Auto-DRS: Classifying Run-outs in Cricket

1 CONTENTS

2 INTRODUCTION

2.1 BACKGROUND AND CONSTRAINTS

Cricket is a team sport popular in many parts of the world. It consists of two teams, eleven-a-side, where the teams alternate between batting and bowling rounds. There are many parallels with baseball, where the batsmen want to hit the ball as far as they can. There have been previous attempts, like Crick-Net [1] for automating detection for some rules of the game. Among other rules which don't yet have a Computer Vision solution, there is a significant rule where after hitting the ball, the batsmen run between the wickets (stumps) and the bowling team tries to get them out by hitting the stumps before the batsmen cross the crease. If the bowling team succeeds in doing so, it is known as a "run-out".

However, in a fast-paced game, the on-field umpires cannot always tell if it was *out* or *not-out*. As a result, competitions employ a Decision-Review System (DRS), also known as a Third Umpire, that consists of someone manually slowing down the clip and making a decision whether it should be called a run-out. This presents an opportunity for automation using Computer Vision, although there are several expectations that such a system must address.

Capture: Currently, the video is recorded using a high-FPS camera so that it can be sufficiently slowed down. However, it is often expensive to constantly record, store and process high-FPS video clips. With the recent advances in Deep Learning, we now have techniques such as Super SloMo [2] to artificially slow-down a video clip using temporal interpolation.

Process: The decision system must be robust to many different types of scenes because there is significant variation in the scenes (colors, number of people, jerseys, viewpoints, etc.). Moreover, the relevant parts of the image may be heavily localized to certain regions. That is, unlike a picture of a dog or cat, the pertinent information here is just the batsmen crossing the crease and stumps being hit i.e. a very small fraction of pixels. The process must thus successfully work within this constraint and avoid overfitting.

Data: To the best of my knowledge, there is no freely available dataset for this problem that is large enough. As a result, I must manually gather and annotate data. This imposes feasibility constraints on the size of the dataset that can be made available. Further, when I was preparing the dataset, I found that the number of instances of *out* were much more frequent than *not-out* on the video clips I gathered from the internet. Consequently, the system must find a way around this severe class imbalance.

Reporting: Cricket is a very emotional sport to many people, and like many other sports, there is a strong necessity for interpretability. The audience must be able to deduce why a certain decision was made. As a result, a successful model should not only make a decision between *out* and *not-out*, but also provide some reasoning e.g. it must localize exactly the deciding frame.

2.2 COMPLEXITIES

In terms of the data, a decision may not always be that straightforward. Firstly, there are many different colors (depending on the teams) and moving bodies. Sometimes, numerically, there's hardly a significant difference between the background and the wickets (top-left image below), and so relevant objects may be indecipherable. There could also be significant occlusions that hide the stumps or the crease or both. Sometimes, the batsmen might cross the crease but still be in the air (top-right image below) and 3d viewpoint must be inferred from cues such as shadows. In some competitions/matches, the stumps light up when they're hit while they don't in others, thus there's significant within-class variation relative to the variation between classes. This poses challenges when modelling these features, especially because it adds further sparsity to the data (we have fewer images of each type). See examples in appendix, Figure 1.

A more obvious challenge arises from the fact that we must process videos and neglecting the temporal dimension completely may not produce great results. How we encode the motion into a classifier efficiently (without significantly increasing the number of parameters) is an added complexity to the problem. Although some of the complex models can learn to handle these inconsistencies, it is interesting to keep a note for this as I'll mention below how some models fail under certain conditions. Keeping these extreme situations in mind helps us better reason about the failures of our models.

2.3 PROBLEM FORMULATION

To work around the imbalance in the dataset, I propose a different formulation of the problem. Instead of classifying video clips as *out* and *not-out* (which is severely imbalanced), we could formulate it as a simultaneous classification problem with two outputs: (1) "stumps hit?" and (2) "batsman in-crease?" because we can get reasonable amounts of positive and negative examples for both. Also, this increases the granularity of our inputs from videos (or clips within videos) to frames, which we have a lot more of. Thus,

going from a coarse-grained problem to fine-grained multi-output classification, we can get a lot more stability from a machine learning point-of-view.

