Machine Learning Engineer Nanodegree

Capstone Project

Roger Man
November 8th, 2019

I. Definition

Project Overview

ASR (automatic speech recognition) is described by Techopedia as the use of software-based techniques to identify and process human voice. In most cases, ASR typically is used to convert spoken words into text. The transcription output can then be used in many practical domains, the most popularised being virtual assistants such as Siri and the Google Assistant. As businesses and academic institutions begin to recognise the value of unstructured data, transcription will serve as an important feature to a wide variety of machine learning applications.

However, most transcription services available today require a prior language configuration before being able to provide a transcription. In an increasingly globalised world, automatic detection of language from audio will become a necessity.

Using Convolutional Neural Network architectures in audio classification is an effective and increasingly common approach to audio classification. In this project, I hope to use open-source datasets and software packages in order to automate the detection of language from audio using Deep Neural Networks.

Problem Statement

Since widely available speech-to-text transcription APIs currently require a language to be pre-selected before use, there is an opportunity to automate this process.

The solution is to create a model which identifies language from speech audio. This will be a multiclass classification using audio-based features and the target can be one of many languages.

I tackled this problem in a number of steps. Firstly I performed an exploratory data analysis to determine the underlying characteristics of the data in order to find a robust dataset sample for this exercise. The next step was to understand the different transformations available for audio classification. Having settled on using mel spectograms as a feature, preprocessing the audio and extracting features into train/validation/test sets. I selected transfer learning as an approach to classifying audio in order to reduce the training time whilst still capturing more complex features from an image. Convolutional Neural Networks typically perform very well in this image classification space. Using the higher level bottleneck features obtained from the ResNet50 model, I trained a classification model using a simple Deep Neural Network.

Finally, I evaluated the performance of the final model with the test dataset.

Metrics

For this project, I will simply be using accuracy as an evaluation metric. This is the number of correct predictions divided by the total number of predictions.

Accuracy can sometimes be misleading. In the case of imbalanced classes, we may get a high accuracy without solving the problem. I have taken care to ensure the labels of my dataset will be uniformly distributed, therefore accuracy will indeed be a good indicator of performance. In practice, we are likely to look at precision or recall to understand where we are misclassifying between classes. However, this would be just an aid to help us improve accuracy.

II. Analysis

Data Exploration

The Common Voice dataset contains 1945 hours of validated recordings in 29 different languages. The English dataset alone is more than 30gb in storage size. Volunteers record voice clips by reading from a bank of donated sentences. These clips are validated through a voting process, only allowing audio clips with 2 or more votes through to the Common Voice dataset. This dataset is available under a CCO licence. Enhancement and reuse of the works is permitted for any purposes without restriction under copyright or database law.

There is a master tsv (tab separated values) file for each language containing the associated metadata for the audio clips, including the following fields in string format:

- **client_id**: unique identifer of the contributor
- **path**: path to the audio file from the clips folder
- **sentence**: sentence spoken by the contributor
- **up_votes**: number of up votes from other contributors
- **down_votes**: number of down votes from other contributors
- **age**: age grouped by decade
- **gender**: male, female or other
- **accent**: regional accent if reported

The audio clips are all in a consistent mp3 format with a sample rate of 48 kHz. Since a variety of volunteers with different hardware have produced this dataset, there may be cause for concern with the consistency of recordings. The up vote metadata will be used as a proxy in finding the most consistent recordings. The mixed recording conditions may actually contribute positively to the training, producing a final model is practical and generalisable to the wider world.

The sheer size of the dataset cannot be computed using my limited time and resources. I chose to sample 1000 audio files from the three languages: English, Spanish and Chinese. These languages seems very different in terms of linguistics, tone, pitch which I hope will lead to more distinguishable features between the classes.

The Chinese dataset is heavily gender imbalanced with just 32 usable recordings made by females compared to the 4909 clips created by males. There is no opportunity to correct the gender imbalance with oversampling since there are so few records from the female Chinese population. To eliminate the possibility of this gender imbalance becoming a key driver behind the final model created, the decision was made to only use the male dataset for each language.

Exploratory Visualization

![Example mel spectogram][image]

[image]: images/mel spectogram.png "Example mel spectogram"

The image above is a visualisation plot from the mel spectogram extraction of an audio file. The plot was generated using the Librosa The following characteristics can be found:

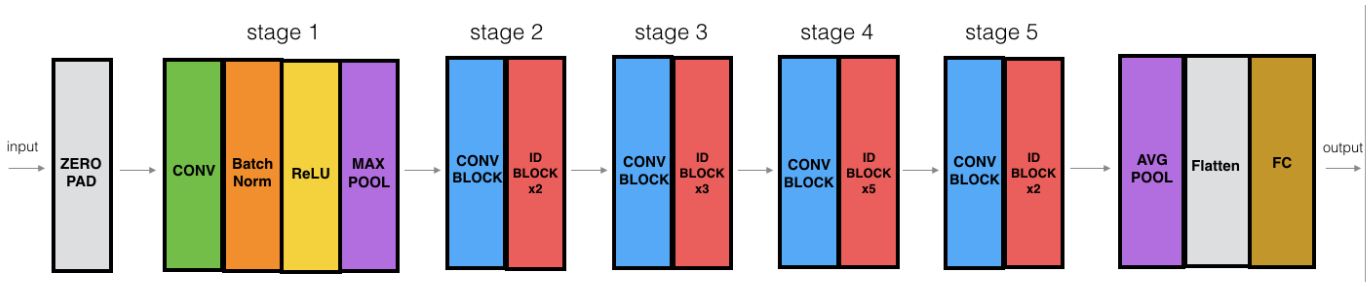
- The y-axis shows the frequency bins in the mel scale going from 200Hz to 8000Hz. Any frequencies above or below are not in the human voice range in log scale.
- The x-axis illustrates the length of the audio in seconds.
- The colour bar on the right hand side denotes the amplitude in log scale.
- There is no silence on either side of the graph as the audio has been trimmed.

