Configuration Manual

MSc Research Project
Data Analytics

Sumit Singh
Student ID: x20135769

School of Computing
National College of Ireland

Supervisor I: Dr. Paul Stynes
Supervisor II: Dr. Pramod Pathak

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sumit Singh
Student ID:	x20135769
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr. Paul Stynes
Submission Due Date:	31/01/2022
Project Title:	Configuration Manual
Word Count:	814
Page Count:	13

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	31st January 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Sumit Singh
x20135769

1 Introduction

The objective of the manual is to provide details of the system setup, software, hardware and network configurations for the research project, A Deep Learning Model for Irish English and Hindi Language Identification

2 Hardware Components

The implementation of the project was performed using a virtual machine created using google cloud platform(GCP). The configuration of the machine are as follows:

Machine Platform	Virtual machine on Google Cloud Platform(GCP)
Machine Type	N1-standard-2(2 vCPU, 7.5 GB Memory)
GPU	1 X NVIDIA Tesla T4
Operating System	Debian 10

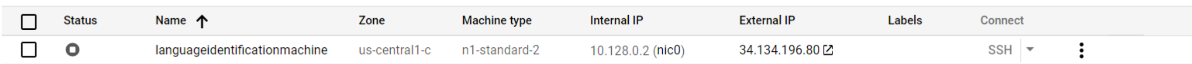


Figure 1: Virtual Machine Instance

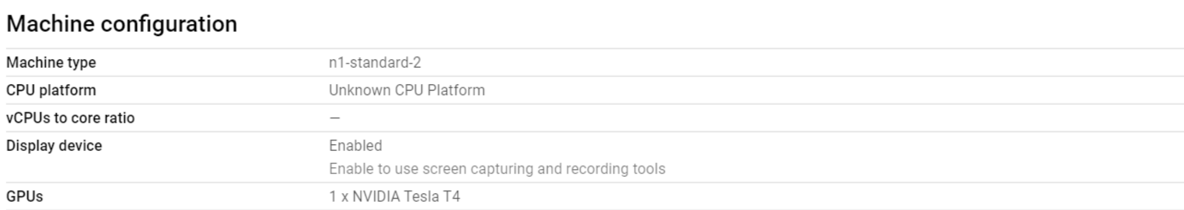
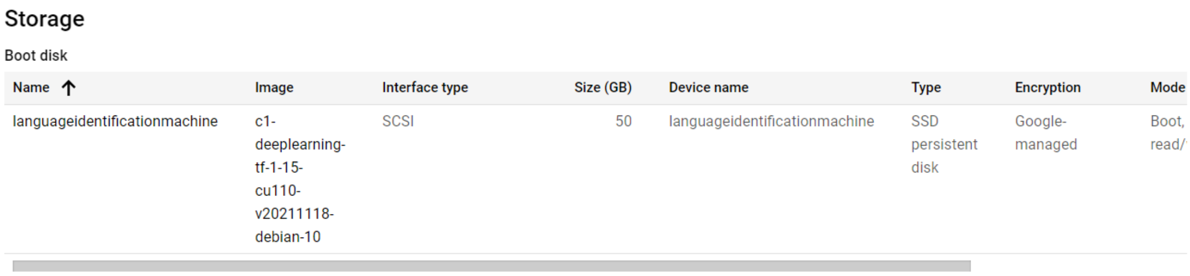


Figure 2: GPU Confirguration of the Virtual Machine

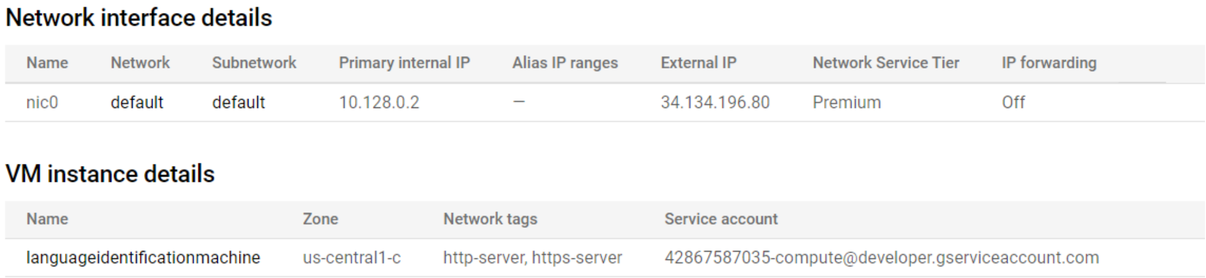


Storage

Boot disk

Name ↑	Image	Interface type	Size (GB)	Device name	Type	Encryption	Mode
languageidentificationmachine	c1-deeplearning-tf-1-15-cu110-v20211118-debian-10	SCSI	50	languageidentificationmachine	SSD persistent disk	Google-managed	Boot, read/

