Speech Recognition: Key Word Spotting through Image Recognition

Sanjay Krishna Gouda
sgouda@ucsc.edu

Salil Kanetkar
skanetka@ucsc.edu

Davan Harrison
vharriso@ucsc.edu

Manfred K Warmuth
manfred@ucsc.edu

ABSTRACT

The problem of identifying voice commands has always been a challenge due to the presence of noise and variability in speed, pitch, etc. We will compare the efficacies of several neural network architectures for the speech recognition problem. In particular, we will build a model to determine whether a one second audio clip contains a particular word (out of a set of 10), an unknown word, or silence. The models to be implemented are a CNN recommended by the Tensorflow Speech Recognition tutorial, a low-latency CNN, and an adversarially trained CNN.

The result is a demonstration of how to convert a problem in audio recognition to the better-studied domain of image classification, where the powerful techniques of convolutional neural networks are fully developed. Additionally, we demonstrate the applicability of the technique of Virtual Adversarial Training (VAT) to this problem domain, functioning as a powerful regularizer with promising potential future applications.

1 INTRODUCTION

In this project, we will work on the task of voice command recognition, following the TensorFlow Speech Recognition Challenge[1] that is being carried out on Kaggle. We will be using the Speech Commands Dataset [16] to train and evaluate our model.

We implement a few different models that each address different aspects of our problem. One consideration in constructing a Deep Neural Network solution is the large demand for memory and computing capacity on the machine running the model. For these reasons we explore model implementations that operate in environments where memory and computation resources are limited, as well as unrestricted environments.

2 AUTOMATIC SPEECH RECOGNITION

Speech Recognition is the subfield of Natural Language Processing that focuses on understanding spoken natural language. This involves mapping auditory input to some word in a language vocabulary. The dataset we plan to work with has a relatively small vocabulary of 30 words. Our proposed model will learn to identify 10 of these 30 words, label any other words as unknown, and silence as silence.

2.1 Keyword Spotting

Keyword spotting is a problem that was historically first defined in the context of speech processing. In speech processing, keyword spotting deals with the identification of keywords in utterances.
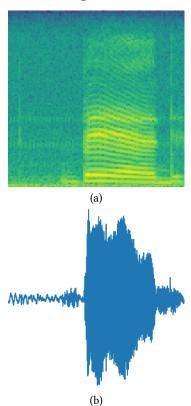

(a)

(b)

Figure 1: A comparison of the spectrogram (a) and the amplitude-vs.-time plot (b) for the same audio recording of a person saying the word "bed".

This is especially useful in picking up on a finite list of command words, for example in voice command applications. Also, by picking out keywords from an utterance, a model can pick up on salient features and use them to get an understanding of the sentence as a whole. Humans perform a version of this task when interpreting hard-to-understand speech, such as an accent which is particularly fast or slurred, or a sentence in a language we do not know very well—we do not necessarily hear every single word that is said, but we pick up on salient key words and contextualize the rest to understand the sentence. Therefore key word spotting can be an important first step in picking up on the meaning of an utterance.

2.2 Pre-processing

One of the main aims of our project is to attempt to apply convolutional neural networks for speech recognition. Typical and well-studied CNN architectures are optimized for two-dimensional image processing tasks, an application where inputs have a very different shape than in audio processing: images are two-dimensional and not far from square (aspect ratios typically range from 1 to 2), so two-dimensional convolutions with square two-dimensional kernels are highly applicable. On the other hand, audio files are extremely long one-dimensional vectors. Our inputs, for example, were single-second audio files at a sample rate of 16 kHz, so each training example was a vector in $\mathbb{R}^{16000}$. In image terms, this is equivalent to a one-pixel-high image with an aspect ratio of 16000, for which two-dimensional convolutions with square kernels are quite inapplicable.

