CS342 Report

For the second assignment I implemented multiple different CNN models and tried using histogram of oriented gradients, image thresholding and a variety of geometric transformations on the data to get more effective predictions using a MLP.

I found most success with my CNN model and using HoG by itself. I tried to increase the accuracy of my model using HoG after applying different geometric transformations, however this ultimately proved unsuccessful. I believe that this is only because of my HoG model massively overfitting the data, and I believe that it would be possible in the future to get a better score using HoG alongside a range of geometric transformations.

Feature Engineering

I decided I would try three different approaches here, HoG, thresholding and using different geometric transformations of the image.

I thought that one of the most important things would be to pick techniques that tried to emphasise the general shape of objects in the picture. Sometimes the fish were very visible in the middle of the picture, and stood out sharply with the background, other times they would be hidden in a dark corner. Due to the variation on the fish image's location and perspective, I also thought that geometric transformations would be useful to train the model to recognise these different perspectives.

I briefly tried many different engineering techniques other than the ones here, I tried multiple different corner detector models but none seemed to give useful data. I thought the watershed transformation and SIFT would be useful techniques however I was unable to get them working in a reasonable time.

Thresholding:

To begin with I just tried using basic binary thresholding using OpenCV. This failed to produce any sort of usable result; as you can see in the pictures it rarely even preserves the basic structure of the fish. I tried using the adaptive thresholding method, this seemed to produce better results, although it seemed to be fairly noisy and I was worried the MLP would mistake some of this noise for other objects, or that fish objects could become lost in the noise.
(See pictures at end)

The **blockSize** parameter is how many neighboring pixels is used in the thresholding calculation. The parameter **c** is the constant that's subtracted from the mean when the calculation is performed. I tried a range of different values on a MLP with the default configuration and one layer with 13398 neurons.

(blockSize,c)	Log loss	Training data size
(21,7)	21.5	690
(7,3)	23.72	690
(9,4)	6.39	690
(9,11)	7.88	690
(9,4)	4.9	1035

I tried some other values too but these were the ones that gave me the consistently best performance.

I tried adding in small amounts of blur, however this nearly always had a negative effect on the data. I also tried using mean adaptive thresholding, however this always performed worse.

Despite trying multiple different MLP models on this, I never got a satisfactory result, always achieving a high log loss which was rarely better than my MLP using raw pixel values, so I didn't use this any further.

HoG

My next feature engineering technique was to use HoG, from very simple research it appeared to be quite useful in solving similar problems. I hoped this would help the MLP recognise the different features of fish, however I wasn't sure how effective it would be on the pictures where it appears the fish blends into the background. It seemed to me that HOG would be most effective when we had clearly defined outlines between the background and the image we wanted to recognise. I used skimage's HoG feature throughout the project.

Using just a default MLP with 11000 neurons, very quickly my model was completely overfitting the data, giving me a log loss of 0.62. HoG works by dividing the image into KxK cells, this is initially set to 8x8. Whilst I thought this was about right given the size of the fish, I tried changing it to 4x4. This gave me a log loss of 0.52, which was even worse. I checked the class predictions using a confusion matrix, whilst it falsely predicted multiple images as ALB, it seemed to delivery fairly accurate predictions.

Looking through the training data I noticed that lots of the images were essentially duplicates. I believe that Kaggle released training images from the same few boats, and then the test images were probably different boats. I suspected that my model was very good at noticing this, and not much else.

I tried to stop this overfitting in multiple different ways by altering my MLP parameters, however I wasn't successful. I thought that HoG seemed like it would deliver good performance if I managed to stop the overfitting, so I decided I would try introducing geometric transformations for my third image technique in order to introduce the model to many different perspectives of fish images.

Geometric Transformations

I wanted to reduce the degree of overfitting I was getting with HoG, I thought that using geometric transformations to rotate and skew the image would help me achieve this. I used an MLP with default parameters and put the translated images through skimage.HoG.

I mostly used two types of transformations, perspective changes and rotations. I used OpenCV warpPerspective transformations for the perspective changes. I always started by selecting the four corners of the image and moving them in some way. I found that I got better predictions when I kept this transformation small, higher values generally tended to make my model overpredict ALB.

The second type of transformation was simply rotating the image. I originally tried to do this with 8 different angles incrementing in 45^{o}, however this required me to make the image a square with big borders which made the MLP perform significantly worse. I ended up just using 90^{o}, 180^{o}, 270^{o} rotations. This generally seemed to have a positive impact on my models, at the cost of making convergence to a solution much slower.

