IMAGE CLASSIFICATION THROUGH TOP-DOWN IMAGE PYRAMID TRAVERSAL

Anonymous authors
Paper under double-blind review

ABSTRACT

The available resolution in our visual world is extremely high, if not infinite. Existing CNNs can be applied in a fully convolutional way to images of arbitrary resolution, but as the size of the input increases, they can not capture contextual information. In addition, computational requirements scale linearly to the number of input pixels, and resources are allocated uniformly across the input, no matter how informative different image regions are. We attempt to address these problems by proposing a novel architecture that traverses an image pyramid in a top-down fashion, while it uses a hard attention mechanism to selectively process only the most informative image parts. We conduct experiments on MNIST and ImageNet datasets, and we show that our models can significantly outperform fully convolutional counterparts, when the resolution of the input is that big that the receptive field of the baselines can not adequately cover the objects of interest. Gains in performance come for less FLOPs, because of the selective processing that we follow. Furthermore, our attention mechanism makes our predictions more interpretable, and creates a trade-off between accuracy and complexity that can be tuned both during training and testing time.

1 INTRODUCTION

Our visual world is very rich, and there is information of interest in an almost infinite number of different scales. As a result, we would like our models to be able to process images of arbitrary resolution, in order to capture visual information with arbitrary level of detail. This is possible with existing CNN architectures, since we can use fully convolutional processing (Long et al. (2015)), coupled with global pooling. However, global pooling ignores the spatial configuration of feature maps, and the output essentially becomes a bag of features[1]. To demonstrate why this an important problem, in Figure 1 (a) and (b) we provide an example of a simple CNN that is processing an image in two different resolutions. In (a) we see that the receptive field of neurons from the second layer suffices to cover half of the kid's body, while in (b) the receptive field of the same neurons cover area that corresponds to the size of a foot. This shows that as the input size increases, the final representation becomes a bag of increasingly more local features, leading to the absence of coarse-level information, and potentially harming performance. We call this phenomenon the *receptive field problem* of fully convolutional processing.

An additional problem is that computational resources are allocated uniformly to all image regions, no matter how important they are for the task at hand. For example, in Figure 1 (b), the same amount of computation is dedicated to process both the left half of the image that contains the kid, and the right half that is merely background. We also have to consider that computational complexity scales linearly with the number of input pixels, and as a result, the bigger the size of the input, the more resources are wasted on processing uninformative regions.

We attempt to resolve the aforementioned problems by proposing a novel architecture that traverses an image pyramid in a top-down fashion, while it visits only the most informative regions along the way.

[1]There are forms of global pooling, like SPP (He et al. (2015)), that maintain spatial information in their output. However, all kinds of global pooling are based on simple grouping operations, because their primary goal is to reduce the spatial dimension of arbitrarily sized inputs. As a result, they cannot capture complex non-linear relationships between the input features. In addition, methods like SPP use a predefined number of bins, and consequently, they can only capture spatial relations in a limited number of different scales.

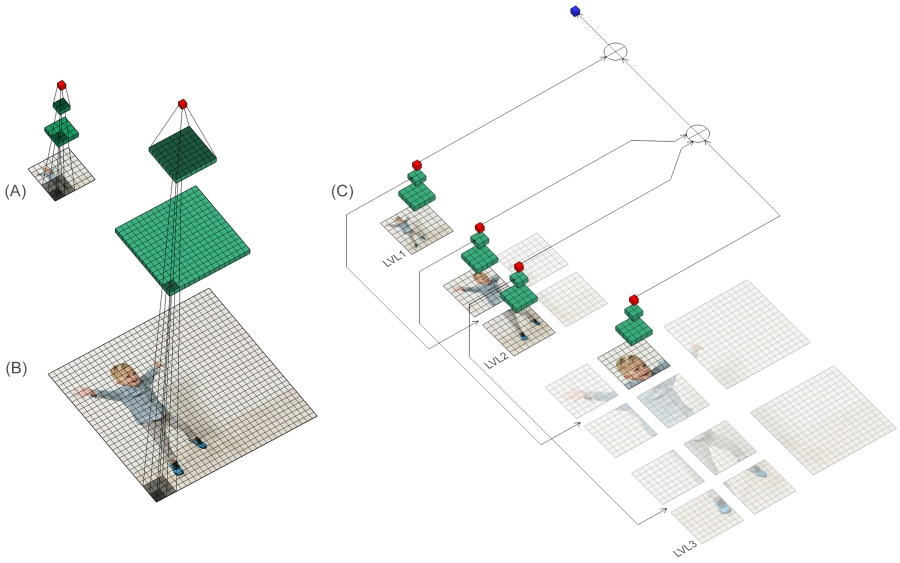

Figure 1: (a), (b) The *receptive field problem* of fully convolutional processing. A simple CNN consisted of 2 convolutional layers (colored green), followed by a global pooling layer (colored red), processes an image in two different resolutions. The shaded regions indicate the receptive fields of neurons from different layers. As the resolution of the input increases, the final latent representation becomes a bag of increasingly more local features, lacking coarse information. (c) A sketch of our proposed architecture. The arrows on the left side of the image demonstrate how we focus on image sub-regions in our top-down traversal, while the arrows on the right show how we combine the extracted features in a bottom-up fashion.

In Figure 1 (c) we provide a simplified sketch of our approach. We start at level 1, where we process the input image in low resolution, to get a coarse description of its content. The extracted features (red cube) are used to select out of a predefined grid, the image regions that are worth processing in higher resolution. This process constitutes a hard attention mechanism, and the arrows on the left side of the image show how we extend processing to 2 additional levels. All extracted features are combined together as denoted by the arrows on the right, to create the final image representation that is used for classification (blue cube).

We evaluate our model on synthetic variations of MNIST (LeCun et al., 1998) and on ImageNet (Deng et al., 2009), while we compare it against fully convolutional baselines. We show that when the resolution of the input is that big, that the receptive field of the baseline[2] covers a relatively small portion of the object of interest, our network performs significantly better. We attribute this behavior to the ability of our model to capture both contextual and local information by extracting features from different pyramid levels, while the baselines suffer from the receptive field problem. Gains in accuracy are achieved for less floating point operations (FLOPs) compared to the baselines, due to the attention mechanism that we use. If we increase the number of attended image locations, computational requirements increase, but the probability of making a correct prediction is expected to increase as well. This is a trade-off between accuracy and computational complexity, that can be tuned during training through regularization, and during testing by stopping processing on early levels. Finally, by inspecting attended regions, we are able to get insights about the image parts that our networks value the most, and to interpret the causes of missclassifications.

2 Related Work

Attention. Attention has been used very successfully in various problems (Bahdanau et al., 2014; Xu et al., 2015; Larochelle & Hinton, 2010; Gregor et al., 2015; Denil et al., 2012). Most similar to

[2]When we refer to the receptive field of a CNN, we refer to the receptive field of the neurons that belong to the last convolutional layer.

our work, are models that use recurrent neural networks to adaptively attend to a sequence of image regions, called glimpses (Ba et al. (2014); Mnih et al. (2014); Eslami et al. (2016); Ba et al. (2016); Ranzato (2014)). There are notable technical differences between such models and our approach. However, the difference that we would like to emphasize, is that we model the image content as a hierarchical structure, and we implicitly create a parsing tree (Zhu et al. (2007)), where each node corresponds to an attended location, and edges connect image regions with sub-regions (an example is provided in Appendix A.1). If we decide to store and reuse information, building such a tree structure offers a number of potential benefits, e.g. efficient indexing. We consider this an important direction to explore, but it is beyond the scope of the current paper.

Multi-scale representations. We identify four broad categories of multi-scale processing methods. (1) *Image pyramid methods* extract multi-scale features by processing multi-scale inputs (Eigen et al. (2014); Pinheiro & Collobert (2014); Najibi et al. (2018)). Our approach belongs to this category. (2) *Encoding schemes* take advantage of the inherently hierarchical nature of deep neural nets, and reuse features from different layers, since they contain information of different scale (Liu et al. (2016); He et al. (2014); Chen et al. (2018)). (3) *Encoding-Decoding schemes* follow up the feed-forward processing (encoding) with a decoder, that gradually recovers the spatial resolution of early feature maps, by combining coarse with fine features (Lin et al. (2017); Ronneberger et al. (2015)). (4) *Spatial modules* are incorporated into the feed forward processing, to alter the feature extraction between layers (Yu & Koltun (2015); Chen et al. (2017); Wang et al. (2019)).

Computational efficiency. We separate existing methods on adjusting the computational cost of deep neural networks, into four categories. (1) *Compression methods* aim to remove redundancy from already trained models (LeCun et al. (1990); Hinton et al. (2015); Yu et al. (2018)). (2) *Lightweight design strategies* are used to replace network components with computationally lighter counterparts (Howard et al. (2017); Iandola et al. (2016); Rastegari et al. (2016); Jaderberg et al. (2014); Wang et al. (2017)). (3) *Partial computation methods* selectively utilize parts of a network, creating paths of computation with different cost (Huang et al. (2017); Wu et al. (2018); Larsson et al. (2016); Figurnov et al. (2017); Zamir et al. (2017)). (4) *Attention methods* selectively process parts of the input, based on their importance for the task at hand (Katharopoulos & Fleuret (2019); Ramapuram et al. (2018); Levi & Ullman (2018)). This is the strategy that we follow in our architecture.

