Football **A**nalysis in **R**eal **T**ime using **T**ransformer

Tony Robinson and Tanner Watts

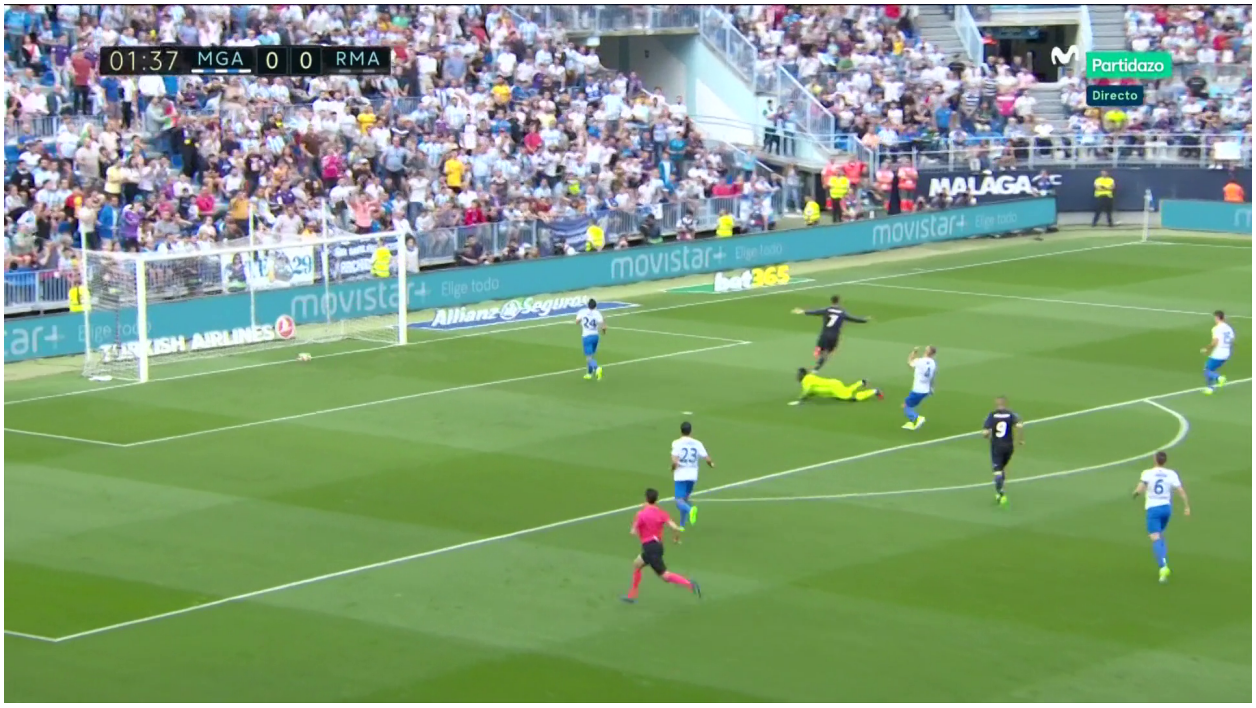
The Problem: Understanding and analyzing soccer games for key moment annotations and outcome prediction.

Our Implemented Solution vs Our Proposed Solution: We proposed a system that can watch, analyze, and annotate videos of soccer games. We wanted the analysis to produce a transcript of what happens throughout the game (i.e. goal scored at X timestamp, throw in at Y timestamp) using a state-of-the-art visual transformer. Unfortunately, we were unable to fully realize our goal. We spent the first two and a half weeks of our project putting many hours into research and debugging. This included figuring out the best frameworks, cleaning the very messy data, and learning all of the nuances of the Transformer world. Because of the large amount of time that we ended up spending on the many problems we encountered (understanding and being able to manipulate the entire dataset, getting the video transformer to work, and RAM requirements in our google colab), we decided to make a calculated pivot to a similar, but a slightly more computationally manageable solution.