The rationale behind using the log scale of frequencies, is that it similar to the way in which humans process or perceive sound.

This visual representation of the audio, shows several dimensions of data relating to sound or voice: time, frequency and amplitude. In my opinion, we have encoded the audio information related to a voice or sound into an image. This would be a great input for an image classification model using Convolutional Neural Networks.

Algorithms and Techniques

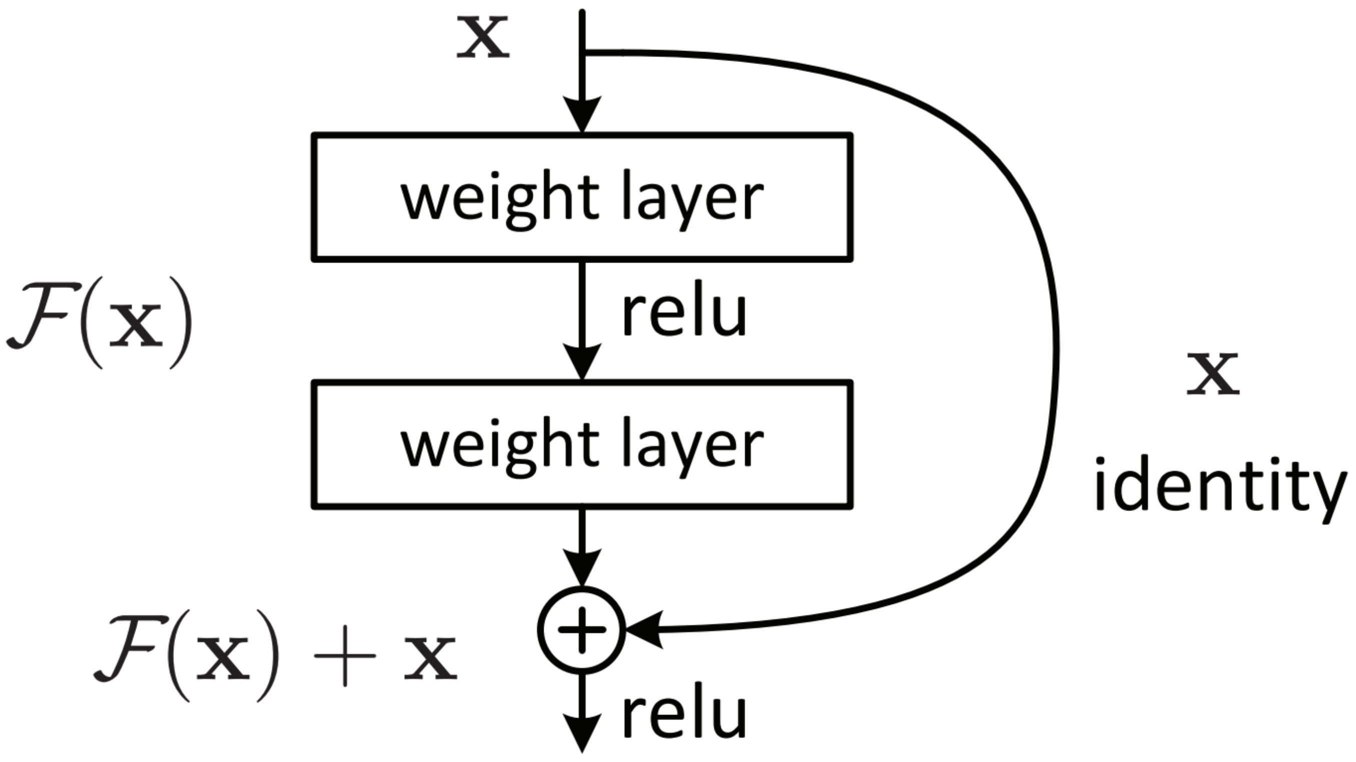
ResNet50 is a state-of-the-art Deep Neural Network architecture with 50 layers trained on the ImageNet dataset. The model architecture was groundbreaking in solving the vanishing gradient problem where deeper neural networks are outperformed by shallower neural networks.



source: medium, Priya Dwivedi Resnet50 model architecture

Similar to other Convolutional Neural Networks (CNNs) such as VGG16, ResNet50 is formed of convolutional layers. These convolutions give spatial context that fully-connected neural networks lack, so in cases where spatial context is important, for instance image classification, CNNs typically outperform fully-connected networks. Due to the max pooling layers, complex CNNs require fewer parameters to train and are less prone to overfitting.

