Deep Encoder-Decoder Models for Saliency Prediction

Jirko Rubruck
Supervisor: Prof. Ben Tatler

April 2021ugust
Word count: 8032

1495

UNIVERSITY OF ABERDEEN

Abstract

Visual attention allows humans to effortlessly extract and process specific aspects of our visual environment. A quest for mechanisms underlying these processes has yielded a variety of computational models attempting to explain visual attention under different conditions. An emerging form of computer vision model — Convolutional Neural Networks (CNNs) — has recently been used with high success to predict allocation of attention in task-free bottom-up driven explorations of natural scenes. Inspired by research on representation learning and image segmentation we test two types of modifications to such models. First, we train a highly successful CNN for saliency prediction with different input corruptions to enhance usefulness and generalisability of learned features. Second, we developed a novel model for saliency prediction with encoder-decoder skip-connections (EDSC-Net) to overcome information loss caused by pooling operations in CNNs. We found that while models trained with corrupted images are still proficient in predicting attention allocation, they perform worse than their counterpart trained without such corruptions. Our EDSC-Net achieved state-of-the art performance marginally outperforming a recently published model. Intriguingly the results show that image corruptions do not appear like a useful tool to enhance model saliency predictions. We hypothesise that this may be in part caused by our use of pretrained encoder weights. Our EDSC-Net demonstrates that skip-connections from encoder to decoder are a powerful tool to overcome information loss in deep saliency models. We discuss the value deep saliency models can offer for our understanding of human visual attention.

Contents

List of Figures

List of Tables

1 Introduction

Despite limited perceptual resources and an environment that regularly confronts us with an overwhelming amount of visual information humans effortlessly make sense of their visual world. The key to solving this problem is visual attention, which allows for preferential extraction and processing of important visual information while leaving the irrelevant unattended (Carrasco, 2011; Chun, Golomb, & Turk-Browne, 2011; Itti, Koch, & Niebur, 1998; Moore & Zirnsak, 2017). Visual attention can be conceptualised as an information processing bottleneck which allows us to focus on specific aspects of the visual world to aid perception and processing (Borji, 2019; Kroner, Senden, Driessens, & Goebel, 2020). However, which computational mechanism enables us to spatially guide limited visual resources and how this process is exactly implemented in the human brain forms an open question. Previous research investigating potential ways of implementing such mechanisms yielded various computational models of visual attention (Borji & Itti, 2012; Borji, 2019; Itti et al., 1998; Kroner et al., 2020). In recent years, many of the best performing attentional models have been based on convolutional neural network (CNN) architectures allowing for tremendous improvements on attention modelling benchmarks (Borji, 2019).

However, selective visual attention is not a monolithic concept but comes in different forms with partly differing neural circuitry. In primate vision these different forms of attention can be usefully characterised through a set of dichotomies describing differences in types of visual attention (Moore & Zirnsak, 2017). Indeed, models of visual attention are not agnostic to these distinctions but usually aim to model specific sub-types of visual attention such as top-down or bottom-up attention (Borji & Itti, 2012). To give a better understanding of the models presented in this paper we will briefly elaborate distinctions between types of visual attention and how they relate to contemporary and historical visual attention modelling.

1.1 Top-Down vs. Bottom-Up Attention

Top-down attention describes preferential processing modulated by endogenous signals, i.e. internal behavioural agendas such as specific aims or goals (Moore & Zirnsak, 2017; Schütt, Rothkegel, Trukenbrod, Engbert, & Wichmann, 2019). In contrast, bottom-up visual attention is selective processing purportedly driven purely by the physical salience of the stimulus independent of the internal states (Schütt et al., 2019; Tatler & Vincent, 2008).

Top-down visual attention has been subject to extensive empirical explorations with many well-documented phenomena (Moore & Zirnsak, 2017). Early findings by Posner (1980) established that voluntary shifts of spatial attention towards a target location improves target detection substantially. Feature based top-down attention, i.e. conscious attending to specific stimulus features, similarly influences visual processing and detection of stimuli (Corbetta, Miezin, Dobmeyer, Shulman, & Petersen, 1990; Giesbrecht, Woldorff, Song, & Mangun, 2003).

The idea that visual attention may be in part influenced by bottom-up factors was originally brought forward in response to the observation that subjects have difficulty moving their gaze away from a suddenly presented stimulus in the peripheral vision even if explicitly instructed to do so (Hallett, 1978; Mokler & Fischer, 1999). In these antisaccade tasks, participants are instructed to move their eyes to a specific point located in the opposite direction of the appearing stimulus. Eye movements are typically characterised by longer latency and inaccuracies with respect to the target point (Hallett, 1978) as well as involuntary saccades towards the stimulus (Mokler & Fischer, 1999). Classical work by Treisman and Gelade (1980) also gives support for bottom-up contributions to visual attention. Treisman and Gelade demonstrated that during visual search targets defined by a single feature 'pop-out' and are identified much faster than targets defined by a conjunction of features. It has been argued that these 'pop-out' targets draw attention in an automatic, bottom-up fashion (Moore & Zirnsak, 2017; Treisman & Sato, 1990). While this distinction has been generally accepted (Egeth & Yantis, 1997; Moore & Zirnsak, 2017; Schütt et al., 2019) it has to be noted that there is still considerable debate about how top-down and bottom-up attention interact during visual processing (Borji & Itti, 2012; Foulsham & Underwood, 2008; Schomaker, Walper, Wittmann, & Einhäuser, 2017).

1.2 Covert vs. Overt Attention

The second dichotomy useful for understanding of computational frameworks is the distinction between covert and overt visual attention. The former can be defined as the allocation of visual attention without visible orienting movements while the latter is characterised as allocation of attention supported by orienting behaviour such as eye or limb movements (Borji & Itti, 2012; Moore & Zirnsak, 2017). In natural environments overt attention is the norm as the fixation of stimuli in the fovea allows for processing with the highest acuity as the resolution of human vision decreases with retinal eccentricity (Cowey & Rolls, 1974; Moore & Zirnsak, 2017). However, most visual attention research has focused on covert attention (Carrasco, 2011). Psychophysical investigations demonstrated that covert spatial attention can improve participant performance in detection (Bashinski & Bacharach, 1980) and discrimination tasks (Downing, 1988). Importantly covert and overt attention work together to create seamless processing of the visual environment. It has been established that shifts in covert visual attention precede overt saccadic eye movements. Hereby the presaccadic shift of visual attention can be seen as preparatory response to later eye movements allowing for more effective exploration of the visual world (Peterson, Kramer, & Irwin, 2004; Zhao, Gersch, Schnitzer, Dosher, & Kowler, 2012). No computational implementations of visual attention have attempted to model covert attention exclusively. The lack of models for pure covert attention is most likely due to a lack of measurements useful for model training. Thus models of human visual attention are primarily concerned with overt eye movement data (Borji & Itti, 2012).

1.3 Origins of Visual Attention Modelling

Early computational models of visual attention have been strongly inspired by Treisman and Gelade's (1980) Feature Integration Theory (FIT). FIT proposes that features are first processed in parallel after which shifts of attention combine features to unified object representations. FIT's emphasis on low-level stimulus features strongly influenced the development of early models predicting gaze allocation on the basis of stimulus features (Borji & Itti, 2012; Itti et al., 1998; Kroner et al., 2020). Later, Koch and Ullman (1987) introduced the concept of a saliency map — a topographic representation indicating most conspicuous parts of a scene which are likely to attract visual attention.

Inspired by these ideas Itti et al. (1998) developed the first model of visual attention. Itti et al.'s model combines detected low level features in a scene (colours, intensity, and orientations) linearly to generate a saliency map indicating image parts capable of attracting attention. Subsequent models also expanded this approach to include features such as faces or text based on the observation that these higher-level semantic features reliably attract fixations during free-viewing of a scene (Cerf, Frady, & Koch, 2009).

1.4 Categorisation of Visual Attention Models

Given the difficulty of measuring covert attention visual attention models are most commonly compared against eye movement data obtained from human participants. For purposes of attention modelling eye movements are commonly regarded as useful, easily measurable proxy for shifts in human attention. Given the tight coupling of overt and covert attention (Peterson et al., 2004) most contemporary models of visual attention should be seen to model both covert and overt attention simultaneously (Borji & Itti, 2012; Borji, 2019).

However, models differ in whether they represent top-down or bottom-up attention (Borji & Itti, 2012). Related to this distinction eye movements differ starkly when a scene is explored with a specific task compared to task-free exploration without defined goals (Yarbus, 1967). In line with this models represent visual attention in two types of tasks: in free-viewing tasks observers are asked to freely examine a scene at their own volition and models predict allocation of attention based on stimulus properties, and in visual search tasks attention is driven endogenously whereby the observer searches for a specific target (Borji & Itti, 2012; Borji, 2019; Tatler, Hayhoe, Land, & Ballard, 2011). With the exception of a few models combining top-down and bottom-up attention (Chikkerur, Serre, Tan, & Poggio, 2010; Fang, Lin, Lau, & Lee, 2011) most models can be located within this dichotomy. The classical model by Itti et al. (1998) approximates attention allocation from low-level stimulus features in a task-free fashion. In contrast top-down models such as the approach taken by Borji, Ahmadabadi, and Araabi (2011) aim to explicitly model visual attention in a task-driven way, e.g. for conjunction search and object detection in natural scenes.

Models derived from participant data in free-viewing tasks usually predict the allocation of gaze from properties of the presented stimuli (Tatler et al., 2011). While these models have provided a partial proof for the assumption that the visual system may allocate gaze partly through the properties of visually salient stimuli their usefulness has been questioned by some researchers. Tatler et al. (2011) argued that models based solely on data from such free-viewing tasks do not generalise to other contexts and artificially remove the main purpose of gaze allocation: the dynamic selection of important aspects of the visual world serving the current task. In other words, models based solely on salience in task-free conditions may not be a useful outside their domain. This is echoed in findings by Koehler, Guo, Zhang, and Eckstein (2014) participant fixation patterns in a search task are more similar to each other compared to fixation patterns in free-viewing. The authors postulated that this may be due to individual variability in what observers regard as salient.

1.5 Recent Models of Visual attention

With the success of large-scale machine learning and specifically convolutional neural networks (CNN) for visual tasks (Krizhevsky, Sutskever, &

Hinton, 2012; LeCun, Bottou, Bengio, & Haffner, 1998) the focus in visual attention modelling in free-viewing tasks has moved away from manual selection of relevant features. Contemporary deep saliency models learn representations of relevant features automatically from raw images and leverage those representations to generate saliency maps indicating fixation probabilities (Kroner et al., 2020). For saliency predictions on free viewing tasks this new class of deep saliency models yielded immense improvements while being trained in an elegant end-to-end fashion (Borji, 2019). The majority of deep saliency models make use of existing CNNs for object recognition and redeploy their complex feature representations for the prediction of saliency (Borji, 2019).

