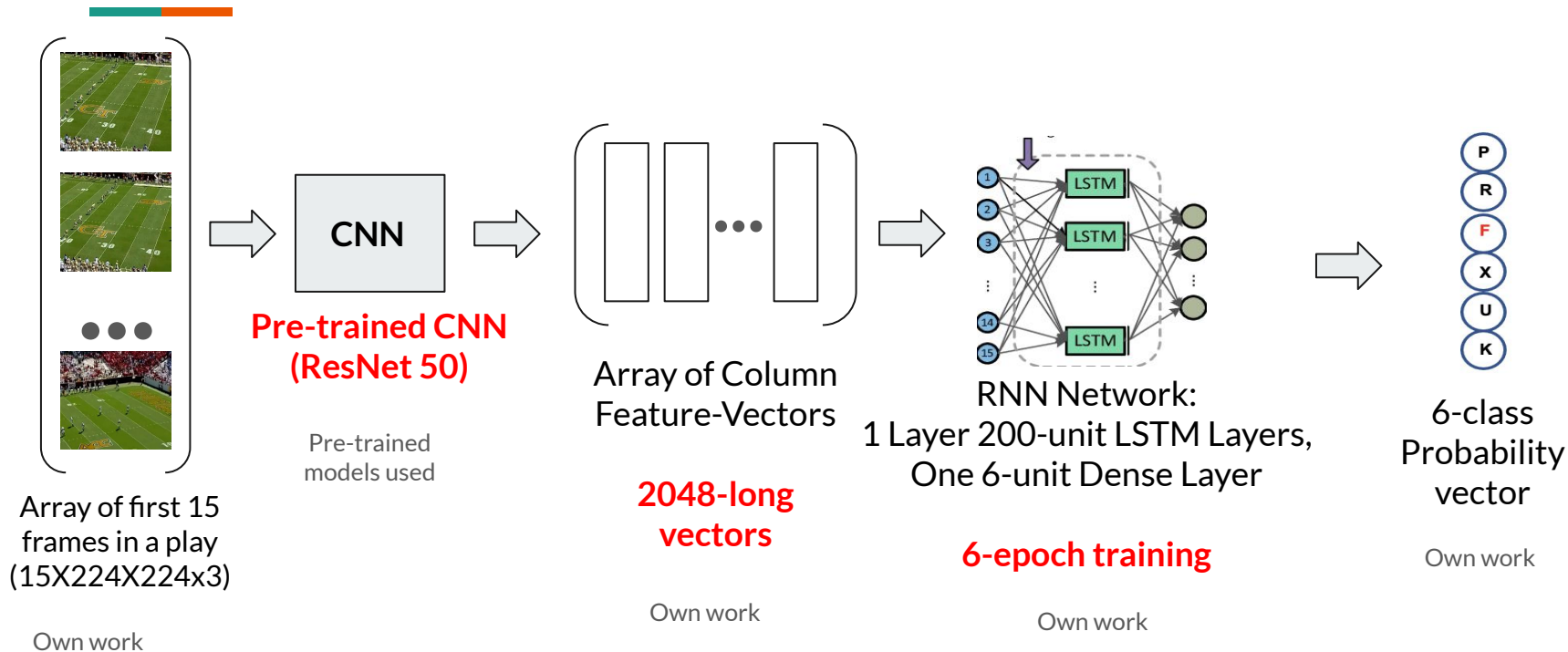


First Model Architecture - VGG16 + (200)LSTM



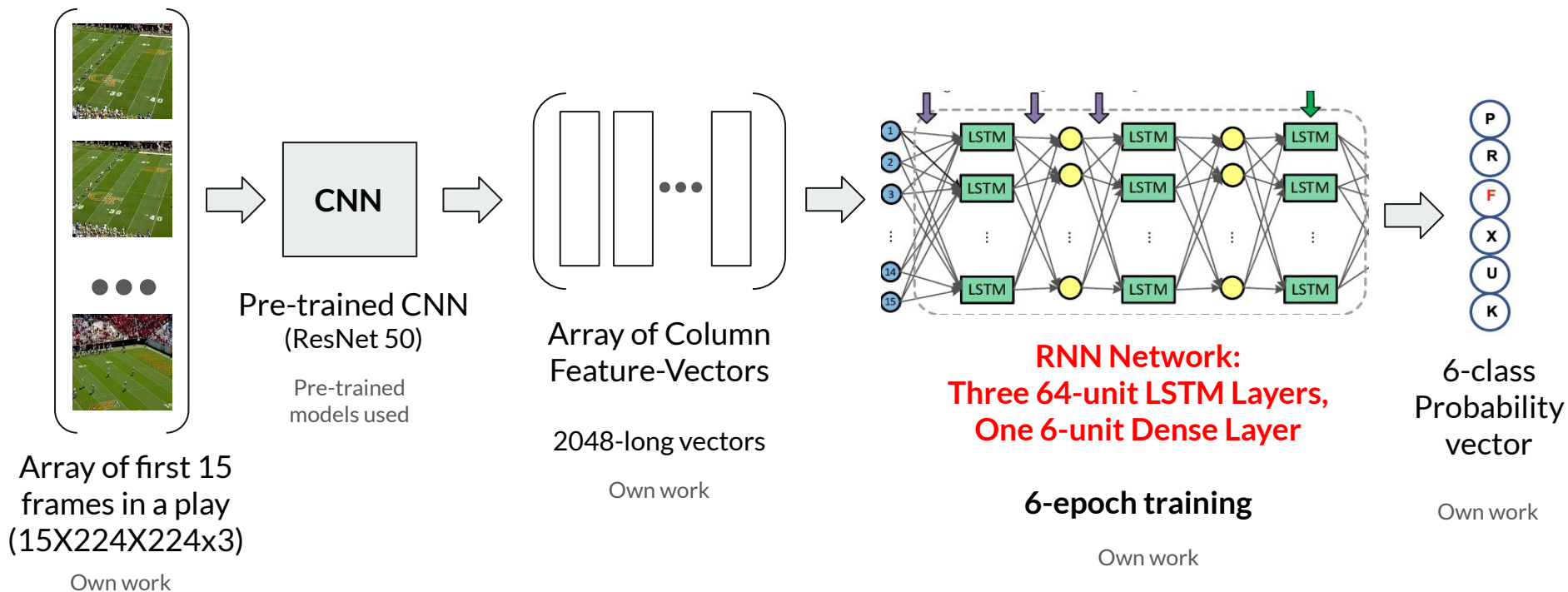
Test Accuracy: 71.78%

Second Model Architecture - ResNet50 + (200)LSTM



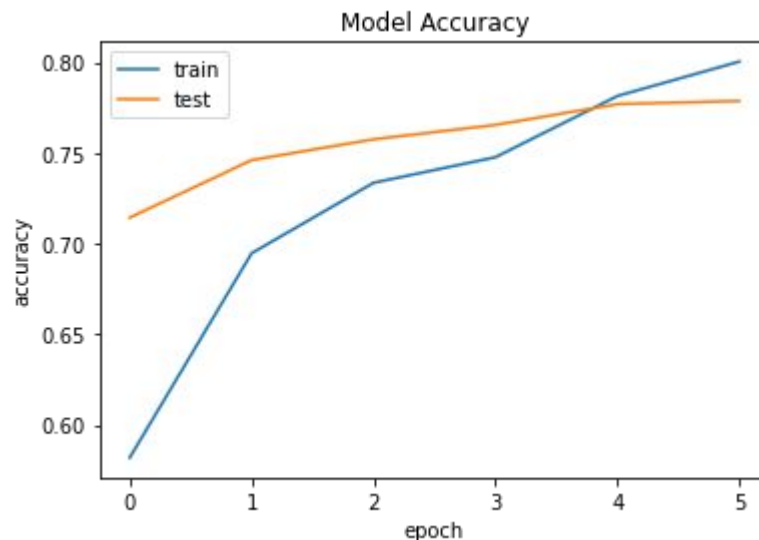
Test Accuracy: **74.78%**

Third Model Architecture - ResNet50 + (3X64)LSTM



Third Model Results

- All play arrays padded to **15 frames**
- Using the feature vectors from **Resnet50** resulted in better accuracy results then VGG16 . It also requires less memory since they have 2048 features compared to 4096.
- Train/Test Split: 80/20
- Testing Accuracy:
 - Total: 77.87%
 - Rush: 88.25 %
 - Pass: 75.70 %
 - Extra Point: 72.73 %
 - Field Goal: 38.39 %
 - Kickoff: 55.88 %
 - Punt: 22.95 %



Third Model Error Analysis



- Most “Field Goals” were incorrectly labeled as “Extra Points”.
 - Could possibly combine both into one class and build a secondary model
- Most incorrectly labeled “Passing” plays were labeled as “Rushes”.
- Some particularly long passes were incorrectly labeled as “Kickoffs”.
- The model performs very poorly with “Punts”.
- Some videos in the dataset were recorded with a camera. The low quality resulted in all those plays being labeled as “Rushes”.
- Potential Biases in our Dataset:
 - All videos are from Bobby Dodd Stadium.
 - Georgia Tech tends to opt for Rushing plays more often than most teams.
 - Our videos do not contain the scoreboard seen on TV.

