FAT - Farm Animal Tracking

Kemal Erdem
Marek Pokropiński

February 24, 2021

Abstract

We're proposing a solution for tracking the farm animals (pigs). The solution is based on SSD FPN [9] network with the addition of custom Siamese Network [5] and heuristic for tracking. Using the Siamese Network for top-view recognition isn't a usual concept but works well and achieves **0.91 weighted F1 score** on our test dataset. Our experiments were limited by available resources to only try a few different settings. Even with this restriction tracking was successful enough to consider future work on this solution.

Contents

1 Problem description, assumptions, and potential problems **3**

2 Dataset **4**
2.1 Overview 4
2.2 Annotations 4

3 EDA **6**

4 Preprocessing **7**
4.1 Bounding Box annotations 7
4.2 Frame generation 7
4.3 Data augmentation 7

5 Proposed solution **9**
5.1 Object detection 9
5.2 Animal recognition 9
5.3 Tracking 9
5.4 Evaluation 10
5.4.1 Detection 10
5.4.2 Siamese Network 10
5.4.3 Tracking 11

6 Experiments **13**
6.1 Object detection 13
6.2 Siamese Network 14
6.3 Tracking 21

7 Conclusions **25**
7.1 Data . . . 25
7.2 Detection . . . 25
7.3 Recognition . . . 25
7.4 Tracking . . . 26
7.5 Final Thoughts . . . 26

Appendices **27**

A Detail Tracking Results **27**

B File Attachments **30**
B.1 Siamese Network structures . . . 30
B.2 Example video with tracking . . . 30

1 Problem description, assumptions, and potential problems

Animal detection and tracking is a common problem on farms. Current SOTA models allow us to quite precisely detect objects (animals) on the video feed but recognizing individual animals requires retraining of the whole network in order to add one more class for that one individual. This could be done in very small environments but on large farms might be difficult to achieve.

Animal tracking might be useful when trying to improve animal's health. Precise tracking can tell us what kind of activities is the animal engaged with and the total amount of movement in a day. With tracking data, we can perform anomaly detection to find any unusual behavior in a given subject.

The main goal of the project is to enable tracking of individual farm animals based only on a video feed. Each animal has a unique ID that might or might not be on a particular video.

This problem can be decomposed into three different parts:

- **detection** - detect all animals present in the frame
- **recognition** - recognize the identity of every detected animal
- **tracking** - store and track position of every individual for the time of the video

Because of the problem complexity, we're going to create a pipeline that consists of multiple stages and have one evaluation metric at the end. As an input for this pipeline, we'll use a video feed and the output is going to be a path for every single animal with its ID.

There are a couple of challenges for animal recognition on the video feed. The first one is a top view from the camera. Usually, for recognition, we're using front cameras to obtain easily distinguishable features from the individual. Top view prevents us from getting those features and forces the network to use only features of the whole body rather than the face. The next problem is video quality, because an image is distorted by the lens, some animal features might be hard to learn. This is a case then comparing animals close to the camera center (no or minimal distortion) to animals on the edge of the video (high distortion).

Tracking has its own problems. When reviewing the videos, we've spotted situations when animals are overlapping and standing very close to each other. This might cause object detection to drop some frames because two boxes are very close to each other and IoU value will remove one of the boxes because it'll be considered as the same box.

Some of the videos are taken at night and have no color values for the pixels. Because we're going to use RGB values as an input for our model might have a problem with GrayScale images. This problem is related to both object detection and object recognition (some features might depend on colors).

2 Dataset

2.1 Overview

Dataset is obtained from *PSRG - University of Nebraska*[10]. It consists of 15 videos, each half-hour long, and recorded at 5fps with a resolution of 1520x2688px. That gives around 135000 individual annotated frames.

Figure 1: Annotated frames example (Source PSRG [10])

Videos are recorded in a couple of different locations with different animal activity levels and time of day (either night or day). Each video is recorded using a wide lens camera and the camera is stationary. The minimum number of animals on the video is 7 and the maximum number is 16. The total number of unique animals present in the dataset is also 16.

2.2 Annotations

Annotations for the dataset are stored in *.mat* files. There are different files for different types of annotations:

- **LocationSelected.mat - 1 x num_of_videos x 4 x num_of_frames x num_of_animals** matrix containing two points (x,y) for every frame of the video and every animal (Fig. 1).
- **InitialLabels.mat - 1 x num_of_videos x 1 x num_of_animals** matrix containing IDs of all animals present on the video.
- **InitialPositions.mat - 1 x num_of_videos x 4 x num_of_animals** matrix containing values for two points describing initial position for all animals in the video, each animal has two points (x,y) for the front and the back on of the animal (Fig. 1).

At this point every animal on every frame has 5 values that define it:

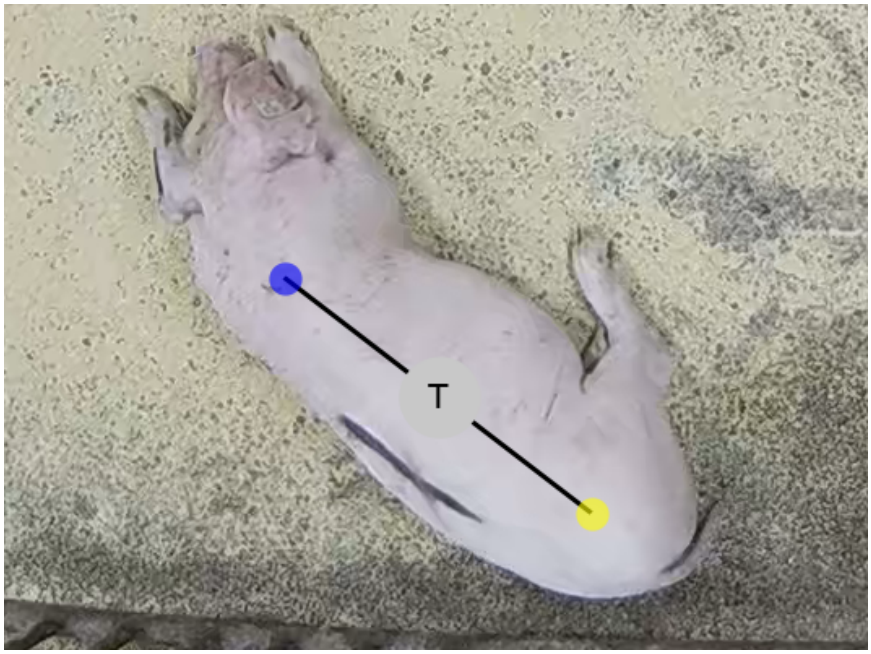

Figure 2: Annotated pig

- **(x,y) of the top part** - pixel coordinates for the top part of the animal (purple dot on Fig. 2)
- **(x,y) of the bottom part** - pixel coordinated for the bottom part of the animal (yellow dot on Fig. 2)
- **ID** - unique identifier of the animal. Has no coordinates but it's presented in Fig. 2 between two points (T)

3 EDA

Dataset is divided into 16 videos, on each of them we have some number of pigs. The minimum number of pigs is **7** and the maximum number is **16** (there are 16 different pigs in the whole dataset). In Fig. 3 we have a pig distribution per video. Each pig has a unique ID.

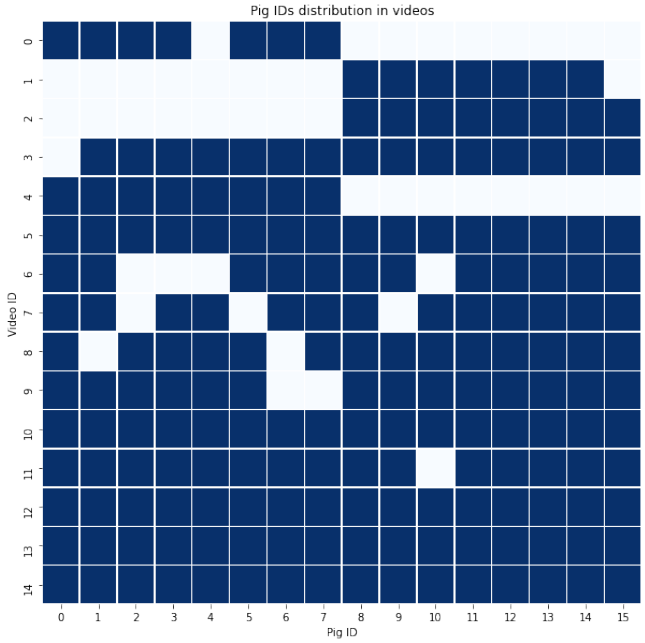

Figure 3: Pig distribution per video

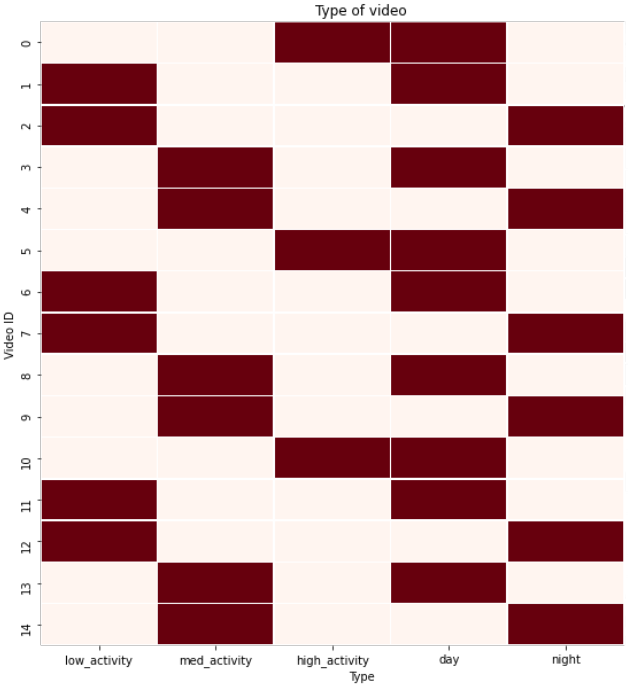

Figure 4: Video types (activity and time of day)

There are different types of videos, we can distinguish those types as **activity**. and **time of day** type. Activity type has 3 possible values (*low_ activity*, *med_ activity*, *high_ activity*), time of day type has 2 possible values (*day*, *night*). Fig. 4 shows the distribution for each video.

To make it easier to read Figures 5, 6, 7 are showing exact counts for every type of data.

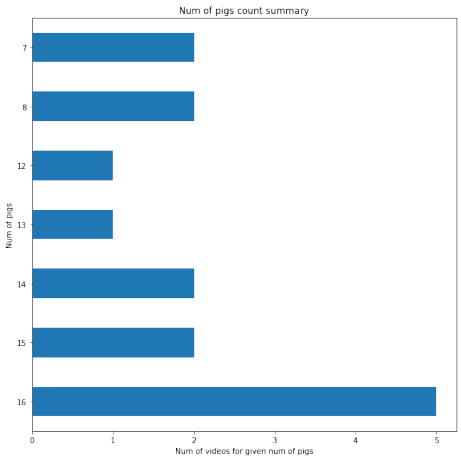

Figure 5: Video types (activity and time of day)

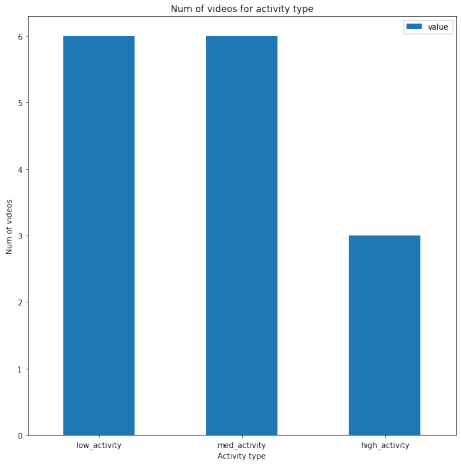

Figure 6: Activity type dist

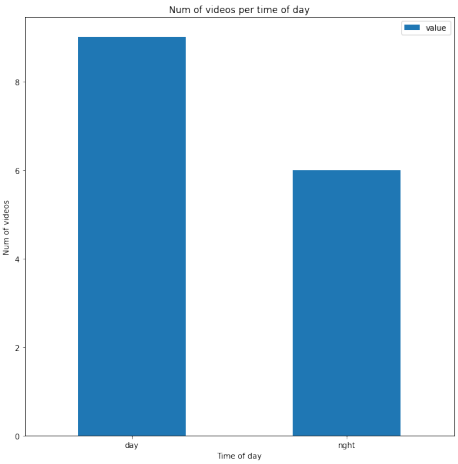

Figure 7: Time of day dist

4 Preprocessing

4.1 Bounding Box annotations

Because the original dataset doesn't provide bounding box coordinates we had to generate them based on location and pose data from annotations.