The basic advantage that CNNs offer over classical fully-connected networks for computer vision tasks can be found in their ability to capture features of increasing complexity from pixel-level image inputs through convolutional operations (Albawi, Mohammed, & Al-Zawi, 2017). In fully connected neural networks an adjustable weight is multiplied with each input pixel value and the sum of these operations is subsequently fed into a nonlinear activation function of a hidden neuron. In contrast, hidden neurons in CNNs do not receive input from all previous neurons, but from a specific local region in the image, i.e. the kernel. This not only reduces the computational complexity of the network but also allows hidden layer neurons to respond to specific features such as edges or corners (Albawi et al., 2017; Krizhevsky et al., 2012). Stacking of such convolutional layers in turn allows neurons in higher levels to respond to the presence of a conjunction of features detected in lower layers. The basic idea is that this allows for the detection of increasingly complex features along the network hierarchy (Albawi et al., 2017; Borji, 2019). During training of such models inputs are fed to the model and a forward-pass generates predictions. In a subsequent backward-pass weights are adjusted with respect to a calculated prediction error (LeCun et al., 1998). Intriguingly the hierarchical feature representations learned by these models have been argued to be analogous to neural representations in primate ventral vision where neural clusters similarly respond to increasingly complex features along the visual hierarchy (Khaligh-Razavi & Kriegeskorte, 2014; Kriegeskorte, 2015).

This effective capturing of image features makes CNNs a good backbone for saliency predictions models. In fact, Oyama and Yamanaka (2018) found that classification accuracy of object recognition backbones is strongly correlated with performance on saliency prediction tasks. In later layers these models do also capture image features that have a direct semantic interpretation which may provide an advantage for saliency prediction in free-viewing conditions (Kroner et al., 2020). Empirical research on gaze allocation demonstrated that fixation locations are substantially influenced by semantically meaningful features. Einhäuser, Spain, and Perona (2008) demonstrated that high level-semantically meaningful objects in a scene predict saliency better than purely low-level feature based saliency maps. Similarly, observers preferred viewing location in natural scenes is usually close to the center of semantically meaningful objects (Nuthmann & Henderson, 2010).

While modern models usually incorporate a pretrained CNN backbone different model designs have been proposed for bottom-up saliency prediction. Usually these models use an encoder-decoder structure in which a pretrained CNN captures image features followed by a trainable decoder which upsamples representations to a complete saliency map (Kroner et al., 2020; Kümmerer, Wallis, & Bethge, 2016). This design pattern resembling an autoencoder architecture (Masci, Meier, Cireşan, & Schmidhuber, 2011) is preserved in most models but a variety of different components have been employed to further boost performance (Borji, 2019). We will now outline a few key design features relevant in our current project and our employed modifications.

1.6 Model Design Patterns and Proposed Modifications

Alongside the canoncial encoder-decoder architecture a range of different design patterns have been employed in deep saliency models (Borji, 2019). To review all previous design choices is beyond the scope of the current work. However, we will concentrate on a few key model features that are of particular relevance for the current paper.

While CNNs do efficiently extract features a substantial amount of information is lost through the combination of convolutional operations and pooling layers. To overcome this information loss and to incorporate information that is represented in earlier layers saliency models have employed different mechanisms to explicitly utilise information from earlier encoder layers (Borji, 2019). To achieve this goal models have concatenated feature maps from several layers of the encoder to a single feature map representing mid- to high-level information. The approach has been used successfully to incorporate features of differing complexity in saliency prediction (Cornia, Baraldi, Serra, & Cucchiara, 2016; Kroner et al., 2020).

Furthermore models have made use of skip-connections to retain lost information. Skip-connections copy layer inputs and add the copied input to the output of a layer later in the model hierarchy (He, Zhang, Ren, & Sun, 2016). Skip-connections are a highly popular tool in deep learning models for object recognition and have also been used in saliency prediction (Qi, Lin, Shi, & Li, 2019). Skip-connections directly from encoder to decoder have also proven highly successful in semantic segmentation tasks (Badrinarayanan, Kendall, & Cipolla, 2017; Ronneberger, Fischer, & Brox, 2015). Importantly for our work Borji (2019) argued that skip connections may not have been fully explored as a tool to tackle information loss in deep saliency models while being prominently featured in object classification and segmentation models.

The first architectural modification proposed in the current work are motivated by findings on representation learning by Vincent, Larochelle, Bengio, and Manzagol (2008). The authors demonstrated that a powerful way to force models to represent useful features is to make learned representations robust to partial input corruptions. Their denoising autoencoders make the assumption that good feature representations should reflect regularities in the input which are highly characteristic and hence robust to partial destruction (Vincent et al., 2008). Thus partial destruction of the input should force the model to learn such particularly useful representations. The authors propose two ways of corrupting input data. First the corruption of input data through additive Gaussian noise, whereby each value in the input matrix is altered by adding a value randomly sampled from zero-centered Gaussian distribution. A second way to corrupt input data is masking noise which randomly forces a fraction of values in the input matrix to zero (Vincent et al., 2010). Considering the importance of feature representation for bottom-up saliency prediction (Oyama & Yamanaka, 2018) adding a denoising element to deep saliency models appears like a potentially useful avenue to improve model feature learning and saliency predictions in turn.

Denoising autoencoders have been demonstrated to be powerful tools in the domain of representation learning while containing an encoder-decoder structure similar to most deep saliency models. Hence we hypothesised that modifying input images during model training with different types of corruptions will allow for the extraction of more robust features to improve saliency predictions. To test the laid-out hypothesis we will train a model implementation similar to Kroner et al. (2020) with and without corruptions.

Secondly we propose a novel model with skip-connections from encoder to decoder with a pretrained VGG16 (Simonyan & Zisserman, 2014) as backbone to investigate the utility of skip-connections to tackle information loss in deep saliency models. We hypothesise that skip-connections from encoder to decoder are a useful way to tackle information loss in deep saliency models comparable to the classical concatenation approach used in other models (Cornia et al., 2016; Kroner et al., 2020).

2 Methods

To investigate the laid out questions we build a Keras (Chollet et al., 2015) implementation of the MSI-Net (Kroner et al., 2020). In addition we created a novel model for saliency prediction with skip-connections from model encoder to decoder similar to Qi et al. (2019), but with a pretrained VGG16 as encoder. Here we briefly explain learning principles in neural networks before elaborating the specifics of the current project.

2.1 Model principles

Neural networks learn patterns from large sets of data through gradient based learning (LeCun et al., 1998). The models have a multi layer architecture with each layer composed of several artificial neurons or nodes. The following derivations are partly taken from Skalski (2018, 2019). In a simple fully connected network given an input vector $\mathbf{x}$ a single neuron computes

$$z = \mathbf{w}^T \cdot \mathbf{x} + \mathbf{b} \tag{1}$$

where $\mathbf{w}$ denotes the adjustable weights connecting input vector $\mathbf{x}$ to the neuron; $\mathbf{b}$ denotes a bias vector. The result, z is subsequently fed to a non-linear function resulting in the neuronal activation $\mathbf{a}$.

$$\mathbf{a} = g(z) \tag{2}$$

Generalising this computation for an entire layer we can rewrite (1) and (2) as

$$z_i^{[l]} = \mathbf{w}_i^T \cdot \mathbf{a}^{[l-1]} + \mathbf{b}_i \quad \text{and} \quad \mathbf{a}_i^{[l]} = g^{[l]}(z_i^{[l]}) \tag{3}$$

superscript l indicates the selected layer and subscript i the index of a neuron in that layer. In the first layer of the model $\mathbf{a}^{[l-1]}$ represents the input vector $\mathbf{x}$. The key idea of CNNs are convolutions, i.e. local receptive fields (kernels) that pass over the input image which allow for shared weights and the capturing of specific features (LeCun et al., 1998). The filter hereby is simply a small matrix of weights w. The formula (4) computing the output of a single neuron z in a CNN is substantially more involved, but allows for a reduction in model parameters and the capturing of features present in an input through filters.

$$z[m,n] = (x * w)[m,n] = \sum_j \sum_k w[j,k]\; x[m+j-1, n+k-1] \tag{4}$$

Hereby m and n denote the rows and columns of the resulting output matrix, x represents the input image (or activation from previous layers in later network layers), $*$ the convolutional operation and j and k indices of filter height and width. Figure 1 illustrates the computation performed in formula (4). Formula (4) can be seen as the CNN equivalent to (1) in a fully connected network.

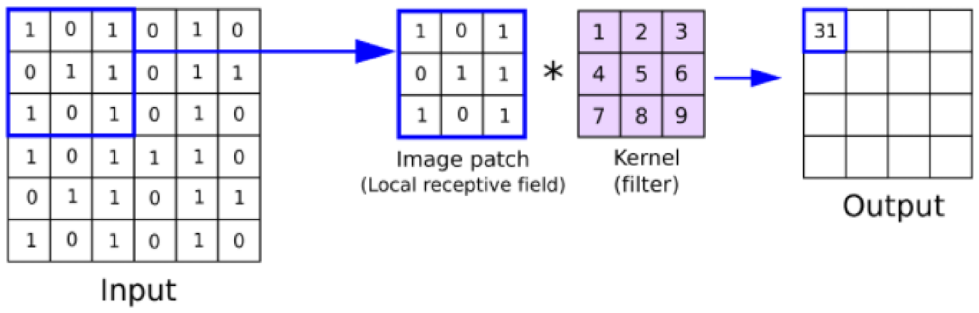

Figure 1: Visualisation of the convolutional operation (Reynolds, 2019).

As in (2) the intermediate value $z[m, n]$ is passed to a non-linear activation function to obtain the final activation for each neuron. In our models we used the popular Rectified Linear Units (RELU).

$$relu(x) = max(0, x) \tag{5}$$

To enable learning in models after a complete forward-pass a loss function computes the difference between our prediction and the corresponding ground-truth. Considering that our model aims to predict fixation probabilities we can use the Kullback-Leibler divergence (KLD) as measure of difference between actual and predicted fixation probabilities in the same manner as Kroner et al. (2020)

$$D_{KL}(P \parallel Q) = \sum_i Q_i \ln(\epsilon + \frac{Q_i}{\epsilon + P_i}) \tag{6}$$

where Q is the ground-truth probability density function (PDF) and P is the PDF predicted by our model after a complete forward-pass. Subscript i represents each pixel; ϵ is a small constant to avoid an undefined expression. The aim of the model during training is to minimise the loss function (6). To achieve this goal the backpropagation algorithm enables us to calculate the partial derivative of the loss function with respect to the adjustable weights $w[j,k]$ in each neuron and layer. For a detailed derivation of the partial derivatives used in CNNs see LeCun et al. (1998). On each iteration we update weights according to

$$w[j,k] = w[j,k] - \alpha \, \frac{\partial L}{\partial w[j,k]} \tag{7}$$

Here L represents our loss function (6) and α is our learning rate. This allows for the gradually adjustment weights in a fashion that minimises our loss function (6). In essence this incremental error-corrective process encompasses forward and backward propagation of information through the network. This is the key computational mechanism enabling the model to improve predictions over the course of training.

