Leander_ACS61011_Project

March 20, 2024

1 Multiclass Automated Speech Recognition using a Baseline and an Advanced Model

1.0.1 Prerequisites

- Download the speech image transformed data from GitHub and unzip it in the current directory:

[43]:
```
# get the data from github and unzip
!wget https://raw.githubusercontent.com/andrsn/data/main/speechImageData.zip
!unzip -q /content/speechImageData.zip
!mv speechImageData\ -\ Copy speechImageData
```

```
--2024-03-19 08:58:45--
https://raw.githubusercontent.com/andrsn/data/main/speechImageData.zip
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.109.133, 185.199.111.133, 185.199.110.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.109.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 9872924 (9.4M) [application/zip]
Saving to: ‘speechImageData.zip.1’

speechImageData.zip 100%[===================>]   9.42M  --.-KB/s    in 0.05s

2024-03-19 08:58:45 (177 MB/s) - ‘speechImageData.zip.1’ saved [9872924/9872924]
```

- Install all the necessary libraries for our notebook

[2]:
```
!pip install scikeras pydub
```

```
Collecting scikeras
  Downloading scikeras-0.12.0-py3-none-any.whl (27 kB)
Collecting pydub
  Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Requirement already satisfied: packaging>=0.21 in
/usr/local/lib/python3.10/dist-packages (from scikeras) (24.0)
Requirement already satisfied: scikit-learn>=1.0.0 in
```

```
/usr/local/lib/python3.10/dist-packages (from scikeras) (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=1.0.0->scikeras) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=1.0.0->scikeras) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=1.0.0->scikeras) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras)
(3.3.0)
Installing collected packages: pydub, scikeras
Successfully installed pydub-0.25.1 scikeras-0.12.0
```

1.0.2 1. Import Libraries and define constants

We will start by importing the necessary libraries and defining the constants that will be used throughout the notebook.

[44]:
```python
import random
import shutil
import librosa
import soundfile as sf
import numpy as np
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt

from keras import optimizers, regularizers
from keras.models import Sequential
from keras.layers import Dense, Dropout, BatchNormalization, Input, Conv2D,␣
 ↪MaxPooling2D, Flatten
from keras.applications import MobileNetV2
from keras.applications.mobilenet_v2 import preprocess_input

from pydub import AudioSegment
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import RandomizedSearchCV
from scikeras.wrappers import KerasClassifier
from hyperopt import fmin, tpe, hp, STATUS_OK, Trials, space_eval


NUM_CLASSES = 12
BATCH_SIZE = 128
IMG_SIZE = (98, 50)
TIMEPOOL_SIZE = 12
```

1.0.3 2. Data Preprocessing

- Create usable keras dataset components from the extracted files.
- In total, there are 12 classes of different spoken words and the spectrograms, which form the input image data are of size 98x50 pixels.

[45]:
```python
# Load the data

train_ds = tf.keras.utils.image_dataset_from_directory(
    directory='/content/speechImageData/TrainData',
    labels='inferred',
    color_mode="grayscale",
    label_mode='categorical',
    batch_size=BATCH_SIZE,
    image_size=IMG_SIZE
)

val_ds = tf.keras.utils.image_dataset_from_directory(
    directory='/content/speechImageData/ValData',
    labels='inferred',
    color_mode="grayscale",
    label_mode='categorical',
    batch_size=BATCH_SIZE,
    image_size=IMG_SIZE
)

# Extract the training input images and output class labels
x_train = []
y_train = []
for images, labels in train_ds.take(-1):
    x_train.append(images.numpy())
    y_train.append(labels.numpy())

x_train = np.concatenate(x_train, axis=0)
y_train = np.concatenate(y_train, axis=0)

print(y_train)

# Extract the validation input images and output class labels
x_val = []
y_val = []
for images, labels in val_ds.take(-1):
    x_val.append(images.numpy())
    y_val.append(labels.numpy())

x_val = np.concatenate(x_val, axis=0)
y_val = np.concatenate(y_val, axis=0)
```

```python
print(y_val)
```

```
Found 2023 files belonging to 12 classes.
Found 1181 files belonging to 12 classes.
[[0. 0. 1. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 1.]
 [0. 0. 0. … 0. 0. 0.]
 …
 [0. 0. 0. … 0. 0. 0.]
 [1. 0. 0. … 0. 0. 0.]
 [0. 1. 0. … 0. 0. 0.]]
[[0. 0. 0. … 0. 0. 0.]
 [0. 1. 0. … 0. 0. 0.]
 [1. 0. 0. … 0. 0. 0.]
 …
 [0. 0. 0. … 1. 0. 0.]
 [0. 1. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]]
```

1.0.4 3. Model Design

We now attempt to approach the problem with five tasks:

Task 1: Baseline Model

Model features: The following are its features:

- The baseline model is a simple Convolutional Neural Network (CNN) with one input layer, four hidden layers, one fully connected layer and an output layer.
- The input layer consists of the following:
  - A Conv2D layer with 32 filters, a kernel size of 3x3, and a ReLU activation function.
  - A BatchNormalization layer.
  - A MaxPooling2D layer with a pool size of 2x2.
- The four hidden layers are replicated from the input layer.
- A time pooling layer is added to the model to combat the start time of the audio.
- The fully connected layer consists of 1024 units and a ReLU activation function.
- The output layer consists of 12 units and a softmax activation function.

Additional model hyperparameters:

- The model uses the Adam optimizer with a learning rate of 0.001.
- The L2 regularization parameter is set to 0.001.
- While training, an early stopping callback is used to stop the training process if the validation accuracy does not decrease for 5 epochs.

[5]:
```python
# define t1 model
def t1_model(num_layers, num_filters, passthrough=False):
    # number of convolutional filters
    input_num_filters = 32
    fully_connected_num_filters = 1024

    # define model
    model = Sequential()

    # input layer
    model.add(Input(shape=(IMG_SIZE[0], IMG_SIZE[1], 1)))
    model.add(Conv2D(input_num_filters, kernel_size =(3, 3), padding='same',␣
↪activation='relu'))
    model.add(BatchNormalization())

    # hidden layers
    for i in range(num_layers):
        if passthrough:
            model.add(Conv2D(num_filters[i], kernel_size =(3, 3),␣
↪padding='same', activation='relu'))
        else:
            model.add(Conv2D(num_filters, kernel_size =(3, 3), padding='same',␣
↪activation='relu'))
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size =(2, 2), strides=(2, 2),␣
↪padding='same'))

    # Time based pooling
    model.add(MaxPooling2D(pool_size=(TIMEPOOL_SIZE, 1),␣
↪strides=(TIMEPOOL_SIZE, 1), padding='same'))

    # fully connected layer
    model.add(Flatten())
    model.add(Dense(fully_connected_num_filters,␣
↪kernel_regularizer=regularizers.l2(0.01), activation='relu'))
    model.add(Dropout(0.2))

    # output layer
    model.add(Dense(NUM_CLASSES, activation='softmax'))

    # set adam optimizer
    opt = optimizers.Adam(learning_rate=0.001)
    model.compile(loss="categorical_crossentropy", optimizer=opt,␣
↪metrics=["accuracy"])

    return model
```

```
hidden_num_layers = [2, 3, 4, 5]
hidden_num_filters = [128, 128, 128, 128]
task1_model = t1_model(len(hidden_num_layers), hidden_num_filters,␣
 ↪passthrough=True)
task1_model.summary()

early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',␣
 ↪patience=5)
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 98, 50, 32)        320

 batch_normalization (Batch  (None, 98, 50, 32)        128
 Normalization)

 conv2d_1 (Conv2D)           (None, 98, 50, 128)       36992

 batch_normalization_1 (Bat  (None, 98, 50, 128)       512
 chNormalization)

 max_pooling2d (MaxPooling2  (None, 49, 25, 128)       0
 D)

 conv2d_2 (Conv2D)           (None, 49, 25, 128)       147584

 batch_normalization_2 (Bat  (None, 49, 25, 128)       512
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 25, 13, 128)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 25, 13, 128)       147584

 batch_normalization_3 (Bat  (None, 25, 13, 128)       512
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 13, 7, 128)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 13, 7, 128)        147584

 batch_normalization_4 (Bat  (None, 13, 7, 128)        512
 chNormalization)
```

```
 max_pooling2d_3 (MaxPoolin  (None, 7, 4, 128)         0
 g2D)

 max_pooling2d_4 (MaxPoolin  (None, 1, 4, 128)         0
 g2D)

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 1024)              525312

 dropout (Dropout)           (None, 1024)              0

 dense_1 (Dense)             (None, 12)                12300

=================================================================
Total params: 1019852 (3.89 MB)
Trainable params: 1018764 (3.89 MB)
Non-trainable params: 1088 (4.25 KB)
_________________________________________________________________
```

T1.A - Model Training This section trains the deep convolutional network using the Adam algorithm.

[6]:
```
history = task1_model.fit(x_train, y_train, batch_size=BATCH_SIZE, epochs=15,␣
↪validation_data=(x_val, y_val), callbacks=[early_stopping])
```

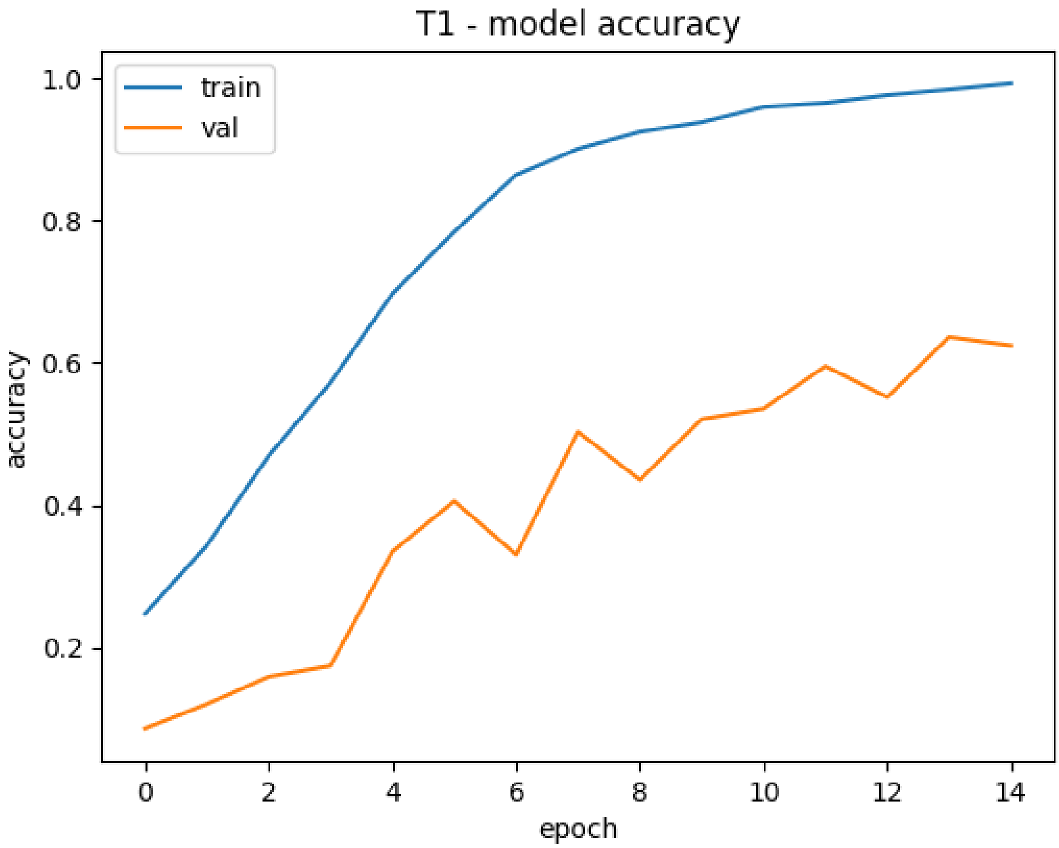
```
Epoch 1/15
16/16 [==============================] - 17s 369ms/step - loss: 10.7208 -
accuracy: 0.2474 - val_loss: 14.2474 - val_accuracy: 0.0863
Epoch 2/15
16/16 [==============================] - 2s 128ms/step - loss: 7.5581 -
accuracy: 0.3433 - val_loss: 11.3006 - val_accuracy: 0.1204
Epoch 3/15
16/16 [==============================] - 2s 128ms/step - loss: 6.4265 -
accuracy: 0.4693 - val_loss: 8.7848 - val_accuracy: 0.1588
Epoch 4/15
16/16 [==============================] - 2s 131ms/step - loss: 5.5289 -
accuracy: 0.5722 - val_loss: 7.1032 - val_accuracy: 0.1742
Epoch 5/15
16/16 [==============================] - 2s 135ms/step - loss: 4.6883 -
accuracy: 0.6972 - val_loss: 5.4696 - val_accuracy: 0.3348
Epoch 6/15
16/16 [==============================] - 2s 132ms/step - loss: 3.9862 -
accuracy: 0.7841 - val_loss: 4.8391 - val_accuracy: 0.4056
Epoch 7/15
16/16 [==============================] - 2s 132ms/step - loss: 3.4144 -
accuracy: 0.8641 - val_loss: 4.8459 - val_accuracy: 0.3305
```

