# DYNAMIC CAPACITY NETWORKS

**Amjad Almahairi**[1]**, Nicolas Ballas**[1]**, Tim Cooijmans**[1]**, Yin Zheng**[2]**,**
**Hugo Larochelle**[3] **& Aaron Courville**[1]
[1]MILA, Université de Montréal.
[2]Hulu.
[3]Twitter Cortex.

## ABSTRACT

We introduce the Dynamic Capacity Network (DCN), a neural network that can adaptively assign its capacity across different portions of the input data. This is achieved by combining modules of two types: low-capacity sub-networks and high-capacity sub-networks. The low-capacity sub-networks are applied across most of the input, but also provide a guide to select a few portions of the input on which to apply the high-capacity sub-networks. The selection is made using a novel gradient-based attention mechanism, that efficiently identifies the modules and input features for which the DCN's output is most sensitive and to which we should devote more capacity. We focus our empirical evaluation on the cluttered MNIST and SVHN image datasets. Our findings indicate that DCNs are able to drastically reduce the number of computations, compared to traditional convolutional neural networks, while maintaining similar performance.

## 1 INTRODUCTION

Deep neural networks have recently exhibited state-of-the-art performance across a wide range of tasks, including object recognition (Szegedy et al., 2014) and speech recognition (Graves & Jaitly, 2014). Top-performing systems, however, are based on very deep and wide networks that are computationally intensive. Those networks lead to time-consuming training and inference (prediction), in large part because they require a larger number of weight/activation multiplications.

In this work we exploit the observation that task-relevant information is often not uniformly distributed across the input data. For example, objects in images are spatially localized, i.e. they exist only in specific sub-regions of the image. Yet, an underlying assumption of many deep models is that all input regions contain the same amount of information. Indeed, convolutional neural networks apply the same set of filters uniformly across the spatial input (Szegedy et al., 2014), while recurrent neural networks apply the same input-to-hidden transformation at every time step (Graves & Jaitly, 2014). We argue that we should be able to drastically reduce computations, without significant loss in performance, by applying high capacity sub-networks only on the input's most informative regions. Such reduction in computations would be especially beneficial when dealing with very high-dimensional inputs, such as long and high-resolution videos. This is the same motivation behind much of the recent surge of interest in *attention-based* systems (Mnih et al., 2014), that learn to focus the model capacity on salient sub-regions of the input. However, the models proposed so far require defining an explicit predictive model of attention, whose training can pose challenges.

We introduce the *Dynamic Capacity Network* (DCN) that adaptively assigns its capacity across different portions of the input data via a gradient-based attention process that avoids the difficult task of training a separate attention network. The DCN combines two types of modules: small, low-capacity, sub-networks that are active on the whole input, and large, high-capacity, sub-networks which are directed, via our attention mechanism, to task-relevant sub-regions of the input.

The DCN model can be trained end-to-end, where we jointly learn the low and high-capacity modules. We validate this end-to-end approach in an image classification task using the cluttered MNIST dataset (Mnih et al., 2014), on which we outperform the state-of-the-art. Furthermore, we demonstrate that DCNs can be leveraged in a transfer-learning scenario where the low and high capacity modules are trained independently and only combined at test-time. We investigate this transfer-

learning setting in a more challenging task of transcribing multi-digit sequences from natural images using the Street View House Numbers (SVHN) dataset (Netzer et al., 2011). In particular, we show that DCN is able to efficiently recognize multi-digit sequences, directly from the original images, without using any prior information on the digits location.

## 2 DYNAMIC CAPACITY NETWORKS

We start by considering a separation of the computations in a deep neural network into two parts: the *bottom layers* and the *top layers*. Often, This separation should be made strategically, such that the bulk of computations are made within the bottom layers. For instance, in a convolutional neural network, the bottom layers would contain most of the convolutional and pooling layers, while the top layers would consist mostly of the network's fully connected layers. In the remainder of this paper, we will focus our description of the DCN in the context of this specific case.

We denote the decomposition into top layers and bottom layers as $h(\mathbf{x}) = g(f(\mathbf{x}))$ with $g$ and $f$ representing the top and bottom layers respectively. We consider the application of two alternative sub-networks for the bottom layers: the *coarse layers* $f_c$ or the *fine layers* $f_f$. As bottom layers, the coarse and fine layers both operate directly on the input and produce feature maps, i.e. feature vectors for each position (e.g. 2D spatial positions for images) in the input. However, coarse and fine layers differ in their capacity. The fine layers correspond to a high-capacity sub-network which has a high-computational requirement, while the coarse layers constitute a low-capacity sub-network. As for the top layers $g$, they can take features from either type of bottom layers and output a distribution over labels.

Consider applying the top layers only on the fine features, i.e. $h(\mathbf{x}) = g(f_f(\mathbf{x}))$, where $\mathbf{x}$ is some input data. We refer to the composition $h = g \circ f_f$ as the *fine model*. We assume that the fine model can achieve very good performance, but is computationally expensive. Alternatively, consider applying the top layers only on the coarse features of $\mathbf{x}$, i.e. $h_c(\mathbf{x}) = g(f_c(\mathbf{x}))$. We refer to this composition $h_c = g \circ f_c$ as the *coarse model*. Conceptually, the coarse model can be much more computationally efficient, but is expected to have worse performance than the fine model.