Machine Learning Models

MLP

My initial MLP was just a MLP Classifier using the default configuration. I calculated the log loss using cross validation.

Change Made	Training Data	Log Loss
	64*64, RGB flattened array	20+
Increased to 4096 nodes on first layer	64*64, RGB flattened array	20+
5184 nodes on first layer	96*54, RGB flattened array	20+
Added another layer with 54 nodes	96*54, RGB flattened array	Did not converge in 3 hours

15552 nodes on first layer	144*77, RGB flattened array	17
Changed to grayscale array	144*77, Grayscale array	20+
Set alpha = 0.001, initial learning rate to 0.0001	144*77 Grayscale array	13
Set alpha = 0.0001, learning rate to 0.0005	144*77 Grayscale array	7.94
Increased the amount of data from 690 to 1600	144*77 Grayscale array	4.34

I tested lots more after this but never found a way to improve past 4 log loss on my validation. My best score on Kaggle was **2.89568**.

MLP with Feature Engineering:
I tried many different combinations of HoG and geometric transformations, as image thresholding never produced anything substantially better than using default pixel values. As I am limited by space I will not list the changes I made to MLP parameters, however I used cross validation to find the optimal values.

My first approach was to use a small perspective shift of 2 pixels on half the data, and keep the other half normal. I would then apply HoG and fit the MLP using this data. This produced a log loss of 1.16.

I tried to increase the amount I shifted the perspective to 5 pixels, this produced a log loss of 1.8 and made the MLP frequently overpredict ALB.

My next approach was to try using a small perspective shift on half the data again, then rotate half the data by 180^{o}. I managed to get a log loss of 1.3 from this.

I tried only rotating half the data with no perspective shift. This produced a log loss of 0.8. At this point I was curious as to which would perform better on Kaggle so I made my first predictions.

Rotation only: **2.34300**
Rotation and perspective change: **2.67229**

Rotation only seemed slightly better at this stage. I decided to try again, this time with the entire training dataset and a different model to see if using rotations only was definitely better.

My first rotated a quarter of the data by 90^{o}, a quarter by 180^{o}… and applied HoG afterwards. I used CV to get the best possible log loss from this and got 1.1. Worse than just rotating by 180^{o},

but it seemed to converge much more slowly so I was hoping it would have overfitted the training data less. The second was the same as before, just with 4 times more data.

Rotation only:	**2.20908**
Rotation and perspective change:	**2.32302**

Using HoG by itself overfitted the data massively, giving me a log loss of around 0.5 to 0.6. I wanted to see what sort of score this would get on Kaggle so I submitted a test file. To my surprise this got a score of **1.46238**. I ran it again with a higher resolution image and much more data, and got a score of **1.40867**.

CNN

For my first CNN model I just implemented 2 basic convolutional layers followed by a dense layer. This immediately converged after 2 epochs with a log loss of 10.

I learnt about dropout layers and decided I would try adding them after the dense layers to see if it stopped my CNN converging instantly. It did, but now the loss per epoch got higher as my model ran for longer. I tried adding 2 Convolution2D layers of size 128 and this helped performance massively, it converged to around 2.1 loss per epoch and my log loss decreased to 4.1.

I tried adding a BatchNormalization layer to my CNN. This immediately dropped the loss to 1.8 in the first epoch and by the 10th it had decreased to 1.31. My total log loss on my validation set went down to 1.19. I knew this was probably overfitting but it seemed like a much better result. When I looked at the outputted probabilities it seemed like it was overfitting ALB, nearly always giving it a >0.6 probability. I tried switching to softmax activation to fix this, this made the CNN overfit and made the loss decrease very rapidly, however I got a Log Loss of 1.1. I submitted my model to Kaggle and got a score of **1.50595.**

I tried adding a GaussianNoise layer and increased the number of dropout layers, hoping this would stop it overfitting the training images. This gave me a score on kaggle of **1.44407.**

I varied the number of layers and the parameters of my layers a significant amount, the most impactful change I made was increasing the size of the first Dense layer to 1024 and adding a Convolution2D layer of size 256. With this, I managed to get a score of **1.36662.** The most surprising thing about this was one Convolution2D per layer nearly always seemed to outperform two.

Conclusion

I think that one of the main challenges when it comes to this problem is getting your model to fit to the images of the fish themselves rather than fitting to the whole image. If I were to spend more time on this project, my method would be:

- Use something like Watershed to split the images up into different objects.
- Use a CNN to try to predict the class of the image based on this object.
- Use this data to improve object detection.

