Evaluating the Robustness of Cloud Auto-AI Models against Adversarial Attacks

Aashka Trivedi (aht323) and Anya Trivedi (aht324)

Abstract

Neural Networks have shown remarkable performance in a wide variety of tasks. Their ubiquity and performance, along with the availability of large amounts of training data has led to the rise in *Auto-AI* models, whose training is fully automated by cloud providers, without any need of ML expertise. However, while neural networks are powerful, they tend to be easily fooled by inputs with slight perturbations. Such *adversarial examples* often have distortions so minute that they are undetectable to the human eye, but can catastrophically alter the prediction accuracy of a machine learning model. This project aims to evaluate the robustness of Cloud Auto-AI models against Adversarial Attacks on Image Classification Tasks. Specifically, we train AutoAI models to recognize handwritten digits of the MNIST dataset on two cloud providers- IBM Cloud and Google Cloud Platform, and test their robustness to adversarial examples. We find that while both IBM's and Google's AutoAI models perform extremely well on test data (achieving around 99% accuracy), they are highly susceptible to adversarial attacks- recognizing only about 10% of the adversarial examples correctly. We also evaluate the potency of *defensive distillation* techniques in decreasing the effect of adversarial examples, and find that while they improve the robustness to some extent, distilled models are still highly vulnerable to adversarial attacks.

1 Introduction

An active domain of research is the study of the robustness of Machine Learning models (in particular, Neural Networks) against *adversarial* examples [1, 2, 3, 4, 5, 6]. In all cases, inputs are generated by applying small but effective perturbations, causing the classifier to either output an adversarial "target" class (Targeted attacks), or simply output a class different from the real correct class (Untargeted attacks). These attacks have been carried out in a variety of domains, ranging from Image classification [2] to Automatic Speech Recognition systems [5]. However, most studies dealing with the robustness of neural networks in adversarial domains are done using models that are either pre-trained on large datasets or trained from scratch, both using local GPUs. An interesting analysis would be to study how popular cloud-based Software as a Service (SaaS) platforms compare in terms of robustness towards adversarial examples, considering the growing popularity of these platforms in both industry and academia.

This project aims to compare the performance of off-the-shelf Cloud Auto-AI services when exposed to adversarial examples. Various cloud providers such as IBM cloud [7] or Google Cloud Platorm (GCP)[8] house powerful "Auto-AI" platforms that allow users to build world-class Machine Learning models without much ML expertise. These SaaS house pre-trained models that can both be customised by users, or left as is, but their training (and sometimes, deployment) is fully automated on the cloud. We aim to compare the robustness of IBM Cloud's Auto-AI service [9] and Google Cloud's Auto-ML service [10], which are both high-performance tools that can be used to analyse data and automatically build candidate model pipelines customized for predictive modeling problems. In particular, we aim to see how robust these Auto-AI models are towards adversarial examples in the Image Classification domain, and perform a comparative study between the two cloud platforms' models.

We train AutoAI models to classify handwritten digits of the MNIST dataset [11], using default parameters, to gauge the performance of cloud generated AutoAI models without any other type of additional hyperparameter tuning. We then generate Adversarial Examples from the MNIST Test set using the Fast Gradient Sign Attack [1], and evaluate the robustness of the AutoAI models to these examples. We also attempt to use Defensive Distillation [12] as a method to improve the robustness of models. Specifically, we use the cloud AutoAI models as teachers to distil into a LeNet-like student (in a black-box fashion), and critically evaluate whether defensively distilled models are robust to adversarial attacks, and if so- to what extent.

The code for this project can be found at the Github Repository: `https://github.com/aashka-trivedi/cloud-autoai-adversarial-robustness`.

2 Related Work

We begin by discussing a brief literature review of the three main components of this project- AutoAI models, Adversarial Attacks and Defensive Distillation.

Automated Artificial Intelligence and Machine Learning Automated Artificial Intelligence and Machine Learning automates the entire AI Lifecycle- including data preparation, model selection, training, evaluation and deployment [13]. This allows end users to use artificial intelligence pipelines without need of expertise or experience in the domain. Various cloud providers, such as IBM Cloud, Google Cloud Platform and Amazon Web Services have their own versions of AutoAI services, and allow creating machine learning models for application in the vision, speech and text domain. IBM Cloud's AutoAI [9] and Google Cloud Platform's AutoML [10] can be used to generate high-performance models trained on a variety of tasks, and optimized with different evaluation metrics, and have shown to give high performance in various domains. Typically, these services train and evaluate multiple models that use different algorithms or optimization methods, and evaluate these models on a user-specified metric. The best performing model is then chosen, which can be deployed directly by the user, or in some cases, further tested upon.

Adversarial Attacks Most Neural Network models achieve high performance on training and test data, but can easily be fooled by inputs with slight modifications, or perturbations. These types of examples, which have been intentionally distorted in order to be misclassified by the model, are known as Adversarial Examples [1, 2] and the corresponding "attacks" are known as Adversarial Attacks.

Adversarial examples are of two types [1, 2] - targeted adversarial attacks, which aim to cause the model to give a specific incorrect label to the adversarial example, and un-targeted adversarial attacks, which is successful as long as the model outputs a label that does not match the correct label. Targeted attacks are strictly harder than untargeted ones, which is why many prevalent attacks are untargeted, while research is done to defend targeted ones [2]. Moreover, attacks can either be whitebox attacks or blackbox attacks. Whitebox attacks are those in which the adversary is given complete access to the model, and has information on the architecture, outputs, and other details of the model. Blackbox attacks are those in which the adversary knows only the inputs and the corresponding outputs (confidence scores) of the model [14]. Blackbox attacks are harder to conduct, and less potent, so whitebox attacks are more prevalent. This is made possible by the *transferrability property* of adversarial examples- an example that is adversarial to one model is also adversarial to another. A common strategy is to train a *substitute model*, which is known to the adversary, and use that to generate whitebox adversarial examples [1, 14]. This is the strategy used in this work as well.

There are many recent works detailing the methods of generating adversarial examples, where the prominent methods are the Fast Gradient Sign Method [1], L-BFGS [15], Jacobian-based Saliency Map Attack (JSMA) [3], DeepFool [4], and Carlini & Wagner (C&W) Attack [2]. This work uses the Fast Gradient Sign Method Attack, which is easy, fast, and commonly used by adversaries. This method uses gradients to alter the image such that the loss (after forward propagation) is maximised [1]. This alteration is done by adding noise in the direction of the gradient, and the amount of this noise is controlled by a hyperparameter, *epsilon*. This is a white box attack- which needs the model in order to obtain the gradients. Thus, this attack is usually performed on a substitute model, and the generated images are then used on the target model.

Defensive Distillation Defensive Distillation is a method to add flexibility to the learning process of a Neural Network, making the model more robust to adversarial examples. The idea here is that a "student" model mimics how the "teacher" model learns data [16], by learning to predict the soft logit labels (classification probabilities) of the teacher model. Using these soft labels help encode the differences in the labels, which prevents overfitting, and can be used to increase the robustness of the neural network [12].

While it has been shown that defensive distillation helps reduce the effects of adversarial attacks, recent works show that this method is not as efficient against all attacks [2, 17]. Carlini and Wagner [17] show that by slightly modifying the distillation process, we can reduce the efficiency of defensive distillation. This work experiments with the proposed method [17] to see if defensive distillation still remains effective in increasing the robustness of cloud AutoAI models.

3 AutoAI Models on the Cloud

Various cloud providers have their own form of AutoAI services, relying on different algorithms, hyperparameter tuning methods, and evaluation criteria. The main goal of these services is to provide the user with the best model for the given data and task, with minimal ML expertise required by the user. In this section, we describe the process of training AutoAI models on the MNIST dataset on IBM Cloud and Google Cloud Platform, and then compare their ease to build, and performance.

