An exploration of Shallow Convolutional Networks for Saliency Prediction

1st Jacob Chalk
dept. Computer Science
University of Bristol
jc17360@bristol.ac.uk

2nd Owen Coyne
dept. Computer Science
University of Bristol
oc17838@bristol.ac.uk

I. Introduction

Image saliency generally refers to the recovery of the most important information within an image, however in this more specific context it aims to mimic the human visual attention system and thus find the key objects or aspects of an image scene. We distinguish that the saliency problem addressed here is one of "bottom-up" salience where the relative measure of information is not specific to some predefined task.

The prediction of salient areas of an image has long been an important task in computer vision and traditionally has been achieved through a range of hand-crafted feature maps [1]. Given the advent of large-scale annotated data sets for image saliency (SALICON, MIT300, CAT200 etc) it became a viable strategy to apply deep learning methods such as convolutional networks, more commonly used for image classification, to the problem at hand.

As such Pan et al. proposed a unique model [2] which contextualised image saliency as a regression problem and trained a shallow convolutional network end-to-end using the SALICON dataset.

II. Related Work

The work of Pan et al. takes its place within a wide range of methods which leverage Convolutional Neural Networks (CNNs) for image saliency prediction. Since its publication a number of new models have been proposed in the same vein that each uniquely extend this general structure to further improve their prediction. Herein we present a small selection that we have deemed noteworthy.

Pan et al. have more recently developed an adversarial network to generate samples of Saliency maps. This approach trains two networks: A generator network with a convolutional encoder-decoder architecture, which predicts (or generates) saliency maps from raw pixels of an input image and also a discriminator network, which evaluates the generators output to determine whether it is predicted (generated) or ground truth [3]. The two networks partake in a zero-sum game, with the aim of making the predicted saliency maps indistinguishable from the ground truth. The network achieved near state-of-the-art performance on the SALICON dataset specifically, however, its performance generalised well to a range of saliency prediction metrics and scenarios, in contrast with it's contemporaries [3].

EML-NET, as presented in Jia and Bruce's 2018 paper [4], proposes an encoder-decoder network where the encoder stage is made up of any number of CNN models whose weights can be trained separately for the goal of saliency prediction. This allows the use of convolutional nets, with differing architectures, in tandem and also provides an opportunity to use models pre-trained for other tasks. These pre-trained models can extract features that aid in saliency prediction in the form of face [5] or object recognition [6]. The decoder stage is then trained to combine the learned features from multiple layers of the different pre-trained CNNs. As such the method allows each of the encoding stages and decoding stage to be trained separately, massively reducing complexity, whilst allowing the elegant incorporation of features from a wide range of deep convolutional networks.

A novel approach was considered in the transformed domain methods of Jiang et al. [7], that exist within a subclass of methods who utilise the Discrete Fourier Transform of an image to recover image structure through its phase and amplitude spectrum's. Preceding the model, it was shown that the phase spectrum of an image contains the vast majority of "saliency cues" that aid in image saliency prediction, in comparison to the amplitude spectrum. As such a deep CNN was combined with an encoder-decoder network and a unique tripartite loss function to predict saliency maps, based on a complex combination of the features of the original image and it's phase and amplitude spectrum's. The method achieved state of the art results on the CAT200 data set but has yet to be tested within the context of SALICON.

III. Dataset

The dataset used for training and validation was the SALICON dataset, which was was built from images of the *Microsoft CoCo: Common Objects in Context* [8].

The saliency maps in this dataset were collected with mouse clicks captured in a crowd-sourcing campaign, rather than the usual eye tracking camera. [2] While this is an approximation, results show that mouse tracking is a good replacement of eye tracking in model evaluation [9] and the relative ease of this approximation allows for far larger quantities of annotated data to be gathered.