Let's call the distance between annotated points of a single pig v. To create a bounding box first we extend given points by factors given on 8 to be outside of pig image (red points on image). Then we create points in the perpendicular direction so the pig fits in a rotated rectangle. To get an axis-aligned bounding box we calculate the minimum and maximum coordinates of points showed on the image as blue.

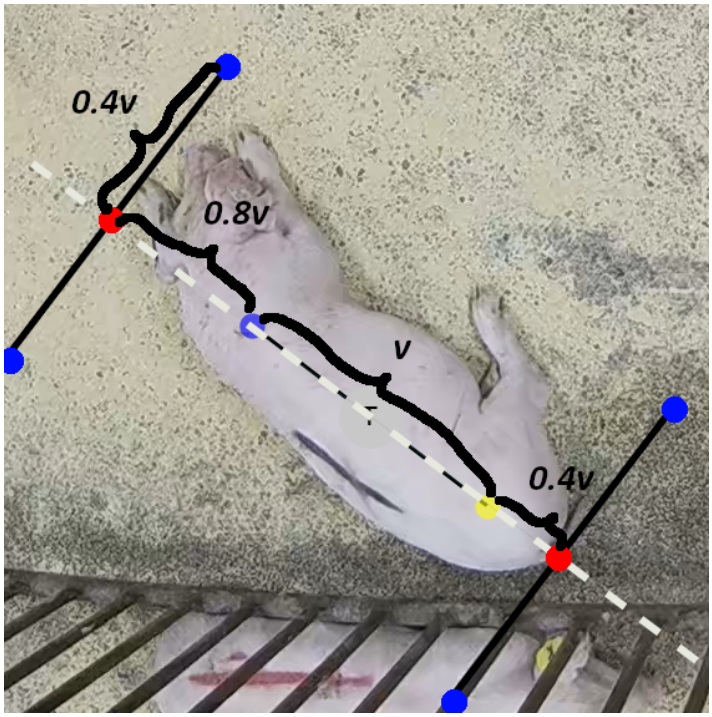

Figure 8: Generation of bounding boxes

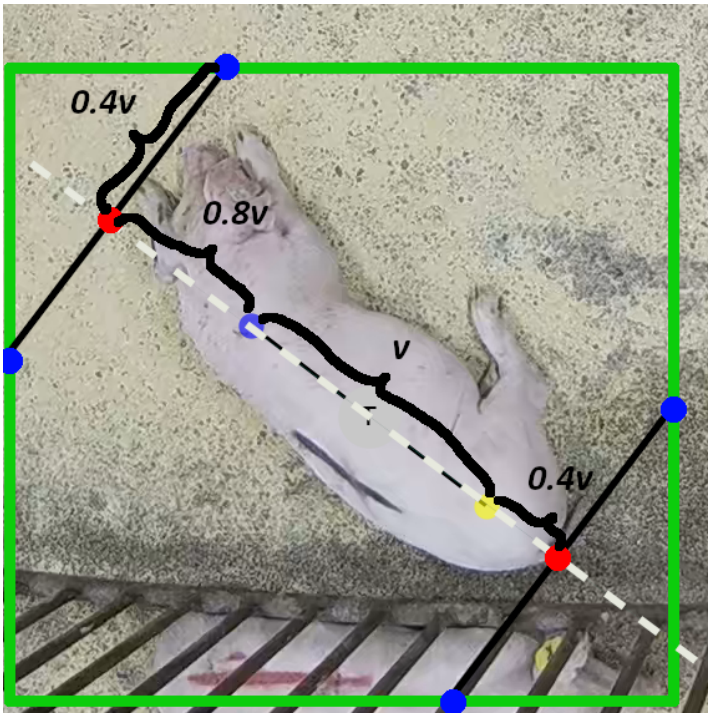

Figure 9: Generated bounding box

4.2 Frame generation

To generate frames from the video OpenCV library was used. We opened videos as streams and saved every 9th frame as png image. This generated 100 images per video. Getting every 9th frame is caused by minimal differences in positions on those frames. Even a fast-moving pig is not able to move much in that time and we're preventing from getting very similar images in the dataset that way.

Images were split into training and test sets by taking the last 33 images from each video to the test dataset.

4.3 Data augmentation

To increase the size of the training dataset for animal recognition we used image augmentation. We took cropped images and augmented them by randomly changing their width, height, rotation, and brightness.

Width and height were changed by a random fraction in the range $[-20\%, 20\%)$ of total width/height. Brightness was changed in the range $[0.5, 1.5)$ of the original brightness. The image was rotated by random degree from range $[-10°, 10°)$.

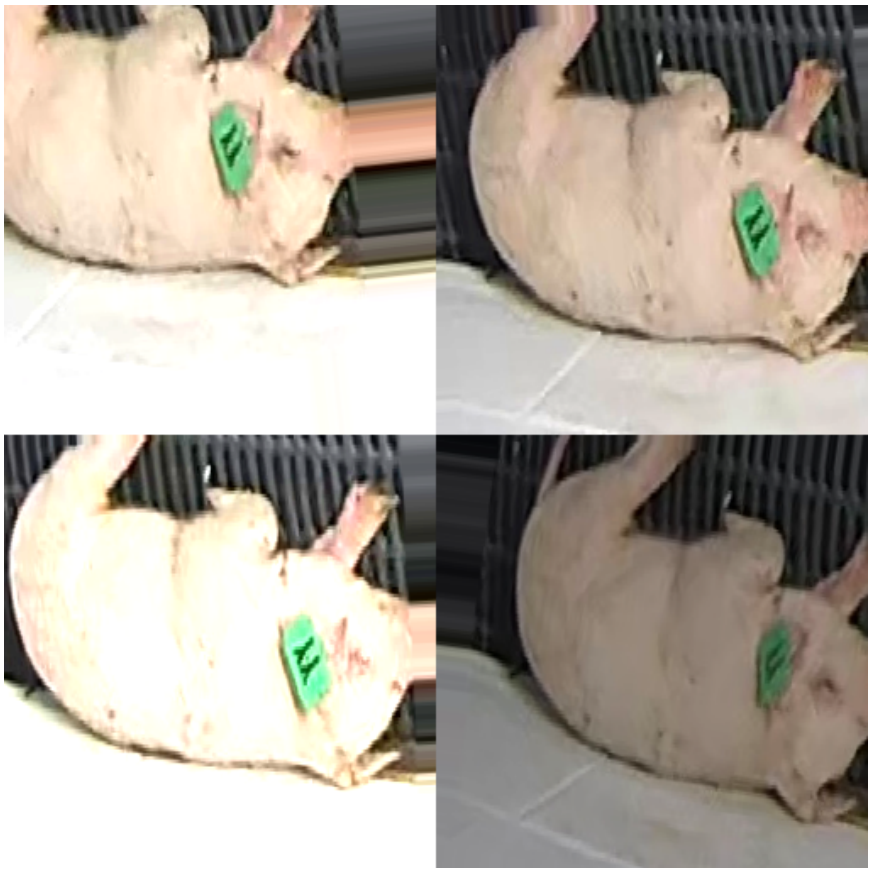

Figure 10: Example of augmented data

To create an augmented dataset, original generated images were used like in the Fig. 10.

5 Proposed solution

Our proposed solution utilizes deep neural network models. We implemented those in Tensorflow 2 framework.

5.1 Object detection

For object detection, we're going to use Feature Pyramid Network [6] with Single Shot Detector [9] base on Resnet50 [2] backbone. When selecting the detection model we decided to fine-tune the already existing solution because training our own detection model from scratch requires a lot of resources and current SOTA detection models are easily adjustable to new problems. To train detection model we used TensorFlow Object Detection API.

Object detection is going to be the first part of the solution. Detected bounding boxes (*bboxes*) are going to be used as feed information to the second stage, which is a Siamese Network [5].

5.2 Animal recognition

After obtaining cropped images for individuals using object detection we're going to use Siamese Neural Networks for object recognition [5]. This network is going to use pre-trained CNN (with fine-tuning) as a feature extractor and then FC layers on top with Triplet Loss function [14]. Siamese Network produces a feature vector for every image and those vectors is going to be used to compare individuals.

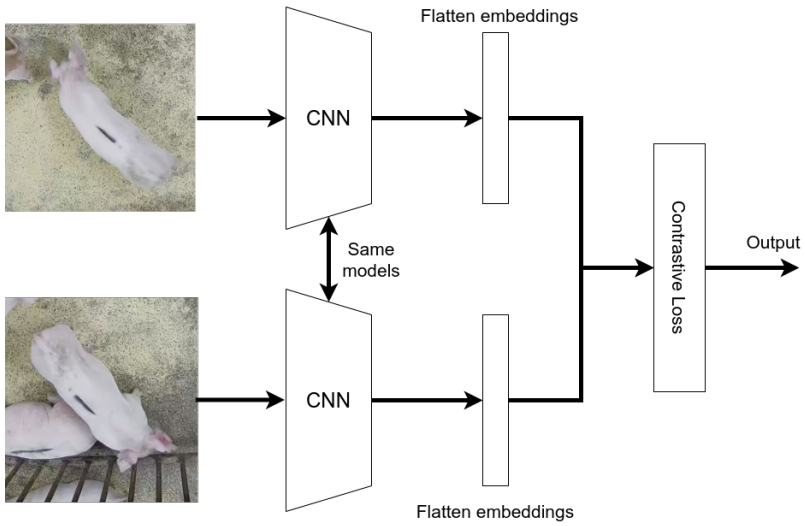

Figure 11: Siamese Network structure

We're going to use multiple CNN bases for our Siamese Network and compare results to select the best one. Because of limited resources, we cannot tune all hyperparameters and/or select the best structure for the network. Instead, we're going to use similar top layers in our network and retrain it with different CNN structures.

5.3 Tracking

For image tracking we're going to compare a couple approaches:

- **Positon comparison**

- **Tracking with Kalman Filter**
- **Siamese network with averaging**

Position comparison using only the position of a previous frame centers to generate tracks for each BB. When the next frame is generated, the position is compared between each BBox in the current frame and the previous one. The center of a new BBox is compared to every BBox from the previous frame and then all distances are sorted to get the closest one. This is repeated for every BBox and then assigned from the closest to the farthest.

Tracking with distance and appearance similarity uses the distance of bounding boxes from the previous frame and appearance similarity of previous k frames of a path to assign new detection to paths. For appearance similarity cosine similarity between embeddings of cropped images is used. We take average similarity of k previous images assigned to a path.
For distance euclidean or Mahalanobis distance is used. Mahalanobis distance is used if paths are predicted with a Kalman filter.
The total distance between new detection and path is calculated by aggregating position distance and appearance similarity using a weighted sum.

$$d^{app}(i,j) = 1 - similarity(i,j) \tag{1}$$

$$d(i,j) = (1-\alpha) * d^{pos}(i,j) + \alpha * d^{app}(i,j) \tag{2}$$

Siamese network with averaging is going to assume that tracking is done not in real-time. This approach is going to average recognition for a tracked animal to reduce mismatching. Each frame recognition is stored and at the end of the video interval, all recognitions are averaged to obtain one ID. Intervals might vary but in our tracking, we're going to use an average of 10 frames. After the ID is defined for an interval, the path is appended onto the path with the same ID. This tracker is going to use Position Tracking as well to reduce mistakes in case the Siamese Network predicts incorrect ID.