2.2 Model Architectures

2.2.1 MSI-Net

The first model used in our experiments is our Keras (Chollet et al., 2015) implementation of Kroner's (2020) MSI-Net. The encoder backbone in this model is the successful VGG16 (Simonyan & Zisserman, 2014). The VGG16 consists of 16 weight layers with stacks of 2-3 convolutional layers followed by max-pooling layers. The MSI-Net removes the last three fully connected layers, to preserve spatial information (Kroner et al., 2020). Furthermore stride in the final two pooling layers is reduced from two to one. This reduction in stride yields features maps of the same dimensions in the respective layers necessary for later concatenation. To incorporate features from several layers and to counteract information loss the encoder MSI-Net concatenates outputs from the last three encoder pooling layers into a single tensor. Inspired by semantic segmentation tasks Kroner et al. (2020) use an Atrous Spatial Pyramid Pooling (ASPP) module (Chen, Papandreou, Kokkinos, Murphy, & Yuille, 2017) to capture information at several scales. The module feeds the concatenated input tensor to several convolutional layers with different dilation rates. For more details on specific parameters consult Kroner et al. (2020). To aggregate the features encoded in the ASPP module to a saliency map the decoder uses bilinear upsampling layers followed by convolutional layers whereby the number of feature maps is halved in each step. The activation for all but the final model layer are determined by RELU activation functions (equation 5) and final encoder outputs are normalised. A visualisation of the architecture can be seen in Figure 2.

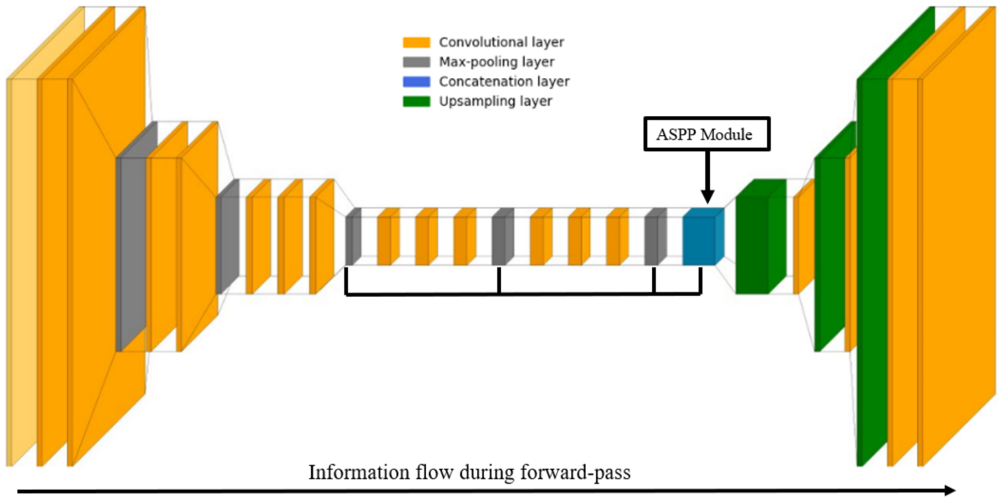

Figure 2: Visualisation of the MSI-Net by (Kroner et al., 2020). The black lines connecting central layers indicate feature maps used for concatenation. Beige layer on left hand side indicates input. Last layer on the right represents the outputed saliency map.

2.2.2 Encoder-Decoder With Skip Connections (EDSC-Net)

The second model used in the current project is our novel encoder-decoder architecture with skip connections (EDSC-Net) inspired by Badrinarayanan et al.'s (2017) SegNet. SegNet has been used for image segmentation which is a task type that has been argued to be closely related to saliency prediction (Borji, 2019; Kroner et al., 2020). The key similarity between SegNet and our proposed model is the utilisation of skip connections which concatenate feature maps from encoder directly with the corresponding feature map in our decoder. We chose this design to counteract information loss and to make use of features of differing complexity in our decoding process. Similar to MSI-Net we also use a pretrained VGG16 as our encoder whereby we discard its last three fully connected layers. However, strides are left at two for all pooling layers. Our decoder consists of a series of layers performing bilinear upsampling and convolutions with filter size of 3×3 and the number of feature maps is halved in each convolutional layer. We used RELU (equation 5) as our activation function with exception of the final output layer. As a final step model outputs were normalised. Our way of using skip connections from encoder to decoder has been used in similar fashion for saliency prediction by Qi et al. (2019), but to our knowledge we are the first to use this design pattern in conjunction with a VGG16 encoder. Our network architecture is illustrated in Figure 3 (Model code can be found in Appendix A.1-A.3).

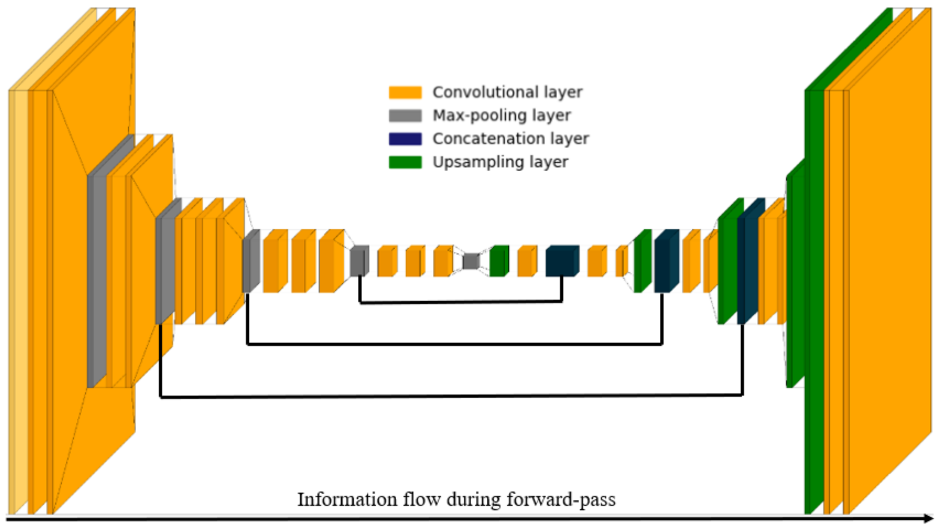

Figure 3: Visualisation of the EDSC-Net. The black arches mark our skip connections from encoder to decoder. Beige layer on left hand side indicates input. Last layer on the right represents the outputed saliency map.

2.3 Noise Corruptions

To test if partial destruction of input images forces the model to learn particularly useful representations we corrupted input data. In line with Vincent et al. (2010) we used two different types of corruptions. Our first way of corrupting our input data is additive Gaussian noise (GN). We corrupt our initial input matrix $\mathbf{x}$ in the following way

$$z \sim \mathcal{N}(0, \sigma^2), \quad \tilde{\mathbf{x}} = \mathbf{x} + z \tag{8}$$

here z is a tensor of the same dimensions as $\mathbf{x}$ containing random values sampled from $\mathcal{N}(0, \sigma^2)$, $\tilde{\mathbf{x}}$ represents the corrupted input. We subsequently used value clipping so that $\tilde{\mathbf{x}} \in [0, 1]$. For our experiments we set $\sigma^2 = 0.25$.

We also trained the models with inputs corrupted by Masking noise (MN). Hereby a fraction v of all values in $\mathbf{x}$ is set equal to 0. While the fraction remains constant the specific entries forced to 0 are chosen at random for each example. For our experiments we set $v = 0.5$ in line with work by Vincent et al. (2010). Figure 4 illustrates images with proposed corruptions.

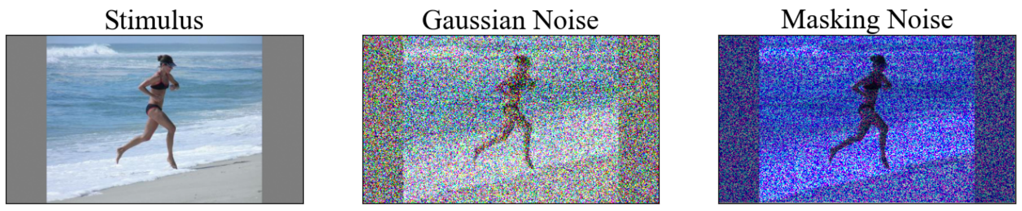

Figure 4: Visualisation of the corruptions and an example stimulus. Additive GN in the center and MN on the right.

2.4 Training

In the beginning of training weights in the MSI-Net decoder and ASPP module were initialised according to the Glorot-uniform method (Glorot & Bengio, 2010). Hereby weights in each layer are initialised sampled randomly form a uniform distribution between $-r$ and $+r$ with

$$r = \sqrt{\frac{3}{n_{in} + n_{out}}} \tag{9}$$

Here n_{in} are the number of incoming connections to the layer and n_{out} are the number of outgoing connections from that layer. The decoder of our EDSC-Net was initialised in the same fashion. The weights loaded into both model encoders were pretrained on the 1000 object categories of ImageNet

(Deng et al., 2009) and on scene recognition on the 365 categories of Places2 (Zhou, Lapedriza, Khosla, Oliva, & Torralba, 2017). As we use pretrained weights we decided to freeze the encoder weights for the first five epochs of training for the MSI-Net to avoid large error gradients in the beginning of training to damage weights (Géron, 2019).

Since we employ a KLD (equation 6) loss function values in each prediction P and ground truth Q have to be non-negative and sum to one as KLD assumes probability density functions as input (Kroner et al., 2020). Consequently all models normalise ground truth maps and predictions accordingly before computing loss. To minimise our loss function we used Adam optimisation (Kingma & Ba, 2014) with a learning rate of 10^{-5}. We incorporated image corruptions in the first layer of our models and therefore used a batch size of 1 as this allowed for easier implementation in the Keras API. Models were trained for 10 epochs on the SALICON dataset (Jiang, Huang, Duan, & Zhao, 2015) and for 15 on the CAT2000 (Borji & Itti, 2015) on a NVIDIA GeForce GTX 1660 SUPER GPU (details of the datasets are described below). For inference we saved weights which performed best on a validation split of our data.

To assess if image corruptions aid feature learning and saliency prediction we trained MSI-Net in three different configurations. We trained the models on plain input data without corruptions, a second set of weights was trained using data corrupted by additive GN, and a third set of weights was trained with MN. Figure 5 illustrates all training configurations.

