Applied Deep Learning Coursework

Ainesh Sevellaraja
as17284@bristol.ac.uk

Karthik Sridhar
ks17226@bristol.ac.uk

Chanthru Uthaya-Shankar
cu17780@bristol.ac.uk

***Abstract*—Pan et al introduced two approaches to saliency prediction using convolutional neural networks, formulating it as an end-to-end regression problem. In this paper, we implement their shallow network and attempt to replicate their quantitative and qualitative results. We also consider one change to the original architecture to improve our results.**

***Index Terms*—deep learning, CNN, visual saliency, neural networks**

I. Introduction

Saliency prediction in images has traditionally been addressed with hand-crafted features based on neuroscience principles. Recent works use convolutional neural networks (CNNs), which are widely-used in object detection. The use of them in saliency prediction has two challenges not present in object detection, which are collecting large training datasets, and assigning a saliency score to each pixel of an input image.

Pan et al [1] provides two completely data-driven approaches of predicting salient regions in images using CNNs. They addressed this as an end-to-end regression problem, and used large datasets of annotated data for saliency prediction collected through crowdsourcing. Their objective was to compute images' saliency maps, which encode the probability of visual attention, defined as the eye-gaze fixation points. They proposed a shallow CNN, which was trained from scratch, and a deep CNN, which reuses parameters previously-trained for classification.

II. Related Work

The two key areas Pan et al addresses are: using convolutional neural networks to generate saliency maps for coloured images; and training it as an end to end regression problem. In this section, we discuss more recent publishings in this field and how the solutions have evolved with the advances in deep learning and availability of data.

One such approach to the use case was seen in **Visual Saliency Prediction Based on DeepLearning** [2]. It proposes a deep learning encoder-decoder architecture which is based on a transfer learning technique to predict visual saliency. The encoder stage consists of five convolutional blocks which are learned by down sampling (applying various receptive field sizes) to create the feature maps. The decoder stage also consists of five deconvolution blocks, which help sample the feature maps and generate an output of the same size as the input. The convolutional blocks of the proposed model use the VGG-16 network [3] to help predict the label for every pixel in the input image. The training was done on the SALICON dataset and was evaluated over several other datasets including TORONTO, MIT300 and DUT-OMRON [4], eventually achieving a **96.22%** global accuracy for the prediction of visual saliency.

Saliency Prediction in the Deep Learning Era [5] by Ali Borji reviews image and video saliency models over two image benchmarks; MIT (gold standard) and SALICON as well as large-scale video datasets. They include DIEM [6], HOLLYWOOD-2 and UCF-Sports [7]. He primarily explores the procedure of training deep saliency models on large image datasets followed by fine tuning them on smaller eye movement or click datasets. This proved to be more flexible as the semantic visual knowledge learned in CNNs were effectively reused and transferred to the task of saliency. He explored various deep saliency models on both MIT300 and SALICON datasets. However, the following two were the best when compared with our metrics results:

- **DeepGaze** [8] - A relatively deeper CNN used for saliency prediction where the outputs of the convolutional layers were used to train a linear model to compute image salience.
- **ML-Net** [9] - Unlike the usual convention of using the features at the final CNN layer, this model combines feature maps extracted from different levels of the VGG network to compute saliency. It learns a set of Gaussian parameters as opposed to a fixed Gaussian to fill convolutional layers, whilst using a different loss function which satisfies three objectives: to better measure similarity with the ground truth, to make prediction maps invariant to their maximum, and to give higher weights to locations with higher fixation probability.

III. Dataset

The SALICON dataset [10] was one of the datasets used to train and evaluate Pan et al's models. It is the largest available dataset for saliency prediction. It is based on the popular Microsoft Common Objects in context (MS COCO) image database [11]. MS COCO contains 10,000 training, 5,000 validating and 5,000 testing images, where every image has a coloured resolution of 480 x 640 pixels. The data in SALICON was collected through a new paradigm that allowed using general-purpose mouse instead of eye tracker to record viewing behaviours.

The SALICON dataset we used is split into two sections; 80% of it is intended for training (20,000 data points) and the remaining 20% (500 data points) for periodic validation. Each data point is represented as a Python dictionary. The contents

inside the respective sections are encoded in the dictionaries as follows:

$$Train = \{X, y, f\}$$

$$Test = \{X, y, y_o, X_o, f\}$$

where X_o is the original coloured 480x640 image, X is the scaled down image (96x96) of the original, y_o is the original 480x640 saliency map, y is the corresponding 48x48 saliency map and f is the respective file name of the image.

