TASK DETECTORS FOR PROGRESSIVE SYSTEMS

by

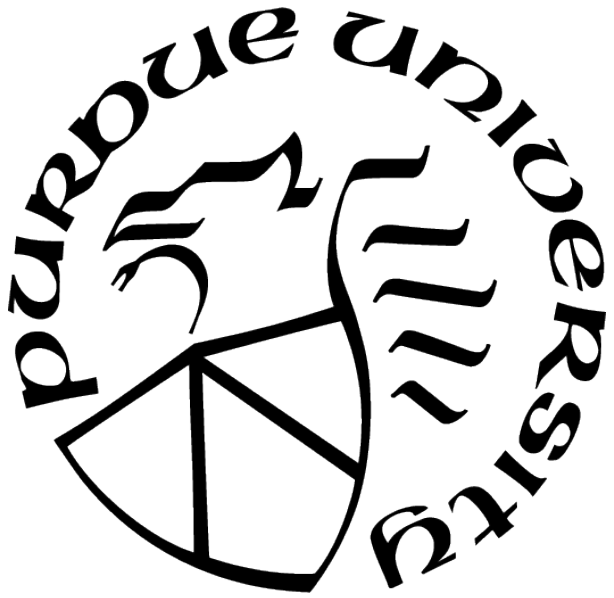
Maxwell J. Jacobson

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science



Department of Computer Science

West Lafayette, Indiana

December 2020

1 Abstract

While methods like learning-without-forgetting [1] and elastic weight consolidation [2] accomplish high-quality transfer learning while mitigating catastrophic forgetting, progressive techniques such as Deepmind's progressive neural network accomplish this while completely nullifying forgetting. However, progressive systems like this strictly require task labels during test time. In this paper, I introduce a novel task recognizer built from anomaly detection autoencoders that is capable of detecting the nature of the required task from input data. Alongside a progressive neural network or other progressive learning system, this task-aware network is capable of operating without task labels during run time while maintaining any catastrophic forgetting reduction measures implemented by the task model.

2 Introduction

Transfer learning is the process of transferring knowledge effectively from one task-model pair to another. Some advantages of powerful transfer learning methods include the ability to leverage pre-trained models in new tasks, and the option of further training models that were once trained on a proprietary or unavailable dataset. Transfer learning is an important and widely-applied area of study. However, the Holy Grail of transfer learning is the promise of creating a continual learning system.

While many attempts have been made to construct a true continual learning system — a program capable of learning while simultaneously performing its job effectively — none have yet succeeded. However, a necessary step forward came in 2016 when Deepmind released *Progressive Neural Networks* [3], and introduced a neural algorithm capable of learning new tasks and leveraging transfer learning in a powerful way all while completely immunizing the model to catastrophic forgetting. According to the paper's introduction, "progressive networks naturally accumulate experiences and are immune to catastrophic forgetting by design, making them an ideal springboard for tackling long-standing problems of continual or lifelong learning" [3].

With the objective of incrementing further towards a continual learning system, this article introduces a simple yet novel extension to progressive networks that dramatically

increases test time functionality while maintaining the catastrophic forgetting immunity of the task model. I also document two experiments that demonstrate the efficacy of the task detector. Finally, I propose further avenues of research that could lead to great improvements in the competence of the task detector algorithm.

3 Background

3.1 Types of Learning

A trinity of scholarly frameworks largely comprises the field of machine learning: supervised, unsupervised, and reinforcement learning. Perhaps the most straightforward and well-known of these paradigms is supervised learning, in which a machine learning model is tasked with learning to mimic a function — mapping inputs to outputs. This is accomplished with a labeled training dataset. Examples of supervised learning tasks include classification and regression. Neural networks lend themselves to this style of learning particularly well.

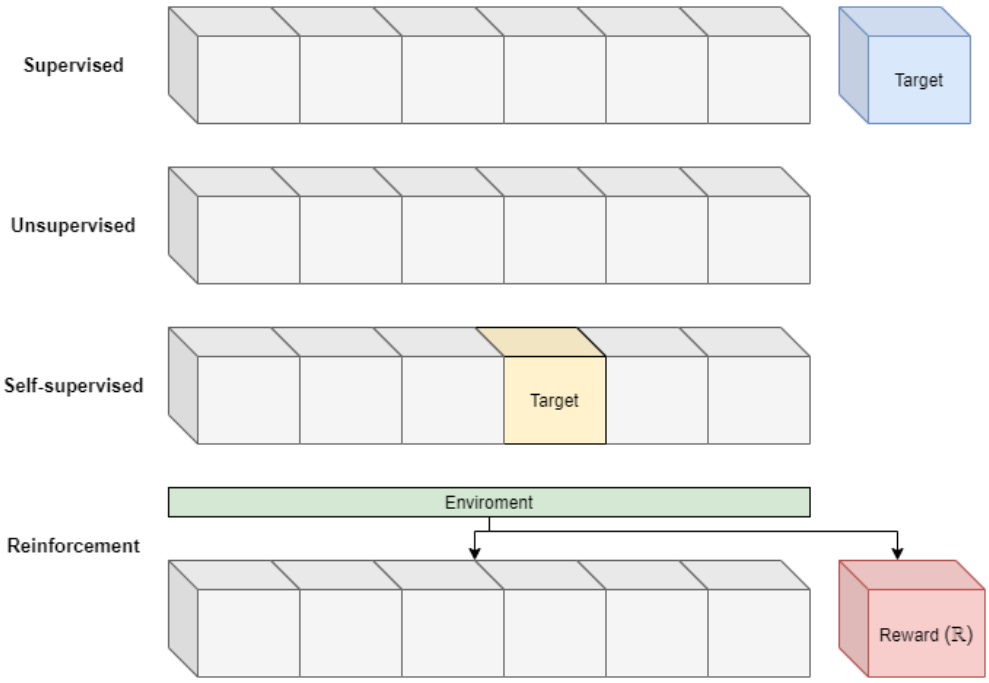


Figure 1. A visual representation of data supplied for supervised, unsupervised, self-supervised, and reinforcement learning.