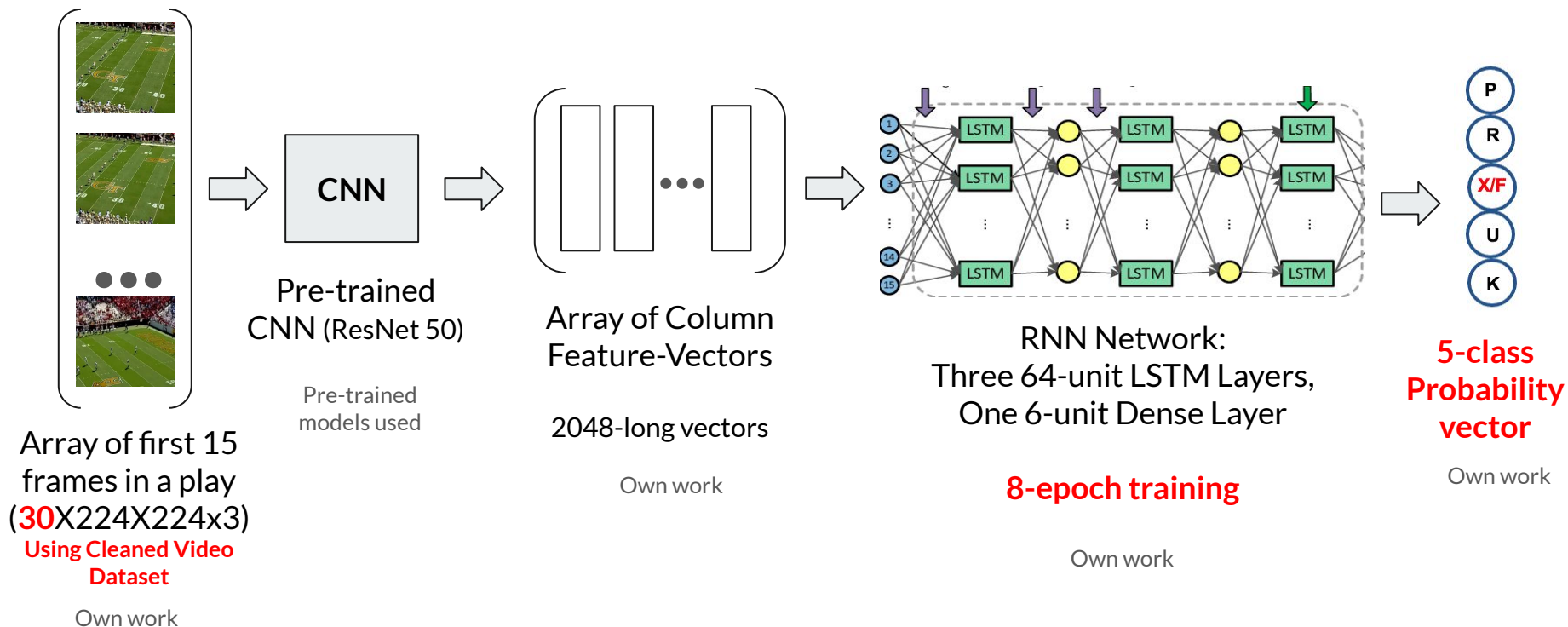
Optimizations to the Third Model

- All play arrays padded now to **30 frames**.
- Increased from 6 to **8 epochs**.
- Continued to use **Resnet50** which continued to result in better accuracy results. It also required less memory (2048).
- Transformed **Extra Point** and **Field Goal** in the same category.
- Removed games that were recorded with poor resolution and from far away (2017 season).



