Deep Learning for Classification of Speech Accents in Video Games

Sergio Poo Hernandez and **Vadim Bulitko**
Computing Science
University of Alberta
Edmonton, AB
{pooherna | bulitko}@ualberta.ca

Shelby Carleton
English and Film Studies
University of Alberta
Edmonton, AB
scarleto@ualberta.ca

Astrid Ensslin and **Tejasvi Goorimoorthee**
Humanities Computing
University of Alberta
Edmonton, AB
{ensslin | tejasvi}@ualberta.ca

Abstract

In many video games, a wide range of non-playable characters make up the worlds players inhabit. Such characters are often voiced by human actors and their accents can have an influence on their perceived moral inclination, level of trustworthiness, social class, level of education and ethnic background. We use deep learning to train a neural network to classify speech accents. Such a machine-learned tool would provide game developers with an ability to analyze accent distribution in their titles as well as possibly help screening voiceover actors applying for a role. To make the approach accessible we used a readily available off the shelf deep network and commodity GPU hardware. Preliminary results are promising with a 71% test accuracy achieved over two accents in a commercial video game.

1 Introduction

Modern video games often feature numerous non-playable characters (NPCs) that populate the in-game world, contributing to the atmosphere, gameplay and storytelling. Such characters are usually available to interact with the player and are frequently voiced by well-known actors (e.g., Martin Sheen in *Mass Effect 2* (BioWare 2010)). As in movies, different accents in the same language (e.g., English) contribute to an ethnic, social and moral image of an NPC. Thus it is important for game developers to be aware of and control assignment of accents to in-game characters. Having a fast and low-cost way of determining an accent of a voiceover can help developers screen sent-in audition files or take an inventory of accents within a development prototype.

In this paper we demonstrate how machine learning can be used to automatically classify speech accents in videogame voiceovers. The approach is designed to be accessible to small-scale game developers and individual researchers in the field of game studies. Specifically, we train an off-the-self deep neural network on commodity hardware. The network achieved a 71% test accuracy over American and British accents in the commercial videogame *Dragon Age: Origins* (BioWare 2009).

The rest of the paper is organized as follows. We formulate the problem precisely in Section 2, then discuss related work in Section 3. We then present our approach in Section 4 and detail results of an empirical evaluation in Section 5. The paper is concluded with a discussion of the current shortcomings and the corresponding future work.

2 Problem Formulation

The problem is mapping from an audio file containing speech to an accent label from a pre-determined set. A file is assumed to have a single speaker whose accent is consistent throughout the file. We evaluate performance of such a mapping by measuring its accuracy on a test set of files. The objective is then to increase the test accuracy while keeping the approach accessible to game developers as well as researchers from different disciplines.

3 Related Work

In speech recognition Graves, Mohamed, and Hinton (2013) used recurrent neural networks (Goodfellow, Bengio, and Courville 2016) to recognize phonemes in the TIMIT database (Garofalo et al. 1993). Other work in phoneme classification in speech signals using convolutional neural networks (CNN) (Palaz, Collobert, and Doss 2013; Song and Cai 2015; Zhang et al. 2017) used logarithmic mel-filter-bank coefficients and hybrid networks composed of a CNN and a recurrent neural networks (RNN). Their primary task is different from ours in that they are identifying phonemes to recognize words as opposed to accents. Yet other work on phoneme recognition (Hinton et al. 2012) highlighted the importance of weight initialization when recognizing phonemes. Once again the problem they were tackling is substantively different from ours. Research by Espi et al. (2015) of acoustic event detection emphasized the importance and feasibility of local feature extraction in detecting and classifying non-speech acoustic events occurring, for instance, in conversation scenes. This work is indirectly related to our current work as it does not label accents in a conversation, however it can be combined with our approach to detect and remove sections of the audio without speech.

Work has been done on detecting emotions in speech through spectrograms (Huang et al. 2014; Badshah et al. 2017). While this is not the task we were trying to solve, it is similar in its use of spectrograms as the input to their neural network.