While there is some precedent for using one-dimensional convolutional neural networks for audio processing (c.f. [9], whose abstract even begins by noting that CNNs are usually two-dimensional), we opted instead to modify the input data in order to make well-studied image processing techniques more applicable, by translating the input audio data into spectrograms. A spectrogram is a method of using discrete Fourier transforms to translate a one-dimensional time sequence, most frequently but not necessarily a segment of audio, into a two-dimensional image containing the same information, but organized in a way that makes locality meaningful in two dimensions rather than just one. Each column of the image represents the magnitude of the Fourier transform of the time sequence taken in a different contiguous window of time; the image as a whole represents a two-dimensional function from time and frequency to intensity rather than a one-dimensional function from time to absolute amplitude. Thus, in the resulting image, it is possible to directly see which frequency components change and how the frequency spectrum of the audio changes with time.

One would expect intuitively that this input format would be more conducive to analysis because the relevant information is directly visible; a comparison of the two types of sequence can be seen in Figure 1. While both figures show that there is a single loud section towards the end, only the spectrogram also shows the pitch contour of the word: each of the horizontal lines represents a single pitch component, and they all point somewhat downward. Additionally, there is biological precedent for the use of this input format: the human cochlea has different regions which are responsive to different frequencies. The result is that the input to the brain from the auditory sensory organ at any given time is comparable to a single column of the spectrogram: there are hundreds of different input neurons, each of whose firing rate is proportional to the strength of a particular frequency component.

3 LOW LATENCY CONVOLUTIONAL MODEL

Over the last decade the popularity of mobile devices has been increasing. Many applications that run on these devices use machine learning models. For example, it is now standard for new mobile devices to provide a speech-to-text service. Unfortunately, many machine learning models—especially deep neural networks—require heavy use of computation and memory resources that are lacking on mobile devices. In this section we describe a Low Latency Convolutional Neural Network that is designed to reduce its memory footprint by limiting the number of model parameters.

The model described in this section is similar to the model called "cnn-one-fstride4" in [12]. (We based our implementation of this model on a script distributed by the TensorFlow-affiliated authors of the Kaggle challenge.) However, there are some small differences between our model and cnn-one-fstride4 that arise from how we preprocessed the audio inputs. The differences are shown in Table 1.

Model	Filter size	Stride	Channels	Dense	Params
(a)	8	4	1	128	47.6k
(b)	7	3	3	100	63.8k

Table 1: Design of each model. (a) cnn-one-fstride4 described in [12]. (b) our model.

3.1 Model Design

The low latency model consists of a single convolutional layer that feeds into a Deep Neural Network. A block diagram of this model is shown in Figure 2. A high level description of our network structure is as follows:

(1) convolution layer with nonlinearity
(2) fully connected feed forward layer without nonlinearity
(3) fully connected feed forward layer with nonlinearity
(4) softmax output layer

The rest of this section will discuss the details of each layer.

3.2 Variable Initialization

Since training the CNN and DNN using Stochastic Gradient Descent is a non-convex optimization problem, parameter initialization is important.

We use Xavier Initialization to initialize our model weights. Xavier Initialization is described as follows: For an $m \times n$ dimensional matrix M, M_{ij} is assigned values selected uniformly from the distribution $[-\epsilon, \epsilon]$, where

$$\epsilon = \frac{\sqrt{6}}{\sqrt{m+n}}$$

We also initialize variables using a truncated normal distribution. In this initialization method, values are first selected from a normal distribution. Then any values that are over two standard deviations from the mean are rejected and re-drawn from the distribution. We sample values from a normal distribution that has a standard deviation of 0.01.

4 MNIST TENSORFLOW CNN

We use the MNIST sample CNN architecture **[citation needed]** for this process.It can be found at github repository of tensorflow. We will have to do some minor tweaks in the CNN's first layer so as not to get any dimension errors.
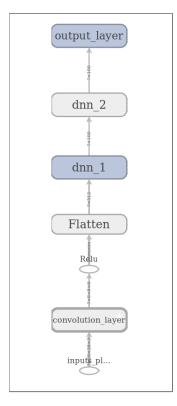
Figure 2: A view of the computation graph for the low computation convolutional model.