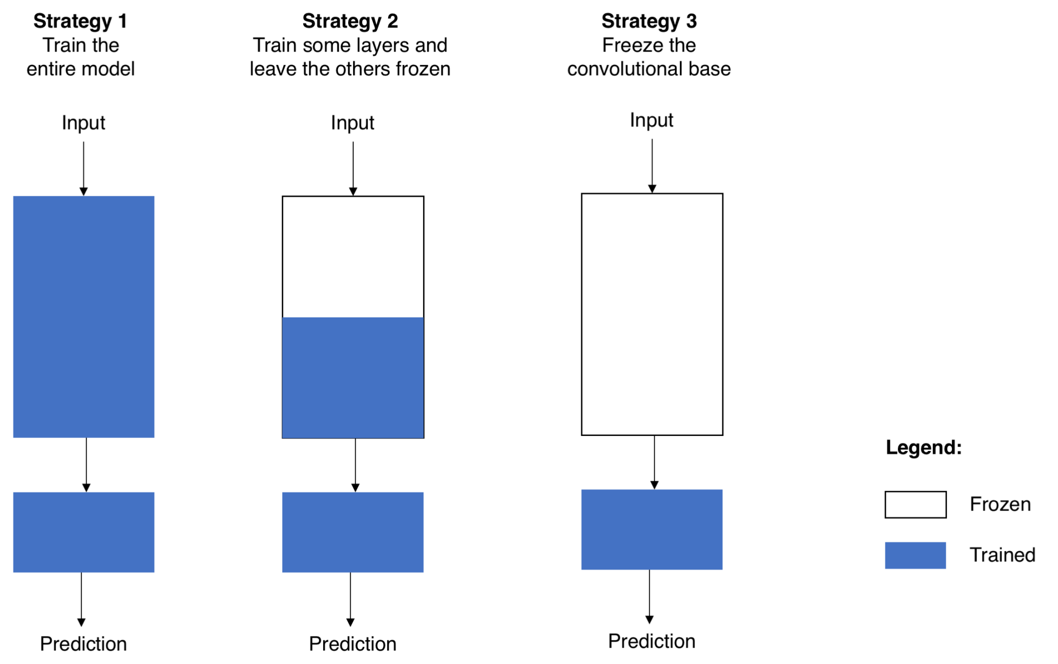
However, traditional CNNs such as VGG16 still face vanishing gradients. ResNet50 eliminates this problem through the use of identity mapping. Identity mapping or skip connections, take the input from one layer and propagates to a layer further ahead than it normally would - the concept is outlined below in a residual block. This generates alternate paths for the gradient to move through the network - thus allowing us to build much deeper networks than traditionally possible.



source: imgur Residual block

Since the ResNet50 model is a great classifier for images, **Transfer Learning** can be applied using the ResNet50 model. Rather than having to build a whole new model architecture and train weights from scratch, we can remove the fully-connected layers or the classifier from the ResNet50 model. We can then build our own model architecture on top then apply one of the following strategies (as shown in the image below):

- Train the entire model (initialising pre-trained weights or random)
- Freeze some of the initial layers and train the other layers
- Freeze the bottleneck features and train using the output of the convolution layers as input to your own model architecture



source: towardsdatascience, Pedro Marcelino Transfer learning strategies

Transfer learning offers the ability to leverage the learnings made by another model and apply those learnings to a similar use case. The high performance from the ResNet50 model in the image classification space is the main rationale behind using transfer learning with ResNet50.

In all strategies outlined above, we always have our own classifier with at least one fully-connected layer. The following parameters will be used to build, train and tune a classifier:

- **Model architecture**
  - Number of layers (depth of the network)
  - Layer types and their respective parameters (including activation functions)
    - dense (number of nodes, regularisers)
    - convolution (filters, stride, padding)
    - dropout (probability)
    - pooling (max pooling, average pooling)
- **Model Training parameters**
  - Number of epochs (number of complete passes)
  - Batch size (number of samples for every update)
  - Loss function (the error function e.g. MSE, categorical_crossentropy)
  - Optimser (function used to minimise the loss function e.g. RMSprop, ADAM, SGD)
  - Learning rate (default usually 0.001)
  - Momentum (for SGD)
  - Metrics (accuracy)

In terms of data inputs and the transformations into in the model, the flow of data would be as follows:

- Audio files are converted into mel spectograms and saved to disk. The number of bins has been set to 224 so that a image of dimensions width x 224 will be created - the height is well-suited for ResNet50.
- The keras function `preprocess_input` from `keras.applications.resnet50` was able to take the images with varying heights and resize into arrays of (224,224,3).
  - The width of the images were rescaled to fit the 224 width dimension.
  - Since these image were grayscale, the `preprocess_input` function manages to automatically create three channels to replicate RGB by copying the original image two times over.
- The ResNet50 model needs to initalised with the parameter `include_top=False`. This removes the final fully-connected in the original model, allowing us to create our own model architecture to classify on top.
- **[OPTIONAL]** If we want to freeze all the convolutional layers, we want a convenient way to store the botteneck features from the convolutional layers of ResNet50. Since the weights are preloaded from Keras, next step is use the `model.predict()` function with the (1800, 224,224,3).shape array as input - the 1800 denotes the number of training files. This would be the input to my own fully-connected layers.

Benchmark

The benchmark model will be a classifier which defaults to one language. This would replicate the current scenario in using ASR where we have no prior knowledge of the audio language. Our data has been split into three equally balanced classes. Using the evaluation metric defined above, this benchmark would achieve a 33.3% accuracy score. Success would be achieved if the final model beats this score.

III. Methodology

Data Preprocessing

I chose to split the data into train/validation/test sets. The train set will be used to train the classifier, with the validation set providing context on the generalisation of the model. The test set will be kept to evaluate the performance of the final model.

There were four stages to preprocessing:

- Sampling
- Creating folders
- Generating and saving images

The process to generate my sample datasets from the metadata can be summarised in the following steps:

1. The audio metadata is ordered by up votes for each language
2. The audio metadata is filtered by male
3. The top 1000 audio files are selected for each language
4. The sampled metadata is randomised for each language
5. The metadata is divided to create train/validation/test sets for each language
6. The train/validation/test sets for the three languages are combined together to create equally balanced classes

This generated 1800 training datapoints, 600 validation datapoints and 600 test datapoints.