Since the datasets are relatively large to store and move as CSVs, they were stored as pickle (.pkl) [12] binary files. Python's pickle module was used to load the train and test sets. We also utilised this module to save the model's saliency predictions for further evaluation and visualisation, as described in section VI of this paper.

IV. Input

Visual saliency is the distinct subjective perceptual quality which makes some items in the world stand out from their neighbours and immediately grab our attention [13]. This property is evident in images, where the presence of a visually-salient object drives our attention towards that object's region in the image. The saliency of an object depends on various factors, such as its brightness and contrast relative to other regions of the image.

A saliency map encodes this visual saliency topographically by assigning a saliency score to each pixel of the original image, from a range of 0 to 1. Low saliency scores are given to non-salient regions, which contrast salient objects that have high scores. These scores can be transformed to lie between the range of 0 to 255 to be viewed as an image alongside its input image. Figure 1 shows an image with its saliency map. The bright region in the saliency map corresponds to the most salient region in the image, which is the man's face, as our attention is highly attracted to facial features.

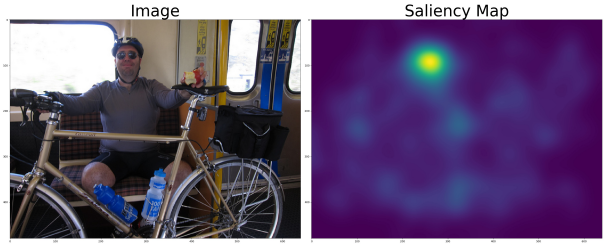

Fig. 1. An image from the test dataset with its ground truth saliency map

V. Shallow Architecture

The architecture of Pan et al's shallow network is presented in Table I. It has three convolutional layers, each followed by a rectified linear unit non-linearity (ReLU) and a max pooling layer. This then feeds into a fully-connected layer. The output of this layer is split into equal halves and fed into a maxout layer, which takes the element-wise maximum of the two halves. Finally, a second fully-connected layer is applied. Pan et al also used norm constraint regularization for the maxout layers to reduce overfitting.

TABLE I
Shallow architecture as described by Pan et al

Layer	Output Size	Details
Input	96x96x3	
Convolutional 1	96x96x32	Kernel 5x5x32
Max Pooling 1	48x48x32	Kernel 2x2, stride 2
Convolutional 2	48x48x64	Kernel 3x3x64
Max Pooling 2	23x23x64	Kernel 3x3, stride 2
Convolutional 3	23x23x128	Kernel 3x3x128
Max Pooling 3	11x11x128	Kernel 3x3, stride 2
Fully-Connected 1	4608	11x11x128 flattened to 15488
Maxout	2304	Element-wise between each half
Fully-Connected 2	2304	

VI. Implementation Details

We started our implementation by building the architecture of the CNN in Python with Pytorch, according to Table I. We created a file called `CNN.py` which holds the CNN class. The constructor creates and initialises each layer, and the forward function performs the forward pass. We initialised the weights of each layer according to a Gaussian distribution of mean 0 and standard deviation 0.01, and the bias to 0.1. We applied a padding of 2 for the first convolutional layer, and a padding of 1 for the second and third convolutional layers. This prevents the size of the input from decreasing when passed through these layers. We also did not apply ReLU after the first fully-connected layer.

We then created a training file, `train.py`, which takes in command-line arguments to instantiate the CNN object, optimizer and cost function, and load in the train and test datasets. We used Stochastic Gradient Descent (SGD) as the optimiser and Mean-Squared Error Loss (MSELoss) as the loss function. We created a Trainer class which holds a train method to train the model for a specified number of epochs, and a validate method to test the model on the test dataset every few epochs. During training, we print out metrics regarding the training progress such as the current epoch and batch loss. We also log the train and test loss at each step using Tensorboard's `SummaryWriter` to visualise the train and test curves after training. In addition, we save the model parameters every few epochs in case training stops midway.

Next, we wrote a testing file, `test.py`, which loads our saved model parameters, runs a forward pass with the test dataset and saves the predictions to a pickle file. These predictions can be loaded into `evaluation.py` and `visualisation.py` to calculate the saliency metrics of our model and visualise three saliency map predictions alongside the input image and ground truth.