The key idea behind DCN is to have $g$ use features from either the coarse or fine models in an adaptive, dynamic way. Specifically, we apply the coarse layers $f_c$ on the whole input $\mathbf{x}$, and leverage the fine layers $f_f$ only on a few "important" subsets of the input, where the fine features replace the coarse features. This way, the DCN can leverage the capacity of $f_f$, but at a lower computational cost, by applying the fine features only on salient subsets of the input. To achieve this, we have two requirements.

First, we must define an attentional mechanism, whose task is to identify the positions of features in $f_c(\mathbf{x})$ that are good candidates to be replaced by the corresponding features from $f_f(\mathbf{x})$. For this, we use a novel approach for attention that uses backpropagation to identify the features to which the distribution over the class label is most sensitive. This mechanism, along with the complete process for computing the DCN's output, is described in Section 2.1.

Second, we require that the learned representations in $f_f(\mathbf{x})$ and $f_c(\mathbf{x})$ be interchangeable. We impose that the output number of feature maps in $f_f(\mathbf{x})$ and $f_c(\mathbf{x})$ have to be the same[1]. This will allow us to construct a heterogeneous input to $g$ that can mix dimensions from both $f_f$ and $f_c$.

We propose a procedure to train the DCN model end-to-end, which leverages our attention mechanism to learn $f_f$ and $f_c$ jointly, and encourages their representations to be both discriminative for the task at hand and comparable. Training the DCN is described in detail in Section 2.2. In addition, we also explore in Section 4.2 applying DCN to a transfer-learning scenario, where the outputs of pre-trained $f_f$ and $f_c$ are combined using our attention mechanism.

In the remainder of this section, we focus on 2-dimensional inputs. However, our DCN model can be easily extended to be applied to any type of N-dimensional data.
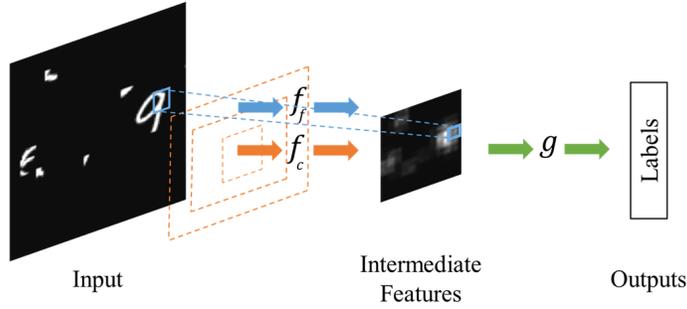
Figure 1: DCN overview. Our model applies the coarse layers on the whole image to get $f_c(\mathbf{x})$, applies the fine model only on the salient patches $\mathbf{X}^s$ to obtain few fine features $f_f(\mathbf{X}^s)$, and combines them to make the final prediction.

## 2.1 ATTENTION-BASED INFERENCE

As mentioned, inference in the DCN model starts by computing all coarse features of $f_c$ and then identifying a few input region positions at which to use instead the fine features of $f_f$.

Given an input image $\mathbf{x}$, we first apply the coarse layers on all input regions to compute the coarse feature maps $f_c(\mathbf{x}) \in \mathbb{R}^{c \times d_1 \times d_2}$, where $d_1$ and $d_2$ are spatial dimensions and $c$ is the number of features. We can think of $f_c(\mathbf{x})$ as a collection of $c$-dimensional feature vectors each located in a specific spatial location. We denote the a feature vector located at the position $(i, j)$ in the coarse feature maps by $f_c(\mathbf{x})_{i,j}$. Each one of those feature vectors corresponds to a specific receptive field or a patch (input region), in the input image $\mathbf{x}$. We then compute the output of the model based on the coarse feature maps, i.e. the coarse model's output $h_c(\mathbf{x}) = g(f_c(\mathbf{x}))$.

Next, we would like to obtain better predictions than those made by the coarse model, by selecting few positions in $f_c(\mathbf{x})$ where we replace the coarse feature vectors with fine ones. The corresponding receptive fields of the selected feature vectors defines our *salient* input regions. We propose to identify those salient input regions using an attentional mechanism that exploits a *saliency map* generated from the coarse model output. The specific measure of saliency we choose is based on the *entropy* of the coarse model output, defined as:

$$H = -\sum_{l=1}^{C} h_c^{(l)} \log h_c^{(l)}, \tag{1}$$

where $h_c = g(f_c(\mathbf{x}))$ is the output of the coarse model and $C$ is the number of class labels. The saliency $\mathbf{M}$ at position $(i, j)$ is given by the norm of the gradient of the entropy $H$ with respect to the coarse feature vector $f_c(\mathbf{x})_{i,j}$ at that position:

$$M_{i,j} = ||\nabla_{f_c(\mathbf{x})_{i,j}} H||_2 = \sqrt{\sum_{s=1}^{c} \left( \frac{\partial}{\partial f_c(\mathbf{x})_{i,j,s}} - \sum_{l=1}^{C} h_c^{(l)} \log h_c^{(l)} \right)^2}, \tag{2}$$

where $\mathbf{M} \in \mathbb{R}^{d_1 \times d_2}$. Our use of entropy gradient encourages the selection of feature vectors that could most affect the model's prediction uncertainty. In addition, computing the entropy of the output distribution does not require observing the true label, which makes it applicable at inference time. Note that computing all entries in matrix $\mathbf{M}$ can be done using a single backward pass of backpropagation and is thus efficient and simple to implement.