4.1 Model Design

Its a deep NN with two convolution layers and two maxpool layers. The structure of the CNN is as follows:

(1) First Convolutional Layer
(2) Second Convolutional Layer
(3) Densely Connected Layer
(4) Softmax output layer

A diagram of this model's architecture is shown in Figure 3. The rest of this section will discuss the details of each layer.

4.1.1 First Convolutional Layer. The first layer consists of convolution, followed by max pooling. The convolution will compute 32 features for each 5x5 patch. Its weight tensor will have a shape of [5, 5, 1, 32]. The first two dimensions are the patch size, the next is the number of input channels, and the last is the number of output channels. We will also have a bias vector with a component for each output channel.

4.1.2 Second Convolutional Layer. In order to build a deep network, we stack several layers of this type. The second layer will have 64 features for each 5x5 patch.

4.1.3 Densely Connected Layer. Now that the image size has been reduced to 7x7, we add a fully-connected layer with 1024 neurons to allow processing on the entire image. We reshape the tensor from the pooling layer into a batch of vectors, multiply by a
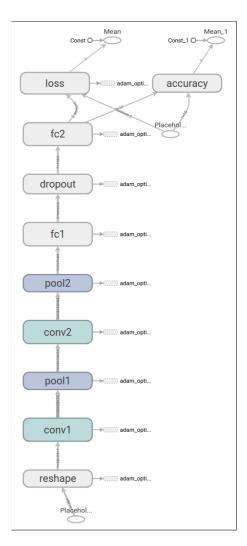

Figure 3: A view of the computation graph for the MNIST model.

weight matrix, add a bias, and apply a ReLU. To avoid overfitting we also apply dropout.

4.2 Base Model CNN

A baseline architecture is described in [12].

An additional variable to consider is the difference between time- and frequency-domain inputs. While one approach to the problem of improving non-locality in the network is to either add an attentional mechanism to an RNN or use dilated convolutions in a CNN, it is also possible to pre-process the data in order to perform learning in the frequency domain. This has a precedent in the best known speech recognition system, the human ear, which in effect performs frequency analysis as a preprocessing step.
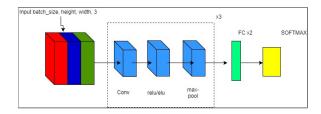
Figure 4: Shallow CNN Architecture

5 CNNS AND VIRTUAL ADVERSARIAL TRAINING

5.1 Architecture

We have also tried a different architecture in CNNs. We followed MCDNN [1] and AlexNet [8] closely in having a sequence of convolutional layers followed by non linear activations and max-pooling and eventually flattening the convolved outputs into two fully connected layers.
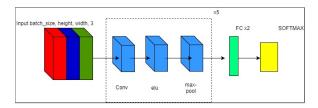

Figure 5: Deep CNN Architecture

Our first CNN model has a 5 layers where the first 3 layers are the sequence of conv-relu-maxpool (C-R-M) followed by two fully connected layer and then output through a softmax layer. We are not considering the softmax conversion as part of the neural network architecture. In the second, larger architecture, we increased the width of neural network now including 5 layers of the C-R-M sequence and two fully connected layers.

Because it was easier and quicker to train the shallow CNN, we used this for prototyping and hyperparameter tuning. On this architecture, we used different number of filters used in each convolution layer, different window size and strides and also tried different activations. Effect of activations was more apparent in the run time, this could be because of the implementations of the activation functions in TensorFlow. We have tried and tested relu, elu, sigmoid and tanh activation functions. Using sigmoid and tanh gave the worst results in terms of cost and accuracy while relu, elu activation units were much faster and more accurate for the same number of epochs. In all the tests, we used the Xavier initialization[4] method for all the weights. The convergence test followed the results form the original paper where elu was introduced by Clevert and others in [2].

