DL Assignment3

Akshara B

211AI012

Dataset:

I have used an audio dataset for the Multi-class classification task. The dataset consists of wav-format audio files which are of length 3-seconds. They contain the siren sound of Emergency Vehicles - Ambulance and Firetruck. A third category named Traffic also exists where it contains 3-second .wav format audio files of plain traffic sound. Each category contains 200 sound files. Total 600 files.

Link: https://www.kaggle.com/datasets/vishnu0399/emergency-vehicle-siren-sounds/data

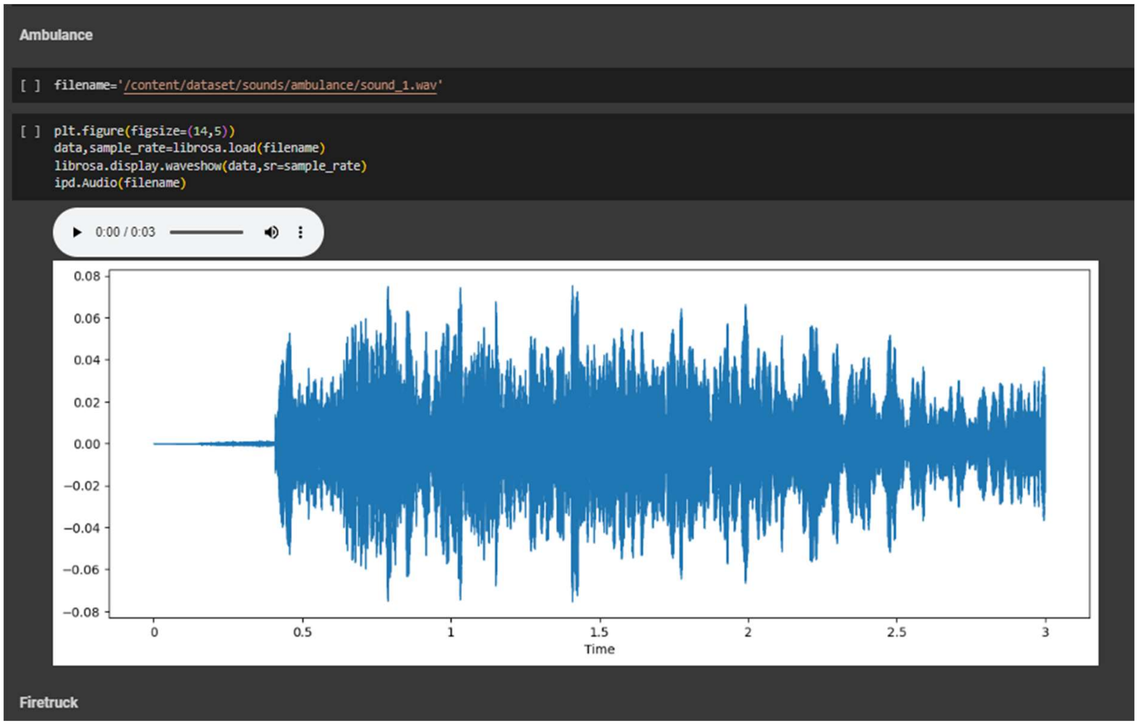
Google colab notebook:

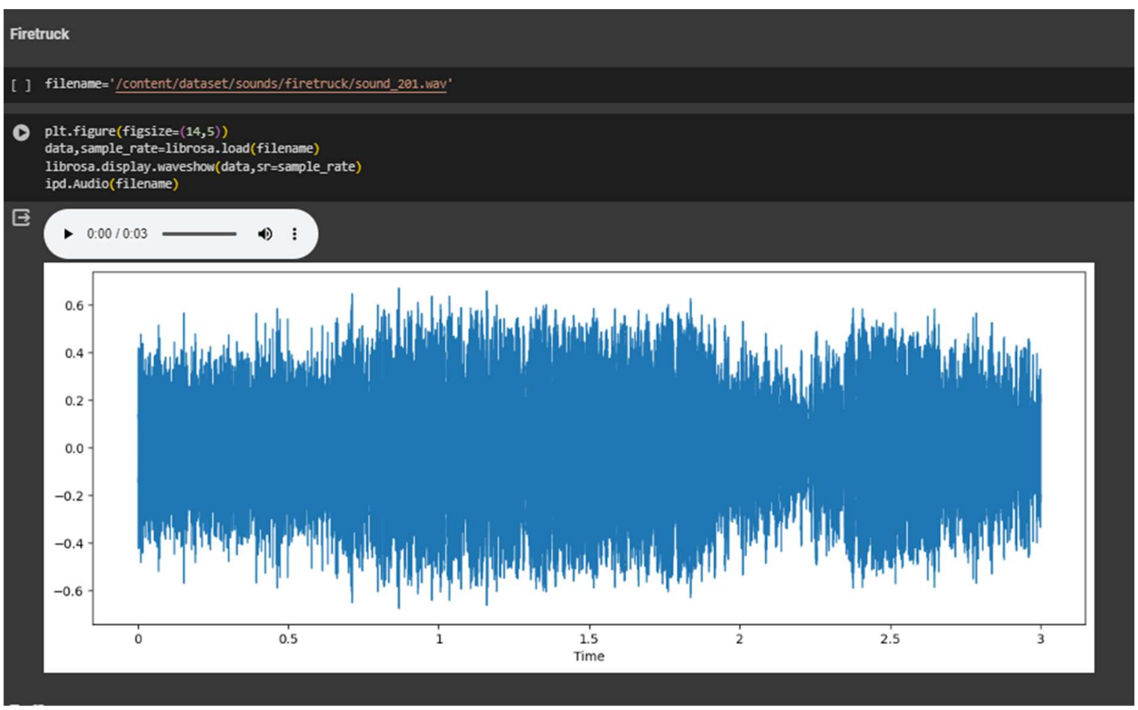
code can be found here –

https://colab.research.google.com/drive/1cHC1WOxZPR4rUVZffjlNT-i54Asv3N7P?usp=sharing

Task 1 and 3

First we display the audio as a wave using librosa.display.waveshow and Ipython.display



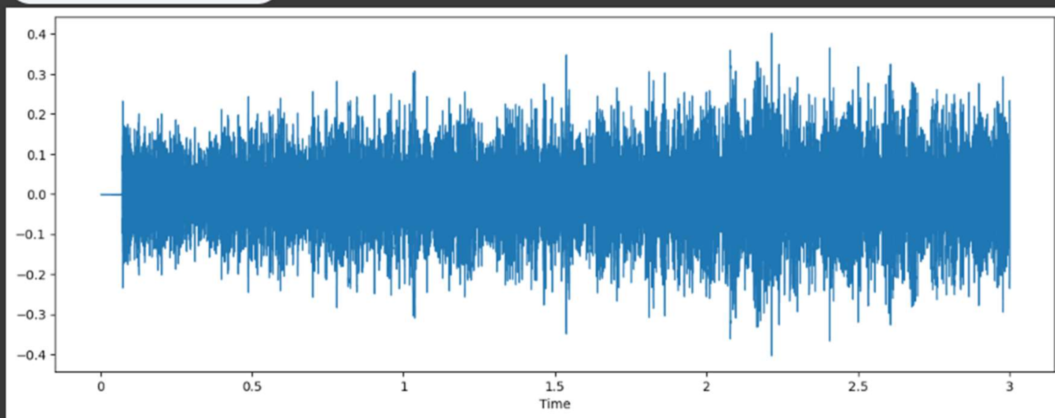


Traffic

```
filename='/content/dataset/sounds/traffic/sound_401.wav'
```

```
plt.figure(figsize=(14,5))
data,sample_rate=librosa.load(filename)
librosa.display.waveshow(data,sr=sample_rate)
ipd.Audio(filename)
```