3.1 IBM Cloud

To train an AutoAI model on MNIST on IBM Cloud, a comma seperated values (csv) file consisting of the training data and the label is provided. For the same, we converted each 28x28 pixel image of the MNIST dataset into 784 dimensional array, to be used as the training input. To fairly compare the AutoAI models in both cloud platforms, the default settings were used to build the AutoAI experiment.

IBM's AutoAI experiment evaluates 3 classification algorithms, and 4 configurations of feature selection and hyperparamter optimization for each algorithm, resulting in 12 experiment model pipelines. Each model is evaluated trained on 90% of the training data, and is validated on 10% of a 10-fold cross validation dataset. Each pipeline took about 10 minutes to train on average, and the whole experiment took less than 2 hours to complete. The experiment selected a Light Gradient Boost Classifier, Gradient Boost Classifier and an XGBoost Classifer as the candidate algorithms. Each algorithm had 4 corresponding pipelines with different optimization configuration, one with no enhancements, one with "Hyperparamer Optimization 1" (HPO-1), one with HPO-1 and "Feature Engineering" (FE) and one with HPO-1, FE and "Hyperparamer Optimization 2" (HPO-2). A screenshot of the progress map has been shown in Figure 1.

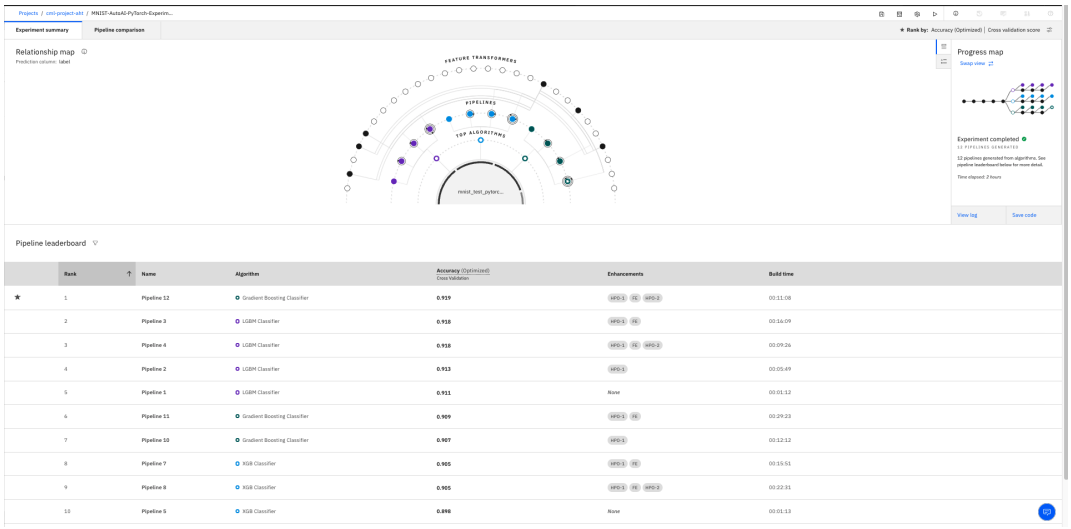

Figure 1: IBM Cloud AutoAI Model Experiments for MNIST

The default evaluation criteria of accuracy was used to choose the model, but IBM Cloud allows us to compare all pipelines through evaluation metrics like accuracy, loss, precision, recall and F1-Score. The pipeline comparison for all experiments have been shown in Figure 2.

The best selected model was Pipeline 12 - a Gradient Boosting Classifer with HPO-1 FE and HPO-2, which achieved a 91.9% cross validation accuracy. The AutoAI User Interface allows for a thorough evaluation of the selected model, with charts showing the F1-Threshold, Precision Threshold and Recall Threshold, along with the Precision-Recall curves, and ROC Curves for all holdout validation sets. While the table of the evaluation metrics for Pipeline 12 have been shown in Table 1, all other plots have been added to Appendix A.

3.2 Google Cloud

In order to maintain consistency in training for both the IBM Cloud and GCP models, the same CSV file of the MNIST data is used on both platforms. For the same, we converted each 28x28 pixel image of the MNIST dataset

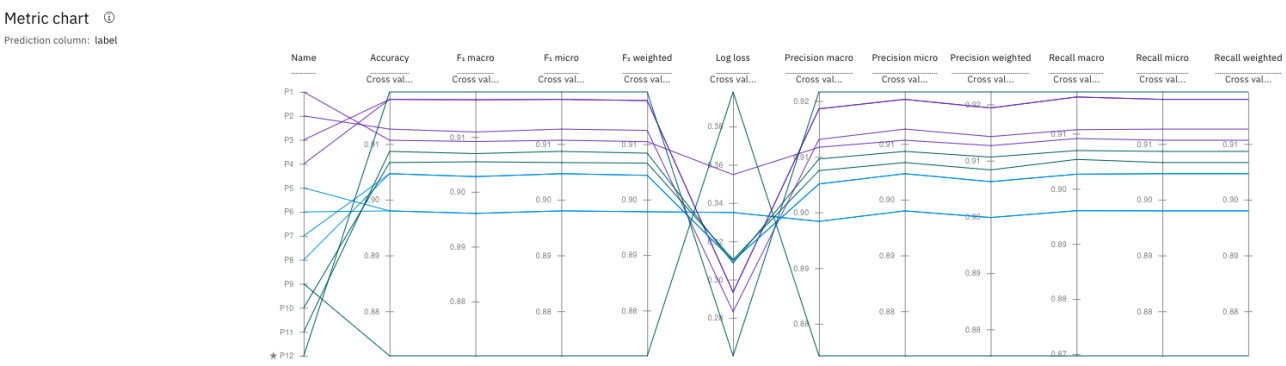

Figure 2: IBM Cloud AutoAI Model Pipeline Comparision

Measures	Holdout Score	Cross Validation Score
Precision Macro	0.948	0.922
Accuracy	0.946	0.919
Recall Macro	0.944	0.918
Weighted Precision	0.948	0.922
F1 Macro	0.945	0.918
Weighted F1 Measure	0.946	0.919
Weighted Recall	0.946	0.919
Log Loss	0.140	0.260

Table 1: IBM Cloud Best AutoAI Pipeline Model Evaluation

into 784 dimensional array, to be used as the training input. To fairly compare the AutoAI models in both cloud platforms, the default settings were used to build the AutoAI experiment.

GCP's AutoML platform is divided into products based on the ML Task, such as AutoML Vision (for image classification) and AutoML Text (for Natural Language Processing Tasks). On a deeper level, these services are also divided based on the *type* of input file expected. For example, AutoML's Vision *only* takes complete images, and not CSV files. In order to use the same CSV file used for training in IBM Cloud, the service used was AutoML Tables- an ML Service specifically for training on structured CSV Files.

GCP's AutoML experiment evaluates multi-class classification algorithms, with multiple hyperparameter settings. Unfortunately, the downside of GCP AutoML is that it is a completely black-boxed model, where there is no indication on which model is used, or what the best hyperparameters selected are. Each model is evaluated trained on 90% of the training data, and is validated on 10% of a 10-fold cross validation dataset.

Measures	Scores
Accuracy	0.986
Precision	0.989
Recall	0.981
F1 Score	0.985
AUC PR	0.997
AUC ROC	0.999
Log Loss	0.057

Table 2: GCP AutoML Model Evaluation

The models are evaluated by minimising Log Loss- the only available optimization metric for multi-class classification. The UI for AutoML tables provides a variety of metrics that the model can be evaluated on, such as precision, recall and F1 scores with variable thresholds, and the results are summarized in Table 2. The UI view is in Appendix B. The obtained confusion matrix is shown in Figure 3.