Unsupervised learning posits much less in the way of structured data; tasks involve learning patterns in unlabeled data or knowledge about the data's underlying distribution. Clustering and generation tasks are commonly accomplished with unsupervised methods. An exceptionally interesting and useful form of unsupervised learning is self-supervised learning — a set of methods that allows an unsupervised system to learn in a supervised way by exploiting features of the data. It does this by learning to predict any part of the data from one given part of the data. Yann LeCun, speaking at École polytechnique fédérale de Lausanne explained "you can use supervised learning for [predicting any part of the data] but the supervisory signal is the input itself at a different location or time" [4]. Autoencoders — discussed in-depth later in this paper — use self-supervised learning to accomplish representation learning and data transformation tasks. Other important examples are recent state-of-the-art language models such as Word2Vec [5, 6], Glove [7], ELMO [8], BERT [9], and GPT [10] — these language models train on uncurated text strings, occluding one token at a time and attempting to discover the missing item.

Finally, reinforcement learning is a type of learning for tasks in which the system operates an agent within an environment. The environment supplies the agent with a state S_t. Then the agent sends an action A_t to the environment, which responds with a scalar reward R_t and a new state S_{t+1}. Using only the data of the state, the agent must learn to maximize the reward with its actions. This structure makes reinforcement learning systems excellent for game playing, where the optimal action is rarely known but a score or other quantitative measure of success may be available. For this reason, reinforcement learning is a great framework for testing artificial intelligence systems — as games are constructed for the purpose of challenging humans.

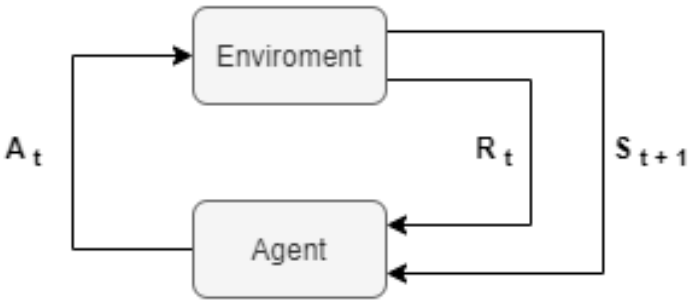


Figure 2. A diagram of the reinforcement learning framework. The agent observes and affects the environment to maximize reward.

Reinforcement learning is itself divided into three broad approaches. The first — value iteration — involves learning to predict the expected reward given certain conditions. An example of value iteration is Q-learning, in which the expected reward is predicted given the current state and a potential action according to the action-value Bellman equation. While Q-learning at its conception was implemented as a dynamic programming algorithm, a neural version has since been defined — the deep Q-learning network or DQN [11].

$$Q_{\pi}(s_t, a_t) \;=\; E(r_{t+1} + \gamma * r_{t+2} + \gamma^2 * r_{t+3} + ... \mid s_t, a_t)$$

Figure 3. The Q function mapping state and action to reward. γ is a $(0.0, 1.0)$ discount factor that reduces the importance of rewards farther in the future.

The second family of reinforcement learning methods is policy optimization. With a policy function defined as a function mapping states to actions, policy optimization algorithms directly optimize a policy network. One commonly used policy optimization procedure is the REINFORCE algorithm [12, 13], which applies gradient ascent to log probabilities of Monte Carlo sampled trajectories.

$$\theta \;\leftarrow\; \theta + \alpha * \nabla_{\theta} log(\boldsymbol{\pi}_{\theta}(s_t, a_t)) * v_t$$

Figure 4. Update rule for REINFORCE, variations of which are used in many policy optimization and actor-critic algorithms. v_t is the sum of rewards over the training step's trajectory τ.

Actor-Critic methods — the final major branch of reinforcement learning — combine the concepts of value iteration and policy optimization. In actor-critic algorithms, at least two models are maintained: the actor models the policy function and is trained with the help of a value-iterating critic. Actor-Critics like A2C and A3C [14] use the advantage function as a critic — the reward advantage of choosing a given action over the baseline reward value of the state. Another popular Actor-Critic is Deep Deterministic Policy Gradient (DDPG) [15], which uses a deterministic policy network, a DQN critic network, and a pair of target networks to train the recursively-defined actor and critic against a static parameter set.

Machine learning is a field of enormous depth, and the many algorithms that allow computers to perform human tasks are a testament to this fact. These algorithms, especially neural networks, continue to improve in their ability to interact with human tasks, but can, however, still struggle with combining the learned knowledge of these tasks into a comprehensive system.

3.2 Transfer Learning & Progressive Systems

Transfer learning is the process of transferring the competence from one machine learning system learning a task to another system learning a related task. Transfer learning is deeply entangled with the concept of continual learning — that is, ML systems that are capable of constantly improving while remaining stable on previous tasks. Many solutions to transfer learning exist, the simplest being finetuning, the retraining of an initial task on a new task, sometimes with additional parameters or a smaller learning rate. While finetuning is suitable for simple transfers, it may not be fully suitable due to numerous potential problems, chiefly, a deleterious event known as *catastrophic forgetting* [16, 17], which occurs when important parameters within the network are changed to fit the new data, compromising the network's ability to handle previously-learned data. Some innovations have been made in the effort to curb catastrophic forgetting, such as the method postulated in *Learning without Forgetting* [1], but these merely resist the effect, and will not fully prevent it given particularly difficult or intricate tasks.

A step towards true continual learning was made in the 2016 Deepmind paper *Progressive Neural Networks*, in which the authors present a simple yet novel approach to progressive

transfer learning in neural networks. Comparing progressive networks with conventional finetuning, they explain "While finetuning incorporates prior knowledge only at initialization, progressive networks retain a pool of pretrained models throughout training, and learn lateral connections from these to extract useful features for the new task. By combining previously learned features in this manner, progressive networks achieve a richer compositionality, in which prior knowledge is no longer transient and can be integrated at each layer of the feature hierarchy. Moreover, the addition of new capacity alongside pretrained networks gives these models the flexibility to both reuse old computations and learn new ones" [3].

Each task network is represented as a column made up of blocks with lateral connections linking them to the preceding blocks of all previous columns. These lateral connections allow future columns to access and finetune learned knowledge. The progressive network starts with just one L-block column learning a single task $k = 0$. Each block $h_{\mathrm{i},0}$ contains parameters $W_{\mathrm{i},0}$. Once the column is trained, its parameters are frozen. This step grants the progressive network immunity to catastrophic forgetting, as learned parameters are never modified after task training.

$$h_{0,\mathrm{i}} \;=\; f\left(W_{0,\mathrm{i}} \cdot h_{0,\mathrm{i}-1} + b_{0,\mathrm{i}}\right)$$

$$h_{k,\mathrm{i}} \;=\; f\left(W_{k,\mathrm{i}} \cdot h_{k,\mathrm{i}-1} + \sum_{\mathrm{j}<k} \left(U_{\mathrm{j,i}} \cdot h_{\mathrm{j,i}-1}\right) + b_{k,\mathrm{i}}\right)$$

Figure 5. The equations governing layer h_{i} in column k, where b is the bias term, W are the weight parameters of the blocks, and U are the weight parameters of the lateral connections.