With parameters obtained from these experiments, we trained the wider CNN. The reason behind doing so is quite obvious, the wider CNN takes a longer time to train and it is not feasible to search for hyperparameters in this setting. With the hope that the optimal hyperparameters of the shallow networks also worked for the deeper CNN because they follow the same architecture we

begun our experiments. The experiments on the wider CNN were done in two folds.

Analogous to the number of hidden units and layers in traditional densely connected neural networks, adding layers corresponded to finding hidden features of higher orders .Having more units in the same layer meant better feature extraction in the CNNs. Following this intuition, we have increased the number of filters used in each convolution layer. This increased the number of parameters and the training time. It eventually helped in getting better overall results which are explained in further sections

One interesting experiment we did for this setting was the use of dropout[13] as a regularization method. We tried different usages of the additional dropout layer corresponding to its addition in various layers and combinations. This is again a search for an optimal hyperparameter (let's say keep probability). We again tried this on the shallow CNN due to time constraints. The results though were counterintuitive. As we trained the model for higher training steps (epochs), we expected the model to overfit which was the case when we did not use dropout. Surprisingly, the results followed a similar pattern even with dropout but with a lower training and validation accuracy. There is some research[3, 7, 8] that uses dropout in CNNs and advices against using it.
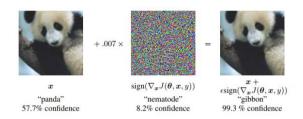

Figure 6: A demonstration of fast adversarial example generation applied to GoogLeNet (Szegedy et al., 2014a)[14] on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNetâĂŹs classification of the image. Image:[6]

5.2 Virtual Adversarial Training

In the Deep Learning book by Goodfellow et al,[5], several other regularization methods like image augmentation, noise addition, early stopping and virtual adversarial training[6, 10, 14] were mentioned. For our problem, because we converted the audio files into images and thereby converting a speech recognition problem into a viable image recognition one just to use CNN, we could use most of the usual regularization methods in CNN domain. Among these, we decided to try Adversarial Training as a regularization method for our problem. The idea behind this was that the dataset originally contained audio samples of 31 different classes but within each class the audio samples seem to follow similar distributions (this is more natural too, think of how peoples voice differ [11], this could be learned as one hidden feature in the image domain).

By adding small noise to the original distribution, we could potentially generate images that are "visually similar" but are actually

different and train the model on a mixture of original and noisy images with same labels so that the model generalizes to these small disturbances. Although we successfully generated visually similar images, we do not actually know the effect of this transformation in the audio domain.

Following the ideas in [6, 10, 14], we augmented the dataset with interesting changes. First we used OpenCV library in Python to convert the images (obtained from converting audio files) into Numpy arrays and then normalized the pixel intensities (divided by 255). We then added a perturbed version (Numpy array) of the signum output of the matrix created in the previous step to itself. We also created a new matrix whose entries $\hat{X}_{i,j,k,l} = X_{i,j,k,l} + 0.001 * std(X)$ and these two generated noisy inputs were concatenated with the training data drawn from the original matrix along with corresponding labels. When trained with the same model on this with same hyperparameters, the model has shown faster convergence (in terms of epochs) as can be seen figure 9. This could be because of generalization (regularization) or because the model has seen thrice as much data. So we sampled equal amount of training data from this
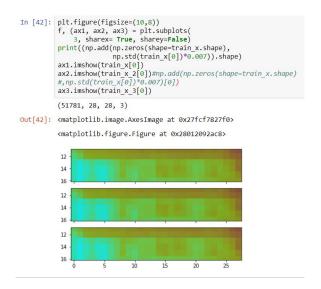
```
In [42]:  plt.figure(figsize=(10,8))
          f, (ax1, ax2, ax3) = plt.subplots(
              3, sharex= True, sharey=False)
          print((np.add(np.zeros(shape=train_x.shape),
                        np.std(train_x[0])*0.007)).shape)
          ax1.imshow(train_x[0])
          ax2.imshow(train_x_2[0])#np.add(np.zeros(shape=train_x.shape)
          #,np.std(train_x[0])*0.007)[0])
          ax3.imshow(train_x_3[0])

          (51781, 28, 28, 3)

Out[42]:  <matplotlib.image.AxesImage at 0x27fcf7827f0>

          <matplotlib.figure.Figure at 0x28012092ac8>
```