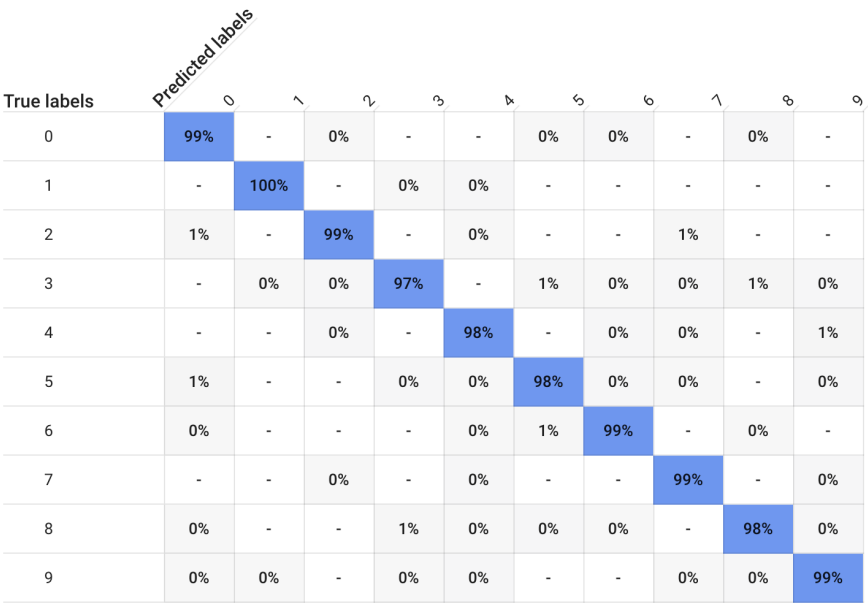

Figure 3: GCP AutoML Model Confusion Matrix

3.3 Comparison

Below is an in-depth comparison between IBM Cloud's AutoAI and GCP's AutoML platform.

Input IBM Cloud only supports CSV Files for all AutoAI jobs. This is a clear disadvantage compared to GCP's AutoML, where different ML services are catered to different input types. GCP uses Tables for CSV files, Vision for images, and Text for textual data like paragraphs. It is easy to upload the data either directly to the Service UI or access them via GCP Storage buckets, which provides the added convenience of accessing them across multiple platforms.

Training Platforms and Ease of Access A clear advantage of IBM Cloud's training is the ease to convert experiments and platforms into Jupyter Notebooks (which is done automatically), and define the model using an sklearn-based model. Unfortunately, GCP's Table platform is catered to non-ML experts, and it's models are not very customizeable. The most common way to train on Tables is using the UI, and tutorials using Jupyter notebooks practically do not exist. However, it is easy to link GCP project keys and buckets to a Notebook via the GCP Console API, and this allows training via Notebook on Google Colab or GCP's AI Platform. The advantage of of using AI Platform is that authentication and bucket access is easy, and require no additional CLI tools. IBM Cloud tends to require multiple authentication attempts even if the Notebook is housed in the same Watson Studio projects, however it is easy to upload files and use them in the Notebook, and automatically adds code to access the files.

Training Time and Flexibility IBM Cloud AutoAI models allow more control in terms of hyperparameter definition and training. GCP's AutoML models have limited optimization metrics (for example, minimizing log loss is the only metric for multi-class classification). Further, GCP takes a much larger training time compared to IBM Cloud, and an additional lag for deployment. However, the increase training time can be attributed to GCP's very intensive Hyperparameter tuning, as GCP trains over a large number of models with hyperparameter settings. IBM Cloud's AutoAI trains a specific number of models, each with a specific optimization strategy. This takes relatively less time to train depending on the optimization strategy, but it notably quick.

Resources IBM Cloud's main disadvantage is the fact that it requires all inputs to be in the form of CSV files. However, tutorials and online resources are well kept and up-to-date. Using GCP's UI is fairly straighforawrd, but client API's to access resources via Notebook change haphazardly, resulting in outdated resources.

Testing the Deployed Model Both IBM Cloud and GCP allow for the deployment of the configured model. However, to test the deployment on the cloud's UI, IBM Cloud only allows the use of a JSON-formatted input. This is a disadvantage compared to GCP, which allows for the use of more commonly available CSV-formatted inputs to be evaluated on the cloud.

Deployment and Test Times IBM Cloud takes less time to train, but takes a considerable time to deploy and test. GCP however, deploys and tests in better time, albeit with a much larger training time. The exact deployment durations for the GCP model is in Appendix B.

Performance of Models IBM Cloud's AutoAI model trained on the MNIST dataset ultimately achieves a 96.5% on the MNIST Validation set and a whopping 99.6% accuracy on the MNIST Test set (these results can be seen in Table 3). The GCP AutoML model performs well on the MNIST dataset as well, it achieves a 97.3% accuracy on the MNIST Validation set, and a 98.6% accuracy on the MNIST Test set (these results can be seen in Table 4), which is slightly lower than IBM Cloud's model.

4 Adversarial Example Generation

Adversarial Examples use minimal perturbations to the input in order to cause a mis-classification by the target model. In this project we generate *untargetted adversarial examples* using the **Fast Gradient Sign Method** [1].The FGSM attack uses a model's gradients by adjusting the input data in order to maximise the loss. We have been motivated to use this attack as it is one of the most commonly used attacks, used to gauge robustness, and generating adversarial examples using this algorithm is easy and fast.

The FGSM attack is a *white box attack*, meaning that it requires knowledge about the neural network in order to produce adversarial examples. Because AutoAI models are often black box in the sense that we do not have access to their architectural details, it is not technically possible to use this algorithm on the AutoAI model itself. Instead, we exploit the **transferability property** of adversarial examples, which states that *examples that are adversarial on one model are also adversarial on another model* [1, 2] . We therefore perform the FGSM attack on a substitute LeNet Model trained on the MNIST dataset and available as an open source model on PyTorch's Github [18], and then use the adversarial examples generated on LeNet to test the robustness of AutoAI models. The success of the attack proves that while a white box adversarial attack cannot be conducted on AutoAI models themselves, it is still easy to find adversarial examples to attack them using substitute models. Moreover, whitebox attacks are easier than blackbox ones, and are thus the more prevalent attack method.

A major goal of producing adversarial examples is that the distortion to the image should be as minimal as possible, as while more distortion may cause a misclassification with a higher probability, ot is also more apparent to the human eye. The distortion of the images produced by the FGSM algorithm can be controlled by the hyperparameter *Epsilon*. In this project, we experiment with epsilon values of 0.10, 0.15, 0.20, 0.25 and 0.30, and run the FGSM algorithm on the entire MNIST Test set. Some of the generated adversarial images are shown in Figure 4, along with the true labels, and the predicted labels from the LeNet, IBM AutoAI model and Google AutoAI model.

5 Evaluation: Adversarial Robustness

We evaluate the robustness of AutoAI models against adversarial examples generated on the MNIST test set using the FGSM Algorithm by computing the accuracy of the predictions when the model is presented with the perturbed image inputs. As stated before, we experiment with five values of epsilon (corresponding to the degree of perturbations), an epsilon of 0.10, 0.15, 0.20, 0.25, 0.3.

One argument could be whether the decrease in performance is because of the success of the attack or the performance (specifically, overfitting) of the model. This can be tested by comparing the accuracy of the model on an unseen test set, and the adversarial examples generated on *the same test set*. A model with a high performance on the test set by definition does not overfit, and if it has a poor performance on the adversarial examples, then the attack can be considered to be successful.