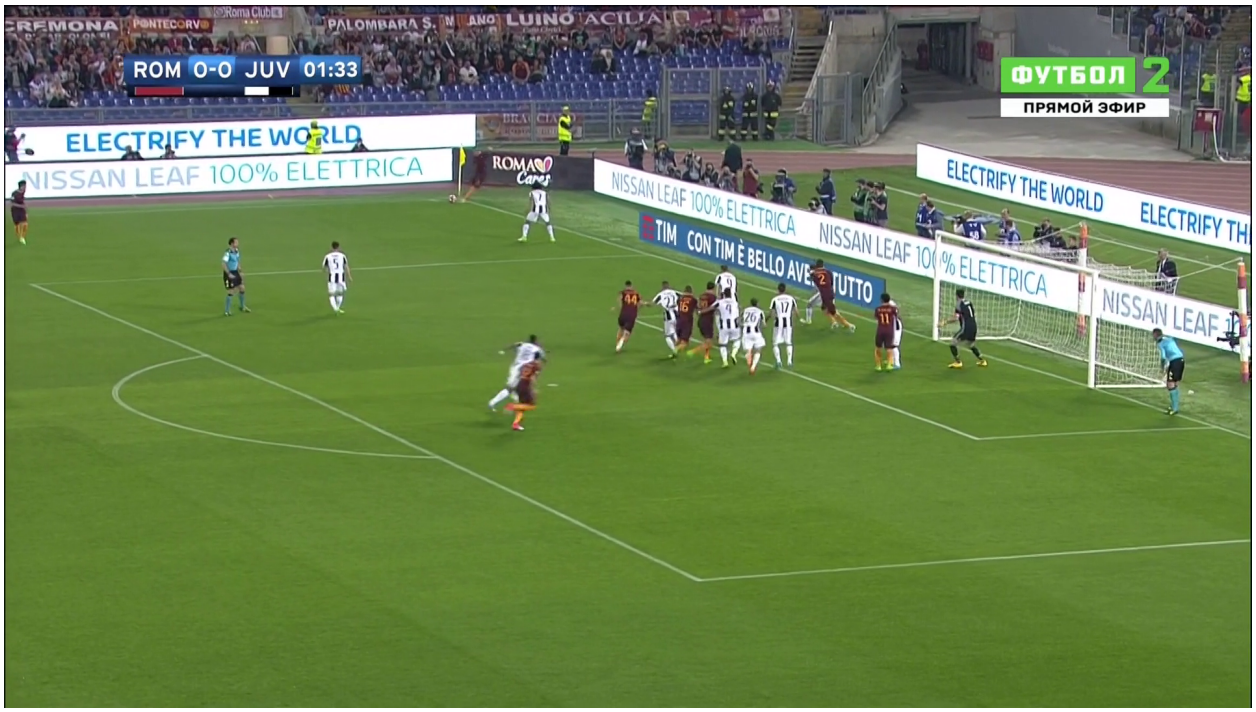
The Pivot: Instead of building a transformer to analyze a video of an entire game, we built a transformer to do image classification of “key moments”. With this, we could run inference on video frames. We also wanted to see how this transformer would compare to a high-performing CNN, so we also implemented a ResNet to compare the results of classifying these 10 classes:

```
['Foul', 'Goal', 'Ball out of play', 'Direct free-kick',
'Clearance', 'Corner', 'Offside', 'Penalty', 'Yellow card',
'Kick-off']
```

Goal:



Corner Kick



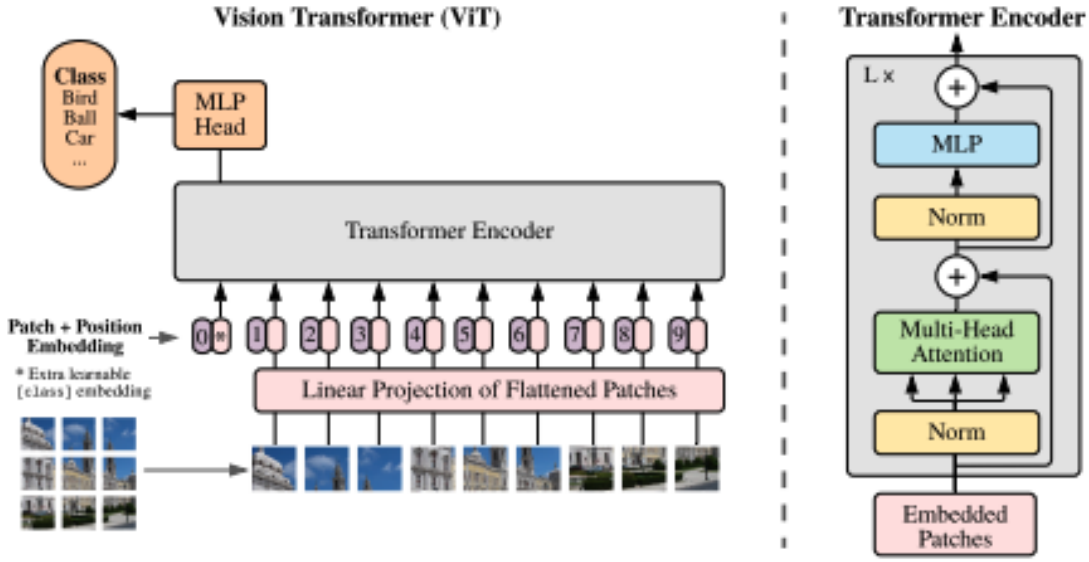
Dataset:

SoccerNet-v2 is the dataset we will use for this project. It constitutes the most inclusive dataset for soccer video understanding and production, with ~300k annotations, 3 computer vision tasks, and multiple benchmark results.

Data Preparation:

We used this project as an opportunity to learn about industry standard frameworks. After shopping around, we decided to use PyTorch lightning. To do this, we had to learn about and implement PyTorch data modules and data loaders. The final result was that we had a setup where we could simply plug the two different models into the same dataset. This setup would be amazing for further projects as the modularity allows for easy experimentation. Pytorch Lighting also allows for the use of loggers which help with making interesting graphs regarding training specifics. We used the wandb logger to produce the nice graphs that we present in our results section.

Model used:



VIT(Vision Transformer): *An Image is Worth 16x16 Words* was the paper that inspired our transformer model. ViT is a transformer-based architecture that is applied directly to images for image classification tasks. It works by first dividing an input image into a grid of patches and then representing each patch as a linear embedding. The patches are

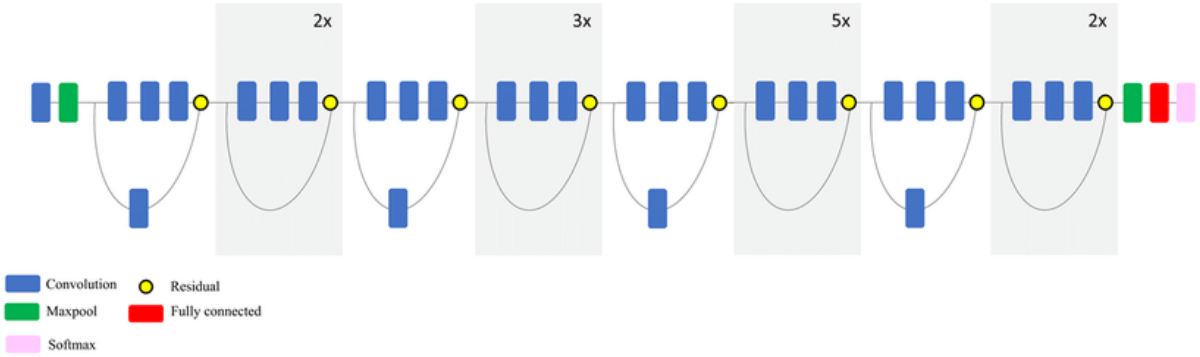
created in the `PatchEmbeddings` class by using a Conv2d layer with a non-overlapping stride and patch_size. After the patches are created, positional embeddings are built of these patches in the `PositionalEmbeddings` class. These encodings are generated using nn.Parameter, and make up the majority of the size of the model. In fact, the embeddings alone have 1.3 million trainable parameters. These patch embeddings are treated as tokens and input into the transformer architecture as a sequence.

After the patch + position embeddings are created they enter into a transformer encoder model. The transformer encoder model consists of 4 classic transformer layers and then a LayerNorm at the end. The classic transformer layers come straight from the original paper and consist of `MultiHeadAttention` which is then passed through a feedforward network. The self-attention mechanism allows the model to consider the relationships between different patches in the image and weigh their importance for the task at hand. The feedforward network is used to transform the patch embeddings and further process the information.

The final layer of the transformer produces a prediction for the image classification task by performing classification on the weighted patch embeddings. This is done in our `ClassificationHead` class. The classification scores are calculated by simply passing the embeddings through a linear layer, a ReLU activation, and another linear layer with out_channels being the num classes.

