CHILDNet: Curiosity-driven Human-In-the-Loop Deep Network

Byungwoo Kang
Stanford University
Department of Physics
bkang@stanford.edu

Hyun Sik Kim
Stanford University
Department of Electrical Engineering
hsik@stanford.edu

Donsuk Lee
Stanford University
Department of Computer Science
donlee90@stanford.edu

Abstract

While deep learning has been remarkably successful in the domain of computer vision, most of its success so far has relied on large-scale training data that are human-annotated. With the goal of reducing the enormous cost associated with compiling such large-scale dataset in mind, we develop an actively learning model that can incrementally learn new visual objects. The main novelty of our model is to have separate vision and Reinforcement Learning (RL) modules. The vision module extracts relevant features in such a way that allows few-shot learning, and the RL module makes decisions whether to request a label or make a prediction based on the features extracted from the vision module. Our model outperforms the model proposed in [11] by achieving a higher prediction accuracy with fewer label requests on the same test setting as theirs.

1. Introduction

Many of the most successful deep learning models in the domain of computer vision are trained on large-scale datasets in a strongly supervised fashion. As such large-scale datasets are often labelled by humans, which may be rather expensive, it is desirable to have an actively learning agent that can learn on its own by exploring and interacting with its environment.

Based on this motivation, we develop an actively learning model that can incrementally learn new visual objects. Our model continuously discovers new object classes, request their labels, and learn to recognize them. For simplicity, we focus on training on single-object image datasets, such as Omniglot. Our model sequentially receive a stream of images one by one and learn to make a prediction if it is confident that the image is in a class already seen, or request a label otherwise.

To build our model, we combine ideas from the siamese network [5], iCaRL [8], and active one-shot learning [11]. Our model consists of vision and Reinforcement Learning (RL) modules.The vision module trained in the siamese fashion extracts relevant features from the input image and keeps track of class prototypes in a manner inspired by [8]. The RL module inspired by [11] makes decisions whether to make a prediction or request a label based on the new image's feature vector and the stored class prototypes. The vision module then either makes a prediction or requests a label, according to the decision made by the RL module.

Our main contribution is twofold. First, our model serves as a proof of concept that an RL module can find a reasonably good policy regarding whether to make a prediction or request a label based on high-level features from a vision module. Second, to the best of our knowledge, our model is the first neural network model to incrementally learn an arbitrary number of classes while requesting class labels for examples that they are uncertain about. Moreover, for a fixed number of classes, our model outperforms the model proposed in [11] by achieving a higher prediction accuracy while requiring fewer label annotations.

2. Related Work

Since image streams that our model is expected to encounter are likely to have only few examples per class, our vision module needs to be able to do few-shot learning. Some notable previous proposals to tackle few-shot learning include the siamese network [5], memory-augmented neural network, [9], matching network [10], and model-agnostic meta-learning [2]. For simplicity, we use the siamese network as our vision module. In addition, since it classifies images by a nearest-neighbor algorithm in the

learned metric space, it is particularly suitable for incremental learning. However, we expect our general strategy to be applicable to more sophisticated models designed for few-shot learning including the ones mentioned above.

On the other hand, our vision module needs to encounter and learn an increasing number of classes over time. This type of learning problem, called 'incremental learning' has been addressed in previous works including notably [6, 8]. We find the approach of [8] particularly appealing, because it uses a nearest-neighbor algorithm in the final feature space for classification of images as in the siamese network. Although their problem setting differs from ours in that all instances from each class are consecutively arranged in the image stream, we take inspiration from their idea to use the average of the feature vectors for each class as class prototypes.

Finally, our model needs to be able to decide when to make a prediction and request a label, based on its current knowledge. A problem setting very similar to ours is addressed by [11], which also partly inspired this project. The only difference from our problem setting is that their image stream contains only a fixed number of classes. Also, unlike our model where the vision and RL modules are separate, in their model, a single LSTM plays the role of both modules simultaneously. More precisely, their LSTM approximates the optimal action-value function where the state is the concatenation of a new image flattened into a vector with the previously requested label (if no label is requested, a zero vector is concatenated) and the action is either the predicted label or label request. We instead use the policy gradient method as explained more in details below.