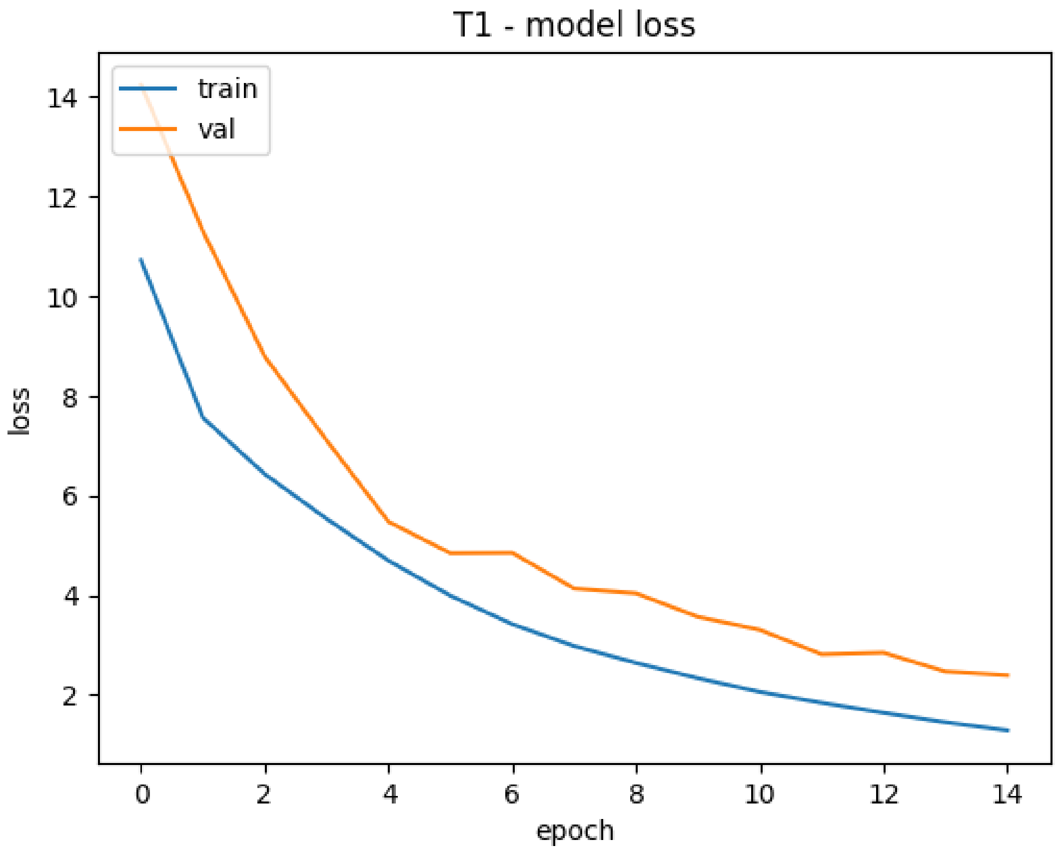
```
Epoch 8/15
16/16 [==============================] - 2s 129ms/step - loss: 2.9752 -
accuracy: 0.9005 - val_loss: 4.1311 - val_accuracy: 0.5030
Epoch 9/15
16/16 [==============================] - 2s 129ms/step - loss: 2.6376 -
accuracy: 0.9245 - val_loss: 4.0338 - val_accuracy: 0.4355
Epoch 10/15
16/16 [==============================] - 2s 131ms/step - loss: 2.3295 -
accuracy: 0.9380 - val_loss: 3.5626 - val_accuracy: 0.5209
Epoch 11/15
16/16 [==============================] - 2s 132ms/step - loss: 2.0557 -
accuracy: 0.9595 - val_loss: 3.3018 - val_accuracy: 0.5354
Epoch 12/15
16/16 [==============================] - 2s 142ms/step - loss: 1.8363 -
accuracy: 0.9650 - val_loss: 2.8098 - val_accuracy: 0.5952
Epoch 13/15
16/16 [==============================] - 2s 142ms/step - loss: 1.6353 -
accuracy: 0.9760 - val_loss: 2.8411 - val_accuracy: 0.5517
Epoch 14/15
16/16 [==============================] - 2s 130ms/step - loss: 1.4460 -
accuracy: 0.9840 - val_loss: 2.4654 - val_accuracy: 0.6362
Epoch 15/15
16/16 [==============================] - 2s 133ms/step - loss: 1.2841 -
accuracy: 0.9925 - val_loss: 2.3897 - val_accuracy: 0.6243
```

T1.B - Plot Training History and Confusion Matrix Here, we plot the training history and confusion matrix of the baseline model.

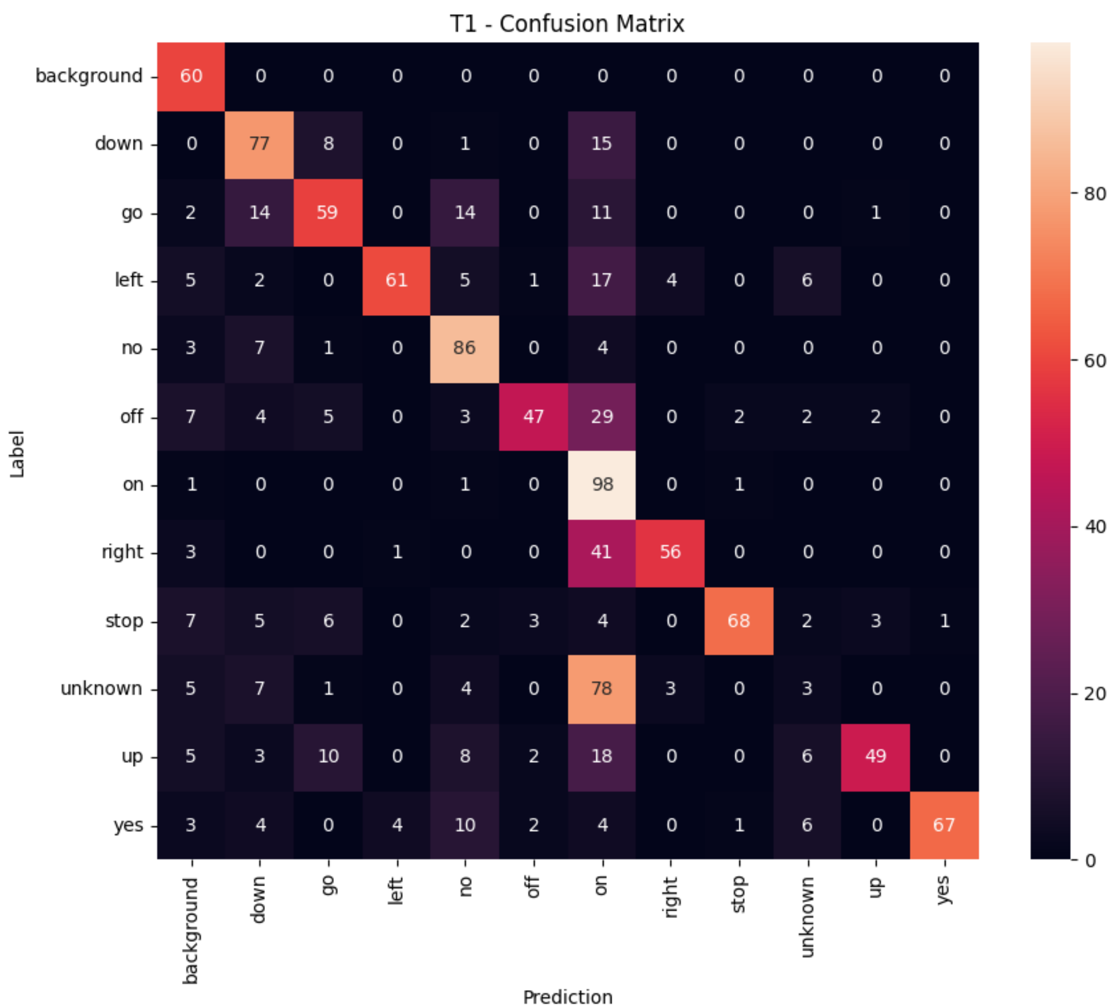
[7]:
```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('T1 - model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('T1 - model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# Print accuracy
score = task1_model.evaluate(x_val, y_val, verbose=0)
```

```
print('T1 - validation accuracy:', score[1])

# Print confusion matrix
y_pred = task1_model.predict(x_val)
y_pred = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_val, axis=1)
class_labels = list(val_ds.class_names)
confusion_mtx = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx, xticklabels=class_labels, yticklabels=class_labels,␣
 ↪annot=True, fmt='d')
plt.xlabel('Prediction')
plt.ylabel('Label')
plt.title('T1 - Confusion Matrix')
plt.show()
```





T1 - validation accuracy: 0.6242527961730957
37/37 [==============================] - 0s 8ms/step



As we can see, the model is not performing well on the validation set, especially for the class 'unknown'. But that is expected as the class is a blanket of words not detected by any other classes. Overall, this baseline model achieves a validation accuracy of around 0.6.

We now proceed to the next task.

Task 2: Random Search for Hyperparameter Tuning We now use random search to find the best hyperparameters for the baseline model. Random search is better than grid search because it is faster and more efficient. Here, we use the RandomSearchCV class from the scikit-learn library to find the best hyperparameters for the baseline model.

[8]:
```
# define the random search parameters
param_grid = {
    'num_layers': hidden_num_layers,
    'num_filters': hidden_num_filters,
}
```

```
# Create a KerasClassifier
task2_model = KerasClassifier(model=t1_model, epochs=40, batch_size=BATCH_SIZE,␣
↪verbose=1, num_layers=hidden_num_layers, num_filters=hidden_num_filters)

# Create a RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=task2_model,␣
↪param_distributions=param_grid, n_iter=2, cv=3, verbose=2)

# Fit the RandomizedSearchCV
random_result = random_search.fit(x_train, y_train, validation_data=(x_val,␣
↪y_val), callbacks=[early_stopping], verbose=1)

# Summarize results
print("Best: %f using %s" % (random_result.best_score_, random_result.
↪best_params_))
```

```
Fitting 3 folds for each of 2 candidates, totalling 6 fits
Epoch 1/40
11/11 [==============================] - 8s 393ms/step - loss: 11.3078 -
accuracy: 0.2264 - val_loss: 13.8767 - val_accuracy: 0.0931
Epoch 2/40
11/11 [==============================] - 2s 141ms/step - loss: 8.2811 -
accuracy: 0.2879 - val_loss: 11.5925 - val_accuracy: 0.1178
Epoch 3/40
11/11 [==============================] - 1s 135ms/step - loss: 7.3522 -
accuracy: 0.3598 - val_loss: 9.1822 - val_accuracy: 0.1776
Epoch 4/40
11/11 [==============================] - 1s 135ms/step - loss: 6.5350 -
accuracy: 0.4558 - val_loss: 8.1694 - val_accuracy: 0.1067
Epoch 5/40
11/11 [==============================] - 1s 137ms/step - loss: 5.7682 -
accuracy: 0.5840 - val_loss: 7.3425 - val_accuracy: 0.1119
Epoch 6/40
11/11 [==============================] - 1s 136ms/step - loss: 5.0496 -
accuracy: 0.6769 - val_loss: 6.1923 - val_accuracy: 0.1827
Epoch 7/40
11/11 [==============================] - 1s 136ms/step - loss: 4.4414 -
accuracy: 0.7774 - val_loss: 6.1963 - val_accuracy: 0.1324
Epoch 8/40
11/11 [==============================] - 1s 136ms/step - loss: 3.8705 -
accuracy: 0.8658 - val_loss: 5.4464 - val_accuracy: 0.1887
Epoch 9/40
11/11 [==============================] - 2s 142ms/step - loss: 3.4794 -
accuracy: 0.8808 - val_loss: 4.8642 - val_accuracy: 0.3501
Epoch 10/40
11/11 [==============================] - 1s 139ms/step - loss: 3.1433 -
```

```
accuracy: 0.9915 - val_loss: 1.4961 - val_accuracy: 0.7301
Epoch 39/40
16/16 [==============================] - 2s 139ms/step - loss: 0.1419 -
accuracy: 0.9990 - val_loss: 1.3936 - val_accuracy: 0.7173
Epoch 40/40
16/16 [==============================] - 2s 141ms/step - loss: 0.1210 -
accuracy: 1.0000 - val_loss: 1.2234 - val_accuracy: 0.7199
Best: 0.845077 using {'num_layers': 4, 'num_filters': 128}
```

As seen during the cross validation split of the Randomized Search for the best fit, the accuracy of the model has improved to 0.84. The best values for the hyperparameters are: * `num_layers`: 4 * `num_filters`: 128

We now redefine the model with these hyperparameters.

