

# 4 Deep Learning Models for Speech Recognition

## Data Science - Capstone Project Submission

- Student Name: **James Toop**
- Student Pace: **Self Paced**
- Scheduled project review date/time: **29th October 2021 @ 21:30 BST**
- Instructor name: **Jeff Herman / James Irving**
- Blog URL: <https://toopster.github.io/> (<https://toopster.github.io/>)

In [1]:

```
1  # Import relevant libraries and modules for creating and training networks
2  import pandas as pd
3  import numpy as np
4  import matplotlib.pyplot as plt
5  import seaborn as sns
6  import wave
7  import soundfile as sf
8  import librosa, librosa.display
9  import IPython.display as ipd
10 import os
11 import json
12
13 import tensorflow as tf
14 from tensorflow.keras.layers.experimental import preprocessing
15 from tensorflow.keras import layers
16 from tensorflow.keras import models
17 from sklearn.model_selection import train_test_split
18 from sklearn.preprocessing import LabelEncoder
19 from sklearn.metrics import accuracy_score, confusion_matrix
20
21 import logging
22 logging.getLogger("tensorflow").setLevel(logging.ERROR)
23
24 import pathlib
25 from pathlib import Path
26
27 import shared_functions.preprocessing as preprocess
```

In [2]:

```
1  # Set seed for reproducibility
2  seed = 123
3  tf.random.set_seed(seed)
4  np.random.seed(seed)
```

### 4.0.1 Create model evaluation functions

In [3]:

```
1 def visualise_results(results):
2     '''
3     Visualise the results from the model history plotting training and
4     validation accuracy and loss vs. epoch
5
6     Params:
7         results: Model history
8     '''
9     history = results.history
10
11     plt.figure(figsize=(20,8))
12     plt.xticks(fontsize=12)
13     plt.yticks(fontsize=12)
14
15     plt.subplot(1, 2, 1)
16     plt.plot(history['val_loss'])
17     plt.plot(history['loss'])
18     plt.legend(['Validation Loss', 'Training Loss'], fontsize=12)
19     plt.title('Loss', fontsize=18)
20     plt.xlabel('Epochs', fontsize=14)
21     plt.ylabel('Loss', fontsize=14)
22
23     plt.subplot(1, 2, 2)
24     plt.plot(history['val_acc'])
25     plt.plot(history['acc'])
26     plt.legend(['Validation Accuracy', 'Training Accuracy'], fontsize=12)
27     plt.title('Accuracy', fontsize=18)
28     plt.xlabel('Epochs', fontsize=14)
29     plt.ylabel('Accuracy', fontsize=14)
30     plt.show()
```

In [4]:

```
1 def model_performance(input_model,
2                       input_train_X,
3                       input_train_y,
4                       input_test_X,
5                       input_test_y
6                       ):
7     '''
8     Visualise the results from the model history plotting training and
9     validation accuracy and loss vs. epoch
10
11     Params:
12         input_model:
13         input_train_X (ndarray): Inputs for the training dataset
14         input_train_y (ndarray): Targets for the training dataset
15         input_test_X (ndarray): Inputs for the test dataset
16         input_test_y (ndarray): Targets for the test dataset
17     '''
18     # Evaluate training results
19     results_train = input_model.evaluate(input_train_X,
20                                         input_train_y,
21                                         verbose=0)
22
23     # Evaluate test results
24     results_test = input_model.evaluate(input_test_X,
25                                       input_test_y,
26                                       verbose=0)
27
28     # Create predictions based on model
29     predictions = input_model.predict(input_test_X)
30
31     # Calculate confusion matrix
32     conf_matrix = confusion_matrix(input_test_y.argmax(axis=1),
33                                   np.round(predictions.argmax(axis=1)))
34
35     # Output performance evaluation
36     print('CONFUSION MATRIX -----')
37     print(conf_matrix)
38
39     print('\nTRAIN METRICS -----')
40     print('Loss: {}'.format(results_train[0]))
41     print('Accuracy: {}'.format(np.round(results_train[1]*100), 2))
42
43     print('\nTEST METRICS -----')
44     print('Loss: {}'.format(results_test[0]))
45     print('Accuracy: {}'.format(np.round(results_test[1]*100), 2))
```

In [5]:

```
1 def check_array_shapes(X_train, y_train, X_val, y_val, X_test, y_test):
2     '''
3     Output the training, test and validation dataset shapes
4
5     Params:
6         X_train (ndarray): Inputs for the training dataset
7         y_train (ndarray): Targets for the training dataset
8         X_val (ndarray): Inputs for the validation dataset
9         y_val (ndarray): Targets for the validation dataset
10        X_test (ndarray): Inputs for the test dataset
11        y_test (ndarray): Targets for the test dataset
12    '''
13    print ("Number of training samples: " + str(X_train.shape[0]))
14    print ("Number of testing samples: " + str(X_test.shape[0]))
15    print ("Number of validation samples: " + str(X_val.shape[0]))
16    print ("X_train shape: " + str(X_train.shape))
17    print ("y_train shape: " + str(y_train.shape))
18    print ("X_test shape: " + str(X_test.shape))
19    print ("y_test shape: " + str(y_test.shape))
20    print ("X_val shape: " + str(X_val.shape))
21    print ("y_val shape: " + str(y_val.shape))
```

In [6]:

```
1 def reformat_y(y):
2     '''
3     Reformats / One Hot Encodes targets
4
5     Params:
6         y (ndarray): Input targets
7
8     Returns:
9         y (ndarray): One hot encoded targets
10    '''
11    y = LabelEncoder().fit_transform(y)
12    y = tf.keras.utils.to_categorical(y)
13    return y
```

In [7]:

```
1 def save_model(save_model, save_path):
2     '''
3     Save the model
4
5     Params:
6         save_model : Input Tensorflow model
7         save_path (str): Path to save model including file extension .h5
8     '''
9    save_model.save(save_path)
```

## 4.0.2 Create training, test and validation datasets

In [8]:

```
1 # Create the train, test and validation datasets for the Speech Commands dataset
2 sc_data_path = "speech_commands_data.json"
3 (
4     sc_X_train,
5     sc_y_train,
6     sc_X_val,
7     sc_y_val,
8     sc_X_test,
9     sc_y_test,
10 ) = preprocess.create_train_test(sc_data_path, 'MFCCs')
```

Datasets loaded...

In [9]:

```
1 # Create the train, test and validation datasets for the Ultrasuite Top 35 dataset
2 us_data_path = 'ultrasuite_top35_data.json'
3 (
4     us_X_train,
5     us_y_train,
6     us_X_val,
7     us_y_val,
8     us_X_test,
9     us_y_test,
10 ) = preprocess.create_train_test(us_data_path, 'MFCCs')
```

Datasets loaded...