3. Approach

Our goal is to build a learning system, realized by a neural network, that incrementally learns to recognize new image classes from a continuous unlabelled image stream. To achieve this goal, it has to decide whether a given image belongs to a class it has never seen before, and if so, request an external annotator (e.g. human experts, or crowdworkers) to provide a label for it. On the other hand, if it decides that the given image belongs to the learned classes, it makes a prediction. This setting can be thought of as a combination of the two well-known learning problems: online active learning [7] and incremental learning [8].

Formally, we have an image data stream $X = \{x_1, x_2, ...\}$ with their corresponding class labels given by $Y = \{y_1, y_2, ...\}$, which are only revealed upon requests. A requested label is supplied before proceeding to the next image in the stream. We also emphasize that y_i's are not necessarily distinct from each other. The number of distinct class labels may increase over time, as an instance of a new class may appear at any time. Ideally, our neural network has to request a label for every new class it encounters and makes prefect predictions for images belonging to the learned classes. Therefore, as training progresses, we expect that label request percentage will increase for the first instance of a class while decreasing for later instances. Similarly, we expect prediction accuracy to increase for later instances of a class. In other words, our network should have a sense of what it knows and does not know, and ask for information about novel examples.

Since our network has to learn from images presented in a sequence, and since the images from each object class are randomly and sparsely distributed in the sequence, it needs to be able to do few-shot learning. That is, from the first instance of a particular class, it needs to extract enough relevant features so that it can make an accurate prediction next time it sees an instance of that class. At the same time, the number of the object classes to classify is not fixed, but increases as the network sees more images. In other words, the network needs to be an incremental classifier. Moreover, it should decide whether a given image belongs to a class it has already seen or a new class. Based on this decision, it either makes a prediction or request a label. To build such a network, we combine three different ideas proposed before: siamese network [5], iCaRL [8], and active one-shot learning [11].

The siamese network is trained to perform verification task, the goal of which is to decide whether a given pair of images belong to the same or different classes. It tells whether image pairs are from the same class or not by extracting feature vectors through a convolutional network and then measuring the pair's similarity by a learned similarity metric. It turns out that the siamese network, trained only for verification task, also excels at few-shot learning. When adapted for the few-shot learning task, it classifies an image by measuring how similar its feature vector is to the stored features vectors whose class identities are known. The class of the image is then predicted to be the class identity of the most similar stored feature vector. Since this nearest-neighbor approach can work regardless of the number of the stored feature vectors, it is also suitable for incremental learning. Given the necessity of few-shot and incremental learning in our problem setting, the siamese network is therefore a natural choice for the vision module in our network.

Figure 1 illustrates the detailed architecture of our vision module. Specifically, it extracts feature vectors through four identical convolutional modules each of which consists of a 3×3 convolutions with 64 filters, batch normalization, a ReLU nonlinearity, and 2×2 max-pooling. Then, for a given pair of images, we take the absolute element-wise difference between their flattened feature vectors, and apply an affine transformation and sigmoid function on the difference vector to get the probability of the pair belonging to the same class.