5.4 Evaluation

5.4.1 Detection

For detection evaluation we're going to use standard metrics like **mAP** (mean Average Precision) and **mAR** (mean Average Recall).

5.4.2 Siamese Network

Because our network is going to use Siamese Network for class recognition we have to define a metric that is going to evaluate the use of that network in our problem instead of just showing the Loss value used in optimization. The most common evaluation for classification problem is simple **F1 score**[18] per every class and that is going to be our main metric. Additionally, we're going to create **confusion matrix**[17] to visualize potential problems with a specific version of the Siamese Network. Because Siamese Network creates embedding space with minimal loss value, that loss cannot be used as an understandable metric. We could get very dense space but clearly divided by our classes where distances are small, on the other hand, we could get sparse space where vectors are far away from each other but mixed between classes. We're going to use space projections [16] to visualize those embedding spaces for every iteration of the siamese network. This should allow us to compare spaces and check if it creates clusters for a given dataset. This projection is using **UMAP**[8] (*Uniform Manifold Approximation and Projection*).

5.4.3 Tracking

To evaluate the whole process we have to create a metric that works on the pipeline rather than just on the one model from this pipeline. Metric for the whole process is going to use both tracking position and class prediction. If the classification is done correctly we're going to compare path using **MSE** (Mean Square Error) [19].

$$\text{MSE}(\text{true\_y}, \text{pred\_y}) = \frac{1}{N}\sum_{i=1}^{N}(\text{true\_y}_i - \text{pred\_y}_i)^2 \tag{3}$$

Where:

- **i** - either x or y from prediction of label
- **true_y** - true value of (x,y) for given class
- **pred_y** - predicted of (x,y) for given class
- **MSE** - Mean Squared Error (Eq. 3)

This metric allows us to calculate the error between annotated tracking position and predicted position. Error is calculated for every class so we don't have to use class in calculating error. If a class is wrong then the distance value for a given class will result in a maximum error ($2\sqrt{2}$) (length of the image's diagonal). Error is calculated in two different ways:

- **total** - sum up all errors from the entire track
- **interval** - calculate error per each interval (number of frames)

The second way of calculating the error is useful when the model makes a lot of mistakes and only partially classifies the correct track. The result for a given animal might look like in Fig. 12.

```
"total":{
  "abs_err":2.376041792402685,
  "avg_err":0.015329301886468936
},
"intervals":{
  "interval":10,,
  "avg_err":0.012911492549204156,
  "parts":[
    0.01234610854577355,
    0.012286282277825484,
    0.012911492549204156,
    0.01162852954522169,
    ...
  ]
}
```

Figure 12: Score format

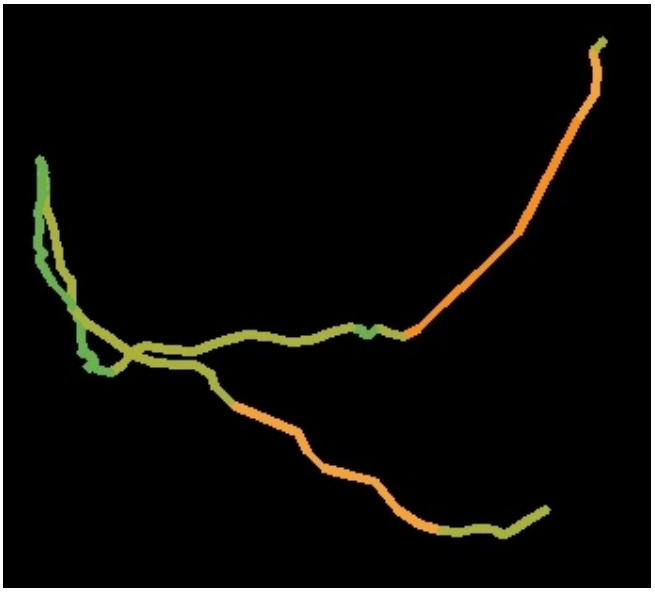

Figure 13: Score visualization on single track

Except for score values, each evaluation will have additional visualization of the scored track (based on *intervals* metrics) (Fig. 13). Each interval's value is marked with specific threshold color based on score (avg. MSE for that interval). Default Thresholds are as in Fig.14.

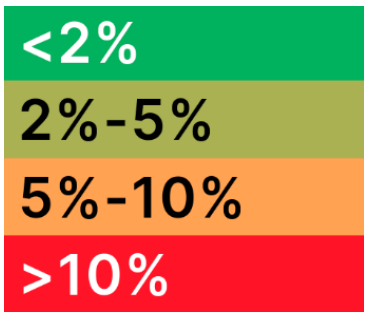

Figure 14: Thresholds

6 Experiments

6.1 Object detection

The object detection model is implemented with a help of the *object_ detection* library provided by TensorFlow [3].

Backbone Resnet50 is initially trained on COCO Dataset [7] and then fine-tuned on our train dataset. The training was done using similar parameters to those used in the training SOTA model with *random horizontal flip* and *random crop*. Optimizer is the **SGD** (Stochastic gradient descent)[11] with additional momentum.

- **learning rate** - 0.04
- **batch size** - 5
- **epochs** - 25000
- **warmup steps** - 2000
- **momentum** - 0.9

After training results on the test dataset are as defined in Table 1 and Table 2.

	mAP	mAP_{50}	mAP_{75}
SSD Resnet50 FPN	72.92	97.03	81.82

Table 1: mAP score

	mAR_1	mAR_{10}	mAR_{100}	mAR_{small}	mAR_{medium}	mAR_{large}
SSD Resnet50 FPN	7.04	60.14	78.73	27.69	69.33	84.63

Table 2: mAR score

To improve BB (*Bounding Box*) selection we're going to use Soft-NMS [1]. Soft-NMS applies scaling on detection scores, it uses the Gaussian function of the overlap (IoU). The final prediction example can be seen in (Fig. 15).

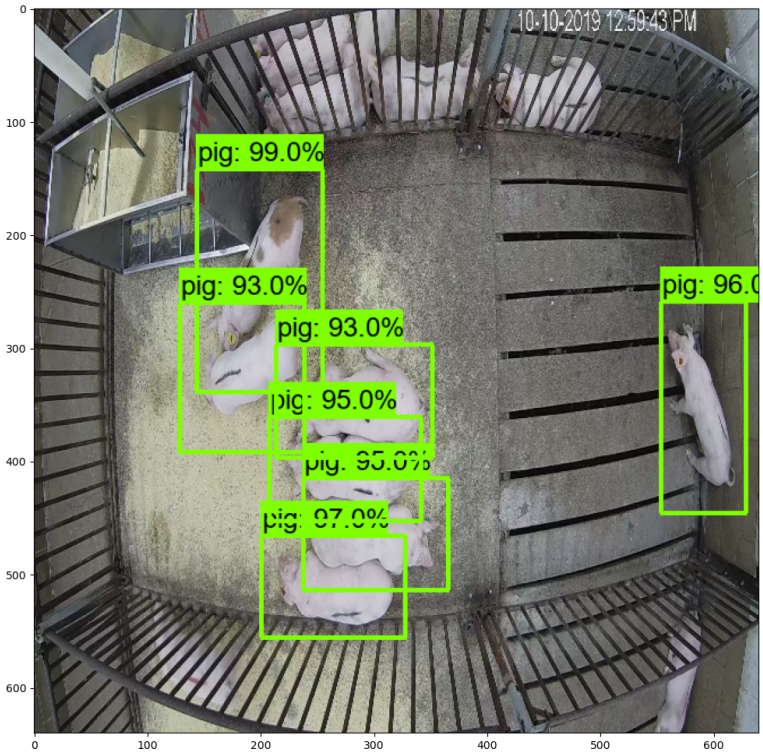

Figure 15: BB prediction example, with Soft-NMS

6.2 Siamese Network

For Siamese Network we've used three different CNN bases for feature extraction with fine-tuning.

- **EfficientNetB5** [15] - *block3c_add*
- **MobileNetV2 - alfa 1.0** [12] - *block_10_project_BN*
- **MobileNetV2 - alfa 0.75** [12] - *block_10_project_BN* (only for embedding space check)
- **ResNet101V2** [2] - *conv4_block23_out*

We decided not to use the last conv layer as a feature extractor like most transfer learning practitioners do when training classification models. Instead, we thought that dropping mode layers from the top could allow us to use more fine-grain features (we have to distinguish between very similar objects). In the list above, we're showing which layer was taken as the last layer in a given model. Because ResNet101V2 is a very large network we weren't able to fine-tune the whole network, instead we've unlocked only layers with *conv4** prefix. The same applies to EfficientNetB5 where only layers with *block3* were unlocked. The top layers for each network are the base but because the output of the base network is different, those layers have a different number of trainable parameters (Fig. 16. To train the siamese network we used the *TripletLoss* function and the following parameters:

- **learning rate** - 0.0001 or 0.001 (0.001 only for EfficientNetB5 base)
- **batch size** - 64
- **epochs** - 150
- **optimizer** - Adam

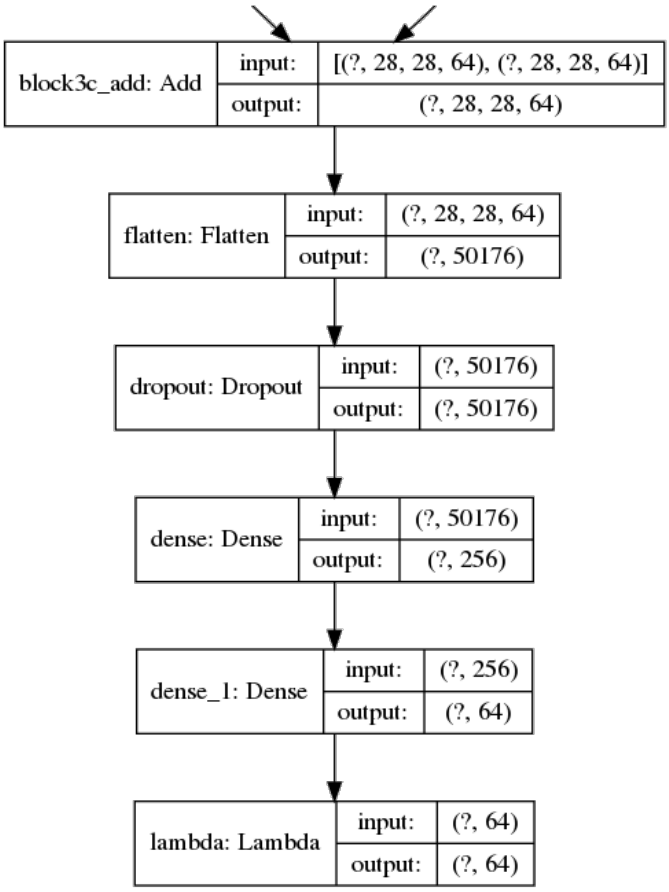

Figure 16: Top Layers of the Siamese Network where the base is *block3c_ add* layer of EfficientNetB5. Networks structures are available as attached files referred in the Appendix B.1

It is important to notice that *batch size* is not a random number. Each batch has to contain at least 2 images from possible classes. Because we have 16 classes it's advised to use at least 48 or more examples. Larger batch sizes might help but because our images are really similar we decided to use 64 so the loss function could compare hard examples. The network was trained on the original and the augmented dataset, but unlike in object detection, using augmented data didn't improve the results and just made training more difficult because of an epoch length. The best results in Table 3 are obtained on the original dataset.

	loss	epoch
MobileNetV2 alpha=0.75	0.1798	108
MobileNetV2 alpha=1.0	0.0633	118
ResNet101V2	0.0141	69
EfficientNetB5	0.2488	147

Table 3: Best Triplet Loss values per CNN base obtained on given epoch

Because loss metric is not accurate for our problem we've calculated classification scores for *MobileNetV2 alpha=1.0*, *ResNet101V2* and *EfficientNetB5* bases. To classify the embedding we were using an average of each class vector's position. To explain how this work first visualize embeddings of the train dataset using **UMAP**. To do that we have to calculate 64D embedding vector for examples in the train dataset

and then display them on 3D projected space (Fig. 17) (proper average embeddings are stored as 64D vectors).