Figure 7: First image is the original example, second is the sign perturbed image and the last one is the standard deviation perturbed image

augmented dataset and carried out results. We achieved slightly better accuracies on both training and validation parts. This is significant because using dropout gave around 53% accuracy while the vanilla CNN and the CNN with adversarial training gave more than 88% accuracy. This shows that Virtual Adversarial Training (VAT) is a better regularization method, at least for this setting.

6 EXPERIMENT SET UP

6.1 Dataset

The Speech Commands Dataset [16] which was used for this project is freely available. The data consists of 65,000 labeled audio clips, sampled from thousands of different speakers from all over the world, and each recording is of different quality. Each training

example is a one-second-long audio file where a voice speaks one English word. While there are a total of 30 different possible words that can appear in a training example, our model will identify ten particular words, corresponding to the most basic voice commands: *YES*, *NO*, *UP*, *DOWN*, *LEFT*, *RIGHT*, *ON*, *OFF*, *STOP*, and *GO*. All other words will receive a label of *UNKNOWN*. Additionally, the dataset contains audio examples containing only noise, which our model must identify as *SILENCE*.

80 percent of the data was used for training, and 20 percent for validation. The raw audio data has a sample rate of 16 kHz and is one second long, resulting in 16000-item vectors. Sections of the project which use spectrograms use images resampled to a size of 28x28 grayscale pixels, while amplitude-vs.-time plots were 100x100 pixels.

6.2 Framework

The package TensorFlow was used to build our models, chosen over competing libraries and in preference to higher-level approaches such as Keras because of the increased flexibility afforded by a lower-level framework, combined with the simple fact of our greater familiarity with TensorFlow.

7 RESULTS AND DISCUSSION

7.1 CNNs and Virtual Adversarial Training

img res	batch_size	n_epochs	l_rate	tr_acc	te_acc	activation	loss
28_28	1024	5000	0.001	0.877098	0.86575	elu	0.38977
288_288	1024	5000	0.001	0.878353	0.867295	elu	0.39376
28_28 + VAT	1024	5000	0.001	0.719849	0.868531	elu	0.37929
288_288 + VAT	1024	5000	0.001	0.718285	0.860188	elu	0.38031

Figure 8: Some results using less filters on wide CNN

In figure 8 we show the results on two sets of datasets 28x28 corresponds to the pixel size 28x28 and 288x288 pixel size. It did not help to use a higher resolution image. In this table, the results correspond to a run where the model was allowed to stop training when it hits a cost threshold shown as "thres" in the image. The "VAT" indicates that the model was trained with adversarial inputs. It is evident that during the VAT training, convergence occurred much sooner than the corresponding vanilla (column n_epochs with (e) indicates exit/ending epoch). We also show some results obtained on the deeper CNN but with less number of filters.

In figures 10 and 11 we show the results in terms of training / validation accuracies vs epochs of the models that we trained and tested. The second image (figure 11) shows the behavior of the models in the first 10 epochs. It can be seen that the accuracies are much closer and are scaling at a similar rate when we use adversarial Training.

In figure 12 we can see the behavior of loss function with respect to time for all the models. Given the model capacity and training data, the effect of regularization induced by VAT is not entirely clear from the cost vs epochs graph. Although, it can be seen that the cost function with larger VAT model has converged quickly after approximately 120-150 epochs whereas the other models appear to have converged a few tens of epochs later.