When new columns are added, a set of lateral parameters $U_{\mathrm{j,i}}$ is added connecting column k block i to column j block $\mathrm{i}-1$ for all columns $\mathrm{j} < k$. Each of these laterals are treated as a combined layer with the parameters in the block itself. Activation function f is then applied to the entire block. With this system, new columns are capable of learning new knowledge in the W parameters and finetuning old knowledge in the U parameters, without ever changing the weights of previously learned columns.

Progressive neural networks prove to be a powerful model for many task sets. In *Progressive Neural Networks*, examples demonstrate that the system performs well in various reinforcement learning tasks including maze, Atari, and pong variant games. Further, papers including *Sim-to-Real Robot Learning from Pixels with Progressive Nets* [18] and *Progressive Neural Networks for Transfer Learning in Emotion Recognition* [19] demonstrate the breadth of application for this form of transfer learning. However, progressive networks are not without limitation — for example, the rate of parameter growth and the inefficient information density of new parameters. Another limitation is the necessity of task labels, even after training is complete — as stated in *Progressive Neural Networks*, "while progressive networks retain the ability to solve all K tasks at test time, choosing which column to use for inference requires knowledge of the task label".

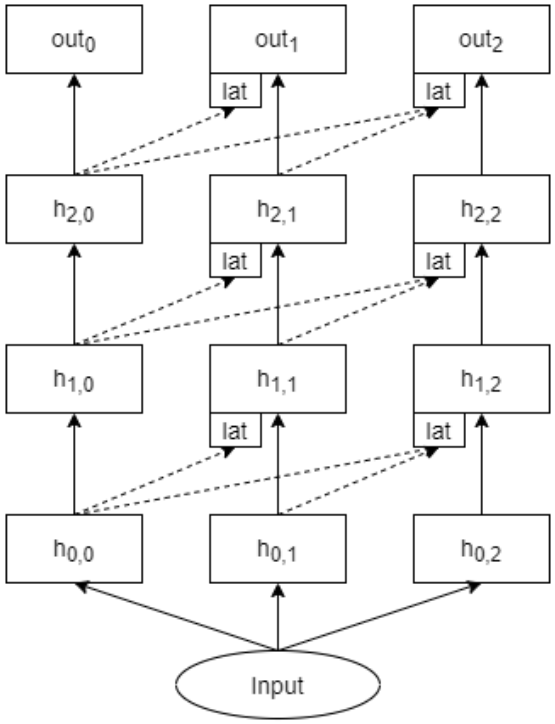


Figure 6. A 3 column progressive network. Trained columns are frozen and laterals are used for transfer learning.

Following the publication of *Progressive Neural Networks*, other methods have been tested to reduce catastrophic forgetting while allowing for efficient transfer learning. For example, the PNAS article *Overcoming catastrophic forgetting in neural networks* [2] established a

methodology that was able to reduce forgetting in finetuned neural networks by slowing down training on the most important parameters for a task, accomplished through a Bayesian approach introduced as elastic weight consolidation (EWC). The paper *REMIND Your Neural Network to Prevent Catastrophic Forgetting* [20] takes a different approach, using both frozen and plastic sections as well as a complex replay system that leverages efficient compressed representations of data from previous tasks.

Other methods take an approach more similar to that of progressive networks — adding capacity to the network with new connected extensions, such as *Lifelong Learning with Dynamically Expandable Networks* [21] in which instead of retraining the whole model like finetuning based approaches, or doing no retraining like progressive networks, dynamically expandable networks (DENs) selectively retrain and then dynamically expand the network. This approach succeeds in mitigating the inefficiencies of progressive networks, but it trades this boon for less protection against forgetting. Despite the success of these methods, none completely avoid catastrophic forgetting, instead focusing on lessening its effect.

3.3 Anomaly Detection & Autoencoders

Anomaly detection — also sometimes called novelty detection — is the task of identifying anomalies, outliers, or unusual data records among a class. This task is similar to the concept of one-class classification, in which ML systems must decide whether a data record is part of a single class or not. Techniques that have been used successfully to accomplish anomaly detection including one-class SVMs [22], one-class neural networks [23], and unsupervised methods such as K-means clustering. However, one of the most popular methods is the use of reconstructive autoencoders.

At their core, autoencoders are neural algorithms that perform a sort of compression and decompression, learning to concentrate input data into an abstract latent space, and then reconstruct it into a more desirable form. Three components make up the autoencoder: the encoder, latent vector, and decoder. The encoder takes as input a vector of size d_{in} and outputs a vector of size z where d_{in} is significantly larger than z. Its purpose is to learn a function transforming input into a latent representation, which will be stored in the latent vector. Likewise, the decoder learns to take in a vector of size z — data that

has been transformed by the encoder — and outputs a vector of size d_{out} where d_{out} is significantly larger than z. Essentially, the autoencoder uses the latent dimension along with self-supervised learning to gain knowledge of the data it is reconstructing. This nimble architecture has many uses, for example, by training an autoencoder to take in a grainy or noisy image and output a cleaned version, the network can learn to denoise all images similar to its training examples. Autoencoders are also ideal for encoding large data records into a smaller feature vector. By training the autoencoder to reconstruct high dimension data, the encoder section can later be used to transform data into a lower-dimensioned embedding, which a task network can use for various purposes. Autoencoders can even be improved with semi-supervised learning — gaining knowledge from unlabeled data through self-supervised learning and labeled data through supervised learning [24].

Anomaly detection autoencoders [25] operate under a simple premise: as information is restricted to the bandwidth of the latent dimension, the autoencoder must learn abstract features of the learned data. If the autoencoder is trained to reconstruct data from a certain class, it will perform poorly in reconstructing data sufficiently different from the class. The poor performance can be measured by a novelty criterion — usually a simple error function. Selecting an upper bound and lower bound of error, the autoencoder will be able to determine the unique status of a data record by yielding an error value outside the selected threshold.