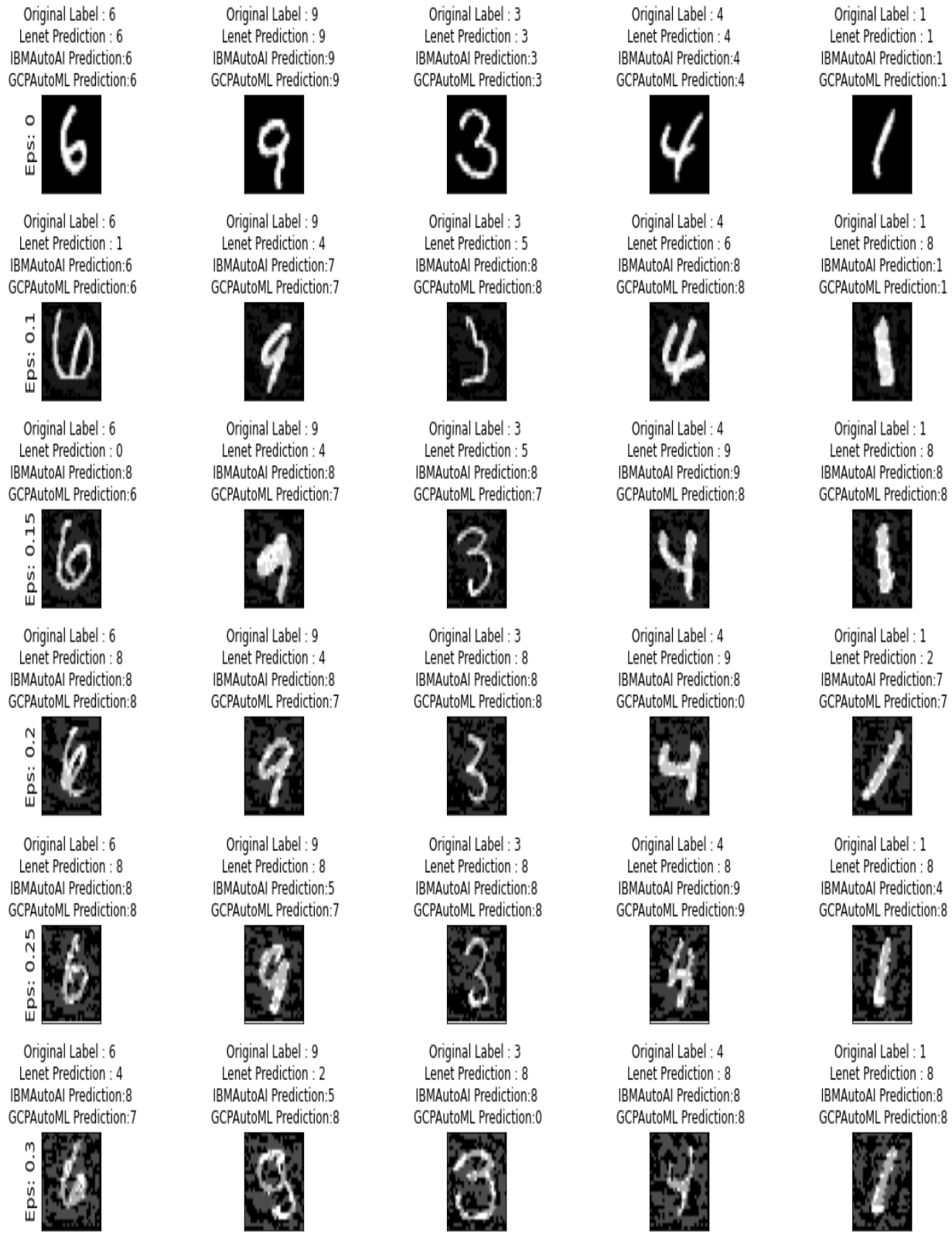

Figure 4: Adversarial Example Generation with different values of Epsilon. The title shows the true label, along with the label predicted by LeNet, IBM AutoAI and GCP AutoML models. Eps=0 refers to the un-distorted test image.

5.1 IBM Cloud

The results for the robustness analysis of the IBM Cloud generated AutoAI model is shown in Table 3. As shown, the AutoAI model performs extremely well on the validation and the test set, achieving a 99% accuracy on the test set. This shows that the AutoAI pipeline provided by IBM Cloud results in powerful models, with an ability to achieve high accuracies. However, these models are not at all robust to adversarial attacks, with minute perturbations dropping the accuracy to only 11%.

Perturbation	Metric	Score
None	Validation Accuracy	96.534
None	Test Accuracy	99.643
Epsilon = 0.10	Test Accuracy	**11.002**
Epsilon = 0.15	Test Accuracy	**10.489**
Epsilon = 0.20	Test Accuracy	**10.001**
Epsilon = 0.25	Test Accuracy	**9.441**
Epsilon = 0.30	Test Accuracy	**8.903**

Table 3: Evaluating the robustness of the IBM Cloud Auto AI model to Adversarial Attacks

As shown in Figure 4, the adversarial images are mostly identical to the original image, with some slight noise- this shows that while the AutoAI model does not overfit to the training data, it definitely is not robust to adversarial attacks or even the addition of noise. A further cause of concern is that the adversarial attacks were generated using the transferability property, and a simple attack algorithm- and much more catastrophic results may be achieved using a more powerful atatck algorithm.

More perturbations lead to lesser accuracy of the model, but more prominent changes to the source image. This is shown in Figure 5, where we can see the degrading performance with increasing epsilon.

5.2 Google Cloud

The results for the robustness analysis of the GCP AutoML model is shown in Table 4. As shown, the AutoAML model performs extremely well on the validation and the test set, achieving a 98% accuracy on the test set. While the model details themselves are hidden, thus making the architecture itself a variable black box, the ability of GCP's AutoAI models to learn classifications without overfitting is commendable. However, these models are not at all robust to adversarial attacks. We see a drop in performance as measured by accuracy with minute perturbations dropping the accuracy to only 21%.

Perturbation	Metric	Score
None	Validation Accuracy	97.316
None	Test Accuracy	98.632
Epsilon = 0.10	Test Accuracy	**21.806**
Epsilon = 0.15	Test Accuracy	**18.073**
Epsilon = 0.20	Test Accuracy	**15.062**
Epsilon = 0.25	Test Accuracy	**11.290**
Epsilon = 0.30	Test Accuracy	**10.812**

Table 4: Evaluating the robustness of the GCP AutoML model to Adversarial Attacks

As shown in Figure 4, the adversarial images are mostly identical to the original image, with some slight noise. The large drop in accuracies from the test set to the adversarial set shows that the GCP model does not overfit, but is has extremely poor robustness to adversaries. The trend observed in the accuracy of the GCP model with varying epsilon is depicted in Figure 5.

5.3 Discussion

As shown in Figure 4, the adversarial images are mostly identical to the original image, with some slight noise- this shows that while both the AutoAI models do not overfit to the training data, neither are robust to adversarial attacks or even the addition of noise. In fact, with high values of epsilon, the accuracy drops to about 10% , which is no better than random guessing between the possible output labels. A further cause of concern is that the

adversarial attacks were generated using the transferability property, and a relatively simple attack algorithm- and much more catastrophic results may be achieved using a more powerful attack algorithm , like, for example, the C&W algorithm [2].

It is interesting to note however that while GCP Models perform slightly worse when training and testing on the normal MNIST data as compared to IBM Cloud, they perform significantly better in their robustness to adversaries. Their relative performance with different values of epsilon are shown in Figure 5. This seems to indicate that a minor decrease in performance of the model may be justified by a greater robustness to adversaries, an observation made by [2].