The behavior of cost function in the first 10 epochs is seen image 13.

```
Run 1 thres <= 0.1
img res        batch_size  n_epochs  l_rate  tr_acc    te_acc    activation  loss

28_28          1024        168(e)    0.009   0.973021  0.926078  elu         0.0906
288_288        1024        171(e)    0.009   0.965238  0.918894  elu         0.0942
28_28 + VAT    1024        105(e)    0.009   0.969892  0.921984  elu         0.0954
288_288 + VAT  1024        97 (e)    0.009   0.960991  0.915109  elu         0.0978

Run 2 thres <= 0.05

img res        batch_size  n_epochs  l_rate  tr_acc    te_acc    activation  loss

28_28          1024        411(e)    0.009   0.986115  0.921675  elu         0.0539
288_288        1024        440(e)    0.009   0.983778  0.922756  elu         0.0535
28_28 + VAT    1024        243(e)    0.009   0.981424  0.918198  elu         0.0549
288_288 + VAT  1024        236(e)    0.009   0.977743  0.915572  elu         0.0551
```

Figure 9: Adversarial Training Results and Comparison with Vanilla CNN
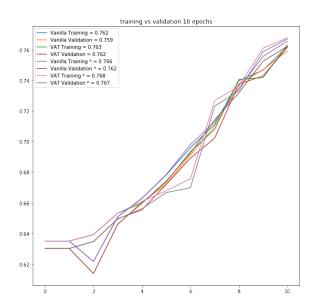
Figure 10: increase in training and validation accuracy over the first ten epochs for the low-latency convolution and VAT models: a zoomed-in version of Figure 12

7.2 Tensorflow MNIST and Basic CNN

We also tried the CNN architecture described in the MNIST Tutorial by Tensorflow[15] on two different conversions of our training data to image format. When trained on images of the amplitude-vs.-time graph of the audio file, the network attained a validation accuracy of 50.3%, compared to 52.5% on log spectrogram images, suggesting that our intuition was correct regarding the most informative input formats.

7.3 Low-latency Convolutional Model

Beginning with the basic low-latency convolutional model described above, we tried varying a number of different hyperparameters and comparing results.

Optimization Algorithm. Because our optimization problem is nonconvex, choices such as the initialization and the precise method of optimization used begin to matter. With this in mind, we compared two different approaches to gradient descent: the adaptive
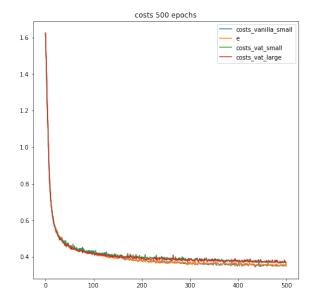

Figure 11: decrease of costs over 500 epochs for the low-latency convolution and VAT models: a zoomed-out version of Figure 13
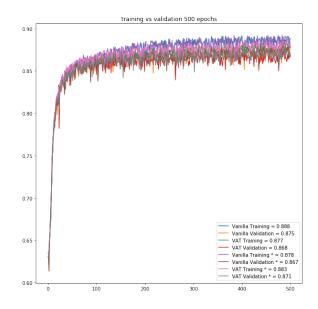

Figure 12: increase of training and validation accuracy over 500 epochs for the low-latency convolution and VAT models: a zoomed-out version of Figure 10

ADAM optimization algorithm vs. a typical stochastic gradient descent. Figure 14 compares the evolution of both loss and accuracy over time between Adam and SGD. By both metrics, Adam converges noticeably faster without substantially changing the final point of convergence.

Initialization. The nonconvexity of our problem also led us to consider alternate initialization methods. We compared the Xavier
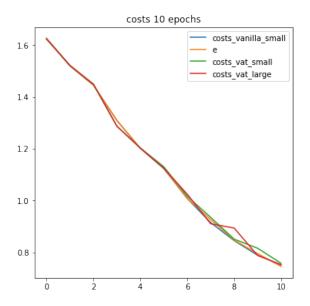
Figure 13: reduction of cost over the first ten epochs for the low-latency convolution and VAT models: a zoomed-in version of Figure 11
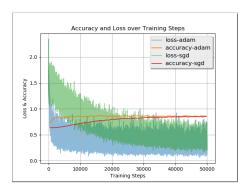