2.4 DATASET

There are two big competitions in Cricket, among others: the ICC World Cup (CWC) and the Indian Premier League (IPL). On their website, they have some clips of sensational run outs and close calls. I made a list of links that show such runouts and used Selenium (a web-scraping tool) to download them all. Then I used OpenCV to split it into individual frames, and manually annotated them. Note that because they usually upload only sensational clips, there's a lot fewer instances of *not-outs* than there are *outs*. Thus, given the problem formulation above, we have 4 classes for the frames:

- 00 (stumps not hit, batsman not in-crease),
- 01 (stumps not hit, batsman in-crease),
- 10 (stumps hit, batsman not in-crease), and
- 11 (stumps hit, batsman in-crease).

Class	Stumps hit?	In-crease?
Positive	442	261
Negative	671	852

A total of 1113 frames were extracted and annotated from 33 video clips. The number of positive and negative samples for each class have been listed in the table above. Note that the class imbalance is still very significant, especially for in-crease. However, both the absolute number of data samples and the ratio of positive to negative samples is significantly better now than having 5 *not-out* clips out of 33, for example.

2.5 METHODOLOGY

Firstly, considering the lack of quality data, traditional Computer Vision approaches may seem more intelligible for this problem. Using some heuristics, perhaps it may be possible to consistently detect items in the scene. It seems sensible to compare this to ML approaches that have few enough parameters to generalize well.

In particular, I look at half of the problem first: detecting whether the stumps have been hit. Intuitively, the stumps are approximately the same color (blue-green), and some even light up when they have been hit. Perhaps we can detect this change in lighting by some simple thresholding. I explore the following ideas/techniques for this problem:

- Color thresholding (heuristic-based approach)
- Multi-scale template matching (pyramid-based approach)
- 3-layer CNN (gray vs color image)

Clearly, this half of the problem is much simpler than detecting when the batsman has crossed the crease. There is constant motion, which sometimes also causes artefacts, and more importantly, there are many different orientations so it's now clear how scale-invariant template matching can directly be generalized. However, perhaps information regarding the motion might help segment the batsman and thus aid classification. Thus, I explore the following alternatives for the other half:

- 3-layer CNN (feature learning)
- 3-layer CNN with Optical Flow (feature learning with motion)

Note that a pretrained version of the model in Super SlowMo paper is used to correct the scope of the project after my partner dropped the course. This only needs to be used at test time to offer more granularity and thus more information. Applying it to the training set would also increase the annotation time which is infeasible at this point. Also: I tried SIFT, Logistic Regression and PCA, code included on the repo, but they don't perform very well and I cannot fully cover them in this report for space constraints.

3 RESULTS

3.1 COLOR THRESHOLDING

Color thresholding is a really simple idea: when the ball hits the stumps, they light up. This creates a sudden change in brightness that we can hopefully detect simply by looking at pixel intensities on the red channel, for example.

In particular, I employ the following algorithm to the video to extract the timestep at which the stumps were first hit.

1. For each frame, compute a "reds"-value by doing the following:

    a. Gaussian blur (9x9 filter with sigma=5) the image and trim the image to cut out the top and bottom quarter of the image (only look at middle half where all the action usually happens).
    b. Normalize image by dividing maximum intensity across channels on each spatial location i.e. each pixel should have value 1 on some channel for better brightness invariance.
    c. Sum over pixel intensities where red > 0.9 AND blue/green < 0.8 i.e. only the very red pixels.
2. Smooth the resulting 1D signal corresponding to red levels that changes with respect to the time.
3. Normalize this signal by subtracting the mean and dividing the standard deviation (compute Z-score). The minimum time t such that $z_t > \min(z) + 1.5$ is predicted as the timestep where the stumps were first hit.

