Electronic Preprint for Journal of Information Processing Vol.23 No.6

Regular Paper

A Multi-Label Convolutional Neural Network for Automatic Image Annotation

Alexis Vallet[1,a)] Hiroyasu Sakamoto[2,b)]

Received: January 8, 2015, Accepted: July 1, 2015

Abstract: Over the past few years, convolutional neural networks (CNN) have set the state of the art in a wide variety of supervised computer vision problems. Most research effort has focused on single-label classification, due to the availability of the large scale ImageNet dataset. Via pre-training on this dataset, CNNs have also shown the ability to outperform traditional methods for multi-label classification. Such methods, however, typically require evaluating many expensive forward passes to produce a multi-label distribution. Furthermore, due to the lack of a large scale multi-label dataset, little effort has been invested into training CNNs from scratch with multi-label data. In this paper, we address both issues by introducing a multi-label cost function adequate for deep CNNs, and a prediction method requiring only a single forward pass to produce multi-label predictions. We show the performance of our method on a newly introduced large scale multi-label dataset of animation images. Here, our method reaches 75.1% precision and 66.5% accuracy, making it suitable for automated annotation in practice. Additionally, we apply our method to the Pascal VOC 2007 dataset of natural images, and show that our prediction method outperforms a comparable model for a fraction of the computational cost.

Keywords: convolutional neural networks, multi-label classification, animation images

1. Introduction

First introduced by Fukushima [3] and refined by LeCun et al. [11], CNNs have made breakthroughs in the recent past for single label natural image classification [10], [16] thanks to the availability of larger annotated datasets, stronger regularization techniques such as dropout [8], rectified linear units (ReLU) and by leveraging the parallel computing power of GPUs and computer clusters [2].

More recently, CNNs have been successfully applied to multi-label image classification by Wei et al. [15] via pre-training on the ImageNet dataset and using pooling over many expensive object hypotheses. Building upon this work, we introduce a method to train a CNN from scratch with a large scale multi-label dataset, and a computationally cheap method to produce high quality multi-label predictions.

In Section 2, we detail the specifics of our method. We present a multi-label cost function which allows us to output a confidence value for each label. Further, we introduce an efficient prediction method which propagates confidence values across multiple scale and uses spatial information to prune unnecessary labels.

In Section 3, we introduce our large scale, multi-label dataset of animation images. We describe our data collection and labelling methodology, which allowed us to obtain "high enough" quality ground truth with very few resources by relying on community-checked tags.

In Section 4, we measure the performance of our method on both this dataset of animation images and the Pascal natural images dataset.

2. A Multi-label Convolutional Neural Network for Efficient and Accurate Predictions

In this section, we will describe the details of our method. Our algorithm is based on mini-batch stochastic gradient descent, an iterative method which works not with the entire training set but rather with small batches of images randomly picked from the training set. At iteration $t \in \mathbb{N}^*$ we consider a training batch $D_t = \{(I_1, Y_1), ..., (I_{n_t}, Y_{n_t})\}$ where $n_t \in \mathbb{N}^*$ is the number of samples in batch D_t, and for all $i \in \{1, ..., n_t\}$, I_i is a color image depicting one or multiple characters, while $Y_i \subseteq L$ is a set of labels representing the names of the characters depicted in I_i (as strings for instance). L is the finite set of all possible labels, which we conflate with $\{1, ..., l\}$ where $l = |L|$ for notational convenience with matrix notation. When we are talking about a single batch and there is no possible confusion, we will denote $n = n_t$ the number of samples in the batch. Also for convenience in notation, we will denote images, feature maps and convolution filters in channel, row and column notation - i.e., an image with d channels, r columns and c rows is a 3 dimensional array in $\mathbb{R}^{d \times r \times c}$.

2.1 Preprocessing

We perform a minimal amount of preprocessing and some label-preserving transformations on the images prior to training or prediction, mostly following the method used in Krizhevsky et

[1] Graduate School of Design, Kyushu University, Fukuoka 815–8540, Japan
[2] Faculty of Design, Kyushu University, Fukuoka 815–8540, Japan
[a)] alexis.vallet@gmail.com
[b)] sakamoto@design.kyushu-u.ac.jp

al. [10].

2.1.1 Resizing and Pixel Value Scaling

All images are resized to 256 pixels minimum dimension, using OpenCV's pixel area resizing method. Further, pixel values are converted from the integer range {0, ..., 255} to the [0, 1] single precision floating point range by dividing them by 255, to avoid possible feature scaling issues when training our network.

2.1.2 Mean Subtraction

To obtain input features with (roughly) 0 mean, we compute the mean pixel value across all images in the training set. We then subtract this value from each image. At test time, we subtract the same mean computed from the training data to all test images. Note that, unlike the method in Ref. [10], we do not crop images to the same size at this point, which allows images to have varying aspect ratios.