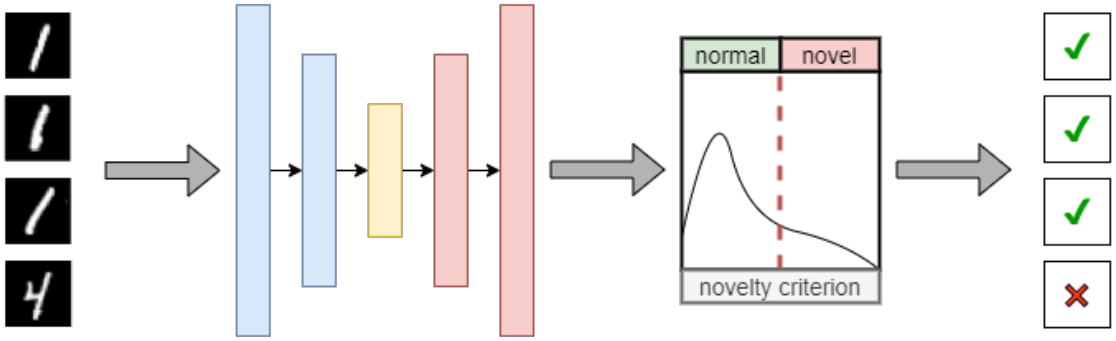


Figure 7. Diagram of an anomaly detection autoencoder learning to reconstruct handwritten '1' digits from the MNIST [26] dataset. The autoencoder reconstructs the data and checks against a threshold of the novelty criterion to classify the data as normal or novel.

The concept of reconstructive autoencoders as anomaly detectors is applied and expanded upon in many studies, including Robust Anomaly Detection in Images using Adversarial Autoencoders [27]. In this work, the authors present better anomaly detection results when using an adversarial autoencoder and utilizing Iterative Training Set Refinement (ITSR) to reduce the influence of anomalies that may be present in the training set. When applied to the MNIST and Fashion-MNIST datasets, the adversarial autoencoder with ITSR outperformed both a standard autoencoder and an adversarial autoencoder trained without ITSR. This anomaly detection technique can also be used to recognize and discriminate between handwriting styles — *Convolutional Autoencoder for Discriminating Handwriting Styles* [28] demonstrated this with a convolutional autoencoder applied to a custom dataset of handwritten text.

Autoencoders also proved their worth as textual anomaly detectors in *Spam review detection using LSTM autoencoder: an unsupervised approach* [29], in which the authors presented an LSTM autoencoder capable of accurately distinguishing between normal reviews and anomalous reviews likely to be spam. Using the reconstruction and novelty criterion method, the anomaly detector learned to classify a YouTube comment database with an F1 score of 0.99. These results are further evidence of the versatility and effectiveness of the anomaly detection autoencoder.

4 Task Detector

The solution to task recognition relies on one important assumption: the task must be in some way recognizable based only on input data from that task. For reinforcement learning tasks, this can be state data from the environment. For supervised or unsupervised tasks, this can instead be the entire data record — the task detector uses these inputs to learn abstract features of the task data through reconstruction. During test time, the task detector applies the abstract features in order to act as a scoring function, measuring the similarity of the data to a theoretical version of itself belonging to the task. The highest scoring function can then be selected as the likely candidate for the task label.

Upon initialization, the task detector is composed of just one anomaly detection autoencoder indexed to a progressive network's first column. The initial anomaly detector

can be trained on the same data as the progressive network column — though a separate pre-processing step may be necessary to achieve optimal results. The anomaly detectors are trained using self-supervised learning to deconstruct and reconstruct inputs through the latent vector. Once the column network has finished training, its parameters are frozen, as are the parameters of the anomaly detector.

As new tasks are learned, more autoencoders are added alongside progressive net columns. Each anomaly detector remains indexed to just one column net, and is only trained on data from the task associated with that column. Immunity to catastrophic forgetting is preserved because the anomaly detectors are only trained on data from a single task.

Algorithm: Training Task Detector with Progressive Net

Input: Progressive neural network P, task detector D, and task set T of size K.

Initialize networks P and D

for $k \leftarrow 0$ ***to*** K **do**

  $t \leftarrow T_k$;

  Add column k to P;

  Add autoencoder k to D;

  Train P_k on t;

  Train D_k to reconstruct inputs from t;

  Freeze P_k and D_k;

end

return P and D;

During test time, the anomaly detectors take on a similarity-measurement role. So long as a sufficiently low-capacity latent vector has been selected, the anomaly detector will do one of two things. If the unlabeled input belongs to the task associated with the anomaly detector it will reconstruct the data incorrectly — either transforming it into a false version of itself more similar to the associated task, or if the autoencoder has no generative capabilities, transforming it into incomprehensible garbage data. However, if the input does belong to the associated task, the autoencoder will reconstruct it more effectively. Therefore, the detector with the least error is most likely to be indexed to the correct task. The novelty score from

each anomaly detector can be calculated as the negative error — defined by the test time criterion — between the input and the reconstruction. Finally, the predicted column is selected as the $argmax$ of all the novelty scores. The task detector can be formally defined as:

$$\hat{k} = argmax_{\mathrm{i}\in\{0,\ldots,K\}}(score(x, d_{\mathrm{i}}(x)))$$

Where k_{hat} is the predicted task label, $score$ is the negation of the novelty criterion, and d_{i} is the anomaly detector at index i.

Algorithm: Testing Task Detector with Progressive Net

Input: Progressive neural network P, task detector D, and input data x.

$maxK \leftarrow null$;

$maxScore \leftarrow -infinity$;

for $k \leftarrow 0$ ***to*** K **do**

  $score \leftarrow -novelty\_criterion(D_k(x))$;

  if $score > maxScore$ **then**

    $maxK \leftarrow k$;

    $maxScore \leftarrow score$;

  end

end

return $P_{maxK}(x)$;

For some task sets, the accepted range of the novelty criterion is different between tasks. This can occur if the inputs of one task are harder to reconstruct than the others. While there are more sophisticated ways to remedy this, for the autoencoder task detector I chose to normalize the scores to a distribution learned during training. This simple measure allows the task detector to be much more dynamic in learning to recognize different tasks.