With the train/validation/test split in mind, I used the following folder structure to manage my mel spectogram images:

- data/train-mel spectogram/
- data/validation-mel spectogram/
- data/test-mel spectogram/

From the audio file path from each of these metadatapoints above, I ran the following workflow:

1. Load audio file using path
2. Trim the silence from the beginning and end of the audio file
3. Extract the log mel-frequency spectogram array
4. Scale array elements between 0 and 255
5. Pivot the array (width represents time)
6. Initialise mel-frequency spectogram image from array
7. Save file to relevant folder outined above

I used the following packages to generate and save images:

- Librosa - in-built functionality to explore audio features including MFCCs, spectograms and mel spectograms.
- Pillow - functions to convert, save and crop images from numpy arrays
- Numpy - helpful in manipulating arrays efficiently

This generated 1800 training mel spectograms, 600 validation mel spectograms and 600 test mel spectograms. These images were all 224 pixels in height, though they did vary in width. I did not feel that this would be an issue since the CNN architectures would learn from many examples that this is not a distinguishing feature.

Implementation

Data extraction and preprocessing

I did spend a majority of my time working on feature selection and the subsequent data processing. Investigating the different kinds of audio features such as MFCCs, spectograms, mel-frequency and spectograms was fairly challenging with no background in signal processing. The different sources of data (Chinese, English, Spanish) and the number of files meant that a bottleneck in the process was the processing time for singular processes (creating the mel spectogram). To alleviate that bottleneck, I decided to create a multiprocessing function which made the reprocessing files up to 16 times faster.

Modelling

I used the Keras python library in the creation of the final solution:

- **ResNet50:** A pre-trained ResNet50 model is available on Keras. The Keras API is intuituve with in-built presets for hyperparameters but with the flexibility for lower-level tweaking. This means that we can quickly build a baseline model which then can be refined.

- **Transfer learning:** Keras enables the application of transfer learning by allowing chosen layers to be frozen or unfrozen depending on the chosen strategy. The object `model.layers` contains all of the layers. Each of these layers has an attribute `trainable` which freezes the layer if set to `False` (vice versa).

- **Speed and versatility:** Although training the a neural network can be slow, this can be alleviated using cloud computing or powerful hardware. Using a GPU can greatly accelerate the training time with very little change in the code by running the code below.

> ```
> sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
> ```

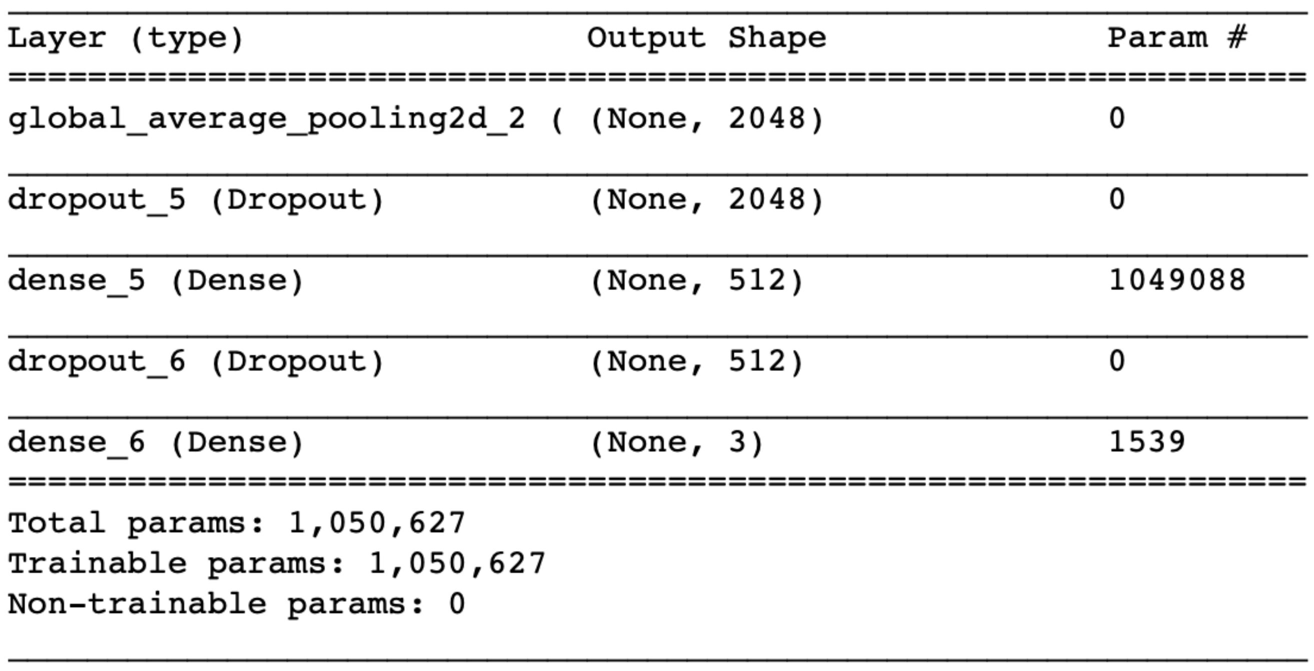
source

I used a ResNet50 model to apply transfer learning to the dataset. In the interests of training time, I froze all convolutional layers and kept the ImageNet weights to extract the bottleneck features which I then used as features into my own model architecture. The implementation steps are documented extensively in the *Algorithm and Techniques* section, so I will only provide brief commentary on what I previously covered and focus on the implementation details of the initial solution.

1. Initalise pre-trained ResNet50 model from Keras with `include_top=False` as an input parameter.
2. Using `image` class from `keras.preprocessing` module load mel spectogram images using the `image.load(<path>)` method. Scaling the RGB array elements between 0 and 1 in the same step.
3. Apply `preprocess_input` method to ensure the inputs will work as expected.
4. Extract bottleneck features from the initialised model using `predict` method.

At this point, the output from the convolutional layers (bottleneck features) were stored in an array with shape (1800, 7, 7, 2048). Steps 1-5 were also applied to the test and validation set. This numpy array represents the 1800 training files and the complex features generated from ResNet50.