Due to the sheer size of this model, training was challenging. Even with Colab pro + we had many issues with running out of GPU storage which caused a major delay in the training process. After downsizing the model and the images we were able to get the model to train.

ResNet:
We implemented the same ResNet we learned about in class and implemented it in the homework. Here is the model architecture. We used ResNet 32 for our training.



Results:

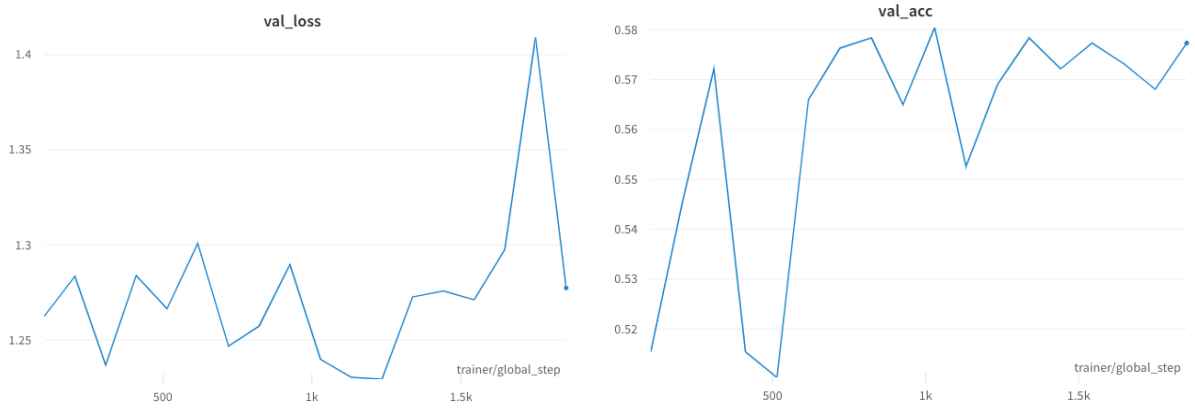
ResNet

Test metric	DataLoader 0
test_acc:acc	0.40934282541275024
test_loss:loss	1.6682822704315186