0:00 / 0:03



Converting audio to spectagram images

Now we create a function to create spectrograms from audio files.

```
import os
import librosa
import numpy as np
import matplotlib.pyplot as plt

# Define paths
data_folder = '/content/dataset/sounds'
output_folder = '/content/spectrogram_images'

# Function to create spectrogram
def create_spectrogram(file_path, output_path):
    # Load audio file
    audio, sr = librosa.load(file_path, sr=None)

    # Compute spectrogram
    S = librosa.feature.melspectrogram(y=audio, sr=sr)
    S_dB = librosa.power_to_db(S, ref=np.max)

    # Plot and save spectrogram
    plt.figure(figsize=(10, 4))
    librosa.display.specshow(S_dB, sr=sr, x_axis='time', y_axis='mel')
    plt.colorbar(format='%+2.0f dB')
    plt.title('Mel-frequency spectrogram')
    plt.savefig(output_path)
    plt.close()

# Iterate through all files in the data folder
file_count = 0
total_files = sum([len(files) for _, _, files in os.walk(data_folder)])
for root, dirs, files in os.walk(data_folder):
    for file in files:
        if file.endswith('.wav'):
            file_count += 1
            print(f"Processing file {file_count}/{total_files}...")

            # Extract class from filename
            class_name = os.path.basename(root)

            # Create output folder if not exists
            class_folder = os.path.join(output_folder, class_name)
            os.makedirs(class_folder, exist_ok=True)

            # Generate spectrogram and save
            file_path = os.path.join(root, file)
            output_path = os.path.join(class_folder, file.replace('.wav', '.png'))
            create_spectrogram(file_path, output_path)

print("Spectrogram generation completed.")
```

Using this function we iterate through all 600 audio files and convert them to spectrogram images and make new dataset 'spectrogram_images'.It utilizes the `librosa` library for audio processing and `matplotlib` for visualization.

Here,

- `data_folder`: Path to the directory containing the audio files.

- `output_folder`: Path to the directory where spectrogram images will be saved.

Create Spectrogram Function: This function `create_spectrogram` takes a file path as input, loads the audio file using `librosa`, computes the mel-spectrogram, converts it to decibels (dB), plots the spectrogram using `matplotlib`, and saves the plot as an image file (`.png`) in the specified output path.

We iterate through all files in the `data_folder` using `os.walk()`. For each `.wav` file encountered, it:

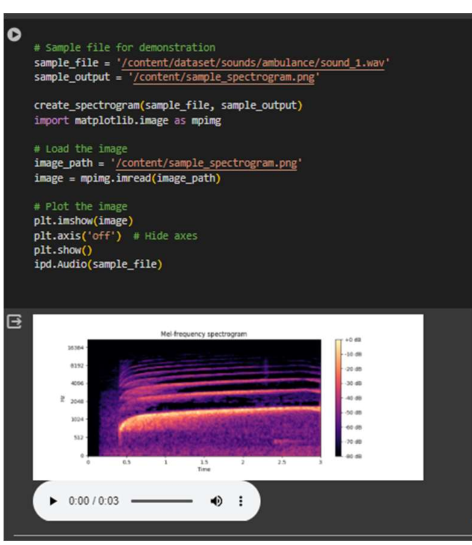
- Increments the `file_count`.

- Extracts the class (category) name from the directory structure.

- Creates a subfolder in the `output_folder` corresponding to the class name if it doesn't exist.

- Generates a spectrogram for the audio file and saves it in the appropriate class subfolder.

The generated spectrogram images can then be used as input data for CNN.



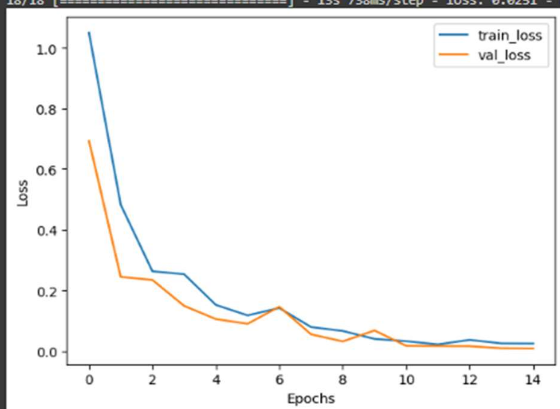
This is a sample audio file and its spectrogram