First model architecture



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_2 ( (None, 2048)              0
_________________________________________________________________
dropout_5 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               1049088
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 3)                 1539
=================================================================
Total params: 1,050,627
Trainable params: 1,050,627
Non-trainable params: 0
_________________________________________________________________
```

The overall model architecture was fairly lightweight.

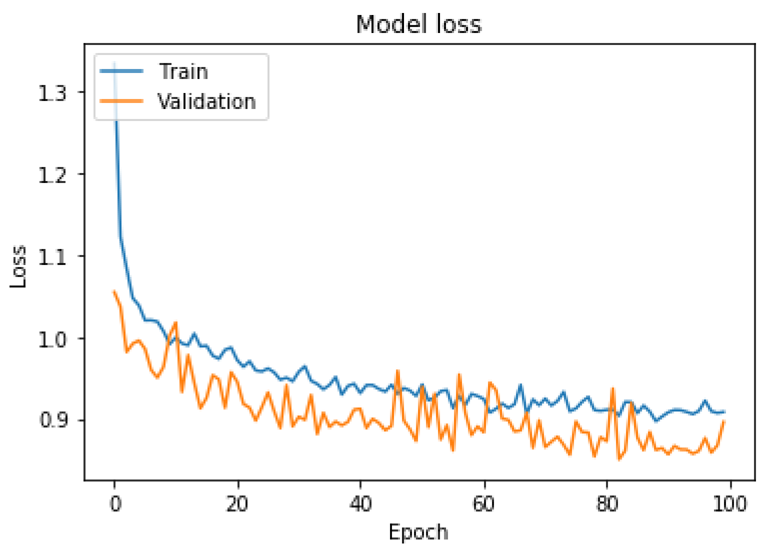
- The input layer was a global average pooling layer. This had an effect of dimension reduction and greatly reduces the number of parameters for the next layers.
- A dropout layer was put in place next to reduce overfitting. This enables the random dropping of nodes - in this case the probability of dropping a ode was 50%.
- The next layer was a dense layer with 512 nodes. This is a fully-connected layer in which every node in this layer is connected to every node in the next layer. The ReLU (Rectified Linear Unit) activation function was

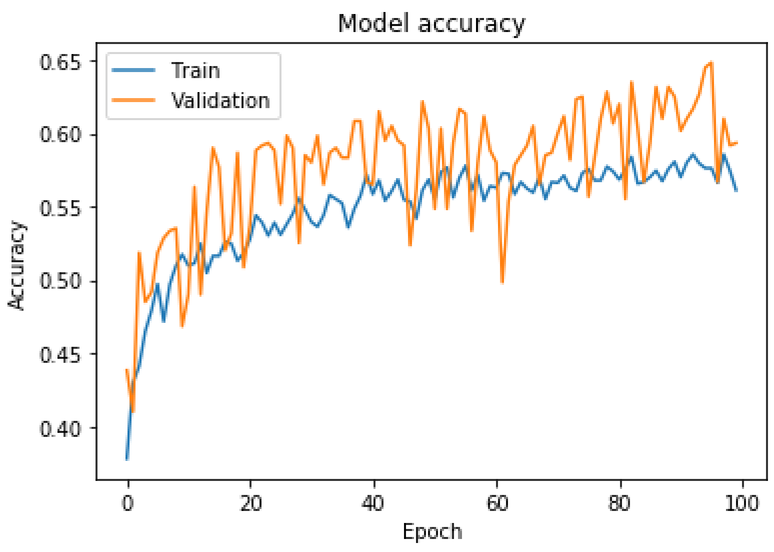
used. The ReLU activation function typically increases training speed and alleviates to some extent the vanishing gradient problem. This is the default activation function for most DNNs.
- A dropout layer with 0.2 probability.
- The output layer was a dense layer with three nodes (representing the three classes) and a softmax activation function. The output from the activation function adds up to 1.

At this point, I was ready to train the model. I used the following parameters for the first model:

- Loss function: `categorical_crossentropy`
- Optimiser: `rmsprop` (default)
- Metrics: `accuracy`
- Epochs: `100`
- Batch size: `25`

Refinement





The initial performance of the model was as follows: 61.8% accuracy on the test dataset. This is a good performance for a first pass. However, I was worried by the near convergence of the train validation loss. It seemed like there wasn't much I could do from just changing the hyperparameters.

In search of better performance, I experimented with different elements of the process defined above:

- Different model architectures (VGG16, MobileNet_V2). Using the same transfer learning method, these models produced slightly worse results. Accuracy on the test set came in around 55%.
- Unfreezing layers (last 4 layers - this took too long). My machine was not up to the task. This was physically impossible with my time constraints. This may have been possible with cloud computing.
- Further subsampling to find mel spectograms of consistent size and transformations in the form of cropping mel spectograms images. Accuracy using the same model was poor ~40% accuracy on the test set.
- Increased the size of training/test/validation dataset (up to 4000). However, the increased training time did not coincide with greater performance. Accuracy on the test set was 55% at best.

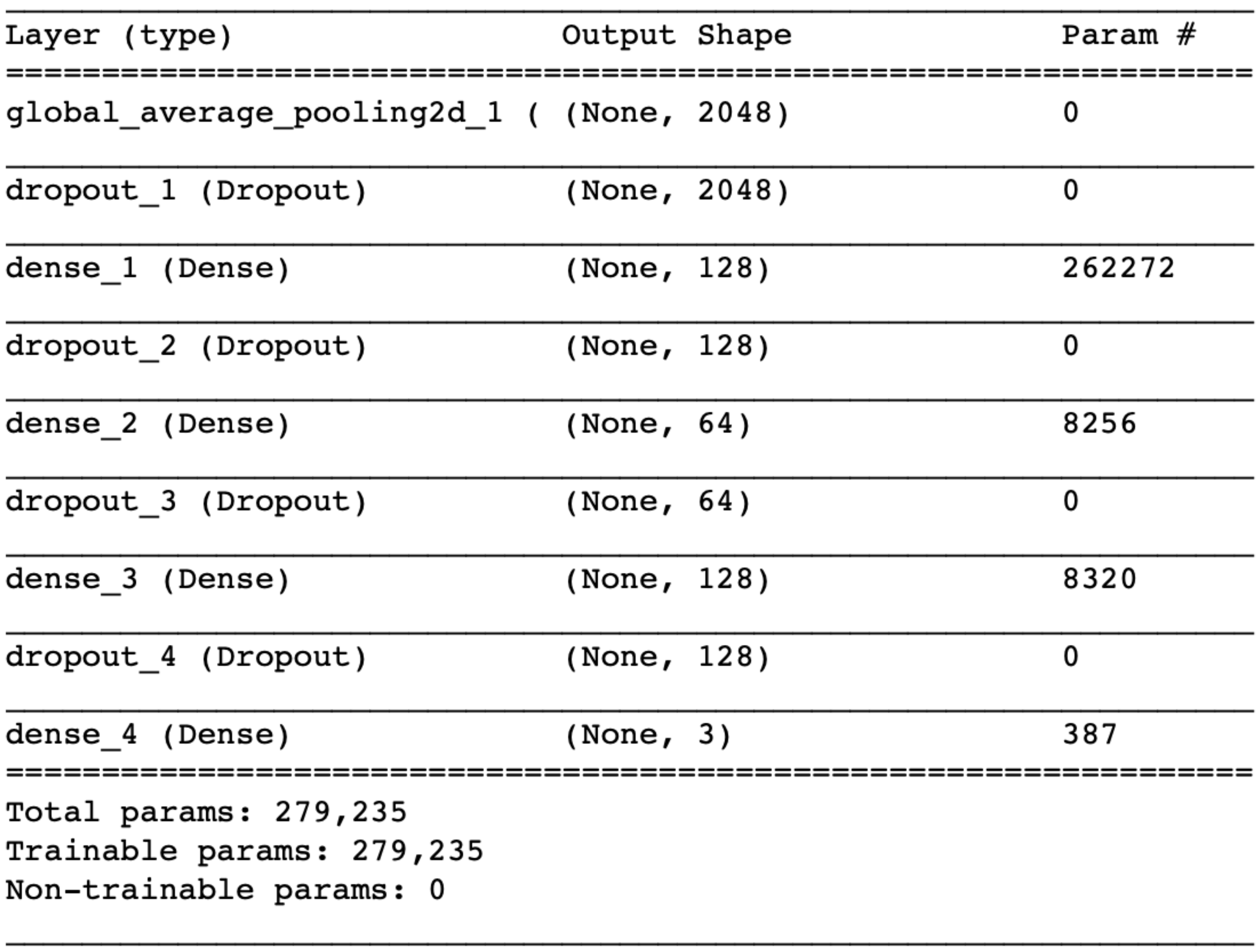
The only method which increased the model accuracy on the test set was tuning the hyperparameters and rebuilding the model architecture of the fully-connected layers. I believe the most important factors were as follows:

- Changing the architecture to include more layers, less nodes and more dropouts proved to increase accuracy slightly and made the new model less prone to overfitting. Though the model did take longer to converge.
- Changing the optimser - ADAM (Adaptive Moment Optimization) seemed to perform better than RMSProp.
- Increasing the number of epochs. This allowed more time for the model to seek out gradients and allowing for the training and validation loss to converge.
- Increasing batch size did not have much of an effect on overall performance.

IV. Results

Model Evaluation and Validation

Final model architecture



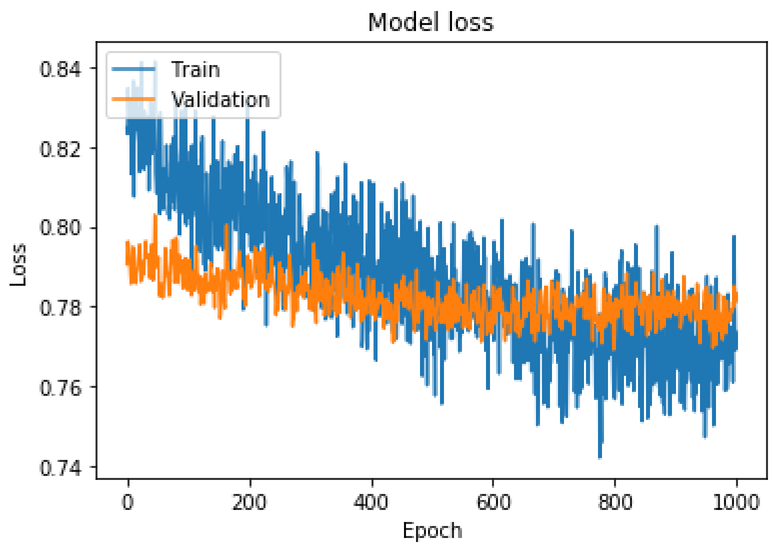
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_1 ( (None, 2048)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               262272
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               8320
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 3)                 387
=================================================================
Total params: 279,235
Trainable params: 279,235
Non-trainable params: 0
_________________________________________________________________
```