The training set, *train.pkl*, contained a list of 20000 data points. Each data point is represented as a dictionary which

consists of three keys:

- ***X*** **-** 3x96x96 training image
- ***y*** **-** 48x48 target saliency map
- ***file_name*** **-** filename of the image

The validation set, *val.pkl*, was a list of 500 data points. Again, each data point is represented as a dictionary. However here, each data point consists of five keys:

- ***X*** **-** 3x96x96 validation image
- ***y*** **-** 48x48 target saliency map
- ***y_original*** **-** original 480x640 saliency map, before down-scaling to 48x48
- ***X_original*** **-** original 3x480x640 training data before down-scaling
- ***file_name*** **-** filename of the image

The downsized *X* and *y* data were used for training. *X_original* and *y_original* were used for evaluation and visualisation. This left use with a training and validation set split size of: 20000 training images to 500 validation images.

Data is saved to a *.pkl* file by converting a Python object into a byte stream (pickling) and loaded (unpickling) by the inverse process. The *pickle* module implements binary protocols for serializing and deserializing a Python object structure to support these operations [10].

IV. Input

A saliency map is an image where the value of each pixel indicates areas that share common "salient" properties of an input image. In regards to the SALICON dataset, the salient property can be thought of as approximating the likelihood of eye fixation at a point in the image. As such the saliency map forms a distribution where the colour of each pixel is determined by the aggregation of participant fixation maps.

Given this, when viewing the saliency map, it acts to highlight the regions of the scene that draw viewer attention. This usually corresponds to key objects, colours or regions that are structurally important to the scene.

An example can be seen in Fig. 1, where the brightest regions of the saliency map correspond to objects at the center of the scene, the television and coffee table respectively, that are more likely to draw a viewer's gaze. Other, less prominent, objects in the fore and background of the image are also represented with a less intense saliency value in the map (e.g. the Christmas tree and background window). Areas which rarely draw visual attention, such as the blank walls and carpet, appear dark in the presented saliency map.

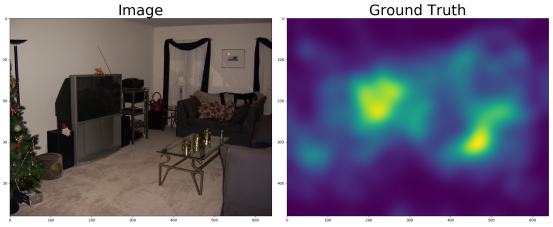

Fig. 1. An example image (left) with it's saliency map (right)

V. Shallow Architecture (Pan et al)

Table. I shows the shallow architecture described by Pan et al. [2]. The input consists of a 96x96 RGB image, proceeded by 3 rounds of convolution, where their outputs are passed through a rectified linear unit non-linearity (ReLU) activation function and max pooling layer. Following the third convolution/pooling round, the output is flattened to a 15488-sized vector and passed through a fully connected layer, which outputs a 4608-sized vector. This output is then sliced into two 2304-sized vectors and the maximum value between adjacent pairs of pixels is taken via the Maxout layer. Maxouts output is then passed through a second fully connected layer, which produces the 2304-sized feature output as the final prediction. The contrasting output after the third convolutional layer from Pan et al. ([11x11]vs [10x10]) is as a result of the image padding

TABLE I
Shallow Architecture (Pan et al.)

Layer	Filter Size, Count	Stride	Padding	Output Size
Image	-	-	-	96x96x3
Convolution 1	(5x5), 32	(1x1)	(1x1)	96x96x32
ReLU 1	-	-	-	96x96x32
Max-Pool 1	(2x2), 32	(2x2)	-	48x48x32
Convolution 2	(3x3), 64	(1x1)	(1x1)	48x48x64
ReLU 2	-	-	-	48x48x64
Max-Pool 2	(3x3), 64	(2x2)	-	23x23x64
Convolution 3	(3x3), 128	(1x1)	(1x1)	23x23x128
ReLU 3	-	-	-	23x23x128
Max-Pool 3	(3x3), 128	(2x2)	-	11x11x128
FC1	-	-	-	4608
Slice1, Slice 2	-	-	-	2x2304
Maxout	-	-	-	2304
FC2	-	-	-	2304

VI. Implementation Details

This section discusses the steps taken in order to replicate the shallow architecture in Table. I and achieve the bench-mark replicated results.