Figure 3: Operating Image Details of the Virtual Machine



Network interface details

Name	Network	Subnetwork	Primary internal IP	Alias IP ranges	External IP	Network Service Tier	IP forwarding
nic0	default	default	10.128.0.2	—	34.134.196.80	Premium	Off

VM instance details

Name	Zone	Network tags	Service account
languageidentificationmachine	us-central1-c	http-server, https-server	42867587035-compute@developer.gserviceaccount.com

Figure 4: Network Configurations of the Virtual Machine

3 Software Components

Following software components were used for the development of the project:

Programming tool	Jupyter Notebook
Programming language	Python 3, VBScript
Office Applications	Microsoft Excel 365
Report Compilation	Overleaf
Browser	Google Chrome Version 96.0.4664.93

4 Python library package

Following libraries must be installed before running the code for the project implementation:

PyDub	!pip install pydub
ffmpeg	!pip install ffmpeg
Librosa	!pip install librosa
Numpy	!pip install numpy
Mathplotlib	!pip install matplotlib
Pandas	!pip install pandas
Keras	!pip install keras
Tenserflow	!pip install tensorflow ==1.15
Tenserflow GPU	!pip install tensorflow-gpu==1.15
Seaborn	!pip install seaborn
EfficientNet Model	!pip install -U efficientnet

5 Google Cloud Platform (GCP) Setup

5.1 Create a virtual machine instance

GCP is a cloud platform and to create a virtual machine on the cloud following steps must be followed:

1. Sign-up and create a login account with Google Cloud Platform.
2. On sign-up new users are provided with $300 for free services.
3. Create a new instance in the compute engine console and select a region for and zone for machine. The machine used in the experiment was created in us-central1(Iowa) region under us-central1-c zone.
4. Select GPU type as NVIDIA Tesla T4 and number of GPU's as 1.