2.1.3 Random Crops

As our algorithm requires images of fixed dimension, each time an image is picked during training we choose a random crop of the image of size 256 by 256 pixels. This makes the size of inputs fixed at training time, allowing much more efficient processing of mini-batches on the GPU. It also helps against over-fitting.

2.1.4 Random Flips

Each time an image crop is picked during training, we flip it horizontally with probability 0.5, once again in order to combat over-fitting.

2.2 Network Architecture

Our CNN architecture is composed of the following types of layers:

- Convolutional layers, which given an input feature map (an image in the case of the first layer) $I \in \mathbb{R}^{d \times r \times c}$ applies a collection of k convolution filters $F_i \in \mathbb{R}^{d \times r' \times c'}$ for $i \in \{1, ..., k\}$. The result of the convolution of each filter with the feature maps plus a bias vector $b \in \mathbb{R}^k$ with ReLU non-linearity $x \mapsto \max(0, x)$ is stored into feature maps $I' \in \mathbb{R}^{k \times (r-r'+1) \times (c-c'+1)}$, such that for $i \in \{1, ..., k\}$:

  $$I'_i = \max(0, I * F_i + b\mathbf{1}_k^T).$$

  Where $\mathbf{1}_k \in \mathbb{R}^k$ denotes the column vector of all ones with k coefficients. Strided convolution means the convolution kernels are applied with a stride on the input feature maps, and not at every possible position.
- Max-pooling layers, given input feature maps $I \in \mathbb{R}^{d \times r \times c}$ and a pooling window of size $r' \times c'$, only pick the maximum value for each individual feature map under this window. This window is applied over each input feature map, possibly with a stride.
- Fully-connected layers, which are just linear layers followed by an element-wise ReLU non-linearity. The number of units corresponds to the number of desired outputs.
- Dropout layers, which following a fully-connected layer randomly sets to 0 some outputs with a given probability for each time a sample is presented to the network during training.

Our network, like some recent deep CNNs [12], [14], uses global average pooling between convolutional layers and fully-connected layers at training time. This forces the network to produce confidence maps for each output class. The key insight here is that adding fully connected layers on top of this global average pooling layer does not change that fact; indeed, one simply need to remove the average pooling layer entirely, and apply the FC layers "pixel-wise" on each output feature maps to get the final confidence maps. This "pixel-wise" operation is precisely a 1×1 convolution which shares the weights of the corresponding fully connected layer.

Table 1 Full architecture of our CNN, used on our dataset of animation images. We numbered only the weight layers. The layers on the same line of the test and training architectures share the same weights.

Layer	Architecture
1	Convolutional, 64 7×7 filters, stride 2, padding 3 Max pooling, 3×3 window and stride 2
2	Convolutional, 128 3×3 filters, stride 2 and padding 1
3 to 6	Convolutional, 128 3×3 filters and padding 1 Max pooling, 3×3 window and stride 2
7 to 11	Convolutional, 256 3×3 filters and padding 1 Max pooling, 3×3 window and stride 2
12 to 16	Convolutional, 512 3×3 filters and padding 1

Layer	Training architecture	Test architecture
	Global average pooling	
17	Fully connected, 4096 units Dropout, drop probability 0.5	Convolutional, 4096 1×1 filters Dropout, drop probability 0.5
18	Fully connected, 4096 units Dropout, drop probability 0.5	Convolutional, 4096 1×1 filters Dropout, drop probability 0.5
19	Fully connected, 115 units	Convolutional, 115 1×1 filters

This equivalence between fully connected layers and 1×1 convolutions in the global average pooling setting allows us to use 2 different architectures for training and testing. At training time, the cost function only requires a single scalar confidence for each label and each sample. It would therefore be computationally wasteful to compute 1×1 convolutions over entire feature maps. We simply use global average pooling to discard spatial information entirely, then apply the fully connected layers to this single output value. At test time, our prediction method uses spatial information to prune labels (see Section 2.6), and so we convert the weights previously trained in the fully connected layers for use in 1×1 convolutions.

For full details about our CNN architecture, see **Table 1**. For notational convenience, as it would be unwieldy to manipulate all the parameters of the network explicitly, we denote $W \in \mathbb{R}^m$ the vectorized, concatenated parameters of all weight layers in our network. In our case, it is the vectorized concatenation of all filters and biases from convolutional layers, as well as weights and biases from linear and fully-connected layers. We denote $C(W) \in \mathbb{R}^{n \times l}$, the output of our network with weights W on a given training batch, which represent confidence values for each sample in the batch and each class.

2.3 A Multi-label Error Function

The softmax used in most deep CNNs is designed to output probabilities, which is desirable for single-label classification. One can then use the multinomial logistic regression (MLR) cost function very straightforwardly to train the network.

