Using Transfer Learning, Spectrogram Audio Classification, and MIT App Inventor to Facilitate Machine Learning Understanding

Nikhil Bhatia[1], Natalie Lao[1]
[1]Massachusetts Institute of Technology, USA
nwbhatia@mit.edu, natalie@mit.edu

ABSTRACT

Recent advancements in deep learning have brought machine learning and its many applications to the forefront of our everyday lives. As technology has become more and more integrated into our educational curriculum, researchers have focused on creating deep learning tools that allow students to interact with machine learning in a way that incites curiosity and teaches important concepts. Our research contribution focuses on applying transfer learning and spectrogram audio classification methods to teach basic machine learning concepts to students. We introduce the Personal Audio Classifier (PAC), a web interface that allows users to train and test custom audio classification models that can classify 1-2 second sound bites recorded by the user. We also contribute a custom App Inventor extension that allows users to use the output of the web interface to create App Inventor applications that rely on their trained custom audio classification model.

1. INTRODUCTION

From personal voice assistants to self-driving cars, machine learning applications have permeated every aspect of our daily lives. Much of these advances are thanks to the subfield of machine learning known as deep learning, a field primarily concerned with building large neural networks to perform specialized tasks. Yet as researchers began to make significant advancements in deep learning during the past decade, it became clear that computational complexity, training time, and esoteric development tools could pose as a deterrent to widespread development of deep learning applications. Transfer learning was born out of this deficiency, spurred by Yosinski's 2014 work [1] on transferable features in deep neural networks.

1.1. Transfer Learning

Transfer learning is a machine learning method where an existing deep learning model is used as the starting point to train a model specialized for a slightly different task. The ability to start with a pre-trained model allows new developers to apply deep learning to solve novel problems without the vast compute and time resources normally needed to train neural networks from scratch. To take an example, we can look at one recent notable application from the field of deep learning: the development of robust deep convolutional neural networks used for image classification and object recognition. In 2012, Hinton [2] first showed that their deep neural network with over 60 million parameters and 650,000 neurons could be trained to classify ImageNet images with startling accuracy. However, developing this complex model required a uniquely efficient GPU implementation of the convolution operation, tens of thousands of dollars in state-of-the-art GPU hardware, and months of training and testing. While this development process is likely only accessible to researchers or institutions with deep pockets, the result is one that should be available to developers of all levels and even students of any age. Transfer learning has allowed for just this, giving machine learning enthusiasts around the world the ability to build their own models using complex models like Hinton's as a starting point.

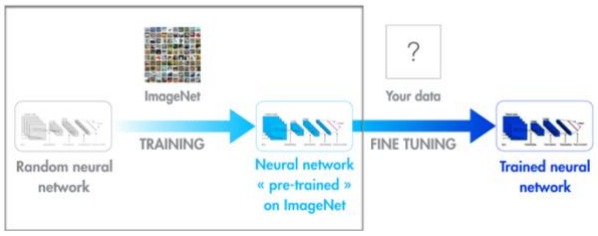

Figure 1. Transfer learning starts with a pre-trained model and fine-tunes the output layers to specialize towards a new task.

1.2. TensorFlow.JS and App Inventor

By using transfer learning as a starting point, the possibilities for novices to experiment and understand machine learning are endless. However, if we want to create effective machine learning engagement tools for students and machine learning novices, we have to think beyond the neural network. To remain engaged, students must interact with machine learning in a way that precipitates curiosity while demonstrating the power of deep learning models.

Suppose Oli, a high school student, has recently been introduced to machine learning. Oli has only a surface-level understanding of machine learning and computer science concepts, but has learned to identify instances of machine learning in his everyday life: things like FaceID on his iPhone, or interacting with Siri and Alexa. Recently, Oli has become especially fascinated by the power of personal assistants, and has noticed that Siri and Google Assistant can be trained to only respond to his own voice. Motivated by the ubiquity of privacy and technology in today's media headlines, Oli wants to combine his interest in machine learning with an attempt to build his own privacy-conscious application. He comes up with the following goals:

1. Learn how audio-classification works, and train his own custom audio-classifier that can detect features like speaker, emotion, language, etc.