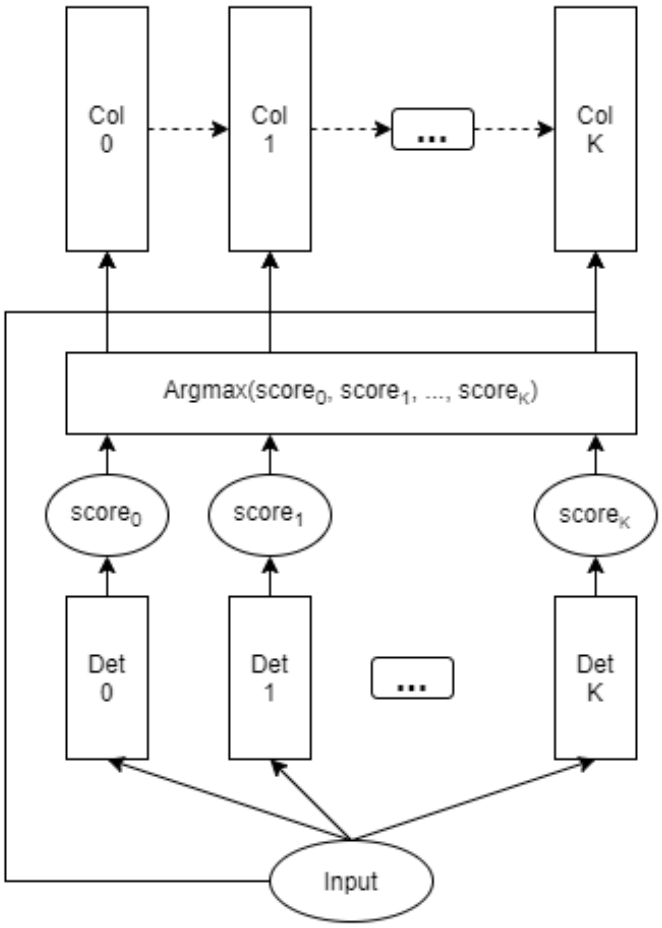


Figure 8. A diagram of the combined progressive system. The task detector makes up the K detectors and the argmax operation. It facilitates the use of the progressive neural network — composed of K laterally connected columns — located above it.

5 Experiment Methodology

To evaluate the efficacy of the task detector algorithm, I ran it through two difficult experiments. First, the task detector learned to recognize Atari games from simulated screen frames [30]. This experiment was selected to reflect its use in *Progressive Neural Networks*. It was also well-suited because the games could be recognized based only on screen frames. Nine games were selected: Breakout, Pong, Space Invaders, Ms Pacman, Assault, Asteroids, Boxing, Phoenix, and Alien. These games can be further classified by type of game (paddle games, ship games, maze games, and fighting games) or by color of screen assets (i.e.mostly black, mostly colorful). This distribution of games tests the task detector against similar

looking games and similarly colored games, ensuring that the networks are learning the color and the shape of the game.

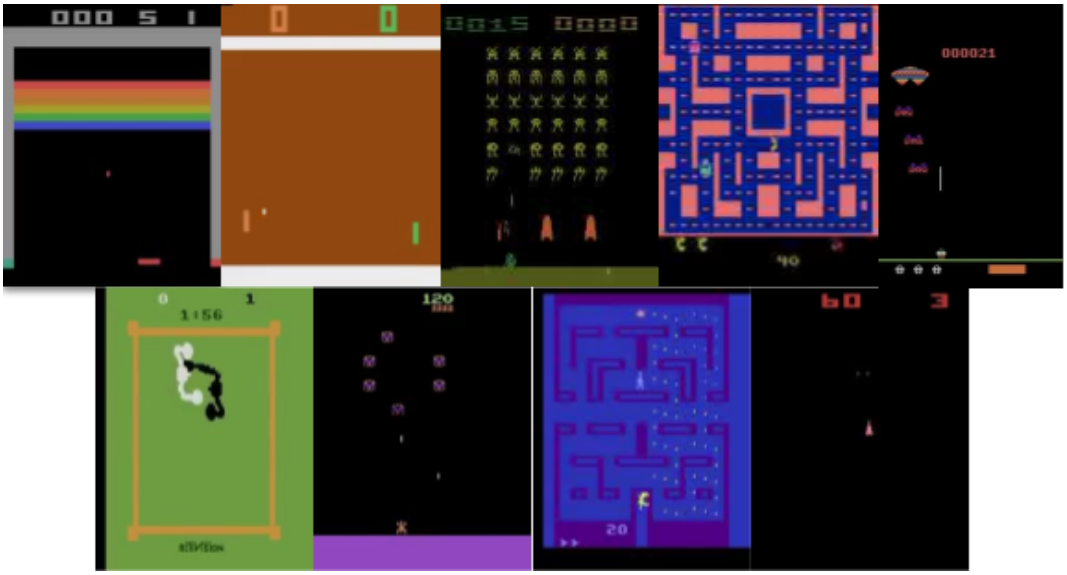


Figure 9. Atari games used in experiment 1.

Each anomaly detection autoencoder is trained one frame at a time alongside a random agent, used instead of a progressive system due to resource constraints and the knowledge that progressive neural networks have already been proven in this domain. The model architecture is detailed in Figure 10 below. Leaky ReLU with a 0.1 negative-value gradient was used as an internal activation function. As the frames were converted to a -1 to 1 range, a hyperbolic tangent function was used as the activation for the output. Several optimizers were tested, but RMSProp with a small learning rate and small weight decay was found to yield the best results.

Two different loss functions were needed to implement the task detector: a novelty criterion, and a training criterion. For the novelty criterion, I selected mean squared error (MSE) as it is a simple error function that punishes inconsistencies heavily. Several training criterions were tested. All yielded good reconstruction results, but the best results were achieved by implementing a novel loss function.

Kinetically-adjusted mean squared error ($KAMSE$) is a variant of MSE that adjusts the loss based on changed pixels between two frames. Given a frame y_0, a reconstructed frame $\hat{y_0}$, a subsequent frame y_1, and a kinetic weight κ:

$$KAMSE(y_0, \hat{y_0}, y_1) = MSE(y_0, \hat{y_0}) + \kappa * MSE(M_{y_0,y_1} * (y_0), M_{y_0,y_1} * (\hat{y_0}))$$

$$M_{y_0,y_1,\mathrm{i}} = \begin{Bmatrix} 0 & y_{0,\mathrm{i}} - y_{1,\mathrm{i}} = 0, \\ 1 & otherwise. \end{Bmatrix}$$

The intuition for this function is that pixels that change between frames are likely to be vital and differentiating in any Atari game. Usually, these correspond to objects of importance like the paddle and ball in paddle games or the space ship in ship games. Utilizing $KAMSE$ as the training criterion with a κ value of 10 greatly improved the reconstruction quality of the autoencoders and moderately improved the predictive power of the task detector as a whole. $KAMSE$ did not produce significant improvements as a novelty criterion, so was not selected for that role.