In the case of multi-label classification though, probability outputs are undesirable - rather, we would like to output confidence

scores for each possible label. Multi-label cost functions have been proposed for neural networks in the past, such as Back Propagation for Multi-Label Learning (BP-MLL) by Zhang et al. [17], which we found difficult to use. Indeed, switching from the MLR cost function to BP-MLL changes drastically the range of values the cost and its gradient can take. This in turns requires finding new hyper-parameters both for the network architecture and the optimization procedure. Finding new hyper-parameters is usually done through (cross) validation, which is somewhat impractical with deep CNNs which usually takes weeks to train. Therefore we attempted to stay as close as possible to the usual MLR cost function. We used the following error function E_i for a given sample i:

$$E_i(W) = -\log\left(\frac{\sum_{j\in Y_i}\exp(C(W)_{i,j})}{\sum_{j\in L}\exp(C(W)_{i,j})}\right)$$

which generalizes the MLR error function to multiple labels. The final cost function is the mean of the error for all training samples in a mini-batch:

$$E(W) = \frac{1}{n}\sum_{i=1}^{n} E_i$$

Note that, while the error function $E(W)$ of a CNN with ReLU non-linearity and/or max-pooling is non-differentiable with regards to W, it is still sub-differentiable, and by abuse of language we will use the term "the gradient" to refer to any sub-gradient of the error function in the rest of the text.

2.4 Weights Initialization

Although many successful deep CNNs for object recognition used random Gaussian initial weights with hand-tuned standard deviations [10], [16], we were unable to train our network with this initialization procedure - the training error never decreased.

We speculated that the smaller number of labels (compared to ImageNet) produced a bottleneck at the end of the network, hindering proper back-propagation of gradients. The initialization procedure recommended by Glorot et al. [5] was designed specifically for proper forward and backward propagation in the network, which is why we attempted to use it. Although their derivation is specific to sigmoid and hyperbolic tangent activation functions, we found empirically that it works well in practice with rectified linear activation functions as well.

After switching to this procedure, we found empirically that training was taking place by observing the training error going down much faster. For that reason, this is the weight initialization we ended up using. We refer to Ref. [5] for more details, as it generalizes to convolutional layers and dropout-regularized layers in a straightforward fashion.

Biases were initialized to 0 for the lowest and uppermost layers - we refer to a layer as lower when it is closer to the image, and upper when it is closer to the output. To force most hidden units to be active at the start of learning, we initialize the biases of all other layers to 1.

2.5 Training

Once initialized, we train our network using a variation on the RMSprop algorithm [7]. RMSprop is basically mini-batch stochastic gradient descent (SGD), where the gradient of the error for a given mini-batch with regards to weights W_t at a given iteration t is divided by a running estimate of the mean gradient magnitude over previous iterations. For that, we keep track of a squared gradient magnitude estimate separately for each of the m weights of the network in a vector $Msqr_t \in \mathbb{R}^m$.

We initialize the initial mean square gradient to the all ones vector: $Msqr_0 = \mathbf{1}_m$. While the algorithm still requires an initial learning rate $\alpha_0 \in \mathbb{R}^+$, we found that any "good enough" learning rate worked well after a few iterations, and that no annealing schedule for the learning rate was necessary until the end of training. We also found it useful to introduce a maximum learning rate $\alpha_{\max} \in \mathbb{R}^+$ which avoids some numerical issues and prevents learning from overshooting too far. Taking into account these 2 parameters, the update rule for our algorithm becomes:

$$Msqr_t = 0.9Msqr_{t-1} + 0.1\left(\frac{\delta E(W_t)}{\delta W_t}\right)^2$$

$$W_t = W_{t-1} - \min\left(\alpha_{\max}\mathbf{1}_m, \frac{\alpha_0}{\sqrt{Msqr_t}}\right)\frac{\delta E(W_t)}{\delta W_t}$$

where all operations - division, squaring, square root and minimum - are element-wise. We run the algorithm for a fixed number of epochs (i.e., passes over the entire training set), picking the mini-batches randomly without replacement for each epoch, so that each sample is picked exactly once per epoch. Unless specified otherwise, we used $\alpha_0 = \alpha_{\max} = 0.01$ in our experiments. We used a mini-batch size of about 128 images for training on the dataset of animation images. The actual number varies from batch to batch, as our dataset size is not divisible by 128 exactly.

2.6 Prediction

Although the output confidence values from our network can be directly thresholded to get multi-label predictions, we found in practice that it suffered from a key drawback. Often, the highest confidence predictions would be multiple predictions for the same object in the image. In the context of automatic image annotation which we target, it would be desirable to instead only pick the highest confidence prediction for a given object.

This is usually achieved for object detection via non-maximum suppression: given a set of overlapping predicted bounding boxes, only keep the highest-confidence one. This idea was used successfully for object detection [4] and for multi-label classification [15]. However, both these approaches wastefully evaluate a full forward pass for each object hypothesis in a single image, which makes them impractical in many settings.

