Automated Smart TV UI Performance Testing with Visual Recognition

Peng (Peter) Li
Stanford University
peter888@stanford.edu

Shim-Young (Ricky) Lee
Stanford University
sylee1@stanford.edu

Abstract

In this project, we applied the current convolutional neural network based image classification, and cluster algorithm to automation of Smart TV UI performance testing. Applications such as YouTube specify upper bounds for launch and page transition time that device manufacturers need to meet, and this performance test is currently done manually. Although measuring page transition time is a relatively trivial task for humans, the fact that application displays different texts and images each time (due to its recommendation system) precludes simple solutions such as pixel comparison. At the current stage of development, the convolutional neural network (Inception v3) [3] was shown to be able to classify different types of UI pages extremely well. We will now focus on the detection of movement of displayed images using object localization.

1. Introduction

Smart TV applications such as YouTube and Netflix have performance requirements. YouTube certification program specifies requirements regarding UI performance for OEM providers, system integrators and smart TV vendors that wish to ship their devices with YouTube application. Among those requirements is page transition performance, which refers to the time the transition between different types of pages (Guide, Menu, Settings etc.) takes. The YouTube certification program provides upper bounds needed to be met for different types of page transition. Although measuring page transition time seems like a trivial task for human eyes, it is currently measured manually, and the goal of this project is the automation of this performance testing procedure using current computer vision techniques.

Deep learning based visual recognition algorithms such as Convolutional Neural Network can be used to recognize different pages and measure application's launch time and page loading time. Sub-page transition, which refers to the movement or change of displayed images or highlighting windows within the same UI page and loading time, can be recognized with object localization.

To automate the measurement of page transition, we retrieve an image at each time step and run image classification algorithm to determine whether the YouTube application has transitioned from displaying one type of page to another. Additionally, we can incorporate some prior knowledge about the UI to eliminate some of the possible outcomes to improve the classification result. We call this “context dependent”, since the classifier relies not only on the input images but also other “context” information. For image (screenshot) retrieval and collection, we will use BlackMagic image capture card. The outcome is a program that automatically measures the page transition performance.

Intuitively, the evaluation metric should measure how accurately the program classifies the transition (which page to which page) and pinpoints the time of transition, which allows us to measure the page transition performance and ultimately automate the testing procedure. Note that this is not the same as the classification rate (accuracy) of the neural network for collection of screenshots. The details of the metric are parts of the engineering challenges of this project and will be developed as we progress.

2. Related Work

Although the project itself has a very specific application, the core engineering task is a fairly standard image classification. Thus, we plan to review literatures on important neural network architectures that have come out in recent years. Starting from the convolutional neural network architecture used in ImageNet competition by Alex Krizhevsky *et al.* [1], we reviewed Inception architectures [2, 3], and ResNet [4] to name a few.

Regarding the YouTube certification program for devices, we refer to "YouTube TV HTML5 Technical Requirements 2017" [9], which specifies different test and criteria needed to be performed and met.

3. Methods

Overview of the pipeline. The pipeline for the automated performance testing can be divided into mainly two stages: 1) data gathering and model training and 2) multiple measurements and final decision. First, the we gather the image data for training by taking screenshots over many launches of the application. Once the images are labeled, we train a classifier.

In the test run, the automated testing program will retrieve an image at each time step as the YouTube application launches and runs, and an image classification algorithm determines what type of page the application is currently displaying. The program measures the page transition times based on the classification output. The measurement is repeated many times and the final statistic and the decision is given.

3.1 Image Classification

The final program needs to perform image classification at each time step, and the classifiers were trained to classify 5 different page types ($y \in \mathbb{R}^5$).

3.1.1 Convolutional Neural Networks

Model Architecture. We tested two convolutional neural network architectures, Inception-v3 classifier [3] provided by the latest version of TensorFlow, and a simple convolutional neural network architecture resembling the model by Alex Krizhevsky *et al.* [1] The simple convolutional neural network used one convolution layer with a 4×4 filter and a single fully connected layer. The cross-entropy loss was used as a loss function. The idea was to build the simplest architecture possible to reduce the number of parameters and the classification speed.

3.1.3 Non-Neural Network Classifiers

To provide a benchmark for convolutional neural networks, we also tested simpler classifiers, which are linear classifier with the softmax, Support Vector Machine (SVM) and k-nearest neighbors (k-NN).