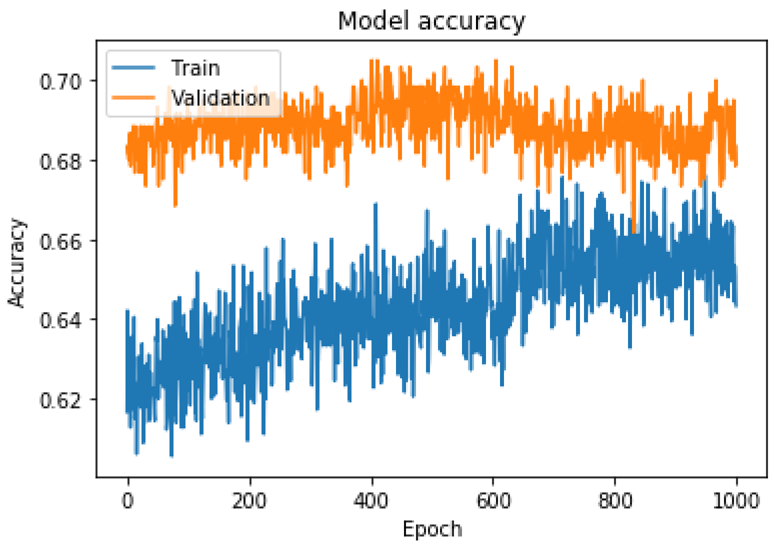
The overall model architecture was fairly lightweight. In total there were just 279,235 parameters to train which is fairly fast to train.