Recently we explored training a neural network on an existing (non-video-game) accent database and then used the trained network to detect accents in audio files from a video game(Ensslin et al. 2017). They thought that training on curated accent database would yield a better classifier. There were two problems with their approach. First it required access to a separate accent database. Second, their test accuracy was poor. Our approach is similar but trains on videogame audio files directly and yields better test accuracy.

4 Our Approach

To keep our approach accessible to a broad set of game developers and researchers, we used a common off the shelf deep neural network: AlexNet (Krizhevsky, Sutskever, and Hinton 2012). This approach has yielded state-of-the-art results when classifying bird species using their song (Knight et al. 2018).

4.1 Converting Audio to Images

As AlexNet was originally designed to classify images, we converted audio files to spectrograms in a fashion similar to our previous work (Ensslin et al. 2017). Specifically, each spectrogram consisted of four different image quadrants computed by Algorithm 1 as follows.

The audio file S gets partitioned into m parts of w seconds each (line 3). For each part s_x, we apply the Fast Fourier Transform to it, resulting in a sequence of amplitudes A (line 4). We remove all amplitudes for frequencies below $f_{\min}$ and above $f_{\max}$ (line 5). We then partition the remaining frequency range $[f_{\min}, f_{\max}]$ into b linearly (if $\mathcal{L}_f$ holds) or logarithmically spaced segments (line 6). For each segment $B(y)$ we sum the corresponding amplitudes into the scalar a_y (line 8). We then map a_y to a spectrogram pixel $I(x, y)$ using a color mapping $\mathcal{C}$ (line 9). Optionally we take a logarithm of the amplitude a_y (if $\mathcal{L}_a$ is false).

Algorithm 1: Create spectrogram quadrant

```
input : $S, f_{\min}, f_{\max}, w, b, \mathcal{L}_f, \mathcal{L}_a, \mathcal{C}$
output: spectrogram image $I$
1 $m \leftarrow \left\lceil \frac{|S|}{w} \right\rceil$
2 for $x \in \{1, 2, \dots, m\}$ do
3     $s_x \leftarrow x^{\text{th}}$ window from $S$
4     $A \leftarrow \texttt{fft}(s_x)$
5     $A \leftarrow A|_{f_{\min}}^{f_{\max}}$
6     $B \leftarrow \begin{cases} \texttt{linspace}(f_{\min}, f_{\max}, b), & \text{if } \mathcal{L}_f \\ \texttt{logspace}(f_{\min}, f_{\max}, b), & \text{otherwise} \end{cases}$
7     for $y \in \{1, 2, \dots, b\}$ do
8         $a_y \leftarrow \sum A|_{B(y)}$
9         $I(x, y) \leftarrow \begin{cases} \mathcal{C}(a_y), & \text{if } \mathcal{L}_a \\ \mathcal{C}(\log(a_y)), & \text{otherwise} \end{cases}$
```

Figure 1 illustrates the process on a simple piano-roll audio file. The top-left quadrant of the image is produced by setting both $\mathcal{L}_f$ and $\mathcal{L}_a$ to true and therefore uses linearly spaced frequency-range segments B. The top-right quadrant has $\mathcal{L}_a$ set to false and thus runs a logarithm on cumulative amplitude a_y before converting it to the RGB color via color mapping $\mathcal{C}$ (which maps lower amplitudes to colder/blue colors and higher amplitudes to warmer/red colors). The bottom-left quadrant spaces frequency-range segments logarithmically (i.e., $\mathcal{L}_f$ is set to false). Finally, the bottom-right quadrant uses logarithmically spaced frequency-range segments as well as applies logarithm to amplitudes.

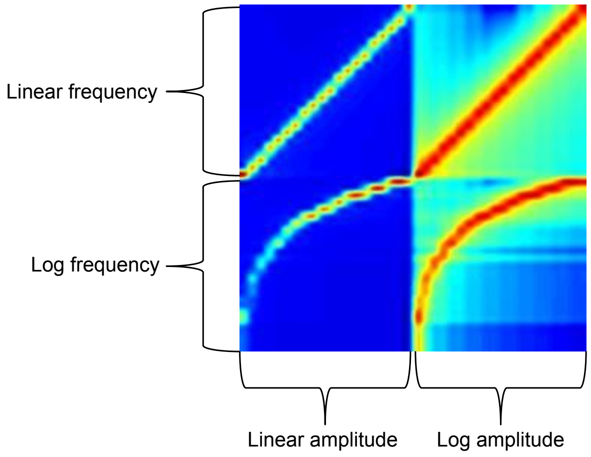

Figure 1: A composite spectrogram of a piano-roll audio.