Using the saliency map $\mathbf{M}$, we select a set of $k$ positions of feature vectors with the highest saliency values. We denote the set of feature positions by $\mathbf{I}^s = \{(i, j); i \in [1, d_1], j \in [1, d_2]\}$. The set of corresponding patches is $\mathbf{X}^s = \{\mathbf{x}_{i,j}; (i, j) \in \mathbf{I}^s\}$, where $\mathbf{x}_{i,j}$ is the receptive field of the feature located at $(i, j)$. Next we apply the fine layers $f_f$ on the selected patches and obtain a small set of fine features, $f_f(\mathbf{X}^s) = \{f_f(\mathbf{x}_{i,j}); \mathbf{x}_{i,j} \in \mathbf{X}^s\}$. We assume that $f_f(\mathbf{x}_{i,j}) \in \mathbb{R}^c$, i.e. they have the

---

[1] The spatial size of $f_f(\mathbf{x})$ and $f_c(\mathbf{x})$ can be different, as long as the top layers can dynamically adapt to different input sizes, e.g. using pooling.

same dimensionality as the coarse features, allowing us to swap in the fine features in place of the corresponding coarse features. As a result, we obtain *refined feature maps* $f_r(\mathbf{x})$ composed of both coarse and fine features:

$$f_r(\mathbf{x}) = \begin{cases} f_f(\mathbf{x}_{i,j}), & \text{if } \mathbf{x}_{i,j} \in \mathbf{X}^s \\ f_c(\mathbf{x}_{i,j}), & \text{otherwise.} \end{cases} \tag{3}$$

Finally, the DCN output is obtained by feeding our refined feature maps into the top layers, $g(f_r(\mathbf{x}))$. We denote the composition $g \circ f_r$ by the *refined model*.

## 2.2 TRAINING

In the context of image classification, suppose we have a training set $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)}); i = 1 \ldots m\}$, where each $\mathbf{x}^{(i)} \in \mathbb{R}^{h \times w}$ is an image, and $y^{(i)} \in \{1, \ldots, C\}$ is its corresponding label. We denote the parameters of the coarse, fine and top layers by $\theta_c$, $\theta_f$, and $\theta_t$ respectively. We learn all of these parameters (denoted as $\theta$) by minimizing the cross-entropy objective function (which is equivalent to maximizing the log-likelihood of the correct labels):

$$J = -\sum_{i=1}^{m} \log p\left(y^{(i)} \mid \mathbf{x}^{(i)}; \theta\right), \tag{4}$$

where $p(. \mid \mathbf{x}^{(i)}; \theta) = g(f_r(\mathbf{x}^{(i)}))$ is the conditional multinomial distribution defined over the $C$ labels given by the refined model (Figure 1). Gradients are computed by standard back-propagation through the refined model, i.e. propagating gradients at each position into either the coarse or fine features, depending on which was used.

An important aspect of the DCN model is that the final prediction is based on combining features from two different sets of layers, namely the coarse layers $f_c$ and the fine layers $f_f$. Intuitively, we would like those features to have close values such that they can be interchangeable. This is important for two reasons. First, we expect the top layers to have more success in correctly classifying the input if the transition from coarse to fine features is smooth. The second is that, since the saliency map is based on the gradient *at the coarse feature values* and since the gradient is a local measure of variation, it is less likely to reflect the benefit of using the fine features if the latter is very different from the former.

To encourage similarly between the coarse and fine representations, we use a hint-based training approach inspired by Romero et al. (2014). Specifically, we add an additional term to the training that minimizes the the squared distance between coarse and fine representations:

$$\sum_{i=1}^{k} \|f_c(\mathbf{x}_i^s) - f_f(\mathbf{x}_i^s)\|_2^2. \tag{5}$$

There are two important points to note here. First, we use this term to optimize *only the coarse layers*. That is, we encourage the coarse layers to mimic the fine ones, while letting fine layers focus only on the signal coming from top layers. Secondly, computing the above *hint* objective over features at all positions would be as expensive as computing the full fine model; therefore, we encourage similarity only over the selected salient patches. The role of this term is discussed further in Section 6.3.

## 3 RELATED WORK

This work can be classified as a conditional computation approach. The goal of conditional computation, as put forward by Bengio (2013), is to train very large models for the same computational cost of smaller ones, by avoiding certain computation paths depending on the input. There have been several attempts in this direction. Bengio et al. (2013) use stochastic neurons as gating units that activate specific parts of a neural network. Our approach, on the other hand, uses an attention mechanism that helps the model to focus its computationally expensive paths only on important input regions, which helps in both scaling to larger effective models and larger input sizes.

| Model | Test Error |
|---|---|
| DRAW | 3.36% |
| Coarse Model | 3.69% |
| Fine Model | 1.70% |
| DCN, 8 patches, 14x14 | **1.39**% |

Table 1: Results on Cluttered MNIST dataset.