We trained our model on the SALICON train dataset for 1000 epochs using a batch size of 128 and linear learning rate decay from 0.03 to 0.0001. We periodically calculated the test loss using the test dataset. We used L2 weight decay with a value of 0.0005 and Nesterov momentum with a value of

0.9. We utilised BlueCrystal4's GPUs for training which took around 3.5 hours.

VII. Replicating Quantitative Results

TABLE II
THE RESULTS OBTAINED BY PAN ET AL, OUR ORIGINAL AND IMPROVED IMPLEMENTATIONS

Salicon (val.)	CC	AUC Shuffled	AUC Borji
Pan et al	0.58	0.67	0.83
Our implementation	0.66	0.55	0.72
Improved implementation	0.71	0.57	0.73

Table II displays the quantitative results of our trained network compared to the results achieved by Pan et al, as well as our improved results obtained in section X. Our model obtained worse results for the AUC Shuffled and AUC Borji metrics, but performed better in the CC metric.

VIII. Training Curves

Fig. 2. Training (red) and testing (blue) loss curves of our replication

The train and test loss curves of our model's training process are shown in Figure 2. Both the train and test loss decrease steadily until around time step 60k (epoch 400) before remaining relatively constant until the end of training. There is minimal overfitting in our model as the testing curve does not diverge from the training curve by a large amount. Hence, our model is generalising. The training loss of our model does not reach zero, indicating that it does not train to completion.

The training curve exhibits noise due to the train loss of consecutive batches differing by a considerable amount. This is because our model overfits to some train batches, causing following batches to produce a high train loss. This results in an unstable, fluctuating training curve.

The filters learnt by the first convolutional layer are shown in Figure 3. Most of the filters contain similar dark regions towards the bottom, indicating that our model extracted this feature most out of all features present in the training set. The lack of variation in the filters is due to our model generalising, as it picks a more general feature rather than picking more specific ones that may not be present in the test data. Another reason that we obtained these filters is that our model did not fully learn from the train data since its train loss stagnates

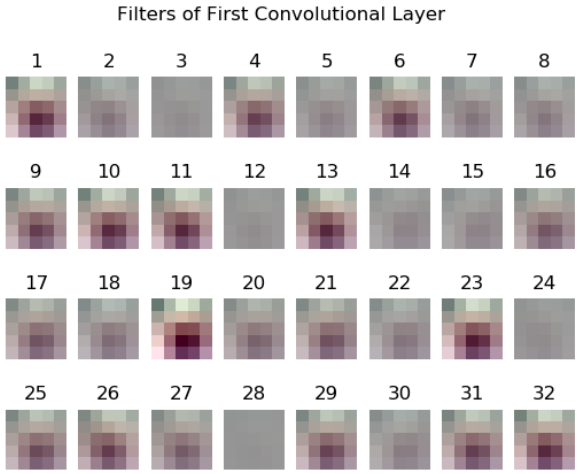

Fig. 3. Filters learnt by the first convolutional layer

during training, so it is unable to extract more meaningful features from the data.

IX. Qualitative Results

The saliency predictions of our model contain strong centre bias. Most of them have a salient region in the centre which gradually becomes less salient towards the four sides of the saliency map. This is because our model is generalising the task. Most images contain a salient region in the centre, so producing an output as such makes the model perform equally well on the train and test sets. Our model also does not fully exploit the train data, so this centre bias may have been the most prominent attribute it learnt during training.

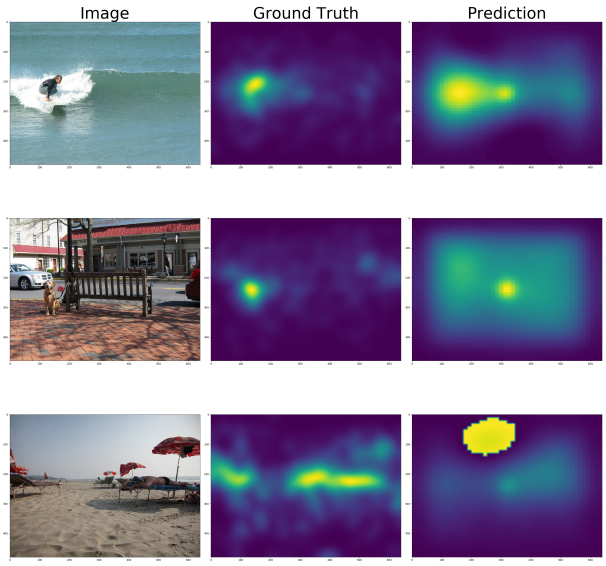

Fig. 4. Our model's predictions on three images from the test dataset

Figure 4 presents three predictions made by our model, alongside the input image and ground truth saliency map. The

first prediction is similar to the ground truth despite displaying slight centre bias. This accurate prediction is likely due to the salient region being distinctively brighter than the rest of the image, causing our model to detect it.