3 ARCHITECTURE

We present our architecture by walking through the example in Figure 2, where we process an image with original resolution of 128×128 px (① in the top left corner). In the fist level, we downscale the image to 32×32 px and pass it through the *feature extraction module*, in order to produce a feature vector V_1 that contains a coarse description of the original image. The *feature extraction module* is a CNN that accepts inputs in a fixed resolution, that we call *base resolution*. We can provide V_1 as direct input to the *classification module* in order to get a rapid prediction. If we end processing here, our model is equivalent to a standard CNN operating on 32×32 px inputs. However, since the original resolution of the image is 128×128 px, we can take advantage of the additional information by moving to the second processing level.

In the second level, we feed the last feature map, F_1, of the *feature extraction module*, to the *location module*. The *location module* considers a number of candidate locations within the image described by F_1, and predicts how important it is to process in higher detail each one of them. In this particular example, the candidate locations form a 2×2 regular grid (②), and the *location module* yields 4 probabilities, that we use as parameters of 4 Bernoulli distributions in order to sample a hard attention mask (③). Based on this mask, we crop the corresponding regions, and we pass them through the *feature extraction module*, creating V_{21} and V_{23}. If we want to stop at this processing level, we can directly pass V_{21} and V_{23} through the *aggregation module* (skipping the *merging module* ④). The *aggregation module* combines the features from individual image regions into a single feature vector V_1^{agg}, that describes the original image solely based on fine information. This means that V_1^{agg} is complementary to V_1, and both vectors are combined by the *merging module*, which integrates fine information (V_1^{agg}) with its context (V_1), creating a single comprehensive representation $V_1^{'}$. Then, $V_1^{'}$ can be used for the final prediction.

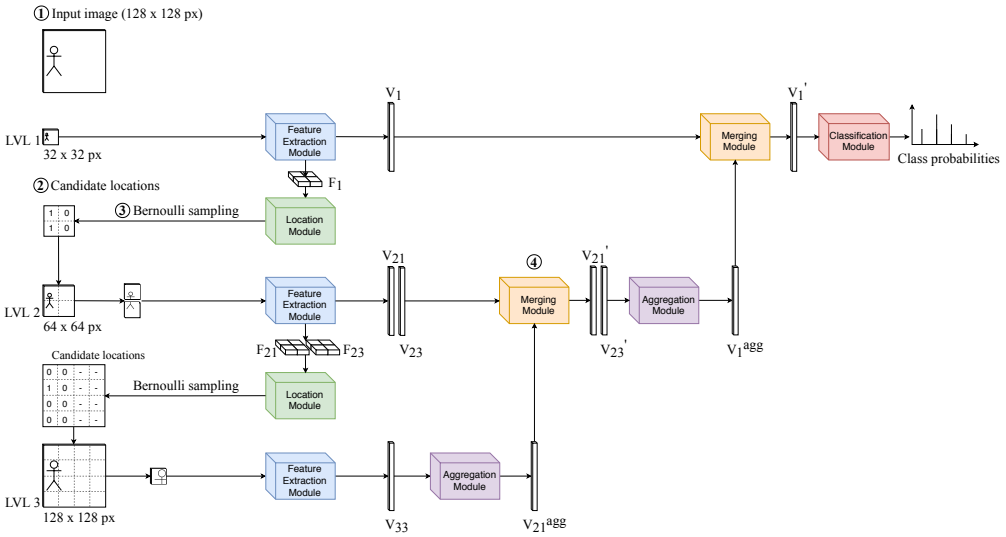

Figure 2: Three unrolled processing levels of our architecture. The network starts by processing the coarsest scale (*Feature Extraction Module*), and uses the extracted features to decide where to focus (*Location Module*). Features extracted from subsequent detail levels are combined together (*Aggregation Module*), and then are used to enrich the features from the coarser scales (*Merging Module*) before the final classification (*Classification Module*).

We can extend our processing to a third level, where F_{21} and F_{23} are fed to the *location module* to create two binary masks. No locations are selected from the image patch described by V_{23}, and the *aggregation module* only creates V_{21}^{agg}. Then, we start moving upwards for the final prediction. In Appendix A.2 we provide additional details about the modules of our architecture. In Appendix A.3 we express the feature extraction process with a single recursive equation.

4 Training

Our model is not end-to-end differentiable, because of the Bernoulli sampling involved in the location selection process. To overcome this problem, we use a variant of *REINFORCE* (Williams, 1992):

$$L_F = \frac{1}{N \cdot M} \sum_{i=1}^{N \cdot M} \left[\frac{\partial \log p(y_i|l^i, x_i, w)}{\partial w} + \lambda_f (\log p(y_i|l^i, x_i, w) - b) \frac{\partial \log p(l^i|x_i, w)}{\partial w} \right] \quad (1)$$

where $N \cdot M$ is the number of images we use for each update, x_i is the ith image, y_i is its label, and w are the parameters of our model. $p(l^i|x_i, w)$ is the probability that the sequence of locations l^i is attended for image x_i, and $p(y_i|l^i, x_i, w)$ is the probability of predicting the correct label after attending l^i. The size of our original batch B is N, but in the derivation of (1) we approximate with a Monte Carlo estimator of M samples, the expectation $\sum_{l^i} p(l^i|x_i, w) \left[\frac{\partial \log p(y_i|l^i, x_i, w)}{\partial w} + \log p(y_i|l^i, x_i, w) \frac{\partial \log p(l^i|x_i, w)}{\partial w} \right]$ for each image x_i in B. Based on this, for simplicity we just consider that our batch has size $N \cdot M$. b is a baseline that we use to reduce the variance of our estimators, and λ_f is a weighting hyperparameter. The first term of L_F allows us to update the parameters in order to maximize the probability of each correct label. The second term allows us to update the location selection process, according to the utility of attended locations for the prediction of the correct labels. In Appendix A.4.1 we provide the exact derivation of learning rule (1).

We experimentally identified two problems related to (1). First, our model tends to attend all the available locations, maximizing the computational cost. To regulate this, we add the following term:

$$R_t = -\lambda_t \frac{1}{2} \Big(\frac{1}{NM} \sum_{i=1}^{NM} p_t^{l^i} - c_t \Big)^2 \quad (2)$$

where $p_t^{l^i}$ approximates the expected number of attended locations for image x_i base on l^i, and quantity $\frac{1}{NM}\sum_{i=1}^{MN} p_t^{l^i}$ calculates the average expected number of attended locations per image in our augmented batch of NM images. The purpose of R_t is to make the average number of attended locations per image equal to c_t, which is a hyperparameter selected according to our computational cost requirements. λ_t is simply a weighting hyperparameter.

The second problem we identified while using learning rule (1), is that the learned attention policy may not be diverse enough. In order to encourages exploration, we add the following term:

$$R_r = -\lambda_r \frac{1}{g}\sum_{k=1}^{g}\frac{1}{2}(p_k - c_r)^2 \tag{3}$$

where p_k is the average probability of attending the kth out of the g candidate locations, that the location module considers every time it is applied during the processing of our NM images. R_r encourages the location module to attend with the same average probability c_r, locations that are placed at different regions inside the sampling grid. λ_r is a weighting hyperparameter. In Appendix A.4.2 we provide additional details about terms (2) and (3). Our final learning rule is the following:

$$L_r = L_F + \frac{\partial R_t}{\partial w} + \frac{\partial R_r}{\partial w} \tag{4}$$

Gradual Learning. The quality of the updates we make by using (4), depends on the quality of the Monte Carlo estimators. When we increase the number of processing levels that our model can go through, the number of location sequences that can be attended increases exponentially. Based on this, if we allow our model to go through multiple processing levels, and we use a small number of samples in order to have a moderate cost per update, we expect our estimators to have high variance. To avoid this, we separate training into stages, where in the first stage we allow only 2 processing levels, and at every subsequent stage an additional processing level is allowed. This way, we expect the location module to gradually learn which image parts are the most informative, narrowing down the location sequences that have a considerable probability to be attended at each training stage, and allowing a small number of samples to provide acceptable estimators. We call this training strategy *gradual learning*, and the learning rule that we use at each stage s, is the following:

$$L_r^s = L_F^s + \frac{\partial R_t^s}{\partial w} + \frac{\partial R_r^s}{\partial w} \tag{5}$$

where L_r^s is equivalent to (4), with s superscripts indicating that the hyperparameters of each term can be adjusted at each training stage. The maximum number of possible training stages depends on the resolution of the training images.

Multi-Level Learning. By following the gradual learning paradigm, a typical behavior that we observe is the following. After the first stage of training, our model is able to classify images with satisfactory accuracy by going through 2 processing levels. After the second stage of training, our model can go through 3 processing levels, and as we expect, accuracy increases since finer information can be incorporated. However, if we force our model to stop processing after 2 levels, the obtained accuracy is significantly lower than the one we were able to achieve after the first stage of training. This is a behavior that we observe whenever we finish a new training stage, and it is an important problem, because we would like to have a flexible model that can make accurate predictions after each processing level. To achieve this, we introduce the following learning rule:

$$L_{ml}^s = \sum_{z=0}^{s} \lambda_z^s \cdot L_r^z \tag{6}$$

where L_r^z is learning rule (5) with $z+1$ processing levels allowed for our model. Each λ_z^s is a hyperparameter that specifies the relative importance of making accurate predictions after processing level $z+1$, while we are at training stage s.