2. Leverage this audio-classifier to create a private voice diary on his own phone that would only unlock at the sound of his voice.

To help Oli achieve this goal, we can build him a flexible audio-classification tool that uses transfer learning to retrain a model to classify unique types of audio. With such a tool, Oli could easily train a model to identify different types of emotion, or even the sound of his own voice. However, Oli does not come from a computer science background and would likely be unable to use machine learning libraries like PyTorch [3] or TensorFlow [4]. Furthermore, Oli has little experience developing mobile applications and hopes to find a way to achieve his goals without needing to learn an entire mobile development stack.

Thus we introduce two important technologies, Tensorflow.js [5] and MIT App Inventor [6], that this project utilizes to help students like Oli develop exposure to machine learning concepts without requiring a deep computer science background. Tensorflow.js is a Javascript machine learning library that has recently found success in the niche bridging machine learning implementation and educational tools. It allows for deep learning models to be trained and run right in the browser, and when combined with a well-designed web GUI, can hide the complexities of programming syntax while still allowing users to interface with machine learning models. Similarly, MIT App Inventor is a free open-source web platform that allows users to create mobile applications via a drag-and-drop interface, requiring little to no programming experience while still offering rich application functionality. App Inventor also offers the ability to add custom extensions to any app, allowing us to build an audio classification extension that Oli could upload and use to help build his private diary app. With these two technologies, we've created a web app that blends PIC [7] and Teachable Machine [8], allowing users to train an audio classification model that can recognize 1-2 second audio clips. After using this web app to train a custom model, users will be able to download this model and plug it into MIT App Inventor as an extension to build apps with custom audio-classification functionality.

2. RELATED WORK

Spectrogram audio classification has been a topic of interest within audio classification for some time now. By converting audio clips to their visual frequency representation (a 1-to-1 mapping that retains all audio information), we can easily apply existing CNN methods to a classification problem that would normally require more complex RNN methods to see reliable results. Many of these methods are implemented in Tensorflow [9] or PyTorch [10]. Less work has been done to create a flexible spectrogram audio classification tool that can run in the browser. However, Google has released a pre-trained Tensorflow.js audio classifier that can run in the browser and recognizes the 9 digits and a few other words [11]. This model was trained on 100,000+ audio files, but does not provide much flexibility in terms of possible classifications as its homogenous training dataset precludes it from generalizing well beyond spectrograms of specific audio clips. Our contribution, the Personal Audio Classifer (PAC) is a novel approach that uses Mobilenet [12] (trained on 100,000+ non-spectrogram images) to provide a more robust base CNN model that has learned from the diversity present in general image datasets. We can then retrain MobileNet to focus on the more trivial task of classifying spectrograms, given spectrogram images are significantly less variant than the images present in MobileNet's training dataset.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 2. The pretrained MobileNet model, with 28 layers, provides the base model for our in-browser spectrogram classifier.

3. APPROACH

3.1. Personal Audio Classifier

We present a web application (Personal Audio Classifier, or PAC) that allows users to train a custom audio classifier using Tensorflow.js within the browser. The application is available to the public at c1.appinventor.mit.edu. This section will detail the basic functionality, as well as the machine learning tools that were used to implement an in-browser audio classifier. First, users are prompted to add custom labels that the classifier will attempt to differentiate between. Users can then record an unlimited number of audio clips for each label that will be used to train the internal model. Each audio clip is one second long, and client side JavaScript is used to up-sample each audio clip to 384,000 Hz. Each element in the audio buffer is passed through a Fast Fourier Transform to draw the audio frequencies onto a single pixel sliver of our output spectrogram. This spectrogram provides a visual representation of the recorded audio bite, and is attached to the corresponding label so that the user can view each audio clip in the browser.