Several recent contributions use various attention mechanisms to capture visual structure with biologically inspired, foveation-like methods, e.g. (Larochelle & Hinton, 2010; Denil et al., 2012; Ranzato, 2014; Mnih et al., 2014; Ba et al., 2014; Gregor et al., 2015). In Mnih et al. (2014); Ba et al. (2014) a learned sequential attention model is used to make a hard decision of as to where to look in the image, i.e. which glimpse of the image is considered in each time step. This so-called "hard" attention mechanism can reduce computation for inference. The learning of the attention mechanism is formulated as a policy search problem in a reinforcement learning setup. In practice, this approach can be computationally expensive during training, as a result of needing to sample multiple interaction sequences with the environment. On the other hand, the DRAW model (Gregor et al., 2015) uses a "soft" attention mechanism that is fully differentiable, but requires processing the whole input in each time step. Our approach provides a simpler attention mechanism with computational advantages in both inference and learning.

The use of a regression cost for enforcing representations to be close has been exploited for achieving model compression (Bucilu et al., 2006; Hinton et al., 2015; Romero et al., 2014). The goal of model compression is to train a small model, which is faster in deployment, to imitate a much larger model (or an ensemble of models). Furthermore, Romero et al. (2014) have shown that middle layer hints can improve learning in deep and thin neural networks. In contrary to these approaches, our DCN model does not require training a large model a priori.

Other works such as matrix factorization (Jaderberg et al., 2014; Denton et al., 2014) and quantization schemes (Chen et al., 2010; Jégou et al., 2011; Gong et al., 2014) focus on speeding-up at the expense of slightly deteriorating their performances. Those works apply the same transformation on the model weights in "input-independent" fashion. By contrast, DCN effectively speeds-up a model in an adaptive, "input-dependent" way, by assigning its capacity differently across portions of the input. Such approaches are complementary to DCN and could be used to further speed-up the DCN model.

## 4 EXPERIMENTS

In this section, we present an experimental evaluation of the proposed DCN model. To validate the effectiveness of our approach, we first investigate the Cluttered MNIST dataset (Mnih et al., 2014). We then apply our model in a transfer-learning setting to a real-world object recognition task using the Street View House Numbers (SVHN) dataset (Netzer et al., 2011).

### 4.1 CLUTTERED MNIST

We use the $100 \times 100$ Cluttered MNIST digit classification dataset (Mnih et al., 2014). Each image in this dataset is an MNIST hand-written digit located randomly on a $100 \times 100$ black canvas and cluttered with digit-like fragments. Therefore, the dataset has the same size of MNIST: 60000 images for training and 10000 for testing.

In this experiment, we use 2 convolutional layers as coarse layers, 5 convolutional layers as fine layers, and one convolutional layer followed by global max pooling as the top layers. Details of this architecture can be found in Appendix 6.1. We take here patches of size $14 \times 14$ pixels, and feed them to the fine layers, which map each patch into one spatial location.

Results of the fine model, coarse model and the DCN model with eight $14 \times 14$ patches are shown in Table 1. We can see that DCN performs significantly better than the previous state-of-the-art result achieved by DRAW. It also outperforms the fine model, which is a result of being able to focus only

(a) Sample of selected Patches.

(b) Number of patches vs. test error.
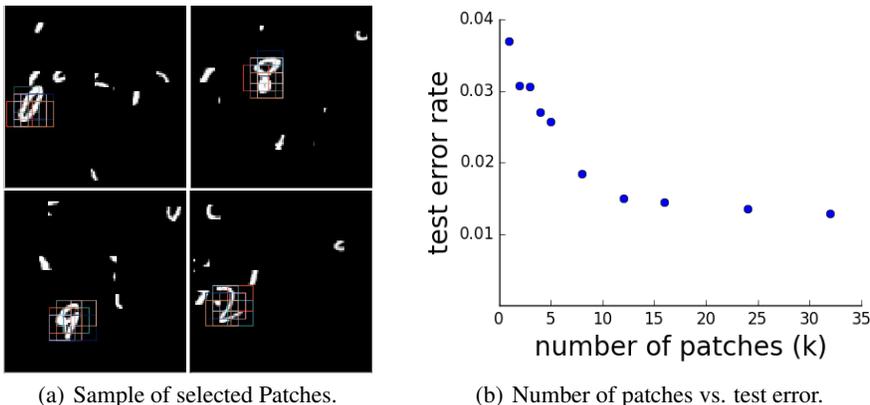
Figure 2: *Left*: Performance/computation trade-off in Cluttered MNIST. The more patches we select the better is the performance, but also the greater is the computation. *Right*: Quantitative and qualitative evaluation of the selected patches on Cluttered-MNIST.

on the digit and ignore clutter. To show how the attention mechanism of the DCN model can help it focus on the digit, we plot in Figure 2(a) the patches it finds in some images from the validation set, after only 9 epochs of training.