In [11]:

```
1 # Check the array shapes for the Ultrasuite dataset
2 check_array_shapes(sc_X_train,
3                     sc_y_train,
4                     sc_X_val,
5                     sc_y_val,
6                     sc_X_test,
7                     sc_y_test)
```

Number of training samples: 61052  
Number of testing samples: 19079  
Number of validation samples: 15263  
X\_train shape: (61052, 44, 13, 1)  
y\_train shape: (61052,)  
X\_test shape: (19079, 44, 13, 1)  
y\_test shape: (19079,)  
X\_val shape: (15263, 44, 13, 1)  
y\_val shape: (15263,)

In [10]:

```
1 # Check the array shapes for the Ultrasuite dataset
2 check_array_shapes(us_X_train,
3                   us_y_train,
4                   us_X_val,
5                   us_y_val,
6                   us_X_test,
7                   us_y_test)
```

Number of training samples: 3942  
Number of testing samples: 1233  
Number of validation samples: 986  
X\_train shape: (3942, 44, 13, 1)  
y\_train shape: (3942,)  
X\_test shape: (1233, 44, 13, 1)  
y\_test shape: (1233,)  
X\_val shape: (986, 44, 13, 1)  
y\_val shape: (986,)

## 4.1 Model 1: Create a simple baseline model

In [12]:

```
1 def build_baseline_model(input_shape,
2                           output_units,
3                           loss_func='categorical_crossentropy',
4                           learning_rate=0.0001):
5     '''
6     Build a baseline model
7
8     Params:
9         input_shape (tuple): Shape of array representing a sample train
10        output_units (int): Number of targets / categories
11        loss_func (str): Loss function to use
12        learning_rate (float): Learning rate
13
14    Returns:
15        baseline_model: Tensorflow model
16    '''
17    baseline_model = tf.keras.models.Sequential()
18    baseline_model.add(tf.keras.layers.InputLayer(input_shape=input_shape))
19    baseline_model.add(tf.keras.layers.Flatten())
20    baseline_model.add(tf.keras.layers.BatchNormalization())
21    baseline_model.add(tf.keras.layers.Dense(output_units, activation='softmax'))
22
23    # Set optimizer and learning rate
24    optimiser = tf.optimizers.Adam(learning_rate=learning_rate)
25
26    # Compile the baseline model
27    baseline_model.compile(loss=loss_func,
28                          optimizer=optimiser,
29                          metrics=['acc'])
30
31    # Print summary for model
32    baseline_model.summary()
33
34    return baseline_model
```

In [13]:

```
1  # Function for fitting the model
2  def fit_model(model,
3                epochs,
4                batch_size,
5                patience,
6                X_train,
7                y_train,
8                X_val,
9                y_val):
10     '''
11     Fit the model
12
13     Params:
14         model : Input Tensorflow model
15         epochs (int): Number of training epochs
16         batch_size (int): Number of samples per batch
17         patience (int): Number of epochs to wait before early stop,
18                        if there no improvement on accuracy
19         X_train (ndarray): Inputs for the training dataset
20         y_train (ndarray): Targets for the training dataset
21         X_val (ndarray): Inputs for the validation dataset
22         y_val (ndarray): Targets for the training dataset
23
24     Returns:
25         results: Training history
26     '''
27     # Define early stopping criteria
28     early_stopping = tf.keras.callbacks.EarlyStopping(monitor='accuracy',
29                                                       min_delta=0.001,
30                                                       patience=patience)
31
32     # Fit the model
33     results = model.fit(X_train,
34                        y_train,
35                        epochs=epochs,
36                        batch_size=batch_size,
37                        validation_data=(X_val, y_val),
38                        callbacks=[early_stopping])
39     return results
```

#### 4.1.1 Baseline model for the Speech Commands dataset

In [14]:

```
1 # One-hot encode Speech Commands labels
2 sc_train_y = reformat_y(sc_y_train)
3 sc_test_y = reformat_y(sc_y_test)
4 sc_val_y = reformat_y(sc_y_val)
5
6 # Create baseline model for Speech Commands dataset
7 sc_input_shape = (sc_X_train[0].shape)
8 sc_output_units = 35
9 sc_baseline_model = build_baseline_model(sc_input_shape,
10                                         sc_output_units,
11                                         learning_rate=0.0001)
12
13 # Fit model
14 sc_epochs = 40
15 sc_batch_size = 32
16 sc_patience = 5
17 sc_baseline_results = fit_model(sc_baseline_model,
18                                sc_epochs,
19                                sc_batch_size,
20                                sc_patience,
21                                sc_X_train,
22                                sc_train_y,
23                                sc_X_val,
24                                sc_val_y)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
flatten (Flatten)	(None, 572)	0
=====		
batch_normalization (BatchNormalizatio	(None, 572)	2288
=====		
dense (Dense)	(None, 35)	20055
=====		
Total params: 22,343		
Trainable params: 21,199		
Non-trainable params: 1,144		

Epoch 1/40

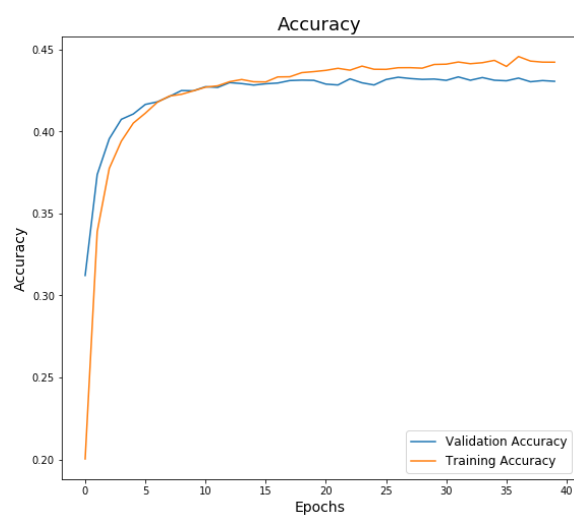
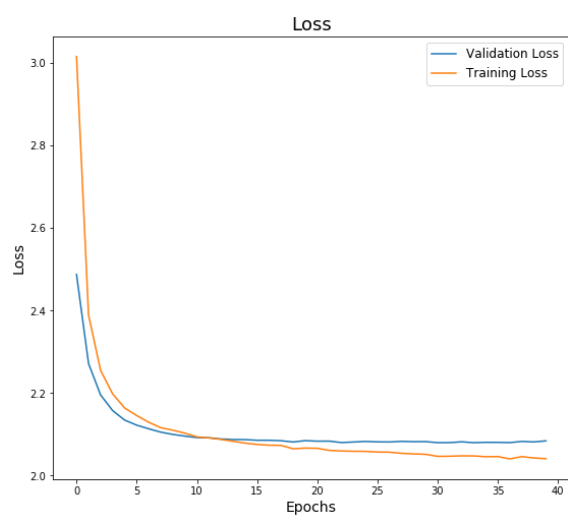
WARNING: AutoGraph could not transform <function Model.make\_train\_function.<locals>.train\_function at 0x7ffc6d73bd40> and will run it as-is.

Please report this to the TensorFlow team. When filing the bug, set the environment variable `TF_CPP_MIN_LOG_LEVEL=0` and attach the AutoGraph traceback above.



In [15]:

```
1 # Visualise the training and validation loss and accuracy across epochs
2 visualise_results(sc_baseline_results)
```



In [17]:

```
1 # Evaluate the model performance
2 model_performance(sc_baseline_model,
3                   sc_X_train,
4                   sc_train_y,
5                   sc_X_test,
6                   sc_test_y
7                   )
```

CONFUSION MATRIX -----

```
[[233  25   6 ...   4   6  10]
 [ 12 357   1 ...   0   5   2]
 [ 11  10 124 ...   0  24   3]
 ...
 [  6   1   1 ...  94   2  59]
 [  5   0   3 ...   4 365  18]
 [  4   1   3 ...  29  20 313]]
```

TRAIN METRICS -----

Loss: 1.947283148765564

Accuracy: 47.0%

TEST METRICS -----

Loss: 2.0892109870910645

Accuracy: 43.0%

## 4.1.2 Baseline model for the Ultrasuite dataset

In [18]:

```
1 # One-hot encode Ultrasuite Top 35 labels
2 us_train_y = reformat_y(us_y_train)
3 us_test_y = reformat_y(us_y_test)
4 us_val_y = reformat_y(us_y_val)
5
6 # Create baseline model for Ultrasuite dataset
7 us_input_shape = (us_X_train[0].shape)
8 us_output_units = 35
9 us_baseline_model = build_baseline_model(us_input_shape,
10                                         us_output_units,
11                                         learning_rate=0.0001)
12
13 # Fit model
14 us_epochs = 40
15 us_batch_size = 32
16 us_patience = 5
17 us_baseline_results = fit_model(us_baseline_model,
18                                us_epochs,
19                                us_batch_size,
20                                us_patience,
21                                us_X_train,
22                                us_train_y,
23                                us_X_val,
24                                us_val_y)
```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
=====		
flatten_1 (Flatten)	(None, 572)	0
=====		
batch_normalization_1 (Batch Normalization)	(None, 572)	2288
=====		
dense_1 (Dense)	(None, 35)	20055
=====		

Total params: 22,343

Trainable params: 21,199

Non-trainable params: 1,144

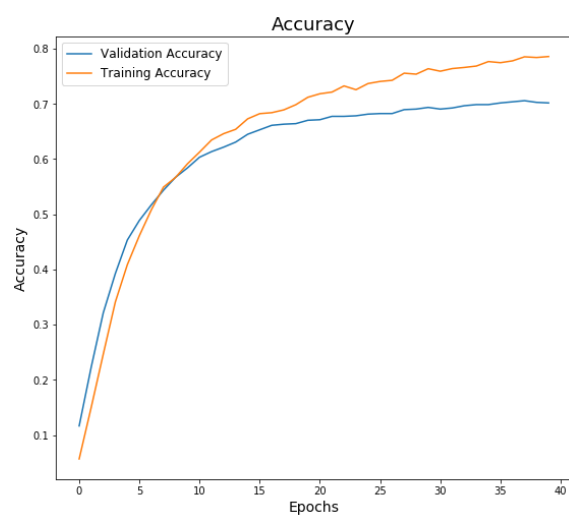
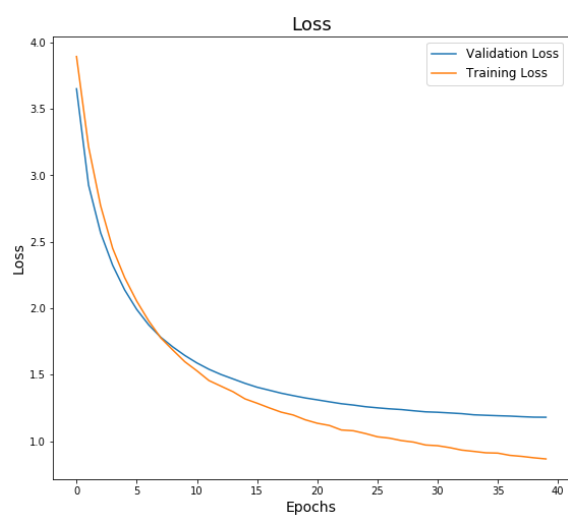
Epoch 1/40

WARNING: AutoGraph could not transform <function Model.make\_train\_function.<locals>.train\_function at 0x7ffc57590680> and will run it as-is.

Please report this to the TensorFlow team. When filing the bug, set t

In [19]:

```
1 # Visualise the training and validation loss and accuracy across epochs
2 visualise_results(us_baseline_results)
```



In [21]:

```
1 # Evaluate the model performance
2 model_performance(us_baseline_model,
3                   us_X_train,
4                   us_train_y,
5                   us_X_test,
6                   us_test_y
7                   )
```

CONFUSION MATRIX -----

```
[[15  0  0 ...  0  0  0]
 [ 0 32  0 ...  1  2  0]
 [ 1  1 24 ...  0  0  0]
 ...
 [ 0  1  0 ... 13  1  2]
 [ 0  1  0 ...  0 40  0]
 [ 0  0  0 ...  0  0 12]]
```

TRAIN METRICS -----

Loss: 0.7894143462181091

Accuracy: 81.0%

TEST METRICS -----

Loss: 1.254226803779602

Accuracy: 70.0%

### 4.1.3 Model Conclusion

It's starting point with the training accuracy for the Ultrasuite dataset being 81%. The model is clearly overfitting the training data and is not generalising well when shown unseen data given the validation accuracy of 70%.

What is of particular note is the vastly different accuracies when compared to the Speech Commands dataset which are both under 50%.

This could suggest that a more simple model works better for audio samples from children with a speech sound disorder.

## 4.2 Model 2: Baseline model with increased learning rate and batch size

In [22]:

```
1 # Create second baseline model on Ultrasuite dataset changing the learning rate
2 us_input_shape = (us_X_train[0].shape)
3 us_output_units = 35
4 us_baseline_model_2 = build_baseline_model(us_input_shape,
5                                           us_output_units,
6                                           learning_rate=0.001)
7
8
9 # Fit model
10 us_epochs = 40
11 us_batch_size = 64
12 us_patience = 5
13 us_baseline_results_2 = fit_model(us_baseline_model_2,
14                                   us_epochs,
15                                   us_batch_size,
16                                   us_patience,
17                                   us_X_train,
18                                   us_train_y,
19                                   us_X_val,
20                                   us_val_y)
```

---

Total params: 22,343

Trainable params: 21,199

Non-trainable params: 1,144

---

Epoch 1/40

WARNING: AutoGraph could not transform <function Model.make\_train\_function.<locals>.train\_function at 0x7ffc592b58c0> and will run it as-is.

Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH\_VERBOSITY=10`) and attach the full output.

Cause: 'arguments' object has no attribute 'posonlyargs'

To silence this warning, decorate the function with @tf.autograph.experimental.do\_not\_convert

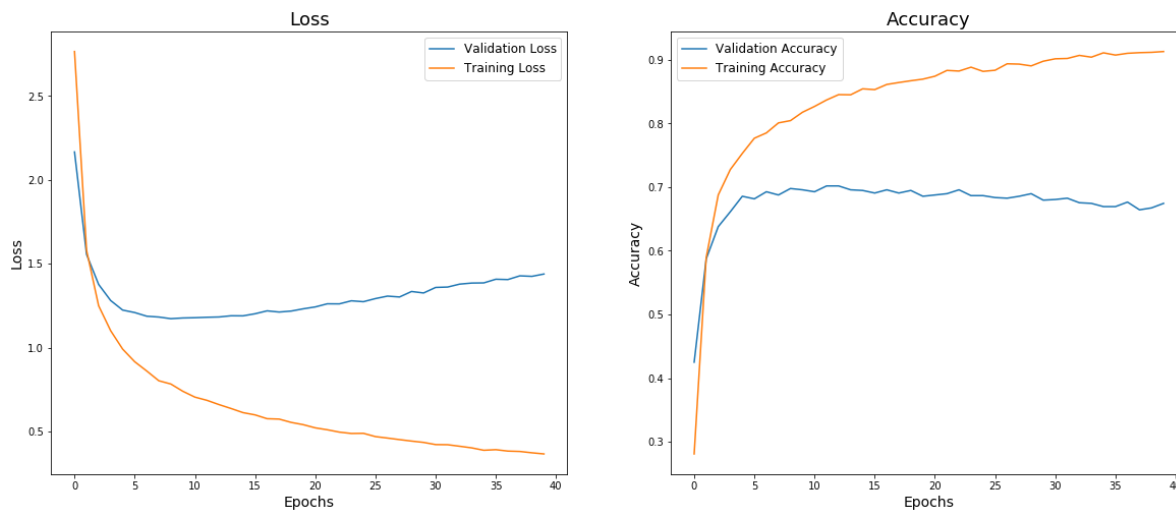
62/62 [=====] - ETA: 0s - loss: 2.7641 - acc:

0.2808WARNING: AutoGraph could not transform <function Model.make\_test\_function.<locals>.test\_function at 0x7ffc58f88f80> and will run it as-is.

Please report this to the TensorFlow team. When filing the bug, set t

In [23]:

```
1 # Visualise the training and validation loss and accuracy across epochs
2 visualise_results(us_baseline_results_2)
```



In [25]:

```
1 # Evaluate the model performance
2 model_performance(us_baseline_model_2,
3                  us_X_train,
4                  us_train_y,
5                  us_X_test,
6                  us_test_y
7                  )
```

```
CONFUSION MATRIX -----
[[14  0  0 ...  0  0  1]
 [ 0 28  0 ...  3  1  0]
 [ 1  0 28 ...  0  0  0]
 ...
 [ 1  1  0 ... 13  0  1]
 [ 1  0  0 ...  0 43  0]
 [ 0  0  0 ...  0  0 18]]
```

```
TRAIN METRICS -----
Loss: 0.28699809312820435
Accuracy: 94.0%
```

```
TEST METRICS -----
Loss: 1.5567854642868042
Accuracy: 65.0%
```

## 4.2.1 Model Conclusion

We have improved the training accuracy of the baseline model, to 94%, by increasing the learning rate and increasing the batch size but this has only served to exacerbate the issue of overfitting with test accuracy not reducing to 65% and the difference between the two plots being dramatic.

In an effort to deal with the issue of overfitting, we will introduce some regularization layers and see what effect they have on performance.

## 4.3 Model 3: Adding hidden layers to the baseline model, deepening the network

In [26]:

```
1  # Build a second baseline model
2  def build_model_2(input_shape,
3                    output_units,
4                    loss_func='categorical_crossentropy',
5                    learning_rate=0.0001):
6
7      model_2 = tf.keras.models.Sequential()
8      model_2.add(tf.keras.layers.InputLayer(input_shape=input_shape))
9      model_2.add(tf.keras.layers.Flatten())
10     model_2.add(tf.keras.layers.BatchNormalization())
11     model_2.add(tf.keras.layers.Dense(256, activation='relu'))
12     # model_2.add(tf.keras.layers.Dropout(0.3))
13     model_2.add(tf.keras.layers.Dense(128, activation='relu'))
14     model_2.add(tf.keras.layers.Dense(64, activation='relu'))
15     model_2.add(tf.keras.layers.Dense(output_units, activation='softmax'))
16
17     # Set optimizer and learning rate
18     optimiser = tf.optimizers.Adam(learning_rate=learning_rate)
19
20     # Compile the baseline model
21     model_2.compile(loss=loss_func,
22                    optimizer=optimiser,
23                    metrics=['acc'])
24
25     # Print summary for model
26     model_2.summary()
27
28     return model_2
```



In [27]:

```
1 # Create baseline model for Ultrasuite dataset
2 us_input_shape = (us_X_train[0].shape)
3 us_output_units = 35
4 us_model_2 = build_model_2(us_input_shape,
5                             us_output_units,
6                             learning_rate=0.0001)
7
8 # Fit model
9 us_epochs = 40
10 us_batch_size = 64
11 us_patience = 3
12 us_results_3 = fit_model(us_model_2,
13                           us_epochs,
14                           us_batch_size,
15                           us_patience,
16                           us_X_train,
17                           us_train_y,
18                           us_X_val,
19                           us_val_y)
```

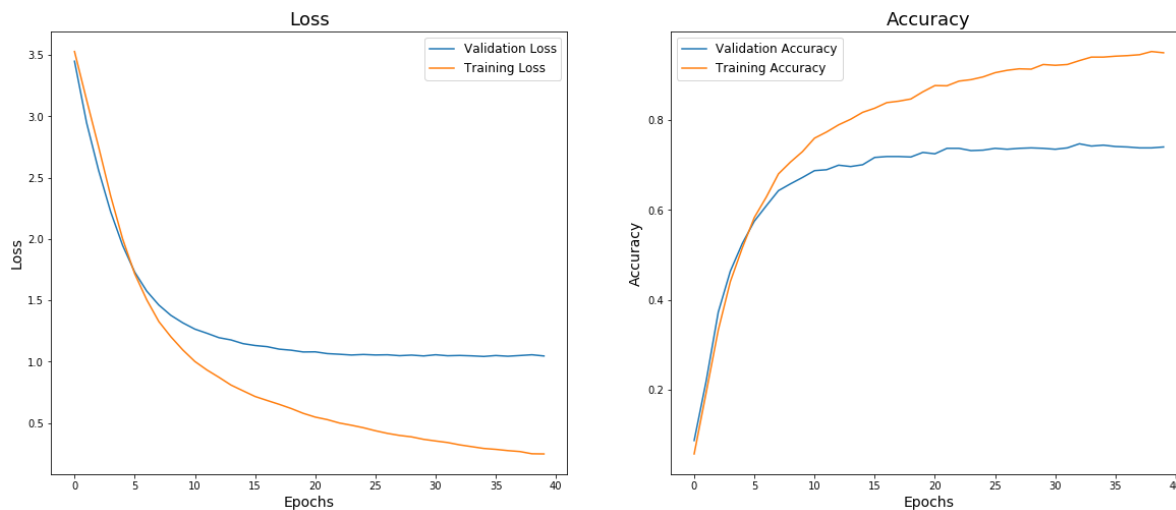
Model: "sequential\_3"

Layer (type)	Output Shape	Param #
flatten_3 (Flatten)	(None, 572)	0
batch_normalization_3 (Batch Normalization)	(None, 572)	2288
dense_3 (Dense)	(None, 256)	146688
dense_4 (Dense)	(None, 128)	32896
dense_5 (Dense)	(None, 64)	8256
dense_6 (Dense)	(None, 35)	2275

Total params: 192,403  
Trainable params: 191,259  
Non-trainable params: 1,144

In [28]:

```
1 # Visualise the training and validation loss and accuracy across epochs
2 visualise_results(us_results_3)
```



In [30]:

```
1 # Evaluate the model performance
2 model_performance(us_model_2,
3                   us_X_train,
4                   us_train_y,
5                   us_X_test,
6                   us_test_y
7                   )
```

CONFUSION MATRIX -----

```
[ [19  0  0 ...  1  1  0]
  [ 0 34  0 ...  0  2  0]
  [ 0  0 26 ...  0  0  0]
  ...
  [ 1  0  0 ... 16  0  1]
  [ 0  1  0 ...  0 43  0]
  [ 0  0  0 ...  1  0 18]]
```

TRAIN METRICS -----

Loss: 0.19082011282444  
Accuracy: 96.0%

TEST METRICS -----

Loss: 1.085261344909668  
Accuracy: 72.0%

### 4.3.1 Running the model using Mel Spectrograms instead of MFCCs

A thought exercise more than anything else, just to see if there are any performance gains that can be made by using Mel Spectrograms instead of MFCCs.

In [31]:

```
1  # Create the train, test and val datasets for the Ultrasuite top 35 subset using  
2  us_data_path = 'ultrasuite_top35_data_melspec.json'  
3  (  
4      X_train_mel,  
5      y_train_mel,  
6      X_val_mel,  
7      y_val_mel,  
8      X_test_mel,  
9      y_test_mel,  
10 ) = preprocess.create_train_test(us_data_path, 'mel_specs')
```

Datasets loaded...

In [32]:

```
1 # Create model using Mel Spectrograms instead of MFCCs
2 us_mel_input_shape = (X_train_mel.shape[1], X_train_mel.shape[2], 1)
3 us_mel_output_units = 35
4 us_mel_model = build_model_2(us_mel_input_shape,
5                               us_mel_output_units,
6                               learning_rate=0.0001)
7
8 # One-hot encode Speech Commands labels
9 mel_train_y = reformat_y(y_train_mel)
10 mel_test_y = reformat_y(y_test_mel)
11 mel_val_y = reformat_y(y_val_mel)
12
13 # Fit model
14 us_mel_epochs = 40
15 us_mel_batch_size = 64
16 us_mel_patience = 3
17 us_mel_results = fit_model(us_mel_model,
18                             us_mel_epochs,
19                             us_mel_batch_size,
20                             us_mel_patience,
21                             X_train_mel,
22                             mel_train_y,
23                             X_val_mel,
24                             mel_val_y)
```

Model: "sequential\_4"

Layer (type)	Output Shape	Param #
=====		
flatten_4 (Flatten)	(None, 5632)	0
=====		
batch_normalization_4 (Batch Normalization)	(None, 5632)	22528
=====		
dense_7 (Dense)	(None, 256)	1442048
=====		
dense_8 (Dense)	(None, 128)	32896
=====		
dense_9 (Dense)	(None, 64)	8256
=====		
dense_10 (Dense)	(None, 35)	2275
=====		

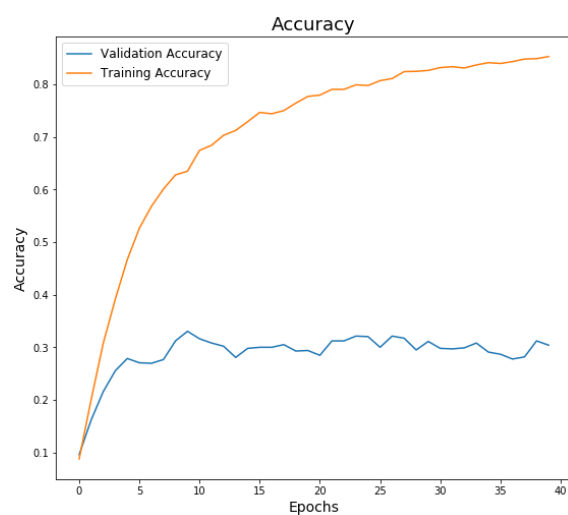
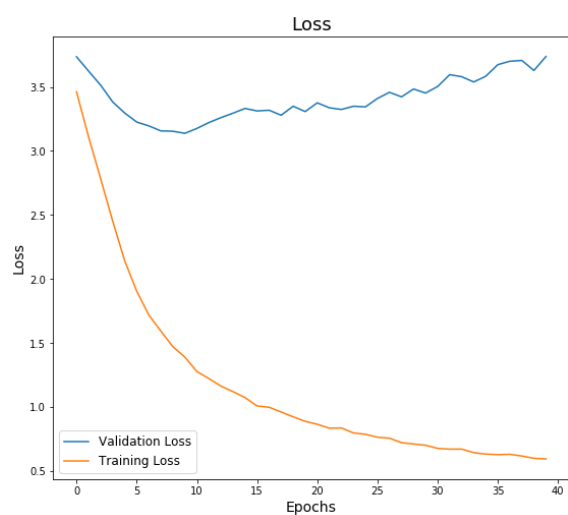
Total params: 1,508,003

Trainable params: 1,496,739

Non-trainable params: 11,264

In [33]:

```
1 # Visualise the training and validation loss and accuracy across epochs
2 visualise_results(us_mel_results)
```



In [35]:

```
1 # Evaluate the model performance
2 model_performance(us_mel_model,
3                   x_train_mel,
4                   mel_train_y,
5                   x_test_mel,
6                   mel_test_y
7                   )
```

CONFUSION MATRIX -----

```
[[ 6  0  0 ...  3  2  0]
 [ 1 18  0 ...  0  6  0]
 [ 0  1  3 ...  1  9  0]
 ...
 [ 0  0  0 ...  5  7  0]
 [ 0  3  0 ...  0 40  0]
 [ 3  1  0 ...  0  3  4]]
```

TRAIN METRICS -----

Loss: 1.8458300828933716

Accuracy: 53.0%

TEST METRICS -----

Loss: 3.5853798389434814

Accuracy: 30.0%

## 4.4 Model 4: Convolutional Neural Network

In [36]:

```
1  # Build a CNN model
2
3  def build_cnn_model(input_shape,
4                      output_units,
5                      loss='categorical_crossentropy',
6                      learning_rate=0.0001):
7
8      cnn_model = tf.keras.models.Sequential()
9
10     # 1st convolutional layer
11     cnn_model.add(tf.keras.layers.Conv2D(32, (3, 3),
12                                           activation='relu',
13                                           input_shape=input_shape,
14                                           kernel_regularizer=tf.keras.regularizers.L2(0.01)))
15     cnn_model.add(tf.keras.layers.BatchNormalization())
16     cnn_model.add(tf.keras.layers.MaxPooling2D((2, 2),
17                                                 strides=(2, 2),
18                                                 padding='same'))
19
20     # 2nd convolutional layer
21     cnn_model.add(tf.keras.layers.Conv2D(32, (4, 4),
22                                           activation='relu',
23                                           kernel_regularizer=tf.keras.regularizers.L2(0.01)))
24     cnn_model.add(tf.keras.layers.BatchNormalization())
25     cnn_model.add(tf.keras.layers.MaxPooling2D((3, 3),
26                                                 strides=(2, 2),
27                                                 padding='same'))
28
29     # 3rd convolutional layer
30     cnn_model.add(tf.keras.layers.Conv2D(64, (2, 2),
31                                           activation='relu',
32                                           kernel_regularizer=tf.keras.regularizers.L2(0.01)))
33     cnn_model.add(tf.keras.layers.BatchNormalization())
34     cnn_model.add(tf.keras.layers.MaxPooling2D((2, 2),
35                                                 strides=(2, 2),
36                                                 padding='same'))
37
38     # Flatten output and feed into dense layer
39     cnn_model.add(tf.keras.layers.Flatten())
40     cnn_model.add(tf.keras.layers.Dense(32, activation='relu'))
41     cnn_model.add(tf.keras.layers.Dropout(0.3))
42
43     # Softmax output layer
44     cnn_model.add(tf.keras.layers.Dense(output_units, activation='softmax'))
45
46     optimiser = tf.optimizers.Adam(learning_rate=learning_rate)
47
48     # Compile model
49     cnn_model.compile(optimizer=optimiser,
50                      loss=loss,
51                      metrics=['acc'])
52
53     # Print summary for model
54     cnn_model.summary()
55
56     return cnn_model
```

## 4.4.1 CNN model for the Speech Commands dataset

In [37]:

```
1 # Create CNN model using Speech Commands MFCCs
2 sc_cnn_input_shape = (sc_X_train.shape[1], sc_X_train.shape[2], 1)
3 sc_output_units = 35
4 sc_cnn_model = build_cnn_model(sc_cnn_input_shape,
5                               sc_output_units,
6                               learning_rate=0.0001)
7
8
9 # Fit model to Speech Commands data
10 sc_epochs = 40
11 sc_batch_size = 64
12 sc_patience = 3
13 sc_cnn_results = fit_model(sc_cnn_model,
14                            sc_epochs,
15                            sc_batch_size,
16                            sc_patience,
17                            sc_X_train,
18                            sc_train_y,
19                            sc_X_val,
20                            sc_val_y)
```

flatten_5 (Flatten)	(None, 256)	0
dense_11 (Dense)	(None, 32)	8224
dropout (Dropout)	(None, 32)	0
dense_12 (Dense)	(None, 35)	1155

=====  
Total params: 34,883  
Trainable params: 34,627  
Non-trainable params: 256

Epoch 1/40

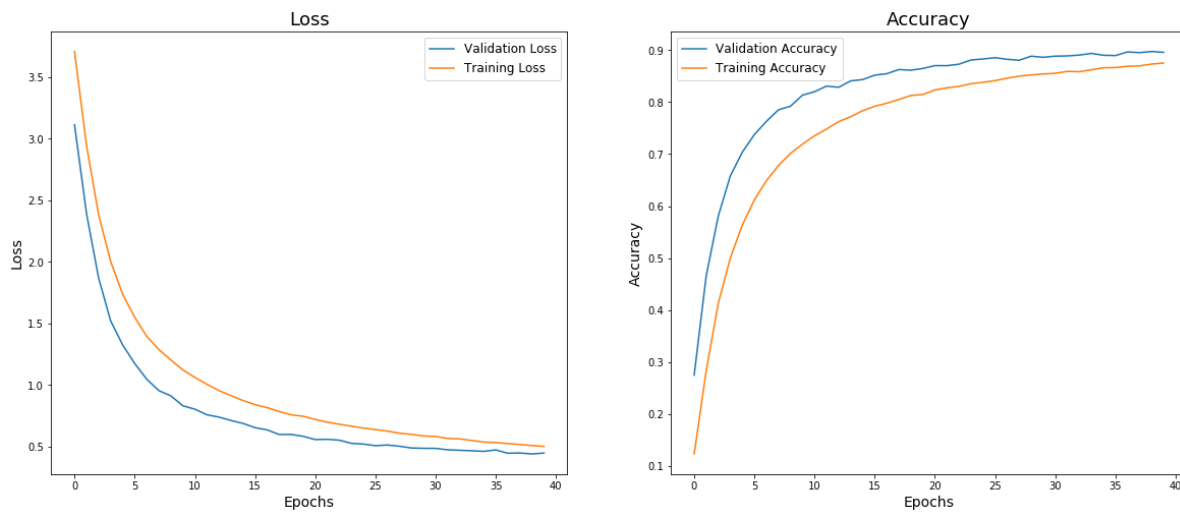
WARNING: AutoGraph could not transform <function Model.make\_train\_function.<locals>.train\_function at 0x7ffc553c7c20> and will run it as-is.

Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH\_VERBOSITY=10`) and attach the full output.



In [38]:

```
1 # Visualise the training and validation loss and accuracy across epochs
2 visualise_results(sc_cnn_results)
```



In [40]:

```
1 # Evaluate the model performance
2 model_performance(sc_cnn_model,
3                  sc_X_train,
4                  sc_train_y,
5                  sc_X_test,
6                  sc_test_y
7                  )
```

CONFUSION MATRIX -----

```
[[613  3  0 ...  0  1  2]
 [ 5 629  0 ...  0  2  1]
 [ 0  1 339 ...  0  4  0]
 ...
 [ 0  0  0 ... 225  0 46]
 [ 1  0  1 ...  1 604  2]
 [ 0  0  3 ... 34  4 602]]
```

TRAIN METRICS -----

Loss: 0.32391873002052307

Accuracy: 93.0%

TEST METRICS -----

Loss: 0.4416976571083069

Accuracy: 90.0%

## 4.4.2 CNN model for the Ultrasuite dataset

In [41]:

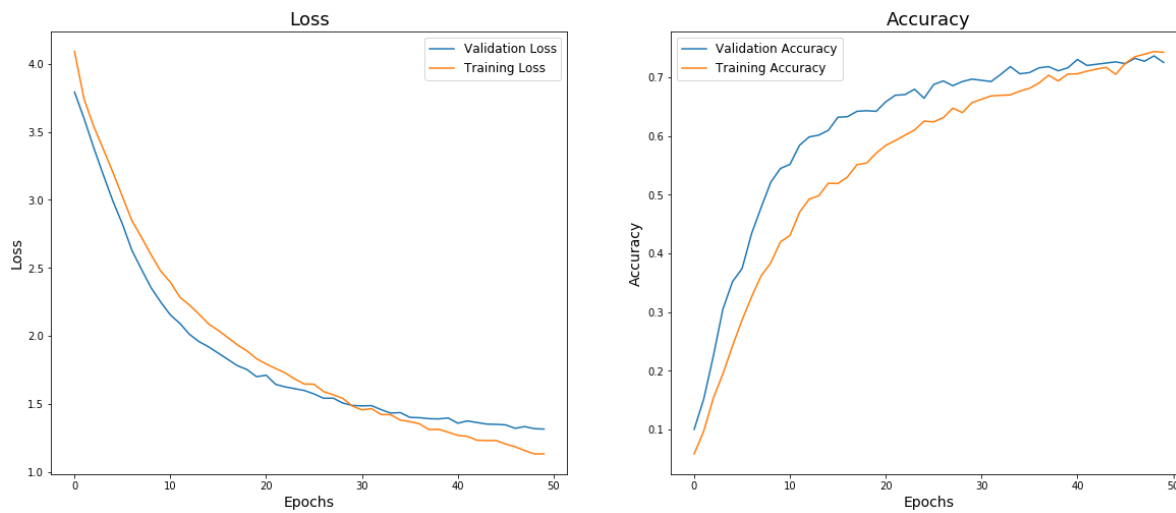
```
1 # Create CNN model using Ultrasuite MFCCs
2 us_cnn_input_shape = (us_X_train.shape[1], us_X_train.shape[2], 1)
3 us_output_units = 35
4 us_cnn_model = build_cnn_model(us_cnn_input_shape,
5                               us_output_units,
6                               learning_rate=0.0001)
7
8
9 # Fit model to Ultrasuite data
10 us_epochs = 50
11 us_batch_size = 16
12 us_patience = 3
13 us_cnn_results = fit_model(us_cnn_model,
14                             us_epochs,
15                             us_batch_size,
16                             us_patience,
17                             us_X_train,
18                             us_train_y,
19                             us_X_val,
20                             us_val_y)
```

Model: "sequential\_6"

Layer (type)	Output Shape	Param #
=====		
conv2d_3 (Conv2D)	(None, 42, 11, 32)	320
-----		
batch_normalization_8 (Batch Normalization)	(None, 42, 11, 32)	128
-----		
max_pooling2d_3 (MaxPooling2D)	(None, 21, 6, 32)	0
-----		
conv2d_4 (Conv2D)	(None, 18, 3, 32)	16416
-----		
batch_normalization_9 (Batch Normalization)	(None, 18, 3, 32)	128
-----		
max_pooling2d_4 (MaxPooling2D)	(None, 9, 2, 32)	0
-----		
conv2d_5 (Conv2D)	(None, 8, 1, 64)	8256
-----		
batch_normalization_10 (Batch Normalization)	(None, 8, 1, 64)	256

In [42]:

```
1 # Visualise the training and validation loss and accuracy across epochs
2 visualise_results(us_cnn_results)
```



In [46]:

```
1 save_model(us_cnn_model, 'final_model.h5')
```

## 4.5 Final Model Performance Evaluation

In [44]:

```
1 model_performance(us_cnn_model,
2                   us_X_train,
3                   us_train_y,
4                   us_X_test,
5                   us_test_y
6                   )
```

CONFUSION MATRIX -----

```
[[17  0  0 ...  0  0  1]
 [ 0 35  0 ...  1  0  1]
 [ 0  0 29 ...  0  1  0]
 ...
 [ 1  0  0 ... 15  1  0]
 [ 0  1  2 ...  0 43  0]
 [ 1  0  0 ...  0  0 16]]
```

TRAIN METRICS -----

Loss: 0.7186428308486938

Accuracy: 89.0%

TEST METRICS -----

Loss: 1.3699027299880981

Accuracy: 72.0%

In [48]:

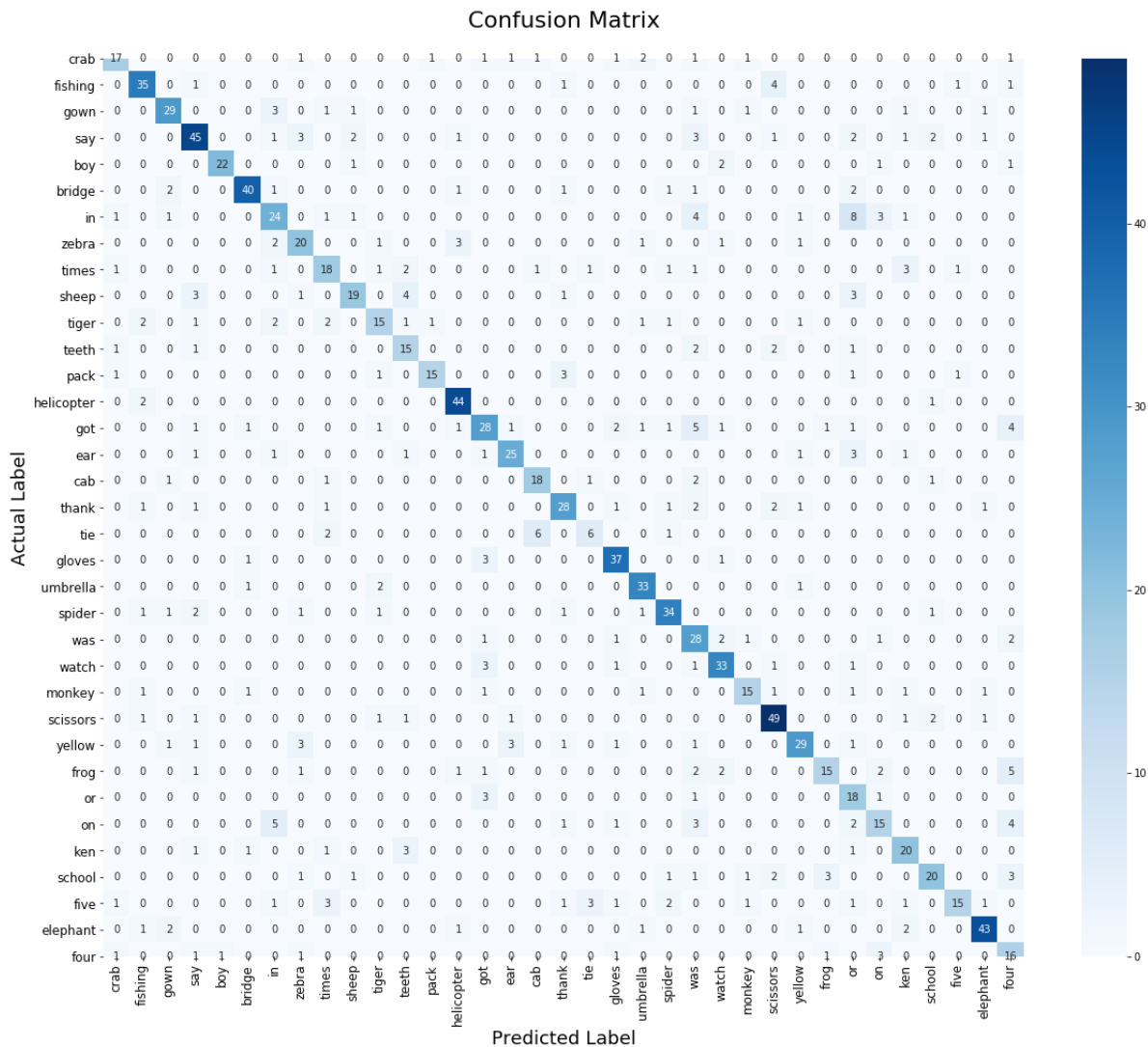
```
1 # Import necessary libraries for classification report
2 from sklearn.metrics import classification_report
3
4 preds = us_cnn_model.predict(us_X_test)
5
6 us_keywords = [
7     'crab',
8     'fishing',
9     'gown',
10    'say',
11    'boy',
12    'bridge',
13    'in',
14    'zebra',
15    'times',
16    'sheep',
17    'tiger',
18    'teeth',
19    'pack',
20    'helicopter',
21    'got',
22    'ear',
23    'cab',
24    'thank',
25    'tie',
26    'gloves',
27    'umbrella',
28    'spider',
29    'was',
30    'watch',
31    'monkey',
32    'scissors',
33    'yellow',
34    'frog',
35    'or',
36    'on',
37    'ken',
38    'school',
39    'five',
40    'elephant',
41    'four']
42
43 # Print the classification report
44 print(classification_report(us_test_y.argmax(axis=1),
45                             preds.argmax(axis=1),
46                             target_names=us_keywords))
```

	precision	recall	f1-score	support
crab	0.74	0.61	0.67	28
fishing	0.80	0.81	0.80	43
gown	0.78	0.76	0.77	38
say	0.74	0.73	0.73	62
boy	0.96	0.81	0.88	27
bridge	0.89	0.82	0.85	49
in	0.59	0.53	0.56	45
zebra	0.62	0.69	0.66	29
times	0.60	0.58	0.59	31

sheep	0.76	0.61	0.68	31
tiger	0.65	0.56	0.60	27
teeth	0.56	0.68	0.61	22
pack	0.88	0.68	0.77	22
helicopter	0.85	0.94	0.89	47
got	0.67	0.57	0.62	49
ear	0.81	0.74	0.77	34
cab	0.69	0.75	0.72	24
thank	0.74	0.72	0.73	39
tie	0.55	0.40	0.46	15
gloves	0.79	0.88	0.83	42
umbrella	0.80	0.89	0.85	37
spider	0.79	0.79	0.79	43
was	0.47	0.78	0.59	36
watch	0.79	0.82	0.80	40
monkey	0.75	0.65	0.70	23
scissors	0.79	0.84	0.82	58
yellow	0.81	0.71	0.75	41
frog	0.75	0.50	0.60	30
or	0.39	0.78	0.52	23
on	0.58	0.48	0.53	31
ken	0.62	0.74	0.68	27
school	0.74	0.61	0.67	33
five	0.83	0.48	0.61	31
elephant	0.88	0.84	0.86	51
four	0.43	0.64	0.52	25
accuracy			0.72	1233
macro avg	0.72	0.70	0.70	1233
weighted avg	0.73	0.72	0.72	1233

In [59]:

```
1 # Use Seaborn to make the confusion matrix more visually presentable
2 cm = confusion_matrix(us_test_y.argmax(axis=1),
3                       np.round(preds.argmax(axis=1)))
4
5 plt.figure(figsize=(20,16))
6 ax = plt.subplot()
7 sns.heatmap(cm, annot=True, ax=ax, fmt='g', cmap='Blues')
8
9 ax.set_title('Confusion Matrix', fontsize=22, pad=30)
10 ax.set_xlabel('Predicted Label', fontsize=18)
11 ax.set_ylabel('Actual Label', fontsize=18)
12 ax.xaxis.set_ticklabels(us_keywords, rotation=90, fontsize=12)
13 ax.yaxis.set_ticklabels(us_keywords, rotation=0, fontsize=12)
14 plt.show();
```



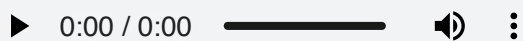
### 4.5.1 Making a prediction on an unseen audio sample

In [56]:

```
1 # Load the audio sample and preview
2 target_sample = 'audio/martha-umbrella.wav'
3 target_label = 'Umbrella'
4 audio_sample, sr = librosa.load(target_sample)
5 print('Audio sample:', target_label)
6 ipd.Audio(audio_sample, rate=sr)
```

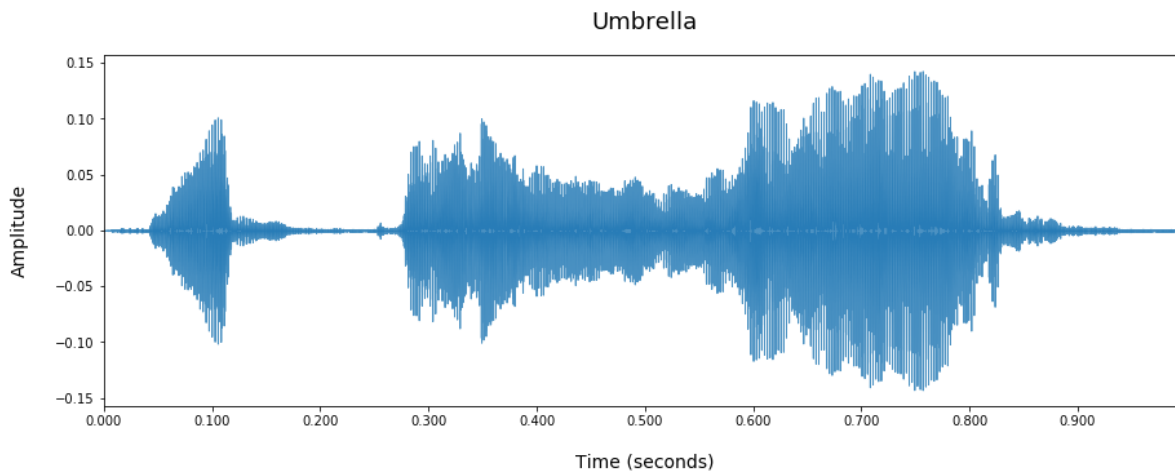
Audio sample: Umbrella

Out[56]:



In [57]:

```
1 # Plot the waveform for the specific audio sample
2 plt.figure(figsize=(15, 5))
3 plt.title(target_label, fontsize=18, pad=20)
4 librosa.display.waveplot(audio_sample, sr, alpha=0.8)
5 plt.xlabel('Time (seconds)', fontsize=14, labelpad=20)
6 plt.ylabel('Amplitude', fontsize=14, labelpad=20)
7 plt.show();
```



In [58]:

```
1 # Run inference on the unseen audio file
2 mfccs = librosa.feature.mfcc(audio_sample,
3                               sr,
4                               n_mfcc=13,
5                               n_fft=2048,
6                               hop_length=512)
7 mfccs = mfccs.T
8 mfccs = mfccs[np.newaxis, ..., np.newaxis]
9
10 prediction = us_cnn_model.predict(mfccs)
11 predicted_index = np.argmax(prediction)
12
13 predicted_keyword = us_keywords[predicted_index]
14 print('Martha says...', predicted_keyword, '!')
```

Martha says... umbrella !

## 4.6 Overall Conclusion

Initial models did not generalise well and tended to overfit the training data. Subsequent changes to parameters significantly improved the training accuracy but, again were overfitting the training data.

Whilst it does not have a high accuracy score, this final model using a Convolutional Neural Network produced the best results classifying over 70% of the “unseen” audio samples whilst minimising the overfitting to training data. Looking at the classification report, it is also clear that the model performs better for more identifiable words such as `scissors` or `umbrella` .

The very nature of speech sound disorders mean that a model that has been simply trained on audio samples of "typical" speech will generally be more accurate than one that has been trained on audio samples of "atypical" speech as demonstrated above with the model comparison between the Speech Commands and Ultrasuite datasets.



## 4.6.1 Recommendations

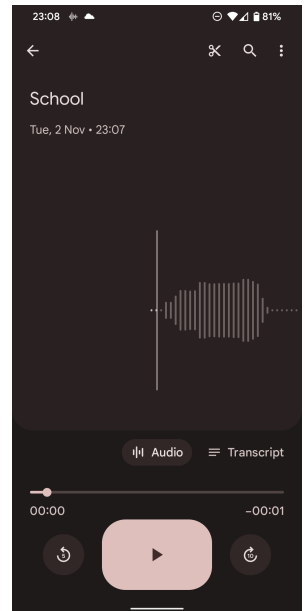
It is envisaged that the model would be used in the form of a mobile app, similar to that of Google Recorder (see image), that could be activated by the parent in order to capture the child speaking.

The app would also enable the parent to crop or isolate the audio sample and would provide a side-by-side prediction or transcript of what the child was saying.

This could even be provided as a setting within the Google recorder app itself that allowed the user, in this case the parent, to switch between the standard speech recognition model and our final model specifically trained for children with speech disorders.

Given this eventual usage of this model, it is arguable the app would be more useful if it suggested three potential words in order of likelihood, giving the parent options of what the child might be trying to communicate.

The accuracy of the final model is suitable enough to go ahead with a soft launch of the app to a controlled group of parents, in part for testing but also as a way of collecting additional data to improve the model.



## 4.6.2 Future Work

Future work to improve the model could include:

- Utilise other model architectures and enable them to accept longer audio samples as these are more representative of atypical speech patterns.
- Source additional data in the form of further audio samples potentially even using the app as a means for gathering additional samples and improving the model.
- Use data augmentation when training the model, in particular the [MixSpeech](https://arxiv.org/abs/2102.12664) (<https://arxiv.org/abs/2102.12664>) method that could take a weighted combination of mel-spectrograms and MFCCs in order to improve model performance.

---

### Sources / Code adapted from:

\* [Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow](https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/) - Aurélien Géron  
(<https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/>).

\* [Deep Learning Audio Application from Design to Deployment](https://github.com/musikalkemist/Deep-Learning-Audio-Application-From-Design-to-Deployment) - Valerio Velardo - The Sound of AI  
(<https://github.com/musikalkemist/Deep-Learning-Audio-Application-From-Design-to-Deployment>).