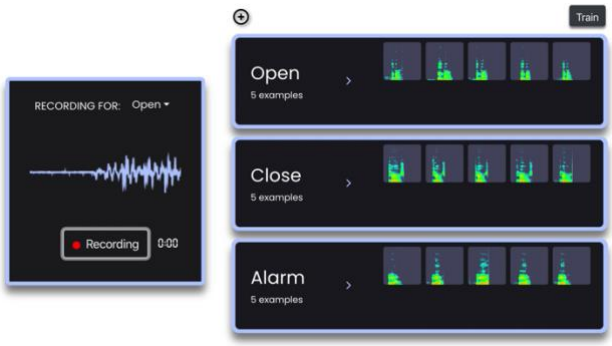

Figure 3. The label view allows users to add custom labels and record corresponding audio clips. Audio clips are up-sampled and converted to spectrograms in the browser.

After inputting a number of labels and recording the corresponding audio clips, the user is prompted to train a custom model using their provided training data, specifying hyperparameters like Learning Rate, Optimizer, Epochs, and Training Data Fraction. The web application then proceeds to load a pretrained ImageNet model (MobileNet), and trains a custom machine learning model in the browser using the activations outputted from passing the training data through the pretrained model. After experimenting with a variety of model architectures, we decided to standardize the custom model to have a single convolutional layer, a single flatten layer, and two dense layers. The output of the model is then passed through a SoftMax layer to generate probabilities that correspond to the user-inputted labels.

A separate page allows the user to use this custom trained model as a classifier, recording audio clips that are passed back through the model and assigned to one of their original labels. The corresponding label confidences are displayed after each clip is recorded, and we aggregate the test results so the user can analyze the success of their custom classifier, and even download the custom model for use in the App Inventor extension.

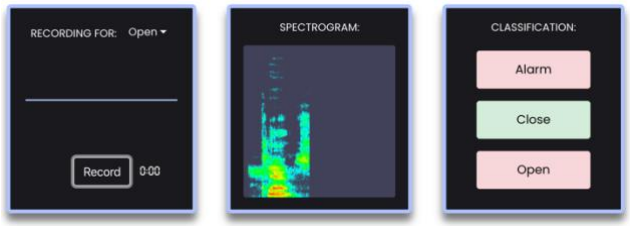

Figure 4. The test view allows users to record and classify audio clips that are converted to spectrograms and passed through the custom classifier.

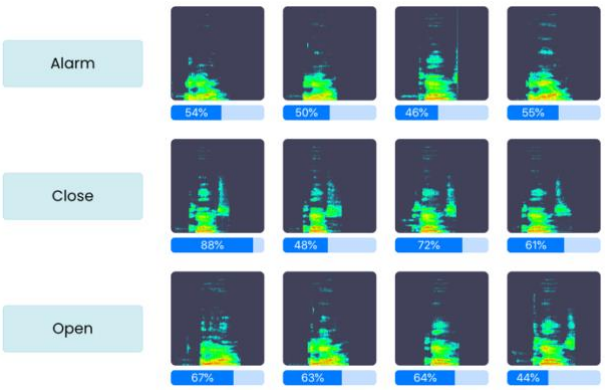

Figure 5: The test view also provides the aggregated results from classifying user-recorded audio clips.

3.2. App Inventor Extension

Our final contribution is a custom App Inventor extension that allows for users to upload their PAC model and build applications in App Inventor using their custom audio classifier. The extension is built in Java and Javascript and fully integrates with App Inventor, providing users with an API to interact with and classify audio clips that are recorded within App Inventor applications. The following event functions are provided to the user:

1. ClassifierReady: An event function indicating that the classifier is ready.
2. GotClassification: Event indicating that classification has finished successfully. Result is of the form [[class1, confidence1], [class2, confidence2], ..., [class10, confidence10]].
3. ClassifySoundData: Performs classification on the image at the given path and triggers the GotClassification event when classification is finished successfully.
4. Error: Event indicating that an error has occurred.

4. EVALUATION

To evaluate PAC, we utilize the web interface to train a number of custom classifiers. We test each classifier in the browser on a 2015 Macbook Pro, and provide five training clips for each label. The following test cases are evaluated.