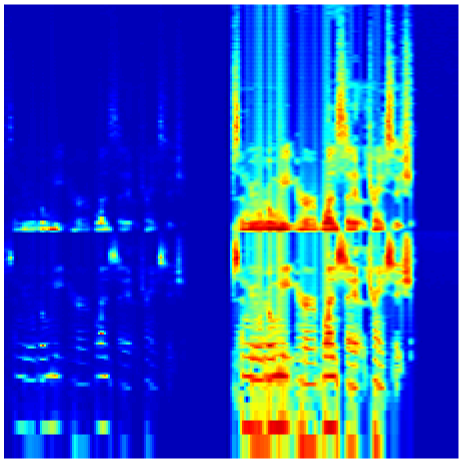

Figure 2: A composite spectrogram of a voiceover file with an American accent.

Each quadrant is a color image of the height of b pixels and the width of m pixels. The resulting composite spectrogram thus has $2b$ rows and $2m$ columns. Figure 2 shows a composite spectrogram of an actual videogame audio file.

4.2 Training and Testing the Network

Algorithm 1 converts a set of audio files $\{S_k\}$ to a set of spectrograms I_k. As the audio files increase in duration, the width of each quadrant (m pixels) will necessarily increase in order to keep the same temporal resolution. Since off the shelf deep neural networks tend to require the input image to be of a fixed size, we divided each audio file S_k into segments $\{S_k^i\}$ of up to s seconds. This ensures that the spectrogram maintains a temporal resolution of at least $\frac{m}{s}$ pixels per second of audio.

Our original dataset of audio files and their accent labels $\{(S_k, l_k)\}$ thus becomes a dataset of audio segments each of which inherits the accent label of the original file: $\{(S_k^i, l_k)\}$. Once converted to spectrograms by Algorithm 1 these become $\{(I_k^i, l_k)\}$.

To get robust results, and avoid overfitting the data, we conduct the training and testing process in the standard fashion with T independent trials. On each trial t, we split the dataset $\{(S_k, l_k)\}$ into α (complete) audio files to be used for training and $1-\alpha$ which are used for testing: $\{(S_k, l_k)\} = \mathbf{S}^t_{\text{train}} \cup \mathbf{S}^t_{\text{test}}$ with $|\mathbf{S}^t_{\text{train}}| = \lfloor \alpha |\{(S_k, l_k)\}| \rfloor$. Expressed at the level of spectrograms of audio segments we have $\{(I_k^i, l_k)\} = \mathbf{I}^t_{\text{train}} \cup \mathbf{I}^t_{\text{test}}$.

On each trial t, we train a neural network on $\mathbf{I}^t_{\text{train}}$ using three hyperparameters: the number of epochs, the batch size for stochastic gradient descent and the learning rate. Once the network $\mathcal{N}_t$ is trained we freeze its weights and test it on $\mathbf{I}^t_{\text{test}}$. The per-segment accuracy of the trained network is the percentage of audio file segments for which the accent level output by the network matched that in the test set:

$$A_t^{\text{per-segment}} = \frac{|\{(I_k^i, l_k) \in \mathbf{I}^t_{\text{test}} \mid \mathcal{N}_t(I_k^i) = l_k\}|}{|\mathbf{I}^t_{\text{test}}|}.$$

The per-segment accuracy of the net is then averaged over all T trials: $A^{\text{per-segment}} = \text{avg}_t\, A_t^{\text{per-segment}}$.

We also calculate per-file accuracy. For that we run the network on all segments comprising an audio file from the test set and take the majority vote on the labels the network produces.[1] Thus we define $\mathcal{N}_t(I_k)$ as the majority vote of the network's classifications of each segment: $\mathcal{N}_t(I_k^i)$. Then the per-file accuracy is defined as:

$$A_t^{\text{per-file}} = \frac{|\{(I_k, l_k) \in \mathbf{I}^t_{\text{test}} \mid \mathcal{N}_t(I_k) = l_k\}|}{|\mathbf{I}^t_{\text{test}}|}.$$

As before: $A^{\text{per-file}} = \text{avg}_t\, A_t^{\text{per-file}}$.