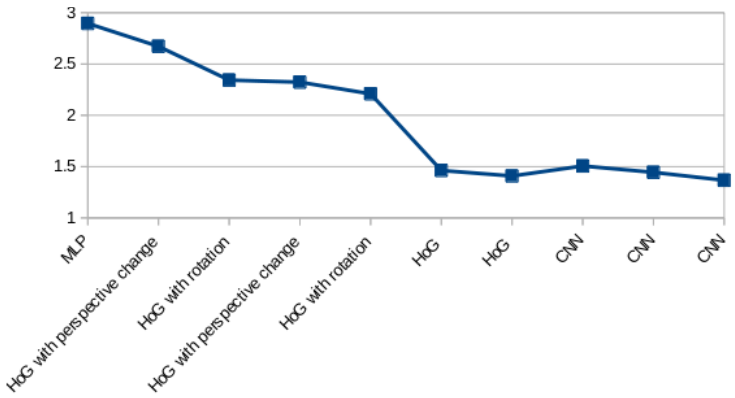
Whilst I don't know if this specific method would work, I feel like using the whole image to make predictions will never achieve a high performance on this task.

Whilst I could not achieve a higher performance on my MLP than by using HoG by itself, I believe using geometric translations would produce a better overall model. I think this could be achieved by fitting the data more closely, or inputting all 3786 training images 4 times, once at every orientation.

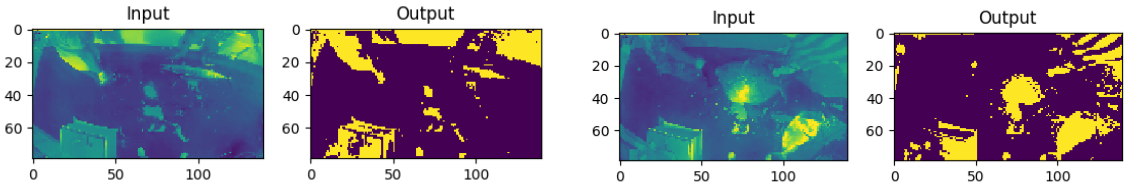
My final best Kaggle scores for each section:

mlp.py:	2.89568
Hog.py:	1.40867
finalcnn.py:	**1.36662**

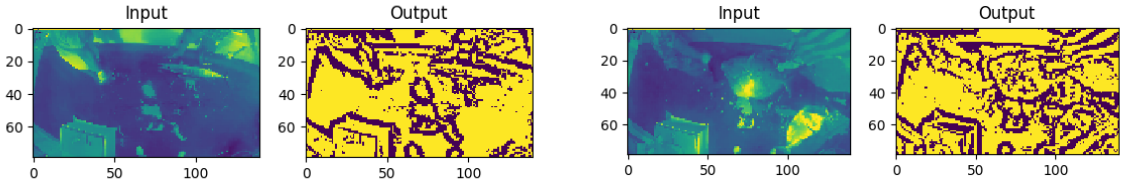
My Kaggle display name is cs325_1401663



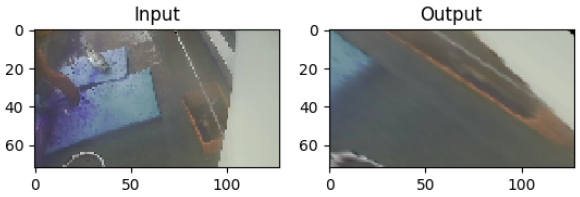
Basic thresholding



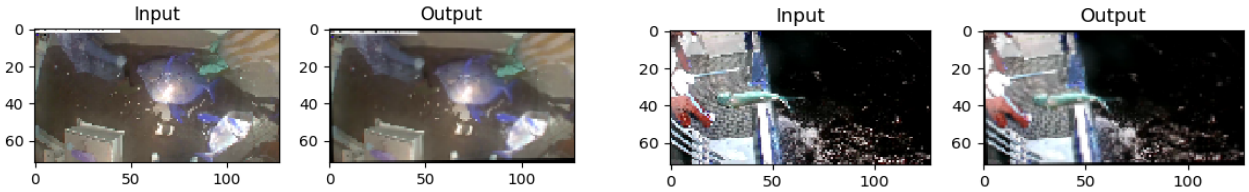
Using adaptive gaussian thresholding



Example of a bad transformation, cutting the fish out the picture.



The first perspective transformations I used.



The second perspective transformations I used.