3.2 Page Transition Time Measurement

Once the classifier is trained, new screenshots are gathered over multiple launches of the application. The classifier classifies these test images, and gives a series of classifier outputs for each launch. To get an accurate measurement of the page transition time, we summarize the statistic with a confidence interval.

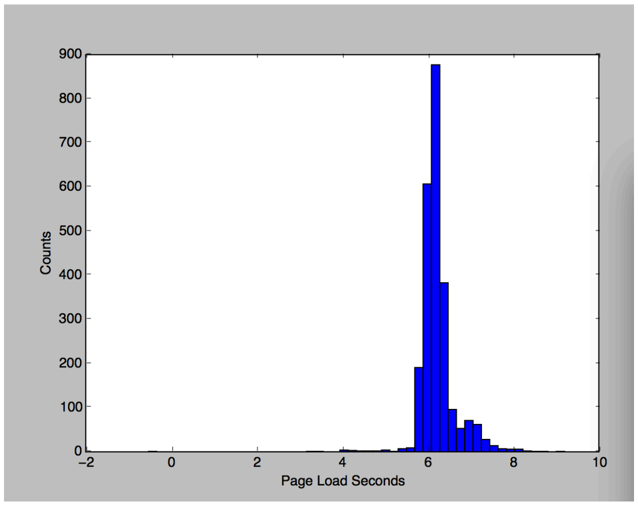

Figure 1. Histogram of application launch time.

The figure 1 shows the distribution of launch time, which we approximate with a normal distribution. After obtaining n measurements, a confidence interval (90% or 95%) based on the mean and the standard deviation calculated from those measurements is calculated.

The decision rule for the performance test is arbitrary, because it is a criterion set by the testers (for example, YouTube). Of the simplest decision rules are the rules based on the mean, median or the maximum of the transition time:

$$\hat{\boldsymbol{f}}(\boldsymbol{x}) = \mathbb{I}\{\bar{\boldsymbol{x}} < \boldsymbol{\gamma}\} \qquad (1)$$

$$\hat{f}(x) = \mathbb{I}\{\tilde{x} < \gamma\} \qquad (2)$$

$$\hat{f}(x) = \mathbb{I}\{\max(x) < \gamma\} \qquad (3)$$

If we use the first two rules, however, roughly 50% of the transition times in practice will be greater than the reported value used for decision. Given that the measurement of the page transition time has previously been done manually with a stopwatch, these criteria are still acceptable, but the previously stated characteristic of these rules might not be attractive enough especially when the statistic is barely smaller than the requirement γ.

The third criterion is the most rigorous, and the simplest. It is the best choice if the testers demand that all runs of the application must achieve a page transition time less than the requirement.

The decision rule we propose is whether the upper bound of the confidence interval is less than the specified transition time.

$$\hat{f}(x) = \mathbb{I}\left\{\bar{x} + t^* \frac{s}{\sqrt{n}} < \gamma\right\} \quad (4)$$

$$s = \sqrt{\sum \frac{(x-\bar{x})^2}{n-1}} \quad (5)$$

This criterion is more conservative than the mean or the median based rule, and is more resistant to the outliers in the right tail of the distribution. This, of course, is not an intrinsically desirable quality of a decision rule; if the tester requires that the page transition time should be less than the specified time in all circumstances, the maximum criterion is more appropriate. One advantage of this rule is that it gives a rough (though not statistically correct) heuristic that the probability that true mean is greater than the obtain upper bound is about 2.5% or 5% (if the confidence level is 95% or 90%). Also, the rigor of the test can be controlled with the confidence level, which is a parameter.

4. Dataset and Features

4.1 Dataset Description

When the YouTube application launches, it sequentially displays 5 different page, namely, White Screen, Logo, Spin Loading, Text Loaded, and Image Loaded. White Screen and Logo pages are static pages, lacking dynamic elements. Spin Loading page only has a spinning wheel in the center of the screen.

Shown below are typical images for YouTube application. The pages of class White Screen, Logo, Spin Loading are not shown. The labeled YouTube screenshots can be found in: https://goo.gl/QGJ3i3.