Our approach first combines Lin et al.'s [12] global average pooling to produce confidence maps - with additional 1×1 convolutions in our case - and He et al.'s [6] insight that convolutional networks can be applied to full images instead of crops to produce a confidence map for each label in a single forward pass.

At this point, it should be noted that the network has not been trained to produce confidence values at varying scales, since we discard all spatial information at training time. We therefore found it beneficial to average out the confidence from multiple scales to obtain multi-scale confidence maps before pro-

ceeding further. For this purpose, we apply average pooling to each individual confidence map with a pyramid of kernel sizes: $\{3 \times 3, 5 \times 5, 7 \times 7\}$. We then upsample the resulting confidence maps to the original resolution, and average them out to obtain confidence maps which are scale-independent to some degree - i.e., whether the object detected is small or large in the image, the confidence will still be high at that point on the confidence map.

From these scale independent confidence maps, we can then solve the issues of multiple detections for a single object. Using the simple assumption that any given point of the image must belong to one and only one object, we only keep the highest confidence label for each spatial point in the feature map, and prune all other predictions entirely. This is similar in effect to non-maximum suppression for object bounding boxes in object detection, and to hypothesis max-pooling as in Wei et al. [15]. We were able to implement this method to run on a single GPU in negligible time compared to a forward pass. This makes our multi-label prediction method orders of magnitude more efficient than the state of the art in multi-label image classification [15], which requires as many forward passes as object hypotheses.

This last step produces a single confidence map for all labels, where each "pixel" of the confidence map has been associated a single label. We then simply sum up the remaining confidence values from this confidence map for each label. As the resolution of the feature map is typically much smaller than the number of labels, this results in most confidence values being 0, and reduces significantly the label space to choose from.

Once we have these final confidence values, we need to pick a confidence threshold to actually pick the set of labels to output. For this, we used the method recommended by Ref. [17], which consists in picking a threshold as a linear function of the confidence values, trained to maximize multi-label accuracy on the training data. In the case of our animation images dataset, since the training data is unnecessarily big for that purpose, we used the validation set instead. We refer to Ref. [17] for details.

2.7 Implementation Details

As many efficient and open source implementations of CNNs are available, we will not go into the full details of implementing convolutional, max-pooling or fully connected layers. For these, we relied on the routines provided by the Theano library [1], itself based on the Nvidia CuDNN library.

In a way similar to Krizhevsky et al. [10], at any given iteration t of SGD, we parallelized the computation of the gradient and update of the weights for batch D_t on the GPU with the pre-processing of the next batch D_{t+1} on the CPU. This allows pre-processing - e.g., loading the images from the disk, decompressing the images, picking random windows and flips - to essentially happen "for free" in terms of total computation time. Note that we found it necessary to store images on a solid-state drive to get pre-processing times short enough, and that JPEG decompression was then by far the most expensive operation.

During optimization of the cost function, all the network weights and mean squared gradients are stored on the GPU. At each SGD iteration, we transfer a pre-processed 128 images mini-batch to the GPU.

Using these techniques, we were able to train our network on the animation images dataset to convergence in about 3 weeks. Note that we achieved this run time using only a single Nvidia GTX 760 4 GB GPU.

3. A Large Scale Multi-label Dataset of Animation Images

In order to show the performance of our multi-label cost function, we needed a multi-label dataset adequate to train a deep CNN. Large, deep CNNs such as the ones used for object recognition have required large datasets in the past, of the order of one million images, to be trained successfully without over-fitting - like the ImageNet dataset. As animation images is the specific application domain we are interested in, we collected a dataset of about 1 million animation images. Our goal is to identify the characters in these images, for the purposes of automatic artwork annotation in web artist communities.

As most of the characters depicted in the images are under copyright from various entities, and as individual artworks are under copyright from their respective authors, it is not possible for us at the moment to redistribute our dataset publicly - although we would like to in the future.

In this section, we describe in details our collection methodology so that it is possible for other researchers to collect similar datasets and reproduce our results. Using this methodology, we were able to collect a 1 million images dataset suitable for training deep CNNs within a couple weeks - most of which was spent waiting for all images to be downloaded, actual work by the authors amounted to no more than a few days.

See **Figs. 1** and **2** for some statistics regarding our dataset.

3.1 Selecting Labels

Our first step was to manually collect the tags corresponding to the 100 most popular characters (in terms of number of tagged images) in the Pixiv web artist community. We chose the most popular characters for practical reasons - there is more interest in identifying popular characters, and these characters also are the ones for which we have the largest number of training samples, increasing prediction quality. We chose the 100 most popular

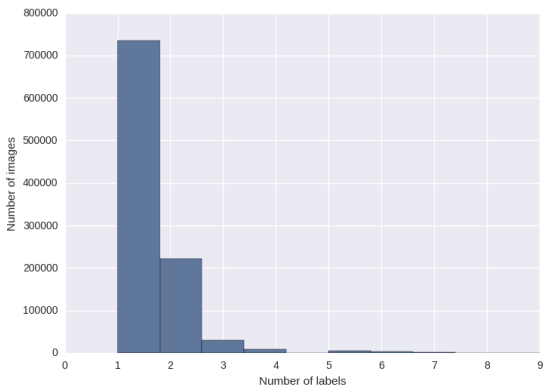


Fig. 1 Histogram of the number of labels per image. The average number of label per image is about 1.3, but there are some outliers with up to 9 labels. Note that the maximum number of tags per image in the Pixiv web artist community is 10, which effectively puts an upper bound on the number of labels per image in our dataset.