I visualized this 1D signal by overlaying it as points on the video. Moreover, once the bails are hit, I draw a green rectangle on the top of the image to indicate detection. The results are impressive. The red levels do experience a sudden increase as soon as the bails go off. [Video]

Out of the 33 clips, the stumps don't light up in 6 of them (for some day matches in CWC). We exclude these clips in the evaluation because we work under the assumption that they light up. Out of the remaining 27 clips, this technique works on 20 of them (~ 74%). The 7 clips that it fails on have some things in common: (1) the change in intensity of light isn't very significant in the video (because light isn't bright enough or it is occluded), and/or (2) there are moving red bodies (players with red jerseys) which can lead to significant frame-on-frame change in intensities of reds.

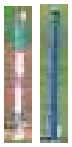
This is a really good baseline considering that there is no learned model and it only has a few hyperparameters (within thresholding). It lacks robustness to special cases, but in practical use-cases, we could enforce a fixed viewpoint so that we have a camera coordinates for the stump to directly calculated changes in reds at a specific location. Moreover, we can work around occlusions by capturing the stumps from multiple viewpoints.

Note that I also tried thresholding the color for the bat and use horizontal edges to detect the crease. In practice, it doesn't work very well because of noise in the image and the amount of color variation.

3.2 Template Matching

The next sensible thing to try is look for the stumps using template matching approaches because we know that the structure of the stumps is relatively homogenous. We can have multiple templates (e.g. for "ON" and "OFF") that classify the state of the stumps as well. However, while the structure is relatively homogenous, the scale may be different. This is why, I employ a multi-scale template matching process that utilizes image pyramids. Moreover, note that I found empirically that the color-space that we perform template matching on can have a significant impact on the performance. In particular, for problems such as this one, there is some work such as [3] which suggests that using YCrCb color space with Normalized Cross-Correlation (NCC) for tracking using template matching performs better than RGB and grayscale. Therefore, I convert images into that color-space in the algorithm below.

Note that this method has a big advantage: it is not only a technique for classification but also detection, i.e. we get a bounding box around the stumps. If this works well, we can use an ML model for example around that crop to significantly reduce the feature dimensions. The templates are shown on the right.



The algorithm I employed for this is as follows:

1. For each frame, go through all the templates and do the following:
    a. Convert image and template into YCrCb color space, and construct image pyramid.
    b. Perform template matching on each level of the pyramid using NCC and extract maximum.
    c. Convert these into coordinates of the original image by rescaling according to scale.
    d. Find the template and corresponding maxima with highest similarity. Using the original dimensions of the template, compute the final bounding box and class label (using template index).
2. Draw the bounding box. Use the class label for classification.

To really evaluate this well, we would need to manually annotate bounding boxes around each frame in each video. This is not really feasible given that this is a solo project. Therefore, I evaluated correct and incorrect samples based on a visual estimate instead of using IOU. Only approximately 57% (19 out of 33) videos had decent bounding boxes.

The reason for this terrible accuracy is sensible. Firstly, the stumps themselves don't have enough texture due to which we need to include some background in the templates to get a decent bounding box. This makes the matching process very sensitive to what

lies behind the stumps which means it is not very robust for most use cases. Another problem is that once the stumps are hit, their orientation changes and since template matching does not have rotation invariance, detection worsens. [Video]

3.3 Feature Learning

A simple convolutional neural network model seems sensible for this problem, especially for the classification of stump state. Below are the details regarding the architecture and training.

3.3.1 Training

For this part, I trimmed each image by cutting out the top 25% and bottom 20% of the image, and then resized each image to 100x50. This cuts out some of the irrelevant parts and focuses on the most informative regions. Since we have relatively less data, the small image size helps keep the feature dimensions small which would consequently lead to fewer parameters and less overfitting.

I split the data into 75-15-10 train-validation-test split. Note that the validation and test split are relatively small to give the model enough data to train on. I also used three data augmentation techniques to artificially get some more data for training: random flips (horizontal and vertical), gaussian noise (sigma=1) and brightness change (+/- 2%). For optimization, I used Stochastic Gradient Descent with a batch size of 32, learning rate of 0.001 and momentum of 0.8.