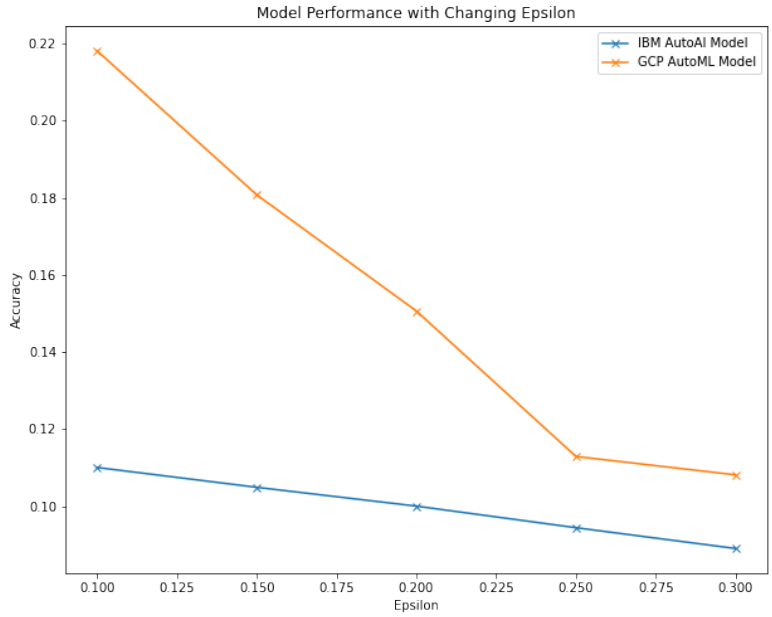

Figure 5: Model Performance with varying epsilon values. Accuracy degrades with higher epsilons

Further, GCP AutoML systems give no information about the model architecture or the hyperparameter values, which implies that only black-box attacks can be used against this Cloud Platform.

A very interesting observation made is that both IBM Cloud and GCP AutoAI models tend to predict adversaries as 8, with both of their predictions on adversarial examples consisting mostly of 8s, with more proportion of the same when epsilon increases. A plausible explanation is that many parts of 8 are similar to other numbers (think about how digital watches generate their displays).

6 Defensive Distillation

Defensive Distillation is a technique that may improve robustness towards adversarial examples [12]. This is done by training a "student" model to predict the logits of the "teacher" model as a soft training label. Here, logits refer to the probability of the input belonging to a specific class, for each of the classes. These logits encompass the differences between classification categories, and hence learning these labels help understanding the implicit knowledge gained by the teacher, without the chance of overfitting to the data. In thoery, using distillation improves the robustness of the model by having it learn inherent properties of the image, without learning on local dependencies that may be more susceptible to adversarial attacks.

For the same, we distil the IBM Cloud AutoAI model and the GCP AutoML model into two LeNet-like students having the exact same architecture, and under the same training conditions (same training examples, hyperparameters, etc). The distillation process is implemented as presented in prior work [12, 17, 16]. A flowchart of the distillation process is shown in Figure 6. The "training data" for the distillation process is the input data along with the prediction probabilities (for each class) as outputted by the teacher model. In this case, we use 80% of

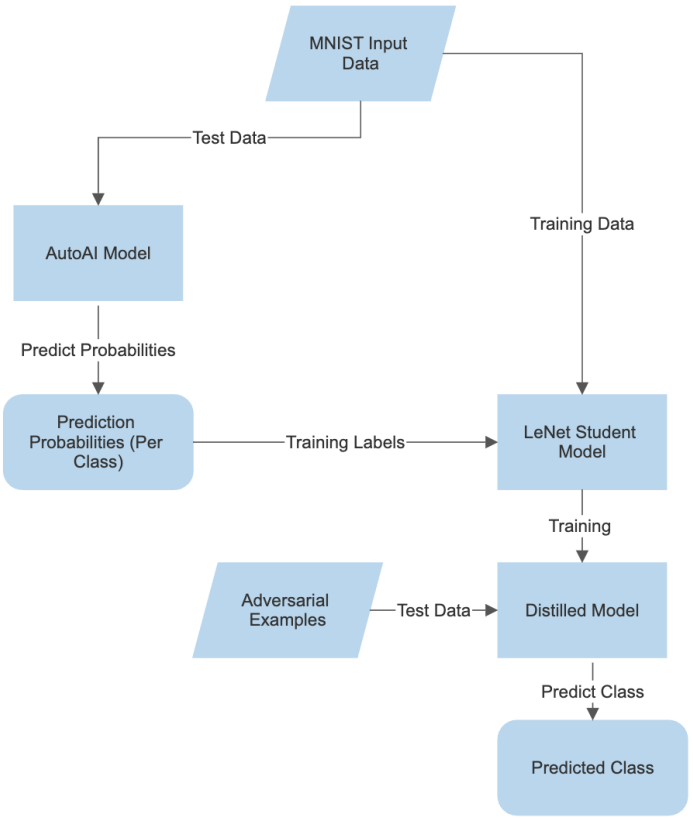

Figure 6: Flowchart for Defensively Distilling AutoAI Models

the MNIST Test data as the input, and the corresponding logit scores from the AutoAI models when given the same input as the target label. Not using the training data helps to prevent the possibility of overfitting, as the models have extremely high accuracies on the test and validation set. We test the data on the remaining 20% of the MNIST Test set, along with the adversarial examples using epsilon 0.1, 0.15, 0.2, 0.25 and 0.3 (as produced in previous sections).

Perturbation	IBM AutoAI Teacher	GCP AutoML Teacher
None (Test Accuracy)	98.981	97.795
Epsilon = 0.10	74.193	75.690
Epsilon = 0.15	68.645	75.134
Epsilon = 0.20	49.981	52.332
Epsilon = 0.25	20.451	26.552
Epsilon = 0.30	19.001	18.141

Table 5: Robustness of Defensively Distilled Models against Adversarial Examples

The accuracy of the resultant students trained on the two AutoAI models are shown in Table 5. As shown, the model distilled with the GCP AutoML Teacher performs better in almost all adversarial settings, but has a slightly lower accuracy on the test set. This shows that while the IBM AutoAI model performs better in a secure setting, a model distilled from it is more prone to adversarial attacks than one distilled from the GCP AutoML Teacher.

We see that defensive distillation improves the robustness to an extent, i.e, the accuracy scores in adversarial settings are always higher for the distilled model as compared to the original. However, defensive distillation is not a panacea for adversarial defences. This is shown in Figure 7. We see that there still remains a major decrease in accuracy for large values of epsilon, so this work concurs with prior work [2, 17] that *defensive distillation is not robust to adversarial examples*, at least not without heavy optimization of the distillation pipeline, and other