Building CNN models

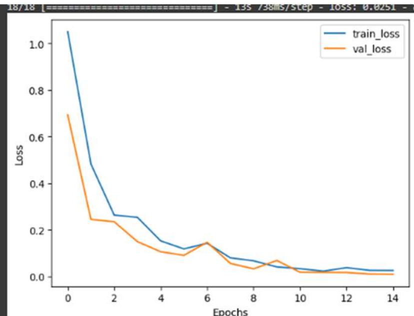
1. Model1 has 4 conv2d layers with filters 32,64,128,256 respectively. Kernel size is 3x3 and stride=1. Maaxpooling size=2x2. This is a very popular and basic CNN architecture. This model gave an accuracy of 98% on train set and 99% on test and validation set. Categorical Crossentropy loss was used and adam was used as optimizer. The model was trained for 15 epochs with batch size=32 and 18 timesteps per epoch. We see that here the model converges at 10 epochs. Accuracy for classes ambulace and traffic are 100% and class firetruck has accuracy 99%. This model takes significant amount of time to run using cpu and gives good results in the end.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 124, 124, 32)      2432

 max_pooling2d_4 (MaxPoolin  (None, 62, 62, 32)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 58, 58, 64)        51264

 max_pooling2d_5 (MaxPoolin  (None, 29, 29, 64)        0
 g2D)

 conv2d_6 (Conv2D)           (None, 25, 25, 128)       204928

 max_pooling2d_6 (MaxPoolin  (None, 12, 12, 128)       0
 g2D)

 conv2d_7 (Conv2D)           (None, 8, 8, 256)         819456

 max_pooling2d_7 (MaxPoolin  (None, 4, 4, 256)         0
 g2D)

 flatten_1 (Flatten)         (None, 4096)              0

 dense_2 (Dense)             (None, 512)               2097664

 dropout_1 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 3)                 1539

=================================================================
Total params: 3177283 (12.12 MB)
Trainable params: 3177283 (12.12 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/15
```

```
18/18 [==============================] - 24s 946ms/step - loss: 1.0506 - accuracy: 0.5106 - val_loss: 0.6929 - val_accuracy: 0.6806
Epoch 2/15
18/18 [==============================] - 16s 925ms/step - loss: 0.4821 - accuracy: 0.7905 - val_loss: 0.2444 - val_accuracy: 0.9045
Epoch 3/15
18/18 [==============================] - 14s 784ms/step - loss: 0.2628 - accuracy: 0.8891 - val_loss: 0.2343 - val_accuracy: 0.9045
Epoch 4/15
18/18 [==============================] - 16s 947ms/step - loss: 0.2532 - accuracy: 0.8979 - val_loss: 0.1489 - val_accuracy: 0.9462
Epoch 5/15
18/18 [==============================] - 13s 741ms/step - loss: 0.1522 - accuracy: 0.9366 - val_loss: 0.1056 - val_accuracy: 0.9514
Epoch 6/15
18/18 [==============================] - 16s 940ms/step - loss: 0.1176 - accuracy: 0.9489 - val_loss: 0.0900 - val_accuracy: 0.9618
Epoch 7/15
18/18 [==============================] - 13s 746ms/step - loss: 0.1420 - accuracy: 0.9542 - val_loss: 0.1462 - val_accuracy: 0.9392
Epoch 8/15
18/18 [==============================] - 13s 737ms/step - loss: 0.0793 - accuracy: 0.9648 - val_loss: 0.0554 - val_accuracy: 0.9809
Epoch 9/15
18/18 [==============================] - 13s 737ms/step - loss: 0.0665 - accuracy: 0.9736 - val_loss: 0.0321 - val_accuracy: 0.9878
Epoch 10/15
18/18 [==============================] - 14s 782ms/step - loss: 0.0403 - accuracy: 0.9859 - val_loss: 0.0680 - val_accuracy: 0.9688
Epoch 11/15
18/18 [==============================] - 13s 757ms/step - loss: 0.0327 - accuracy: 0.9894 - val_loss: 0.0175 - val_accuracy: 0.9913
Epoch 12/15
18/18 [==============================] - 13s 742ms/step - loss: 0.0222 - accuracy: 0.9912 - val_loss: 0.0168 - val_accuracy: 0.9878
Epoch 13/15
18/18 [==============================] - 16s 929ms/step - loss: 0.0371 - accuracy: 0.9894 - val_loss: 0.0164 - val_accuracy: 0.9931
Epoch 14/15
18/18 [==============================] - 16s 928ms/step - loss: 0.0255 - accuracy: 0.9930 - val_loss: 0.0096 - val_accuracy: 0.9983
Epoch 15/15
18/18 [==============================] - 13s 738ms/step - loss: 0.0251 - accuracy: 0.9894 - val_loss: 0.0087 - val_accuracy: 0.9983
```