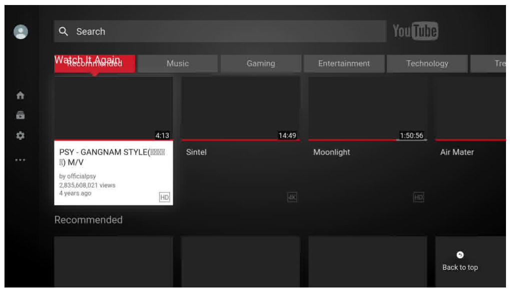

Figure 2. Image of type "Text Loaded". The text may change for different launches, making simple pixel comparison methods difficult.

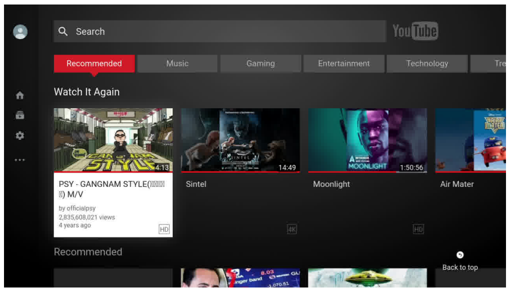

Figure 3. Image of type "Image Loaded". Again, video thumbnails and text may change for different launches.

4.2 Data Acquisition and Labeling

For image retrieval and collection, we used Decklink image capture card from BlackMagic [10], the capture is based on STB's output 720P, and frame rate is 60Hz, capturing 60 screenshots per second. The images were labeled in a semi-automated fashion: since the pages are shown sequentially, we used the average launch time and the average durations for which each page is displayed to roughly divide and save the images in subfolders. Then each folder was manually checked to move the wrongly assigned images to the correct folders.

Each class has over 400 images, and the dataset contains a little more than 4000 labeled images in total. To gather this dataset, the application was launched more than 1000 times, which took roughly 5 hours. Since the size of each class is proportional to how long the application displays that page, there was a class imbalance in training and test data. We did not address this issue, mainly because the result obtained with this raw dataset was sufficiently good.

The size of the dataset is quite small compared to a typical dataset size for training of neural networks, but the result suggests that the current size is sufficient especially because the in-class variability of the images is very small and each class was sufficiently different from the other classes. The number of training samples needed depends on the model choice. The number of training samples can be increased or decreased flexibly since the tester has a complete control over the size of the dataset.

Labeling with K-Means Clustering. In addition to the original dataset, for which the labeling needed some manual correction, a dataset labeled with k-means clustering was gathered, taking advantage of the fact that our image data has a low in-class variability. The idea was to test the possibility of a full automation of the data labeling process.

Instead of manually sorting and labeling the images, we clustered the dataset into 5 clusters using k-means clustering. Since clustering algorithm assigns the class label arbitrarily, Bayer-Moore majority vote algorithm [12] was used to find the right permutation of class labels that matches labeling of the test data. However, this is only necessary for the testing of clustering performance.

4.3 Pre-process Images

The original images are 3-channel images, but were re-sized to 72×128 greyscale images.

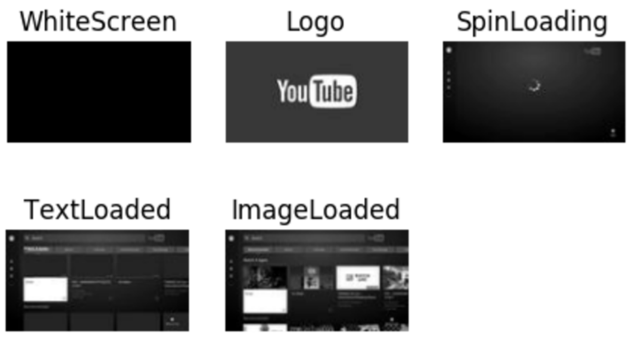

Figure 4. Images after pre-process.

5. Results

5.1 Classification Results

The table below summarizes the test accuracies of the classification algorithms we tested.

Model	Test Accuracy
k-NN	1.00
SVM	0.89
Softmax Linear	1.00
Simple-CNN	1.00
Inception V3	1.00

Table 1. the test accuracy of different classification algorithms.

The classification task was easy enough for most classifiers used, and all classifiers except for the SVM achieved 100% test time accuracies. This is not surprising given our dataset since the in-class variability was very low.

Linear classifier with softmax was the fastest classifier, and for this YouTube dataset, was also the best classifier, since it was the fastest classifier that achieved 100% accuracy. Although the k-NN achieved 100% accuracy as well, it was noticeably slower than the other classifiers, since it has to search for the closest image each time.