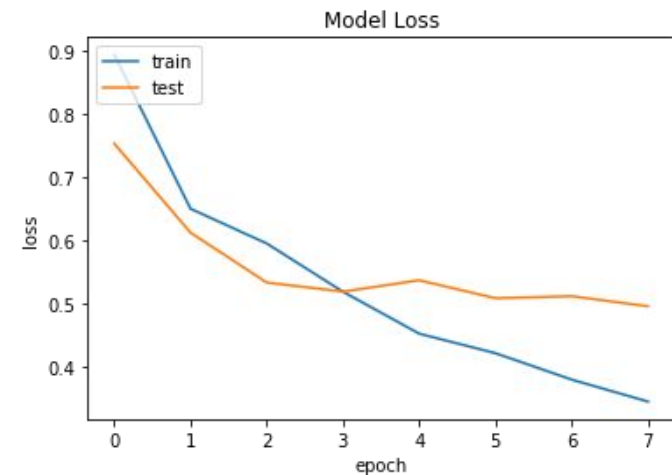
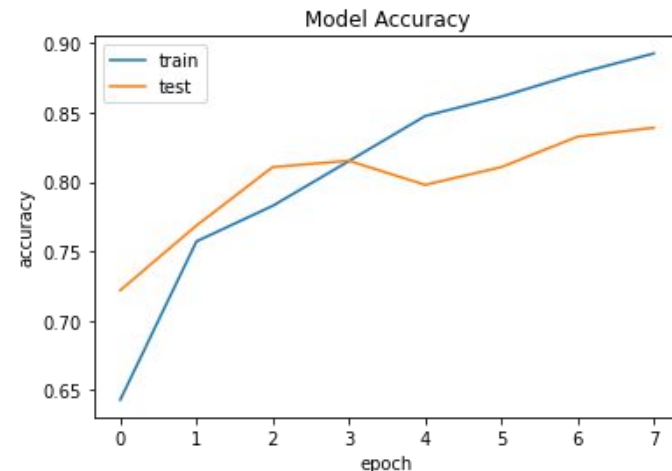
Example of low-quality video from the 2017 season.

Final Model Architecture - ResNet50 + (3X64)LSTM + Improvements



Final Model Results

- Train/Test Split: 80/20
- Testing Accuracy:
 - Total: 83.90% (6.3% increase)
 - Rush: 87.95%
 - Pass: 81.82% (6.1% increase)
 - Extra Point/FG: 89.47% (34% increase)
 - Kickoff: 83.58% (27.7% increase)
 - Punt: 55.56% (32.6% increase)



Comparison Results



System Specifications: macOS Catalina (10.15.6), 2 GHz Quad-Core Intel Core i5 (10th Generation), 16 GB RAM, 500 GB SSD

Model	Accuracy	Training Time	Prediction Time
1. VGG-16 + ₍₂₀₀₎ LSTM _(initial)	71.78%	3 h 57min* + ~55sec**	~2.57 ms
2. ResNet50 + ₍₂₀₀₎ LSTM _(1st iteration)	74.78%	1 h 56min* + ~57sec**	~3.02 ms
3. ResNet50 + _(3X64) LSTM _(2nd iteration)	77.87%	1 h 56min* + ~37sec**	~4.27 ms
4. ResNet50 + _(3X64) LSTM _(final)	83.90%	1 h 56min* + ~93 sec**	~7.32 ms

* This is the time that it takes to process all the static frames through the CNN to obtain the feature vectors to be inputted to the LSTM network (this process only needs to be performed once)

** This is the time that it takes to train the LSTM network with 80% of the training data

Different Difficulty Cases

- Easy Case:

Rushing Play @ GT Stadium



- Challenging Case:

Punts @ GT Stadium



- Hard Case:

Plays from other stadiums, e.g.
Boise State



Future work



- CNN-RNN Model:
 - Gather new datasets from other schools and other teams
 - Test our model against new video datasets and compare performance
- Possible implementation of HMMs to improve the model:
 - Considers feature vectors statistically dependent, a Markov Mesh
 - Two-Dimensional transition probabilities conditioned on the states of neighboring blocks
 - HMM parameters estimated by EM algorithm
- Web application:
 - Collect feedback and improve current application
 - Advertise for teams to use and evaluate performance in real-time

Deliverables



- Project's Prototype: <http://54.221.66.111:5000/>
- Demo Video:
<https://drive.google.com/drive/folders/1jrTH6goCk2LdTeCqENvkFtggAZpF66Zm?usp=sharing>
- Final Report:
<https://docs.google.com/presentation/d/17E08CT51vZASu4Ge4pzX7h86cAs7u2bDj2hUwY9ZDmo/edit?usp=sharing>
- Source Code: https://github.com/phpinto/football_plays_video_recognition

(please let us know if any permission is needed)