5 Empirical Evaluation

In this section we will present a specific implementation of our approach and an evaluation of its performance on audio files from a commercial video game.

5.1 Data Collection

We used voiceover audio files captured from *Dragon Age: Origins* (BioWare 2009) — a game with a wide variety of accents and characters.

The background music in the game was turned off so only the speech is present. We tried to capture as many NPCs as possible and only one recording per NPC was used to form our dataset. The audio files were separately labeled by three individuals Each individual listened to each audio file and labeled its accent. Then the multiple labelers compared their labels and debated any differences until a consensus was reached.[2] This process resulted in 295 audio files such that each file contained a single speaker labeled with a single accent label: 147 with an American accent and 148 with a British accent.

The audio files were from 2 to 40 seconds in duration. Using a segment length of 3 seconds, we created a data set $\{(I_k^i, l_k)\}$ of 1100 segment spectrograms. The majority classification average of this set is 51.1%; this means that if we classified the data by selecting the label with the most elements we would classify 51.1% of the segments correctly.

5.2 Implementation Details

In our implementation of the approach we used the `audioread` and `fft` functions in MATLAB to read each audio file in, average the two channels of the clip and perform the Fast Fourier transform. The spectrogram was converted to an RGB image using `jet(100)` colormap in MATLAB. The composite spectrogram (four quadrants) was then resized to 227×227 pixels for input to the network using the `imresize` function in MATLAB, which uses a bi-cubic interpolation method.

We used a version of AlexNet that is available for download as a MATLAB add-on `alexnet`.[3] We trained it with the MATLAB neural network toolbox via `trainNetwork` function using stochastic gradient descent with a learning rate of 0.01 with a drop learn rate factor of 0.1. We ran all experiments on an Intel Core i7 980X workstation with a six-core 3.33 GHz CPU and 24Gb of RAM. It hosted two Nvidia Maxwell-based Titan X GPUs with 12Gb of video-RAM each. This allowed us to run two learning trials in parallel (one trial per GPU).

5.3 Single-accent Classification

Spectrograms for audio segments were divided into a training set, used to train the network, and a testing set. The training set contained 75% of the audio files of each class, while the remaining 25% were used in the testing set. We made sure that all spectrograms belonging to the same audio file are in the same set (i.e., $\forall k \nexists i_1, i_2 \left[(I_k^{i_1}, l_k) \in \mathbf{I}_{\text{train}} \;\&\; (I_k^{i_2}, l_k) \in \mathbf{I}_{\text{test}}\right]$).

There were four control parameters we varied for the data preparation and network training: the number of epochs and the batch size which relate to the network training configuration; the number of frequency filters b and the time window size w which determine the specification of the spectrograms. We did not know the best combination for the dataset at hand so we conducted a parameter sweep. To reduce the sweep time we factored the parameter space into a product

[1]If an equal number of segments was labeled with the same accent then we break the tie between the labels in favor of the label of the earliest such segment. For instance, if a five-segment audio file is labeled by the network as [British, British, American, American, Spanish] then we break the tie between British and American in favor of British. This method was used since we initially assumed there are no ties, so we always select the first most frequent segment-level label.

[2]If no agreement could be achieved then the audio file was excluded from the set.

[3]We used MATLAB for training because of the convenience of parameter sweep and data analysis as well as access to an existing code base.

of two subspaces: one defined by the number of epochs and the batch size and the other defined by the number of frequency filters and the time window size.

We then fixed a single pair of parameters from the second subspace and tried all $4 \cdot 5$ combinations of parameters from the first subspace. For each try we ran four independent trials of training and testing, splitting the dataset into training and testing partitions randomly on each trial. Test accuracy averaged over the four trials defined the quality of the parameter pair from the first subspace, given the fixed values from the second subspace. We then picked the highest-quality parameter pair from the first subspace and, keeping it fixed, swept the second subspace trying all of its $4 \cdot 6$ pairs. If the best quality and the second-subspace parameters found matched those found before, we stopped the process. Otherwise, we picked another (untried) parameter pair from the second subspace and repeated the steps above.