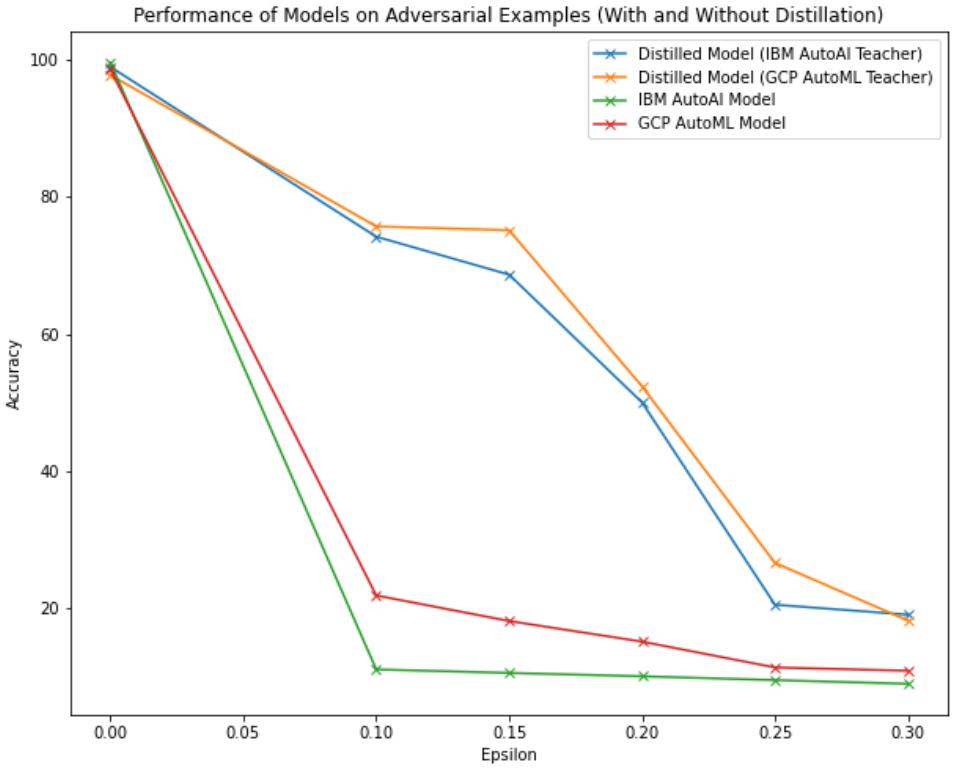

Figure 7: Model Robustness with and without Defensive Distillation Techniques

methods to improve performance. A possible future dimension of this work could be in optimizing the distillation process itself, by using more training data, robust hyperparameter tuning, and finding if a student architecture exists that is better suited for distillation.

7 Conclusion

The ubiquity and performance of neural network models have helped users easily harness the power of machine learning to make predictions, without the need of too much knowledge of ML. However, these models are not robust to minor perturbations in the input, or *adversarial examples*. The major goal of this project was to compare the performance of AutoAI models on two cloud platforms (IBM Cloud and Google Cloud Platform), and evaluate their robustness towards adversarial examples. We begin by training models on the MNIST dataset on both IBM Cloud and GCP, and compare the model generation process. We find both cloud provider's AutoAI training to be fast, efficient and extremely intuitive to users, with minimum need of expertise in the domain. After generating adversarial examples on the MNIST dataset using the Fast Gradient Sign Method, we use 5 measures of distortion to evaluate the robustness of these AutoAI models. We find that while IBM Cloud's AutoAI model has better performance on the original test set, Google Cloud's AutoML model is more robust to adversarial examples. That being said, both models are extremely susceptible to adversarial attacks, with accuracies dropping from 99% to around 10% for images with higher distortion, even though this distortion is almost indiscernible to the human eye. We further find that defensive distillation may be useful in slightly improving the robustness, but cannot be relied on as a method of defense, even against simpler white-box adversarial attacks.

References

[1] Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy, 2015. *Explaining and Harnessing Adversarial Examples*

[2] Nicholas Carlini and David Wagner, 2017. *Towards Evaluating the Robustness of Neural Networks*

[3] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, Ananthram Swami, 2015. *The Limitations of Deep Learning in Adversarial Settings*

[4] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Pascal Frossard, 2015. *DeepFool: a simple and accurate method to fool deep neural networks*

[5] Lea Schonherr et al, 2018. *Adversarial Attacks Against Automatic Speech Recognition Systems via Psychoacoustic Hiding*

[6] Nicholas Carlini et al, 2019. *On Evaluating Adversarial Robustness*

[7] *IBM Cloud*, https://www.ibm.com/cloud

[8] *Google Cloud Platform*, https://cloud.google.com/

[9] *Watson Studio's AutoAI*, https://www.ibm.com/cloud/watson-studio/autoai

[10] *Cloud AutoML*, https://cloud.google.com/automl

[11] Yann LeCun and Corinna Cortes, 2010. *MNIST handwritten digit database*

[12] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha and Ananthram Swami, 2016. *Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks*

[13] Dakuo Wang, Parikshit Ram, Daniel Karl I. Weidele, Sijia Liu, Michael Muller, Justin D. Weisz, Abel Valente, Arunima Chaudhary, Dustin Torres, Horst Samulowitz, and Lisa Amini. 2020. *AutoAI: Automating the End-to-End AI Lifecycle with Humans-in-the-Loop.*

[14] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, Cho-Jui Hsieh, 2017. *ZOO: Zeroth Order Optimization based Black-box Attacks to Deep Neural Networks without Training Substitute Models*

[15] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, Rob Fergus, 2013. *Intriguing properties of neural networks*

[16] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, 2015. *Distilling the Knowledge in a Neural Network*

[17] Nicholas Carlini and David Wagner, 2016. *Defensive Distillation is Not Robust to Adversarial Example*

[18] *PyTorch Examples*, https://github.com/pytorch/examples/tree/master/mnist

Appendix A: IBM Cloud AutoAI Performance

The figures for the performance of the selected AutoAI pipeline, including the F1-Threshold, Precision Threshold and Recall Threshold, along with the Precision-Recall curves, and ROC Curves for all holdout validation sets are shown below.