```
19/19 - 7s - loss: 0.0088 - accuracy: 0.9983 - 7s/epoch - 362ms/step
```




```
19/19 - 7s - loss: 0.0088 - accuracy: 0.9983 - 7s/epoch - 362ms/step
19/19 [==============================] - 7s 365ms/step
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       200
           1       1.00      0.99      1.00       200
           2       1.00      1.00      1.00       200

    accuracy                           1.00       600
   macro avg       1.00      1.00      1.00       600
weighted avg       1.00      1.00      1.00       600

Test accuracy: 0.9983333349227905
```

```
           0       0.97      0.99      0.98       200
           1       0.99      0.97      0.98       200
           2       1.00      1.00      1.00       200

    accuracy                           0.99       600
   macro avg       0.99      0.99      0.99       600
weighted avg       0.99      0.99      0.99       600

Test accuracy: 0.9866666793823242
```

```python
from sklearn.metrics import confusion_matrix
# Calculate confusion matrix
conf_matrix = confusion_matrix(validation_generator.classes, y_pred_classes)

# Print confusion matrix
print("\nConfusion Matrix:")
print(conf_matrix)
```

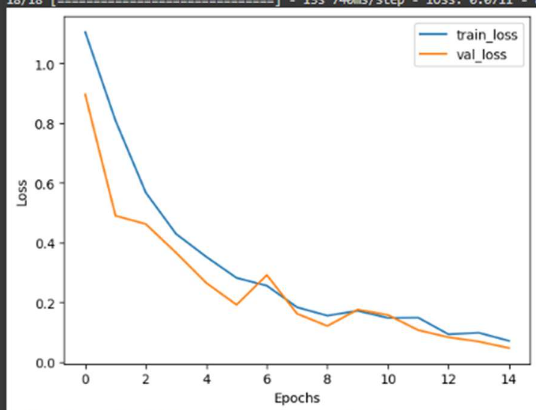
```
Confusion Matrix:
[[198   2   0]
 [  6 194   0]
 [  0   0 200]]
```

2. Model2 has 4 conv2d layers with filters 32,64,128,256 respectively. Kernel size is 5x5 and stride=1. Maaxpooling size=2x2. Here we increased the kernel size This model gave an accuracy of 97% on train set and 98% on test and validation set. Categorical Crossentropy loss was used and adam was used as optimizer. The model was trained for 15 epochs with

batch size=32 and 18 timesteps per epoch. We see that here the model doesn't really converge even after 15 epochs it converges properly at 18 epochs. So with increase in kernel size number of epochs to converge increased. This might be due to increase in number of trainable parameters compared to model1. Accuracy for class traffic is 100% and rest of the two classes have 98%. This model takes longest amount of time to run using cpu and gives decent(not too good compared to model 1) results in the end.

```
Found 600 images belonging to 3 classes.
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 32)      896

 max_pooling2d (MaxPooling2  (None, 63, 63, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 61, 61, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 30, 30, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 28, 28, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 14, 14, 128)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 12, 12, 256)       295168

 max_pooling2d_3 (MaxPoolin  (None, 6, 6, 256)         0
 g2D)