This factored sweep can stop short of finding the global optimum in the overall four-dimensional parameter space. On the positive side, it is likely to be faster as it sweeps a single two-dimensional subspace at a time. In our evaluation the process stopped after 4 iterations, each consisting of two subspace sweeps. Thus only $4 \cdot (4 \cdot 5 + 4 \cdot 6) = 176$ parameter combinations were tried in total (as opposed to $4 \cdot 5 \cdot 4 \cdot 6 = 480$ that would be required to sweep the original space).

Table 1: Control parameter space.

Parameter	Values
number of epochs	$\{10, 50, 100, 200\}$
batch size	$\{3, 5, 10, 50, 100\}$
window size w	$\{0.05, 0.025, 0.01, 0.001\}$ seconds
frequency filters b	$\{50, 75, 113, 227, 250, 280\}$

We ran four trials per parameter combination and reported the average accuracy of the four trials. We found that the best parameters were 280 frequency filters and window size of 0.05 seconds, 50 epochs and a batch size of 5; these yield a test accuracy of $A^{\text{per-segment}} = 63.25 \pm 4\%$.

We then locked in the control parameters listed above and ran 10 additional trials. The resulting confusion matrix averaged over the four trials is listed in Table 2, left.

File-level Labeling. We then examined test accuracy at the file level. As described earlier in the paper file-level test accuracy $A_t^{\text{per-file}}$ is computed by training the network to classify segments but then labeling the file with a majority vote over the segments. For instance, if an audio file was split into 7 segments, and the network labeled 3 of them as American accent and 4 as British accent, we would label the entire audio file as British.

We re-ran the experiment, the best parameters for this run were 250 frequency filters and a window size of 0.05 seconds, 50 epochs and a batch size of 5, which yielded an average accuracy of $A^{\text{per-file}} = 68 \pm 3.6\%$. We ran 10 additional trials to compute the confusion matrix: Table 3, left.

Segment Duration. Given that some audio files were shorter than 3 seconds, we also tried the segment duration of 1 second. For per-segment accuracy, the best parameters were 280 frequency filters and a window size of 0.05 seconds, 50 epochs and a batch size of 5. These parameters yielded an average accuracy of $A^{\text{per-segment}} = 63.5 \pm 3\%$ over four trials which is similar to that with 3-second segments. The corresponding confusion matrix (over additional 10 trials) is found in Table 2, right. For per-file accuracy the best parameters were 75 frequency filters and a window size of 0.01 seconds with a test accuracy of $A^{\text{per-file}} = 71 \pm 4.5\%$ averaged over 4 trials. The corresponding confusion matrix computed over 10 additional trials is found in Table 3, right.

6 Current Challenges and Future Work

Humans may use certain speech features (e.g., the way the speaker pronounces 'r') to identify accents in an audio file. Those features are present only occasionally and thus short audio files can be mislabeled by humans. Furthermore, human labelers can be inconsistent in their labels. Such problems with the dataset may reduce test accuracy. Future work will scale up the number of human labelers as well as the length of the files to produce a more accurate/consistent dataset. We will also attempt to train a network for more than two accents, including fantasy accents. We will also extend the space of control parameter space to gain a better understanding on how they affect the accuracy of the network.

It will also be of interest to segment audio files in a content-aware way (instead of fixed 1- or 3-second segments). Doing so may also allow the classifier to automatically remove silent parts of an audio file and thus avoid dilution of dataset with meaningless data. Future work will compare the spectrogram-based representation of an audio file to mel-filter-bank coefficients (Palaz, Collobert, and Doss 2013; Song and Cai 2015; Zhang et al. 2017) as well as use other neural networks such as VGG (Simonyan and Zisserman 2014).

Finally, measuring portability of a deep neural accent detector across games as well as its sensitivity to background music is a natural direction for future work.