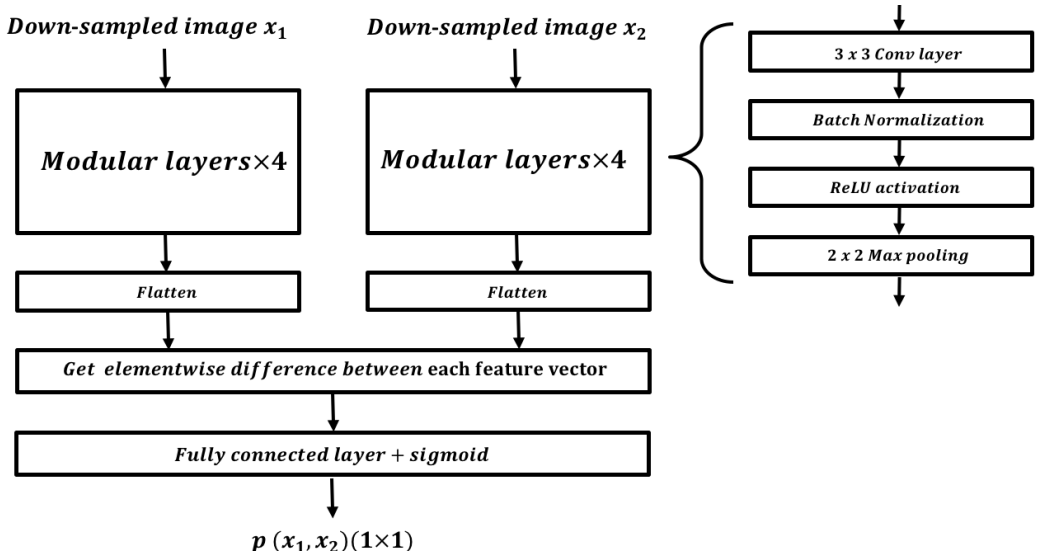

Figure 1: Our vision module's architecture

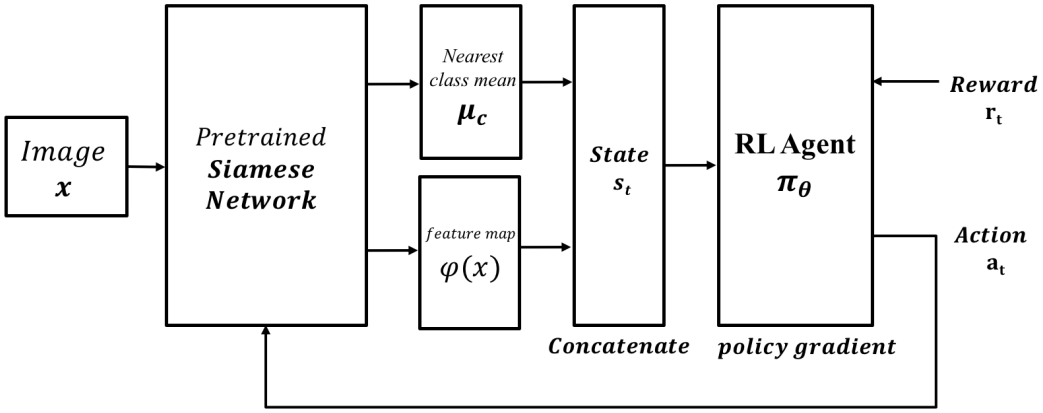

Figure 2: A schematic view of our entire model.

For classification task, the vision module needs to keep and update the sets of representative feature vectors for the learned classes, which we refer to as 'class prototypes'. A nice algorithm is proposed in [8] to systematically manage such class prototypes. Although we do not exactly follow the iCaRL algorithm proposed in [8] due to differences in details of the problem setting, we take inspiration from their general idea and use the running average of feature vectors encountered so far for each learned class as the class prototype for that class. More details on updating class prototypes are illustrated in Algorithm 1.

Finally, inspired by the idea of active one-shot learning [11], we introduce a Reinforcement Learning (RL) module to make decisions whether to make a prediction or request a label. Our main novelty is to use separate vision and RL modules and train the RL module using the policy gradient method. A schematic description of the two modules interacting with each other is given in Figure 2. The input to the RL module is the concatenation of the new image feature vector and the nearest mean. In principle, we could concatenate this with other running means further away, but we found that the RL module performs well enough with just the nearest running mean. Intuitively, the RL module has to decide to make a prediction if the new image feature vector is close enough to the nearest running mean, and request a label otherwise. If it decides to request a label, the new image's feature vector is used to update (or create, in case it belongs to a genuinely new class) the running mean cor-

Algorithm 1 UPDATECLASSPROTOTYPE

input y // class label
input $\varphi_y \in R^d$ // feature of an instance of class y
require $P = \{p_c\}_{c=1}^{C}$ // class mean of known classes
require λ // decay rate
  if $y \in \{1, ..., C\}$ **then**
    $p_y \leftarrow (1 - \lambda)p_y + \lambda\varphi_y$
  else
    $p_y \leftarrow \varphi_y$
    $P \leftarrow P \cup \{p_y\}$
  end if

responding to its class. We give the RL module a positive reward for a correction prediction, a negative reward for an incorrect prediction, and a small negative reward for a label request, because in practice label annotations by crowdworkers are expensive. The decision made by the RL module is sent to the vision module, which either makes a prediction or requests a label as dictated by the RL module. The reward signal is finally sent to the RL module based on the correctness of the prediction in case the vision module makes a prediction. Otherwise, a fixed reward for a label request is sent to the RL module.