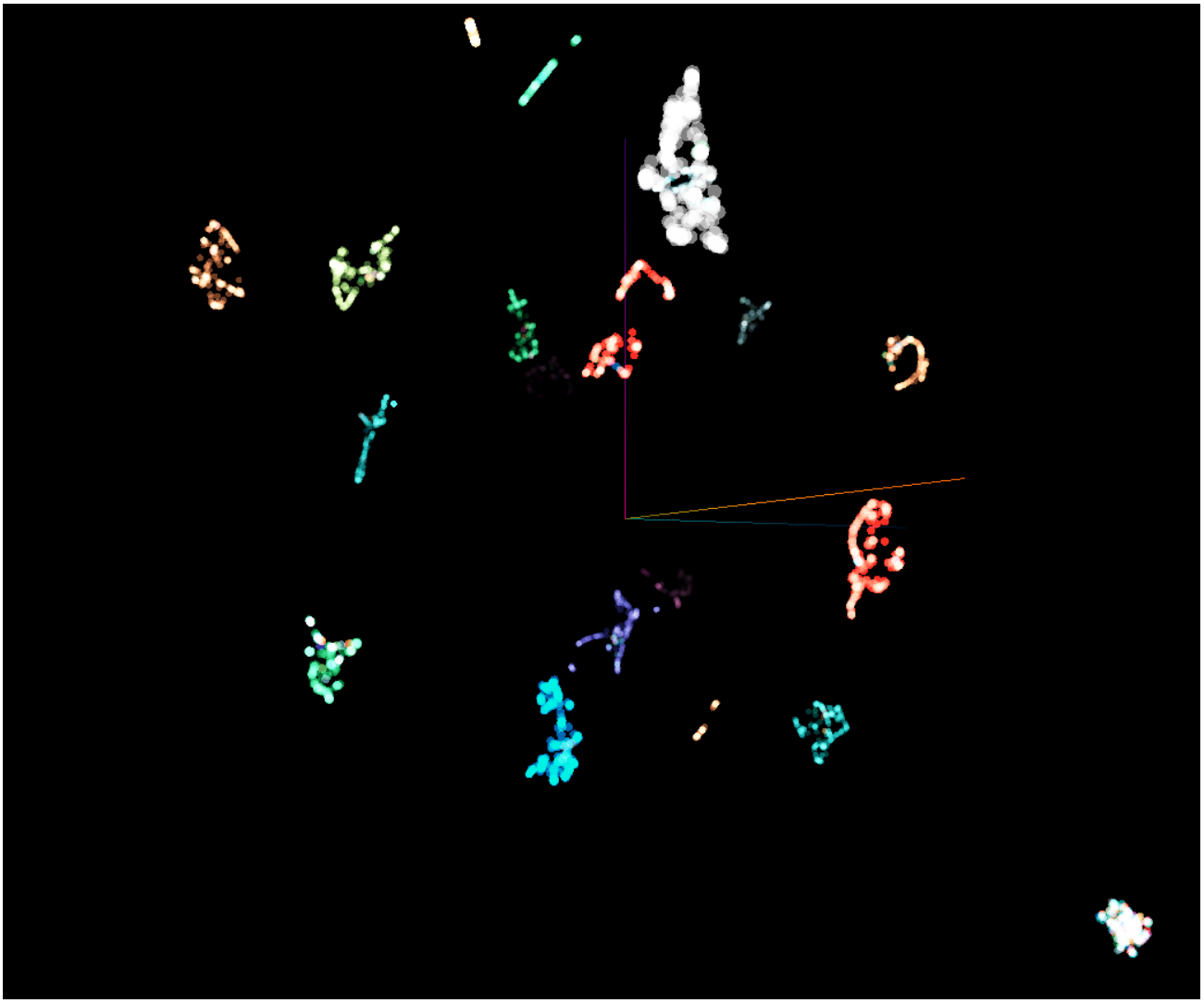

Figure 17: Train dataset visualization using MobileNetV2 base Siamese Network, each color corresponds to a different class

3D projection of the training dataset for MobileNetV2 base Siamese Network shows us clearly separable clusters and one cluster of outliers (right-bottom corner of Fig. 17. Those outliers are going to show on classification results but at this moment are not that important. From this point onward we're going to calculate the average value for every class (not cluster) and display only one vector per class in the training set (outliers are assimilated into average vectors) (Fig. 18).

To perform classification on the test dataset we're comparing each image's embedding to the average embedding generated on the train dataset. Table 4 shows the classification results per class (an assigned name for each pig). Network based on MobileNetV2 shows the best results with **avg. F1-score of 0.90** with ResNet101V2 second and EfficientNetB5 third.

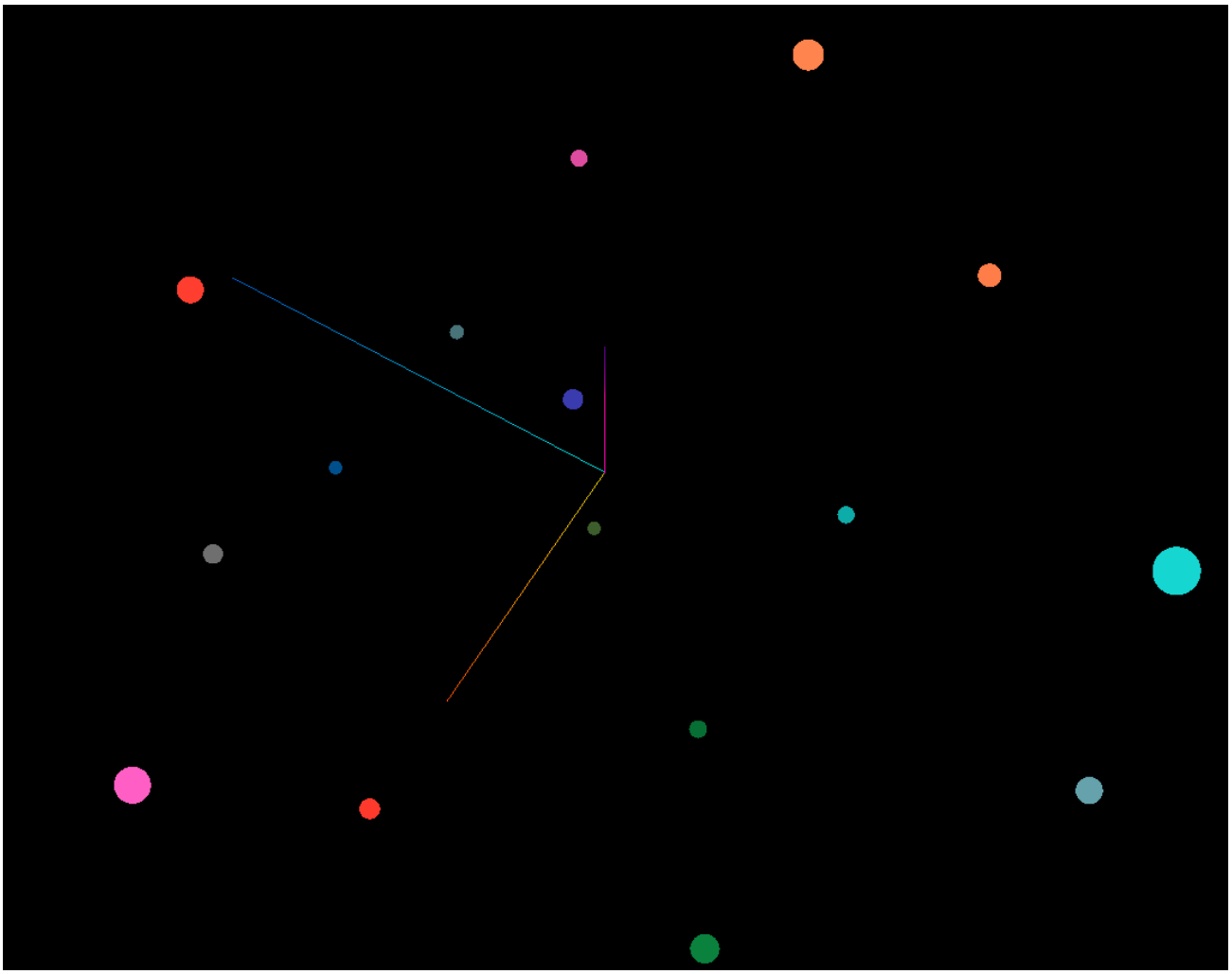

Figure 18: Average vectors visualization for MobileNetV2 base Siamese Network

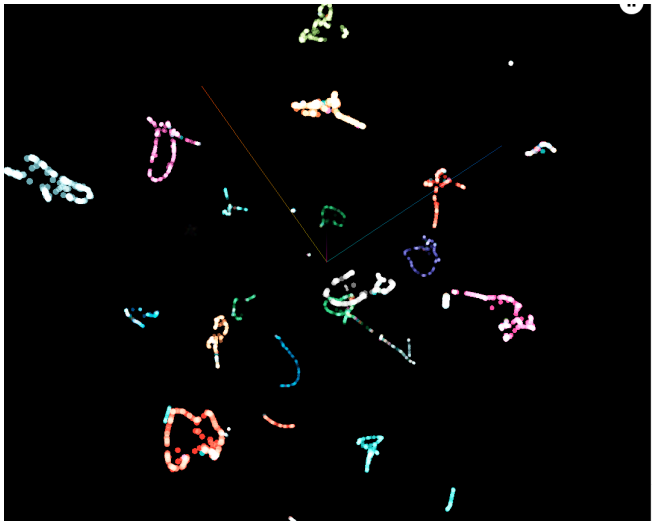

Figure 19: EfficientNetB5 vectors visualization
.