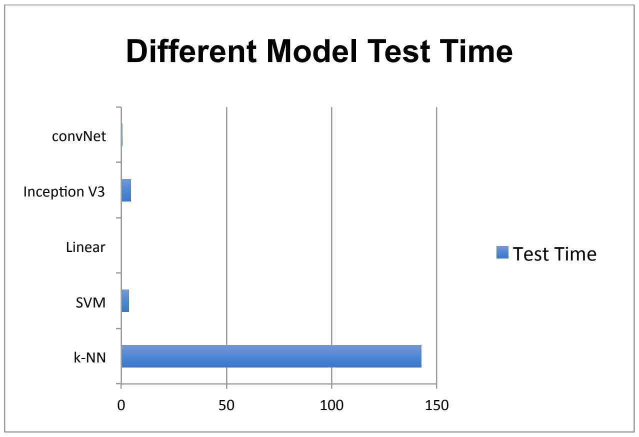

Figure 5. Different model test time performance.

For the UI performance testing of applications that yield similar in-class distribution of images, the linear classifier with softmax is likely to perform very well in both accuracy and speed. The launch time measurement falls into this category, since many apps simply show a sequence of static images when it launches.

If the in-class variability is large, and the images of different classes are similar, the softmax as well as the other non-convolutional neural network classifiers might not be able to achieve 100% accuracy. Convolutional neural networks, however, are much more powerful in their classification capability and will likely be able to achieve 100% accuracy on almost all kinds of UI page transition, since the in-class variability and the similarity between different image classes are inherently very limited compared to the standard image classification tasks such as ImageNet.

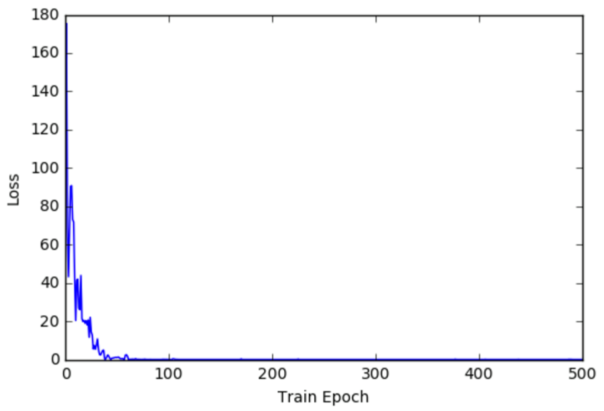

Figure 6. The training time loss converges very fast and well for the simple CNN, with little hyperparameter search.

As expected, the training of the convolutional neural network for this task is fairly easy, and the loss converges very fast with no complication. This is a very important property, since we want to minimize the time the testers need to spend to search for a convolutional neural network architectures and a set of hyperparameters that give 100% or close to 100% accuracy, so that even testers with little to no knowledge of neural network and model training can easily use the pipeline with minimal instructions.

5.2 K-means Clustering for Data Labeling

Taking advantage of the fact that the in-class variability is small and the classes are dissimilar enough, we also tested a k-means clustering for labeling of the data, to test the possibility of full automation of the data gathering process.

To test the clustering performance, the original training dataset (that we manually labeled) was clustered with k-means clustering algorithm, and the accuracy was measured by comparing it to the true label. The direct application of clustering to the entire training dataset only achieved about 80% accuracy, which is unusable for the purpose of automatic labeling (and of course, classification).

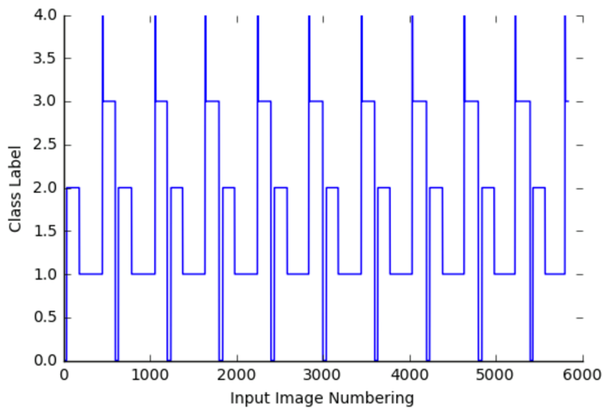

Figure 7. The plot of the class label assignment by k-means clustering vs. sequential input images over multiple launches of the application.