[9]:
```python
hidden_num_filters = [128, 128, 128, 128]
hidden_num_layers = [2, 3, 4, 5]

task2_model = t1_model(len(hidden_num_layers), hidden_num_filters,
 ↪passthrough=True)
task2_model.summary()
```

```
Model: "sequential_8"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_34 (Conv2D)          (None, 98, 50, 32)        320

 batch_normalization_34 (Ba  (None, 98, 50, 32)        128
 tchNormalization)

 conv2d_35 (Conv2D)          (None, 98, 50, 128)       36992

 batch_normalization_35 (Ba  (None, 98, 50, 128)       512
 tchNormalization)

 max_pooling2d_34 (MaxPooli  (None, 49, 25, 128)       0
 ng2D)

 conv2d_36 (Conv2D)          (None, 49, 25, 128)       147584

 batch_normalization_36 (Ba  (None, 49, 25, 128)       512
 tchNormalization)

 max_pooling2d_35 (MaxPooli  (None, 25, 13, 128)       0
 ng2D)

 conv2d_37 (Conv2D)          (None, 25, 13, 128)       147584
```

```
 batch_normalization_37 (Ba  (None, 25, 13, 128)       512
 tchNormalization)

 max_pooling2d_36 (MaxPooli  (None, 13, 7, 128)        0
 ng2D)

 conv2d_38 (Conv2D)          (None, 13, 7, 128)        147584

 batch_normalization_38 (Ba  (None, 13, 7, 128)        512
 tchNormalization)

 max_pooling2d_37 (MaxPooli  (None, 7, 4, 128)         0
 ng2D)

 max_pooling2d_38 (MaxPooli  (None, 1, 4, 128)         0
 ng2D)

 flatten_8 (Flatten)         (None, 512)               0

 dense_16 (Dense)            (None, 1024)              525312

 dropout_8 (Dropout)         (None, 1024)              0

 dense_17 (Dense)            (None, 12)                12300

=================================================================
Total params: 1019852 (3.89 MB)
Trainable params: 1018764 (3.89 MB)
Non-trainable params: 1088 (4.25 KB)
_________________________________________________________________
```

Task 3: Model Averaging Scheme Here, we randomly sample three subsets of the training data and train the T2 model on each of them. We then proceed with a series of voting schemes to determine the final prediction.

[10]:
```
# define number of samples
train_samples = 2001

# create data index
data_index = list(range(1, train_samples))

subsets = 3

# keep track of history
shuffle_history = []

for i in range(subsets):
```

```
    # create random index using sampling with replacement
    idx = random.choices(data_index, k=train_samples)

    # define first shuffle
    x_train_shuffle = np.zeros([train_samples, IMG_SIZE[0], IMG_SIZE[1], 1])
    y_train_shuffle = np.zeros([train_samples, NUM_CLASSES])

    # resample the data
    for j in range(train_samples):
        x_train_shuffle[j] = x_train[idx[j], :, :, :]
        y_train_shuffle[j] = y_train[idx[j], :]

    # train the model
    shuffle_history.append(task2_model.fit(x_train_shuffle, y_train_shuffle,␣
 ↪batch_size=BATCH_SIZE, epochs=40, validation_data=(x_val, y_val),␣
 ↪callbacks=[early_stopping]))


# Take the majority class prediction and use the mode for all the three models␣
 ↪to determine final prediction
# Create a matrix of predictions
y_pred = np.zeros([len(y_val), subsets])
# iterate through the models
for i in range(subsets):
    # save the predictions
    y_pred[:, i] = np.argmax(shuffle_history[i].model.predict(x_val), axis=1)

# convert to integer
y_pred = np.array(y_pred, dtype=int)
# take the mode
y_pred = np.squeeze(np.apply_along_axis(lambda x: np.bincount(x).argmax(),␣
 ↪axis=1, arr=y_pred))

# Print accuracy
score = task2_model.evaluate(x_val, y_val, verbose=0)
print('T3 - validation accuracy:', score[1])
```

```
Epoch 1/40
16/16 [==============================] - 5s 160ms/step - loss: 10.5197 -
accuracy: 0.2214 - val_loss: 18.5219 - val_accuracy: 0.1016
Epoch 2/40
16/16 [==============================] - 2s 153ms/step - loss: 7.5967 -
accuracy: 0.3593 - val_loss: 20.7082 - val_accuracy: 0.0897
Epoch 3/40
16/16 [==============================] - 2s 143ms/step - loss: 6.4641 -
accuracy: 0.4873 - val_loss: 16.3425 - val_accuracy: 0.1332
Epoch 4/40
```

```
16/16 [==============================] - 2s 135ms/step - loss: 0.1437 -
accuracy: 0.9995 - val_loss: 1.2630 - val_accuracy: 0.7071
Epoch 9/40
16/16 [==============================] - 2s 135ms/step - loss: 0.1284 -
accuracy: 0.9990 - val_loss: 1.2872 - val_accuracy: 0.7182
Epoch 10/40
16/16 [==============================] - 2s 135ms/step - loss: 0.1115 -
accuracy: 0.9995 - val_loss: 1.2885 - val_accuracy: 0.7242
Epoch 11/40
16/16 [==============================] - 2s 134ms/step - loss: 0.0994 -
accuracy: 0.9995 - val_loss: 1.1652 - val_accuracy: 0.7370
Epoch 12/40
16/16 [==============================] - 2s 134ms/step - loss: 0.0875 -
accuracy: 0.9995 - val_loss: 1.1433 - val_accuracy: 0.7387
Epoch 13/40
16/16 [==============================] - 2s 136ms/step - loss: 0.0755 -
accuracy: 0.9995 - val_loss: 1.1431 - val_accuracy: 0.7395
Epoch 14/40
16/16 [==============================] - 2s 139ms/step - loss: 0.0687 -
accuracy: 0.9995 - val_loss: 1.0792 - val_accuracy: 0.7515
Epoch 15/40
16/16 [==============================] - 2s 136ms/step - loss: 0.0598 -
accuracy: 0.9995 - val_loss: 1.1098 - val_accuracy: 0.7404
Epoch 16/40
16/16 [==============================] - 2s 143ms/step - loss: 0.0534 -
accuracy: 0.9995 - val_loss: 1.0631 - val_accuracy: 0.7455
Epoch 17/40
16/16 [==============================] - 2s 143ms/step - loss: 0.0478 -
accuracy: 1.0000 - val_loss: 1.1018 - val_accuracy: 0.7421
Epoch 18/40
16/16 [==============================] - 2s 137ms/step - loss: 0.0429 -
accuracy: 1.0000 - val_loss: 1.0780 - val_accuracy: 0.7387
Epoch 19/40
16/16 [==============================] - 2s 138ms/step - loss: 0.0387 -
accuracy: 1.0000 - val_loss: 1.0424 - val_accuracy: 0.7421
37/37 [==============================] - 0s 10ms/step
37/37 [==============================] - 0s 8ms/step
37/37 [==============================] - 0s 8ms/step
T3 - validation accuracy: 0.7421007752418518
```

After resampling for three times, the accuracy drops to about 0.75.

This is maybe an indication that resampling with replacement reduces the overall accuracy of the model due to a drop in the amount of data available for training.

1.0.5 Task 4: Hyperparameter Tuning using Bayesian Optimization

We now use Bayesian optimization to find the best hyperparameters for the advanced model.

Bayesian optimization is a probabilistic model-based optimization algorithm that is used to find the best hyperparameters for a model.

[12]:
```
# Define the search space for hyperparameters
space = {'num_layers': hp.choice('num_layers', [1, 2, 3, 4, 5]),
    'num_filters': hp.choice('num_filters', [8, 16, 32, 64, 128]),}

# Define the objective function
def t4_model(params):
    # number of convolutional filters
    input_num_filters = 32
    fully_connected_num_filters = 1024

    # define model
    model = Sequential()

    # input layer
    model.add(Input(shape=(IMG_SIZE[0], IMG_SIZE[1], 1)))
    model.add(Conv2D(input_num_filters, kernel_size =(3, 3), padding='same',␣
 ↪activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size =(2, 2), strides=(2, 2), padding='same'))

    # hidden layers
    for _ in range(params['num_layers']):
        model.add(Conv2D(params['num_filters'], kernel_size =(3, 3),␣
 ↪padding='same', activation='relu'))
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size =(2, 2), strides=(2, 2),␣
 ↪padding='same'))

    # Time based pooling
    model.add(MaxPooling2D(pool_size=(TIMEPOOL_SIZE, 1),␣
 ↪strides=(TIMEPOOL_SIZE, 1), padding='same'))

    # fully connected layer
    model.add(Flatten())
    model.add(Dense(fully_connected_num_filters,␣
 ↪kernel_regularizer=regularizers.l2(0.01), activation='relu'))
    model.add(Dropout(0.2))

    # output layer
    model.add(Dense(NUM_CLASSES, activation='softmax'))

    # set adam optimizer
    opt = optimizers.Adam(learning_rate=0.001)
```

```
    model.compile(loss="categorical_crossentropy", optimizer=opt,␣
↪metrics=["accuracy"])

    # train the model
    model.fit(x_train, y_train, batch_size=int(BATCH_SIZE/6), epochs=30,␣
↪validation_data=(x_val, y_val), callbacks=[early_stopping])

    # evaluate the model
    _, accuracy = model.evaluate(x_val, y_val, verbose=0)

    return {'loss': -accuracy, 'status': STATUS_OK}

# Run the hyperparameter search
best = fmin(fn=t4_model, space=space, algo=tpe.suggest, max_evals=20)

# Print the best hyperparameters of number of layers and number of filters
print(space_eval(space, best))
```

```
Streaming output truncated to the last 5000 lines.
62/96 [==================>…] - ETA: 0s - loss: 0.4662 - accuracy:
0.9201
66/96 [===================>…] - ETA: 0s - loss: 0.4668 - accuracy:
0.9214
70/96 [====================>…] - ETA: 0s - loss: 0.4728 - accuracy:
0.9197
74/96 [======================>…] - ETA: 0s - loss: 0.4719 - accuracy:
0.9196
78/96 [=======================>…] - ETA: 0s - loss: 0.4745 - accuracy:
0.9182
82/96 [========================>…] - ETA: 0s - loss: 0.4721 - accuracy:
0.9199
86/96 [=========================>…] - ETA: 0s - loss: 0.4757 - accuracy:
0.9208
90/96 [===========================>..] - ETA: 0s - loss: 0.4769 - accuracy:
0.9206
94/96 [============================>.] - ETA: 0s - loss: 0.4783 - accuracy:
0.9184
96/96 [==============================] - 2s 18ms/step - loss: 0.4758 - accuracy:
0.9195 - val_loss: 1.5420 - val_accuracy: 0.6396

Epoch 11/30

 1/96 […] - ETA: 2s - loss: 0.8127 - accuracy:
0.9048
 5/96 [>…] - ETA: 1s - loss: 0.6106 - accuracy:
0.9048
 9/96 [=>…] - ETA: 1s - loss: 0.4971 - accuracy:
```

```
87/96 [=========================>…] - ETA: 0s - loss: 0.3880 - accuracy:
0.9152
92/96 [===========================>..] - ETA: 0s - loss: 0.3915 - accuracy:
0.9136
96/96 [==============================] - 2s 16ms/step - loss: 0.3947 - accuracy:
0.9120 - val_loss: 1.5834 - val_accuracy: 0.6072

100%|     | 20/20 [14:19<00:00, 42.99s/trial, best loss:
-0.716481626033783]
{'num_filters': 128, 'num_layers': 4}
```

The Bayesian optimization gives us the best hyperparameters for the advanced model as: * `num_layers`: 4 * `num_filters`: 128

Overall, the accuracy of the advanced model is about as similar as Model Averaging at 0.72. This seems to find the best parameters as that of Random Search. But there is a drop in accuracy during pursuit of these hyperparameters. This can be due to overtraining the model leading to a overfitted model.