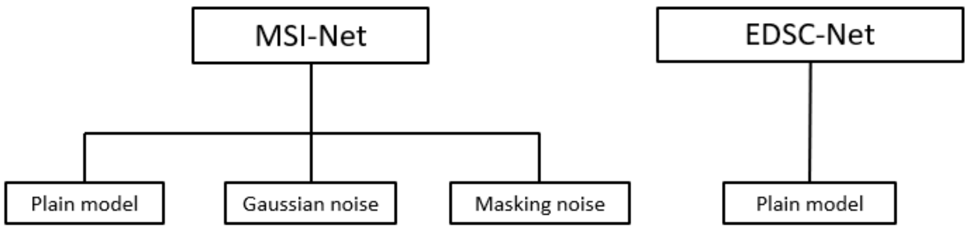

Figure 5: Visualisation our training configurations.

2.5 Datasets

Considering the slow incremental error-corrective process outlined in section 2.1 any deep learning model requires large amounts of annotated data. In saliency prediction many of the recent advantages have been made possible through the availability of datasets containing a wealth of images and corresponding ground truth saliency maps (Borji, 2019). For our training procedure we employed two datasets. Given the expensive nature of eye-tracking data acquisition the SALICON dataset (Jiang et al., 2015) uses mouse movements as a proxy for human visual attention when freely exploring natural scenes. The approach allowed the authors to acquire a large number of images and corresponding mouse-tracking saliency maps through crowd sourcing. While mouse movements are by no means a perfect proxy of visual attention in free-viewing this technique has been found to significantly improve saliency predictions (Tavakoli, Ahmed, Borji, & Laaksonen, 2017). SALICON encompasses 10000 training and 5000 validation images depicting natural scenes — a subset of the MS COCO database (Lin et al., 2014). Specifically images in SALICON present scenes with rich contextual information usually displaying several objects in their common environment. Each image is viewed by 60 observers for 5 seconds and all mouse movements are aggregated into a single saliency map.

The CAT2000 (Borji & Itti, 2015) encompasses 2000 training images with corresponding ground truth observer fixations recorded using an EyeLink1000 eyetracker. For the CAT2000 each image was viewed by 18 observers for 5 seconds in a free-viewing fashion. The total number of observers is 120 (40 male, 80 female) with a mean age of 20.15 (min = 18, max = 27) (Borji & Itti, 2015). Fixations of all observers on an image are aggregated into a single saliency map. Images fall into 20 different categories depending on image content such as 'art' or 'social', which stands in contrast to SALICON which is solely based on natural scenes. We refer to Borji and Itti (2015) and Jiang et al. (2015) for extensive details on these datasets.

We resized all images and ground truth maps from SALICON to 240×320 pixels for the MSI-Net and to 224×320 for the EDSC-Net. Images and maps from CAT2000 were resized to 216×384 for the MSI-Net and to 224×384 for the EDSC-Net.

Given the larger size of the data set we first trained our models on SALICON. Subsequently we fine-tuned the model on CAT2000 in which fixation maps are recorded using precise eye tracking tools. For SALICON we used their predefined train-validation split (10000 train and 5000 validation images). For the CAT2000 we defined 70% of the available data set as training set, 15% for validation and, 15% for testing, i.e. 1400, 300, and 300 images respectively. As CAT2000 contains images belonging to 20 categories this split similarly applies to each of the 20 categories.

3 Results

3.1 Learning Curves

We will first examine learning curves during training on SALICON and CAT2000 to see that the models converge after a few epochs, i.e. a point is reached where validation-loss does not decrease. While training loss decreases further we use the point of best performance on the validation set for inference. Hereby, validation loss is the performance of the model on the validation set of our data which is not used for training. This early stopping allows us to counteract overfitting and generate better performance on inference (Géron, 2019).

In Figure 6 we illustrate the training loss as a function of training epochs for the different MSI-net configurations. Hereby one epoch is one full forward and backward propagation of the entire dataset. First, when unfreezing encoder weights after five epochs on SALICON there is a marked decline in KLD-loss in all configurations (Figure 6., left) indicating additional learning in the encoder aides model performance. Second when fine-tuning the model on the CAT2000 (Figure 6., right) validation-loss only decreases in the first few epochs. This indicates pretraining on SALICON resulted in good performance and additional improvements are rather small.

For the proposed EDSC-Net training loss over epochs is illustrated in Fig-

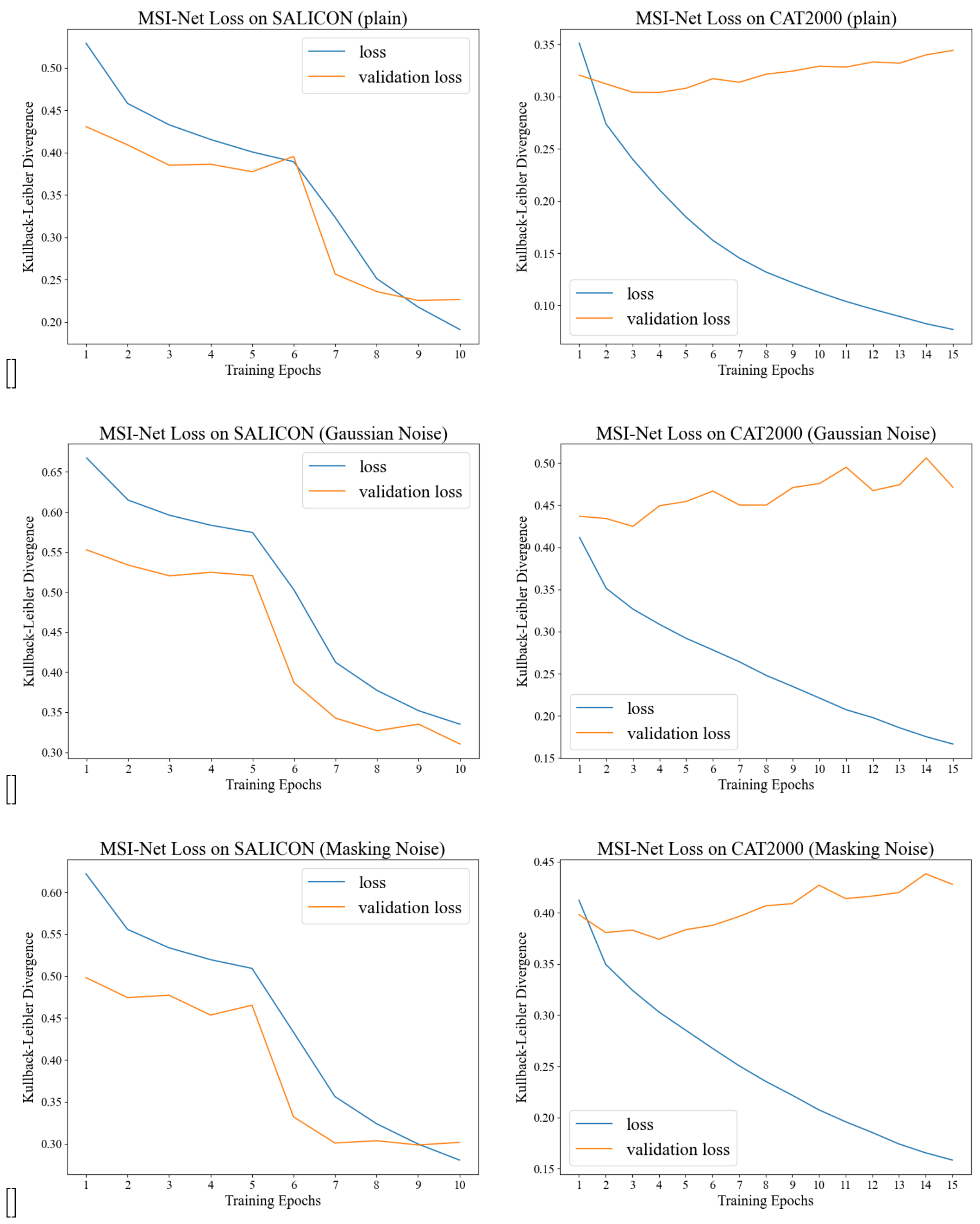

Figure 6: Learning curves of the MSI-Net trained with and without image corruptions. (Left) loss on Salicon, (Right) Loss on CAT2000, (top) model trained without corruptions, (middle) model trained with additive GN, (bottom) model trained with MN.

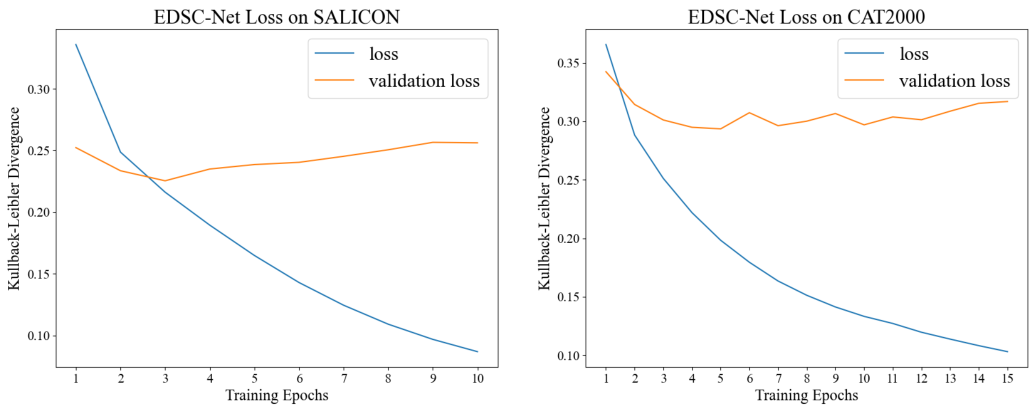

Figure 7: Learning curves of the EDSC-Net (Left) loss on Salicon, (Right) Loss on CAT2000.

ure 7. As we did not freeze encoder weights during training of the EDSC-Net there is no drop in training error after five epochs (Figure 7., left). Loss for the SALICON and CAT2000 validation sets converges after a small number of epochs. The training performance of the model with added skip connections appears to be comparable to the MSI-Net trained without corruptions. While the number of training epochs is a parameter that has to be chosen on volition of the researcher in all cases it allowed for good model convergence.

3.2 Evaluation of models on Independent Test Set

Learning curves give a good first indication of how well our models minimise the KLD loss-function. In machine learning the ability of the model to generalise to new cases is tested on a split of the dataset not used in training to characterise the *out-of-sample error* (Géron, 2019). Here we test the model on the held-out split of the CAT2000 (300 images). While our models were pre-trained on the SALICON dataset we are not specifically interested in the performance of our models on this mouse tracking dataset, but we merely leverage its data to obtain better predictions.

To understand if training a model with noise corruptions enhances saliency predictions we will examine the performance of our MSI-Net trained with and without corruptions. Table 1 displays the result of this test. While there are several commonly used metrics available to evaluate fixation predictions (Bylinskii, Judd, Oliva, Torralba, & Durand, 2018) we decided to evaluate performance using KLD (same as our loss function) and Pearson's Correlation Coefficient (CC). In the case of saliency prediction CC measures the linear relationship between the ground truth Q and our prediction P, treating Q and P as random variables (Bylinskii et al., 2018)

$$CC(P,Q) = \frac{\sigma(P,Q)}{\sigma(P) \times \sigma(Q)} \tag{10}$$

here $\sigma(P,Q)$ represents the covariance of P and Q. With respect to the model performance Table 1 shows that the MSI-Net trained without corruptions performs better than models trained with corruptions. The results thus contradict out hypothesis that model training with input-corruptions would improve predictive performance.