Our model does not perform well in the second and third predictions. In the former, the ground truth salient region is towards the left whereas our model predicts it to be in the centre, due to its centre bias. This is possibly because the salient region is less distinguishable than in the first image. The third prediction contains a sharp, bright spot in a region with a low ground truth saliency score. This is because our model is predicting a value slightly below zero for these pixels. Hence, due to overflow when cast as an unsigned integer, these pixels' scores are set to a value close to 255 when visualising the saliency maps.

X. Improvements

To improve our model's performance, we applied batch normalisation to the output of each convolutional layer and the first fully-connected layer. For the convolutional layers, we applied batch normalisation before their respective ReLU non-linearities. Our rationale behind adding these layers of normalisation is that our model fails to learn the task of saliency prediction fully, since its train loss only decreases by a small amount during the training process. Learning slows down and effectively halts by time step 60k (around epoch 400) as the train loss remains constant. We aimed to make this train loss decrease further so that our model learns a better representation of the task from the train dataset and can make more accurate predictions.

Batch normalisation [14] is a technique that pushes networks to learn more effectively during training. Without this technique, the distribution of the input of each layer of a network often changes during training, which makes learning slower and more difficult. This phenomenon is referred to as a high internal covariate shift. Batch normalisation standardises the distribution of these inputs by normalising them, hence reducing the internal covariate shift. In the case of our model, normalising the output of each convolutional layer and the first fully-connected layer restricts the distribution of input to the next layer. We felt that this would improve our model's ability to learn.

To implement this improvement, we first duplicated the `CNN.py`, `train.py` and `test.py` files and named the duplicates `CNN_batchnorm.py`, `train_batchnorm.py` and `test_batchnorm.py` respectively. For batch normalisation, we used Pytorch's `BatchNorm2d` class for the convolutional layers and `BatchNorm1d` for the fully-connected layer. We modified our forward pass function to apply the normalisation after each of these layers, but before the ReLU units in the case of the convolutional layers.

The quantitative results of our model with batch normalisation were slightly better than our regular implementation. These results are presented in Table II. The improved model obtained values higher than our original model for all three evaluation metrics.

Fig. 5. Train and test loss curves of our improved model

The training and testing loss curves of our improved model are shown in Figure 5. It is noticeable that the training curve converges at a value lower than that of our original model, indicating that our improved model learnt more effectively from the train data. However, it suffers from slight overfitting as the testing curve diverges and remains at a slightly higher value by the end of training.

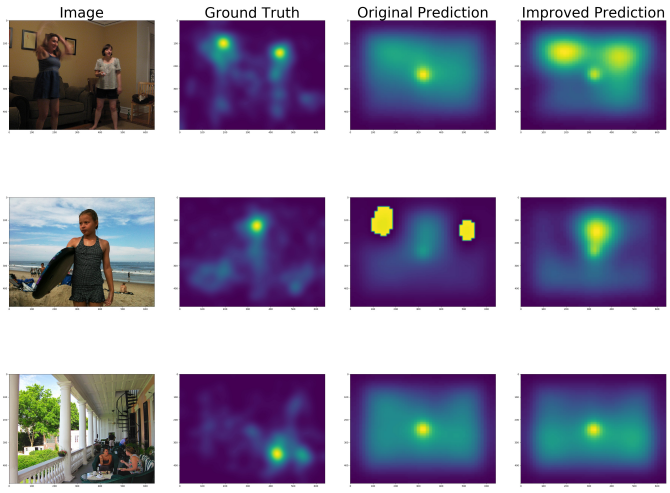

Fig. 6. Comparison of our original and improved models' performances on three images

Qualitatively, our improved model produces saliency maps much closer to the ground truth compared to our original model. Figure 6 displays the performance of both of our models on three input images from the test dataset. For the first image, although exhibiting slight centre bias, our improved model is able to detect both salient regions unlike our original model. Our improved model also performs well on the second image. The two bright spots in our original model's prediction are not present in the improved prediction, and the salient region is now detected. This is due to batch normalisation imposing a tighter constraint on the range of values the output prediction contains, reducing the likelihood of negative values. For the final image, our improved model is still unable to detect the salient region, however it can be observed that

a lower saliency score is assigned to the non-salient region towards the top of the image than in our original model.