The DCN model is also more computationally efficient. In this experiment, a forward pass through the fine layers requires about 79.2M multiplications, while the coarse layers require only about 2.6M multiplications. Computing each patch using the fine model requires 1.15M multiplications, so computing 8 patches results in 9.2M multiplications, which in addition to the 2.6M multiplications from the coarse model would give 11.8M multiplications. The attention mechanism of the DCN model, however, requires an additional forward and backward pass through the top layers. Each pass through the top layers here requires 5.3M multiplications. Therefore, the fine model has a total of $5.3 + 79.2 = 84.5$M multiplications, while the DCN model has $3 \times 5.3 + 11.8 = 27.7$M multiplications. As a result, the DCN model here has 3 times fewer multiplications than the fine model.

We show in Figure 2(b) how the test error behaves when we increase the number of patches. We can see that choosing a lot of patches can be a waste of computation, and that performance does not improve significantly by taking more than 10 patches.

## 4.2 SVHN

We tackle in this section a more challenging task of transcribing multi-digit sequences from natural images using the Street View House Numbers (SVHN) dataset (Netzer et al., 2011). SVHN is composed of real-world pictures containing house numbers and taken from house fronts. The task is to recognize the full digit sequence corresponding to a house number, which can be of length 1 to 5 digits. The dataset has three subsets: train (33k), extra (202k) and test (13k). In the following, we trained our models on 230k images from both the train and extra subsets, where we take a 5k random sample as a validation set for choosing hyper-parameters.

The typical experimental setting in previous literature, e.g. (Goodfellow et al., 2013; Ba et al., 2014; Jaderberg et al., 2015), uses the location of digit bounding boxes as extra information. Input images are generally cropped, such that digit sequences are centred and most of the background and clutter information is pruned. We argue that our DCN model can deal effectively with real-world noisy images having large portions of clutter or background information. To demonstrate this ability, we investigate a more general problem setting where the images are uncropped and the digits locations are unknown. We apply our models on SVHN images in their original sizes and we do not use any extra bounding box information.

An important property of the SVHN dataset is the large discrepancy between the train/extra sets and the test set. Most training images have little background and are well-centred around the digits,

while most test images have large backgrounds with unrelated information. We discuss this further in Appendix 6.4. Given the specific architecture of DCN, we can leverage this training/test dataset discrepancy in a transfer-learning setting. We train the coarse and fine model independently on the training images that have little background-clutter, and then apply the DCN inference at test time to let our attention mechanism select where to apply the fine model. We show in the remainder of this section that our "test-time" variant of DCN can effectively deal with large background-clutter and apply the fine model on few input regions to refine predictions of the coarse model.

### 4.2.1 MODEL DESCRIPTION

We follow the model proposed in (Goodfellow et al., 2013) for learning a probabilistic model of the digit sequence given an input image $\mathbf{x}$. The output sequence $\mathbf{S}$ is defined using a collection of $N$ random variables, $S_1, \ldots, S_N$, representing the elements of the sequence and an extra random variable $S_0$ representing its length. The probability of a given sequence $\mathbf{s} = \{s_1, \ldots, s_n\}$ is given by:

$$p(\mathbf{S} = \mathbf{s} \mid \mathbf{x}) = p(S_0 = n \mid \mathbf{x}) \prod_{i=1}^{n} p(S_i = s_i \mid \mathbf{x}), \tag{6}$$

where $p(S_0 = n \mid \mathbf{x})$ is the conditional distribution of the sequence length and $p(S_i = s_i \mid \mathbf{x})$ is the conditional distribution of the $i$-th digit in the sequence. In particular, our models on SVHN have 6 outputs; the length of the sequence (from 1 to 5), and the identity of each digit or a null character if no digit is present (11 categories). Training is done by maximizing $\log p(\mathbf{S} \mid \mathbf{x})$ on the training set using SGD with a gradient computed using the backpropagation algorithm. We train coarse and fine models (details of their architectures can be found in Appendix 6.2). We convert all images to grayscale, and for the purposes of training only we resize images to $64 \times 128$.

### 4.2.2 INFERENCE WITH VARIABLE SIZED IMAGES

At test time, our baseline models (coarse and fine models) are applied, in a fully-convolutional way, on images in their original and variable sizes. Given an image $\mathbf{x}$, the model produces a probability map $\mathbf{P}$ which has a spatial size $d_1 \times d_2$, where the values of $d_1$ and $d_2$ depend on the image input. We denote the model's prediction for the $i$-th output at a specific spatial location $j, k$ by $p(S_{i,j,k} \mid \mathbf{x})$ where $S_{i,j,k}$ is a random variable representing the $i$-th element of the sequence at the $j, k$ location. The final prediction is computed by taking an average of the distributions over all spatial locations, i.e.

$$p(S_i \mid \mathbf{x}) = \frac{1}{d_1 \times d_2} \sum_{j,k} p(S_{i,j,k} \mid \mathbf{x}). \tag{7}$$

Since background information is expected to be dominant in input images, we argue that it is possible to improve model predictions by emphasizing predictions from "important" spatial locations. As in Section 2.1, we can leverage the *entropy* of the location-specific output as a measure of its importance. We can therefore define a "soft"-attention model which takes into account all location-specific outputs of the coarse but weighs them by their inverse entropies, i.e.

$$p(S_i \mid \mathbf{x}) = \sum_{j,k} w_{i,j,k} p(S_{i,j,k} \mid \mathbf{x}). \tag{8}$$