	KLD ↓	CC ↑
MSI-Net (Plain)	**0.30**	**0.85**
MSI-Net (Gaussian Noise)	0.41	0.77
MSI-Net (Masking Noise)	0.37	0.81

Table 1: Results of MSI-Net configurations on the CAT2000 test split. Arrows indicate the type of metric: (↓)metric assess dissimilarity, (↑) metric assess similarity. (**Bold**) best performance.

Secondly we will compare the test performance of the EDSC-Net with the MSI-Net to understand if skip connections are a useful tool to counteract information loss in deep saliency models. Table 2 presents the results of our test. Our model with skip-connections from encoder to decoder achieves state-of-the-art performance. Our model achieves comparable performance to the MSI-Net. Importantly the MSI-Net (Kroner et al., 2020) is a recently established, leading saliency model performing well on various benchmarks. We marginally beat the MSI-Net on the test split of the CAT2000 on the CC and KLD. The results indicate that skip-connections are an adequate way to counteract information loss in deep saliency models. Importantly our model achieves this performance with a marginally smaller number of trainable parameters than the MSI-Net. While the MSI-Net has a total of 24,934,209 trainable weights the EDSC-Net contains a total of 24,840,513 trainable weights.

	KLD ↓	CC ↑
MSI-Net (Plain)	0.30	0.85
EDSC-Net (Plain)	**0.29**	**0.86**

Table 2: Results of EDSC-Net and MSI-Net on the CAT2000 test split. Arrows indicate the type of metric: (↓) metric assess dissimilarity, (↑) metric assess similarity. (**Bold**) best performance.

	KLD ↓
Pattern	0.17
Fractal	0.21
Inverted	0.21
OutdoorNatural	0.23
BlackWhite	0.24
LowResolution	0.24
Sketch	0.25
Social	0.26
Random	0.26
Satellite	0.28
OutdoorManMade	0.29
Art	0.30
Object	0.30
Noisy	0.31
Jumbled	0.32
Action	0.33
Affective	0.34
LineDrawing	0.37
Indoor	0.55
Cartoon	0.61

Table 3: Results of EDSC-Net on the CAT2000 test split grouped by categories of the dataset. Arrow indicates the type of metric: (↓) metric assesses dissimilarity.

As outlined in section 2.5 scenes in the CAT2000 dataset are organised in several categories. This allows us to examine the performance of the EDSC-Net with respect to these specific subsections of the data. Table 3 shows the predictive performance with respect to each scene category($M = 0.29$, $SD = 0.10$). Interestingly when examining category level performance we can find outliers. For example, KLD for the category containing cartoon images is 3 standard deviations above the mean.

3.3 Qualitative Examination of Predictions

A qualitative examination of predictions confirms that the EDSC-Net successfully predicts regions of an image that are semantically meaningful and attract visual attention in free-viewing. Figure 8 shows that the EDSC-Net captures the importance of different semantic features such as a human body (Action), human faces (Social), written text (Indoor), or pop-out stimuli (Pattern) for the allocation of attention in free viewing. Importantly these predictions are very similar to predictions by MSI-Net (Figure 8).

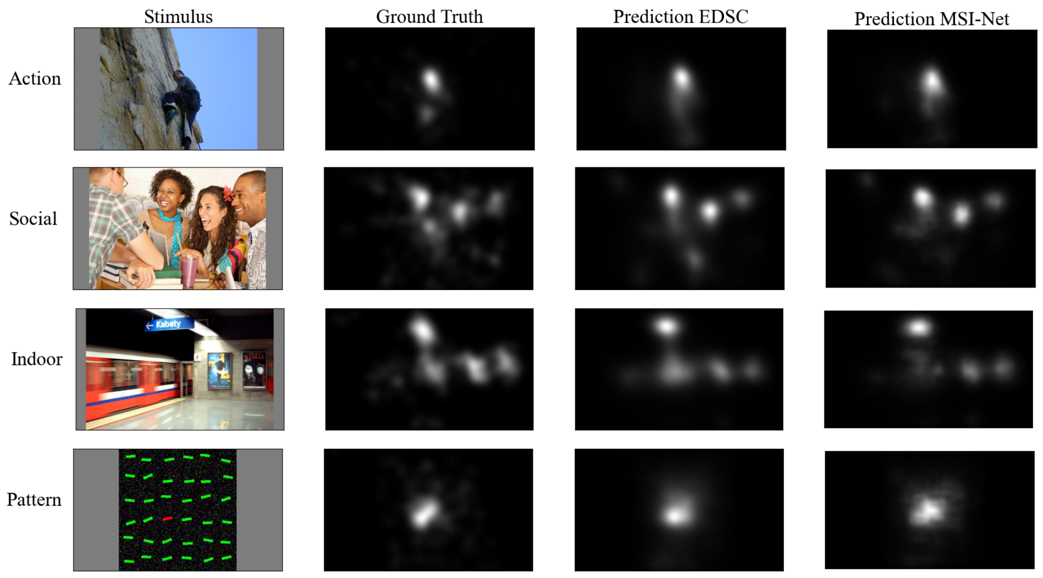

Figure 8: Demonstration of the EDSC-Net's and the MSI-Net's ability to predict salient image regions for novel images from four different categories. (Ground Truth) saliency maps derived from human eye tracking data.

We can also qualitatively compare predictions by the MSI-Net trained with noise-corrupted images with the 'plain' MSI-Net trained with uncorrupted input. Figure 9 illustrates this for an example stimulus. Qualitatively the 'plain' model appears to gives a slightly more accurate prediction.

We will also examine examples of EDSC-Net's saliency predictions for categories where the model is performing comparably poor such as the Car-

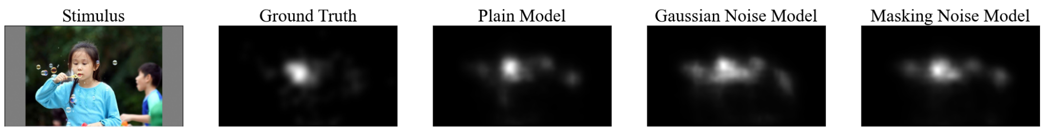

Figure 9: Visualisation of saliency predictions by the MSI-Net trained with and without input corruptions.

toon or Indoor category see table 3. Figure 10 exemplifies this performance shortcoming on specific categories. The predictions appear qualitatively less accurate than the other examples shown above.

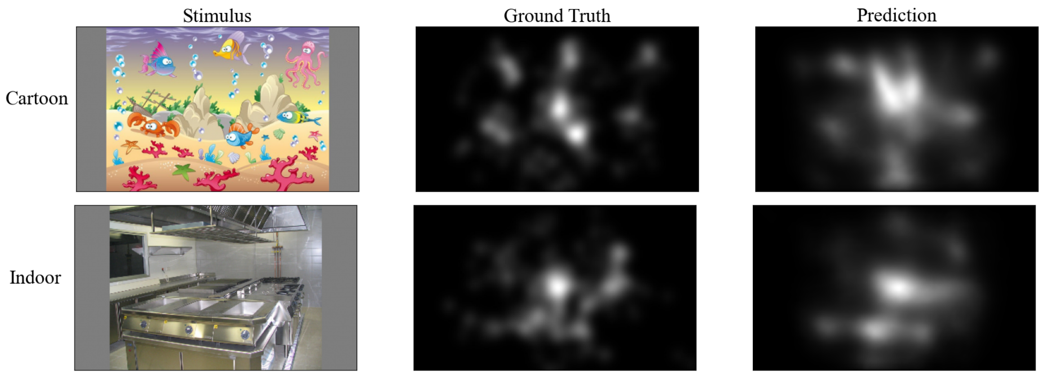

Figure 10: Demonstration of the EDSC-Net's predictions for examples of low-performance categories. (Ground Truth) saliency maps derived from human eye tracking data.

Lastly we exemplify that the EDSC-Net in certain instances is unable to incorporate high-level understanding of contextual information in their predictions as also observed in previous saliency models (Borji, 2019; Kroner et al., 2020). Figure 11 shows that the model is unable to predict allocation of attention stemming from a social cue (gaze direction).

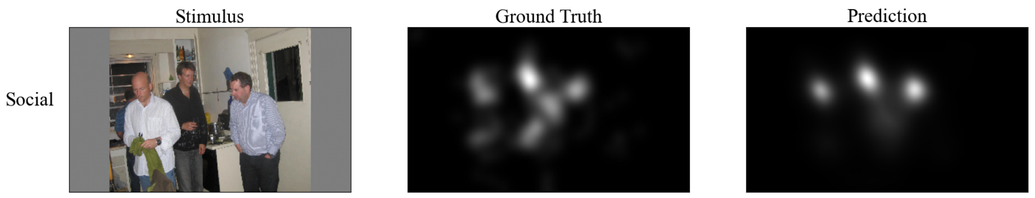

Figure 11: Visualisation of a saliency predictions by the EDSC-Net whereby the model fails to capture gaze direction. (Ground Truth) saliency maps derived from human eye tracking data.

4 Discussion

The current work set out to investigate the influence of two distinct CNN design features on saliency predictions in deep learning models. Specifically we examined the power of our models to predict the allocation of visual attention via the proxy of gaze in free-viewing tasks. First inspired by work on unsupervised representation learning by Vincent et al. (2008, 2010) we examined the influence of training deep saliency models with additive Gaussian and Masking noise. We hypothesised that training the models with corrupted inputs would force the learning of more useful representations and thus improve saliency predictions. Contradicting our hypothesis we found that training the MSI-Net with input corruption lead to a performance decrease. The results indicate that training models with corrupted inputs does not improve performance. The explanation for this result is likely to be found in the difference between the task our model attempts to solve and the classical representation learning setting in which denoising autoencoders are usually employed (Bengio, Courville, & Vincent, 2013). Image corruptions are mostly employed for autoencoder image reconstruction tasks (Bengio et al., 2013; Vincent et al., 2008). In contrast, our model is not computing loss between input image and reconstruction but between an extrapolated saliency map and a ground truth. Even more crucial these models usually learn from scratch. In contrast, we did not train model encoders from scratch but initialised the part of the network with carefully pretrained weights for image and scene classification. Thus it may be possible that a model in which weights are already trained to represent highly characteristic features may not benefit from input corruptions. We can find evidence for such effects in work on 'adversarial examples' for image classification CNNs (Szegedy et al., 2013). The authors demonstrate that in CNNs trained to high performance on image classification, minimal perturbations of the input can lead to misclassifications. In the context of our work this may explain the adverse effect that corrupted input had on the learning of our model. Instead of forcing the model to learn more robust feature representation our corruptions may simply hinder good feature extraction by our already trained encoder and thus negatively impact learning in the decoder.