5 Experimental Evaluation

We experiment with two datasets, MNIST (LeCun et al., 1998) and ImageNet (Deng et al., 2009). MNIST is a small dataset that we can easily modify in order to test different aspects of our models'

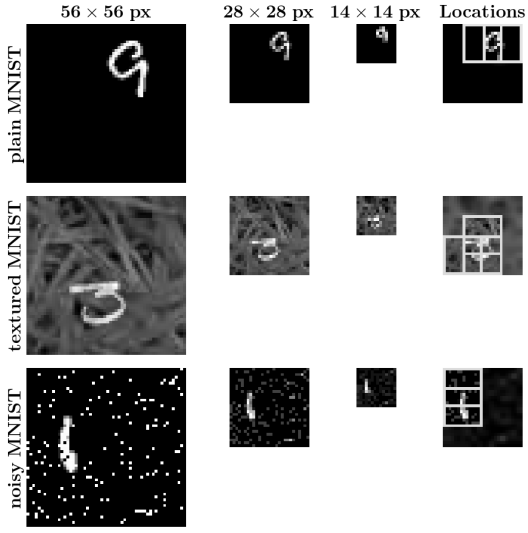

Figure 3: Example images from our MNIST-based datasets, along with the attended locations of M_3^{28} models. We blur the area outside the attended locations, because it is processed only in lower resolution during the first processing level. This way we aim to get a better understanding of what our models "see".

behavior, e.g. the localization capabilities of our attention mechanism. ImageNet has over a million training images, and allows us to evaluate our model on a large scale.

5.1 Experiments With MNIST

Data. MNIST is a dataset with images of handwritten digits that range from 0 to 9, leading to 10 different classes. All images are grayscale, and their size is 28×28 pixels. The dataset is split in a training set of $55,000$ images, a validation set of $5,000$ images, and a test set of $10,000$ images. We modify the MNIST images by placing each digit at a randomly selected location inside a black canvas of size 56×56. We refer to this MNIST variation as *plain MNIST*, and we further modify it to create *noisy MNIST* and *textured MNIST*. In the case of noisy MNIST, we add salt and pepper noise by corrupting 10% of the pixels in every image, with a 0.5 ratio between black and white noise. In the case of textured MNIST, we add textured background that is randomly selected from a large image depicting grass. Example images are provided in the first column of Figure 3.

Models. We create 3 pairs of models $\{M_i, BL_i\}_{i=1}^3$, each composed of a version M_i of our architecture, and a corresponding fully convolutional baseline BL_i for comparison. Ideally, we would like the feature extraction processes of the models that we compare to be consisted of the same layers, in order to eliminate important factors of performance variation like the type or the number of layers. To this end, in every pair of models, the baseline BL_i is equivalent to the feature extraction module of M_i followed by the classification module, with 2 fully connected layers in between. The 2 fully connected layers account for the aggregation and merging modules, which could be considered part of the feature extraction process in M_i. We create 3 pairs of models because we want to study how our architecture performs relatively to fully convolutional models with different receptive fields. To achieve this, BL_1 has 1 convolutional layer and receptive field 3×3 px, BL_2 has 2 convolutional layers and receptive field 8×8 px, and BL_3 has 3 convolutional layers and receptive field 20×20 px. We would like to note that all models $\{M_i\}_{i=1}^3$ have base resolution 14×14 px, and their location modules consider 9 candidate locations which belong to a 3×3 regular grid with 50% overlap. In Appendix A.5.1 we provide the exact architectures of all models.

Training. We describe how we train one pair of models (M_i, BL_i) with one of the 3 MNIST-based datasets that we created. The same procedure is repeated for all datasets, and for every pair of models. The original resolution of our dataset is 56×56 px, and we rescale it to 2 additional resolutions that

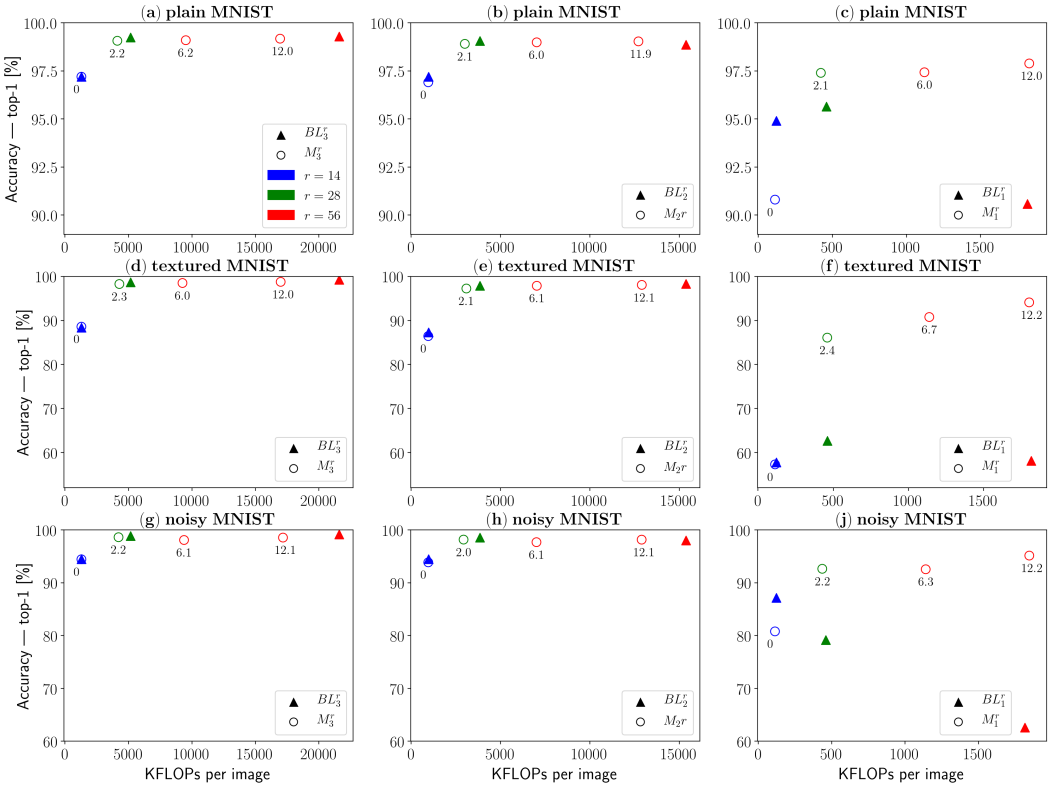

Figure 4: Experimental results on *plain*, *textured* and *noisy MNIST*. The differences in accuracy between many models were very small, and as a result, in the provided graphs we report the average of 20 different evaluations on the validation set, where each time we randomly change the positions of the digits inside the images. For *textured* and *noisy MNIST*, we randomly change the background and the noise pattern of each image as well.

are equal to 28×28 px, and 14×14 px (example images are provided in the first 3 columns of Figure 3). We split our training procedure in 3 sessions, where at each session we train our models with images of different resolution. In the first session we use images of size 14×14 px, and we refer to the resulting models as BL_i^{14} and M_i^{14}. We note that since the resolution of the input is equal to the base resolution of M_i, our model can go through just 1 processing level and the location module is not employed.

In our second training session we use images of resolution 28×28 px, and the increased resolution of the input allows M_i to use the location module and extend processing to 2 processing levels. Based on this, we are able to train M_i multiple times by assigning different values to hyperparameter c_t (2), resulting to models that attend a different average number of locations per image, and as a result, they have different average computational cost and accuracy. We refer to the models from this training session as BL_i^{28} and M_i^{28,n_c}, where n_c is the average number of locations that M_i attended on the validation set, while trained with $c_t = c$. We also define M_i^{28} as the set of all M_i^{28,n_c} models we trained in this session. In the third training session we use images of resolution 56×56 px, and M_i is able to go through 3 processing levels. Following our previous notation, we refer to the models of this session as BL_i^{56} and M_i^{56,n_c}, while we use M_i^{56} to denote the set of all M_i^{56,n_c}. In Appendix A.5.2 we provide additional details about the training sessions, along with the exact hyperparameters that we used to obtain the results that follow.

Results. In the first row of Figure 4, we present performance graphs for all models $\{M_i, BL_i\}_{i=1}^3$ on *plain MNIST*. We note that the annotations under all models $\{M_i\}_{i=1}^3$ indicate the average number of locations n_c that we described in the training section. We start by examining Figure 4 (a), where

Figure 5: Two missclasication examples of model $M_3^{28,2.2}$, that demonstrate the interpretability of our models' decisions because of the employed attention mechanism.

we depict the performance of models M_3 and BL_3 on images of different resolution. The overlapping blue markers indicate models M_3^{14} and BL_3^{14}, which achieve the same accuracy level. The green markers denote models BL_3^{28} and $M_3^{28,2.2}$[3], and as we expect, they achieve higher accuracy compared to BL_3^{14} and M_3^{14}, since they operate on images of higher resolution. The red markers denote models M_3^{56} and BL_3^{56}, and by observing the performance of all M_3 models, we see that as the resolution of the input and the number of attended locations increases, accuracy increases as well, which is the expected trade-off between computation and accuracy. This trade-off follows a form of logarithmic curve, that saturates in models M_3^{56}.

In the last column of the first row of Fig. 3, we provide a representative example of how the attention mechanism of $M_3^{28,2.2}$ operates. We observe that the location module is capable of focusing on the digit, and the generally good performance of the location module is reflected on the accuracy of the model, which is over 99%. However, BL_3^{28} performs slightly better, and in an attempt to understand why, in Fig. 5 we provide two example images which are missclassified by $M_3^{28,2.2}$, but are correctly classified by BL_3^{28}. In both examples, the attended locations partially cover the digits, leading 5 to be interpreted as 1, and 3 to be interpreted as 7. Of course, we are not always able to interpret the cause of a missclassification by inspecting the attended locations, but as the provided examples show, we may be able to get important insights. Besides the performance of our attention mechanism, we don't expect $M_3^{28,2.2}$ to achieve higher accuracy compared to BL_3^{28}. We remind that the images provided to the models are of size $28 \times 28\ px$, the digit within each image covers an area of maximum size $14 \times 14\ px$, and the receptive field of the baseline is $20 \times 20\ px$. As a result, the receptive field can perfectly cover the area of the digit, and the extracted features contain both coarse and fine information. Consequently, our model doesn't offer any particular advantage in terms of feature extraction that could be translated in better accuracy. This is something that we expect to change if the resolution of the input increases, or if the receptive field of the baseline gets smaller, as in the cases of BL_2 and BL_1.