We define $w_{i,j,k}$ as the normalized inverse entropy of the $i$-th output distribution as predicted by the location $j, k$ in $\mathbf{P}$, i.e.

$$w_{i,j,k} = \sum_{p,q} \frac{H_{i,j,k}^{-1}}{H_{i,p,q}^{-1}}, \tag{9}$$

where $H_{i,j,k}$ is defined as:

$$H_{i,j,k} = -\sum_{l=1}^{C} p(S_{i,j,k} = s_l \mid \mathbf{x}) \log p(S_{i,j,k} = s_l \mid \mathbf{x}). \tag{10}$$
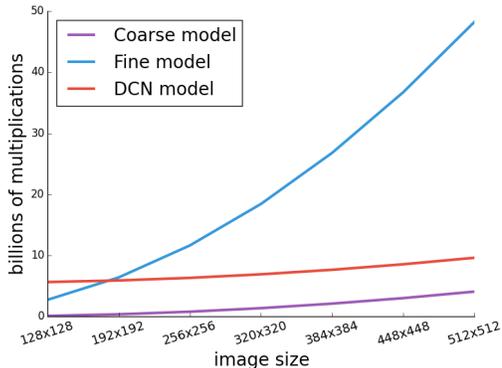
Figure 3: Number of multiplications in the Coarse, Fine and DCN models given different image input sizes.

| Model | Test Error |
|---|---|
| Coarse model, 1 scale | 40.6% |
| Coarse model, 2 scales | 40.0% |
| Coarse model, 3 scales | 40.0% |
| Fine model, 1 scale | 25.2% |
| Fine model, 2 scales | 23.7% |
| Fine model, 3 scales | 23.3% |
| Soft-attention, 1 scale | 31.4% |
| Soft-attention, 2 scales | 31.1% |
| Soft-attention, 3 scales | 30.8% |
| DCN, 6 patches, 1 scale | 20.0% |
| DCN, 6 patches, 2 scales | 18.2% |
| DCN, 9 patches, 3 scales | 16.6% |

Table 2: Results on SVHN dataset without using bounding box information.
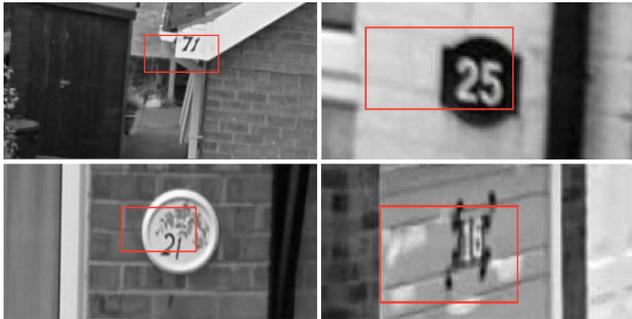


Figure 4: A sample of the selected patches in SVHN images. The images are processed by the DCN inference procedure in their original sizes. They are resized here for illustration purposes.

### 4.2.3 TEST-TIME DCN

In order to apply the DCN approach for this task, we need first to modify our gradient-based attention mechanism to handle multiple outputs. Consider the entropy associated with the $i$-th output:

$$H_i = - \sum_{l=1}^{C} p(S_i = s_l \mid \mathbf{x}) \log p(S_i = s_l \mid \mathbf{x}). \tag{11}$$

We compute the total entropy, denoted as $H$, by summing the entropy of all outputs. The saliency map is computed by taking the gradient of the total entropy $H$ with respect to the input of the coarse model's last layer. We compute here one saliency map based on all outputs because, as we describe shortly, we will use each selected patch to predict the full sequence independently. Using the saliency map, we select the $k_{\text{test}}$ most salient patches, and then give them to the fine model, producing $k_{\text{test}}$ independent predictions. Finally, we take the average of these predictions, each of them weighted by their normalized inverse entropies, as we did in equation (8).

We extend this procedure to a multi-scale approach by processing each image several times at multiple resolutions. This allows the fixed-size patch to capture larger areas of the image and helps us deal with scale variations in the data. We combine the $k_{\text{test}}$ patches selected at each resolution and feed them to the fine model to get $k_{\text{test}} \times N_{\text{scale}}$ independent predictions, where $N_{\text{scale}}$ is the number of scales. These predictions are again combined by taking their weighted average.

### 4.2.4 EMPIRICAL EVALUATION

In Table 2, we report the test errors of the coarse, fine, soft-attention and DCN models on the SVHN datasets, using directly the original images.

Applying the coarse model directly on the images, in a fully convolutional fashion, leads to an error rate of 40.6%. By using our proposed soft-attention mechanism, we decrease the error rate to 31.4%. This confirms that the entropy is a good measure for identifying important regions when task-relevant information is not uniformly distributed across input data.

The fine model, on the other hand, achieves a better error rate of 25.2%, but is more computationally expensive. Our DCN model, which selects only 6 regions on which to apply the high-capacity fine model, achieves an error rate of 20.0%. The DCN model can therefore outperform, in terms of classification accuracy, the other baselines, even in a "test-time" only setting where the fine and coarse model are trained independently. This verifies our assumption that by applying high capacity sub-networks only on the inputs most informative regions, we are able to obtain high classification performance. Figure 4 shows a sample of the selected patches by our attention mechanism.