Our implementation was constructed using Python, NumPy, and the deep learning library PyTorch [11]. Processing was performed on a GPU Node on Blue Crystal 4, the University of Bristol's high performance computing device, where each node consists of two Nvidia P100 GPUs each with 3584 CUDA cores and 16GB GPU Memory.

Our initial steps in the replication of Pan et als. shallow architecture involved the creation of a model structure in PyTorch similar to Table. I. We began by defining a CNN model class, which could host all the necessary layers, and define the forward pass of the model. We replicated all convolutional layers, each followed by a ReLU activation function, and fully connected layers. After each round of convolution, a max pooling layer was used. We trained this model initially with a static learning rate of 0.03 for 10 epochs to establish a baseline of performance and to test the models input and output.

Given the ambiguity in the paper regarding the data slicing for the maxout layer, we implemented two different forms

of maxout layer. The first split the input tensor into two halves lengthwise and then took the element-wise maximum of each index of the two vectors. The second method found the maximum of each of the adjacent entries in the tensor, we applied this by splitting the tensor into 2 dimensions along the even and odd indexes. We then compared both methods and found that the former produced sub-optimal results, with a lower accuracy and an even stronger centre-bias, than that of it's counterpart.

Once the structure of the model was in place, we moved on to set up for training. This involved the introduction of a linearly decreasing learning rate that was initialised to 0.05 and was reduced after each epoch towards the target value of 0.0001. The choice of initial value was between the 0.03 specified in the paper and the 0.05 in the reference implementation provided by Pan et al. [2], after testing both we noticed an improved accuracy when using 0.05.

We also replicated the weight initialisation such that all weights were sampled from a Gaussian distribution with $(\mu = 0, \sigma = 0.01)$ and all biases set to 0.1. Our final action pre-training was the choice of Stochastic Gradient Descent as our optimisation function. To aid our optimiser, we make use of Nesterov Momentum which performs a look-ahead gradient evaluation to correct the momentum term [12]. This prevents the momentum from overshooting an optimum point, potentially resulting in oscillation.

Our optimiser was further aided by L2 Norm Regularisation using a weight decay of 0.0005, which heavily penalises extreme weights in the loss function. This reduces the complexity of the model and thus the generalisation error. The chosen loss function matched that of Pan et al. [2] which is the Mean Squared Error (MSE) loss function.

Once we had initialised the model, we trained the network on the 20000 images in the training set for 1000 epochs using a batch size of 128. We validated our results on the 500 images in validation set every 2 epochs. For validation, we ensured the model was put into validation mode. This ensures that the model does not perform a backwards pass on the data. For training, we ensure the model is in training mode so that it does perform the backwards pass. When logging the data, we took the average loss across the validation set for the test loss and the batch loss every 5 batches for the training loss.

We added a prediction frequency every 200 epochs which output the models current prediction to a *.pkl* file during training. This allowed for the models outputs to be visualised and evaluated on. To ensure that optimal values were not missed due to sparse prediction output, we also added a checkpoint frequency to store the model parameters as a dictionary every 50 epochs. This allowed us to load the model from a certain state/epoch to avoid having to retrain the entire network thus also making our implementation more robust to runtime errors.

VII. Replicating Quantitative Results

Table. II shows our implementation successfully achieves the benchmark replicated results. We exhibit small improvements to each metric, with improvements of 0.01 for AUC Shuffled and AUC Borji, and an improvement of 0.02 for CC. These improvements are likely due to the choice of us implementing the better performing learning rate of 0.05 from the reference implementation, as opposed to 0.03 from the paper.