In Fig. 4 (b) we present the performance of models M_2 and BL_2, and the main difference we observe with (a), is that BL_2^{56} demonstrates lower accuracy compared to BL_2^{28}. We attribute this behavior to the fact that the receptive field of BL_2^{56} covers less than 10% of the area occupied by each digit, and as a result, BL_2^{56} is unable to extract coarse level features which are valuable for classification. Based on this hypothesis, we are able to interpret the behavior of the models in (c) as well. We observe that M_1^{28} and M_1^{56} significantly outperform BL_1^{28} and BL_1^{56}, while BL_1^{56} demonstrates accuracy that is even lower than that of BL_1^{14}. We note that the receptive field of BL_1^{28} covers a little bit over 4.5% of the area occupied by each digit, while for BL_1^{56} this percentage drops to approximately 1%.

In the second and third row of Figure 4, we present our results on *textured* and *noisy MNIST* respectively. Our previous analysis applies to these results as well. In addition, we would like to note that our attention mechanism is robust to textured background and salt and pepper noise, as we can see in the respresentative examples provided in the last column of Fig. 3. In Appendix A.5.3 we provide some additional remarks on the results reported in Fig. 4.

5.2 Experiments On ImageNet

Data. We use the ILSVRC 2012 version of ImageNet, which contains $1,000$ classes of natural images. The training and validation sets contain $1,281,167$ and $50,000$ images, respectively. The average resolution of the original images is over $256\ px$ per dimension. All images are color images,

[3]We don't depict other M_3^{28} models that were trained with different c_t values, because they don't demonstrate any significant changes in accuracy, and would reduce the clarity of our graphs.

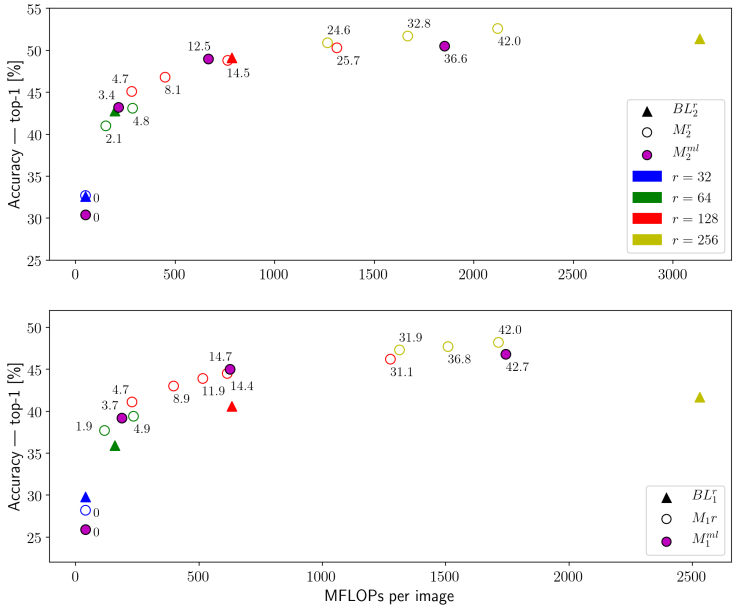

Figure 6: Experimental results on ImageNet. In the y-axis we provide the top-1 accuracy on the validation set, while in the x-axis we provide the required number of FLOPs ($\times 10^6$) per image.

but for simplicity, when we refer to resolution, we will drop the last dimension that corresponds to the color channels.

Models. We create two pairs of models $\{M_i, BL_i\}_{i=1}^2$ by following the same design principles we presented in Section 5.1. Model BL_1 has 3 convolutional layers and receptive field 18×18 px, while BL_2 has 4 convolutional layers and receptive field 38×38 px. Models $\{M_i\}_{i=1}^2$ have base resolution 32×32 px, and their location modules consider 9 candidate locations which belong to a 3×3 regular grid with 50% overlap. In Appendix A.6.1 we provide the architectures of all models.

Training. We follow the training procedure we described in Section 5.1. The main difference is that we rescale our images to 4 different resolutions $\{r \times r | r \in \{32, 64, 128, 256\}\}$, resulting to 4 training sessions. We follow the notation we introduced in Section 5.1l, and we denote the models that result from our first training session as M_i^{32} and BL_i^{32}. We refer to the models that result from the other 3 sessions as M_i^{r,n_c} and BL_i^r, while we use M_i^r to denote the set of all M_i^{r,n_c} models that are trained with different c_t values. For the second training session we have $r = 64$, for the third $r = 128$, and for the fourth $r = 256$, while $i \in \{1, 2\}$. Finally, we use multi-level learning rule (6) to train models that are able to demonstrate high accuracy after each processing level. The resulting models are denoted by $\{M_i^{ml}\}_{i=1}^2$. In Appendix A.6.2 we provide additional training details.

Results. In Figure 6 we provide our experimental results. As in Figure 4, markers of different color denote models which are trained with images of different resolution, while the annotations next to markers that correspond to models $\{M_i\}_{i=1}^2$ indicate the average number of locations n_c. In the first row of Fig. 6, we depict the performance of models M_2 and BL_2. First, we would like to note that we observe the trade-off between accuracy and computatinal complexity that we have identified in Fig. 4. When the number of required FLOPs increases by processing inputs of higher resolution, or by attending a bigger number of locations, accuracy increases as well.

By inspecting the behavior of individual models, we observe that BL_2^{64} has comparable performance to M_2^{64} models, while this is the case for BL_2^{128} and $M_2^{128,14.5}$ models as well. However, the behavior of models BL_2^{256} and M_2^{256} is different. Both $M_2^{256,32.8}$ and $M_2^{256,42}$ achieve higher accuracy compared to BL_2^{256}, even if they require much smaller number of FLOPs. Our explanation is based on the receptive field problem that we relate to fully convolutional models. The receptive field of BL_2^{64} covers around 35% of the image area, while this percentage drops close to 9% for

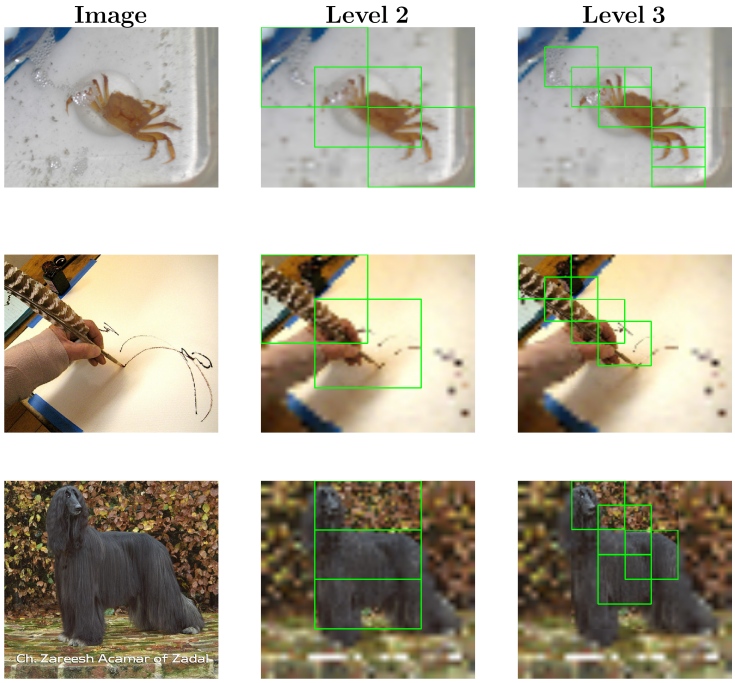

Figure 7: Examples of attended locations from a M_1^{128} model. Image parts are blurred according to the resolution they are processed.

BL_2^{128}, and to 2% for BL_2^{256}. As a result, we assume that the features extracted by BL_2^{256} lack coarse information, and we expect this phenomenon to become even more intense for models BL_1, since they have smaller receptive field. Indeed, as we can see in the second row of Fig. 6, the performance gap between M_1 and BL_1 models is bigger. M_1^{64}, M_1^{128} and M_1^{256} achieve higher accuracy compared to BL_1^{64}, BL_1^{128} and BL_1^{256} respectively.

We would also like to comment on the behavior of models M_2^{ml} and M_1^{ml} that we provide in the graphs of Fig. 6. We observe that both M_2^{ml} and M_1^{ml} are able to maintain comparable performance to models M_2^r and M_1^r respectively, in all processing levels. This shows that we are able to adjust the computational requirements of our models during testing time, by controlling the number of processing levels that we allow them to go through. This is beneficial when we face constraints in the available computational resources, but also when we are processing images which vary in difficultly. Easier images can be classified after a few processing levels, while for harder ones we can extend processing to more levels. Finally, in Figure 7 we provide examples of attended image locations.