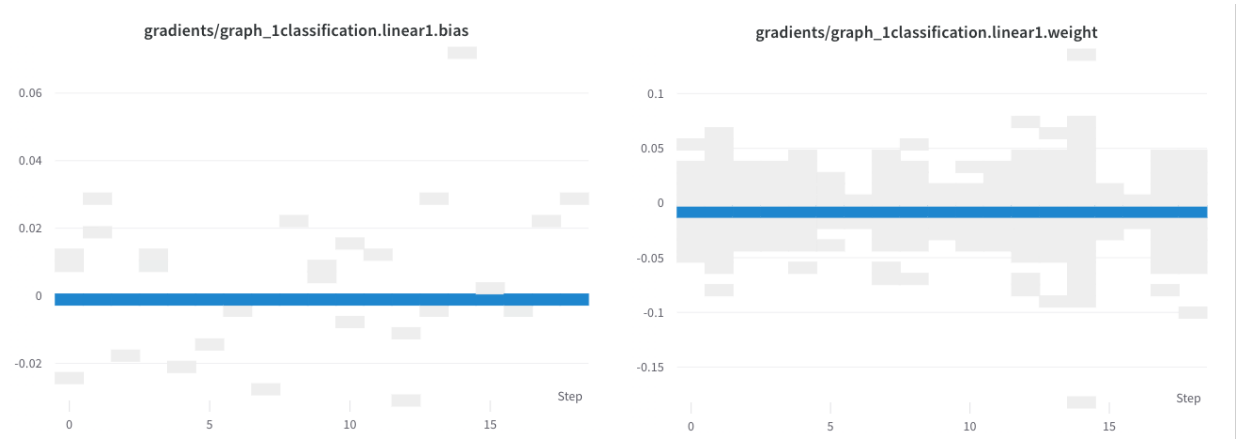
Transformer

Test metric	DataLoader 0
test_acc	0.5336500406265259
test_loss	1.376904010772705

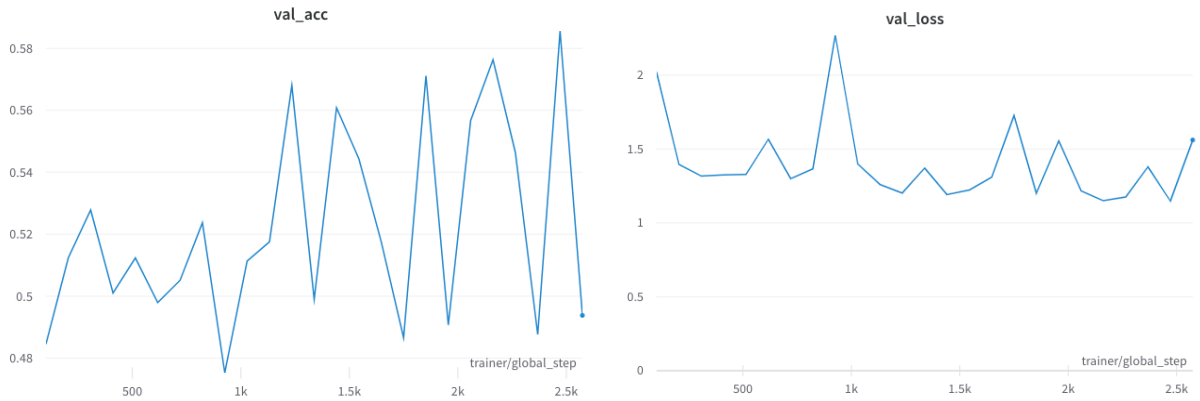
Transformer 25 Epoch Run



Transformer Classification Weight and Bias Gradients



ResNet-32 25 Epoch Run



In summary, our results show that after 25 epochs, the accuracy of the ResNet on the Test Data is 40.9%, and the accuracy of the Transformer on the Test Data is 53.3%.

Example of training for 120 epochs (transformer) val_acc= 0.322

```
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used: True
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU cores
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False, using: 0 IPUs
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPUs
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
INFO:pytorch_lightning.callbacks.model_summary:
  | Name           | Type                  | Params
-----------------------------------------------------------
0 | patch_emb      | PatchEmbeddings       | 12.5 K
1 | pos_emb        | PositionalEmbeddings  | 1.3 M
2 | classification | ClassificationHead    | 139 K
3 | ln             | LayerNorm             | 512
4 | compute_loss   | CrossEntropyLoss      | 0
-----------------------------------------------------------
1.4 M     Trainable params
0         Non-trainable params
1.4 M     Total params
5.732     Total estimated model params size (MB)
```

Epoch 119: 100% 172/172 [05:50<00:00, 2.04s/it, loss=1.87, v_num=4, val_loss=1.870, val_acc=0.322]

Interpretation:
The results show that the Transformer showed a 10% boost in scores over the ResNet. While our hypothesis was that the transformer would outperform the ResNet by a mile, our results show that this is not entirely the case.

Accuracies for both models were not as high as we might expect. While the results are far better from the models randomly guessing, they still perform far worse than other models. This could be attributed to several different factors:

1. The first reason is that our image data set is not as full as we would like it to be. While there were plenty of videos in the SoccerNet dataset, there were far fewer labeled images. Had there been more images in the dataset, we likely would have seen improved results.
2. We had a different number of images in each class. To improve results, we would like to use a dataset that contained an equal number of pictures in each class.
3. The second reason for our poor performance is that we ran into time constraints. Due to google colab, it was tough to train all in one place. With colab pro + (we trained 120 epochs over 12 hours) until the runtime tapped out. Even with checkpoints, it was hard to start training again, in fact, we saw a decrease in accuracy every time we retrained from a checkpoint. The temporary fix was to retrain in less time (25 epochs). Should we have had more reliable computational sources and a week to train, we would likely have seen better results.

What We Learned:

- How to use an open-source dataset to solve a problem
- The specifics of the visual transformer architecture (video transformers and image transformers)
- The PyTorch lightning framework
- Don’t bite off more than you can chew

Next steps if we had more time:

- We only trained for 12 hours with 120 epochs. If we had more time, we would want to train for a whole week. For this, we would need more powerful computers.
- We would consider labeling more images ourselves to make our dataset richer.
- We would figure out how to leverage pre-trained weights for our transformer model.

References:

[1] Dataset Paper: https://arxiv.org/pdf/2011.13367v3.pdf

[2] Video Transformer: https://arxiv.org/pdf/2103.15691.pdf

[3] Visual Transformer: https://arxiv.org/pdf/2010.11929.pdf

[4] Data Set(link): https://www.soccer-net.org/data