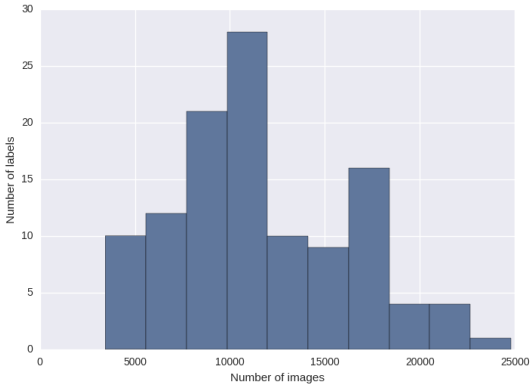


Fig. 2 Histogram of the number of images per label. The average number of images per label is about 11,000. This is much higher than in the ImageNet dataset for instance. The limit of at least 5,000 images is enforced by our collection methodology.

characters, as these characters have on average 13,000 images associated, bringing us (with the fact that these are multi-label images) to a rough total of 1 million images.

Once this step was finished, we obtained the images' tags and other metadata associated with these 100 character tags from Pixiv. This allowed us to notice that many images depicting multiple characters also include characters not within these 100 most popular ones. We therefore added to the label set those character tags which are not among the most popular but have at least 5,000 images in common with the 100 most popular ones. This limit of 5,000 also stems from practical limitations - a smaller limit would include too many tags to check manually, and may make the classes imbalanced in terms of number of samples - which is undesirable for training.

This process led us to a total of 125 character labels, but some of these labels were duplicates due to different names for the same character. As it would take a prohibitively long time to check all possible pairs of labels manually, we automated the process partly by only checking manually those pairs which either have a proportion of images in common greater than 50% for either one, or those whose label names are close in terms of Levenshtein distance. This brought the total to 115 character labels.

3.2 Collecting Images

For these 115 character tags, we downloaded all the (unique) images returned by Pixiv's search function for these tags. As it is too daunting a task to perform manually, it was entirely automated. We used the default search settings of Pixiv, and the only additional filtering was the exclusion of multi-image content: these submissions have a single set of tags for multiple images, such as multiple pages from a comic books, for instance. While it would be interesting to tackle the more general problem of multiple images sharing a single label set in the future, we do not yet know how one could train a CNN using such images. All images were converted to the JPEG format - if they were not originally in the JPEG format - for practical reasons in terms of available storage space. This resulted in a total of 1,003,192 images.

3.3 Training, Validation and Test Sets

The dataset was randomly partitioned into 3 subsets:

- A test set of 15,000 images, which the training algorithm does not have access to. We used it to evaluate the performance of our method after training.
- A validation set of 5,000 images, which the training algorithm uses for early stopping - i.e., we stop training when the validation error stops decreasing. Although this was our intention, we never actually reached that point, and it mostly served as a way to evaluate how quickly our algorithm was learning.
- A training set containing all the remaining images.

Compared to ImageNet, our test and validation sets are quite small. Although we did not complete that task yet for lack of time, we would like to manually check and correct possible errors in the ground truth labels from our collection procedure. We found, checking on small subsets on the order of 100 images, that some (although not many) images have partially incorrect ground truth labels - mostly missing labels, or labels of characters not present in the image. Although this has not proven to be an issue for training - as our network seems to be sufficiently robust to outliers - it is an issue for the proper evaluation of our method at test and validation time. A test + validation set of 20,000 images makes it realistic for a single person to check all the images in a few weeks.

With the aim of giving a rough idea of how many mislabelled images there are in the test set, we counted them to the best of our ability in a randomly selected subset of 1,000 images of the test set. We counted an image as mislabelled whenever there was a character in the image not in the ground truth but within the 115 selected characters, or a label in the ground truth not corresponding to a character in the image. We counted a total of 46 mislabelled images out of 1,000.