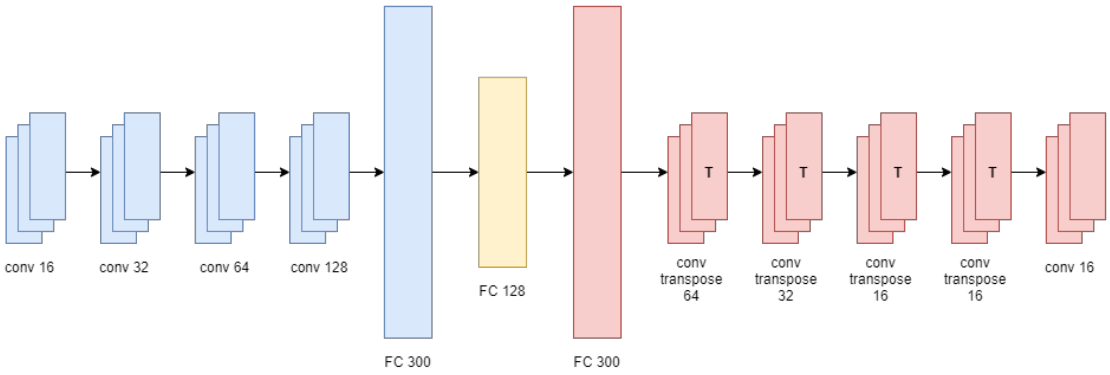


Figure 10. Architecture of an atari game anomaly detector. All convolutional and transposed convolutional layers have a kernel size of 3, a stride of 1, and a padding of 1. Leaky ReLU acts as the activation function for all hidden layers. Tanh acts as the activation for the output layer.

The second experiment is based on the exemplar code packaged with Doric, a free and open-source progressive neural networks library. Doric and several examples associated with it are introduced in the online article *Progressive Neural Networks: Explained & Implemented* [31]. The task detector was trained to recognize which of a set of image processing tasks needed to be applied on a dataset of human faces to uncover the original image. A subset of

the celebA dataset [32] was used. The tasks were implemented with a progressive network and the task detector was trained alongside it. Four image processing tasks were learned: simple reconstruction, denoiseing, colorizing, and inpainting. A pre-processing step was included for transforming each face image to fit the given task.

Figure 11. Demonstration of image tasks. From left to right: reconstruction, denoising, colorizing, and inpainting.

The purpose of this experiment was to test the task detector against a different dataset, and to train it alongside a progressive network. While the face image task set was pre-built into Doric, it had two properties that made it ideal for testing. Firstly, the differences between tasks were limited only to which pre-processing step was taken — adding noise, removing color, introducing a random walk of black pixels, or nothing. The same face images were used. This means that the individual anomaly detectors would need to reconstruct the images with some detail to properly encapsulate the uniqueness of the task. Secondly, because the task set is not particularly time-intensive or resource-intensive to train, it was perfect for testing the effect of batch size when running in test mode.

Anomaly detectors for the face image experiment use the same architecture as their Atari game counterparts except for the latent vectors, which are of size 64. During training, a batch size of 24 is used. As the autoencoders are reconstructing non-serial images in this experiment, KAMSE is set aside in favor of standard MSE for both the training criterion and novelty criterion. Several batch sizes were tested for detection. This setup necessarily assumes that all images within a batch belong to the same task.

6 Results

After training the Atari game task detector, it was tested on 501 game frames. The results are displayed in Figure 12 below. The average F1 score achieved was 0.9996. While these results seem quite good, they are to be expected because while it is difficult to reconstruct an Atari frame perfectly, it is only necessary that the correct detector has a closer reconstruction than all other detectors.

Atari Games Confusion Matrix											
		True Task									
		breakout	pong	space_invaders	ms_pacman	assault	asteroids	boxing	phoenix	alien	total
Predicted Task	breakout	501	0	0	0	0	0	0	0	0	*501*
	pong	0	499	0	0	0	0	0	0	0	*499*
	space_invaders	0	0	501	0	0	0	0	0	0	*501*
	ms_pacman	0	0	0	501	0	0	0	0	0	*501*
	assault	0	0	0	0	501	0	0	0	0	*501*
	asteroids	0	2	0	0	0	501	0	0	0	*503*
	boxing	0	0	0	0	0	0	501	0	0	*501*
	phoenix	0	0	0	0	0	0	0	501	0	*501*
	alien	0	0	0	0	0	0	0	0	501	*501*
	total	*501*	*501*	*501*	*501*	*501*	*501*	*501*	*501*	*501*	*4509*

Figure 12. Results from the Atari game experiment.

The few incorrect classifications are a sign of one of the limitations of the task detector. The Atari implementation of Pong has a strange quirk, where frames at the very start of an episode appear inverted in color. This behavior is demonstrated in Figure 13. In a sense, the anomaly detector is working correctly. As these frames are very rare within a pong game, it is not incorrect to label them as an anomaly. That said, these misclassified frames do indicate a failure of the anomaly detector to abstract shape away from color.

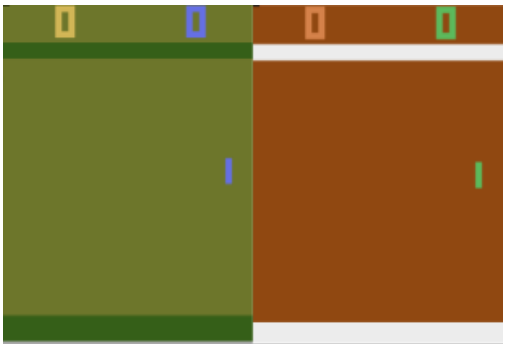


Figure 13. Pong graphical quirk. The left frame is only displayed once per episode at the immediate start of the game. The color scheme in the right image is used for the rest of the game.

As demonstrated in Figure 14, the face image task detector also performed quite well. An average F1 score of 0.9992 was achieved, with the only incorrect results coming from reconstruction being misclassified as inpainting.

Face Image Tasks Confusion Matrix with Batch_Size = 8						
		True Task				
Predicted Task		reconstruction	denoising	colorizing	inpainting	total
	reconstruction	4984	0	0	0	*4984*
	denoising	0	5000	0	0	*5000*
	colorizing	0	0	5000	0	*5000*
	inpainting	16	0	0	5000	*5016*
	total	*5000*	*5000*	*5000*	*5000*	*20000*

Figure 14. Results from the face image tasks experiment.