TABLE II
Our Replicated Quantitative Results

SALICON (val)	CC	AUC Shuffled	AUC Borji
Pan el al [2016]	0.58	0.67	0.83
Benchmark replicated results	0.65	0.55	0.71
Our replication	0.67	0.56	0.72

VIII. Training Curves

Fig. 2 shows our training and testing loss curves. As the number of batches increases, both the testing and training loss exhibit a downwards trend. This shows that our model is generally improving at saliency prediction. However, we note that after the 30,000 batches mark, the gap between the training curve and testing curve begins to widen, this implies that our model is beginning to over-fit the training data. A continued run may even see that whilst the training loss continue to decrease, the testing loss' decline stagnates or even possibly increases.

The filters learned in the first convolutional layer after the final epoch are shown in Fig. 3. Each filter represents an output channel and each pixel within it represents a parameter for that channel. Darker pixels corresponding to a higher parameter weights. The filters bare resemblance to other filters learned for classification convolutional networks [13], such as clear edge detection filters. This could link to very simple saliency features within an image, such as edges of a certain orientation.

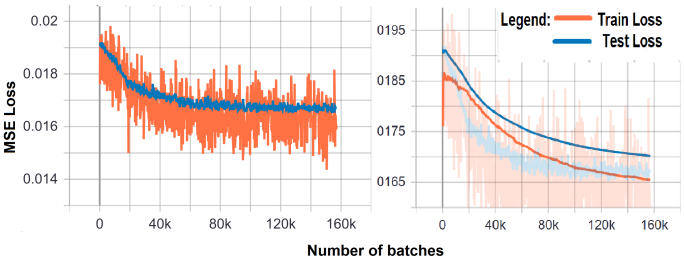

Fig. 2. Original (left) and smoothed (right) training (orange) and testing (blue) loss curves

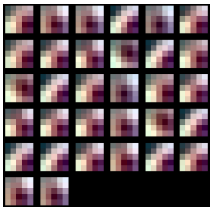

Fig. 3. Filters learnt in the first convolutional layer

IX. Qualitative Results

Fig. 4 demonstrates a strength in our model, where the predicted saliency map produces a similar shape to the ground truth, with a strong prediction in the left-of-centre region. However, Fig. 5 exhibits two modes of failure in our network. The first being the strong central bias of the model. As discussed by Pan et al. [2], the network over-fits to a central bias. However, it can be argued that the bias is implicit within the data, as photography and media tend to present subjects of interest near the center, thus humans implicitly learn over time to expect maximum information gain towards the centre, when prior information about the image is not known [14]. A deeper network would be needed to combat this [2]. The second example demonstrates a complete failure in the saliency prediction, with significant variance from the ground truth. These failures modes were repeated across a selection of images, which could be implicit of some error within the architecture. We noted that these failure modes occurred more frequently in images with backgrounds which are bright or contain complex features, for example images with a cloudy sky or those with bright snow. We hypothesised this could be related to the model unusually attributing saliency to these backgrounds due to some variance from the rest of the data.

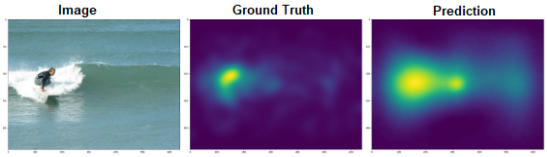

Fig. 4. An example of where our model performs well

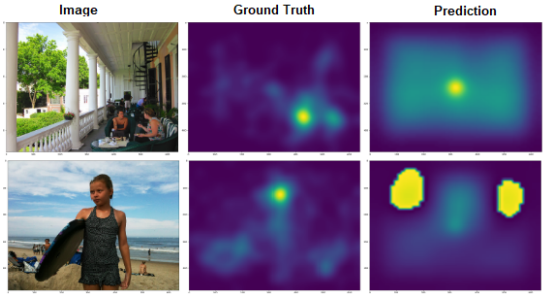

Fig. 5. Two examples showing two different modes of failure of out model

X. Improvements

The work of Jetley, Murray and Vig suggests that "visual attention is a fundamentally stochastic process due to it being a perceptual and therefore subjective phenomenon" [15] as such they propose that a more accurate model of a saliency map is one of a probability distribution where each pixel of the image holds a value that represents "the probability of that pixel being fixated upon" [15].