Task 5: Data Augmentation, Model Validation of Personal Voice Recordings, and Transfer Learning First we record our own voice for about 10 times in two classes 'yes' and 'no'. We also ensure that the recordings are of the same length as the dataset recordings, which is of 1s each.

We augment the data by adding random noise to the recordings to generate more data. We randomly add 5% noise to the recordings.

[14]:
```
!unzip /content/personalRecordings.zip
```

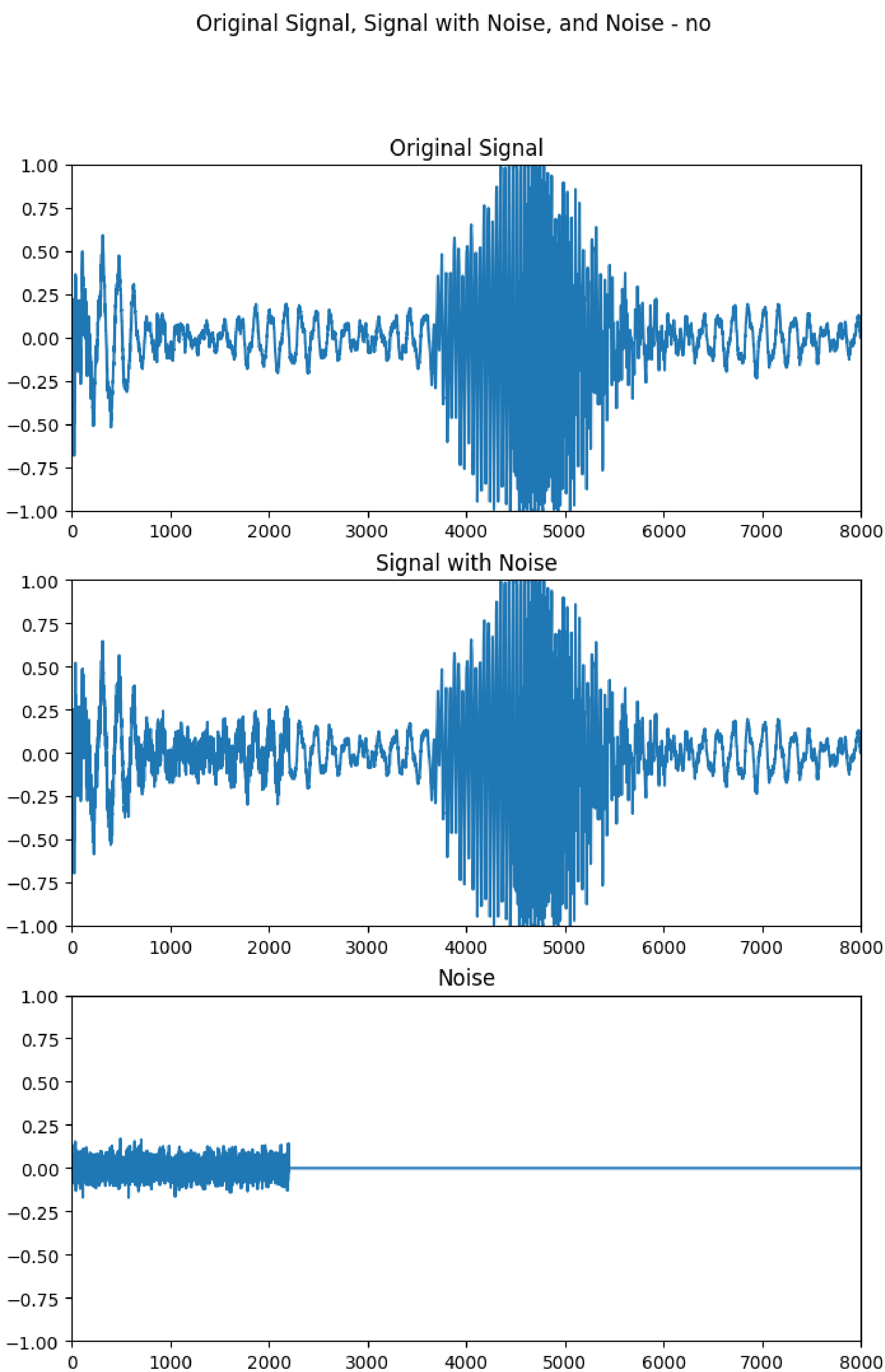
```
Archive:  /content/personalRecordings.zip
   creating: personalRecordings/
   creating: personalRecordings/no/
  inflating: personalRecordings/no/0.wav
  inflating: personalRecordings/no/1.wav
  inflating: personalRecordings/no/2.wav
  inflating: personalRecordings/no/3.wav
  inflating: personalRecordings/no/4.wav
  inflating: personalRecordings/no/5.wav
  inflating: personalRecordings/no/6.wav
  inflating: personalRecordings/no/7.wav
  inflating: personalRecordings/no/8.wav
  inflating: personalRecordings/no/9.wav
  inflating: personalRecordings/no/0_with_noise_0.wav
  inflating: personalRecordings/no/0_with_noise_1.wav
  inflating: personalRecordings/no/1_with_noise_0.wav
  inflating: personalRecordings/no/1_with_noise_1.wav
  inflating: personalRecordings/no/2_with_noise_0.wav
  inflating: personalRecordings/no/2_with_noise_1.wav
  inflating: personalRecordings/no/3_with_noise_0.wav
```

```
  inflating: personalRecordings/no/3_with_noise_1.wav
  inflating: personalRecordings/no/4_with_noise_0.wav
  inflating: personalRecordings/no/4_with_noise_1.wav
  inflating: personalRecordings/no/5_with_noise_0.wav
  inflating: personalRecordings/no/5_with_noise_1.wav
  inflating: personalRecordings/no/6_with_noise_0.wav
  inflating: personalRecordings/no/6_with_noise_1.wav
  inflating: personalRecordings/no/7_with_noise_0.wav
  inflating: personalRecordings/no/7_with_noise_1.wav
  inflating: personalRecordings/no/8_with_noise_0.wav
  inflating: personalRecordings/no/8_with_noise_1.wav
  inflating: personalRecordings/no/9_with_noise_0.wav
  inflating: personalRecordings/no/9_with_noise_1.wav
   creating: personalRecordings/yes/
  inflating: personalRecordings/yes/0.wav
  inflating: personalRecordings/yes/1.wav
  inflating: personalRecordings/yes/2.wav
  inflating: personalRecordings/yes/3.wav
  inflating: personalRecordings/yes/4.wav
  inflating: personalRecordings/yes/5.wav
  inflating: personalRecordings/yes/6.wav
  inflating: personalRecordings/yes/7.wav
  inflating: personalRecordings/yes/8.wav
  inflating: personalRecordings/yes/9.wav
  inflating: personalRecordings/yes/0_with_noise_0.wav
  inflating: personalRecordings/yes/0_with_noise_1.wav
  inflating: personalRecordings/yes/1_with_noise_0.wav
  inflating: personalRecordings/yes/1_with_noise_1.wav
  inflating: personalRecordings/yes/2_with_noise_0.wav
  inflating: personalRecordings/yes/2_with_noise_1.wav
  inflating: personalRecordings/yes/3_with_noise_0.wav
  inflating: personalRecordings/yes/3_with_noise_1.wav
  inflating: personalRecordings/yes/4_with_noise_0.wav
  inflating: personalRecordings/yes/4_with_noise_1.wav
  inflating: personalRecordings/yes/5_with_noise_0.wav
  inflating: personalRecordings/yes/5_with_noise_1.wav
  inflating: personalRecordings/yes/6_with_noise_0.wav
  inflating: personalRecordings/yes/6_with_noise_1.wav
  inflating: personalRecordings/yes/7_with_noise_0.wav
  inflating: personalRecordings/yes/7_with_noise_1.wav
  inflating: personalRecordings/yes/8_with_noise_0.wav
  inflating: personalRecordings/yes/8_with_noise_1.wav
  inflating: personalRecordings/yes/9_with_noise_0.wav
  inflating: personalRecordings/yes/9_with_noise_1.wav
```

[20]:
```
recorded_classes = ['no', 'yes']
num_recordings = 10
```

```
num_synthetic_recordings = 2
noise_percentage = 0.01

viewing_label = True

for word in recorded_classes:
    for i in range(num_recordings):
        sound_path = f'personalRecordings/{word}/{i}.wav'

        # trim the audio file to 1 second
        sound = AudioSegment.from_file(sound_path)
        sound = sound[:1000]
        sound.export(sound_path, format="wav")

        # capture the signal and sample rate
        signal, sr = librosa.load(sound_path, sr=8000)

        for j in range(num_synthetic_recordings):
            # add noise to the signal to only a part of the signal
            random_index = random.randint(0, len(signal) - 1)
            noise = np.random.normal(0, noise_percentage, random_index)
            signal_with_noise = signal.copy()
            signal_with_noise[:random_index] += noise

            signal_noise = signal_with_noise - signal

            # export the signal with noise
            sound_with_noise_path = f'personalRecordings/{word}/
 ↪{i}_with_noise_{j}.wav'
            sf.write(sound_with_noise_path, signal_with_noise, sr)

        if viewing_label:
            viewing_label = False
            # use subplots to plot the original signal, signal with noise and␣
 ↪noise
            fig, axs = plt.subplots(3, figsize=(8, 12))
            fig.suptitle(f'Original Signal, Signal with Noise, and Noise -␣
 ↪{word}')

            # set x-axis and y-axis limits
            axs[0].set_xlim(0, len(signal))
            axs[0].set_ylim(-1, 1)

            axs[1].set_xlim(0, len(signal))
            axs[1].set_ylim(-1, 1)

            axs[2].set_xlim(0, len(signal))
```

```
    axs[2].set_ylim(-1, 1)

    axs[0].plot(signal)
    axs[0].set_title('Original Signal')
    axs[1].plot(signal_with_noise)
    axs[1].set_title('Signal with Noise')
    axs[2].plot(signal_noise)
    axs[2].set_title('Noise')
    plt.show()
```


Original Signal, Signal with Noise, and Noise - no
Original Signal
Signal with Noise
Noise
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
1000
2000
3000
4000
5000
6000
7000
8000

As you can see, random indices are chosen to add noise to the recordings. So, for every iteration, the noise added to the recordings is different.

We now convert the recordings to spectrograms and use the advanced model to predict the classes of the recordings. This is done by the helper script provided in MATLAB.

First let's analyze the confusion matrix of the model on the personal recordings.

T5.1 Analysis of Confusion matrix on 'yes' and 'no' classes on Advanced Model: We now use the random search model hyperparameters to baseoff our personal recordings against.