6 Conclusion

We proposed a novel architecture that is able to process images of arbitrary resolution without sacrificing spatial information, as it typically happens with fully convolutional processing. This is achieved by approaching feature extraction as a top-down image pyramid traversal, that combines information from multiple different scales. The employed attention mechanism allows us to adjust the computational requirements of our models, by changing the number of locations they attend. This way we can exploit the existing trade-off between computational complexity and accuracy. Furthermore, by inspecting the image regions that our models attend, we are able to get important insights about the causes of their decisions. Finally, there are multiple future research directions that we would like to explore. These include the improvement of the localization capabilities of our attention mechanism, and the application of our model to the problem of budgeted batch classification. In addition, we would like our feature extraction process to become more adaptive, by allowing already extracted features to affect the processing of image regions that are attended later on.

REFERENCES

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

Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems*, pp. 4331–4339, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. 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, 2018.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

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

David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pp. 2366–2374, 2014.

SM Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Geoffrey E Hinton, et al. Attend, infer, repeat: Fast scene understanding with generative models. In *Advances in Neural Information Processing Systems*, pp. 3225–3233, 2016.

Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1039–1048, 2017.

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

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. In *International Conference on Machine Learning*, pp. 1462–1471, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European conference on computer vision*, pp. 346–361. Springer, 2014.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

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

Andrew G Howard. Some improvements on deep convolutional neural network based image classification. *arXiv preprint arXiv:1312.5402*, 2013.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*, 2017.

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

Angelos Katharopoulos and François Fleuret. Processing megapixel images with deep attention-sampling models. *arXiv preprint arXiv:1905.03711*, 2019.

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

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

Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Hila Levi and Shimon Ullman. Efficient coarse-to-fine non-local module for the detection of small objects. *arXiv preprint arXiv:1811.12152*, 2018.

Tsung-Yi Lin, Piotr Dollár, Ross B Girshick, Kaiming He, Bharath Hariharan, and Serge J Belongie. Feature pyramid networks for object detection. In *CVPR*, volume 1, pp. 4, 2017.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pp. 21–37. Springer, 2016.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pp. 2204–2212, 2014.

Mahyar Najibi, Bharat Singh, and Larry S Davis. Autofocus: Efficient multi-scale inference. *arXiv preprint arXiv:1812.01600*, 2018.

Pedro HO Pinheiro and Ronan Collobert. Recurrent convolutional neural networks for scene labeling. In *31st International Conference on Machine Learning (ICML)*, number EPFL-CONF-199822, 2014.

Jason Ramapuram, Maurits Diephuis, Russ Webb, and Alexandros Kalousis. Variational saccading: Efficient inference for large resolution images. *arXiv preprint arXiv:1812.03170*, 2018.

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

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pp. 525–542. Springer, 2016.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Huiyu Wang, Aniruddha Kembhavi, Ali Farhadi, Alan L Yuille, and Mohammad Rastegari. Elastic: Improving cnns with dynamic scaling policies. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2258–2267, 2019.

Min Wang, Baoyuan Liu, and Hassan Foroosh. Factorized convolutional neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 545–553, 2017.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8817–8826, 2018.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pp. 2048–2057, 2015.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9194–9203, 2018.

Amir R Zamir, Te-Lin Wu, Lin Sun, William B Shen, Bertram E Shi, Jitendra Malik, and Silvio Savarese. Feedback networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1308–1317, 2017.

Song-Chun Zhu, David Mumford, et al. A stochastic grammar of images. *Foundations and Trends® in Computer Graphics and Vision*, 2(4):259–362, 2007.

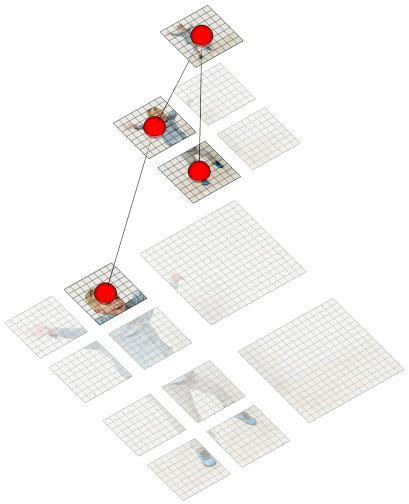

Figure 8: The parsing tree that is implicitly created by our model according to the example of Figure 1 (c). Nodes correspond to the attended image locations, and edges relate each location with its constituent parts.

A APPENDIX

A.1 IMPLICIT PARSING TREE

Following the example of Figure 1, in Figure 8 we provide the parsing tree that our model implicitly creates.

A.2 MODULES

Feature extraction module. It is a CNN that receives as input images of fixed resolution $h \times w \times c$, and outputs feature vectors of fixed size $1 \times f$. In the example of Fig. 2, $h = w = 32$.

Location module. It receives as input feature maps of fixed size $f_h \times f_w \times f_c$, and returns g probabilities, where g corresponds to the number of cells in the grid of candidate locations. In the example of Fig. 2, $g = 4$ since we are using a 2×2 grid.

Aggregation module. It receives g vectors of fixed size $1 \times f$, and outputs a vector of fixed size $1 \times f$. The g input vectors describe the image regions inside a $k \times k$ grid. Image regions that were not selected by the location module, are described by zero vectors. The g input vectors are reorganized into a $1 \times k \times k \times f$ tensor, according to the spatial arrangement of the image regions they describe. The tensor of the reorganized input vectors is used to produce the final output.

Merging module. It receives two vectors of fixed size $1 \times f$, concatenates them into a single vector of size $1 \times 2f$, and outputs a vector of fixed size $1 \times f$.

Classification module. It receives as input a vector of fixed size $1 \times f$, and outputs logits of fixed size $1 \times c$, where c is the number of classes. The logits are fed into a softmax layer to yield class probabilities.

A.3 EQUATION OF FEATURE EXTRACTION

The feature extraction process of our model can be described with the following recursive equation:

$$V_n^{'} = V_n \oplus agg(loc(\{V_m^{'} | m \in l_{V_n}\})) \tag{7}$$

where $\oplus$ denotes the *merging module*, $agg(\cdot)$ denotes the *aggregation module*, $loc(\cdot)$ denotes the outcome of the Bernoulli sampling that is based on the output of the *location module*, and l_{V_n} is the set of indexes that denote the candidate locations related to the image region described by V_n. When recursion ends, $V_m^{'} = V_m \; \forall m$.

A.4 TRAINING

A.4.1 LEARNING RULE DERIVATION

The ***REINFORCE*** rule naturally emerges if we optimize the log likelihood of the labels, while considering the attended locations as latent variables (Ba et al., 2014). For a batch of N images, the log likelihood is given by the following relation:

$$\sum_{i=1}^{N} \log p(y_i|x_i, w) = \sum_{i=1}^{N} \log \sum_{l^i} p(l^i|x_i, w) p(y_i|l^i, x_i, w), \tag{8}$$

$$p(l^i|x_i, w) = \prod_{j=1}^{N^{l^i}} \prod_{k=1}^{g} p(l_{j,k}^i|x_i, w)^{u_{j,k}^{l^i}} \cdot (1 - p(l_{j,k}^i|x_i, w))^{1-u_{j,k}^{l^i}} \tag{8a}$$

where x_i is the ith image in the batch, y_i is its label, and w are the parameters of our model. $p(l^i|x_i, w)$ is the probability that the sequence of locations l^i is attended for image x_i, and $p(y_i|l^i, x_i, w)$ is the probability of predicting the correct label after attending l^i. Equation 8 describes the log likelihood of the labels in terms of all location sequences that could be attended. Equation 8a shows that $p(l^i|x_i, w)$ is a product of Bernoulli distributions. N^{l^i} is the number of times the location module is applied while sequence l^i is attended, and g is the number of candidate locations considered by the location module. In the example of Figure 2, $N^{l^i} = 3$ and $g = 4$. $l_{j,k}^i$ is the kth out of the g candidate locations, that are considered the jth time the location module is applied while attending l^i. $u_{j,k}^{l^i} \in \{0, 1\}$ is equal to 1 when location $l_{j,k}^i$ is attended, and 0 when it is not. $p(l_{j,k}^i|x_i, w)$ is computed by the location module, $u_{j,k}^{l^i}$ is the outcome of Bernoulli sampling, and $p(y_i|l^i, x_i, w)$ is computed by the classification module.

We use Jensen's inequality in equation 8 to derive the following lower bound on the log likelihood:

$$\sum_{i=1}^{N} \log p(y_i|x_i, w) \geq \sum_{i=1}^{N} \sum_{l^i} p(l^i|x_i, w) \log p(y_i|l^i, x_i, w) = F \tag{9}$$

By maximizing the lower bound F, we expect to maximize the log likelihood. The update rule that we use is the partial derivative of F with respect to w, normalized by the number of images in the batch. We get:

$$\begin{aligned} \frac{1}{N}\frac{\partial F}{\partial w} =& \frac{1}{N} \sum_{i=1}^{N} \sum_{l^i} \Big[p(l^i|x_i, w) \frac{\partial \log p(y_i|l^i, x_i, w)}{\partial w} + \log p(y_i|l^i, x_i, w) \frac{\partial p(l^i|x_i, w)}{\partial w} \Big] \Rightarrow \\ \frac{1}{N}\frac{\partial F}{\partial w} =& \frac{1}{N} \sum_{i=1}^{N} \sum_{l^i} p(l^i|x_i, w) \Big[\frac{\partial \log p(y_i|l^i, x_i, w)}{\partial w} + \log p(y_i|l^i, x_i, w) \frac{\partial \log p(l^i|x_i, w)}{\partial w} \Big] \end{aligned} \tag{10}$$