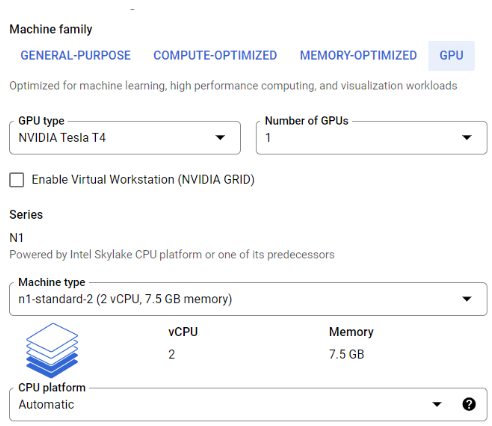


Figure 5: Configure Machine Image

5. Click on the change button in the boot disk section and increase the size of the selected disk to 50 GB.

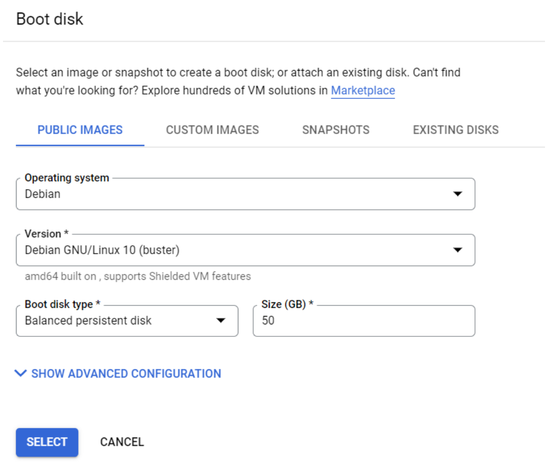


Figure 6: Configure Storage

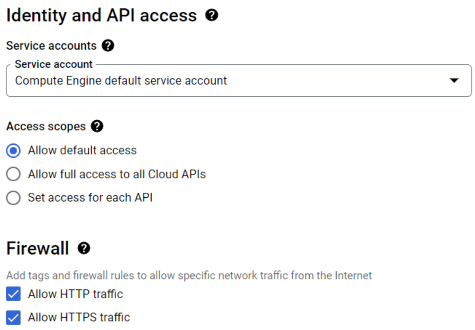
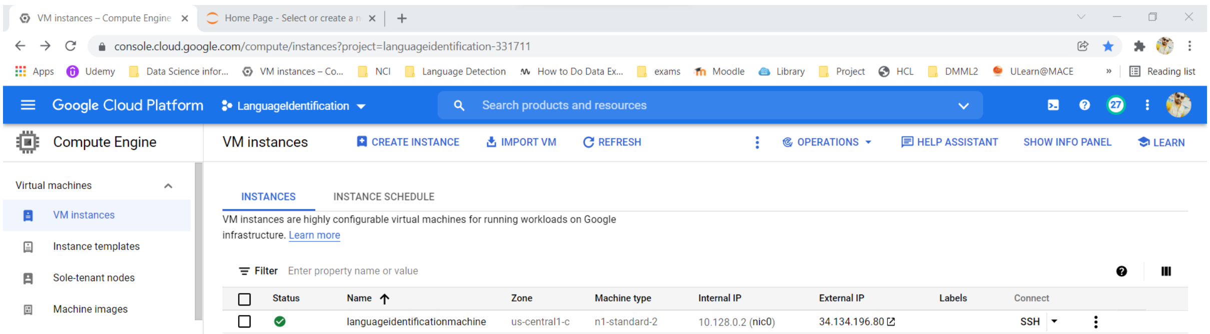


Figure 7: Configure Network Traffic

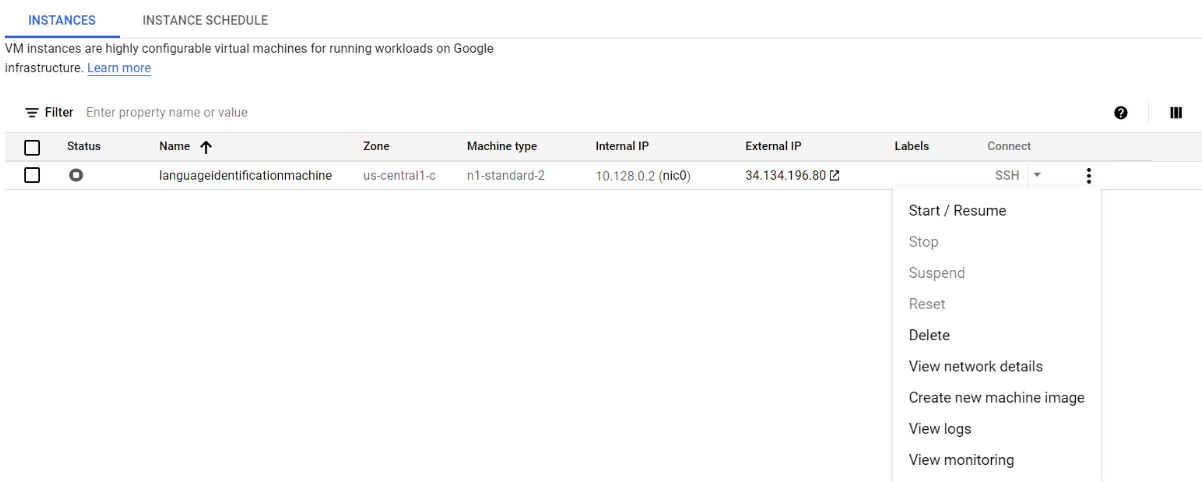
6. Check the Allow HTTP and HTTPS check-boxes under the firewall section
7. Click on create instance button and the instance will be created with 2 to 3 minutes.