The astute reader might wonder whether we can instead use a classifier trained in the standard supervised way to do the decision-making. There are two main reasons why we believe our RL module is superior to the supervised classifier. First, as we show below, if we use reinforcement learning, it is easy to trade off prediction accuracy with reduced label requests by varying the value of the reward for a label request. For example, if the reward becomes more negative, it means that label requests are penalized more and therefore at the expense of less accurate predictions, the RL module can reduce the cost of label requests. On the other hand, if it becomes less negative, at the cost of more label requests, the RL module can make more accurate predictions. This kind of trade-off seems difficult to accomplish in the standard supervised setting. Second, there could be situations where the confidence level of the decision made by the supervised classifier is not high enough that it would be more beneficial in the long run to request a label to update the running means of the class feature vectors. The RL module can potentially deal with these situations better, because it takes an action that will be most beneficial in the long run, as evaluated by the sum of discounted future rewards. Thus, it will request a label for an image that it marginally believes to belong to one of the learned classes, if such an action leads to more accurate knowledge about the image's class and is therefore more advantageous in the long run.

Our combined network is trained as follows. First, we train the vision module on verification task. Once it achieves good performance on one-shot learning, we freeze its parameters and use it as a fixed feature extractor. Then, using features provided by the vision module, the RL module is trained by the policy gradient method. More details can found in Algorithm 2.

Algorithm 2 TRAINCHILDNet

input $\mathcal{D} = \{(x_0, y_0), ..., (x_T, y_T)\}$
require $\varphi : \chi \rightarrow \mathbb{R}^d$ // Feature extractor
require $W : \mathbb{R}^{1\times d}$ // Weight vectors in the final layer of siamese network
require $P = \{p_c\}_{c=1}^{C}$ // Class mean of known classes
require $\pi_\theta : \mathbb{R}^{2d} \rightarrow \{0, 1\}$
require $R_{inc}, R_{cor}, R_{req}$
  for $t = 0, \ldots, T$ **do**
    $C^* \leftarrow argmax_c W|\varphi(x_t) - p_c|$
    $s_t \leftarrow (\varphi(x_t), p_{C^*})$
    $a_t \leftarrow argmax_{a\in\{0,1\}}\pi_\theta(s_t, a)$
    if $a_t = 0$ **then**
      if $C^* = y_t$ **then**
        $r_t \leftarrow R_{cor}$
      else
        $r_t \leftarrow R_{inc}$
      end if
    else
      $r_t \leftarrow R_{req}$
      UPDATECLASSPROTOTYPE$(\varphi(x_t), y_t)$
    end if
  end for
  $\theta \leftarrow \theta + \alpha\nabla_\theta(\pi_\theta(s_t, a_t))v_t$ // update RL agent

4. Experiment

Figure 4: Examples from the Omniglot dateset

4.1. Dataset

We use the Omniglot dataset [1] to train and evaluate our model. Omniglot contains 20 hand-drawn examples for each of 1,623 characters from 50 different alphabets. It is especially suitable for one or few shot classification task,