This experiment also succeeded in showing a relationship between detection batch size and task recognition F1 score. The detector was tested with four batch sizes: 1, 4, 8, and 16. The F1 results are plotted in Figure 15. While the utility of increased batch sizes decreases quickly, having more data to reconstruct before calculating an *argmax* can lead to better results and adds resilience against outlier inputs.

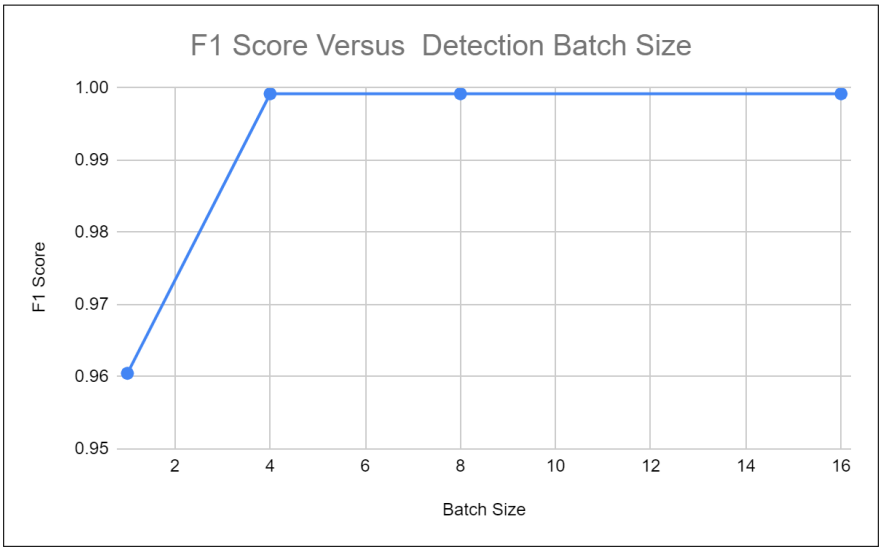


Figure 15. F1 score versus batch size from the face image tasks experiment.

7 Discussion

The results of the two experiments demonstrate that the autoencoder task detector is capable of recognizing tasks distinguished by input patterns. The Atari experiment especially showed the task detector's ability to recognize many tasks from a set, even when some tasks had similar appearances. In both tests, the detector was able to classify the tasks with an F1 score above 0.999, and in the vast majority of cases, the task detector selected the task correctly, meaning a progressive system attached to it would be capable of interacting with the task using the correct sub-network.

Although these results are encouraging, the task recognizer has a number of severe limitations. First and foremost, it relies on the assumption that the task is recognizable from only its input data. An example of a task breaking this assumption would be a version of Pong with inverted controls. The frames of this game would look identical to that of standard Pong, though the progressive system may require a different sub-network to play it effectively. However, any of these tasks could be differentiated with only minor adjustments to the algorithm — adding recurrence or even attempting to reconstruct the next state from the current state and the action taken in the case of inverted pong.

Another limitation is a similar problem to that of the progressive neural network: the task detector described here is often memory intensive. Alongside an already cumbersome progressive network, the whole system is likely to have a much greater memory footprint than is necessary to solve the tasks. It can also increase the training time of the system, as it is essentially training two separate networks at the same time.

Despite its limitations, the task detector also has many strengths. As it uses self-supervised learning, it will rarely require as much data to train as its task network — a comparison that is all the more stark when applied to reinforcement learning tasks. It is also likely to be finished training well before the task network on any sufficiently difficult task, as reconstruction of inputs is a relatively simple job. Further, because the task model and task detector do not interact during training, they can easily be trained in parallel.

In designing and testing the task detector, there were many variants I was not able to test; some of these untested modifications are included here in brief as potential avenues

of future research. Of these, perhaps the most interesting — yet least likely to succeed — is a progressive task detector. The autoencoders that make up the task detector could be linked together with lateral connections and treated as a second progressive network. The advantage of this would be the possibility of transfer learning the reconstruction of inputs, making the training stage of the task detector faster. This modification is unlikely to work because by using many of the same parameters, different anomaly detectors would likely retain some reconstruction ability from previously learned tasks, meaning the correct anomaly detector would need to be even better at reconstruction to compensate despite having fewer trained parameters to draw upon. A progressive task detector would also have even more severe memory requirements. Limited testing supported these theoretical shortcomings, but perhaps a detector with sparser or strategically placed lateral connections would yield better results.

Another variant of the task detector is one implemented with a variational autoencoder (VAE) [33] or an adversarial autoencoder (AAE) [34]. These architectures include desirable traits that could improve the accuracy of input reconstruction. By modeling the distribution of inputs within the latent space, variational autoencoders possess limited generative capabilities. As an anomaly detector, a VAE would be capable of reconstruction with much finer detail, distinguishing between tasks that are similar but for a few important distinctions. Adversarial autoencoders also have generative capabilities, but in these networks they stem from the use of two loss functions: one acting as a simple reconstruction criterion, and another as an adversarial criterion that "matches the aggregated posterior distribution of the latent representation of the autoencoder to an arbitrary prior distribution" [34]. A more homogeneous set of tasks would be needed to experiment with these architectures conclusively.

As stated in Progressive Neural Networks, "Continual learning, the ability to accumulate and transfer knowledge to new domains, is a core characteristic of intelligent beings. Progressive neural networks are a stepping stone towards continual learning" [3]. By expanding on the test time autonomy of progressive networks, allowing the system to identify the necessary column to run while maintaining its immunity to catastrophic forgetting, the autoencoder task detector succeeds in being another stepping stone towards this worthy goal.

8 Code Repository

The code used in this study can be found at www.github.com/arcosin/Task-Detector. The Doric library can be found at www.github.com/arcosin/Doric.

9 Acknowledgments

Thank you to Dr. Gustavo Rodriguez-Rivera for guidance and academic support, to Case Wright for assistance with autoencoders and KAMSE loss, and to David Stucker for editing.

Bibliography

[1] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[2] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[3] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks, 2016.

[4] Yann LeCun. Self-supervised learning: could machines learn like humans? École polytechnique fédérale de Lausanne, 2018.

[5] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[6] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[7] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[8] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[10] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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

[13] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[14] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[15] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[16] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[17] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[18] Andrei A. Rusu, Mel Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets, 2016.

[19] John Gideon, Soheil Khorram, Zakaria Aldeneh, Dimitrios Dimitriadis, and Emily Mower Provost. Progressive neural networks for transfer learning in emotion recognition, 2017.