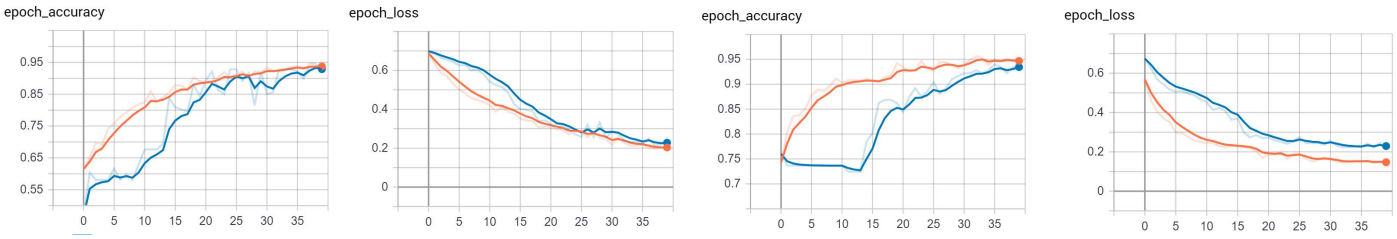
3.3.2 Architecture

Considering the complexity of the problem, a 3-layer CNN seems like a good choice because we can capture some hierarchy of features but at the same time, we don't overfit too much. I performed a grid-search on possible 3-layer CNNs, and the best model had 8 filters in the first layer, 12 in the second layer and 16 in the third layer. Also, empirically, color models perform better on both the problems probably because the color data contains valuable information (e.g. distinguishing other lights from red lights).

The network follows a standard structure (Figure 2, appendix) where convolutional layers are interleaved with pooling and batch normalization for downsizing the spatial dimensions and numerical stability. The activation function used is ReLU for all filters. The total trainable parameters in this model are just under 3,600 which is fewer than the number of pixels in a single image (5,000). Note that this is important because Logistic Regression (code included on repo under `pca.py`) also gets 97% accuracy on crease but it is probably overfitting due to the greater number of parameters and also the fact that when visualized, we don't see any desired relevance patterns for certain regions of the image nor the hierarchical features of a CNN. We can enforce sparsity using Lasso regression (Figure 3 and 4, appendix) but we still don't get translational invariance. We face a similar problem with PCA: while it reduces dimensionality to 100 features (from 5,000) and does get an f1-score of 0.95 on crease, we now are storing 5,000-dimensional eigen images so those technically contribute to overfitting as well. That said, it's a good baseline.

3.3.3 Evaluation

Averaged over 5 trials, the f1-score for the stumps was 0.93 with a mean precision of 0.90 and mean recall of 0.97. The average accuracy was 0.95, across 167 test samples. The same model architecture, using the same augmentation techniques, for the *in-crease* classification model yields f1-score of 0.92 with mean precision of 0.89, mean recall of 0.95 and mean accuracy of 0.96.



Training curves (orange for training set, blue for validation set): stumps model (left) and crease model (right).

Note that instead keeping two outputs in the final layer of the neural network, I use two different networks so that there is less competition between the two parts of the problem. This is important because there would be less transfer in features learnt for stumps vs those learnt for crease. Since we are aware of the class imbalance, it is best to not have shared layers to make those decisions because one might be favored over another.

3.4 Feature Learning with Motion

The big limitation with the traditional CNN approach is that we lose the motion information. As such, it may seem like there is no obvious advantage of motion because each image fully contains information for whether it is in-crease or out-crease, and whether stumps have been hit or not. However, in most video clips, since only the batsman is moving, perhaps we can simplify the problem by using motion information to segment the batsman.

The Super SlowMo paper linked above uses a common technique known as Optical Flow (OF) estimation which computes the motion between two frames. We can use a non-ML approach for OF such as the Farneback method [4] which, intuitively, performs multiscale, using an image pyramid, flow estimation by comparing image patches between frames in a local neighborhood. Below is a demo of how we get a nice segmentation of the batsman using this technique. [Video] I tried performing a simpler technique where I took image gradients in the temporal dimension, but there was a lot of noise in this technique.

So now, we can feed two inputs to our network: the flow and the image and learn to classify whether the batsman is in-crease or not. I used the same general architecture but added a few more filters in the second and third layer to better utilize the flow information (8 filters in first layer, 16 in second layer and 24 in third layer).