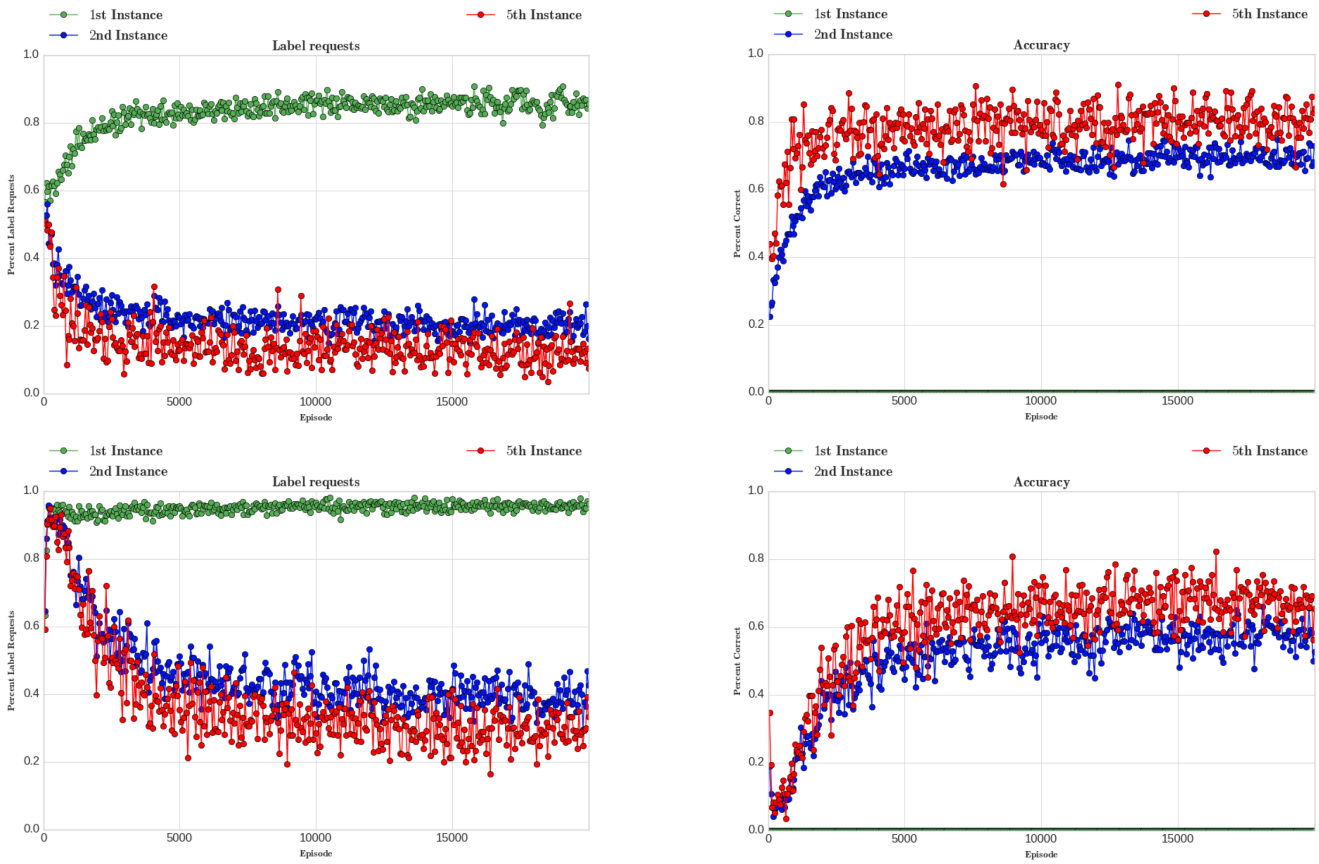

Figure 3: Left: Change of label request percentage over the training episodes for the 1st, 2nd, and 5th instances of all classes. Right: Change of prediction accuracy over the training episodes for the 1st, 2nd, and 5th instances of all classes. The top row corresponds to the case of $R_{inc} = -1$, and the bottom row to $R_{inc} = -10$.

since it has a relatively small number of examples per class and a larger number of classes. In our experiments, we randomly split the Omniglot dataset into 800 training classes, 400 validation classes and 423 test classes. The images are downsampled to 28 x 28 and converted into grayscale.

4.2. Training of the vision module

For verification task, we prepare roughly the same number of same and different pairs. To form the set of same pairs, we sample all possible example pairs for each class in the training set. This results in $152,000 = (10 \times 19) \times 800$ same pairs in total. On the other hand, there are vastly more possible different pairs than the same pairs, because there are $319,600 = 400 \times 799$ distinct-class pairs, and for each distinct-class pair, there are $400 = 20 \times 20$ possible different pairs. To ensure that we have roughly the same number of same and different pairs and that we sample as many different distinct-class pairs as possible, we sample one random different pair for each distinct-class pair to have total $319,600$ different pairs and duplicate each same pair to have total $304,000 = 2 \times 152,000$ same pairs. Therefore, the total number of pair examples in the training set is $623,600$. The validation and test sets are similarly prepared. During training, each image is rotated by a random integer multiple of 90 degrees. The random rotation is not applied at test time. We compute the verification accuracy on the validation set after each training epoch, and select the model with the highest accuracy to be used in the training of the RL module.

4.3. Training of the RL module

We generally follow the experiment steps in [11] with some modifications. Each training episode consists of 30 images sampled randomly from 10 randomly sampled classes. We vary the number of classes per episode during the test time to demonstrate that our model is capable of incremental learning. Specifically, each test episode consists of all the images from N randomly sampled classes, where $N \in \{3, 10, 20, 40\}$. Note that in [11] three randomly selected classes are used per episode at both the training and

test time.