7 Conclusions

Accent classification is an important task in video-game development (e.g., for quality control and pre-screening for voiceover auditions). In the spirit of reducing game-production costs we proposed and evaluated an approach for doing so automatically, via the use of deep learning. To keep our approach low-cost and accessible to practitioners outside of Computing Science, we used a readily available off the shelf deep neural network and a standard deep learning method. We evaluated our approach on a database of voiceover files from a commercial video game *Dragon Age: Origins* where the network achieved the test accuracy of 71%. These results demonstrate a promise of using off the shelf deep learning for game development and open a number of exciting follow-up directions.

8 Acknowledgments

We appreciate the support from Kule Institute for Advanced Study (KIAS), the Social Sciences and Humanities Council

Table 2: The confusion matrix for per-segment labeling. **Left:** 3-second segments. **Right:** 1-second segments.

	Actual	
Classified as	British	American
British	63.6%	35.3%
American	36.4%	64.7%

	Actual	
Classified as	British	American
British	64.3%	38.4%
American	35.7%	61.6%

Table 3: The confusion matrix for per-file labeling. **Left:** 3-second segments. **Right:** 1-second segments.

	Actual	
Classified as	British	American
British	67.4%	31.2%
American	32.6%	68.8%

	Actual	
Classified as	British	American
British	71.2%	29.1%
American	28.8%	70.9%

of Canada (SSHRC) via the Refiguring Innovation in Games (ReFiG) project, the Alberta Conservation Association, the Alberta Biodiversity Monitoring Institute, and Nvidia,.

References

Badshah, A. M.; Ahmad, J.; Rahim, N.; and Baik, S. W. 2017. Speech Emotion Recognition from Spectrograms with Deep Convolutional Neural Network. In *Proceedings of 2017 International Conference on Platform Technology and Service (PlatCon)*, 1–5.

BioWare. 2009. Dragon Age: Origins.

BioWare. 2010. Mass Effect 2.

Ensslin, A.; Goorimoorthee, T.; Carleton, S.; Bulitko, V.; and Poo Hernandez, S. 2017. Deep Learning for Speech Accent Detection in Videogames. In *Proceedings of the Experimental AI in Games (EXAG) Workshop at the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 69–74.

Espi, M.; Fujimoto, M.; Kinoshita, K.; and Nakatani, T. 2015. Exploiting spectro-temporal locality in deep learning based acoustic event detection. *EURASIP Journal on Audio, Speech, and Music Processing* 2015(1):26.

Garofalo, J. S.; Lamel, L. F.; Fisher, W. M.; Fiscus, J. G.; Pallett, D. S.; and Dahlgren, N. L. 1993. The DARPA TIMIT acoustic-phonetic continuous speech corpus cdrom. *Linguistic Data Consortium*.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Graves, A.; Mohamed, A.-R.; and Hinton, G. 2013. Speech Recognition with Deep Recurrent Neural Networks. In *Proceedings of 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 6645–6649.

Hinton, G.; Deng, L.; Yu, D.; Dahl, G. E.; Mohamed, A.-r.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T. N.; et al. 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29(6):82–97.

Huang, Z.; Dong, M.; Mao, Q.; and Zhan, Y. 2014. Speech Emotion Recognition Using CNN. In *Proceedings of the 22nd ACM International Conference on Multimedia*, 801–804.

Knight, E. C.; Poo Hernandez, S.; Bayne, E. M.; Bultiko, V.; and Tucker, B. V. 2018. Pre-processing spectrogram parameters improve the accuracy of birdsong classification using convolutional neural networks. *Under review*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, 1097–1105.

Palaz, D.; Collobert, R.; and Doss, M. M. 2013. Estimating Phoneme Class Conditional Probabilities from Raw Speech Signal using Convolutional Neural Networks. *arXiv preprint arXiv:1304.1018*.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Song, W., and Cai, J. 2015. End-to-End Deep Neural Network for Automatic Speech Recognition. *Technical Report*.

Zhang, Y.; Pezeshki, M.; Brakel, P.; Zhang, S.; Bengio, C. L. Y.; and Courville, A. 2017. Towards End-to-End Speech Recognition with Deep Convolutional Neural Networks. *arXiv preprint arXiv:1701.02720*.