This unique model opens up a wide range of loss functions that can now be used within the context of image saliency mapping. The paper presents a few of these loss functions and shows their improved performance over euclidean distance, given a slightly deeper convolutional architecture not too dissimilar from our own.

We tested two of the presented distribution distance loss functions: the Bhattacharya distance and Kullback-Leibler (KL) divergence. PyTorch does not implicitly support a loss function for the Bhattacharya distance, thus we implemented our own. While Bhattacharya did improve the results, it was not as significant as the KL Divergence, contrasting to the work of Jetley et al. [15]. We believe this is due to the differences in architecture, where Jetleys is a deeper network which uses more convolutions for feature extraction while also initialising using the already trained weights from VGG Net [16]. Thus, our architecture is more suited to the KL divergence loss function.

In order to use distribution distance loss functions, the output feature vector must be a probability distribution itself. Thus, we extend the architecture shown in Table. I with a PyTorch SoftMax layer following the second fully connected layer. We apply this to both our prediction and the ground truth, where PyTorch utilises the following formula, where $\mathbf{x}^p$ is the prediction vector and $\mathbf{x}^g$ is the ground truth:

$$S(x_i^p) = \frac{exp(x_i^p)}{\sum_j exp(x_j^p)},\ S(x_i^g) = \frac{exp(x_i^g)}{\sum_j exp(x_j^g)} \tag{1}$$

This converts the values within the vectors into the range $[0, 1]$, thus transposing them to a probability distribution.

Furthering the implementation details, PyTorch's *KLDivLoss* requires the predictions to be log probabilities[1]. Thus, we pass the following to the loss function:

$$loss = KLDivLoss(ln(S(\mathbf{x}^p)), S(\mathbf{x}^g)) \tag{2}$$

As we are using a distribution distance measure, it is imperative that our implementation uses batch normalization. Without it, internal covariance shift may cause either the input or output distribution to shift, thus the loss function will not be fitting to the true underlying distribution of the data. Therefore, this is required as an implementation detail.

The reported results are shown in Table. III. We exhibit an increase in performance across all three metrics, with the most notable being in CC. This supports the work of Jetley et al., strengthening the idea that loss functions tailored to minimising the loss between distributions are more suitable than the loss functions based on Euclidean Distance found commonly in the literature. KL divergence is robust to outliers when compared to the Mean Squared Error, as it suppresses large differences between probabilities, whereas MSE accredits its abnormally large differences simply as "higher error", which is less effective at mitigating outliers. This robustness is significant and fundamental when evaluating on a subjective, noisy and highly variable process such as visual attention [15], as many outlying points may detract from the underlying distributions the model is trying to fit to. We believe that it is this property of the KL Divergence that provides us such

[1]PyTorch implicitly converts the ground truths into log probabilities, hence only a standard SoftMax layer is needed.

significant improvement. As an additional benefit, the optimal accuracy is reached after just 45 epochs, drastically reducing the training time compared to the mean squared loss.

Fig. 7 shows that our improvement avoided the common failure modes present in the original model as seen in Fig. 5. Not only did this reduce the central bias, but began accurately predicting the cases of previous total failure, which will have significantly contributed to our improved results. Fig. 6 also shows that the testing loss of our improved model far more accurately tracks the training loss over time, implying a significant reduction in overfitting.