[46]:
```
task5_model = t1_model(len([2, 3, 4, 5]), [128, 128, 128, 128],␣
  ↪passthrough=True)
task5_model.summary()
```

```
Model: "sequential_38"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_146 (Conv2D)         (None, 98, 50, 32)        320

 batch_normalization_146 (B  (None, 98, 50, 32)        128
 atchNormalization)

 conv2d_147 (Conv2D)         (None, 98, 50, 128)       36992

 batch_normalization_147 (B  (None, 98, 50, 128)       512
 atchNormalization)

 max_pooling2d_167 (MaxPool  (None, 49, 25, 128)       0
 ing2D)

 conv2d_148 (Conv2D)         (None, 49, 25, 128)       147584

 batch_normalization_148 (B  (None, 49, 25, 128)       512
 atchNormalization)

 max_pooling2d_168 (MaxPool  (None, 25, 13, 128)       0
 ing2D)

 conv2d_149 (Conv2D)         (None, 25, 13, 128)       147584

 batch_normalization_149 (B  (None, 25, 13, 128)       512
 atchNormalization)

 max_pooling2d_169 (MaxPool  (None, 13, 7, 128)        0
```

```
ing2D)

 conv2d_150 (Conv2D)         (None, 13, 7, 128)        147584

 batch_normalization_150 (B  (None, 13, 7, 128)        512
 atchNormalization)

 max_pooling2d_170 (MaxPool  (None, 7, 4, 128)         0
 ing2D)

 max_pooling2d_171 (MaxPool  (None, 1, 4, 128)         0
 ing2D)

 flatten_38 (Flatten)        (None, 512)               0

 dense_76 (Dense)            (None, 1024)              525312

 dropout_38 (Dropout)        (None, 1024)              0

 dense_77 (Dense)            (None, 12)                12300

=================================================================
Total params: 1019852 (3.89 MB)
Trainable params: 1018764 (3.89 MB)
Non-trainable params: 1088 (4.25 KB)
_________________________________________________________________
```

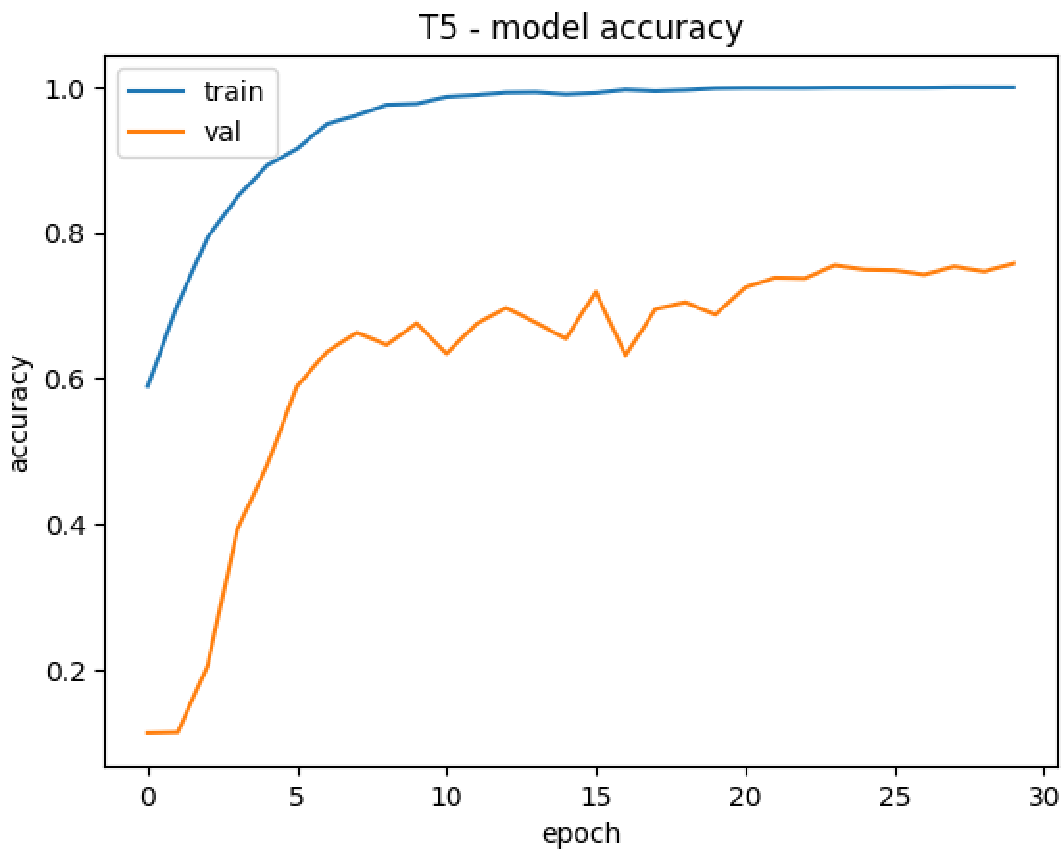
[48]:

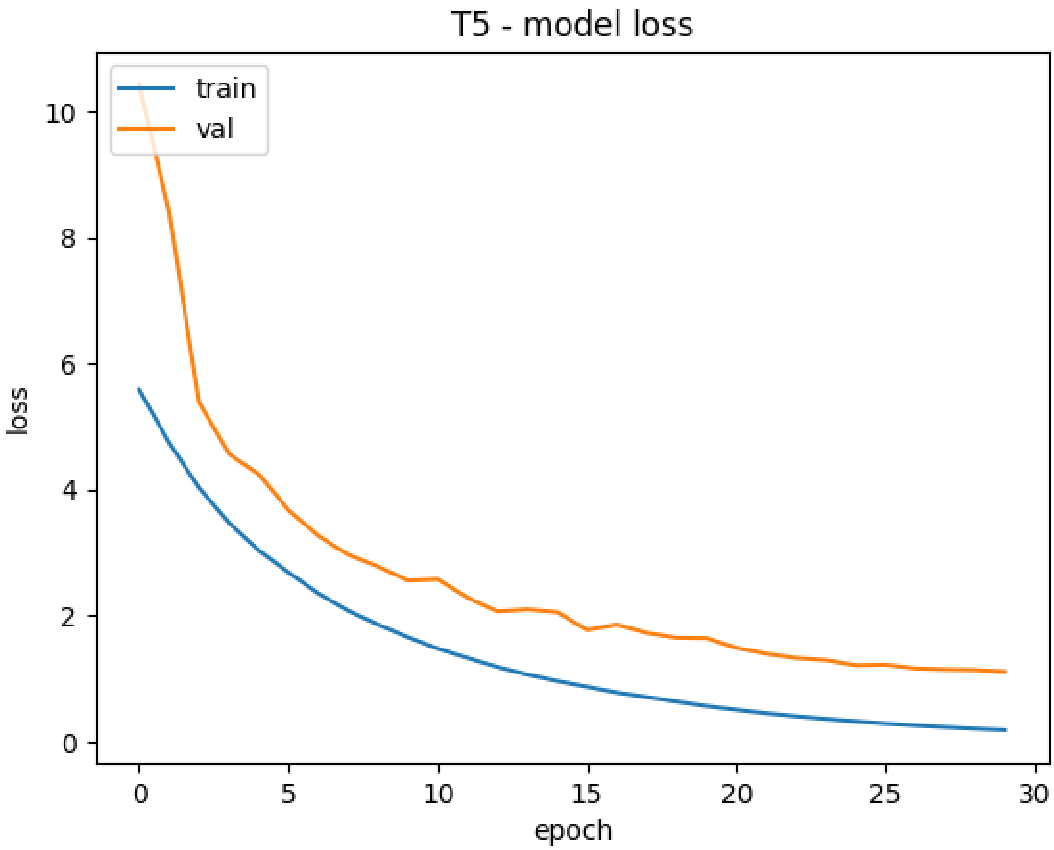
```
task5_history = task5_model.fit(x_train, y_train, batch_size=BATCH_SIZE,
 ↪epochs=30, validation_data=(x_val, y_val))

# summarize history for accuracy
plt.plot(task5_history.history['accuracy'])
plt.plot(task5_history.history['val_accuracy'])
plt.title('T5 - model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(task5_history.history['loss'])
plt.plot(task5_history.history['val_loss'])
plt.title('T5 - model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

```python
# Print accuracy
score = task5_model.evaluate(x_val, y_val, verbose=0)
print('T5 - validation accuracy:', score[1])

# Print confusion matrix
y_pred = task5_model.predict(x_val)
y_pred = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_val, axis=1)
class_labels = list(val_ds.class_names)
confusion_mtx = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx, xticklabels=class_labels, yticklabels=class_labels,␣
 ↪annot=True, fmt='d')
plt.xlabel('Prediction')
plt.ylabel('Label')
plt.title('T5 - Confusion Matrix')
plt.show()
```

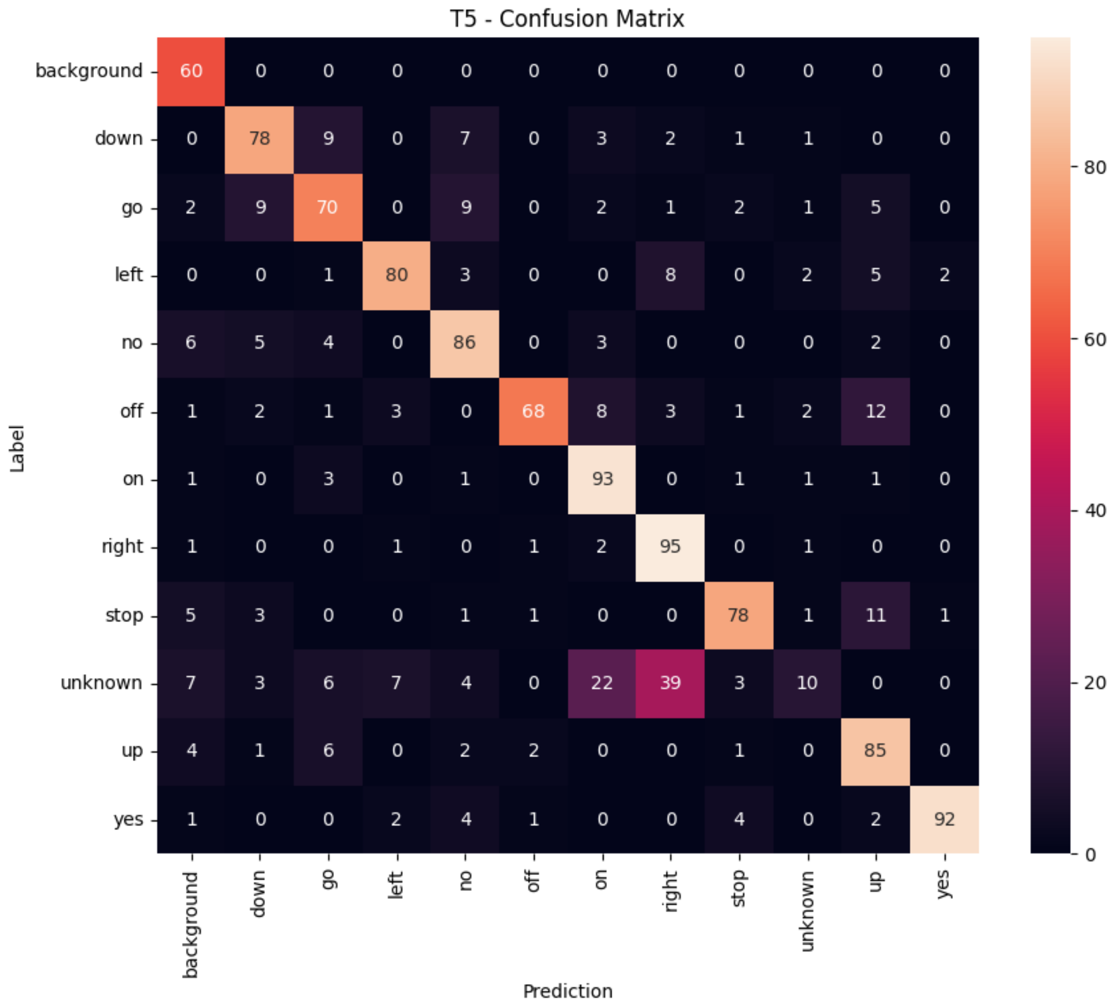
```
Epoch 1/30
16/16 [==============================] - 2s 136ms/step - loss: 5.5834 -
accuracy: 0.5892 - val_loss: 10.4304 - val_accuracy: 0.1126
Epoch 2/30
16/16 [==============================] - 2s 130ms/step - loss: 4.7417 -
accuracy: 0.7014 - val_loss: 8.3976 - val_accuracy: 0.1135
Epoch 3/30
16/16 [==============================] - 2s 134ms/step - loss: 4.0281 -
accuracy: 0.7934 - val_loss: 5.3865 - val_accuracy: 0.2049
Epoch 4/30
16/16 [==============================] - 2s 131ms/step - loss: 3.4725 -
accuracy: 0.8492 - val_loss: 4.5652 - val_accuracy: 0.3920
Epoch 5/30
16/16 [==============================] - 2s 133ms/step - loss: 3.0350 -
accuracy: 0.8927 - val_loss: 4.2441 - val_accuracy: 0.4809
Epoch 6/30
16/16 [==============================] - 2s 133ms/step - loss: 2.6832 -
accuracy: 0.9155 - val_loss: 3.6706 - val_accuracy: 0.5893
Epoch 7/30
16/16 [==============================] - 2s 133ms/step - loss: 2.3489 -
accuracy: 0.9496 - val_loss: 3.2630 - val_accuracy: 0.6367
Epoch 8/30
16/16 [==============================] - 2s 132ms/step - loss: 2.0744 -
accuracy: 0.9614 - val_loss: 2.9644 - val_accuracy: 0.6630
Epoch 9/30
16/16 [==============================] - 2s 134ms/step - loss: 1.8564 -
accuracy: 0.9758 - val_loss: 2.7778 - val_accuracy: 0.6461
Epoch 10/30
```