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 512)               4719104

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 3)                 1539

=================================================================
Total params: 5109059 (19.49 MB)
Trainable params: 5109059 (19.49 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
Epoch 5/15
18/18 [==============================] - 16s 931ms/step - loss: 0.3525 - accuracy: 0.8415 - val_loss: 0.2650 - val_accuracy: 0.8785
Epoch 6/15
18/18 [==============================] - 16s 933ms/step - loss: 0.2820 - accuracy: 0.8891 - val_loss: 0.1920 - val_accuracy: 0.9236
Epoch 7/15
18/18 [==============================] - 16s 936ms/step - loss: 0.2559 - accuracy: 0.9102 - val_loss: 0.2915 - val_accuracy: 0.8854
Epoch 8/15
18/18 [==============================] - 17s 984ms/step - loss: 0.1835 - accuracy: 0.9261 - val_loss: 0.1619 - val_accuracy: 0.9427
Epoch 9/15
18/18 [==============================] - 17s 952ms/step - loss: 0.1554 - accuracy: 0.9454 - val_loss: 0.1209 - val_accuracy: 0.9514
Epoch 10/15
18/18 [==============================] - 13s 744ms/step - loss: 0.1718 - accuracy: 0.9313 - val_loss: 0.1758 - val_accuracy: 0.9427
Epoch 11/15
18/18 [==============================] - 13s 742ms/step - loss: 0.1480 - accuracy: 0.9419 - val_loss: 0.1579 - val_accuracy: 0.9323
Epoch 12/15
18/18 [==============================] - 13s 745ms/step - loss: 0.1488 - accuracy: 0.9472 - val_loss: 0.1071 - val_accuracy: 0.9583
Epoch 13/15
18/18 [==============================] - 17s 975ms/step - loss: 0.0932 - accuracy: 0.9665 - val_loss: 0.0832 - val_accuracy: 0.9688
Epoch 14/15
18/18 [==============================] - 16s 928ms/step - loss: 0.0982 - accuracy: 0.9542 - val_loss: 0.0688 - val_accuracy: 0.9705
Epoch 15/15
18/18 [==============================] - 13s 740ms/step - loss: 0.0711 - accuracy: 0.9754 - val_loss: 0.0471 - val_accuracy: 0.9861
```




```
19/19 - 6s - loss: 0.0453 - accuracy: 0.9867 - 6s/epoch - 321ms/step
```

```
19/19 - 6s - loss: 0.0453 - accuracy: 0.9867 - 6s/epoch - 321ms/step
19/19 [==============================] - 7s 359ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       200
           1       0.99      0.97      0.98       200
           2       1.00      1.00      1.00       200

    accuracy                           0.99       600
   macro avg       0.99      0.99      0.99       600
weighted avg       0.99      0.99      0.99       600


Test accuracy: 0.9866666793823242
```

```python
from sklearn.metrics import confusion_matrix
# Calculate confusion matrix
conf_matrix = confusion_matrix(validation_generator.classes, y_pred_classe

# Print confusion matrix
print("\nConfusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[198   2   0]
 [  6 194   0]
 [  0   0 200]]
```

```python
# CNN model with different stride values
```

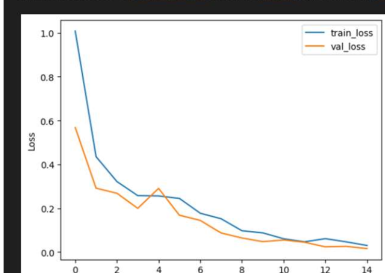
3. Model3 has 4 conv2d layers with filters 32,64,128,256 respectively. Kernel size is 3x3.Maaxpooling size=2x2. Here stride value is taken (2,1) for 2nd convolution layer and (3,2) for 3rd convolution layer. This model performed surprisingly best. gave an accuracy of 99% on train set and valid set and 100% on test set. Categorical Crossentropy loss was used and adam was used as optimizer. The model was trained for 15 epochs

with batch size=32 and 18 timesteps per epoch. We see that here the model converges very quickly at 10 epochs. Accuracy for class ambulance is 99% and rest of the two classes have 100%. This model takes shortest amount of time to run using cpu and gives best results in the end.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 126, 126, 32)      896

 max_pooling2d_8 (MaxPoolin  (None, 63, 63, 32)        0
 g2D)

 conv2d_9 (Conv2D)           (None, 31, 61, 64)        18496

 max_pooling2d_9 (MaxPoolin  (None, 15, 30, 64)        0
 g2D)

 conv2d_10 (Conv2D)          (None, 5, 14, 128)        73856

 flatten_2 (Flatten)         (None, 8960)              0

 dense_4 (Dense)             (None, 512)               4588032

 dropout_2 (Dropout)         (None, 512)               0

 dense_5 (Dense)             (None, 3)                 1539

=================================================================
Total params: 4682819 (17.86 MB)
Trainable params: 4682819 (17.86 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
Epoch 10/15
18/18 [==============================] - 17s 974ms/step - loss: 0.0878 - accuracy: 0.9718 - val_loss: 0.0479 - val_accuracy: 0.9878
Epoch 11/15
18/18 [==============================] - 16s 925ms/step - loss: 0.0610 - accuracy: 0.9736 - val_loss: 0.0554 - val_accuracy: 0.9740
Epoch 12/15
18/18 [==============================] - 13s 740ms/step - loss: 0.0472 - accuracy: 0.9894 - val_loss: 0.0452 - val_accuracy: 0.9774
Epoch 13/15
...
Epoch 14/15
18/18 [==============================] - 13s 744ms/step - loss: 0.0470 - accuracy: 0.9842 - val_loss: 0.0263 - val_accuracy: 0.9931
Epoch 15/15
18/18 [==============================] - 17s 968ms/step - loss: 0.0304 - accuracy: 0.9912 - val_loss: 0.0164 - val_accuracy: 0.9965
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...