To derive equation 10 we used the log derivative trick. As we can see, for each image x_i we need to calculate an expectation according to $p(l^i|x_i, w)$. We approximate each expectation with a Monte Carlo estimator of M samples:

$$\frac{1}{N}\frac{\partial F}{\partial w} \approx \frac{1}{N}\frac{\partial \tilde{F}}{\partial w} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{M} \sum_{m=1}^{M} \Big[\frac{\partial \log p(y_i|l^{i,m}, x_i, w)}{\partial w} + \log p(y_i|l^{i,m}, x_i, w) \frac{\partial \log p(l^{i,m}|x_i, w)}{\partial w} \Big] \tag{11}$$

We get samples from $p(l^i|x_i, w)$ by repeating the processing of image x_i. $l^{i,m}$ is the sequence of locations that is attended during the mth time we process image x_i. In order to reduce the variance of

the estimators, we use the baseline technique from Xu et al. (2015). In particular, the baseline we use is the exponential moving average of the log likelihood, and is updated after the processing of each batch during training. Our baseline after the nth batch is the following:

$$b_n = 0.9 \cdot b_{n-1} + 0.1 \cdot \frac{1}{NM} \sum_{i=1}^{NM} \log p(y_i^n | l^{i,n}, x_i^n, w) \tag{12}$$

where x_i^n is the ith image in the nth batch, y_i^n is its label, and $l^{i,n}$ is the corresponding attended sequence of locations. Since we use M samples for the Monte Carlo estimator of each image, we simply consider that our batch has size NM to simplify the notation. Our updated learning rule is the following:

$$L_F = \frac{1}{NM} \sum_{i=1}^{NM} \Big[\frac{\partial \log p(y_i | l^i, x_i, w)}{\partial w} + \lambda_f (\log p(y_i | l^i, x_i, w) - b) \frac{\partial \log p(l^i | x_i, w)}{\partial w} \Big] \tag{13}$$

For simplicity, we drop the indexes that indicate the batch we are processing. (13) is the learning rule we presented in (1), and this concludes our derivation.

A.4.2 Regularization Terms

We provide the exact equations that we use to calculate quantities $p_t^{l^i}$ and p_k in regularization terms R_t (2) and R_r (3) respectively. We have:

$$p_t^{l^i} = \sum_{j=1}^{N^{l^i}} \sum_{k=1}^{g} p(l_{j,k}^i | x_i, w) \tag{14}$$

$$p_k = \frac{1}{\sum_{i=1}^{MN} N^{l^i}} \sum_{i=1}^{MN} \sum_{j}^{N^{l^i}} p(l_{j,k}^i | x_i, w) \tag{15}$$

Based on the notation we introduced in Appendix A.4.1, $p_t^{l^i}$ approximates the expected number of attended locations for image x_i, by summing the probabilities from all Bernoulli distributions considered during a single processing of x_i under l^i. p_k computes the average probability of attending the kth out of the g candidate locations, that the location module considers every time it is applied. The average is calculated by considering all the times the location module is applied during the processing of the NM images in our augmented batch. Finally, we would like to note that the values of c_t and c_r are interdependent.

A.5 Experiments With MNIST

A.5.1 Architectures

In Tables 1, 2 and 3, we provide the exact architectures of models (M_3, BL_3), (M_2, BL_2) and (M_1, BL_1).

A.5.2 Training Details

We provide details about training one pair of models (M_i, BL_i) with one of the 3 MNIST-based datasets, and the exact same procedure applies to all pairs and to all datasets. In our first training session we optimize the cross entropy loss to train BL_i, and we use learning rule (4) for M_i. We remind that because of the input resolution, M_i goes through only 1 processing level, and behaves as a regular CNN which is consisted of the feature extraction module followed by the classification module. As a result, the only term of learning rule (4) that we actually use, is the first term of L_F, and we end up optimizing the cross entropy of the labels. For both models we use the following hyperparameters. We use learning rate 0.01 that drops by a factor of 0.2 after the completion of 80% and 90% of the total number of training steps. We train for 70 epochs, and we use batches of size 128. We use the Adam optimizer (Kingma & Ba, 2014) with the default values of $\beta_1 = 0.9$,

	Layer Type	Weights' Size (w/o biases)	Stride	Padding	Activation	Output Size	Receptive Field Size	Params	FLOPs
Feature Extraction Module	Input					$14 \times 14 \times 1$			
	Conv	$3 \times 3 \times 1 \times 32$	1×1	SAME	Leaky ReLU	$14 \times 14 \times 32$	3×3	320	$56,488$
	Max Pool	$2 \times 2 \times 1$	2×2	VALID		$7 \times 7 \times 32$	4×4		
	Conv	$3 \times 3 \times 32 \times 64$	1×1	SAME	Leaky ReLU	$7 \times 7 \times 64$	8×8	$18,496$	$903,168$
	Max Pool	$3 \times 3 \times 1$	2×2	VALID		$3 \times 3 \times 64$	12×12		
	Conv	$3 \times 3 \times 64 \times 64$	1×1	SAME	Leaky ReLU	$3 \times 3 \times 64$	20×20	$36,928$	$331,776$
	GAP					$1 \times 1 \times 64$			
	Output					$3 \times 3 \times 64$, 1×64			
Location Module	Input					$3 \times 3 \times 64$			
	Conv	$3 \times 3 \times 64 \times 9$	1×1	VALID	Logistic	$1 \times 1 \times 9$		5193	5184
	Output					1×9			
Aggregation Module	Input					$3 \times 3 \times 64$			
	Conv	$3 \times 3 \times 64 \times 64$	1×1	VALID	Leaky ReLU	$1 \times 1 \times 64$		$36,928$	$36,864$
	Output					1×64			
Merging Module	Input					1×128			
	FC	128×64			Leaky ReLU	1×64		$8,256$	$8,192$
	Output					1×64			
Classification Module	Input					1×64			
	Linear	64×10				1×10		650	640
	Output					1×10			
Baseline CNN	Input					$N \times N \times 1$			
	Conv	$3 \times 3 \times 1 \times 32$	1×1	SAME	Leaky ReLU	$N \times N \times 32$	3×3	320	$288N^2$
	Max Pool	$2 \times 2 \times 1$	2×2	VALID		$N/2 \times N/2 \times 32$	4×4		
	Conv	$3 \times 3 \times 32 \times 64$	1×1	SAME	Leaky ReLU	$N/2 \times N/2 \times 64$	8×8	$18,496$	$4,608N^2$
	Max Pool	$3 \times 3 \times 1$	2×2	VALID		$N/4 \times N/4 \times 32$	12×12		
	Conv	$3 \times 3 \times 64 \times 64$	1×1	SAME	Leaky ReLU	$N/4 \times N/4 \times 64$	20×20	$36,928$	$2,304N^2$
	GAP					$1 \times 1 \times 64$			
	FC	64×64			Leaky ReLU	1×64		$4,160$	$4,096$
	FC	64×64			Leaky ReLU	1×64		$4,160$	$4,096$
	Linear	64×10				1×10		650	640
	Output					1×10			

Table 1: The architectures of models M_3 and BL_3 that we used in our experiments with MNIST. "GAP" denotes a global average pooling layer. The variable output sizes of the baseline CNN are approximately calculated to preserve the clarity of our tables. Based on these approximate output sizes, the computation of the number of FLOPs in the corresponding layers is approximate as well.

$\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. We use xavier initialization (Glorot & Bengio, 2010) for the weights, and zero initialization for the biases. For regularization purposes, we use the following form of data augmentation. Every time we load an image, we randomly change the position of the digit inside the black canvas, as well as the noise pattern and the background, in case we are using *noisy MNIST* or *textured MNIST*. This data augmentation strategy and the aforementioned hyperparameter's values, are used in the other two training sessions as well.

In the second training session we again optimize the cross entropy loss for BL_i, and we use learning rule (4) for M_i. The resolution of the input allows M_i to go through 2 processing levels, and all terms of (4) contribute to the updates of our models' parameters. We would like to note that we stop the gradients' flow from the location module to the other modules, and as a result, the second term of (1), as well as the regularization terms (2) and (3), affect only the parameters of the location module. We do this to have better control on how the location module learns, since we experienced the problems we described in Section 4. In the same context, we set $\lambda_f = 10^{-6}$, $\lambda_r = 10^{-7}$ and $\lambda_t = 10^{-7}$, which lead to very small updates at every training step. For our Monte Carlo estimators we use 2 samples. These hyperparameter values are used in the third training session as well. The models M_i which are reported in Figure 4 and result from this training session (green circles), are trained with $c_t = 2$.