Throughout our experiment, we set the discount factor $\gamma = 0.5$. The rewards for a correct prediction and a label request, R_{cor} and R_{req}, are fixed to 1 and -0.05, respectively, while the reward for an incorrect prediction R_{inc} is varied among three different values $\{-1, -5, -10\}$ to see the aforementioned trade-off. The RL module is optimized using the Adam optimizer [4] with the default hyperparameters and learning rate $\eta = 10^{-4}$. The total number of training episodes is $20,000$, and the parameters are updated after each episode.

4.4. Results

For every 50 training episodes, we count the number of label requests n_{req}, correct predictions n_{cor}, and incorrect predictions n_{inc} for the 1st, 2nd and 5th instances of all classes. The label request percentage and prediction accuracy plotted in Figure 3 is defined as $n_{req}/(n_{cor} + n_{inc} + n_{req})$ and $n_{cor}/(n_{cor} + n_{inc} + n_{req})$, respectively. As shown in the left two plots of Figure 3, our model learns to make more label requests for the first instances and fewer for the later instances. Furthermore, as shown in the right two plots of the same figure, prediction accuracy is higher for later instances than early instances. These results taken together suggest that our model incrementally learns new image classes while requesting labels for instances it is uncertain about.

	Accuracy (%)	Requests (%)
Supervised	93.4	100
Ours ($R_{inc} = -1$)	87.7	17.0
Ours ($R_{inc} = -1$)	92.1	24.1
Ours ($R_{inc} = -1$)	93.2	26.3

Table 1: Trading Accuracy for Requests

Trading off prediction accuracy with reduced label requests. By varying the value of R_{inc}, we are able to trade off prediction accuracy with reduced label requests. We present how prediction accuracy and label request frequency change with the value of R_{inc} in Table 1. As expected, as R_{inc} becomes more negative, incorrect predictions are more severely penalized, and consequently the RL module learns to improve its prediction accuracy by making more label requests. Also, our model achieves nearly the same task performance as the fully supervised control model, which updates its class prototypes using all the examples in the episode before classification, with significantly less information about the class labels.

Varying the number of classes. While our model is trained with only 10 classes per episode, it performs reasonably well even for larger numbers of classes per episode, indicating that it can flexibly deal with varying numbers of classes. Table 2 summarizes our model's performance for different numbers of classes per episode ($R_{inc} = -1$).

	Accuracy (%)	Requests (%)
Number of classes = 3	96.4	16.7
Number of classes = 10	87.7	17.0
Number of classes = 20	77.1	17.2
Number of classes = 40	64.9	16.7

Table 2: Varying number of classes

5. Conclusion

In this project, we presented a model that can learn to recognize new classes online using as few examples as possible. We formulate the online active learning as a reinforcement learning problem. Our results demonstrate that the RL module learns to request labels only when it is uncertain about its prediction. Further, our model can adapt to class-incremental settings, in which the number of classes increases over time.

In future work, we first plan to evaluate our approach on more complex image datasets such as ImageNet. For this, we may need a more powerful visual feature extractor, such as ResNet [3], and more sophisticated one-shot learning approach such as Matching Network [10]. Finally, our ultimate goal is to build an ever-expanding dataset collector with humans in the loop, which continuously crawls images on the web, discover new visual concepts and ask for labels to human annotators. We are planning to scale up our model and employ it on the social media platforms such as Instagram.

References

[1] S. Ager. Omniglot - writing systems and languages of the world. `Inwww.omniglot.com`, 2015.

[2] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *ICML*, 2017.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CVPR*, 2016.

[4] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[5] G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. *ICML*, 2015.

[6] Z. Li and D. Hoiem. Learning without forgetting. *European Conference on Computer Vision*, 2016.

[7] E. Lughofer. Single-pass active learning with conflict and ignorance. *Evolving Systems*, 2012.

[8] S. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. icarl: Incremental classifier and representation learning. *CVPR*, 2017.

[9] A. Santoro, S. Bartunov, M. Botvinick, and D. Wierstra. One-shot learning with memory-augmented neural networks. *ICML*, 2016.

[10] O. Vinyals, C. Blundell, T. Lillicrap, k. kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. *NIPS*, 2016.

[11] M. Woodward and C. Finn. Active one-shot learning. *Workshop on Deep Reinforcement Learning, NIPS*, 2016.