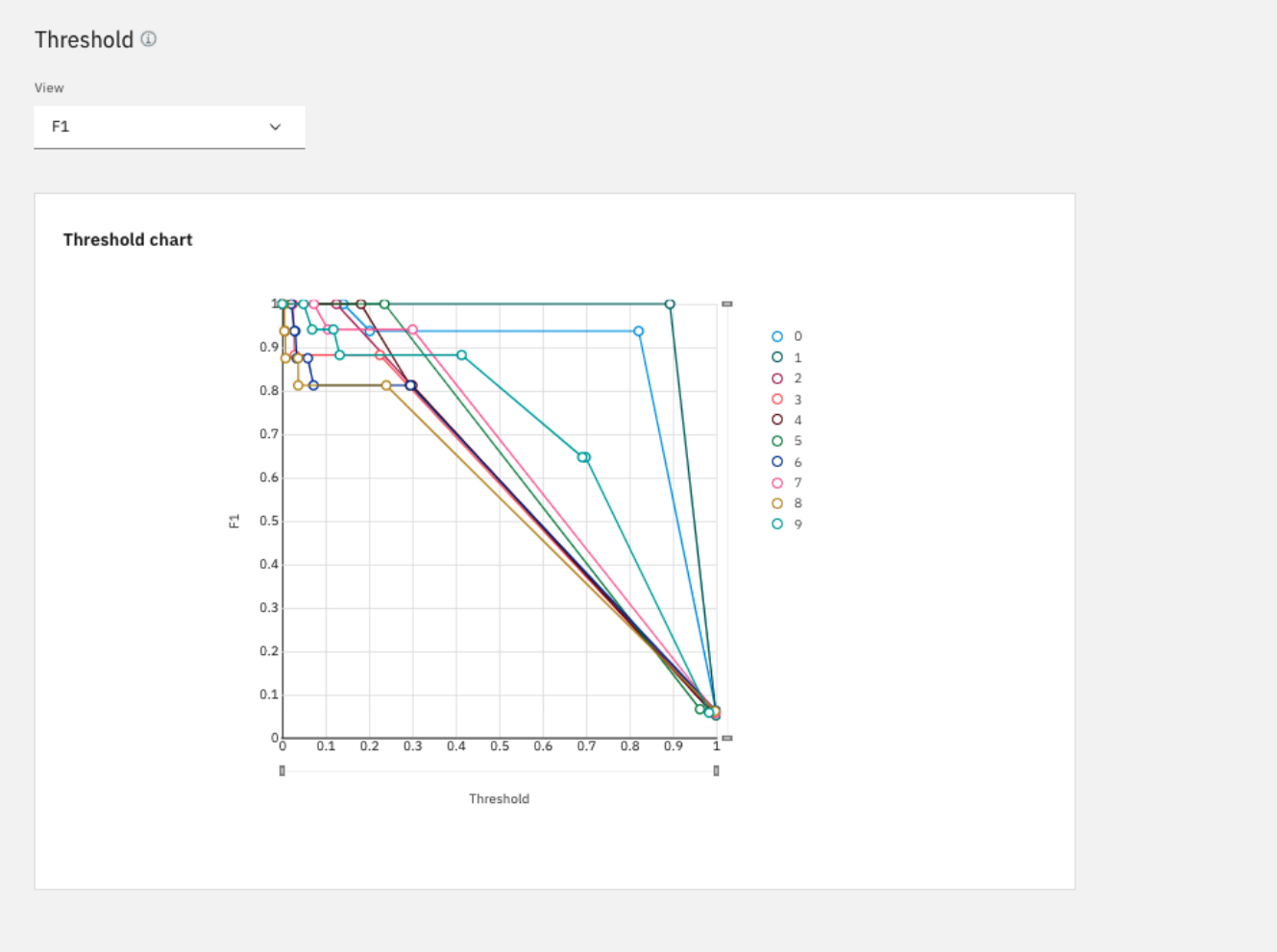

Figure 8: IBM Cloud AutoAI Model Pipeline 12 F1 Threshold for all validation sets

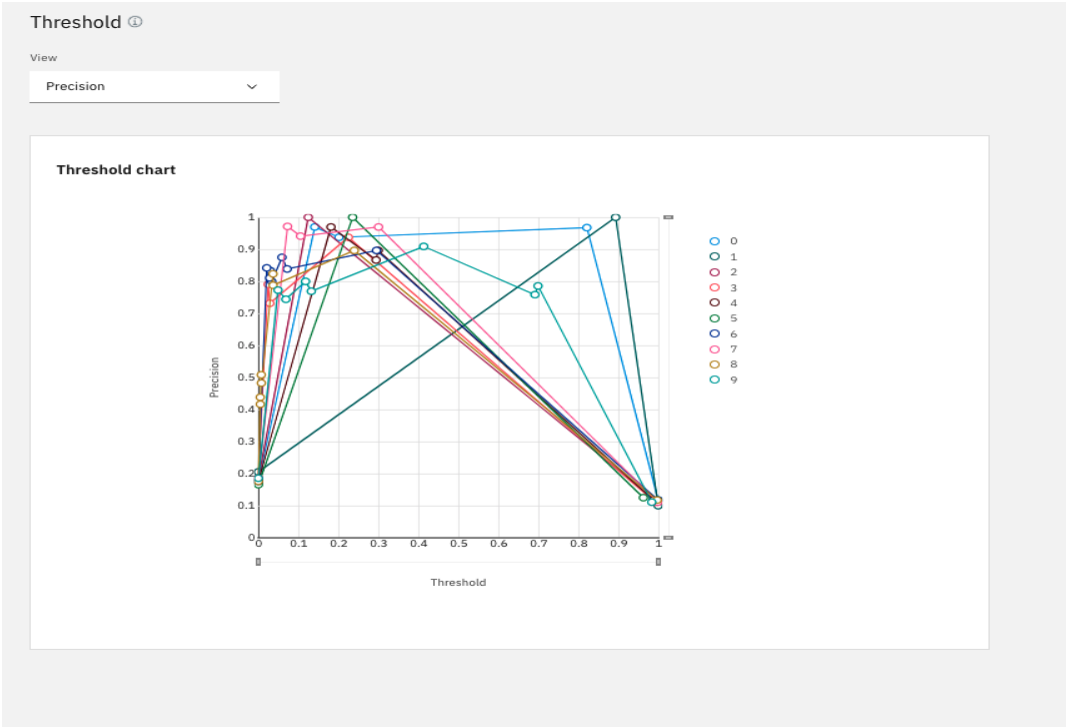

Figure 9: IBM Cloud AutoAI Model Pipeline 12 Precision Threshold for all validation sets

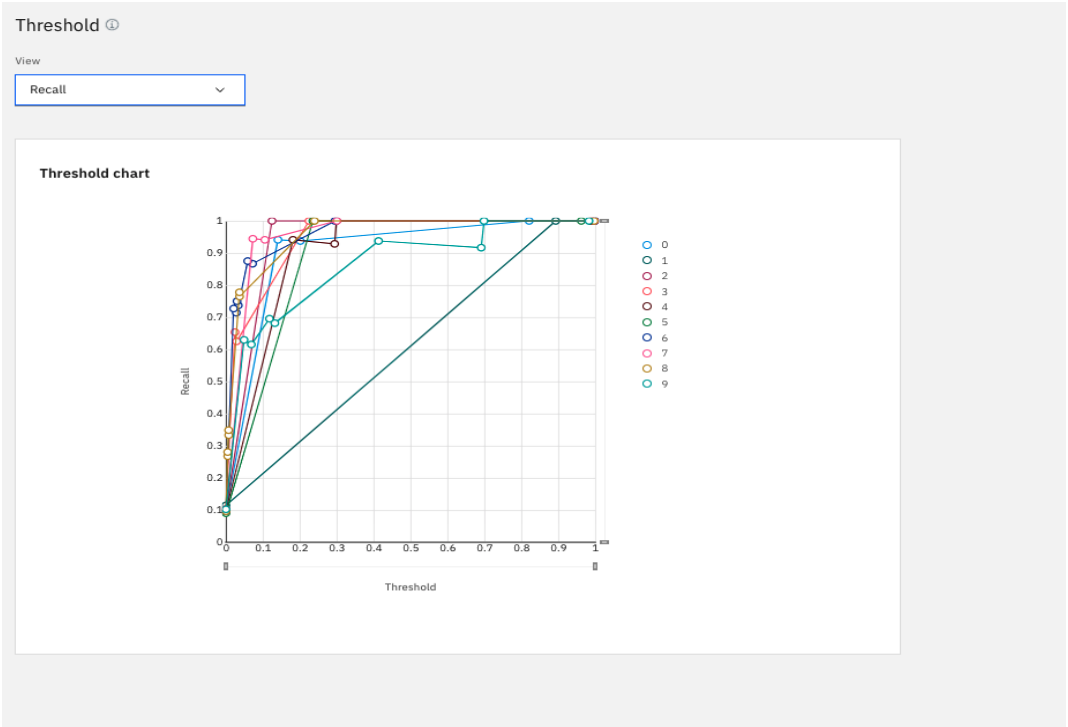

Figure 10: IBM Cloud AutoAI Model Pipeline 12 Recall Threshold for all validation sets