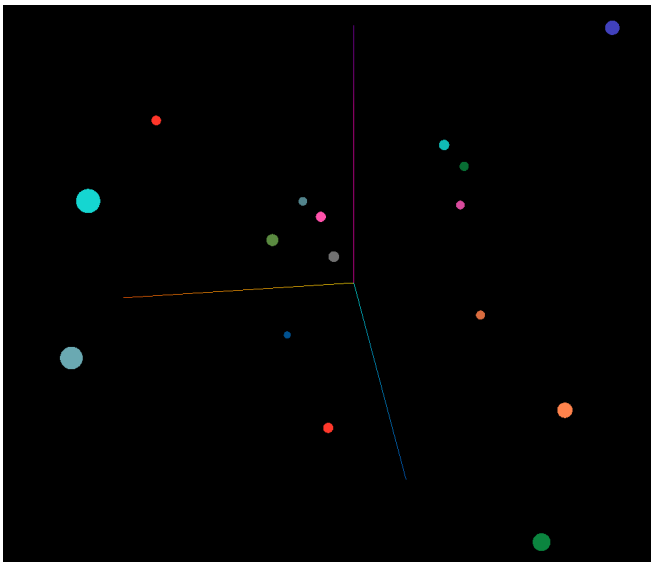

Figure 20: EfficientNetB5 avg embedding visualization

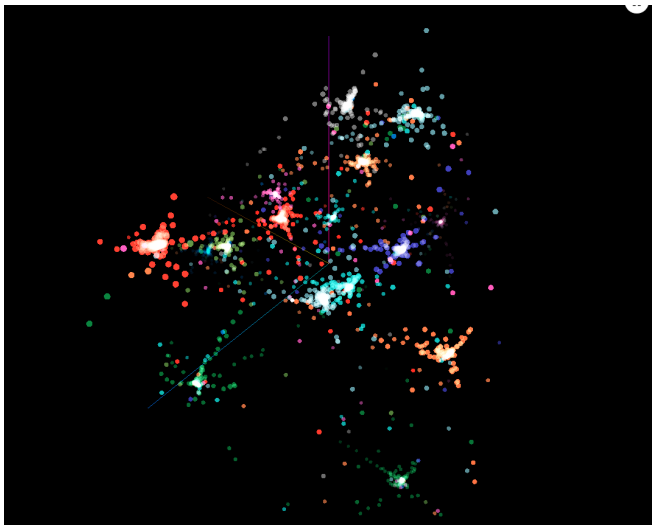

Figure 21: ResNet101V2 base vectors visualization

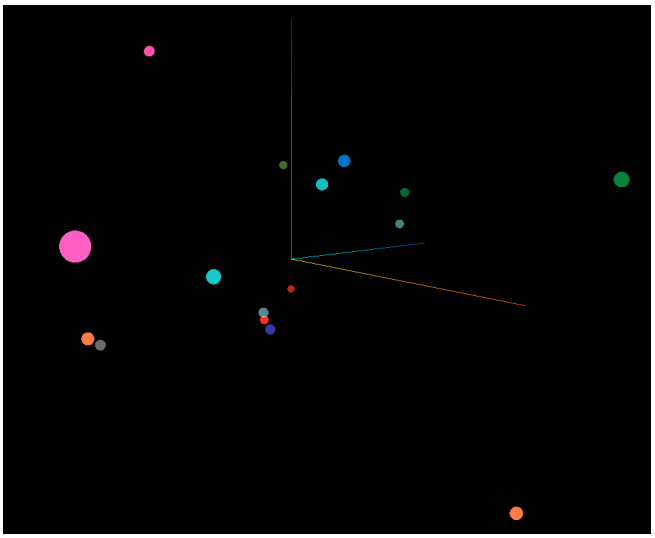

Figure 22: ResNet101V2 base avg embedding visualization

Model	ResNet101V2			**MobileNetV2**			EfficientNetB5			
	precision	recall	f1-score	precision	recall	f1-score	precision	recall	f1-score	support
James	0.62	0.56	0.59	0.98	0.92	0.95	0.70	0.20	0.31	3586
Robert	0.56	0.42	0.49	0.47	0.96	0.63	0.09	0.65	0.16	3281
William	0.21	0.60	0.31	0.92	0.87	0.90	0.25	0.15	0.18	2988
Bob	0.87	0.46	0.60	0.97	0.86	0.91	0.57	0.27	0.36	3283
Charles	0.48	0.19	0.28	0.97	0.85	0.90	0.23	0.08	0.12	2978
Anthony	0.46	0.63	0.53	0.93	0.84	0.88	0.32	0.36	0.34	3285
Paul	0.58	0.47	0.52	0.97	0.87	0.91	0.49	0.25	0.33	2989
Steven	0.41	0.68	0.51	0.98	0.92	0.95	0.32	0.37	0.34	3288
Kevin	0.53	0.48	0.51	0.98	0.89	0.93	0.79	0.24	0.36	3587
George	0.85	0.74	0.79	0.99	0.93	0.96	0.51	0.37	0.43	3282
Brian	0.76	0.52	0.62	0.97	0.92	0.94	0.53	0.28	0.36	2978
Edward	0.57	0.48	0.52	0.97	0.91	0.94	0.40	0.22	0.29	3587
Gary	0.66	0.74	0.70	0.98	0.95	0.96	0.61	0.28	0.39	3585
Eric	0.75	0.57	0.65	0.98	0.95	0.97	0.38	0.33	0.35	3430
Larry	0.51	0.56	0.53	0.97	0.92	0.95	0.60	0.35	0.44	3587
Scott	0.80	0.40	0.53	0.96	0.89	0.92	0.68	0.27	0.38	3265
accuracy	0.53	0.53	0.53	0.90	0.90	0.90	0.29	0.29	0.29	-
macro avg	0.60	0.53	0.54	0.94	0.90	0.91	0.47	0.29	0.32	52979
weighted avg	0.60	0.53	0.55	0.94	0.90	0.91	0.47	0.29	0.32	52979

Table 4: Classification scores for *MobileNetV2 alpha=1.0*, *ResNet101V2* and *EfficientNetB5* base Siamese Networks

Even if the classification results for MobileNetV2 are great there is a problem with one class called *Robert*. Precision on that class is very low for this network and we decided to check why is that happening. The key role in low precision plays our outliers cluster. All our outliers are clustered close to each other and when we calculate the average value for the class there is always one class closest to the outliers cluster. In the case of MobileNetV2 base Network that cluster is named *Robert* so all outliers are classified as *Robert*. We can look at the confusion matrix (Fig. 23) to see that a lot of examples are assigned to Robert but the rest of the dataset is assigned properly. There are some minor misclassifications but it is still more than usable for classification, especially because this network has no information about classes.

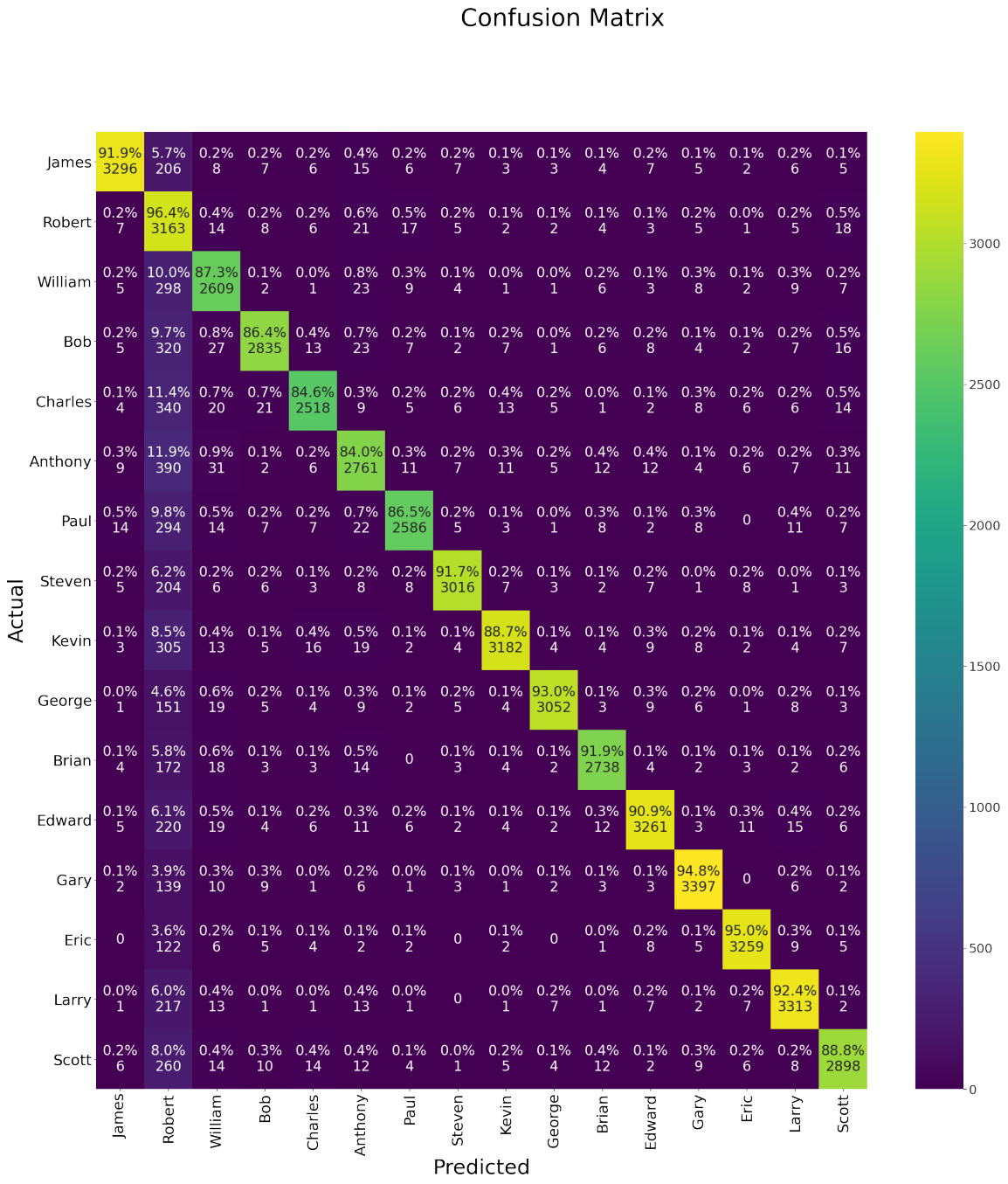

Figure 23: Confusion Matrix for MobileNetV2 base Siamese Network

Results for ResNet101V2 are slightly worse but not horrible (Fig. 24) As we can see on Fig. 21, ResNet101

is no creating embedding space where vectors are grouped into easily separable clusters like in the case of MobileNetV2. This causes the avg. embeddings to be closer to each other and misclassify a lot of examples.