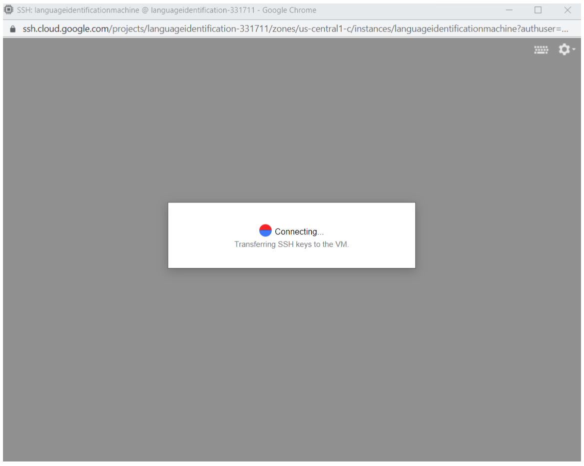
5.2 Open Jupyter notebook on GCP Debian Image

1. Created instance is displayed in the VM instances section of the Compute Engine tab



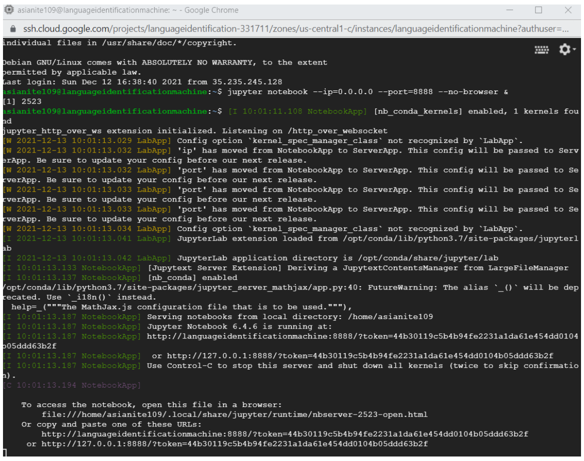
2. Start the instance and a notification is generated on the bottom of the page for confirmation.



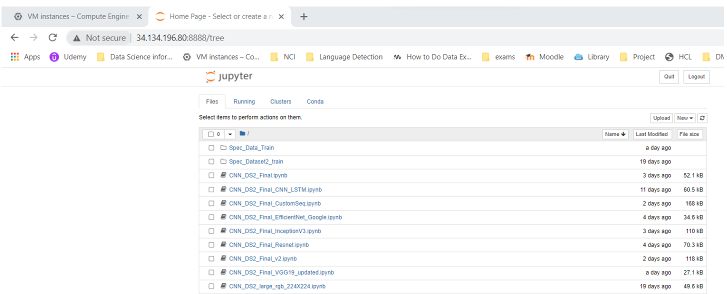


3. Once the machine is started, click on the SSH button on the main instance menu and wait for the connection to be established.

4. A black window pops up, type the following command to start the Jupyter notebook. jupyter notebook –ip=0.0.0.0 –port=8888 –no-browser



5. Use the external IP address of the instance followed by a colon sign along with the port number to open up the Jupyter notebook on the default browser. An example of the address used in the experiment is 34.134.196.80:8888/



5.3 Execution

1. Once the Notebook is open, upload the zip folder into the python notebook using the upload button on the top- right corner..

2. Create a new python notebook and use the below code to unzip the file.

```
In [2]:
# importing unzip module
from zipfile import ZipFile

# specifying the zip file name
file_name = "Language_identification_Code_Artifacts.zip"

# opening the zip file in READ mode
with ZipFile(file_name, 'r') as zip:
    # printing all the contents of the zip file
    zip.printdir()

    # extracting all the files
    print('Extracting all the files now...')
    zip.extractall()
    print('Done!')
```

3. The next step is to convert the provided audio files from .mp3 dataset into .wav dataset. However, the converted datasets are already provided in the unzipped folder. Refer to "Convert_mp3_to_wav.ipynb" python file for execution and reference.

4. The .wav files are renamed using a Vbscript file attached in the folder. The files are renamed for easier tracking and processing.

5. The renamed files name are combined in an excel file and the three different languages are given a label for multi-label classification. The labels are 0 for English, 1 for Hindi and 2 for Irish. Refer to the file "train.csv" for reference.

6. The converted .wav audio files are then converted into mel-spectrograms. Librosa is used to import the audio files and based on research using literature survey, mel banks are defined and mel spectrograms are created for feature extraction. Refer to "Create_Spectrograms.ipynb" python file for reference.