```
19/19 - 6s - loss: 0.0158 - accuracy: 0.9967 - 6s/epoch - 322ms/step
19/19 [==============================] - 6s 319ms/step
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       200
           1       0.99      1.00      1.00       200
           2       1.00      1.00      1.00       200

    accuracy                           1.00       600
   macro avg       1.00      1.00      1.00       600
weighted avg       1.00      1.00      1.00       600


Test accuracy: 0.996666669845581
```

```python
from sklearn.metrics import confusion_matrix
# Calculate confusion matrix
conf_matrix = confusion_matrix(validation_generator.classes,

# Print confusion matrix
print("\nConfusion Matrix:")
print(conf_matrix)
```

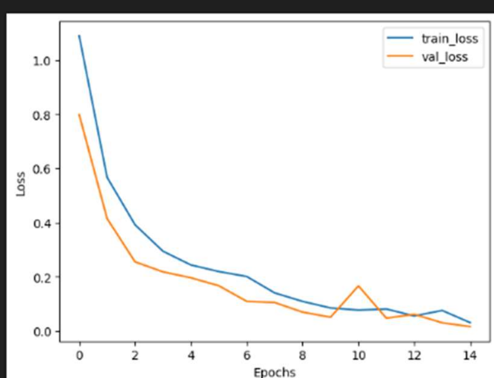
```
Confusion Matrix:
[[198   2   0]
 [  0 200   0]
 [  0   0 200]]
```

4. Model4 has 4 conv2d layers with filters 64,128,256,512 respectively. Kernel size is 3x3.Maaxpooling size=2x2. Here we increased number of filters. This model performed decently. gave an accuracy of 98% on train set ,99% on test set and validation set. Categorical Crossentropy loss was used and adam was used as optimizer. The model was trained for 15 epochs with batch size=32 and 18 timesteps per epoch. We see that here the model converges at 12 epochs. Accuracy for class traffic is 100% and rest of the two classes have 98%. This model takes significant time to run and gives good results.

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_11 (Conv2D)          (None, 127, 127, 64)      832

 max_pooling2d_10 (MaxPooli  (None, 63, 63, 64)        0
 ng2D)

 conv2d_12 (Conv2D)          (None, 62, 62, 128)       32896

 max_pooling2d_11 (MaxPooli  (None, 31, 31, 128)       0
 ng2D)

 conv2d_13 (Conv2D)          (None, 30, 30, 256)       131328

 max_pooling2d_12 (MaxPooli  (None, 15, 15, 256)       0
 ng2D)

 conv2d_14 (Conv2D)          (None, 14, 14, 512)       524800

 max_pooling2d_13 (MaxPooli  (None, 7, 7, 512)         0
 ng2D)

 flatten_3 (Flatten)         (None, 25088)             0

 dense_6 (Dense)             (None, 512)               12845568

 dropout_3 (Dropout)         (None, 512)               0

 dense_7 (Dense)             (None, 3)                 1539

=================================================================
Total params: 13536963 (51.64 MB)
Trainable params: 13536963 (51.64 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/15
```

```
Epoch 11/15
18/18 [==============================] - 16s 939ms/step - loss: 0.0770 - accuracy: 0.9683 - val_loss: 0.1664 - val_accuracy: 0.9340
Epoch 12/15
18/18 [==============================] - 13s 756ms/step - loss: 0.0809 - accuracy: 0.9736 - val_loss: 0.0472 - val_accuracy: 0.9792
Epoch 13/15
...
Epoch 14/15
18/18 [==============================] - 14s 776ms/step - loss: 0.0756 - accuracy: 0.9701 - val_loss: 0.0301 - val_accuracy: 0.9931
Epoch 15/15
18/18 [==============================] - 13s 747ms/step - loss: 0.0308 - accuracy: 0.9894 - val_loss: 0.0161 - val_accuracy: 0.9948
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...