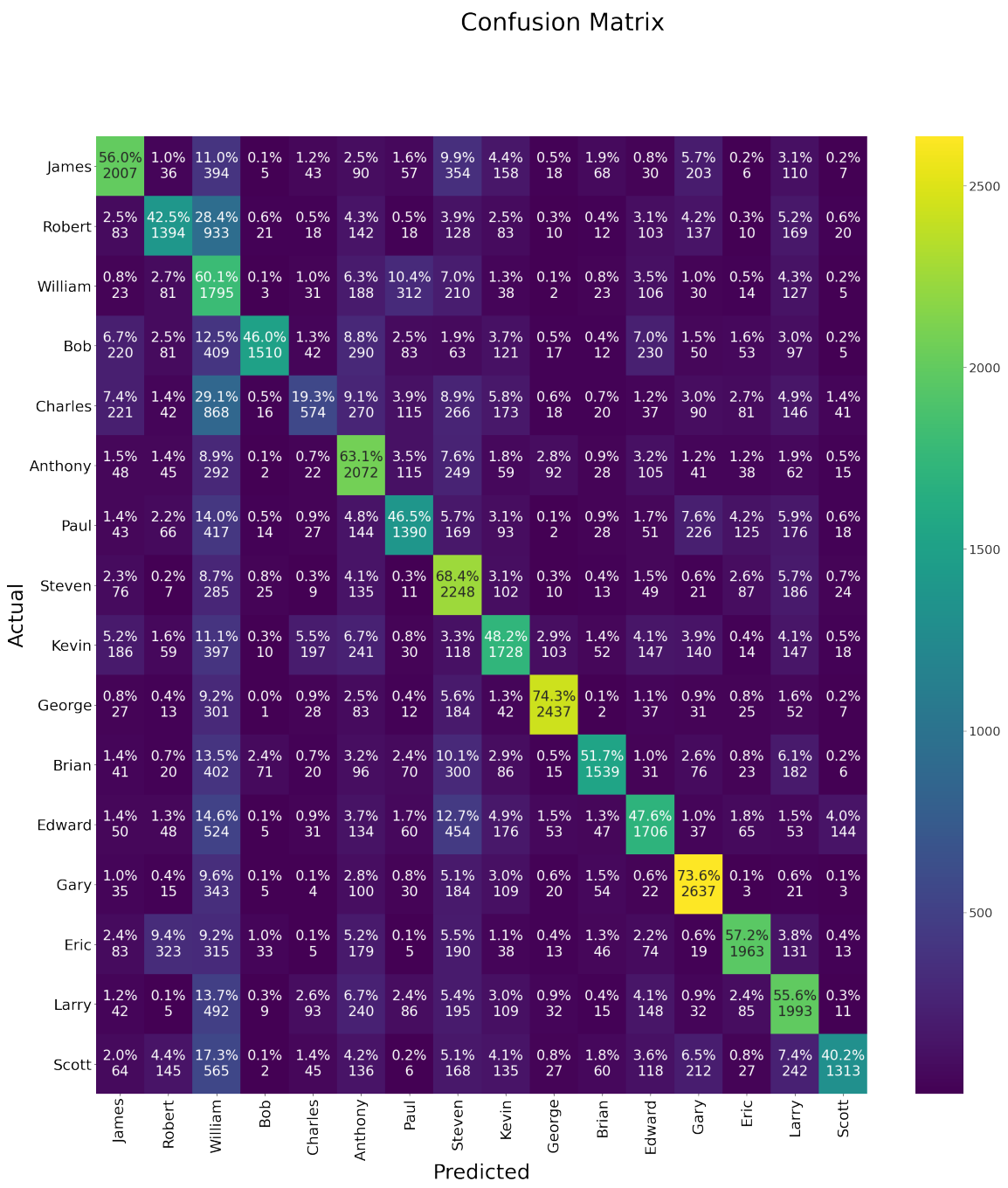

Figure 24: Confusion Matrix for ResNet101V base Siamese Network

EfficientNetB5 has the worst results (Fig. 25) from all three. When looking at the embedding space (Fig. 19) for this CNN base, there is more than one cluster per class. Because of that, when we're calculating the average embedding vector it is placed somewhere between those clusters. We could probably improve results by calculating more than one avg. embedding per class and comparing test results with the closest one.

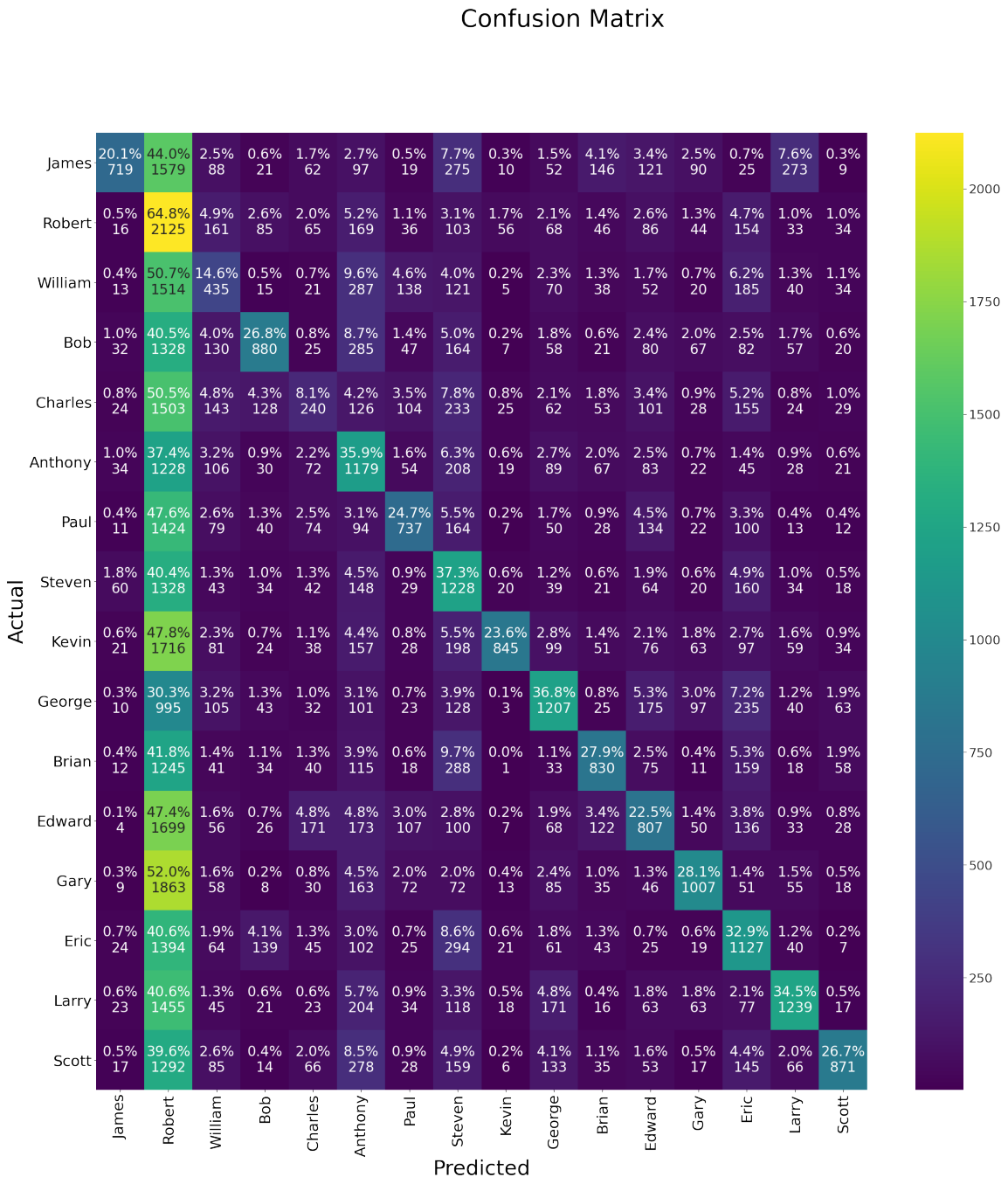

Figure 25: Confusion Matrix for EfficientNetB5 base Siamese Network

From all three networks, network based on MobileNetV2 was chosen for tracking purposes.

6.3 Tracking

Tracking experiments are evaluated on 5 different videos:

- **11_nursery_high_activity_day.mp4** - 16 pigs, hight activity, day, start at 6000 frame
- **12_nursery_low_activity_day.mp4** - 15 pigs, low activity, day, start at 6000 frame
- **13_nursery_low_activity_night.mp4** - 16 pigs, low activity, night, start at 6000 frame

- **14_nursery_medium_activity_day.mp4** - 16 pigs, medium activity, day, start at 6000 frame
- **15_nursery_medium_activity_night.mp4** - 16 pigs, medium activity, night, start at 6000 frame

For each video we're using only part from test dataset (6000 frame onward). Unfortunately there is no *high activity, night* video so we cannot test it (that's understandable because, pigs usually sleep during the night). Videos have 2 different metrics, **total_mse** and **interval_mse** (described in the Section 5.4.3). There are 3 Trackers in each evaluation:

- **DefaultTracker** - short DT
- **AvgEmbeddingTracker** - short AET
- **KalmanTracker** - short KT

Because the whole tracking results consist of hundreds of records they are adds as an Appendix A at the end of the report. From now we're going to use only average values per tracker to show differences between particular trackers.

Table 5: Tracking Error

Table 6: Total

	DT	AET	KT
Vid 11	0.32	0.47	**0.33**
Vid 12	0.28	0.31	**0.28**
Vid 13	0.10	**0.10**	**0.10**
Vid 14	0.31	0.45	**0.31**
Vid 15	0.24	0.42	**0.25**

Table 7: Interval

	DT	AET	KT
Vid 11	0.40	0.52	**0.46**
Vid 12	0.25	0.35	**0.30**
Vid 13	0.10	**0.10**	0.12
Vid 14	0.30	0.53	**0.33**
Vid 15	0.20	0.48	**0.23**

Before discussing the results we have to explain how to approach the results. DefaultTracker is a tracker that has no information about the class. Its values are defined just as a tracking error because classes are ignored (assigned to the correct class). That's why it should have the best results. Other trackers are predicting classes so the results will be equal or worse than the Default Tracker. If we look at the avg. results we can come to the conclusion that Kalman Tracker is better than Avg. Embedding Tracker. That might be misleading because AET (Avg. Embedding Tracker) has a large error variation which is visible when we check results in the Appendix A.

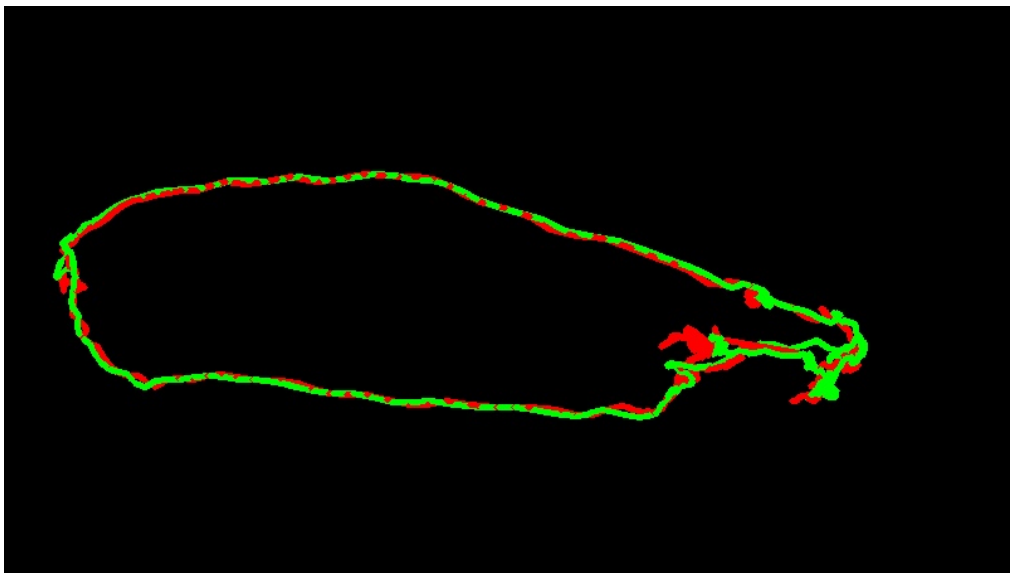

Figure 26: Vid 15 - George's track comparison with annotation

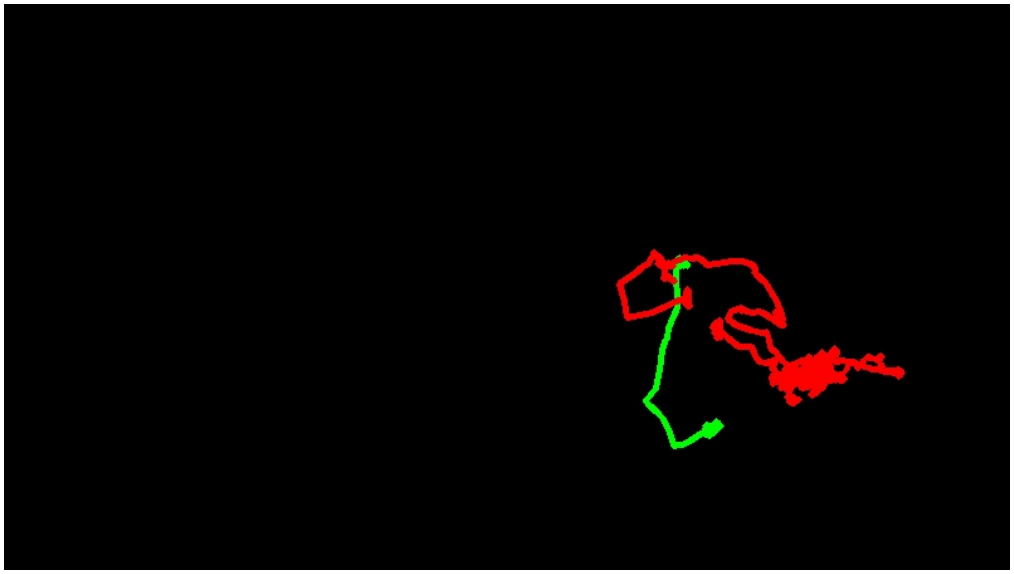

Figure 27: Vid 15 - Eric's track comparison with annotation

To explain how average results are not that precise, we can compare tracking of Robert (Fig. 26) and Eric (Fig. 27). Green path is a path extracted from annotations, the red path is a path generated by the Tracker. We can see that Eric's path has a large error in compare with George's path. An avg. error treats those paths as equal because they have the same number of frames (Eric didn't move that often in the video). This is even worse when some animals are misrecognized by the tracker and could get maximum error available. This especially happens with AET where some tracks have error > 1. Some mismatched tracks are shown in Fig. 28 and Fig. 29. Steven's annotation is matched by Brian's tracking and that causes a large error for both of them.