4. Experimental Results

4.1 Dataset of Animation Images

We trained our algorithm on the training set described in Section 3 for around 50 epochs, at which point the validation error had stopped improving, which took about 3 weeks. We used the hyper-parameters specified in Section 2. Using the prediction method described in Section 2.6, we computed the predicted label sets for each of the 15,000 images from the test set. Let n' denote the number of test samples, P_i denote the set of predicted labels for test sample i and Y_i its ground truth label set. We measured the quality of our prediction using 3 metrics:

- Precision measures the ability of the method to predict correct labels, but does not penalize if some labels are missing:

$$\frac{1}{n'}\sum_{i=1}^{n'}\frac{|Y_i \cap P_i|}{|P_i|}$$

- Recall measures the ability of the method to predict all the labels, but does not penalize if too many labels are predicted:

$$\frac{1}{n'}\sum_{i=1}^{n'}\frac{|Y_i \cap P_i|}{|Y_i|}$$

- Accuracy, also known as the Hamming score, measures the

Lovino Vargas

'Romano Vargas' by SakurahimaArt is licensed under CC BY-NC-ND 3.0.

Wang Yao

Mikasa Ackerman

'APH: Kiku vs. Yao' by AlaisL is licensed under CC BY-NC-ND 3.0.

Kise Ryouta

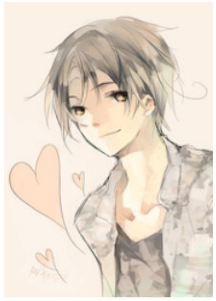
'Shalala Kise' by tsubakiya is licensed under CC BY-NC-ND 3.0.

Houjuu Nue

'The Heian Alien - Houjuu Nue' by Adeshark is licensed under CC BY-NC-ND 3.0.

Konpaku Youmu

'Youmu Konpaku' by KyokoProStudios is licensed under CC BY-NC-SA 3.0.



Feliciano Vargas

'Feliciano Vargas' by 10721 is licensed under CC BY-NC-ND 3.0.

Saigyouji Yuyuko

'yuyuko' by pcmaniac88 is licensed under CC BY-NC-ND 3.0.

Sanji

'Trafalgar Law, Sanji' by xyz263103 is licensed under CC BY-NC-ND 3.0.

Kaburagi T. Kotetsu

'Kotetsu x Barnaby' by moscovia is licensed under CC BY-NC-ND 3.0.

Fujiwara no Mokou

'Fujiwara no Mokou' by relicXth is licensed under CC BY-NC-ND 3.0.

Shameimaru Aya

'Aya Shameimaru' by Pokey-Chan is licensed under CC BY-NC-ND 3.0.

Megurine Luka

'Amami Haruka' by HaRiZiA is licensed under CC BY-NC-ND 3.0.

Mizuhashi Parsee

'Random sketch-Mizuhashi Parsee' by Adeshark is licensed under CC BY-NC-ND 3.0.

Kagiyama Hina

'Random Sniper Hina' by NPkappa is licensed under CC BY-NC-SA 3.0.

Kagamine Ren

'tachibana makoto' by MikaiSakura is licensed under CC BY-NC-ND 3.0.

Patchouli Knowledge

Kirisame Marisa

'Harem of Marisa' by Coffgirl is licensed under CC BY-NC-ND 3.0.

Fig. 3 Examples of predictions for 16 different images. Blue bars correspond to the confidence of the prediction. All images are licensed under a creative commons license. See the Appendix for full attribution information, including links to the images.

ability of the method to predict the correct label set, penalizing both missing and superfluous labels:

$$\frac{1}{n'}\sum_{i=1}^{n'}\frac{|Y_i \cap P_i|}{|Y_i \cup P_i|}$$

We measured 75.1% precision, 71.6% recall and 66.5% accuracy on our test set. As mentioned in Section 3, the test set ground truth is somewhat unreliable, and therefore one should be careful over-interpreting these metrics.

In the context of a web artist community, this kind of classification method could be used to predict the tags of an image as the user uploads it, with possibility of modifying them manually in case there is an erroneous or missing tag. In that case, we can hope that any accuracy number greater than 50% will save some time for the user. Depending on specific design concerns, precision might be more important than accuracy or recall, in the sense that a missing tag may not be as grave as an erroneous tag. During search for instance, the former simply means the image does not appear, while the latter implies it will show up in the wrong category - which is much more noticeable for the end user. In this sense, we believe that the accuracy and precision of 66.5% and 75.1% respectively make our method applicable in practice.

4.2 Pascal Dataset

Additionally, to demonstrate our method's ability to work on natural image datasets we applied it to the Pascal VOC 2007 dataset for multi-label classification. We did not attempt to set the state of the art on this dataset, as natural images are not the specific application domain we have in mind. However we are able to clearly show the value of our prediction method against a baseline.

As the Pascal dataset is too small to train a deep CNN from scratch, we started from a network pre-trained on the ImageNet dataset, which we fine-tuned to the Pascal dataset in a second step. This network is a variant of the network in network architecture by Lin et al. [12] (see **Table 2** for more information).