7. The mel-spectrograms are stored in a folder "Spec_Data_Train" and upload using ImageDataGenerator function from the Keras library. The function is also used for preprocessing using augmentation techniques.

   (a) Experiment 1: CNN model is initialized and and training is done on training and validation dataset using train_generators.fit_generator function is used to predict the languages for the test samples and are evaluated using classification report, confusion matrix, Accuracy and loss plots from the sklearn and Matplotlib libraries. Refer to "SLID_CustomSeq-CNN.ipynb" file for reference

   (b) Experiment 2: Resnet50 model is initialized using keras library and training and testing is performed on the dataset. Refer to "SLID_Resnet.ipynb" file for reference

   (c) Experiment 3: InveptionV3 model is initialized using keras library and training and testing is performed on the dataset. Refer to "SLID_InceptionV3.ipynb" file for reference

   (d) Experiment 4: EfficientNet model is initialized using keras library and training and testing is performed on the dataset. Refer to "SLID_EfficientNet.ipynb" file for reference

6 Project Development

Different crucial sections of the code implementation have been highlighted below.

6.1 Import Libraries

```
In [1]:
import pandas as pd
import numpy as np
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt
from keras import regularizers
from keras.regularizers import l2
from tensorflow.keras import optimizers
from keras.optimizers import RMSprop
from keras.models import Sequential
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Activation, Flatten, Dropout, BatchNormalization
from keras.initializers import glorot_normal, glorot_uniform, he_normal, he_uniform, Constant
from keras.layers import Conv2D, MaxPooling2D

%matplotlib inline

Using TensorFlow backend.
2021-12-10 23:17:07.169110: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Successfully opened dynamic library libcudart.so.11.0
```

6.2 Load Audio and Visual Sound

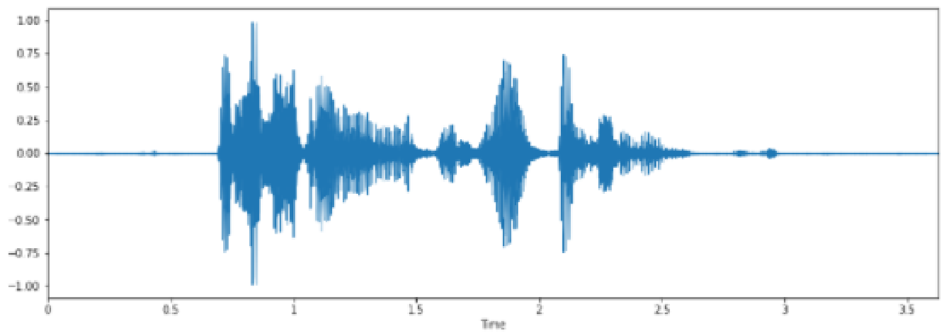
```
In [5]: ipd.Audio(scale_file)
Out[5]: 0:00 / 0:03

In [6]: # Load audio files with librosa
scale, sr = librosa.load(scale_file)

In [9]: import librosa.display
import matplotlib.pyplot as plt

# x-axis has been converted to time using our sample rate.
# matplotlib plt.plot(y), would output the same figure, but with sample
# number on the x-axis instead of seconds
plt.figure(figsize=(15, 5))
librosa.display.waveplot(scale, sr=sr)

Out[9]: <matplotlib.collections.PolyCollection at 0x1ee06365c10>
```



6.3 Define Mel Banks

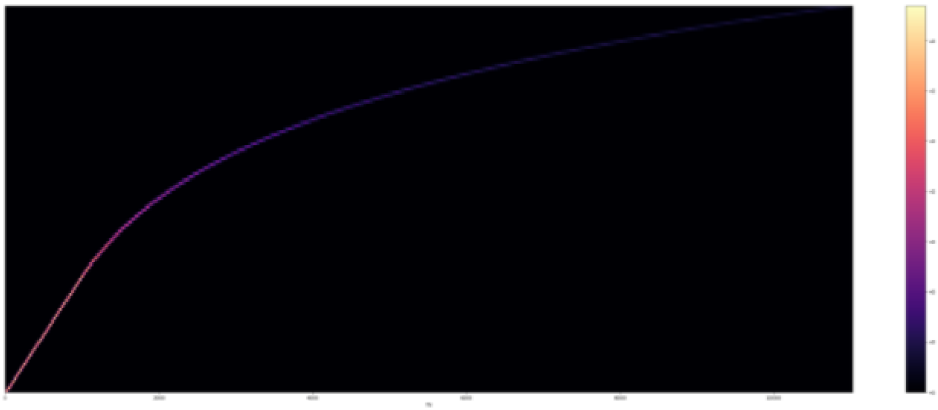
Mel Banks

```
In [19]: filter_banks = librosa.filters.mel(n_fft=2048, sr=22050, n_mels=128)
```

```
In [20]: filter_banks.shape
```

```
Out[20]: (128, 1025)
```

```
In [21]: plt.figure(figsize=(40, 15))
         librosa.display.specshow(filter_banks,
                                  sr=sr,
                                  x_axis="linear")
         plt.colorbar(format="%+2.f")
         plt.show()
```