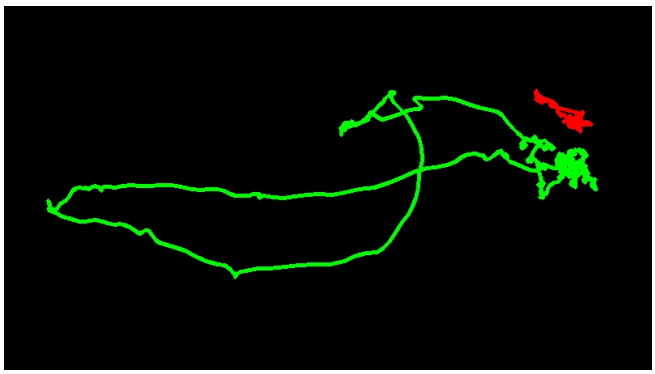

Figure 28: Vid 15 - Steven's track comparison with annotation

Figure 29: Vid 15 - Brian's track comparison with annotation

Another issue with Tracking is when Tracker is switching between animals. This is visible in Kalman Tracker and we can see it in Fig. 30. This could be fixed with a simple heuristic.

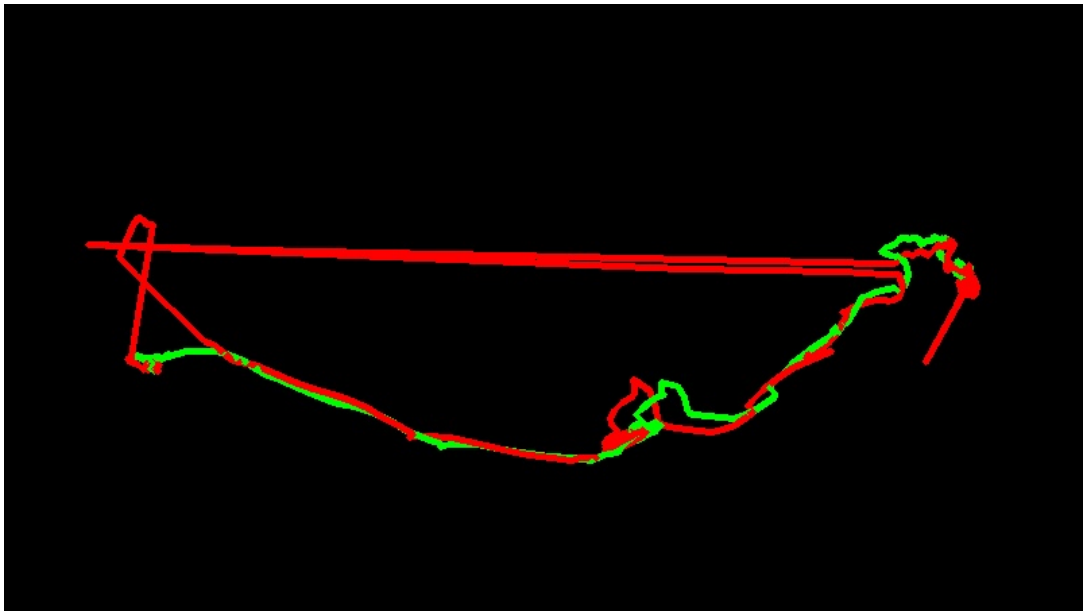

Figure 30: Vid 12 - Kalman Tracker switching classification.

To visualize square error for every frame instead of just comparing tracking with annotation we've generated an additional path with error included in the track's color. Each color threshold is defined in Fig. 14. Those thresholds are averaged in some interval (Fig. 31 shows values calculated per 10 frame interval).

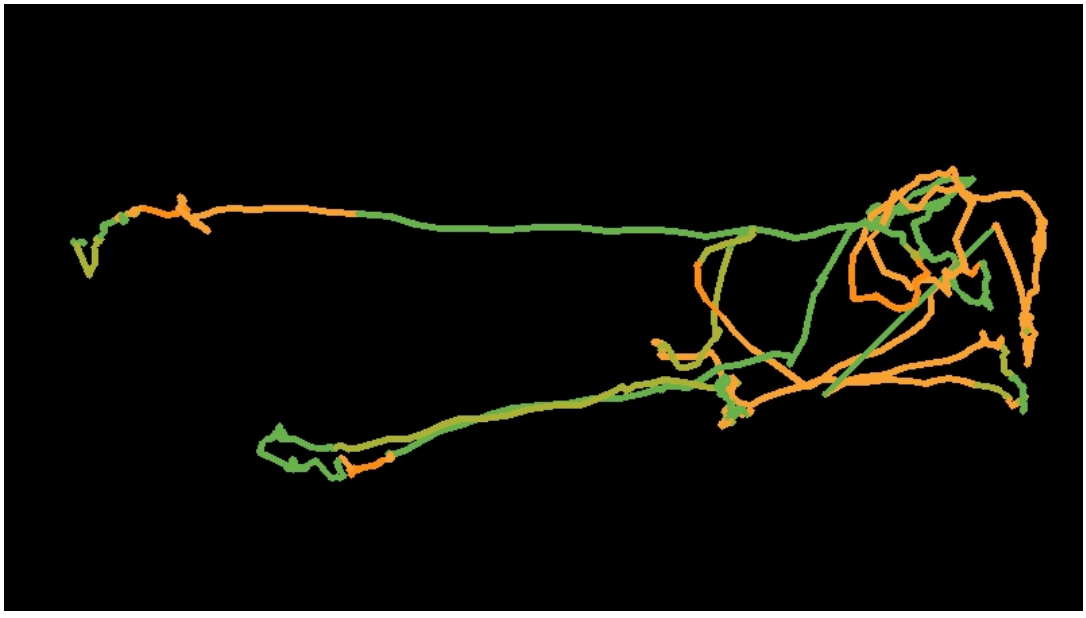

Figure 31: Vid 11 - Steven's track error per frame interval

7 Conclusions

Tracking with recognition is a very complex problem, it consists of multiple failure points and small errors in the early stages, accumulate to affect final tracking. Our approach to tracking was to use a similar solution used in human facial recognition which is a Siamese Network[5] on top of the detection network[9] with a custom tracking heuristic. We're surprised how well Siamese Network is dealing with different conditions (we didn't see a significant difference in classification error between day and night frames). Obtaining **F1 score of 0.9** (Table. 4) for 16 different classes is more than acceptable result. There was only one issue with the siamese network was that almost every network (except EfficientNetB5) was designing a separate cluster for outliers. We could probably improve tracking performance by classifying those outliers as outliers instead of the closest average class vector. We could probably also find a proper structure where that problem is marginal (right now is 10% or all test images).

7.1 Data

Because our dataset was not perfect and didn't have annotations for bounding boxes we had to generate those annotations. Manually annotating over 130k frames was not an option for 2 people. We generated augmented data for training but they were useful only when training detection model. Siamese networks weren't able to train properly with that amount of augmented data.

7.2 Detection

Having proper bounding box annotations would improve the detection model and produce better inputs for the recognition network. Because current SOTA solutions for object detections are good enough if we fine-tune them to our problem, even with inaccurate annotations detection is not the biggest problem. The only problem with detection is detecting the center of the animal. Because the bbox center is not exactly in the center of the tracked object it creates an issue for the tracking heuristic. Because of different conditions (day/night/angle/more than one animal inside the box) we cannot create an easy solution to correct it. One solution for that problem would be to extend the detection network with an additional structure and detect that center point (or even add additional postprocessing similar to PoseNet[4]).

7.3 Recognition

Siamese networks are widely used in recognition problems but usually, the images to recognize is a human face. We're searched for the usage of those networks in animal recognition based on top-view images. Unfortunately, we didn't find any papers on this subject. Siamese Networks were used in animal recognition but mostly still face recognition[13] and our problem is not similar to the presented paper. Our assumption was that animals like pigs have enough features for Siamese Network to distinguish between them just from the top-view image.

Training a Siamese Networks was the most difficult part of the whole research. Defining the final network structure was done base on trials and errors just by checking if the loss value is decreasing. We had to try multiple base structures and injection points for each base. We couldn't run the whole evaluation process for every layer of every base network and retrain everything to evaluate the result. Mainly because the training process for one configuration takes from 3 to 6 hours on GForce GTX 1080 Ti which we had available (we had also a second card with similar performance GForce RTX 2070 Super). Adding evaluation (30-45min) and assuming that every base has around 10-15 injection points, if we assume that every time we're training using the same hyperparameters it's around 192 hours of computation (8 days). That is just for one set of hyperparameters.

Dealing with EfficientNetB5 which had the worst results we've also noticed that network creates proper clusters but more than one per class (Fig. 19). This didn't play well with our evaluation because calculating an average value from similarly sized groups, puts the result value between them. This average value is not representing any of the clusters properly and caused low scores for EfficientNetB5 base. We could improve those results by calculating one of the clusterization metrics for those clusters and defining more than one point per class. We didn't want to change the evaluation for just that network but could be considered as an improvement for the future.

7.4 Tracking

Tracking evaluation was the biggest problem because we had to evaluate not only tracking but also classification. We produced two metrics but the results are not as great as we expected after seeing the evaluation of Siamese Networks. With that high F1 score and good detection, we thought that tracking will be easy. We evaluated 3 different trackers but only two of them (Avg. Embedding Tracking and Kalman Tracker) for both position and classification. Tracking was working in some cases but quite often (especially on videos where animals were running and covering each other), our trackers miss-classify some subjects. Included heuristics which should prevent random empty detections from messing with the results (you couldn't reassign bbox to element which is further than some ϵ value) prevented track corrections as we see in Fig. 28.

We think the problem with tracking is due to mostly defining bboxes when detecting animals. Because some of the boxes are too large (covering more than one animal), or too small (cutting off some animal's elements), a high classification score from the siamese evaluation was not that useful when recognizing cropped images. There is also a problem mentioned in the detection section, the center of the box is not the center of the animal which adds an initial tracking error that differs base on the animal's pose.

We had some ideas to improve that but it would require retraining siamese networks and could create different issues. Some of the ideas include image segmentation to extract particular animals from the surrounding but simple solutions had issues in the night images so we didn't include them in the training.

7.5 Final Thoughts

Because the whole solution is a complex problem that requires training multiple networks it's hard to approach it as a standard "search the hyperparameters" problem. To do that we would need to have extremely large resources and try hundreds of settings to improve the final solution. Despite that, we think that results are good enough to conclude that Siamese Networks could be used in animal recognition and tracking problem. There is a gap in the literature around this issue, mostly due to a lack of datasets. Our dataset is quite unique but still not perfect because annotations are not made for bounding box detection. We think that our solution could be easily transferred to another environment but with some restrictions.