An additional decrease of the test errors can be also obtained by increasing the number of processed scales. In the DCN model, taking 3 patches at 2 scales (original and 0.75 scales), leads to 18.2% error, while taking 3 patches at 3 scales (original, 0.75 and 0.5 scales) leads to an error rate of 16.6%. Our DCN model can reach its best performance of 11.6% by taking all possible patches at 3 scales, but it does not offer an computational benefits over the fine model.

We also investigate the computational benefits of the DCN approach as the dimensions of the input data increase. Table 3 reports the number of multiplications the fine model, coarse model and the DCN model require, given different input sizes. As described before, DCN is composed by coarse "low-capacity" and fine "high-capacity" sub-networks, which are convolutional models in our case. The number of multiplications in a convolutional model scales linearly with the input size. In DCN, we need to apply the "low-capacity" sub-network on the whole input while the "high-capacity" sub-network is used only for a fixed number of patches. Consequently, as the dimensions input size increase, the DCN's number of computations grows at the same rate as the coarse model, but the number of computations devoted to the "high-capacity" module remains constant as the input size grows. Most of the computational benefits of the DCN approach therefore appears for large inputs.

We verify the actual computational time of these models by taking the largest 100 images in the SVHN test set, and computing the average inference time taken by all the models [2]. The smallest of these images has a size of $363 \times 735$ pixels, while the largest has a size of $442 \times 1083$ pixels. On average, the coarse and the soft-attention models take 8.6 milliseconds, while the fine model takes 62.6 milliseconds. On the largest 100 SVHN test images, the DCN requires on average 10.8 milliseconds for inference.

## 5 CONCLUSIONS

We have presented the DCN model, which is a novel approach for conditional computation. We have shown that using our visual attention mechanism, our network can adaptively assign its capacity across different portions of the input data, focusing on important regions of the input. Our model achieved state-of-the-art performance on the Cluttered MNIST digit classification task, and provides computational benefits over traditional convolutional network architectures. We also validated our model on the SVHN dataset, where we tackled the multi-digit recognition problem without using any a priori information on the digits' location. We have shown that our model outperforms a coarse convolutional baseline, while still remaining computationally efficient for inputs with large spatial dimensions.

## REFERENCES

Ba, Jimmy, Mnih, Volodymyr, and Kavukcuoglu, Koray. Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755*, 2014.

---

[2]We evaluate all models on an NVIDIA Titan Black GPU card.

Bengio, Yoshua. Deep learning of representations: Looking forward. In *Statistical Language and Speech Processing*, pp. 1–37. Springer, 2013.

Bengio, Yoshua, Léonard, Nicholas, and Courville, Aaron. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Bucilu, Cristian, Caruana, Rich, and Niculescu-Mizil, Alexandru. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541. ACM, 2006.

Chen, Yongjian, Guan, Tao, and Wang, Cheng. Approximate nearest neighbor search by residual vector quantization. *Sensors*, 10(12):11259–11273, 2010.

Denil, Misha, Bazzani, Loris, Larochelle, Hugo, and de Freitas, Nando. Learning where to attend with deep architectures for image tracking. *Neural computation*, 24(8):2151–2184, 2012.

Denton, Emily L, Zaremba, Wojciech, Bruna, Joan, LeCun, Yann, and Fergus, Rob. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, pp. 1269–1277. 2014.

Gong, Yunchao, Liu, Liu, Yang, Min, and Bourdev, Lubomir. Compressing deep convolutional networks using vector quantization. *CoRR*, abs/1412.6115, 2014.

Goodfellow, Ian J, Bulatov, Yaroslav, Ibarz, Julian, Arnoud, Sacha, and Shet, Vinay. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*, 2013.

Graves, Alex and Jaitly, Navdeep. Towards end-to-end speech recognition with recurrent neural networks. In *ICML)*, 2014.

Gregor, Karol, Danihelka, Ivo, Graves, Alex, and Wierstra, Daan. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

Hinton, Geoffrey, Vinyals, Oriol, and Dean, Jeff. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Jaderberg, M., Vedaldi, A., and Zisserman, A. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

Jaderberg, Max, Simonyan, Karen, Zisserman, Andrew, and Kavukcuoglu, Koray. Spatial transformer networks. *arXiv preprint arXiv:1506.02025*, 2015.

Jégou, Hervé, Douze, Matthijs, and Schmid, Cordelia. Product quantization for nearest neighbor search. *IEEE TPAMI*, 33(1):117–128, 2011.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Larochelle, Hugo and Hinton, Geoffrey E. Learning to combine foveal glimpses with a third-order boltzmann machine. In *Advances in neural information processing systems*, pp. 1243–1251, 2010.

Mnih, Volodymyr, Heess, Nicolas, Graves, Alex, et al. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems*, pp. 2204–2212, 2014.

Netzer, Yuval, Wang, Tao, Coates, Adam, Bissacco, Alessandro, Wu, Bo, and Ng, Andrew Y. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 5. Granada, Spain, 2011.