TABLE III
OUR IMPROVED QUANTITATIVE RESULTS

SALICON (val)	CC	AUC Shuffled	AUC Borji
Pan el al [2016]	0.58	0.67	0.83
Benchmark replicated results	0.65	0.55	0.71
Our original replication	0.67	0.56	0.72
Our improved replication	0.76	0.60	0.74

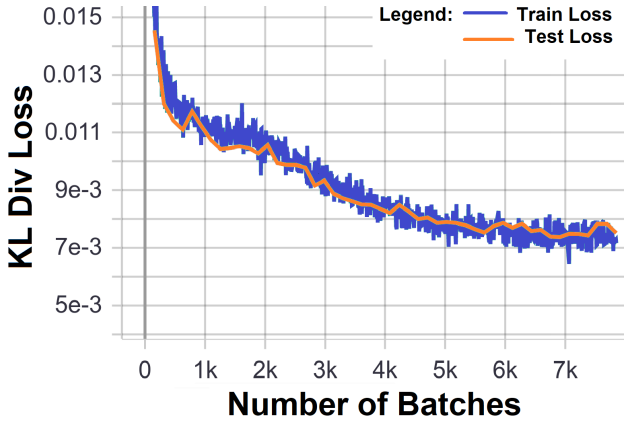

Fig. 6. Our improved training (Blue) and testing (Orange) loss curves.

Fig. 7. Our improved predictions on the failure cases shown in Fig. 5

XI. CONCLUSION AND FUTURE WORK

This report examines the re-implementation and evaluation of the saliency prediction model presented in Pan et al [2]. We chart the steps required to implement the network within the framework of the PyTorch library and through further analysis show both the strengths and failings of the original shallow architecture. We go on to show, inspired by the work of Jetley, Murray and Vig [15], that the use of a Kullback-Leibler loss function improves both the accuracy and training time of the network compared to the original model.

Any future stages of work would consist of creating a deeper architecture in order for more involved feature extraction from the images. This would help formulate a wider range of features which can be used in the final saliency prediction. Furthermore, we would extend the architecture to incorporate the features of CNNs trained for tasks that represent the more complex aspects of saliency prediction, such as face or object detection [5], [6].

REFERENCES

[1] A. Borji and L. Itti. State-of-the-art in visual attention modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):185–207, 2013.

[2] Junting Pan, Elisa Sayrol, Xavier Giro-i Nieto, Kevin McGuinness, and Noel E. O'Connor. Shallow and deep convolutional networks for saliency prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[3] Junting Pan, Cristian Canton-Ferrer, Kevin McGuinness, Noel E. O'Connor, Jordi Torres, Elisa Sayrol, and Xavier Giró-i-Nieto. Salgan: Visual saliency prediction with generative adversarial networks. *CoRR*, abs/1701.01081, 2017.

[4] Sen Jia. EML-NET: an expandable multi-layer network for saliency prediction. *CoRR*, abs/1805.01047, 2018.

[5] Moran Cerf, Jonathan Harel, Wolfgang Einhaeuser, and Christof Koch. Predicting human gaze using low-level saliency combined with face detection. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20, pages 241–248. Curran Associates, Inc., 2008.

[6] Jia Deng Olga Russakovsky, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[7] Lai Jiang, Zhe Wang, Mai Xu, and Zulin Wang. Image saliency prediction in transformed domain: A deep complex neural network method. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:8521–8528, Jul. 2019.

[8] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[9] Ming Jiang, Shengsheng Huang, Juanyong Duan, and Qi Zhao. Salicon: Saliency in context. In *In Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[10] Python Software Foundation. pickle — python object serialization, 2020. https://docs.python.org/3/library/pickle.html, Last accessed on 01-12-2020.

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 8026–8037. Curran Associates, Inc., 2019.

[12] Aleksandar Botev, Guy Lever, and David Barber. Nesterov's accelerated gradient and momentum as approximations to regularised update descent, 2016.

[13] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[14] P. H. Tseng, R. Carmi, I. G. M. Cameron, D. P. Munoz, and L. Itti. Quantifying center bias of observers in free viewing of dynamic natural scenes. *Journal of Vision*, 9(7):4–4, July 2009.

[15] Saumya Jetley, Naila Murray, and Eleonora Vig. End-to-end saliency mapping via probability distribution prediction. *CoRR*, abs/1804.01793, 2018.

[16] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.