- **similar camera distances** - we were training it on cameras that were placed only a couple of meters above the ground and might have a problem recognizing animals that are much smaller or larger.
- **similar camera angles** - videos from the dataset have different camera angles but they are falling into some acceptable range. Probably when recording video from the very acute angle it would not track animals correctly.
- **camera distortion** - all videos used to generate this dataset have similar lens distortion. Changing the value of that distortion is acceptable but switching to something like a fisheye lens would affect detection and recognition.

All of those potential problems could be solved, either by applying filters on the input image or just by retraining the network to fit a new situation. Using Siamese Network that produces vectors for each animal is a lot more flexible than standard classification, especially in environments when the number of animals is changing constantly. With this type of network, adding a new animal doesn't require retraining the whole network, just calculating the vector for that animal from some annotated photos (a job for 1 person). For commercial applications, this solution would need some additional tweaking in terms of approach to track but is a great start.

Appendices

A Detail Tracking Results

Appendix Title

Table 8: Tracking Error on **11_nursery_high_activity_day.mp4**

Table 9: Total

	DT	AET	KT
Anthony	0.39	0.81	0.40
Bob	0.29	0.29	0.29
Brian	0.33	0.30	0.35
Charles	0.40	0.41	0.37
Edward	0.36	0.36	0.34
Eric	0.03	0.08	0.05
Gary	0.38	0.36	0.37
George	0.28	2.05	0.41
James	0.28	0.26	0.35
Kevin	0.39	0.41	0.40
Larry	0.15	0.21	0.17
Paul	0.35	0.30	0.33
Robert	0.32	0.38	0.29
Scott	0.42	0.40	0.42
Steven	0.37	0.50	0.33
William	0.38	0.37	0.38
AVG	0.32	0.47	0.33

Table 10: Interval

	DT	AET	KT
Anthony	0.42	0.44	0.35
Bob	0.46	0.27	0.48
Brian	0.30	0.50	0.50
Charles	0.30	0.44	0.38
Edward	0.34	0.40	0.44
Eric	0.64	0.49	0.52
Gary	0.39	0.40	0.40
George	0.30	2.26	0.42
James	0.44	0.56	0.42
Kevin	0.31	0.38	0.41
Larry	0.21	0.50	0.62
Paul	0.43	0.34	0.39
Robert	0.53	0.44	0.48
Scott	0.43	0.35	0.57
Steven	0.43	0.24	0.42
William	0.51	0.34	0.50
AVG	0.40	0.52	0.46

Table 11: Tracking Error on **12_nursery_low_activity_day.mp4**

Table 12: Total

	DT	AET	KT
Anthony	0.05	0.05	0.05
Bob	0.63	0.65	0.63
Brian	-	-	-
Charles	0.14	0.20	0.20
Edward	0.31	0.63	0.39
Eric	0.13	0.05	0.05
Gary	0.57	0.57	0.57
George	0.04	0.04	0.03
James	0.63	0.93	0.64
Kevin	0.02	0.02	0.02
Larry	0.01	0.01	0.01
Paul	0.02	0.03	0.03
Robert	0.52	0.40	0.48
Scott	0.71	0.64	0.71
Steven	0.02	0.03	0.03
William	0.72	0.66	0.68
AVG	0.28	0.31	0.28

Table 13: Interval

	DT	AET	KT
Anthony	0.10	0.48	0.49
Bob	0.77	0.55	0.55
Brian	-	-	-
Charles	0.15	0.08	0.08
Edward	0.09	0.29	0.29
Eric	0.10	0.15	0.20
Gary	0.68	0.64	0.64
George	0.08	0.08	0.08
James	0.02	2.35	0.75
Kevin	0.02	0.02	0.02
Larry	0.00	0.00	0.00
Paul	0.77	0.02	0.02
Robert	0.01	0.01	0.71
Scott	0.25	0.01	0.01
Steven	0.14	0.07	0.07
William	0.83	0.82	0.83
AVG	0.25	0.35	0.30

Table 14: Tracking Error on **13_nursery_low_activity_night.mp4**

Table 15: Total

	DT	AET	KT
Anthony	0.01	0.01	0.01
Bob	0.11	0.08	0.08
Brian	0.13	0.11	0.11
Charles	0.05	0.06	0.07
Edward	0.04	0.06	0.06
Eric	0.21	0.19	0.18
Gary	0.08	0.07	0.07
George	0.08	0.07	0.07
James	0.19	0.19	0.19
Kevin	0.11	0.12	0.11
Larry	0.10	0.11	0.11
Paul	0.11	0.11	0.11
Robert	0.06	0.06	0.06
Scott	0.11	0.12	0.13
Steven	0.07	0.06	0.06
William	0.13	0.18	0.18
AVG	0.10	0.10	0.10

Table 16: Interval

	DT	AET	KT
Anthony	0.01	0.01	0.01
Bob	0.15	0.13	0.18
Brian	0.09	0.10	0.14
Charles	0.09	0.07	0.09
Edward	0.16	0.08	0.14
Eric	0.11	0.19	0.19
Gary	0.09	0.15	0.18
George	0.18	0.17	0.11
James	0.10	0.09	0.08
Kevin	0.07	0.18	0.14
Larry	0.11	0.10	0.15
Paul	0.12	0.09	0.14
Robert	0.06	0.06	0.11
Scott	0.15	0.12	0.06
Steven	0.08	0.05	0.11
William	0.11	0.07	0.13
AVG	0.10	0.10	0.12

Table 17: Tracking Error on **14_nursery_medium_activity_day.mp4**

Table 18: Total

	DT	AET	KT
Anthony	0.43	1.32	0.38
Bob	0.38	0.40	0.39
Brian	0.49	0.48	0.49
Charles	0.30	0.33	0.31
Edward	0.18	0.20	0.16
Eric	0.40	0.40	0.44
Gary	0.17	0.16	0.17
George	0.25	0.26	0.28
James	0.21	0.21	0.21
Kevin	0.44	1.25	0.44
Larry	0.25	0.42	0.20
Paul	0.17	0.25	0.14
Robert	0.39	0.50	0.40
Scott	0.37	0.37	0.39
Steven	0.23	0.23	0.23
William	0.34	0.44	0.34
AVG	0.31	0.45	0.31

Table 19: Interval

	DT	AET	KT
Anthony	0.41	2.23	0.52
Bob	0.30	0.56	0.42
Brian	0.35	0.35	0.35
Charles	0.34	0.34	0.34
Edward	0.51	0.22	0.32
Eric	0.40	0.14	0.30
Gary	0.12	0.12	0.25
George	0.55	0.52	0.52
James	0.43	0.28	0.23
Kevin	0.20	2.24	0.47
Larry	0.23	0.08	0.46
Paul	0.24	0.19	0.18
Robert	0.20	0.39	0.34
Scott	0.16	0.37	0.31
Steven	0.02	0.02	0.02
William	0.31	0.37	0.32
AVG	0.30	0.53	0.33

Table 20: Tracking Error on **15_nursery_medium_activity_night.mp4**

Table 21: Total

	DT	AET	KT
Anthony	0.19	0.25	0.21
Bob	0.24	0.23	0.24
Brian	0.32	0.27	0.31
Charles	0.22	0.21	0.22
Edward	0.25	1.78	0.43
Eric	0.19	0.19	0.16
Gary	0.32	0.26	0.30
George	0.18	1.75	0.27
James	0.27	0.12	0.15
Kevin	0.24	0.28	0.27
Larry	0.25	0.23	0.24
Paul	0.18	0.20	0.19
Robert	0.14	0.15	0.15
Scott	0.27	0.26	0.27
Steven	0.34	0.35	0.35
William	0.19	0.20	0.19
AVG	0.24	0.42	0.25

Table 22: Interval

	DT	AET	KT
Anthony	0.23	0.26	0.31
Bob	0.22	0.34	0.28
Brian	0.19	0.02	0.24
Charles	0.24	0.15	0.16
Edward	0.20	2.33	0.10
Eric	0.21	0.19	0.19
Gary	0.28	0.29	0.22
George	0.11	2.32	0.29
James	0.24	0.01	0.07
Kevin	0.17	0.12	0.23
Larry	0.09	0.22	0.21
Paul	0.02	0.23	0.28
Robert	0.27	0.32	0.27
Scott	0.16	0.33	0.33
Steven	0.37	0.41	0.24
William	0.17	0.21	0.25
AVG	0.20	0.48	0.23

B File Attachments

B.1 Siamese Network structures

Structures are available in attached files:

- **EfficientNetB5_model_fig.png** - EfficientNetB5 base Siamese Network
- **MobileNetV2_model_fig.png** - MobileNetV2 base Siamese Network
- **ResNet101V2_model_fig.png** - ResNet101V2 base Siamese Network

B.2 Example video with tracking

Example video with tracking information is available as file attachment **test01-2200_out.mp4**.

References

[1] Navaneeth Bodla et al. *Soft-NMS – Improving Object Detection With One Line of Code*. 2017. arXiv: 1704.04503 [cs.CV].

[2] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[3] Jonathan Huang et al. *Speed/accuracy trade-offs for modern convolutional object detectors*. 2017. arXiv: 1611.10012 [cs.CV].

[4] Alex Kendall, Matthew Grimes, and Roberto Cipolla. *PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization*. 2016. arXiv: 1505.07427 [cs.CV].

[5] Gregory R. Koch. "Siamese Neural Networks for One-Shot Image Recognition". In: (2015). URL: https://www.cs.utoronto.ca/~gkoch/files/msc-thesis.pdf.

[6] T. Lin et al. "Feature Pyramid Networks for Object Detection". In: (July 2017), pp. 936–944. ISSN: 1063-6919. DOI: 10.1109/CVPR.2017.106. URL: https://doi.ieeecomputersociety.org/10.1109/CVPR.2017.106.

[7] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2020. URL: https://cocodataset.org/#detection-2020.

[8] Leland McInnes, John Healy, and James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. 2020. arXiv: 1802.03426 [stat.ML].

[9] Y. Pang et al. "Efficient Featurized Image Pyramid Network for Single Shot Detector". In: (2019), pp. 7328–7336. DOI: 10.1109/CVPR.2019.00751. URL: https://openaccess.thecvf.com/content_CVPR_2019/papers/Pang_Efficient_Featurized_Image_Pyramid_Network_for_Single_Shot_Detector_CVPR_2019_paper.pdf.

[10] *Perceptual Systems Research Group - University of Nebraska*. URL: http://psrg.unl.edu/Projects/Details/12-Animal-Tracking.

[11] Herbert Robbins and Sutton Monro. "A Stochastic Approximation Method". In: *Ann. Math. Statist.* 22.3 (Sept. 1951), pp. 400–407. DOI: 10.1214/aoms/1177729586. URL: https://doi.org/10.1214/aoms/1177729586.

[12] Mark Sandler et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. arXiv: 1801.04381 [cs.CV].

[13] Stefan Schneider et al. *Similarity Learning Networks for Animal Individual Re-Identification - Beyond the Capabilities of a Human Observer*. Feb. 2019.

[14] Florian Schroff, D. Kalenichenko, and J. Philbin. "FaceNet: A unified embedding for face recognition and clustering". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 815–823.

[15] Mingxing Tan and Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. arXiv: 1905.11946 [cs.LG].

[16] Tensorflow. *Embedding Projector - Web Access Jan 2021*. URL: https://projector.tensorflow.org/.

[17] Wikipedia. *Confusion Matrix - Web Access Jan 2021*. URL: https://en.wikipedia.org/wiki/Confusion_matrix.

[18] Wikipedia. *F-score - Web Access Jan 2021*. URL: https://en.wikipedia.org/wiki/F-score.

[19] Wikipedia. *Mean Square Error - Web Access Jan 2021*. URL: https://en.wikipedia.org/wiki/Mean_squared_error.