Figure 14: the evolution of training cross-entropy loss (blue and green) and validation accuracy (red and orange) compared between Adam and SGD optimization; Adam converges faster than SGD but reaches the same results

random initialization method (detailed elsewhere, but in brief, values are initialized uniformly within a small interval) to the *de facto* standard truncated normally-distributed initialization. Figure 15 compares the evolution of both loss and accuracy over time between models differing only in the choice between Xavier and truncated normal initialization. In this case even more so than in the case of the choice of optimization algorithm, there is a marked difference in the rate of convergence, but here there may also be a difference in the final result. It is not clear whether the network with truncated normal initialization has fully converged by the end of the training, while the network with Xavier initialization has spent the majority of the same amount of time hovering around the same loss and accuracy, i.e. already converged. Furthermore, by the end of this trial, the accuracy and loss of the network with truncated normal
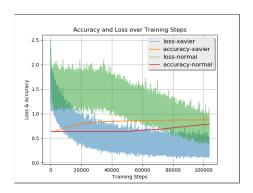

Figure 15: the evolution of training cross-entropy loss (blue and green) and validation accuracy (red and orange) compared between Xavier and truncated normal initialization; Xavier converges much faster and may attain better results

initialization are markedly worse than those of the network with Xavier initialization.

Spectrogram Parameters. Because we converted the problem of audio recognition to the image domain using spectrograms, the input stage of the pipeline also provided a number of different parameters to tune. A spectrogram is generated by taking subsequences of a certain fixed length from the input sequence, multiplying them by a windowing function[2], taking the discrete Fourier transform of each windowed subsequence, and pasting them together in the form of an image. The way those subsequences are chosen does make a difference to the final results. We tried varying three different parameters of the spectrogram: the number of buckets used for frequency counting, the size of the window, and the spacing between different windows. The plots for all of these parameters specify the number of
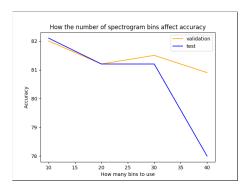

Figure 16: effect of the number of frequency-counting buckets on the accuracy of the low-latency convolution model. The model did not benefit from the increase in available data caused by increasing the number of buckets.

One might expect that increasing the number of buckets in the frequency counting (that is, the number of sections into which the

---

[2]a regularization method without which spurious high-frequency components will invariably be created in the output, obscuring more important parts of the signal

frequency axis was divided, or the pixel height of the spectrogram) would help the network discriminate better due to the increase in the amount of data contained in each training example, but in fact the effect was the opposite: across the whole range of values we tried, from 10 to 40 frequency buckets, increasing the number of buckets only worsened the performance of the network. This can be seen in Figure 16.

The stride between windows was found to have little effect within a certain range. When the stride was too low or too high, however, results began to worsen. This is shown in Figure 18. These results are to be expected, because excessively low stride means that the windows overlap significantly, creating a large amount of redundant data which the network must sort through. Likewise, excessively high stride naturally results in missing out on important parts of the data when certain components happen to fall in the gaps between the windows. This could cause certain features to not show up in the spectrogram at all, in which case it is of course impossible for the network to learn them. Like the stride length,
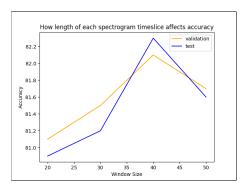

Figure 17: effect of the spectrogram window size on the accuracy of the low-latency convolution model. There is a local optimum, as there was for stride in Figure 18.

the size of the convolutional window had an optimum value about which the final accuracy fell off noticeably. This behavior is shown in Figure 17. One would expect that the only effect of changing the window size would be that the meaning of the stride value changes: since stride remains fixed, a large window will overlap substantially with adjacent windows, while a small window will leave gaps. So the entire observed variation in final accuracy with window size can likely be attributed to the same reasons as the variation in accuracy with stride.