For the second part of our work we investigated the usefulness of a novel design feature to counteract information loss in deep saliency models. Motivated by the use of skip-connections in the semantic segmentation literature (Badrinarayanan et al., 2017; Ronneberger et al., 2015) we developed our novel EDSC-Net employing connections from encoder to decoder. Our comparison with the recently published, highly successful MSI-Net (Kroner et al., 2020) showed that our model reaches comparable performance. The results thus confirm that encoder-decoder skip connections are a useful tool to counteract information loss and to incorporate information from early model layers into saliency predictions. In contrast to our work the MSI-Net uses concatenation and a complex ASPP module (Chen et al., 2017) to achieve this task. Our results show that simple concatenation of feature maps from model encoder to decoder results in similar performance. In the context of previous model designs for fixation prediction by Qi et al. (2019) our work further validates the finding that skip-connections appear as a useful instrument for saliency prediction in free-viewing. Beyond this our findings also highlight the relatedness between saliency prediction and image segmentation tasks on a computational level. The EDSC-Net is inspired by model design patterns first proposed in the semantic segmentation literature which employs encoder-decoder skip connections for pixel-wise image labeling. Similarly, the ASPP module, first used for saliency prediction by Kroner et al. (2020), was originally developed for the task of semantic segmentation. Previously also recurrent neural networks used for image segmentation proved successful in fixation prediction tasks (Borji, 2019). The likely reason for why algorithmic solutions developed for semantic segmentation can be useful in saliency prediction lies in their ability to retain spatial information while simultaneously detecting features. When considering that semantically meaningful features play a crucial role in the allocation of visual attention (Einhäuser et al., 2008; Nuthmann & Henderson, 2010) it appears intuitive that such models have high utility for saliency prediction. In other words, deep saliency models simply detect features and their locations, just as semantic segmentation models. The additional piece of information that saliency models learn is the relative importance of each detected feature for the attraction of visual attention.

The category wise analysis of the EDSC-Net performance on the CAT2000 test set (see Table 3) yields additional insight in the explanatory power of our model. Of particular interest are categories of low performance such as the Cartoon category. A potential reason for the comparably poor performance of our model in the Cartoon category is its artificial content. Our model encoder is pretrained on the ImageNet (Krizhevsky et al., 2012) and Places2 (Zhou et al., 2017) datasets and subsequently trained on the SALICON dataset. All three databases exclusively contain non-artificial images and our model may hence be unfamiliar with features contained in such images. This observation highlights an issue that is fundamental in deep learning. Model are highly dependent on their training data and have fundamental problems generalising to new contexts (Lake, Ullman, Tenenbaum, & Gershman, 2017).

Beyond our proposed architectural and methodological modifications it is also important to ask if such models are indeed helpful tools, capable of improving our understand of mechanisms governing the human allocation of visual attention. We have to consider that saliency models are highly proficient at predicting allocation of visual attention in free-viewing contexts (Borji, 2019). For example when we qualitatively examine predictions by models in Figure 8 it is nearly impossible to tell model predictions from human data. It could be that such models achieve the high-level goal of mirroring neural mechanisms that govern allocation of attention in humans. Our type of saliency models achieve their proficiency in predicting allocation of gaze in task-free conditions through bottom-up, feature-driven mechanisms (Koehler et al., 2014; Tatler et al., 2011). However whether such models are useful instruments in explaining visual attention in a broader sense rests on several assumptions that have been elaborated in length by Tatler et al. (2011) and also apply to our current work.

Models of gaze allocation in free-viewing conditions are based on the implicit assumption that behaviours in such tasks represent a default bottom-up way of allocating visual attention. Tatler et al. (2011) propose that rather than representing a bottom-up default mode free-viewing allows observers to choose their individual strategy when exploring the scene. This proposition is supported by the finding that participants fixations in free-viewing have higher inter-observer variability compared to visual search tasks (Koehler et al., 2014). Therefore the use of free-viewing tasks and in our case the modelling of gaze allocation in free-viewing may be conceptually problematic if we want to infer fundamental conclusions about visual attention as a whole. A further conceptual problem is our models disregard for the temporal aspects of visual attention. The time spent by observers fixating on a single location can vary widely during scene viewing (Findlay, Gilchrist, et al., 2003; Henderson & Smith, 2009) while our models merely predict spatial gaze allocation. Given this variability in fixation duration some information about mechanisms of visual attention should be seen to be contained within this temporal data (Tatler et al., 2011). Previous research suggested that fixation duration varies depending on the need for information extraction at the current location (Hayhoe, Shrivastava, Mruczek, & Pelz, 2003). If we aim to create a model that is able to explain gaze allocation and visual attention holistically it will be important to incorporate this temporal aspect into model predictions.

Despite these conceptual criticisms deep saliency models may offer some explanatory merit for our understanding of visual attention. It is well documented that certain stimuli reliably attract fixation such as faces (Theeuwes & Van der Stigchel, 2006) or abrupt onset stimuli (Irwin, Colcombe, Kramer, & Hahn, 2000). Therefore Tatler et al. (2011) argued that regardless of the usual task driven nature of visual attention there must be mechanisms that capture attention in a reflexive, bottom-up manner. One potential mechanism for this automatic attraction of gaze is Itti and Baldi's (2006) idea that 'surprising' or unexpected visual stimuli attract attention. Surprise is hereby conceptualised as areas that are statistical outliers within visual input. Saliency models may therefore be useful tools to understand gaze allocation if a part of the visual signal is statistically abnormal (Tatler et al., 2011). Therefore saliency models may be useful tools to explain gaze allocation with respect to its stimulus driven components.

Besides conceptual criticisms specific methodological limitations also apply to the current work. The CAT2000 dataset contains 2000 images which we split into train, validation, and test set. However, the CAT2000 contains additional 2000 images with held out fixation maps used for final model evaluation — predictions have to be submitted (Borji & Itti, 2015). Due to the relatively short time-frame available for the current project we were not able to obtain final performance measures on the held out data and our approach hence represents a compromise. To understand final model performance of the EDSC-Net we will submit predictions, which also will allow us to employ our test set as additional training data. A second methodological limitation is found in the stochastic nature of deep learning models. Two factors make exact conclusions about model performance more difficult: random initialisation (Glorot & Bengio, 2010) and stochastic nature of the optimisation algorithm (Ruder, 2016). Our model decoder weights are initialised with a random element (see section 2.4) and gradient based learning is characterised by random fluctuations (Ruder, 2016). Hence it is possible that different runs of the same model and data can yield slightly different performances. To increase confidence in our model it will be advisable to train the network for a number of times yielding a distribution of model predictive performance to control for model randomness. Another important avenue for further improvement becomes evident when examining specific failing examples of our model such as Figure 11 in which the EDSC-Net fails to capture human gaze. The specific failing example connects to the observation by Borji (2019) that in order to model human performance perfectly models will have to gain an understanding of scenes they currently lack. Only this will allow for the reflection of high-level factors such as gaze and subtle social cues in model saliency predictions. Given the feature-driven nature of deep learning models it is likely that to achieve this end models will require fundamental design modifications.

5 Conclusion

The present work shows saliency models as proficient predictors of human gaze allocation in free-viewing. We demonstrated that training a model with corrupted input data led to impairments in model predictive performance. Equipping a saliency model with encoder-decoder skip-connections allows for state-of-the art performance comparable to other contemporary approaches. The results highlight the computational relatedness between image segmentation and saliency prediction tasks. Despite these achievements it is questionable if saliency models are indeed good explanatory tools for human visual attention, due to the narrowness and assumptions inherent to the free-viewing task. To overcome current obstacles, deep saliency models should move beyond the limited domain of free-viewing and make use of architectural modifications.

Acknowledgements

Many people provided me with invaluable support for my thesis which I would like to thank. I would like to thank my supervisor Prof. Ben Tatler for useful advise and suggestions throughout the creation of this work. I am also grateful for his trust in my ideas and modelling. I would like to thank my good friend Ben Lonnqvist for first igniting my fascination for modelling approaches within psychology and neuroscience. I would like to thank my friend Jovan Efferth for being the non-academic friend I can rant to about work and who always supports me. I would like to thank my parents for their trust in my endeavours and their loving support. Last but not least I want to thank my partner, Nurah, for her reassurance at all times and for her ability to enrich my life in her very special manner.

References

Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (icet)* (pp. 1–6).

Badrinarayanan, V., Kendall, A., & Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(12), 2481–2495.

Bashinski, H. S., & Bacharach, V. R. (1980). Enhancement of perceptual sensitivity as the result of selectively attending to spatial locations. *Perception & Psychophysics*, *28*(3), 241–248.

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1798–1828.

Borji, A. (2019). Saliency prediction in the deep learning era: Successes and limitations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *43*, 679-700.

Borji, A., Ahmadabadi, M. N., & Araabi, B. N. (2011). Cost-sensitive learning of top-down modulation for attentional control. *Machine Vision and Applications*, *22*(1), 61–76.

Borji, A., & Itti, L. (2012). State-of-the-art in visual attention modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(1), 185–207.

Borji, A., & Itti, L. (2015). Cat2000: A large scale fixation dataset for boosting saliency research. *arXiv preprint arXiv:1505.03581*.

Bylinskii, Z., Judd, T., Oliva, A., Torralba, A., & Durand, F. (2018). What do different evaluation metrics tell us about saliency models? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *41*(3), 740–757.

Carrasco, M. (2011). Visual attention: The past 25 years. *Vision Research*, *51*(13), 1484–1525.

Cerf, M., Frady, E. P., & Koch, C. (2009). Faces and text attract gaze independent of the task: Experimental data and computer model. *Journal of Vision*, *9*(12), 10–10.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2017). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *40*(4), 834–848.

Chikkerur, S., Serre, T., Tan, C., & Poggio, T. (2010). What and where: a bayesian inference theory of attention. *Vision Research*, *50*(22), 2233–2247.

Chollet, F., et al. (2015). *Keras.* `https://github.com/fchollet/keras`. GitHub.

Chun, M. M., Golomb, J. D., & Turk-Browne, N. B. (2011). A taxonomy of external and internal attention. *Annual Review of Psychology*, *62*, 73–101.

Corbetta, M., Miezin, F. M., Dobmeyer, S., Shulman, G. L., & Petersen, S. E. (1990). Attentional modulation of neural processing of shape, color, and velocity in humans. *Science*, *248*(4962), 1556–1559.

Cornia, M., Baraldi, L., Serra, G., & Cucchiara, R. (2016). A deep multi-level network for saliency prediction. In *2016 23rd international conference on pattern recognition (icpr)* (pp. 3488–3493).