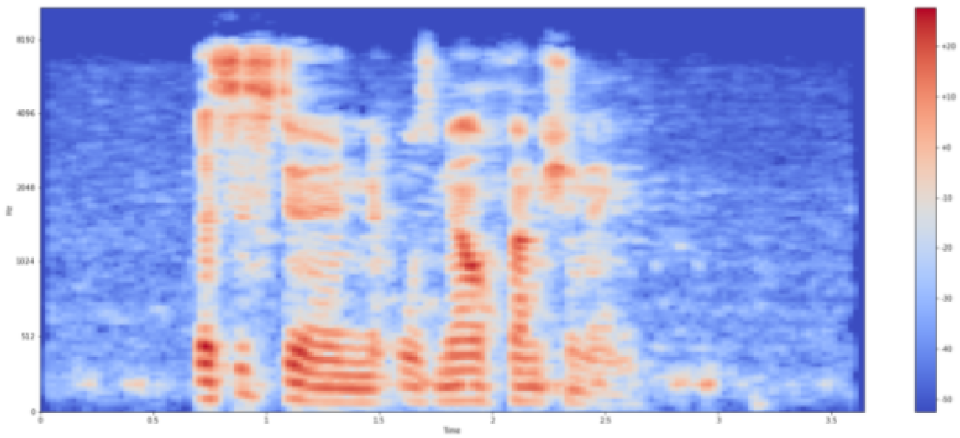


6.4 Extract Spectrograms

```
In [16]: log_mel_spectrogram.shape
```

```
Out[16]: (128, 157)
```

```
In [17]: plt.figure(figsize=(25, 10))
         librosa.display.specshow(log_mel_spectrogram,
                                  x_axis="time",
                                  y_axis="mel",
                                  sr=sr)
         plt.colorbar(format="%+2.f")
         plt.show()
```



6.5 Data Precrocessing

```
In [3]: dataset=pd.read_csv('train.csv')
```

```
In [4]: dict_genres = {'English':0, 'Hindi':1, 'Irish':2}

        reverse_map = {v: k for k, v in dict_genres.items()}
        print(reverse_map)

{0: 'English', 1: 'Hindi', 2: 'Irish'}
```

```
In [5]: shuffled_df = dataset.sample(frac=1)
```

```
In [6]: df = shuffled_df
        df['SpectrogramID'] = df['SpectrogramID'].astype(str) + '.png'
        df.head()
```

Out[6]:

	SpectrogramID	LanguageID
2897	Irish_0900.png	2
1442	Hindi_0445.png	1
2929	Irish_0932.png	2
2186	Irish_0189.png	2
2693	Irish_0696.png	2

```
In [7]: df = pd.concat([df, pd.get_dummies(df['LanguageID'], prefix = 'LanguageID')],axis=1)
        df.drop(['LanguageID'], axis = 1, inplace =True)

        df.head()
```

Out[7]:

	SpectrogramID	LanguageID_0	LanguageID_1	LanguageID_2
2897	Irish_0900.png	0	0	1
1442	Hindi_0445.png	0	1	0
2929	Irish_0932.png	0	0	1
2186	Irish_0189.png	0	0	1
2693	Irish_0696.png	0	0	1