1. **Number Classification:** We train the classifier on 3 different numbers: [One, Seven, Eleven], and evaluate the classifier with 15 test audio clips, five per class. We see a 100% test accuracy with each of the 15 audio clips classified to the correct class.
2. **Word Classification:** We train the classifier on 3 different words: [Open, Close, Alarm], and evaluate the classifier on 15 test audio clips, five per class. We see a 73% test accuracy, with 4 audio clips misclassified. We note that the similarity between the sounds and syllable count notably affect the performance of the classifier, with Open and Alarm classifications commonly mixed.
3. **Voice Classification:** We train the classifier on 2 different voices, with each training example

provided by a single voice pronouncing the word Hello. We evaluate our classifier on 10 test audio clips, and see a 100% test accuracy. We see that the classifier excels at differentiating between voices, especially with one voice provided by a male, and the other provided by a female. This is likely due to the differences in frequencies that manifest more clearly in the spectrogram output.

4. **Emotion/Pitch Classification:** We train the classifier on three different emotions, provided by the same subject pronouncing the word Hello in three unique pitches, representing the emotions angry, happy, and sad. We evaluate our classifier on 15 test audio clips and observe a test accuracy of 67%, with five audio clips misclassified. We note that this result shows that it is harder to distinguish pitch/emotion from spectrogram outputs due to the fact that changing pitch does not necessarily result in clear frequency alterations in the spectrogram output.

We can draw a number of interesting conclusions from these test cases. Primarily, different training labels provided by a user that manifest in distinguishable frequency differences lead to the best model performance. This is to be expected as the trained classifier has to distinguish between very minute differences in one second audio spectrograms in order to correctly classify a test clip. We observe the best model performance on multi-syllable words and unique voices. The model tends to struggle with word and pitch differentiation when the provided test labels are too similar. In the future, we hope to evaluate the performance of App Inventor applications that use PAC to provide voice classification functionality on mobile devices.

5. CONTRIBUTIONS

In this section we summarize our methodology and aggregate the four most significant contributions from this project.

1. We explore transfer learning in the context of in-browser machine learning classification and provide a code framework for building an in-browser audio classifier.
2. We provide a code framework for up-sampling audio clips and converting audio files to spectrogram images in real time in the browser.
3. We introduce the Personal Audio Classifier (PAC), an in-browser audio classifier tool pre-trained on MobileNet that allows users to train and test custom audio spectrogram models.
4. We provide an App Inventor extension that allows users to build App Inventor applications utilizing a custom model exported from PIC.

6. REFERENCES

Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? Retrieved Month day, year, from https://arxiv.org/abs/1411.1792

Hinton, G. (2019). ImageNet classification with deep convolutional neural networks. Retrieved Month day, year, from https://dl.acm.org/citation.cfm?id=3065386

PyTorch. (n.d). *Homepage*. Retrieved December 12, 2019 from https://pytorch.org/

Tensorflow (n.d.). *Homepage*. Retrieved December 12, 2019 from https://tensorflow.org/

Tensorflow.js (n.d.). *Homepage*. Retrieved December 12, 2019 from https://tensorflow.org/js

MIT App Inventor (n.d.). *Homepage*. Retrieved December 12, 2019, from https://appinventor.mit.edu/

Personal Image Classifier (n.d.). *Homepage*. Retrieved December 12, 2019 from https://classifier.appinventor.mit.edu/

Teachable Machine (n.d.). *Homepage*. Retrieved December 12, 2019 from https://teachablemachine.withgoogle.com/

Tensorflow Audio Recognition (n.d.). *tensorflow/docs*. Retrieved December 12, 2019, from https://github.com/tensorflow/docs/blob/master/site/en/r1/tutorials/sequences/audio_recognition.md

Pytorch. (n.d.). *Pytorch Audio Transforms*. Retrieved December 12, 2019, from https://github.com/pytorch/audio/blob/master/torchaudio/transforms.py

Tensorflow. (2019). *Speech Commands*. Retrieved December 12, 2019, from https://github.com/tensorflow/tfjs-models/tree/master/speech-commands

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, Hartwig. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. Retrieved December 12, 2019, from https://arxiv.org/abs/1704.04861