[20] Tyler L Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. Remind your neural network to prevent catastrophic forgetting. *arXiv preprint arXiv:1910.02509*, 2019.

[21] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.

[22] Bernhard Schölkopf, Robert Williamson, Alex Smola, John Shawe-Taylor, and John Platt. Support vector method for novelty detection. In *NIPS'99: Proceedings of the 12th International Conference on Neural Information Processing Systems*, volume 12, pages 582–588, 01 1999.

[23] Raghavendra Chalapathy, Aditya Krishna Menon, and Sanjay Chawla. Anomaly detection using one-class neural networks, 2018.

[24] Anupriya Gogna and Angshul Majumdar. Semi supervised autoencoder. In *Proceedings of the 23rd International Conference on Neural Information Processing - Volume 9948*, page 82–89, Berlin, Heidelberg, 2016. Springer-Verlag.

[25] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, pages 4–11, 2014.

[26] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010.

[27] Laura Beggel, Michael Pfeiffer, and Bernd Bischl. Robust anomaly detection in images using adversarial autoencoders, 2019.

[28] Sanae Boutarfass and Bernard Besserer. Convolutional autoencoder for discriminating handwriting styles. In *2019 8th European Workshop on Visual Information Processing (EUVIP)*, pages 199–204. IEEE, 2019.

[29] Sunil Saumya, Jyoti Prakash Singh, et al. Spam review detection using lstm autoencoder: an unsupervised approach. *Electronic Commerce Research*, pages 1–21, 2020.

[30] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. cite arxiv:1606.01540.

[31] Maxwell J. Jacobson. Progressive neural networks: Explained & implemented, Jun 2020.

[32] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[33] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013.

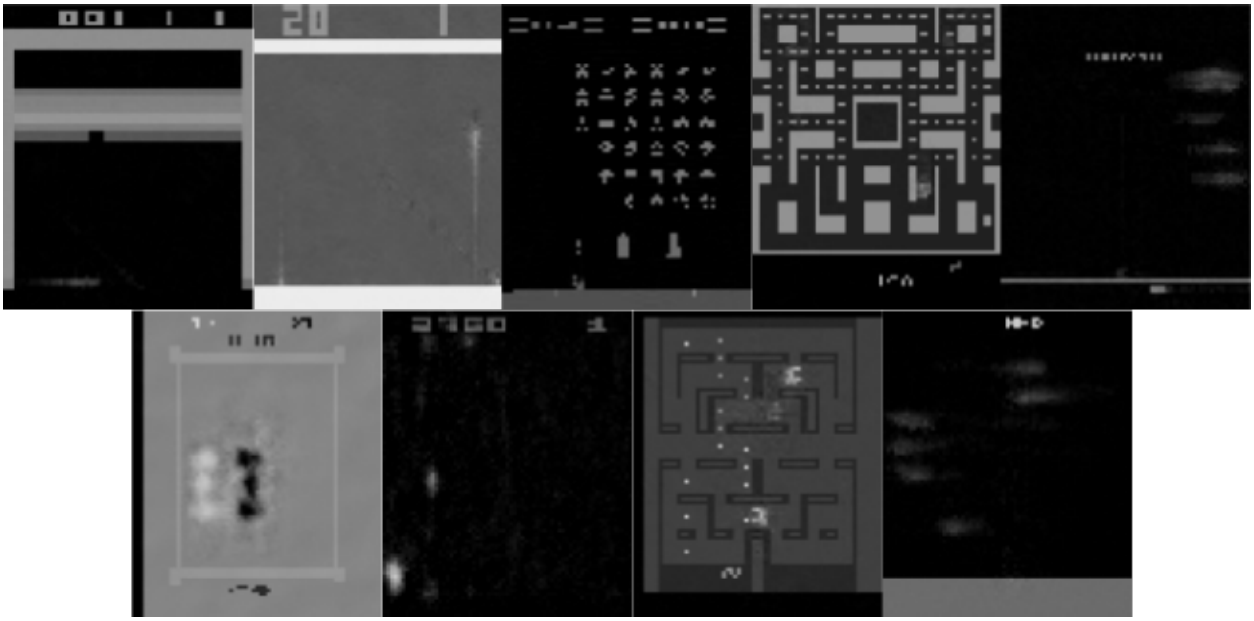
[34] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders, 2016.

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Supplementary Material

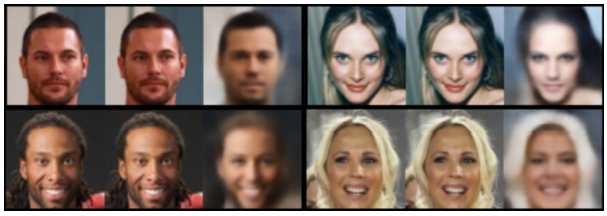
A Anomaly Detector Reconstructions of Atari Games

Reconstructions created by anomaly detectors trained on Atari games.

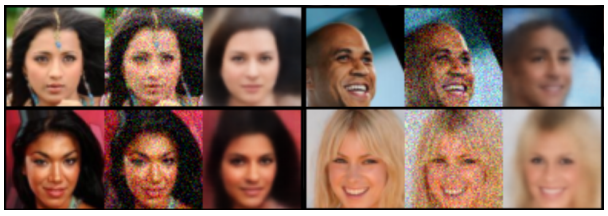


B Face Image Tasks Select Outputs

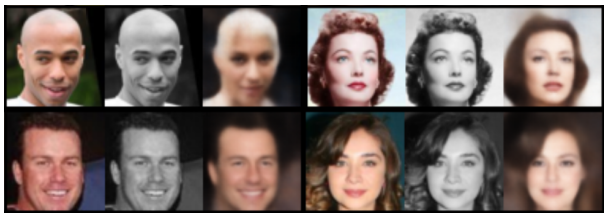
Select outputs from face image task progressive network with task detector. Each item displays the original image, the processed image, and the reconstructed image respectively.



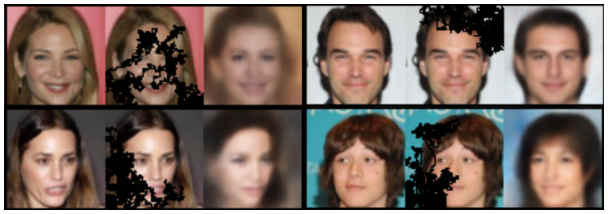
(a) **Reconstruction**



(b) **De-noising**



(a) **Colorizing**



(b) **Inpainting**