In the third training session M_i can go through 3 processing levels, and we can train it by using either learning rule (4), or gradual learning (5). Gradual learning evolves in 2 stages, where in the first stage M_i can go through 2 processing levels, while in the second stage it can go through 3. The first training stage is equivalent to the previous training session, since there is an equivalence between going through a different number of processing levels and processing inputs of different resolution. Based on that, we can directly move to the second training stage of gradual learning, by initializing the variables of M_i with the learned parameters from one of the M_i^{28} models that we already trained. Gradual learning creates an imbalance in terms of how M_i and BL_i are trained, since it evolves in multiple stages. However, if we see gradual learning as a method of training models with images of gradually increasing resolution, it can be applied to the baselines as well. Based on this, we can

	Layer Type	Weights' Size (w/o biases)	Stride	Padding	Activation	Output Size	Receptive Field Size	Params	FLOPs
Feature Extraction Module	Input					$14 \times 14 \times 1$			
	Conv	$3 \times 3 \times 1 \times 32$	1×1	SAME	Leaky ReLU	$14 \times 14 \times 32$	3×3	320	$56,488$
	Max Pool	$2 \times 2 \times 1$	2×2	VALID		$7 \times 7 \times 32$	4×4		
	Conv	$3 \times 3 \times 32 \times 64$	1×1	SAME	Leaky ReLU	$7 \times 7 \times 64$	8×8	$18,496$	$903,168$
	GAP					$1 \times 1 \times 64$			
	Output					$7 \times 7 \times 64$, 1×64			
Location Module	Input					$7 \times 7 \times 64$			
	Conv	$7 \times 7 \times 64 \times 9$	1×1	VALID	Logistic	$1 \times 1 \times 9$		$28,233$	$28,224$
	Output					1×9			
Aggregation Module	Input					$3 \times 3 \times 64$			
	Conv	$3 \times 3 \times 64 \times 64$	1×1	VALID	Leaky ReLU	$1 \times 1 \times 64$		$36,928$	$36,864$
	Output					1×64			
Merging Module	Input					1×128			
	FC	128×64			Leaky ReLU	1×64		$8,256$	$8,192$
	Output					1×64			
Classification Module	Input					1×64			
	Linear	64×10				1×10		650	640
	Output					1×10			
Baseline CNN	Input					$N \times N \times 1$			
	Conv	$3 \times 3 \times 1 \times 32$	1×1	SAME	Leaky ReLU	$N \times N \times 32$	3×3	320	$288N^2$
	Max Pool	$2 \times 2 \times 1$	2×2	VALID		$N/2 \times N/2 \times 32$	4×4		
	Conv	$3 \times 3 \times 32 \times 64$	1×1	SAME	Leaky ReLU	$N/2 \times N/2 \times 64$	8×8	$18,496$	$4,608N^2$
	GAP					$1 \times 1 \times 64$			
	FC	64×64			Leaky ReLU	1×64		$4,160$	$4,096$
	FC	64×64			Leaky ReLU	1×64		$4,160$	$4,096$
	Linear	64×10				1×10		650	640
	Output					1×10			

Table 2: The architectures of models M_2 and BL_2 that we used in our experiments with MNIST. "GAP" denotes a global average pooling layer. The variable output sizes of the baseline CNN are approximately calculated to preserve the clarity of our tables. Based on these approximate output sizes, the computation of the number of FLOPs in the corresponding layers is approximate as well.

	Layer Type	Weights' Size (w/o biases)	Stride	Padding	Activation	Output Size	Receptive Field Size	Params	FLOPs
Feature Extraction Module	Input					$14 \times 14 \times 1$			
	Conv	$3 \times 3 \times 1 \times 64$	1×1	SAME	Leaky ReLU	$14 \times 14 \times 64$	3×3	640	112896
	Max Pool	$2 \times 2 \times 1$	2×2	VALID		$7 \times 7 \times 64$	4×4		
	GAP					$1 \times 1 \times 64$			
	Output					$7 \times 7 \times 64$, 1×64			
Location Module	Input					$7 \times 7 \times 64$			
	Conv	$7 \times 7 \times 64 \times 9$	1×1	VALID	Logistic	$1 \times 1 \times 9$		$28,233$	$28,224$
	Output					1×9			
Aggregation Module	Input					$3 \times 3 \times 64$			
	Conv	$3 \times 3 \times 64 \times 64$	1×1	VALID	Leaky ReLU	$1 \times 1 \times 64$		$36,928$	$36,864$
	Output					1×64			
Merging Module	Input					1×128			
	FC	128×64			Leaky ReLU	1×64		$8,256$	$8,192$
	Output					1×64			
Classification Module	Input					1×64			
	Linear	64×10				1×10		650	640
	Output					1×10			
Baseline CNN	Input					$N \times N \times 1$			
	Conv	$3 \times 3 \times 1 \times 64$	1×1	SAME	Leaky ReLU	$N \times N \times 64$	3×3	640	$576N^2$
	Max Pool	$2 \times 2 \times 1$	2×2	VALID		$N/2 \times N/2 \times 64$	4×4		
	GAP					$1 \times 1 \times 64$			
	FC	64×64			Leaky ReLU	1×64		$4,160$	$4,096$
	FC	64×64			Leaky ReLU	1×64		$4,160$	$4,096$
	Linear	64×10				1×10		650	640
	Output					1×10			

Table 3: The architectures of models M_1 and BL_1 that we used in our experiments with MNIST. "GAP" denotes a global average pooling layer. The variable output sizes of the baseline CNN are approximately calculated to preserve the clarity of our tables. Based on these approximate output sizes, the computation of the number of FLOPs in the corresponding layers is approximate as well.

initialize the variables of BL_i with the learned parameters of BL_i^{28}, and then apply our standard optimization to the cross entropy loss.

In practice, we observe that baselines are almost always benefited by gradual learning, while in some cases our models achieve higher accuracy when they are trained from scratch with learning rule (4). In general, gradual learning and most modifications to the initial learning rule (11), resulted from our experimentation with ImageNet, which is a much bigger and more diverse dataset of natural images. Consequently, our results on the MNIST-based datasets remain almost unchanged even if we simplify our training procedure, e.g. by excluding term (3) from our training rule. However, we kept our training procedure consistent both on the MNIST-based datasets and on ImageNet, we experimented both with gradual learning and with training from scratch at every session, and in the results of Figure 4 we report the best performing models. Finally, the models M_i which are reported in Figure 4 and result from this training session (red circles), are trained with $c_t = 6$ and $c_t = 12$.

A.5.3 Additional Remarks On The Results

In Figure 4 (e), BL_2^{56} achieves higher accuracy compared to BL_2^{28}, which wasn't the case in (b). We hypothesize that the fine information provided by the increased resolution, is valuable to disentangle the digits from the distracting background, and outweighs the lack of coarse level features that stems from the receptive field size of BL_2^{56}. In addition, the differences in accuracy between M_1 and BL_1 models in Fig. 4 (f), are considerably bigger compared to the ones recorded in (c). This shows that our models are more robust to distracting textured background compared to the baselines.

In Fig. 4 (g), the accuracy of $M_3^{56,6.1}$ is lower compared to $M_3^{28,2.2}$. This is surprising, because $M_3^{56,6.1}$ is processing images of higher resolution, and it should be able to reach at least the same level of accuracy as $M_3^{28,2.2}$. Our explanation for this observation is based on the nature of the data. As we can see in the example images that we provide in the last row of Figure 4, when the resolution of an image is reduced, the noise is blurred out. As a result, when $M_3^{56,6.1}$ is processing images of higher resolution, it is processing more intense high frequency noise, and since it is using a limited number of locations, its accuracy drops compared to $M_3^{28,2.2}$. However, BL_3^{56} improves on BL_3^{28} in terms of accuracy, and as a result, we expect that a M_3^{56} model with more attended locations should be able to demonstrate accuracy at least at the level of $M_3^{28,2.2}$. This is what is happening with model $M_3^{56,12.1}$.

This phenomenon is observed in Fig. 4 (h) and (j) as well, since $M_2^{56,6.1}$ and $M_1^{56,6.3}$ achieve lower accuracy compared to $M_2^{28,2.0}$ and $M_1^{28,2.2}$ respectively. The explanation we provided adds a new dimension to the understanding of our models, because so far we were treating fine information as something that is by default beneficial for classification, while high frequency noise of any form may require special consideration in our design and training choices.

A.6 Experiments On ImageNet

A.6.1 Architectures

In Tables 4 and 5, we provide the exact architectures of models (M_2, BL_2) and (M_1, BL_1).

A.6.2 Training Details

We provide details about training one pair of models (M_i, BL_i). We make a distinction between models M_1 and M_2 only when the process we follow, or the values of the hyperparameters that we use, differ. In our first training session we optimize the cross entropy loss to train BL_i, and we use learning rule (4) for M_i. The base resolution of M_i matches the size of the input images ($32 \times 32\ px$), and our model goes through only 1 processing level, without using the location module. As a result, the only term of learning rule (4) that we actually use, is the first term of L_F, and we end up optimizing the cross entropy of the labels.

For both models we use the following hyperparameters. We use learning rate 0.001 that drops by a factor of 0.2 after the completion of 80% and 90% of the total number of training steps. We train for 200 epochs, and we use batches of size 128. We use the Adam optimizer with the default values of $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. We use xavier initialization for the weights, and zero initialization for the biases. For regularization purposes, we use data augmentation that is very similar to the one used by Szegedy et al. (2015). In particular, given a training image, we get a random