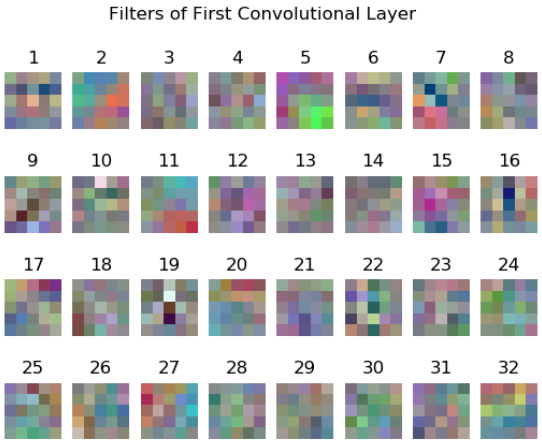

Fig. 7. Filters learnt by first convolutional layer of our improved model

This improved model also managed to learn a wider variety of filters for its first convolutional layer, as seen in Figure 7. Edge detectors and colour gradients can be identified. This indicates that the model is now able to extract more features from the train dataset.

XI. Conclusion and Future Work

In this paper, we aimed to implement a shallow convolutional neural network model proposed by Pan et al to predict salient features in images. The implementation is explained in detail and follows a systematic approach of data loading, network modelling, training, testing, evaluation and visualisation. The replication of the quantitative results on the SALICON dataset is achieved to a certain extent, particularly for the correlation coefficient (CC). The periodic logging of train and test losses assisted in making vital conclusions. The model initially generalised and did not fully learn from the train data. This was visible in the qualitative results, where a heavy centre bias was predicted. The addition of batch normalisation helped in improving the quantitative and qualitative results, making them closer to the results achieved by Pan et al.

Potential future work and improvements can initially focus on working with alternate datasets. This could include collecting and creating ground truths with different data augmentation techniques or using other ready-made datasets like iSUN and MIT300. Furthermore, the model can be facilitated for other similar tasks like scene classification and object detection. The proposed model provides the basis to develop newer models that are capable of learning from already existing networks like the VGG-16 architecture discussed above.

References

[1] Pan J, Sayrol E, Giro-i Nieto X, McGuinness K, O'Connor NE. 2016. Shallow and deep convolutional networks for saliency prediction. In: Proceedings of the IEEE conference on computer vision and pattern recognition. Piscataway: IEEE, 598–606.

[2] B. Ghariba, M. Shehata and P. McGuire, "Visual Saliency Prediction Based on Deep Learning," 2019. Information. 10. 257. 10.3390/info10080257.

[3] M. Hassan, *VGG16 – Convolutional Network for Classification and Detection*, 2019, https://neurohive.io/en/popular-networks/vgg16.

[4] Li, Y.; Hou, X.; Koch, C.; Rehg, J.M.; Yuille, A.L. The secrets of salient object segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; pp. 280–287.

[5] A. Borji, "Saliency Prediction in the Deep Learning Era: Successes, Limitations, and Future Challenges," 2019, arXiv: 1810.03716v3.

[6] P. K. Mital, T. J. Smith, R. L. Hill, and J. M. Henderson. (2011). "Clustering of Gaze During Dynamic Scene Viewing is Predicted by Motion". Cognitive Computation. 3. 5-24. 10.1007/s12559-010-9074-z.

[7] S. Mathe and C. Sminchisescu, "Actions in the Eye: Dynamic Gaze Datasets and Learnt Saliency Models for Visual Recognition," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 37, no. 7, pp. 1408-1424, 1 July 2015, doi: 10.1109/TPAMI.2014.2366154.

[8] M. Kummerer, L. Theis, and M. Bethge, "Deep Gaze I: Boosting saliency prediction with feature maps trained on imagenet," 2014, arXiv preprint arXiv:1411.1045.

[9] M. Cornia, L. Baraldi, G. Serra, and R. Cucchiara, "A deep multilevel network for saliency prediction," in ICPR, 2016, pp. 3488– 3493.

[10] M. Jiang, S. Huang, J. Duan, and Q. Zhao. SALICON: Saliency in context. In IEEE conference on Computer Vision and Pattern Recognition, 2015.

[11] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Doll´ar, C. L. Zitnick, Microsoft coco: Common objects in context, in: Proceedings of the European Conference on Computer Vision, 2014, pp. 740–755.

[12] Pickle module, Python 3 Documentation, https://docs.python.org/3/library/pickle.html.

[13] L. Itti, "Visual Salience," Scholarpedia. http://www.scholarpedia.org/article/Visual_salience.

[14] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167, 2015.