Noise. In addition to varying hyperparameters, we also tried adding noise to the audio signal prior to computing the spectrogram. This is crucial for real-world applications because the desired target words will never be spoken completely in the absence of competing stimuli. There will always be background noise from many sources and in many forms, such as the pure white noise generated by the thermal voltage in the analog microphone amplifier, the unordered crashes of nearby remodeling, or the highly-ordered distraction of an irrelevant conversation. Our sample dataset included samples of the audio interference from various sources of noise, such a dishwasher, which could be mixed in at random to simulate background noise. We tried this at various different noise-to-signal
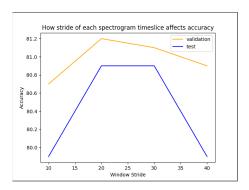

Figure 18: effect of the spectrogram window stride on the accuracy of the low-latency convolution model. For low values of stride, there is too much redundancy, while larger values result in lost information.
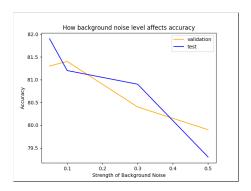

Figure 19: effect of added background noise on the final accuracy of the low-latency convolution model. The horizontal axis is signal-noise ratio in linear units.

ratios over a range from 0 to 0.5 (corresponding to signal-noise-ratios of infinity up to 6 dB), with results as shown in Figure 19. Of course, increased background noise did degrade the performance of the network, but the effect was comparatively slight: on the order of a few percentage points accuracy. This suggests that this approach is fairly robust to noise; even humans' accuracy at comparable tasks is noticeably degraded by low signal-to-noise ratios.

CONCLUSION

In this project we tackled the speech recognition problem by applying different CNN models on image data formed using log spectrograms of the audio clips. We also successfully implemented a regularization method "Virtual Adversarial Training" that achieved a maximum of 92% validation accuracy on 20% random sample of the input data.

The significant work done in this project was the demonstration of how to convert a problem in audio recognition into the better-studied domain of image classification, where the powerful techniques of convolutional neural networks are fully developed. We also saw, particularly in the case of the low-latency convolution model, how crucial good hyperparameter tuning is to the accuracy of the model. A great number of hyperparameters must be tuned,

including the many choices that go into network architecture, and any of the hyperparameters, poorly chosen, can make or break the overall performance of the model. Another contribution was the use of adversarial training to provide a regularization effect in audio recognition; this technique improved results relative even to well-established techniques such as dropout, and therefore has promising applications in the future.

REFERENCES

[1] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. 2012. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 3642–3649.

[2] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).

[3] Yarin Gal and Zoubin Ghahramani. 2015. Bayesian convolutional neural networks with Bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158* (2015).

[4] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 249–256.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[6] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[7] Alex Kendall and Roberto Cipolla. 2016. Modelling uncertainty in deep learning for camera relocalization. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 4762–4769.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[9] Gabriele Manganaro, Paolo Arena, and Luigi Fortuna. 1999. A One-Dimensional Discrete-Time CNN Chip for Audio Signal Processing. In *Cellular Neural Networks*. Springer, 189–204.

[10] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. 2015. Distributional smoothing with virtual adversarial training. *arXiv preprint arXiv:1507.00677* (2015).

[11] Lindasalwa Muda, Mumtaj Begam, and Irraivan Elamvazuthi. 2010. Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques. *arXiv preprint arXiv:1003.4083* (2010).

[12] Tara N. Sainath and Carolina Parada. 2015. Convolutional Neural Networks for Small-footprint Keyword Spotting. (2015).

[13] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research* 15, 1 (2014), 1929–1958.

[14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[15] Tensorflow. [n. d.]. Deep MNIST for Experts. *Description available from https://www.tensorflow.org/versions/r1.2/get_started/mnist/pros* ([n. d.]).

[16] Pete Warden. 2017. Speech Commands: A public dataset for single-word speech recognition. *Dataset available from http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz* (2017).