Since we would need to process the videos frame by frame again, I perform a one nearest neighbor classification to just use the previously labelled frames since these frames will be very similar (if not same) as they come from the same video. Moreover, to solve the class imbalance problem a bit, I perform sampling from the larger class (to only use a subset for training) and augmentation on the smaller class (to artificially increase size for training).

With these feature engineering and data preparation steps, our results significantly improve. On the right are the results for test (top) and validation (bottom) sets respectively.

We achieve 100% accuracy on the test set, and pretty well on the validation set as well. Note that we perform a bit worse on the validation set because it is a larger dataset that we test on. Regardless, both precision and recall improve compared to the previous model. Some samples are shown in the video. [Video]

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	83
1.0	1.00	1.00	1.00	73
accuracy			1.00	156
macro avg	1.00	1.00	1.00	156
weighted avg	1.00	1.00	1.00	156

	precision	recall	f1-score	support
0.0	0.98	0.95	0.96	132
1.0	0.93	0.97	0.95	100
accuracy			0.96	232
macro avg	0.95	0.96	0.96	232
weighted avg	0.96	0.96	0.96	232

3.5 Decision Function

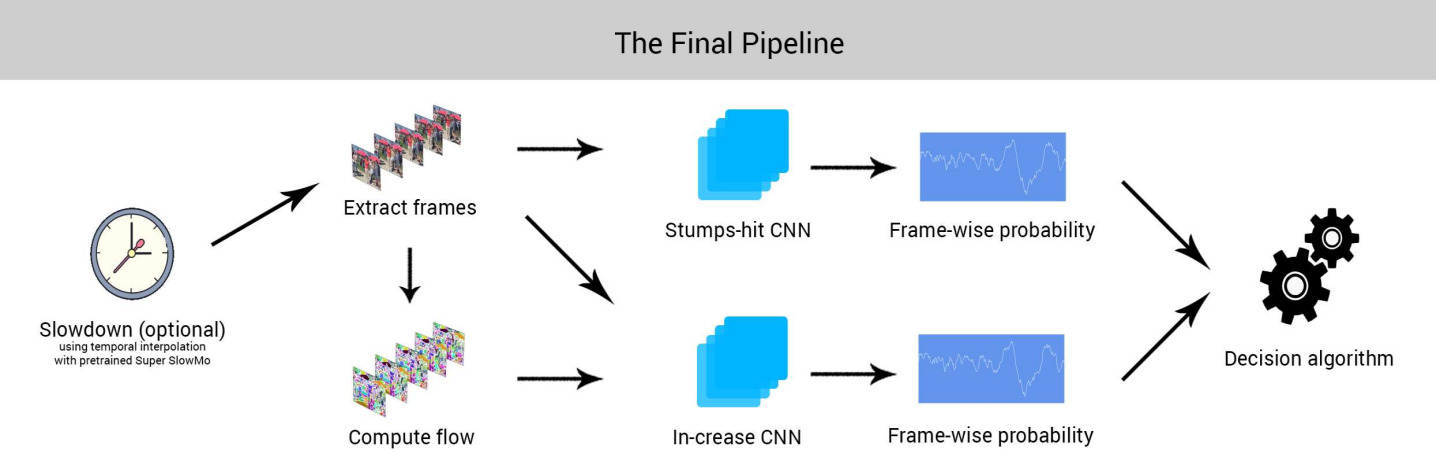
We need a way to map these probabilities of stumps on/off and batsman in/out of crease into a final decision of *out* or *not-out*. To do this, we can look at the two 1D signals we get that represent our model outputs for each frame. We only need to get the ordering right, so even if some classification outputs are not fully accurate, we can get fairly decent results!

Now, note that whenever the DRS system is consulted, there has to be some decision. Thus, in the trimmed clips provided, either (1) the batsman must have crossed the crease or (2) the stumps must have been hit (or both). Sometimes, however, the model might not return high enough probabilities for either case. To get around this, we can threshold at $\min(\max p_i, 0.5)$ instead of 0.5. That is, instead of applying a hard sigmoid, we can look at the probabilities of the two signals and if both of them are smaller than 0.5 for all frames, we attribute the higher probability as 1. Then, we get a final value of the minimum frame where the stumps were hit and the minimum frame where the batsman entered the crease. We can just look at the one that is smaller and make the final decision.