Cowey, A., & Rolls, E. (1974). Human cortical magnification factor and its relation to visual acuity. *Experimental Brain Research*, *21*(5), 447–454.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 ieee conference on computer vision and pattern recognition* (pp. 248–255).

Downing, C. J. (1988). Expectancy and visual-spatial attention: effects on perceptual quality. *Journal of Experimental Psychology: Human Perception and Performance*, *14*(2), 188.

Egeth, H. E., & Yantis, S. (1997). Visual attention: Control, representation, and time course. *Annual Review of Psychology*, *48*(1), 269–297.

Einhäuser, W., Spain, M., & Perona, P. (2008). Objects predict fixations better than early saliency. *Journal of Vision*, *8*(14), 18–18.

Fang, Y., Lin, W., Lau, C. T., & Lee, B.-S. (2011). A visual attention model combining top-down and bottom-up mechanisms for salient object detection. In *2011 ieee international conference on acoustics, speech and signal processing (icassp)* (pp. 1293–1296).

Findlay, J. M., Gilchrist, I. D., et al. (2003). *Active vision: The psychology of looking and seeing* (No. 37). Oxford University Press.

Foulsham, T., & Underwood, G. (2008). What can saliency models predict about eye movements? spatial and sequential aspects of fixations during encoding and recognition. *Journal of Vision*, *8*(2), 6–6.

Géron, A. (2019). *Hands-on machine learning with scikit-learn, keras, and tensorflow: Concepts, tools, and techniques to build intelligent systems.* O'Reilly Media.

Giesbrecht, B., Woldorff, M., Song, A., & Mangun, G. R. (2003). Neural mechanisms of top-down control during spatial and feature attention. *Neuroimage*, *19*(3), 496–512.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249–256).

Hallett, P. (1978). Primary and secondary saccades to goals defined by instructions. *Vision Research*, *18*(10), 1279–1296.

Hayhoe, M. M., Shrivastava, A., Mruczek, R., & Pelz, J. B. (2003). Visual memory and motor planning in a natural task. *Journal of Vision*, *3*(1), 6–6.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 770–778).

Henderson, J. M., & Smith, T. J. (2009). How are eye fixation durations controlled during scene viewing? further evidence from a scene onset delay paradigm. *Visual Cognition*, *17*(6-7), 1055–1082.

Irwin, D. E., Colcombe, A. M., Kramer, A. F., & Hahn, S. (2000). Attentional and oculomotor capture by onset, luminance and color singletons. *Vision Research*, *40*(10-12), 1443–1458.

Itti, L., & Baldi, P. F. (2006). Bayesian surprise attracts human attention. In *Advances in neural information processing systems* (pp. 547–554).

Itti, L., Koch, C., & Niebur, E. (1998). A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on pattern analysis and machine intelligence*, *20*(11), 1254–1259.

Jiang, M., Huang, S., Duan, J., & Zhao, Q. (2015). Salicon: Saliency in context. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 1072–1080).

Khaligh-Razavi, S.-M., & Kriegeskorte, N. (2014). Deep supervised, but not unsupervised, models may explain it cortical representation. *PLoS Computational Biology*, *10*(11), e1003915.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Koch, C., & Ullman, S. (1987). Shifts in selective visual attention: towards the underlying neural circuitry. In *Matters of intelligence* (pp. 115–141). Springer.

Koehler, K., Guo, F., Zhang, S., & Eckstein, M. P. (2014). What do saliency models predict? *Journal of Vision*, *14*(3), 14–14.

Kriegeskorte, N. (2015). Deep neural networks: a new framework for modeling biological vision and brain information processing. *Annual Review of Vision Science*, *1*, 417–446.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, *25*, 1097–1105.

Kroner, A., Senden, M., Driessens, K., & Goebel, R. (2020). Contextual encoder–decoder network for visual saliency prediction. *Neural Networks*, *129*, 261–270.

Kümmerer, M., Wallis, T. S., & Bethge, M. (2016). Deepgaze ii: Reading fixations from deep features trained on object recognition. *arXiv preprint arXiv:1610.01563*.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., & Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and Brain Sciences*, *40*.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., … Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision* (pp. 740–755).

Masci, J., Meier, U., Cireşan, D., & Schmidhuber, J. (2011). Stacked convolutional auto-encoders for hierarchical feature extraction. In *International conference on artificial neural networks* (pp. 52–59).

Mokler, A., & Fischer, B. (1999). The recognition and correction of involuntary prosaccades in an antisaccade task. *Experimental Brain Research*, *125*(4), 511–516.

Moore, T., & Zirnsak, M. (2017). Neural mechanisms of selective visual attention. *Annual Review of Psychology*, *68*, 47–72.

Nuthmann, A., & Henderson, J. M. (2010). Object-based attentional selection in scene viewing. *Journal of Vision*, *10*(8), 20–20.

Oyama, T., & Yamanaka, T. (2018). Influence of image classification accuracy on saliency map estimation. *CAAI Transactions on Intelligence Technology*, *3*(3), 140–152.

Peterson, M. S., Kramer, A. F., & Irwin, D. E. (2004). Covert shifts of attention precede involuntary eye movements. *Perception & Psychophysics*, *66*(3), 398–405.

Posner, M. I. (1980). Orienting of attention. *Quarterly Journal of Experimental Psychology*, *32*(1), 3–25.

Qi, F., Lin, C., Shi, G., & Li, H. (2019). A convolutional encoder-decoder network with skip connections for saliency prediction. *IEEE Access*, *7*, 60428–60438.

Reynolds, A. (2019). Convolutional neural networks (cnns). *Online: https://anhreynolds.com/blogs/cnn.html, access: March.*

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International conference on medical image computing and computer-assisted intervention* (pp. 234–241).

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Schomaker, J., Walper, D., Wittmann, B. C., & Einhäuser, W. (2017). Attention in natural scenes: affective-motivational factors guide gaze independently of visual salience. *Vision Research*, *133*, 161–175.

Schütt, H. H., Rothkegel, L. O., Trukenbrod, H. A., Engbert, R., & Wichmann, F. A. (2019). Disentangling bottom-up versus top-down and low-level versus high-level influences on eye movements over time. *Journal of Vision*, *19*(3), 1–1.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Skalski, P. (2018). Deep dive into math behind deep networks. *Online: https://towardsdatascience.com/https-medium-com-piotr-skalski92-deep-dive-into-deep-networks-math-17660bc376ba, access: March*, *17*.

Skalski, P. (2019). Gentle dive into math behind convolutional neural networks. *Online: https://towardsdatascience. com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9, access: March*, *5*.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Tatler, B. W., Hayhoe, M. M., Land, M. F., & Ballard, D. H. (2011). Eye guidance in natural vision: Reinterpreting salience. *Journal of Vision*, *11*(5), 5–5.

Tatler, B. W., & Vincent, B. T. (2008). Systematic tendencies in scene viewing. *Journal of Eye Movement Research*, *2*(2).

Tavakoli, H. R., Ahmed, F., Borji, A., & Laaksonen, J. (2017). Saliency revisited: Analysis of mouse movements versus fixations. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 1774–1782).

Theeuwes, J., & Van der Stigchel, S. (2006). Faces capture attention: Evidence from inhibition of return. *Visual Cognition*, *13*(6), 657–665.

Treisman, A. M., & Gelade, G. (1980). A feature-integration theory of attention. *Cognitive Psychology*, *12*(1), 97–136.

Treisman, A. M., & Sato, S. (1990). Conjunction search revisited. *Journal of Experimental Psychology: Human Perception and Performance*, *16*(3), 459.

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on machine learning* (pp. 1096–1103).

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.-A., & Bottou, L. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, *11*(12).

Yarbus, A. L. (1967). *Eye movements and vision.* New York: Plenum Press.

Zhao, M., Gersch, T. M., Schnitzer, B. S., Dosher, B. A., & Kowler, E. (2012). Eye movements and attention: The role of pre-saccadic shifts of attention in perception, memory and the control of saccades. *Vision Research*, *74*, 40–60.

Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., & Torralba, A. (2017). Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *40*(6), 1452–1464.