The unexpectedly low performance (given the dataset) was due to the class imbalance. To address the imbalance problem, centroids for each class were obtained from clustering a small number of images evenly selected from each class (and to do this we need a small but hand-labeled data), and the rest of the data was labeled using these centroids. Although this compromises the goal of full automation, the k-means clustering achieved 95% training time accuracy and 92% test time accuracy. The figure 7 shows the class label assignment of a sequential input images. The clear periodic pattern suggests that the clustering can assign the labels reasonably well.

The simple convolutional neural network was trained with this "imperfectly labeled" dataset, and achieved 96% test accuracy on the original test set. With further optimization of the clustering and measurement procedure, this data labeling scheme can be used for a further automation of the pipeline.

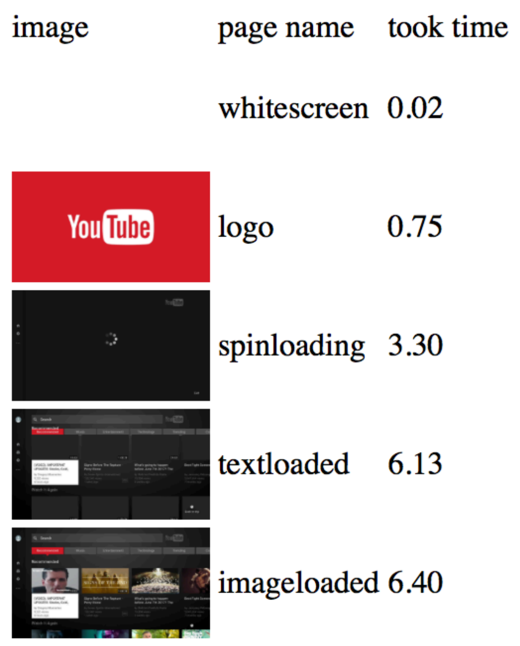

image	page name	took time
	whitescreen	0.02
	logo	0.75
	spinloading	3.30
	textloaded	6.13
	imageloaded	6.40

Figure 9: The screenshot of a demo of the automated tester.

6. Conclusions

As the result section shows, we have successfully applied current deep learning based visual recognition as well as other basic machine learning techniques to the automation of UI performance testing. We have shown that the convolutional neural networks achieve 100% test and training accuracy, showing that the convolutional neural network can easily deal with

7. Future Work

In the future, a similar pipeline and a training procedure can be applied not only to similar UI performance testing of different applications, but also to all UI related test automation that requires similar computer vision capabilities. For example, similar system can be used to iOS and Android UI related test automation, because the details of the system can be adjusted to match iOS and Android UI's image resolutions and test criteria. This can be easily achieved by a pre-processing of training and test image and if necessary, tweaking of model architecture parameters.

During the development of our automated UI tester, the new iOS11, in the recent 2017 Apple Worldwide Developer Conference, was announced to be equipped with the Core Machine Learning Tools (Core ML) for vision tasks. The Core ML supports a variety of machine learning algorithms, from simple classifiers such as trees and SVM to the state-of-the-art neural network architectures such as Inception V3, VGG16 and ResNet50. Using these tools, we can implement the same pipeline, training and performing image classification in the server with iOS11 device. Therefore, this performance measurement scheme can potentially be implemented with a single iOS application.

References

1. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105, 2012.
2. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 1–9, 2015.
3. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. arXiv preprint arXiv:1512.00567, 2015.
4. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.
5. R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.
6. J. Huang V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, K. Murphy, Speed/accuracy trade-offs for modern convolutional object detectors, In *CVPR* 2017
7. J. Dai, Y. Li, K. He, J. Sun. R-FCN: Object Detection via Region-based Fully Convolutional Networks, In *NIPS*, 2016
8. A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In CVPR, 1725–1732, 2014.
9. YouTube TV HTML5 Technical Requirements. https://drive.google.com/file/d/0B0C4aKjz5kL6X3hsTUg5NjJCaWs/view?usp=sharing
10. Decklink Capture Card from Black Magic Design https://www.blackmagicdesign.com/products/decklink
11. Boyer-Moore majority vote algorithm https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_majority_vote_algorithm
12. iOS11 machine learning API https://developer.apple.com/machine-learning/
13. Selenium WebDriver which used to launch the YouTube app http://www.seleniumhq.org/projects/webdriver/
14. Image Module used for preprocessing images – Pillow http://pillow.readthedocs.io/en/3.1.x/reference/Image.html
15. Ffmpeg https://ffmpeg.org/