The network pre-trained on ImageNet is freely available from the Caffe library's model zoo web page [9]. We then fine-tuned our network using RMSprop, with 128 images per mini-batch, initial learning rate of $\alpha_0 = 0.001$ and maximum learning rate $\alpha_{\max} = 0.01$. Images underwent the same pre-processing as our animation images dataset. We used the full training + validation set from Pascal VOC 2007 as training data. We stopped training after 9 epochs, which only took a few minutes.

To show the value of our prediction method, we compared the training architecture as a baseline to the test architecture on the test data from Pascal VOC 2007. The test architecture uses the prediction method described in Section 2.6 to produce confidence values from full images, while the baseline averages confidence values from 10 randomly chosen 224×224 crops. In both case, we trained a linear threshold on the confidence values as in Section 2.6.

The results in **Table 3** show that our method outperforms the baseline significantly in accuracy and precision, while having reduced recall. We believe this to be beneficial in the context of automatic image annotation, as argued in Section 4.1. Notably, our prediction method requires only a single forward pass per image, while the baseline requires 10, which makes the former significantly more efficient.

Table 2 Architecture for the network used on the Pascal dataset. Layers 1 to 10 were pre-trained on ImageNet, subsequent layers were trained from scratch. The last layers share weights across training and test architecture.

Layer	Architecture
1	Convolutional, 96 7×7 filters, stride 4
2, 3	Convolutional, 96 1×1 filters Max pooling, 3×3 window and stride 2
4	Convolutional, 256 5×5 filters, padding 2
5, 6	Convolutional, 256 1×1 filters Max pooling, 3×3 window and stride 2
7	Convolutional, 384 3×3 filters, padding 1
8, 9	Convolutional, 384 1×1 filters Max pooling, 3×3 window and stride 2
10	Convolution, 1024 3×3 filters, padding 1

Layer	Training architecture	Test architecture
	Global average pooling	
11	Fully connected 20 units	Convolutional, 20 1×1 filters

Table 3 Results of our prediction method compared to a baseline method on the Pascal VOC 2007 dataset.

	accuracy	precision	recall
Our method	41.4%	54.5%	49.6%
Baseline	36.8%	47.1%	53.1%

Although our results are far from state of the art on the Pascal VOC 2007 dataset, averaging scores from multiple subwindows is the standard way to perform prediction in a number of state of the art CNN architectures [13], [14]. We believe that our method can both outperform such methods and be much more efficient.

5. Conclusion

In this paper, we introduced a new multi-label CNN design allowing training from large scale multi-label datasets, and efficient, accurate multi-label prediction by leveraging spatial and multi-scale information. Due to the lack of widely available large scale multi-label datasets, we collected a dataset of animation images to show our method's ability in the context of automatic image annotation. Although we can't redistribute the dataset, we make available a demo of our trained network online at the following url:

`avallet.vcd.design.kyushu-u.ac.jp/anime_recognizer`

We demonstrated that our method can provide high accuracy when trained with a large-scale multi-label dataset of animation images. With 66.5% accuracy and 75.1% precision, we believe it is adequate for artwork annotation in practice.

Furthermore, we showed our prediction technique improves accuracy and precision significantly on the Pascal VOC 2007 dataset compared to a baseline while being an order of magnitude more efficient.

References

[1] Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I.J., Bergeron, A., Bouchard, N. and Bengio, Y.: Theano: New features and speed improvements, *Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop* (2012).

[2] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K. and Le, Q.V., et al.: Large scale dis-

tributed deep networks, *Advances in Neural Information Processing Systems*, pp.1223–1231 (2012).
[3] Fukushima, K.: Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position, Vol.202 (1980).
[4] Girshick, R., Donahue, J., Darrell, T. and Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation, *2014 IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), pp.580–587, IEEE (2014).
[5] Glorot, X. and Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks, *International Conference on Artificial Intelligence and Statistics*, pp.249–256 (2010).
[6] He, K., Zhang, X., Ren, S. and Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition, *Computer Vision–ECCV 2014*, pp.346–361, Springer (2014).
[7] Hinton, G.E.: rmsprop: Divide the gradient by a running average of its recent magnitude, Lecture 6e, Neural Networks for Machine Learning (2012), available from ⟨https://www.coursera.org/course/neuralnets⟩.
[8] Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors, *arXiv preprint arXiv:1207.0580* (2012).
[9] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S. and Darrell, T.: Caffe: Convolutional Architecture for Fast Feature Embedding, *arXiv preprint arXiv:1408.5093* (2014).
[10] Krizhevsky, A., Sutskever, I. and Hinton, G.E.: Imagenet classification with deep convolutional neural networks, *Advances in Neural Information Processing Systems*, pp.1097–1105 (2012).
[11] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P.: Gradient-based learning applied to document recognition, *Proc. IEEE*, Vol.86, No.11, pp.2278–2324 (1998).
[12] Lin, M., Chen, Q. and Yan, S.: Network in network, *arXiv preprint arXiv:1312.4400* (2013).
[13] Simonyan, K. and Zisserman, A.: Very deep convolutional networks for large-scale image recognition, *arXiv preprint arXiv:1409.1556*, pp.1–12 (online) (2014), available from ⟨http://arxiv.org/abs/1409.1556⟩.
[14] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A.: Going Deeper with Convolutions, pp.1–12 (online) (2014), available from ⟨http://arxiv.org/abs/1409.4842v1⟩.
[15] Wei, Y., Xia, W., Huang, J., Ni, B., Dong, J., Zhao, Y. and Yan, S.: CNN: Single-label to multi-label, *arXiv preprint arXiv:1406.5726* (2014).
[16] Zeiler, M.D. and Fergus, R.: Visualizing and understanding convolutional networks, *Computer Vision–ECCV 2014*, pp.818–833, Springer (2014).
[17] Zhang, M.-L. and Zhou, Z.-H.: Multilabel neural networks with applications to functional genomics and text categorization, *IEEE Trans. Knowledge and Data Engineering*, Vol.18, No.10, pp.1338–1351 (2006).