A Appendix

A.1 EDSC-Net Model Code

```python
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import matplotlib.pyplot as plt
import os
import cv2
import numpy as np
import pickle

import loss
import data_skip
import CC

K = keras.backend
devices = tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(devices[0], True)
tf.config.experimental_run_functions_eagerly(True)


class Normalise(keras.layers.Layer):
    """ layer that normalise outputs into the range of 0 and 1"""
    def call(self, maps, eps=1e-7):
        min_map = tf.math.reduce_min(maps, axis=(1, 2, 3), keepdims=True)
        maps -= min_map

        max_map = tf.math.reduce_max(maps, axis=(1, 2, 3), keepdims=True)
        maps = tf.divide(maps, eps + max_map)
        return maps


class Clip(keras.layers.Layer):
    """ layer clips values in the range from 0 to 1 after adding gaussian noise"""
    def call(self, inputs):
        inputs = K.clip(inputs, 0., 1.)
        return inputs


"""input dimensions"""
input_shape = (224, 384, 3)


"""encoder based on VGG 16, including the first 13 layers"""

inputs = keras.layers.Input(shape=input_shape)
layer1 = keras.layers.Conv2D(64, 3,
                             padding='same',
                             activation='relu',
                             name='block1_conv1')(inputs)
layer2 = keras.layers.Conv2D(64, 3,
                             padding='same',
                             activation='relu',
                             name='block1_conv2')(layer1)
layer3 = keras.layers.MaxPool2D(2, 2,
                                name='block1_pool',
                                data_format="channels_last")(layer2)
layer4 = keras.layers.Conv2D(128, 3,
                             padding='same',
                             activation='relu',
                             name='block2_conv1')(layer3)
layer5 = keras.layers.Conv2D(128, 3,
                             padding='same',
                             activation='relu',
                             name='block2_conv2')(layer4)
layer6 = keras.layers.MaxPool2D(2, 2,
                                name='block2_pool',
                                data_format="channels_last")(layer5)
layer7 = keras.layers.Conv2D(256, 3,
                             padding='same',
                             activation='relu',
                             name='block3_conv1')(layer6)
layer8 = keras.layers.Conv2D(256, 3,
                             padding='same',
                             activation='relu',
                             name='block3_conv2')(layer7)
layer9 = keras.layers.Conv2D(256, 3,
                             padding='same',
                             activation='relu',
                             name='block3_conv3')(layer8)
layer10 = keras.layers.MaxPool2D(2, 2,
                                 name='block3_pool',
                                 data_format="channels_last")(layer9)
layer11 = keras.layers.Conv2D(512, 3,
                              padding='same',
                              activation='relu',
                              name='block4_conv1')(layer10)
layer12 = keras.layers.Conv2D(512, 3,
                              padding='same',
                              activation='relu',
                              name='block4_conv2')(layer11)
layer13 = keras.layers.Conv2D(512, 3,
                              padding='same',
                              activation='relu',
                              name='block4_conv3')(layer12)
layer14 = keras.layers.MaxPool2D(2, 2,
                                 name='block4_pool',
                                  padding='same',
                                  data_format="channels_last")(layer13)
layer15 = keras.layers.Conv2D(512, 3,
                               padding='same',
                               activation='relu',
                               name='block5_conv1')(layer14)
layer16 = keras.layers.Conv2D(512, 3,
                               padding='same',
                               activation='relu',
                               name='block5_conv2')(layer15)
layer17 = keras.layers.Conv2D(512, 3,
                               padding='same',
                               activation='relu',
                               name='block5_conv3')(layer16)
layer18 = keras.layers.MaxPool2D(2, 2,
                                  name='block5_pool',
                                  padding='same',
                                  data_format="channels_last")(layer17)


"""a decoder applies blocks that each perform bilinear upsampling
 paired with conv layers and concatenation"""

dec_layer1 = keras.layers.UpSampling2D(size=(2, 2),
                                        data_format='channels_last',
                                        interpolation='bilinear')(layer18)
dec_layer2 = keras.layers.Conv2D(512, 3,
                                  padding='same',
                                  activation='relu',
                                  name="decoder/conv1")(dec_layer1)

concat1 = keras.layers.concatenate([dec_layer2, layer14], axis=3)

dec_layer3 = keras.layers.Conv2D(512, 3,
                                  padding='same',
                                  activation='relu',
                                  name="decoder/conv2")(concat1)
dec_layer4 = keras.layers.Conv2D(256, 3,
                                  padding='same',
                                  activation='relu',
                                  name="decoder/conv3")(dec_layer3)
dec_layer5 = keras.layers.UpSampling2D(size=(2, 2),
                                        data_format='channels_last',
                                        interpolation='bilinear')(dec_layer4)

concat2 = keras.layers.concatenate([dec_layer5, layer10], axis=3)

dec_layer6 = keras.layers.Conv2D(256, 3,
                                  padding='same',
                                  activation='relu',
                                  name="decoder/conv4")(concat2)
dec_layer7 = keras.layers.Conv2D(128, 3,
                                  padding='same',
                                  activation='relu',
                                  name="decoder/conv5")(dec_layer6)
dec_layer8 = keras.layers.UpSampling2D(size=(2, 2),
                                        data_format='channels_last',
                                        interpolation='bilinear')(dec_layer7)

concat3 = keras.layers.concatenate([dec_layer8, layer6], axis=3)

dec_layer9 = keras.layers.Conv2D(128, 3,
                                  padding='same',
                                  activation='relu',
                                  name="decoder/conv6")(concat3)
dec_layer10 = keras.layers.Conv2D(64, 3,
                                  padding='same',
                                  activation='relu',
                                  name="decoder/conv7")(dec_layer9)
dec_layer11 = keras.layers.UpSampling2D(size=(2, 2),
                                        data_format='channels_last',
                                        interpolation='bilinear')(dec_layer10)
dec_layer13 = keras.layers.Conv2D(32, 3,
                                  padding='same',
                                  activation='relu',
                                  name="decoder/conv9")(dec_layer11)
dec_layer14 = keras.layers.UpSampling2D(size=(2, 2),
                                        data_format='channels_last',
                                        interpolation='bilinear')(dec_layer13)
dec_layer15 = keras.layers.Conv2D(16, 3,
                                  padding='same',
                                  activation='relu',
                                  name="decoder/conv10")(dec_layer14)
decoder_out = keras.layers.Conv2D(1, 3,
                                  padding='same',
                                  name="decoder/conv11")(dec_layer15)
decoder_out_normalised = Normalise()(decoder_out)

Skipnet = keras.Model(inputs=[inputs], outputs=[decoder_out_normalised])


"""load pretrained VGG16 weights"""
weights_path = 'vgg16-hybrid1365_weights_tf_dim_ordering_tf_kernels_notop.h5'
Skipnet.load_weights(weights_path, by_name=True)

"""training parameters"""
learning_rate = 1e-5
batch_size = 1

"""model compile with our loss and adam optimiser"""
```

```
Skipnet.compile(loss=loss.kld, optimizer=keras.optimizers.Adam(lr=learning_rate))
Skipnet.summary()

if __name__ == "__main__":
    """ add callback to save best weights when training on salicon or CAT2000"""
    weights_file_salicon = 'weights_skip/Skipnet_weights_plain_salicon2.h5'      # path for saving weights
    model_checkpoint_salicon = keras.callbacks.ModelCheckpoint(filepath=weights_file_salicon,
                                                               save_weights_only=True,
                                                               monitor='val_loss',
                                                               mode='min',
                                                               save_best_only=True)

    weights_file_cat2000 = 'weights_skip/Skipnet_weights_plain_cat2.h5'          # path for saving weights
    model_checkpoint_cat2000 = keras.callbacks.ModelCheckpoint(filepath=weights_file_cat2000,
                                                               save_weights_only=True,
                                                               monitor='val_loss',
                                                               mode='min',
                                                               save_best_only=True)

    """train on SALICON"""
    history = Skipnet.fit(data_skip.salicon_generator_train,
                          validation_data=data_skip.salicon_generator_val,
                          validation_steps=5000/batch_size,
                          steps_per_epoch=10000/batch_size,
                          epochs=10,
                          callbacks=[model_checkpoint_salicon])

    np.save('histories_skip/history_Salicon_plain2.npy', history.history)


    # plot loss
    pd.DataFrame(history.history).plot(figsize=(8, 5))
    plt.grid(True)
    plt.show()


    """train on CAT2000, continue from salicon checkpoint"""
    Skipnet.load_weights(weights_file_salicon)
    history = Skipnet.fit(data_skip.cat_train_gen_new,
                          validation_steps=300 / batch_size,
                          steps_per_epoch=1400 / batch_size,
                          validation_data=data_skip.cat_val_gen_new,
                          epochs=15,
                          callbacks=[model_checkpoint_cat2000])

    np.save('histories_skip/history_CAT2000_plain2.npy', history.history)


    # plot loss
    pd.DataFrame(history.history).plot(figsize=(8, 5))
    plt.grid(True)
    plt.show()
```

A.2 Loss Function Code

```
import tensorflow as tf
from tensorflow import keras
K = keras.backend
"""credit to  https://github.com/alexanderkroner/saliency"""


def kld(y_true, y_pred, eps=1e-7):
    """
    calculated the KLD between ground truth
    and and predicted saliency map
    """
    sum_per_image = tf.reduce_sum(y_pred, axis=(1, 2, 3), keepdims=True)
    y_pred /= eps + sum_per_image
    loss = y_true * tf.math.log(eps + y_true / (eps + y_pred))
    loss = tf.reduce_mean(tf.reduce_sum(loss, axis=(1, 2, 3)))

    return loss
```

A.3 Code for Data Prepossessing and Loading

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf

"""
creating generators to load the salicon and the CAT2000 dataset during training, i.e.
images and ground truth saliency maps

"""
# parameters
batchsize = 1
seed = 100
targetsize_cat = (224, 384)
targetsize_salicon = (224, 320)



def create_distribution(salmap):
    """a function to convert the ground truth saliency maps
    into a probability density function"""
    eps = 1e-7
    sum_of_image = tf.reduce_sum(salmap, keepdims=True)
    salmap /= (eps + sum_of_image)
    return salmap


# the datagenerator for salicon
datagen_images = ImageDataGenerator(rescale=1. / 255)
datagen_salmap = ImageDataGenerator(preprocessing_function=create_distribution)

"""Salicon"""
# generators for salicon training
image_generator = datagen_images.flow_from_directory('salicon/train/',
                                                     class_mode=None,
                                                     target_size=targetsize_salicon,
                                                     batch_size=batchsize,
                                                     seed=seed)

salmap_generator = datagen_salmap.flow_from_directory('salicon/maps/train/',
                                                      color_mode='grayscale',
                                                      class_mode=None,
                                                      target_size=targetsize_salicon,
                                                      batch_size=batchsize,
                                                      seed=seed)

salicon_generator_train = zip(image_generator, salmap_generator)

# generators for salicon validation
image_generator = datagen_images.flow_from_directory('salicon/val/',
                                                     class_mode=None,
                                                     target_size=targetsize_salicon,
                                                     batch_size=batchsize, seed=seed)

salmap_generator = datagen_salmap.flow_from_directory('salicon/maps/val/',
                                                      color_mode='grayscale',
                                                      class_mode=None,
                                                      target_size=targetsize_salicon,
                                                      batch_size=batchsize, seed=seed)

salicon_generator_val = zip(image_generator, salmap_generator)

"""Cat2000"""
datagen_images_cat = ImageDataGenerator(rescale=1. / 255)
datagen_salmap_cat = ImageDataGenerator(preprocessing_function=create_distribution)

# generators for Cat 2000 training
cat_train_images = datagen_images_cat.flow_from_directory('CAT2000_split/Stimuli/Train_',
                                                         class_mode=None,
                                                         target_size=targetsize_cat,
                                                         batch_size=batchsize,
                                                         seed=seed,
                                                         follow_links=True)
```

```
cat_train_salmap = datagen_salmap_cat.flow_from_directory('CAT2000_split/Fixation_maps/Train_',
                                                          color_mode='grayscale',
                                                          class_mode=None,
                                                          target_size=targetsize_cat,
                                                          batch_size=batchsize,
                                                          seed=seed,
                                                          follow_links=True)

cat_train_gen_new = zip(cat_train_images, cat_train_salmap)


# generators for Cat 2000 validation
cat_image_val_new = datagen_images_cat.flow_from_directory('CAT2000_split/Stimuli/Val_',
                                                   class_mode=None,
                                                   target_size=targetsize_cat,
                                                   batch_size=batchsize,
                                                   seed=seed,
                                                   follow_links=True)

cat_salmap_val_new = datagen_salmap_cat.flow_from_directory('CAT2000_split/Fixation_maps/Val_',
                                                    color_mode='grayscale',
                                                    class_mode=None,
                                                    target_size=targetsize_cat,
                                                    batch_size=batchsize,
                                                    seed=seed,
                                                    follow_links=True)

cat_val_gen_new = zip(cat_image_val_new, cat_salmap_val_new)

# generators for CAT2000 test

cat_image_test_new = datagen_images_cat.flow_from_directory('CAT2000_split/Stimuli/Test_',
                                                   class_mode=None,
                                                   target_size=targetsize_cat,
                                                   batch_size=batchsize,
                                                   seed=seed,
                                                   follow_links=True)

cat_salmap_test_new = datagen_salmap_cat.flow_from_directory('CAT2000_split/Fixation_maps/Test_',
                                                    color_mode='grayscale',
                                                    class_mode=None,
                                                    target_size=targetsize_cat,
                                                    batch_size=batchsize,
                                                    seed=seed,
                                                    follow_links=True)

cat_test_gen_new = zip(cat_image_test_new, cat_salmap_test_new)
```