Ranzato, Marc'Aurelio. On learning where to look. *arXiv preprint arXiv:1405.5488*, 2014.

Romero, Adriana, Ballas, Nicolas, Kahou, Samira Ebrahimi, Chassang, Antoine, Gatta, Carlo, and Bengio, Yoshua. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

## 6    APPENDIX

### 6.1    CLUTTERED MNIST EXPERIMENT DETAILS

- Coarse layers: 2 convolutional layers, with $7 \times 7$ and $3 \times 3$ filter sizes, 12 and 24 filters, respectively, and a $2 \times 2$ stride. Each feature in the coarse feature maps covers a patch of size $11 \times 11$ pixels, which we extend by 3 pixels in each side to give the fine layers more context. The size of the coarse feature map is $23 \times 23$.

- Fine layers: 5 convolutional layers, each with $3 \times 3$ filter sizes, $1 \times 1$ strides, and 24 filters. We apply $2 \times 2$ pooling with $2 \times 2$ stride after the second and fourth layers. We also use $1 \times 1$ zero padding in all layers except for the first and last layers. This architecture was chosen so that it maps a $14 \times 14$ patch into one spatial location.

- Top layers: one convolutional layer with $4 \times 4$ filter size, $2 \times 2$ stride and 96 filters, followed by global max pooling. The result is fed into a 10-output softmax layer.

We use rectifier non-linearities in all layers. We use Batch Normalization (Ioffe & Szegedy, 2015) and Adam (Kingma & Ba, 2014) for training our models. In DCN we train the coarse layers with a convex combination of cross entropy objective Eq. (4) and hints Eq. (5).

### 6.2    SVHN EXPERIMENT DETAILS

- Coarse model: the model is fully convolutional with 7 convolutional layers. First three layers have 24, 48, 128 filters respectively with size $5 \times 5$ and stride $2 \times 2$. Layer 4 has 192 filters with $4 \times 5$ and stride $1 \times 2$. Layer 5 has 192 filters with size $1 \times 4$. Finally, the last two layers are $1 \times 1$ convolutions with 1024 filters. We use stride of $1 \times 1$ in the last 3 layers and do not use zero padding in any of the coarse layers. The corresponding patch size here is $54 \times 110$.

- Fine layers: 9 convolutional layers, followed by 3 fully connected layers. The first 5 convolutional layers have 48, 64, 128, 160 and 192 filters respectively, with size $5 \times 5$ and zero-padding. After layers 1, 3, and 5 we use $2 \times 2$ max pooling with stride $2 \times 2$. The following layers have $3 \times 3$ convolution with 192 filters. Each of the 3 fully connected layers has 1024 hidden units.

Here we use SGD with momentum and exponential learning rate decay. While training, we take $54 \times 110$ random crop from images, and we use 0.2 dropout on convolutional layers and 0.5 dropout on fully connected layers.

### 6.3    EFFECT OF HINTS ON CLUTTERED MNIST TRAINING

Figure 5 shows the effect of adding the hint objective in Eq. (5) to the DCN objective during training on Cluttered MNIST. We notice that it can indeed minimize the squared distance between coarse and fine representations. This has a regularization effect, where we observe a drop in test error to 1.71% when we do not add it to the training objective.

### 6.4    SVHN TRAINING/TEST DISCREPANCY

Figure 6 shows samples from the extra and test subset of SVHN. The extra subset images dominate training data and are mostly composed of images with well-centred digits and little cluttered background. This is not the case for test images, which can have more variety in terms of digit location and background clutter.
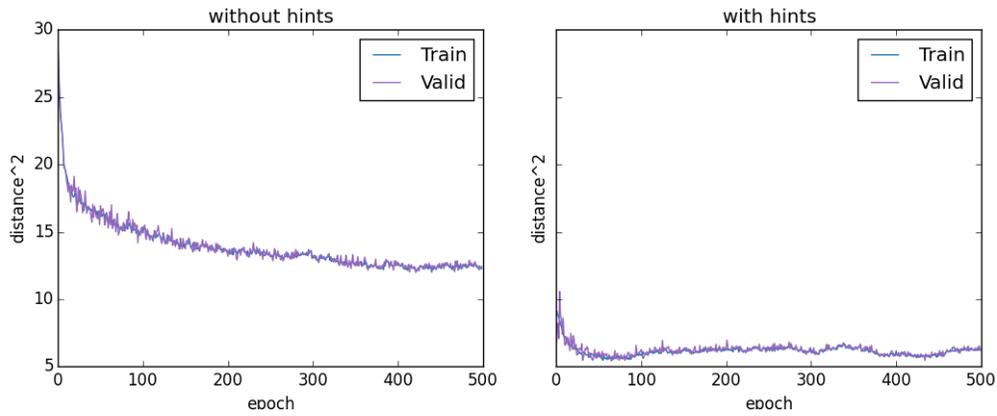
Figure 5: The effect of using the hints objective. We show the squared distance between coarse and fine features over salient regions during training in two cases: with and without using the hints objective. We observe that this regularizer helps in minimizing the distance and improves the model's generalization.



Figure 6: The 4 left images are samples from the extra subset, and the 4 right images are samples from the test subset. We notice that extra images are well-centred have much less background compared to test images.