- The input layer was a global average pooling layer.
- A dropout layer with 0.5 probability of dropping nodes.
- A dense layer with 128 nodes and ReLU activation.
- A dropout layer with 0.5 probability of dropping nodes.
- A dense layer with 64 nodes and ReLU activation.
- A dropout layer with 0.4 probability of dropping nodes.
- A dense layer with 128 nodes and ReLU activation.
- A dropout layer with 0.2 probability of dropping nodes.
- The output layer consisted of a dense layer with three nodes and a softmax activation function.

I used the following parameters for the final model:

- Loss function: `categorical_crossentropy`
- Optimiser: `ADAM`
- Learning rate: `0.0001`
- Metrics: `accuracy`
- Epochs: `1000`
- Batch size: `50`





The final accuracy on the test dataset achieved was 67.3%.

I was hoping for a larger increase in accuracy from the initial model. This represents a 5.5% increase in performance from hyperparameter tuning and changes to the model architecture. The final parameters are appropriate as you can see the model starting to overfit in the 600th epoch, with the training loss exceeding validation loss. This seems to be the maximum we can extract from the data.

In order to validate the model. I created another test set 'unseen' to verify the results we obtained in this exercise. This consisted of files which had less up votes in comparison to the previous dataset the final model was trained upon so there may be a difference in quality.

Putting through 3000 sample files from English, Chinese and Spanish leads to a unseen accuracy of 56.4%. This is not the headline 67.3% recorded by the test set but it is in line when we consider the data may be less clean in comparison to the data we trained upon.

It would be interesting to see the performance of the model when taking recordings from the real world - not in a test environment. I think the model will struggle in such a situation since the underlying data the model is built upon is not generalised to a great amount of noise.

Justification

At 67.3% accuracy, the final model is more than twice as accurate as the benchmark model (33.3%). This is more than twice as good as the benchmark. We have met our goal that we set out to achieve.

Although the solution is not robust enough to be part of a workign product, the results prove that language identification can be done and I am sure that with more research, time and better data we can create a credible solution that can be used commerically. With an accuracy of just 67.3%, it is unlikely anyone will use this solution in their day to day. Those who do use ASR today are unlikely to need automatic language identification. By configuring their language settings (whether it be a voice assistant or something else) to the language they use day to day. In that case, our 67.3% pales in comparison to an almost 100% accuracy in real life.

Nevertheless, this is a great proof of concept in identifying/applying a viable method to solve an issue which I feel will be important to solve in the era of automation.