```
16/16 [==============================] - 2s 139ms/step - loss: 0.2857 -
accuracy: 0.9995 - val_loss: 1.2182 - val_accuracy: 0.7485
Epoch 27/30
16/16 [==============================] - 2s 136ms/step - loss: 0.2555 -
accuracy: 0.9995 - val_loss: 1.1598 - val_accuracy: 0.7426
Epoch 28/30
16/16 [==============================] - 2s 134ms/step - loss: 0.2281 -
accuracy: 1.0000 - val_loss: 1.1438 - val_accuracy: 0.7536
Epoch 29/30
16/16 [==============================] - 2s 134ms/step - loss: 0.2041 -
accuracy: 1.0000 - val_loss: 1.1352 - val_accuracy: 0.7468
Epoch 30/30
16/16 [==============================] - 2s 136ms/step - loss: 0.1827 -
accuracy: 1.0000 - val_loss: 1.1089 - val_accuracy: 0.7578
```






```
T5 - validation accuracy: 0.7578323483467102
37/37 [==============================] - 0s 8ms/step
```



As, we can observe: * The model is able to predict the 'yes' class with an accuracy of 0.8. * The model is able to predict the 'no' class with an accuracy of 0.6.

T5.2 Training the Advanced Model on the Personal Recordings: We now move the spectrograms to the training and validation directories and train the advanced model on the personal recordings.

[49]:
```
recrd_train_imgs = 12
recrd_val_imgs = 6

no_src_path = 'spectrogramImages/no'
yes_src_path = 'spectrogramImages/yes'

for i in range(1, recrd_train_imgs):
    shutil.copy(no_src_path + f'/{i}.png', f'speechImageData/TrainData/no/
 ↪image_train_{1661 + i}.png')
```

```
    shutil.copy(yes_src_path + f'/{i + 12}.png', f'speechImageData/TrainData/
↪yes/image_train_{1672 + i}.png')

for i in range(1, recrd_val_imgs):
    shutil.copy(no_src_path + f'/{i + 12}.png', f'speechImageData/ValData/no/
↪image_val_{1111 + i}.png')
    shutil.copy(yes_src_path + f'/{i + 24}.png', f'speechImageData/ValData/yes/
↪image_val_{1116 + i}.png')
```

We create new instance of x_train, y_train, x_val, y_val and train the advanced model on the personal recordings.

[50]:
```
# Load the data

train_ds = tf.keras.utils.image_dataset_from_directory(
    directory='speechImageData/TrainData',
    labels='inferred',
    color_mode="grayscale",
    label_mode='categorical',
    batch_size=BATCH_SIZE,
    image_size=IMG_SIZE
)

val_ds = tf.keras.utils.image_dataset_from_directory(
    directory='speechImageData/ValData',
    labels='inferred',
    color_mode="grayscale",
    label_mode='categorical',
    batch_size=BATCH_SIZE,
    image_size=IMG_SIZE
)

# Extract the training input images and output class labels
x_train = []
y_train = []
for images, labels in train_ds.take(-1):
    x_train.append(images.numpy())
    y_train.append(labels.numpy())

x_train_recrd = np.concatenate(x_train, axis=0)
y_train_recrd = np.concatenate(y_train, axis=0)

print(y_train_recrd)

# Extract the validation input images and output class labels
x_val = []
y_val = []
```

```
for images, labels in val_ds.take(-1):
    x_val.append(images.numpy())
    y_val.append(labels.numpy())

x_val_recrd = np.concatenate(x_val, axis=0)
y_val_recrd = np.concatenate(y_val, axis=0)

print(y_val_recrd)
```

```
Found 2023 files belonging to 12 classes.
Found 1181 files belonging to 12 classes.
[[0. 0. 0. … 1. 0. 0.]
 [1. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 1.]
 …
 [0. 0. 0. … 1. 0. 0.]
 [1. 0. 0. … 0. 0. 0.]
 [1. 0. 0. … 0. 0. 0.]]
[[0. 0. 0. … 0. 0. 0.]
 [1. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]
 …
 [0. 0. 0. … 0. 0. 0.]
 [0. 1. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]]
```

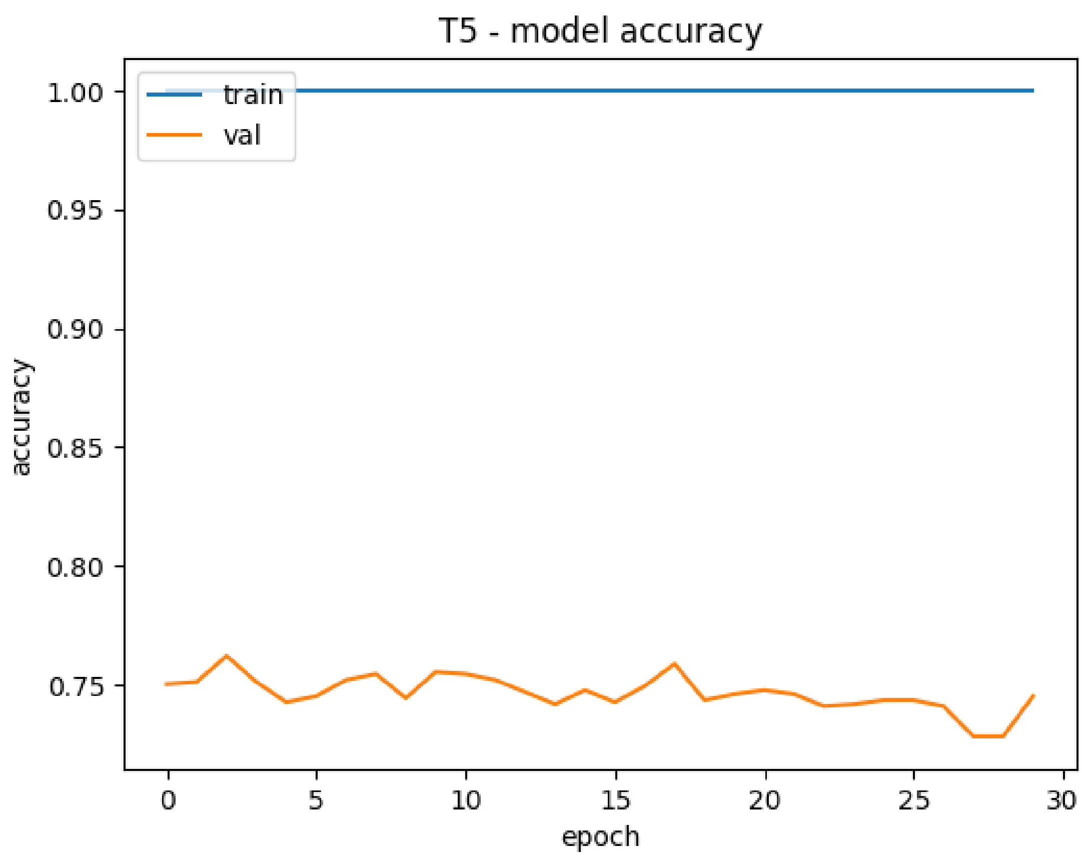
We now plot the training history and confusion matrix of the advanced model on the personal recordings.

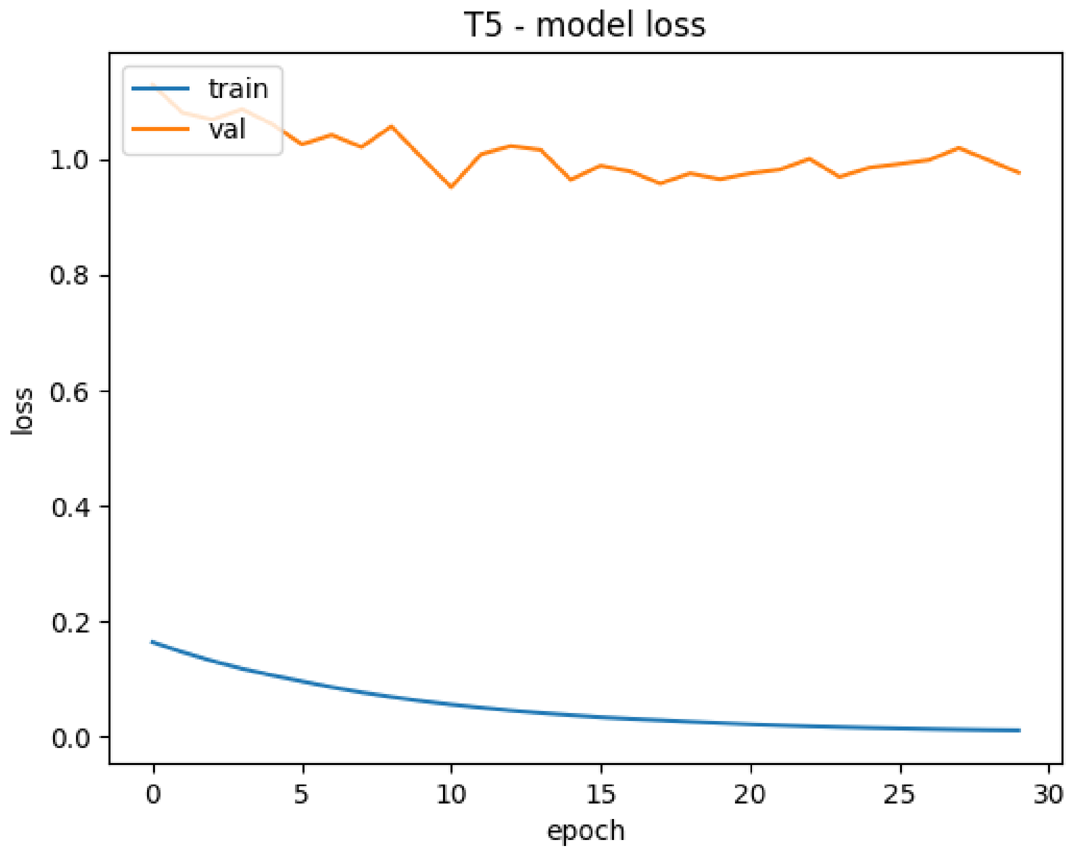
[51]:
```
task5_history = task5_model.fit(x_train_recrd, y_train_recrd,␣
 ↪batch_size=BATCH_SIZE, epochs=30, validation_data=(x_val_recrd, y_val_recrd))

# summarize history for accuracy
plt.plot(task5_history.history['accuracy'])
plt.plot(task5_history.history['val_accuracy'])
plt.title('T5 - model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(task5_history.history['loss'])
plt.plot(task5_history.history['val_loss'])
plt.title('T5 - model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
```

```
plt.show()

# Print accuracy
score = task5_model.evaluate(x_val_recrd, y_val_recrd, verbose=0)
print('T5 - validation accuracy:', score[1])