Using the best models shown above (CNN for stumps and CNN + Optic Flow for crease), we get a 100% accuracy in classifying *outs* and *not-outs* in this scenario. We also get the single best frame that we can decide based on. Note that these best frames are sometimes one or two frames off, but like I mentioned, we only need to get the ordering correct.

Although we need to use all the data we have for now, I tested it on a completely new clip that the model has never seen a single frame of before and we get the correct result. As I mentioned earlier, it is not very easy to get a lot of such video clips on the internet and thus evaluating it on an even larger dataset remains open for future work.

4 CONCLUSION



Overall, there are three big takeaways from this project:

1. *Problem formulation* can significantly improve data quality under certain constraints, and in this case in particular, we went from 7 negative examples and 26 positive examples to over 1000 data points with at least 250 data points for each class. Since many machine learning problems tend to overfit on less data, this offers a model with fewer
2. *Data preparation* can further alleviate the class imbalance problem by sampling and augmenting the larger and smaller classes respectively. While we may get a good accuracy using the original data, this significantly improves precision and recall on the smaller set.
3. *Feature engineering* can improve performance by encoding more information (e.g. about motion) without using significantly more parameters (e.g. multiple frames). In particular, Optic Flow seems like a natural substitute for encoding two frames (6 channels) into 4 channels (which helps with overfitting when we have so little data).

I also tried using PCA for dimensionality reduction (code included on repo), but the results were not very impressive. Moreover, I tried SIFT (code included on repo) for motion tracking. I was hoping this would work out of the box with Canny edge detection to find the crease. However, there was a lot of small motion in the scene due to which, it was hard to detect which keypoints belonged to the batsman. Moreover, often we didn't get the keypoints on the extremity of the bat, so even if the bat crossed the crease, we wouldn't be able to tell with good precision. [Video]

With that in mind, there are also some areas for future work. In particular, there are a lot of special cases of this rule in cricket that are lot harder to implement. For example, it is not always enough for the stumps to be hit. In some cases, when the batsman hits the ball, it must first touch the bowler before it can hit the stumps for it to be called an "out". Also, sometimes the fielder may mistime his motion and hit the stumps even before he gets the ball in his hands. In both of these cases, we must track contact between the ball, the stumps and the fielder throughout the video. While possible, there are way too many cases and viewpoints that this can be captured on and it would require a much larger dataset. Also, the batsman may sometimes enter the crease but not be touching the ground. This is technically not *in-crease* and the batsman could still be out. This would also require a lot more data to train a CNN on. Perhaps using stereo techniques to model this in 3D can help identify this more effectively as well.

In summary, I tried heuristic techniques (thresholding), Template Matching, Convolutional Networks, Temporal Image Gradients, Optical Flow, SIFT, and PCA. For space restrictions, I could only go through the more promising techniques in the report above.

5 References

[1] Harun-Ur-Rashid, Md, et al. "Crick-net: A Convolutional Neural Network based Classification Approach for Detecting Waist High No Balls in Cricket." *arXiv preprint arXiv:1805.05974* (2018).

[2] Jiang, Huaizu, et al. "Super slomo: High quality estimation of multiple intermediate frames for video interpolation." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018.

[3] Sebastian, Patrick, and Yap Vooi Voon. "Tracking using normalized cross correlation and color space." 2007 International Conference on Intelligent and Advanced Systems. IEEE, 2007.

[4] Farnebäck, Gunnar. "Two-frame motion estimation based on polynomial expansion." *Scandinavian conference on Image analysis*. Springer, Berlin, Heidelberg, 2003.