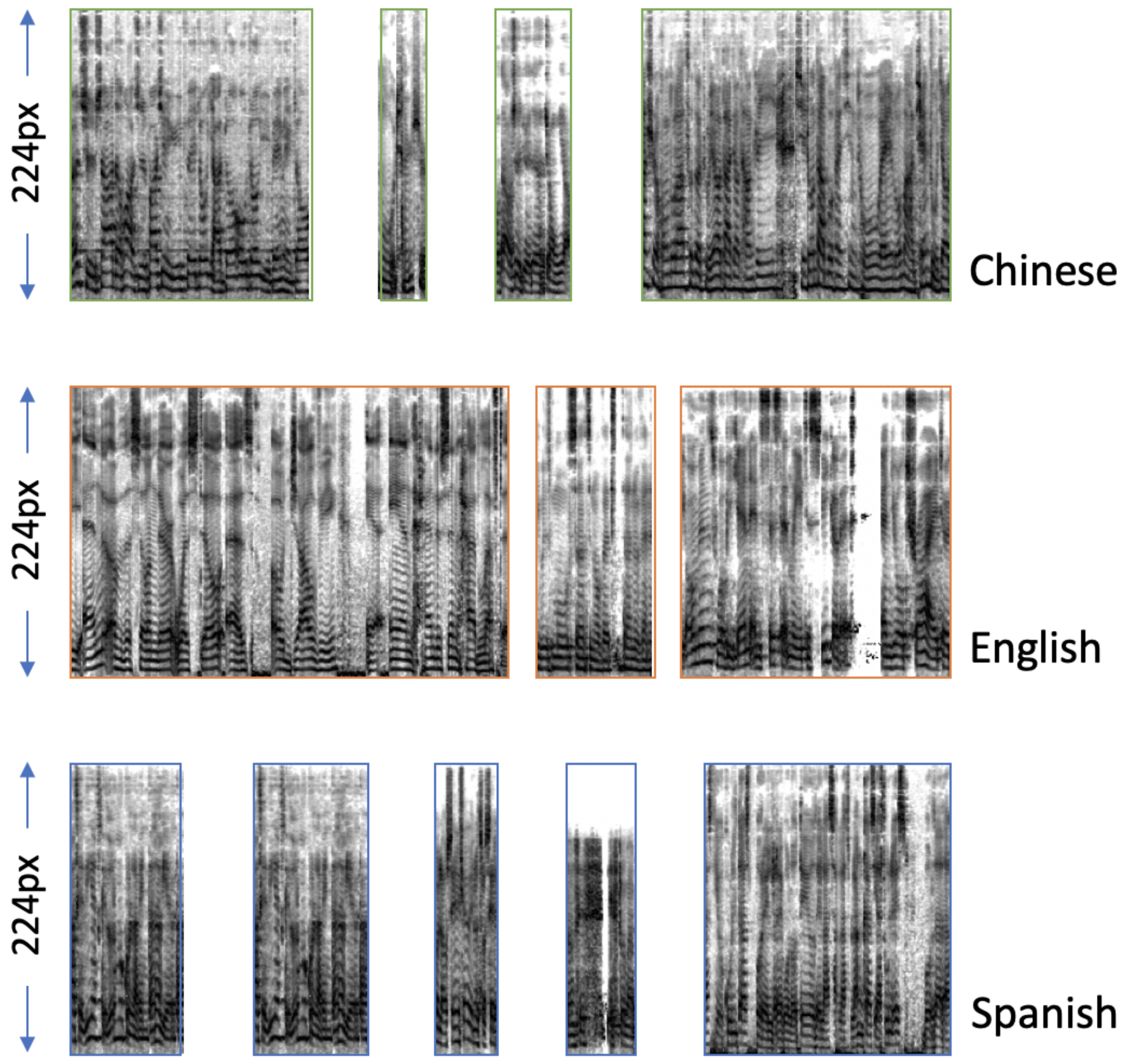
V. Conclusion

Free-Form Visualization

The visualisation below shows the raw input features fed into the Keras `preprocess` function for ResNet50 models. A short analysis of the spectograms identifies the following:

- Consistent heights, but vastly different widths of the mel spectogram images. The huge differences between the image widths implored me to experiment with cropping files as outlined previously (to no avail).
- Large sections of white pixels - an indication of silence. Although the audio files have been trimmed, there seems to exist pauses in speech. This could be a time-related feature that can be picked up by convolutional layers. However, in future we could also approach this in a different way by using voice activity detection methods to remove all silence the mel spectograms.

Preprocessed grayscale mel spectogram features for each language



Reflection

The summary of the steps taken within the project are listed out as follows:

1. Identified problem statement and dataset
2. Exploratory data analysis
3. Sample robust dataset
4. Audio feature transformation analysis
5. Preprocessing mel spectograms for train/validation/test sets
6. ResNet50 bottleneck feature extraction from mel spectograms
7. Build model architecture
8. Iterate
    - Train
    - Validate
    - Tweak hyperparameters

9. Evaluate model with test dataset

- The most interesting part of the project was creating a solutions to micro problems on the journey to finding a larger solution. Although frustrating, some elements of the code were quite useful (such as my multiprocessing script `parallel.py`) and I will likely be able to use these learnings in my future projects.
- The most difficult part of the project was implementing a model in a deeply technical subject matter that I was unfamiliar with - audio. I had to deal with two areas of research deep learning and audio signaling. This was difficult to balance with my day to day. In addition, comparative to image classification, there is very little in terms of material for purely audio classification.
- The final solution does meet expectations. At 67.3%, it is more than twice as accurate as the benchmark model (33.3%). However, I do not believe this is a solution that will be beneficial for day to day applications. The amount of error is just too great for the model to be relied upon - just imagine a voice assistant which gets your language right two thirds of the time! There are also practical considerations in using this tool day to day in terms of the difficult data transformation pipeline.

Improvement

The current model does provide better results than the benchmark model, nevertheless there are still a lot of potential improvements to be made.

- Data cleansing activities could have made a direct impact on the performance of the final model. Retrospectively, I do feel the Chinese dataset did not have the same depth as the Spanish and English sets. There were comparatively fewer votes for the Chinese dataset. If I could choose the languages again, I may have gone for Russian, Italian, German, English and Spanish since they had the most complete data. These languages would have also opened the doors to further increases in the train/validation/test set size and might have led to better accuracy or generalisation at least.

- I would like to use cloud computing next time I apply deep learning to a problem. It seems that deep learning requires many examples to train on in order to become robust. In light of that, cloud computing gives me an oportunity to train with more data and thus is likely to improve results.

- If I used my final solution as the new benchmark, there most definitely is a better solution. There are three areas in which my solution falls down, usability, speed and performance. Google enables multiple languages to be used on it's own voice assistant and language detection though limited to a certain extent. Even if Google's performance is worse, the implementation in terms of outreach is far better than my locally built model. That can be changed with efforts in creating a more efficient pipeline and serving my model through a website (or via API at least).

References:

Deep Residual Learning for Image Recognition Kaiming He et al.

Using CNNs and RNNs for Music Genre Recognition Priya Dwivedi

Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures Renuka Joshi

A COMPARISON OF AUDIO SIGNAL PREPROCESSING METHODS FOR DEEP NEURAL NETWORKS ON MUSIC TAGGING Keunwoo Choi et al.

Deep Learning with Audio Thread Robert Bracco

Combining High-Level Features of Raw Audio Waves and Mel-Spectrograms for Audio Tagging Marcel Lederle, Benjamin Wilhelm

A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning Dipanjan (DJ) Sarkar

Getting to Know the Mel Spectrogram Dalya Gartzman

Audio Classification using FastAI and On-the-Fly Frequency Transforms John Hartquist

An overview of gradient descent optimization algorithms Sebastian Ruder

A bunch of tips and tricks for training deep neural networks Long Ang