# Print confusion matrix
y_pred = task5_model.predict(x_val_recrd)
y_pred = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_val_recrd, axis=1)
class_labels = list(val_ds.class_names)
confusion_mtx = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx, xticklabels=class_labels, yticklabels=class_labels,
    annot=True, fmt='d')
plt.xlabel('Prediction')
plt.ylabel('Label')
plt.title('T1 - Confusion Matrix')
plt.show()
```

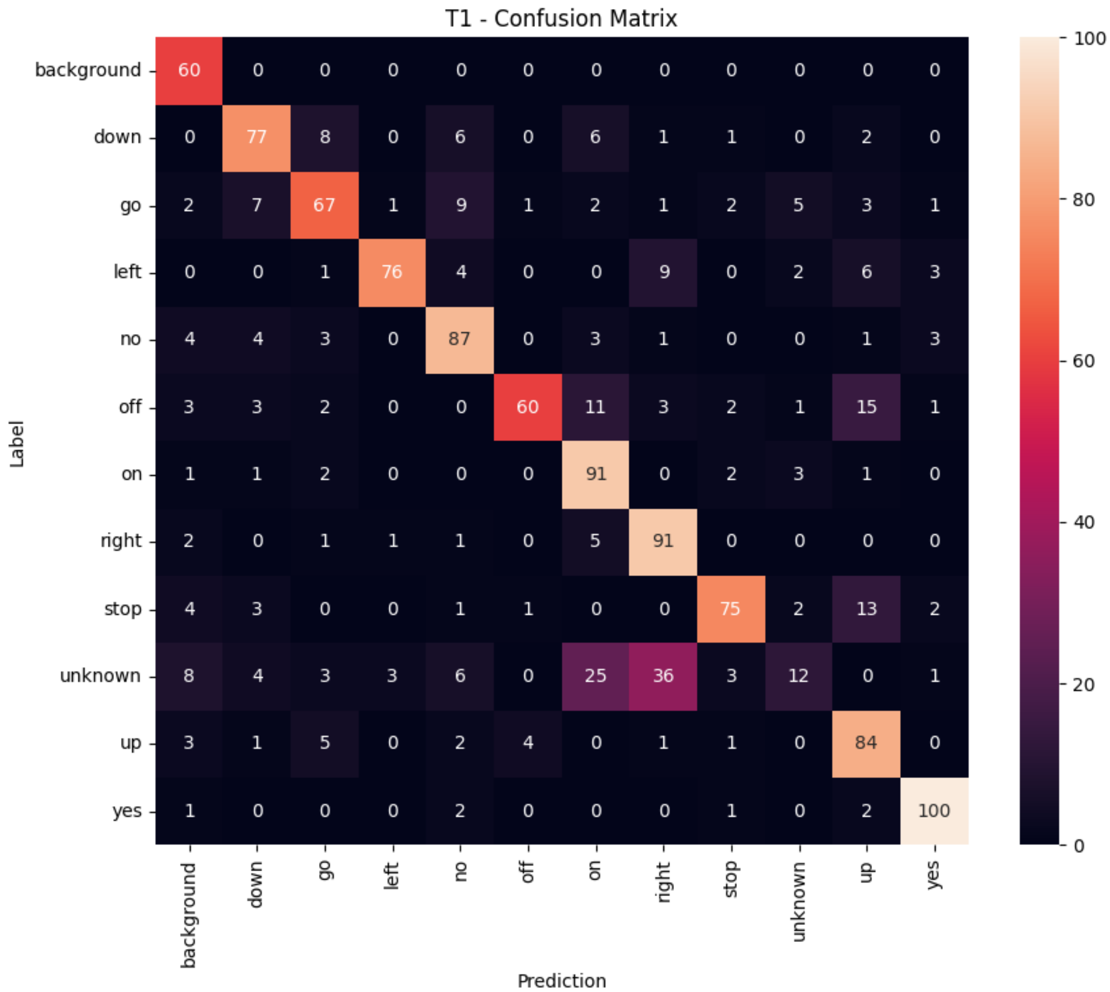
```
Epoch 1/30
16/16 [==============================] - 2s 140ms/step - loss: 0.1636 -
accuracy: 1.0000 - val_loss: 1.1288 - val_accuracy: 0.7502
Epoch 2/30
16/16 [==============================] - 2s 135ms/step - loss: 0.1466 -
accuracy: 1.0000 - val_loss: 1.0800 - val_accuracy: 0.7511
Epoch 3/30
16/16 [==============================] - 2s 134ms/step - loss: 0.1313 -
accuracy: 1.0000 - val_loss: 1.0681 - val_accuracy: 0.7621
Epoch 4/30
16/16 [==============================] - 2s 134ms/step - loss: 0.1177 -
accuracy: 1.0000 - val_loss: 1.0858 - val_accuracy: 0.7511
Epoch 5/30
16/16 [==============================] - 2s 131ms/step - loss: 0.1064 -
accuracy: 1.0000 - val_loss: 1.0604 - val_accuracy: 0.7426
Epoch 6/30
16/16 [==============================] - 2s 134ms/step - loss: 0.0960 -
accuracy: 1.0000 - val_loss: 1.0253 - val_accuracy: 0.7451
Epoch 7/30
16/16 [==============================] - 2s 132ms/step - loss: 0.0859 -
accuracy: 1.0000 - val_loss: 1.0414 - val_accuracy: 0.7519
Epoch 8/30
16/16 [==============================] - 2s 134ms/step - loss: 0.0769 -
accuracy: 1.0000 - val_loss: 1.0206 - val_accuracy: 0.7544
Epoch 9/30
16/16 [==============================] - 2s 136ms/step - loss: 0.0690 -
accuracy: 1.0000 - val_loss: 1.0563 - val_accuracy: 0.7443
```

```
Epoch 26/30
16/16 [==============================] - 2s 136ms/step - loss: 0.0144 -
accuracy: 1.0000 - val_loss: 0.9912 - val_accuracy: 0.7434
Epoch 27/30
16/16 [==============================] - 2s 135ms/step - loss: 0.0134 -
accuracy: 1.0000 - val_loss: 0.9981 - val_accuracy: 0.7409
Epoch 28/30
16/16 [==============================] - 2s 137ms/step - loss: 0.0124 -
accuracy: 1.0000 - val_loss: 1.0189 - val_accuracy: 0.7282
Epoch 29/30
16/16 [==============================] - 2s 138ms/step - loss: 0.0117 -
accuracy: 1.0000 - val_loss: 0.9976 - val_accuracy: 0.7282
Epoch 30/30
16/16 [==============================] - 2s 137ms/step - loss: 0.0112 -
accuracy: 1.0000 - val_loss: 0.9763 - val_accuracy: 0.7451
```






```
T5 - validation accuracy: 0.745131254196167
37/37 [==============================] - 0s 8ms/step
```



As observed, there is an increased match in both the classes, ‘yes’ and ‘no’ with the model’s predictions. There is no dramatic drop of accuracy in these classes, and hence the data augmentation has helped in improving the model’s performance.

T5.3 Transfer Learning using the Advanced Model We now use the popular model mobilenetv2 to perform transfer learning on the advanced model. Since the model requires the image size to be either 96x96, 128x128, 224x224, we resize the images to 96x96, since our original image size is 98x50.

We also reimport our training and validation data as three channeled rgb images instead of grayscale.

[55]:
```
# Load the data

train_ds = tf.keras.utils.image_dataset_from_directory(
    directory='speechImageData/TrainData',
    labels='inferred',
```

```
    color_mode="rgb",
    label_mode='categorical',
    batch_size=BATCH_SIZE,
    image_size=IMG_SIZE
)

val_ds = tf.keras.utils.image_dataset_from_directory(
    directory='speechImageData/ValData',
    labels='inferred',
    color_mode="rgb",
    label_mode='categorical',
    batch_size=BATCH_SIZE,
    image_size=IMG_SIZE
)

# Extract the training input images and output class labels
x_train = []
y_train = []
for images, labels in train_ds.take(-1):
    x_train.append(images.numpy())
    y_train.append(labels.numpy())

x_train_recrd = np.concatenate(x_train, axis=0)
y_train_recrd = np.concatenate(y_train, axis=0)

print(y_train_recrd)

# Extract the validation input images and output class labels
x_val = []
y_val = []
for images, labels in val_ds.take(-1):
    x_val.append(images.numpy())
    y_val.append(labels.numpy())

x_val_recrd = np.concatenate(x_val, axis=0)
y_val_recrd = np.concatenate(y_val, axis=0)

print(y_val_recrd)
```

```
Found 2023 files belonging to 12 classes.
Found 1181 files belonging to 12 classes.
[[0. 0. 0. … 0. 1. 0.]
 [0. 0. 1. … 0. 0. 0.]
 [0. 0. 0. … 1. 0. 0.]
 …
 [0. 0. 0. … 0. 0. 1.]
 [1. 0. 0. … 0. 0. 0.]
```

```
 [1. 0. 0. … 0. 0. 0.]]
[[0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 1. 0. 0.]
 [0. 0. 0. … 0. 0. 1.]
 …
 [0. 0. 0. … 0. 1. 0.]
 [0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]]
```

[56]:
```
# mobilenet image size
MOBILENET_IMG_SIZE = (96, 96)

# resize the images
x_train_recrd = tf.image.resize(x_train_recrd, (MOBILENET_IMG_SIZE[0],␣
↪MOBILENET_IMG_SIZE[1]))
x_val_recrd = tf.image.resize(x_val_recrd, (MOBILENET_IMG_SIZE[0],␣
↪MOBILENET_IMG_SIZE[1]))

base_model = MobileNetV2(input_shape=(MOBILENET_IMG_SIZE[0],␣
↪MOBILENET_IMG_SIZE[1], 3), include_top=False, weights='imagenet')

x_train_recrd = preprocess_input(x_train_recrd)
x_val_recrd = preprocess_input(x_val_recrd)

# extract features
train_features = base_model.predict(x_train_recrd)
val_features = base_model.predict(x_val_recrd)
```

```
64/64 [==============================] - 2s 12ms/step
37/37 [==============================] - 0s 11ms/step
```

We now define the base model for training. We just append the fully connected layer to the base model and output layer to the model.

[62]:
```
mobilenet_model = Sequential()
mobilenet_model.add(Flatten(input_shape=train_features.shape[1:]))
mobilenet_model.add(Dense(1024, kernel_regularizer=regularizers.l2(0.1),␣
↪activation='relu'))
mobilenet_model.add(Dropout(0.2))
mobilenet_model.add(Dense(NUM_CLASSES, activation='softmax'))
mobilenet_model.compile(optimizer=optimizers.Adam(learning_rate=0.0001),␣
↪loss='categorical_crossentropy', metrics=['accuracy'])
mobilenet_model.summary()
```

```
Model: "sequential_40"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 flatten_40 (Flatten)        (None, 11520)             0

 dense_80 (Dense)            (None, 1024)              11797504

 dropout_40 (Dropout)        (None, 1024)              0

 dense_81 (Dense)            (None, 12)                12300

=================================================================
Total params: 11809804 (45.05 MB)
Trainable params: 11809804 (45.05 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

T5.A - Model Training Now, we proceed to train the model with the training and validation features extracted by the base model.

[63]:
```
mobilenet_model.history = mobilenet_model.fit(train_features, y_train_recrd,␣
↪batch_size=BATCH_SIZE, epochs=100, validation_data=(val_features,␣
↪y_val_recrd))
```

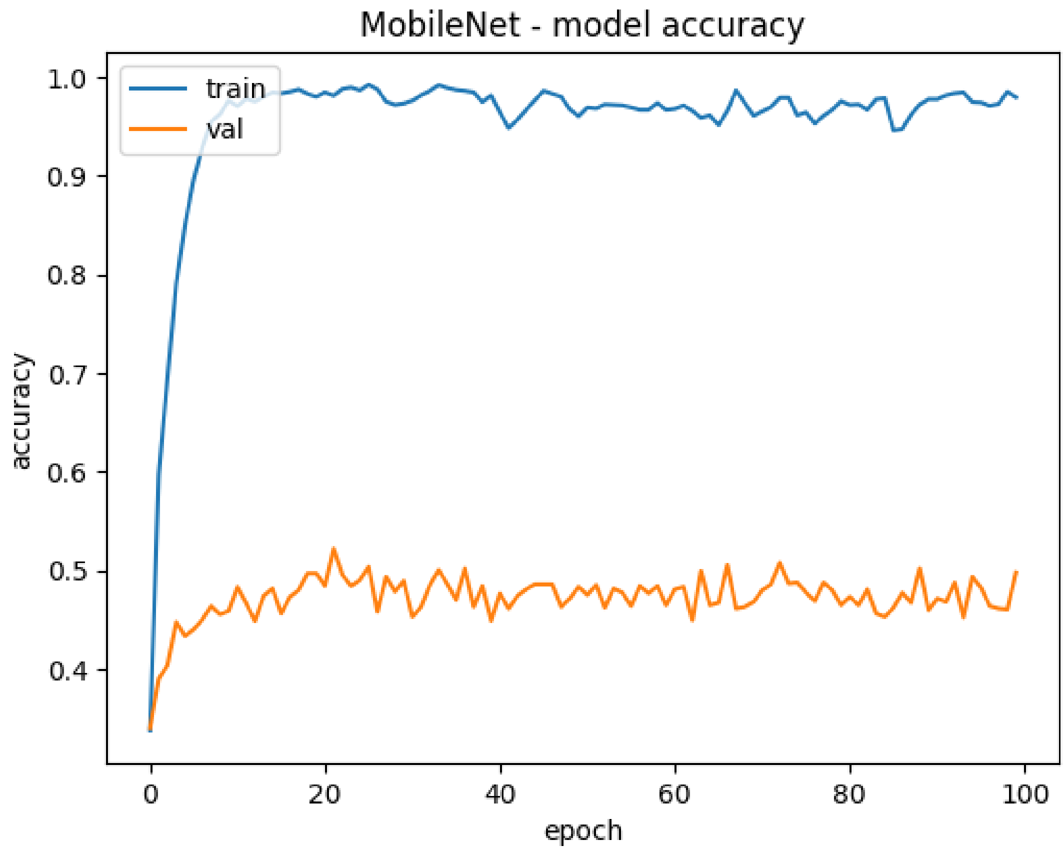
```
Epoch 1/100
16/16 [==============================] - 1s 33ms/step - loss: 176.4016 -
accuracy: 0.3381 - val_loss: 160.5809 - val_accuracy: 0.3404
Epoch 2/100
16/16 [==============================] - 0s 15ms/step - loss: 147.2709 -
accuracy: 0.5971 - val_loss: 134.0095 - val_accuracy: 0.3903
Epoch 3/100
16/16 [==============================] - 0s 15ms/step - loss: 122.2296 -
accuracy: 0.6945 - val_loss: 111.3102 - val_accuracy: 0.4039
Epoch 4/100
16/16 [==============================] - 0s 14ms/step - loss: 100.9884 -
accuracy: 0.7904 - val_loss: 92.1280 - val_accuracy: 0.4479
Epoch 5/100
16/16 [==============================] - 0s 15ms/step - loss: 83.2079 -
accuracy: 0.8492 - val_loss: 76.0616 - val_accuracy: 0.4335
Epoch 6/100
16/16 [==============================] - 0s 13ms/step - loss: 68.3979 -
accuracy: 0.8962 - val_loss: 62.7798 - val_accuracy: 0.4403
Epoch 7/100
16/16 [==============================] - 0s 15ms/step - loss: 56.1458 -
accuracy: 0.9288 - val_loss: 51.8201 - val_accuracy: 0.4505
Epoch 8/100
16/16 [==============================] - 0s 14ms/step - loss: 46.0271 -
accuracy: 0.9545 - val_loss: 42.7247 - val_accuracy: 0.4640
Epoch 9/100
16/16 [==============================] - 0s 13ms/step - loss: 37.7221 -
accuracy: 0.9619 - val_loss: 35.2867 - val_accuracy: 0.4555
```