6 Appendix

Figure 1: Complex scenes with occlusions, various bodies and differing conditions

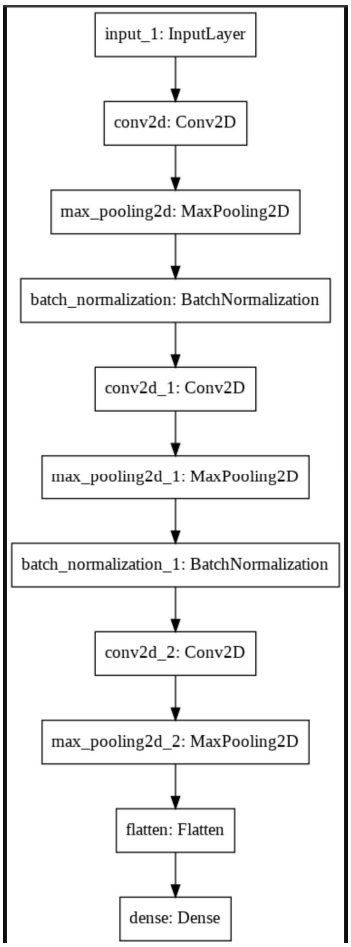


Figure 2: Neural network architecture

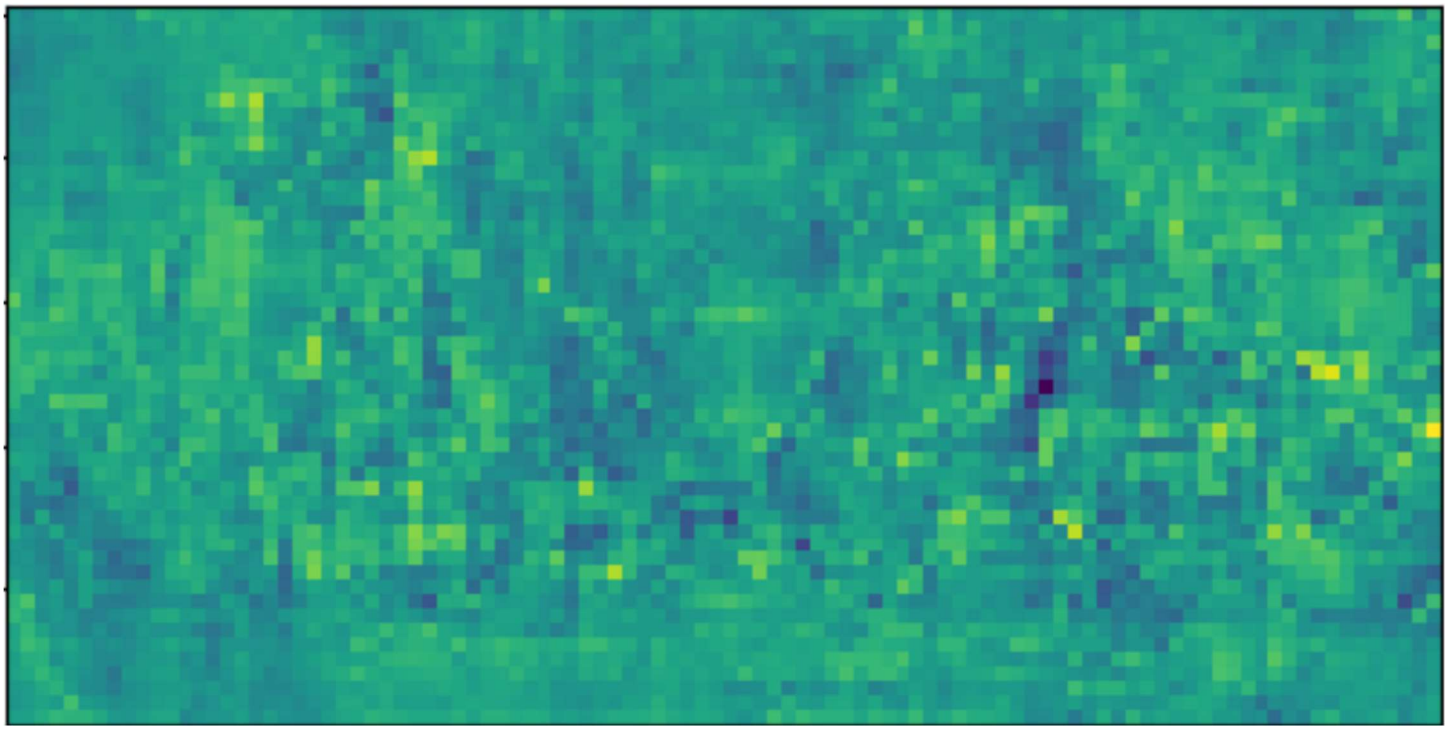


Figure 3: Parameters learned by logistic regression for stumps (no relevance patterns)