Appendix

Image Attributions

Images from Fig. 3, from left to right, top to bottom:

- 'Romano Vargas' by SakurahimaArt is licensed under CC BY-NC-ND 3.0. URL: http://sakurahimeart.deviantart.com/art/Romano-Vargas-37076664
- 'Youmu Konpaku' by KyokoProStudios is licensed under CC BY-NC-SA 3.0. URL: http://kyokoprostudios.deviantart.com/art/Youmu-Konpaku-363218438
- 'Kotetsu x Barnaby' by moscovia is licensed under CC BY-NC-ND 3.0. URL: http://moscovia.deviantart.com/art/Kotetsu-x-Barnaby-260778579
- 'Random sketch-Mizuhashi Parsee' by Adeshark is licensed under CC BY-NC-ND 3.0. URL: http://adeshark.deviantart.com/art/Random-sketch-Mizuhashi-Parsee-124188640
- 'Feliciano Vargas' by 10721 is licensed under CC BY-NC-ND 3.0. URL: http://10721.deviantart.com/art/Feliciano-Vargas-180254301
- 'APH: Kiku vs. Yao' by AlaisL is licensed under CC BY-NC-ND 3.0. URL: http://alaisl.deviantart.com/art/APH-Kiku-vs-Yao-121730274
- 'Fujiwara no Mokou' by relicXth is licensed under CC BY-NC-ND 3.0. URL: http://relicxth.deviantart.com/art/Fujiwara-no-Mokou-141088719
- 'Random Sniper Hina' by NPkappa is licensed under CC BY-NC-SA 3.0. URL: http://npkappa.deviantart.com/art/Random-Sniper-Hina-266381430
- 'yuyuko' by pcmaniac88 is licensed under CC BY-NC-ND 3.0. URL: http://pcmaniac88.deviantart.com/art/yuyuko-163583588
- 'Shalala Kise' by tsubakiya is licensed under CC BY-NC-ND 3.0. URL: http://tsubakiya.deviantart.com/art/Shalala-Kise-306222249
- 'Aya Shameimaru' by Pokey-Chan is licensed under CC BY-NC-ND 3.0. URL: http://pokey-chan.deviantart.com/art/Touhou-Aya-Shameimaru-136315234
- 'tachibana makoto' by MikaiSakura is licensed under CC BY-NC-ND 3.0. URL: http://mikaisakura.deviantart.com/art/tachibana-makoto-384044366
- 'Trafalgar Law, Sanji' by xyz263103 is licensed under CC BY-NC-ND 3.0. URL: http://xyz263103.deviantart.com/art/Trafalgar-Law-Sanji-208527782
- 'The Heian Alien - Houjuu Nue' by Adeshark is licensed under CC BY-NC-ND 3.0. URL: http://adeshark.deviantart.com/art/The-Heian-Alien-Houjuu-Nue-134102089
- 'Amami Haruka' by HaRiZiA is licensed under CC BY-NC-ND 3.0. URL: http://harizia.deviantart.com/art/Amami-Haruka-495899410
- 'Harem of Marisa' by Coffgirl is licensed under CC BY-NC-ND 3.0. URL: http://coffgirl.deviantart.com/art/Harem-of-Marisa-149635501

Alexis Vallet received a M.E. degree from the University of Technology of Belfort-Montbéliard, France, in 2013. He is a Ph.D. student at Kyushu University, Japan. His research interests include computer vision and machine learning, focusing on deep learning and applications to web artist communities.

Hiroyasu Sakamoto received B.E., M.E. and D.E. degrees from the Kyushu Institute of Technology in 1975, 1977 and 2000, respectively. He joined the Kyushu Institute of Technology in 1977 and in 1986 he joined the Kyushu Institute of Design. Presently, he is a professor of Kyushu University. His research interests include computer vision, image processing, pattern recognition and 3D mesh processing. He is a member of IPSJ, IEICE and ITE.