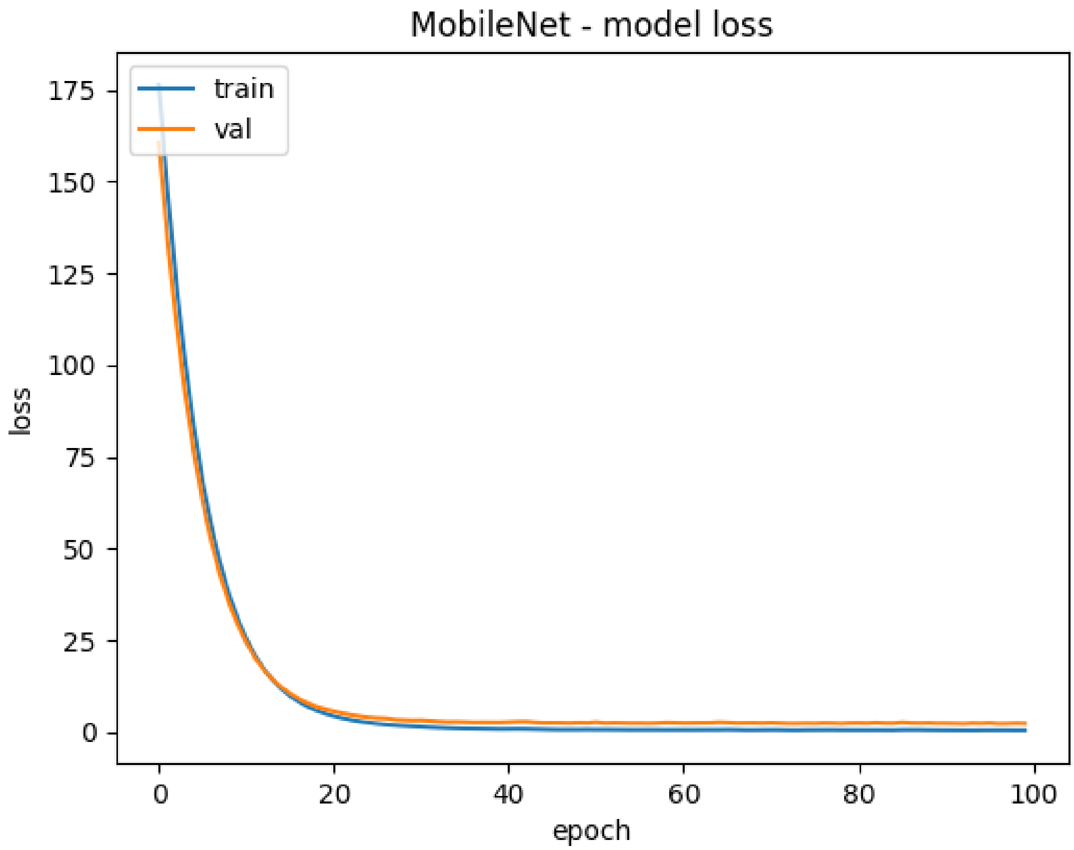
```
Epoch 90/100
16/16 [==============================] - 0s 14ms/step - loss: 0.4615 - accuracy:
0.9773 - val_loss: 2.3257 - val_accuracy: 0.4598
Epoch 91/100
16/16 [==============================] - 0s 13ms/step - loss: 0.4473 - accuracy:
0.9773 - val_loss: 2.3323 - val_accuracy: 0.4716
Epoch 92/100
16/16 [==============================] - 0s 14ms/step - loss: 0.4390 - accuracy:
0.9812 - val_loss: 2.2895 - val_accuracy: 0.4682
Epoch 93/100
16/16 [==============================] - 0s 15ms/step - loss: 0.4203 - accuracy:
0.9832 - val_loss: 2.2334 - val_accuracy: 0.4877
Epoch 94/100
16/16 [==============================] - 0s 13ms/step - loss: 0.4111 - accuracy:
0.9842 - val_loss: 2.3693 - val_accuracy: 0.4522
Epoch 95/100
16/16 [==============================] - 0s 15ms/step - loss: 0.4371 - accuracy:
0.9743 - val_loss: 2.3085 - val_accuracy: 0.4936
Epoch 96/100
16/16 [==============================] - 0s 15ms/step - loss: 0.4407 - accuracy:
0.9738 - val_loss: 2.4025 - val_accuracy: 0.4826
Epoch 97/100
16/16 [==============================] - 0s 13ms/step - loss: 0.4512 - accuracy:
0.9703 - val_loss: 2.2379 - val_accuracy: 0.4640
Epoch 98/100
16/16 [==============================] - 0s 13ms/step - loss: 0.4471 - accuracy:
0.9723 - val_loss: 2.2590 - val_accuracy: 0.4615
Epoch 99/100
16/16 [==============================] - 0s 13ms/step - loss: 0.4262 - accuracy:
0.9847 - val_loss: 2.3434 - val_accuracy: 0.4606
Epoch 100/100
16/16 [==============================] - 0s 14ms/step - loss: 0.4310 - accuracy:
0.9792 - val_loss: 2.2964 - val_accuracy: 0.4979
```

T5.B - Plot Training History and Confusion Matrix Here, we plot the training history and confusion matrix of the advanced model with transfer learning.

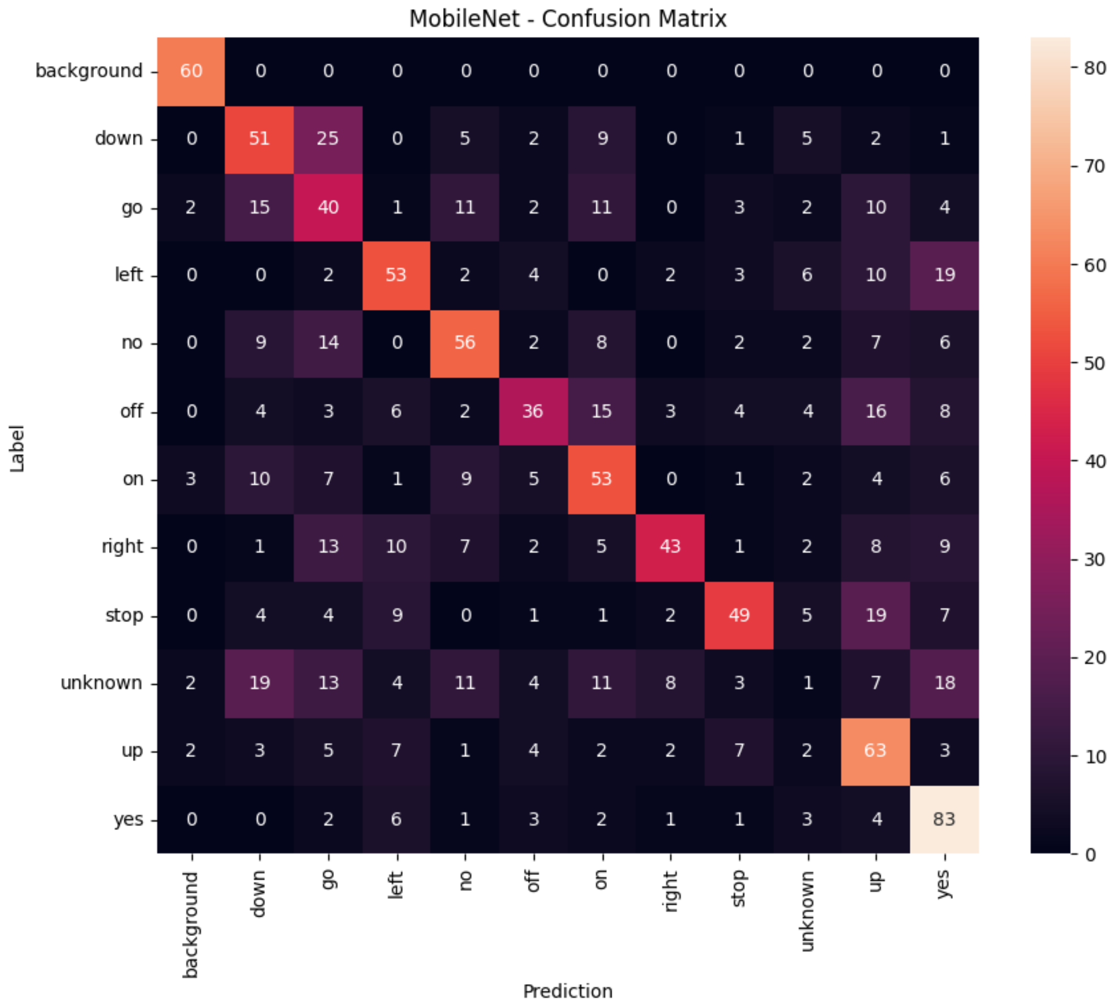
[64]:
```
# summarize history for accuracy
plt.plot(mobilenet_model.history.history['accuracy'])
plt.plot(mobilenet_model.history.history['val_accuracy'])
plt.title('MobileNet - model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# summarize history for loss
```

```
plt.plot(mobilenet_model.history.history['loss'])
plt.plot(mobilenet_model.history.history['val_loss'])
plt.title('MobileNet - model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# Print accuracy
score = mobilenet_model.evaluate(val_features, y_val_recrd, verbose=0)
print('MobileNet - validation accuracy:', score[1])

# Print confusion matrix
y_pred = mobilenet_model.predict(val_features)
y_pred = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_val_recrd, axis=1)
class_labels = list(val_ds.class_names)
confusion_mtx = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx, xticklabels=class_labels, yticklabels=class_labels,␣
 ↪annot=True, fmt='d')
plt.xlabel('Prediction')
plt.ylabel('Label')
plt.title('MobileNet - Confusion Matrix')
plt.show()
```


MobileNet - model accuracy
train
val
1.0
0.9
0.8
0.7
0.6
0.5
0.4
accuracy
0
20
40
60
80
100
epoch




```
MobileNet - validation accuracy: 0.497883141040802
37/37 [==============================] - 0s 3ms/step
```



T5.C - Conclusion The transfer learning model has a poor accuracy of about 0.5. This is because the model is not able to learn the features of the personal recordings well due to the resizing constraint of the model. The resizing jump from 98x50 to 96x96 has caused the model to lose the features of the spectrograms, and hence the model is not able to predict the classes well.

Future steps to improve the model would be taking inspiration of the model's architecture in the transfer learning model and training a new model with the same architecture as the base model without the resizing constraint.

1.1 Overall Conclusion

The advanced model has an accuracy of about 0.84, which is better than the baseline model's accuracy of 0.6. The advanced model is able to predict the classes of the personal recordings with an accuracy of 0.8 for the 'yes' class and 0.6 for the 'no' class.

Additionally, the transfer learning model has a poor accuracy of about 0.5. This is because the model is not able to learn the features of the personal recordings well due to the resizing constraint of the model. All of the above code, helper scripts and data are available in my personal GitHub repository.