	Layer Type	Weights' Size (w/o biases)	Stride	Padding	Activation	Output Size	Receptive Field Size	Params	FLOPs
Feature Extraction Module	Input					$32 \times 32 \times 3$			
	Conv	$3 \times 3 \times 3 \times 64$	1×1	SAME	Leaky ReLU	$32 \times 32 \times 64$	3×3	1792	$1,769,472$
	Max Pool	$2 \times 2 \times 1$	2×2	VALID		$16 \times 16 \times 64$	4×4		
	Conv	$3 \times 3 \times 64 \times 128$	1×1	SAME	Leaky ReLU	$16 \times 16 \times 128$	8×8	$73,856$	$18,874,368$
	Max Pool	$2 \times 2 \times 1$	2×2	VALID		$8 \times 8 \times 128$	10×10		
	Conv	$3 \times 3 \times 128 \times 256$	1×1	SAME	Leaky ReLU	$8 \times 8 \times 256$	18×18	$295,168$	$18,874,368$
	Max Pool	$2 \times 2 \times 1$	2×2	VALID		$4 \times 4 \times 256$	22×22		
	Conv	$3 \times 3 \times 256 \times 256$	1×1	SAME	Leaky ReLU	$4 \times 4 \times 256$	38×38	$590,080$	$9,437,184$
	GAP					$1 \times 1 \times 256$			
	Output					$4 \times 4 \times 256$, 1×256			
Location Module	Input					$4 \times 4 \times 256$			
	Conv	$4 \times 4 \times 256 \times 9$	1×1	VALID	Logistic	$1 \times 1 \times 9$		$36,873$	$36,864$
	Output					1×9			
Aggregation Module	Input					$3 \times 3 \times 256$			
	Conv	$3 \times 3 \times 256 \times 256$	1×1	VALID	Leaky ReLU	$1 \times 1 \times 256$		$590,080$	$589,824$
	Output					1×256			
Merging Module	Input					1×512			
	FC	521×256			Leaky ReLU	1×256		$131,328$	$131,072$
	Output					1×256			
Classification Module	Input					1×256			
	Linear	$256 \times 1,000$				1×10		$256,000$	$256,000$
	Output					1×10			
Baseline CNN	Input					$N \times N \times 1$			
	Conv	$3 \times 3 \times 3 \times 64$	1×1	SAME	Leaky ReLU	$N \times N \times 64$	3×3	1792	$1,728N^2$
	Max Pool	$2 \times 2 \times 1$	2×2	VALID		$N/2 \times N/2 \times 64$	4×4		
	Conv	$3 \times 3 \times 64 \times 128$	1×1	SAME	Leaky ReLU	$N/2 \times N/2 \times 128$	8×8	$73,856$	$18,432N^2$
	Max Pool	$2 \times 2 \times 1$	2×2	VALID		$N/4 \times N/4 \times 128$	10×10		
	Conv	$3 \times 3 \times 128 \times 256$	1×1	SAME	Leaky ReLU	$N/4 \times N/4 \times 256$	18×18	$295,168$	$18,432N^2$
	Max Pool	$2 \times 2 \times 1$	2×2	VALID		$N/8 \times N/8 \times 256$	22×22		
	Conv	$3 \times 3 \times 256 \times 256$	1×1	SAME	Leaky ReLU	$N/8 \times N/8 \times 256$	38×38	$590,080$	$9216N^2$
	GAP					$1 \times 1 \times 256$			
	FC	256×256			Leaky ReLU	1×256		$65,792$	$65,536$
	FC	256×256			Leaky ReLU	1×256		$65,792$	$65,536$
	Linear	$256 \times 1,000$				$1 \times 1,000$		$256,000$	$256,000$
	Output					$1 \times 1,000$			

Table 4: The architectures of models M_2 and BL_2 that we used in our experiments on ImageNet. "GAP" denotes a global average pooling layer. The variable output sizes of the baseline CNN are approximately calculated to preserve the clarity of our tables. Based on these approximate output sizes, the computation of the number of FLOPs in the corresponding layers is approximate as well.

crop that covers at least 85% of the image area, while it has an aspect ration between 0.5 and 2.0. Since we provide inputs of fixed size to our networks, we resize the image crops appropriately. The resizing is performed by randomly selecting between bilinear, nearest neighbor, bicubic, and area interpolation. Also, we randomly flip the resized image crops horizontally, and we apply photometric distortions according to Howard (2013). The final image values are scaled between -1 and 1. This data augmentation strategy and the aforementioned hyperparameter's values, are used in the other training sessions as well, and in the stages of multi-level learning that we describe later.

In the second training session we again optimize the cross entropy loss for BL_i, and we use learning rule (4) for M_i. The resolution of the input allows M_i to go through 2 processing levels, and all terms of (4) contribute to the updates of our models' parameters. As we described in Appendix A.5.2, we stop the gradients' flow from the location module to the other modules. We set $\lambda_f = 10^{-8}$, $\lambda_r = 10^{-9}$ and $\lambda_t = 10^{-9}$, and for our Monte Carlo estimators we use 2 samples. These hyperparameter values are used in the two remaining training sessions as well. The models M_i^{64} which are reported in Figure 6 are trained with $c_t \in \{1.5, 4.5\}$.

In the third training session we use gradual learning (5) for M_i, by initializing its variables with the learned parameters of a M_i^{64} model. The models M_2^{128} which are reported in Figure 6 are trained with $c_t \in \{3.75, 7, 13.5, 24.75\}$, and models M_1^{128} with $c_t \in \{3.75, 8, 11, 13.5, 30\}$. We use gradual learning for BL_i as well, by initializing its parameters with those of BL_i^{64} before we apply the standard optimization to the cross entropy loss.

In the fourth training session we again use gradual learning for M_i, by initializing its variables with the learned parameters of a M_i^{128} model. The models M_2^{256} which are reported in Figure 6 are trained with $c_t \in \{22, 29, 39\}$, and models M_1^{256} with $c_t \in \{30, 35, 40\}$. As in the previous session, we optimize the cross entropy loss to train BL_i, and we initialize its parameters with those of BL_i^{128}.

	Layer Type	Weights' Size (w/o biases)	Stride	Padding	Activation	Output Size	Receptive Field Size	Params	FLOPs
Feature Extraction Module	Input					$32 \times 32 \times 3$			
	Conv	$3 \times 3 \times 3 \times 64$	1×1	SAME	Leaky ReLU	$32 \times 32 \times 64$	3×3	1792	$1,769,472$
	Max Pool	$2 \times 2 \times 1$	2×2	VALID		$16 \times 16 \times 64$	4×4		
	Conv	$3 \times 3 \times 64 \times 128$	1×1	SAME	Leaky ReLU	$16 \times 16 \times 128$	8×8	$73,856$	$18,874,368$
	Max Pool	$2 \times 2 \times 1$	2×2	VALID		$8 \times 8 \times 128$	10×10		
	Conv	$3 \times 3 \times 128 \times 256$	1×1	SAME	Leaky ReLU	$8 \times 8 \times 256$	18×18	$295,168$	$18,874,368$
	GAP					$1 \times 1 \times 256$			
	Output					$8 \times 8 \times 256$, 1×256			
Location Module	Input					$8 \times 8 \times 256$			
	Conv	$8 \times 8 \times 256 \times 9$	1×1	VALID	Logistic	$1 \times 1 \times 9$		$147,465$	$147,456$
	Output					1×9			
Aggregation Module	Input					$3 \times 3 \times 256$			
	Conv	$3 \times 3 \times 256 \times 256$	1×1	VALID	Leaky ReLU	$1 \times 1 \times 256$		$590,080$	$589,824$
	Output					1×256			
Merging Module	Input					1×512			
	FC	521×256			Leaky ReLU	1×256		$131,328$	$131,072$
	Output					1×256			
Classification Module	Input					1×256			
	Linear	$256 \times 1,000$				1×10		$256,000$	$256,000$
	Output					1×10			
Baseline CNN	Input					$N \times N \times 1$			
	Conv	$3 \times 3 \times 3 \times 64$	1×1	SAME	Leaky ReLU	$N \times N \times 64$	3×3	1792	$1,728N^2$
	Max Pool	$2 \times 2 \times 1$	2×2	VALID		$N/2 \times N/2 \times 64$	4×4		
	Conv	$3 \times 3 \times 64 \times 128$	1×1	SAME	Leaky ReLU	$N/2 \times N/2 \times 128$	8×8	$73,856$	$18,432N^2$
	Max Pool	$2 \times 2 \times 1$	2×2	VALID		$N/4 \times N/4 \times 128$	10×10		
	Conv	$3 \times 3 \times 128 \times 256$	1×1	SAME	Leaky ReLU	$N/4 \times N/4 \times 256$	18×18	$295,168$	$18,432N^2$
	GAP					$1 \times 1 \times 256$			
	FC	256×256			Leaky ReLU	1×256		$65,792$	$65,536$
	FC	256×256			Leaky ReLU	1×256		$65,792$	$65,536$
	Linear	$256 \times 1,000$				$1 \times 1,000$		$256,000$	$256,000$
	Output					$1 \times 1,000$			

Table 5: The architectures of models M_1 and BL_1 that we used in our experiments on ImageNet. "GAP" denotes a global average pooling layer. The variable output sizes of the baseline CNN are approximately calculated to preserve the clarity of our tables. Based on these approximate output sizes, the computation of the number of FLOPs in the corresponding layers is approximate as well.

Finally, we train $\{M_i\}_{i=1}^2$ with multi-level learning rule (6) in 3 stages of gradual learning. In the first training stage ($s = 1$) we use images of resolution 64×64 and we set $\lambda_0^1 = \lambda_1^1 = 1$. This means that we assign the same weight on correctly classifying images after the first and the second processing level. The hyperparameters for learning rules L_r^0 and L_r^1 are set as we described before (the same holds for the subsequent training stages), while for M_2 we use $c_t^1 = 3.2$ and for M_1 we use $c_t^1 = 2.9$. In the second training stage ($s = 2$) we use images of resolution 128×128, and we set $\lambda_0^2 = \lambda_1^2 = \lambda_2^2 = 1$. For M_2 we set $c_t^1 = 3.2$ and $c_t^2 = 13.5$, while for M_1 we set $c_t^1 = 2.9$ and $c_t^2 = 11.3$. In the third training stage ($s = 3$) we use images of resolution 256×256, and we set $\lambda_0^3 = \lambda_1^3 = \lambda_2^3 = \lambda_3^3 = 1$. For M_2 we set $c_t^1 = 3.2$, $c_t^2 = 13.5$ and $c_t^3 = 40$, while for M_1 we set $c_t^1 = 2.9$, $c_t^2 = 11.3$ and $c_t^3 = 35.6$. During these 3 stages of training, when we were moving to a new training stage, we were initializing the variables of our models with the learned parameters from the previous training stage.