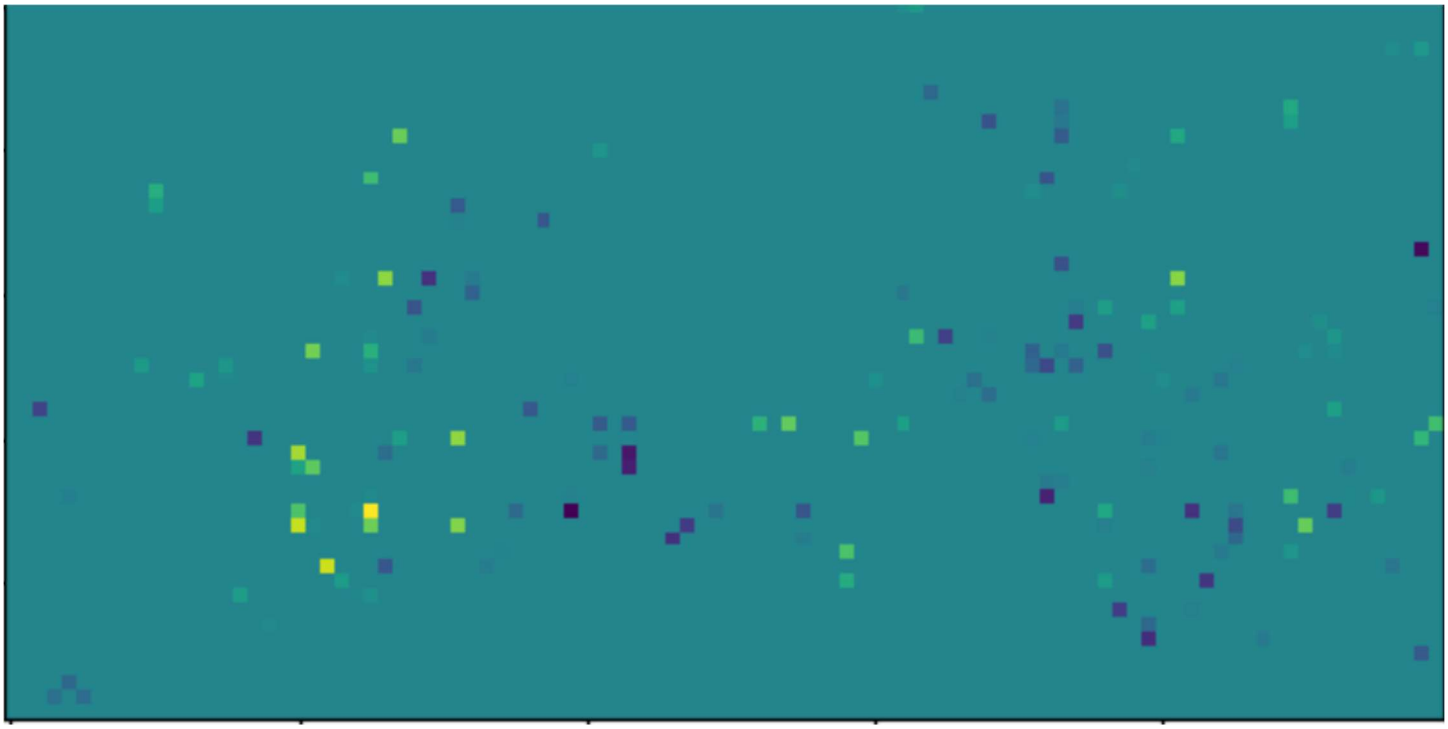


Figure 4: Parameters learned by logistic regression for stumps using L1 penalty for sparsity (no translational invariance)