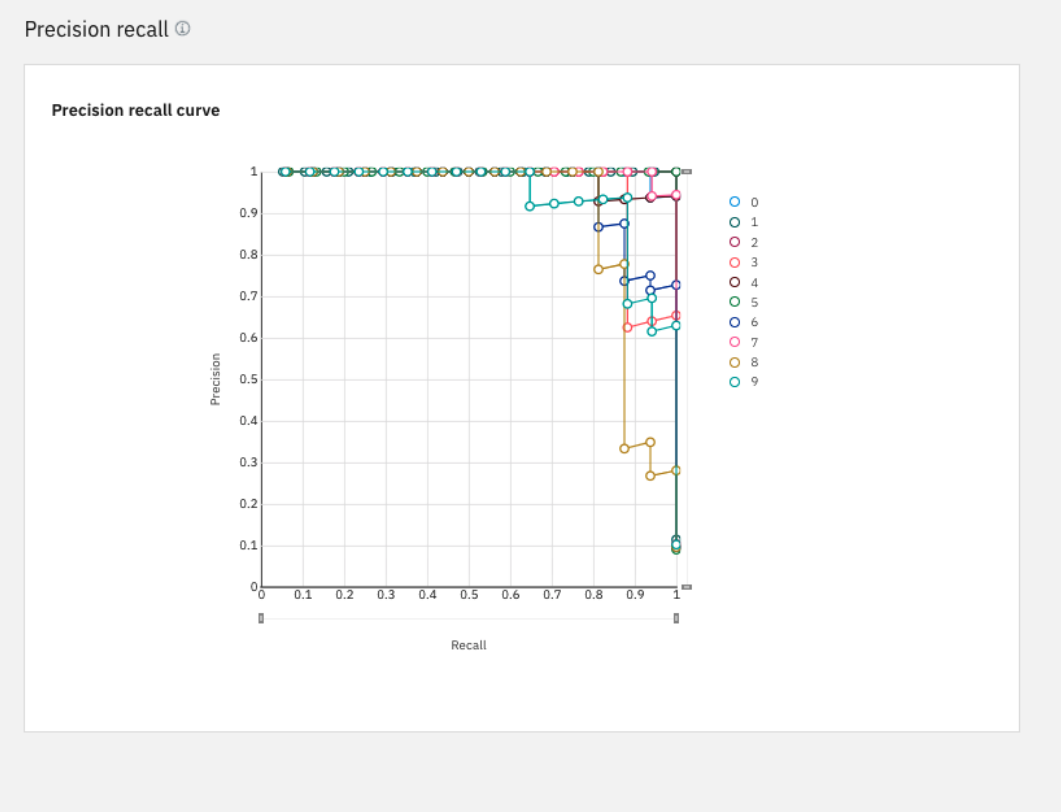

Figure 11: IBM Cloud AutoAI Model Pipeline 12 Precision-Recall Curve for all validation sets

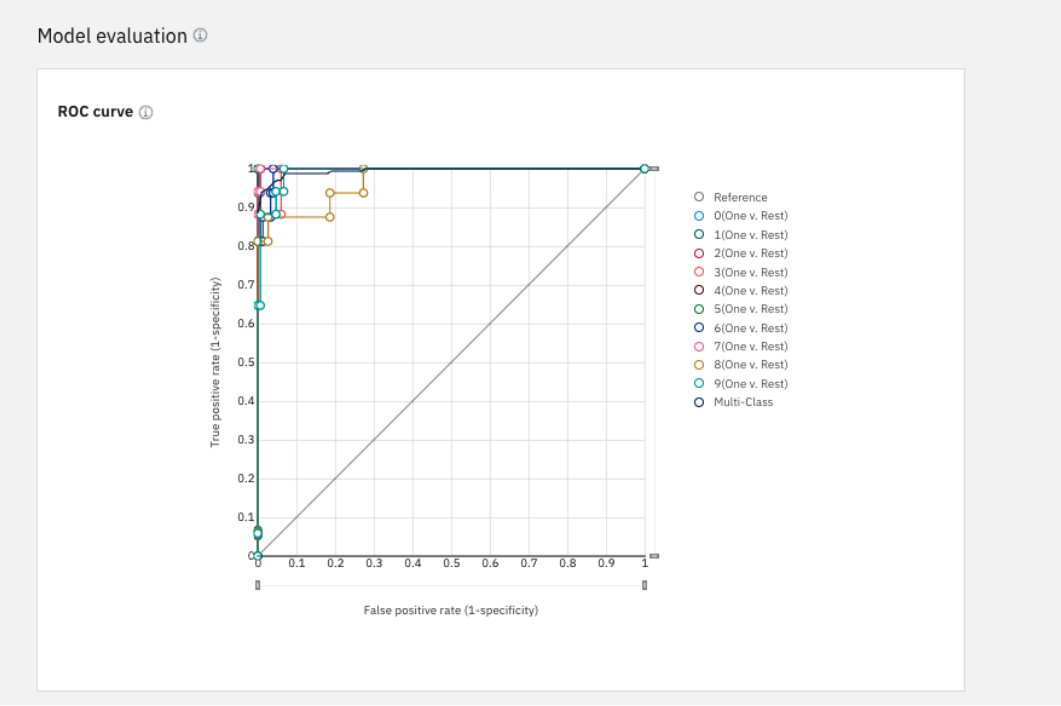

Figure 12: IBM Cloud AutoAI Model Pipeline 12 ROC-Curve for all validation sets

Appendix B: GCP AutoML Performance

The figures for the performance of the selected AutoML pipeline, and the deployment times for the GCP Model.

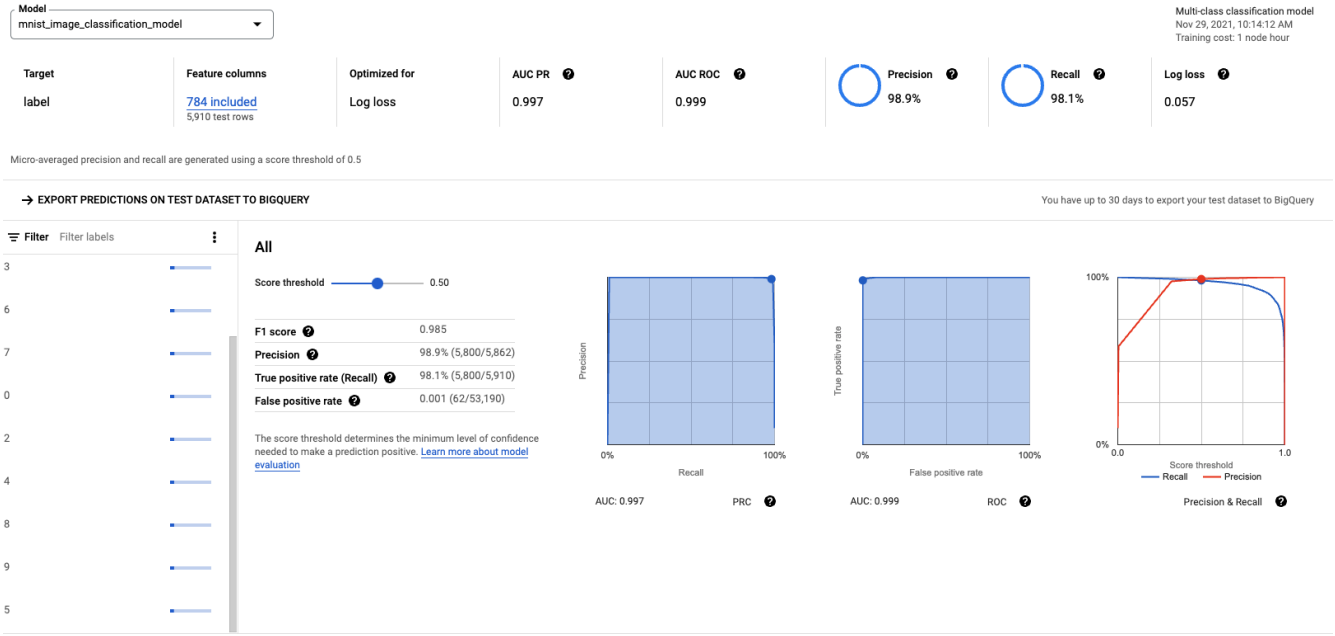

Figure 13: GCP AutoML Model Performance

Model	Deployment Time
MNIST Test data	9 min 1 sec
Adversarial Examples (epsilon 0.10)	9 min 43 sec
Adversarial Examples (epsilon 0.15)	9 min 51 sec
Adversarial Examples (epsilon 0.20)	9 min 40 sec
Adversarial Examples (epsilon 0.25)	9 min 46 sec
Adversarial Examples (epsilon 0.30)	10 min 6 sec

Table 6: GCP AutoML Model Deployment Times