```
19/19 - 6s - loss: 0.0155 - accuracy: 0.9950 - 6s/epoch - 326ms/step
19/19 [==============================] - 6s 324ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       200
           1       0.99      0.99      0.99       200
           2       1.00      1.00      1.00       200

    accuracy                           0.99       600
   macro avg       0.99      0.99      0.99       600
weighted avg       0.99      0.99      0.99       600
```

```
[24]
...
Confusion Matrix:
[[199   1   0]
 [  1 198   1]
 [  0   0 200]]
```

Task2:

We can represent text(sentence) as a matrix of word embeddings of each word. We are using this concept to represent text and we'll use CNN.

Matrix form of text:

```
print("text matrix")
print(X_train)
print(X_train.shape)
```

```
text matrix
[[   0    0    0 ...   19  178   32]
 [   0    0    0 ...   16  145   95]
 [   0    0    0 ...    7  129  113]
 ...
 [   0    0    0 ...    4 3586    2]
 [   0    0    0 ...   12    9   23]
 [   0    0    0 ...  204  131    9]]
(25000, 450)
```

We are using 1d CNN here. If our matrix version of text is of size (m,n) where n are number of words in text and m is size of embedding then kernel size is (n,x) here n is fixed. n here is 25000 number of imdb text reviews. n is max_words = 450

```
(25000,)
```

```python
# Padding the data samples to a maximum review length in words
max_words = 450
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
# Building the CNN Model
model = Sequential()      # initilaizing the Sequential nature for CNN model
# Adding the embedding layer which will take in maximum of 450 words as input and provide a 32 dimensional output of those words which belong in the top_words dictionary
model.add(Embedding(top_words, 32, input_length=max_words))
model.add(Conv1D(32, 3, padding='same', activation='relu'))
model.add(MaxPooling1D())
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```python
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 450, 32)           224000

 conv1d (Conv1D)             (None, 450, 32)           3104

 max_pooling1d (MaxPooling1  (None, 225, 32)           0
 D)

 flatten_4 (Flatten)         (None, 7200)              0

 dense_8 (Dense)             (None, 250)               1800250

 dense_9 (Dense)             (None, 1)                 251

=================================================================
Total params: 2027605 (7.73 MB)
Trainable params: 2027605 (7.73 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```python
# Fitting the data onto model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=2, batch_size=128, verbose=2)
# Getting score metrics from our model
scores = model.evaluate(X_test, y_test, verbose=0)
# Displays the accuracy of correct sentiment prediction over test data
```

1m 30s completed at 12:01 AM

The model first consists of embedding layer in which we will find the embeddings of the top 7000 words into a 32 dimensional embedding and the input we can take in is defined as the maximum length of a review allowed.

Then, we add the convolutional layer and max-pooling layer. Finally, we flatten those matrices into vectors and add dense layers(basically scale,rotating and transform the vector by multiplying Matrix and vector).

The last Dense layer is having one as parameter because we are doing a binary classification and so we need only one output node in our vector.

```python
[21] # Fitting the data onto model
     model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=2, batch_size=128, verbose=2)
     # Getting score metrics from our model
     scores = model.evaluate(X_test, y_test, verbose=0)
     # Displays the accuracy of correct sentiment prediction over test data
     print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Epoch 1/2
196/196 - 30s - loss: 0.4434 - accuracy: 0.7666 - val_loss: 0.2750 - val_accuracy: 0.8846 - 30s/epoch - 155ms/step
Epoch 2/2
196/196 - 25s - loss: 0.2020 - accuracy: 0.9222 - val_loss: 0.2743 - val_accuracy: 0.8864 - 25s/epoch - 125ms/step
Accuracy: 88.64%
```

We achieved an accuracy of 88% on test set.