6.6 Import Spectrograms to Jupyter Notebook and Preprocessing

```
In [8]: columns=["LanguageID_0", "LanguageID_1", "LanguageID_2"]

        train_datagen=ImageDataGenerator(rescale=1./255.)
        test_datagen=ImageDataGenerator(rescale=1./255.)

        train_generator=train_datagen.flow_from_dataframe(
            dataframe=df[:2000],
            directory='Spec_Data_Train',
            x_col="SpectrogramID",
            y_col=columns,
            color_mode='rgb',
            batch_size=100,
            seed=42,
            shuffle=True,
            class_mode="raw",
            target_size=(224, 224))

        valid_generator=test_datagen.flow_from_dataframe(
            dataframe=df[2000:2600],
            directory='Spec_Data_Train',
            x_col="SpectrogramID",
            y_col=columns,
            color_mode='rgb',
            batch_size=50,
            seed=42,
            shuffle=True,
            class_mode="raw",
            target_size=(224, 224))

        test_generator=test_datagen.flow_from_dataframe(
            dataframe=df[2698:3100],
            directory='Spec_Data_Train',
            x_col="SpectrogramID",
            color_mode='rgb',
            batch_size=1,
            seed=42,
            shuffle=False,
            class_mode=None,
            target_size=(224, 224))

Found 2000 validated image filenames.
Found 600 validated image filenames.
Found 400 validated image filenames.
```

6.7 CNN Model

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 224, 224, 128)	3584
activation_10 (Activation)	(None, 224, 224, 128)	0
conv2d_10 (Conv2D)	(None, 224, 224, 128)	147584
activation_11 (Activation)	(None, 224, 224, 128)	0
max_pooling2d_5 (MaxPooling2	(None, 112, 112, 128)	0
dropout_6 (Dropout)	(None, 112, 112, 128)	0
conv2d_11 (Conv2D)	(None, 112, 112, 64)	73792
activation_12 (Activation)	(None, 112, 112, 64)	0
conv2d_12 (Conv2D)	(None, 112, 112, 64)	36928
activation_13 (Activation)	(None, 112, 112, 64)	0
max_pooling2d_6 (MaxPooling2	(None, 56, 56, 64)	0
dropout_7 (Dropout)	(None, 56, 56, 64)	0
conv2d_13 (Conv2D)	(None, 56, 56, 64)	36928
activation_14 (Activation)	(None, 56, 56, 64)	0
conv2d_14 (Conv2D)	(None, 54, 54, 64)	36928
activation_15 (Activation)	(None, 54, 54, 64)	0
max_pooling2d_7 (MaxPooling2	(None, 27, 27, 64)	0
dropout_8 (Dropout)	(None, 27, 27, 64)	0
conv2d_15 (Conv2D)	(None, 27, 27, 32)	18464
activation_16 (Activation)	(None, 27, 27, 32)	0
conv2d_16 (Conv2D)	(None, 25, 25, 32)	9248
activation_17 (Activation)	(None, 25, 25, 32)	0
max_pooling2d_8 (MaxPooling2	(None, 12, 12, 32)	0
dropout_9 (Dropout)	(None, 12, 12, 32)	0
flatten_2 (Flatten)	(None, 4608)	0
dense_3 (Dense)	(None, 512)	2359808
activation_18 (Activation)	(None, 512)	0
dropout_10 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 3)	1539

Total params: 2,724,803
Trainable params: 2,724,803
Non-trainable params: 0

6.8 Model Training

Model Training and Validation

In [37]:

```
STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
STEP_SIZE_VALID=valid_generator.n//valid_generator.batch_size
STEP_SIZE_TEST=test_generator.n//test_generator.batch_size
hist = model.fit_generator(generator=train_generator,
                    steps_per_epoch=STEP_SIZE_TRAIN,
                    validation_data=valid_generator,
                    validation_steps=STEP_SIZE_VALID,
                    epochs=145)
```

```
0.9356
Epoch 140/145
20/20 [==============================] - 31s 2s/step - loss: 0.1782 - accuracy: 0.9658 - val_loss: 0.2093 - val_accuracy:
0.9400
Epoch 141/145
20/20 [==============================] - 31s 2s/step - loss: 0.1757 - accuracy: 0.9657 - val_loss: 0.1997 - val_accuracy:
0.9367
Epoch 142/145
20/20 [==============================] - 31s 2s/step - loss: 0.1737 - accuracy: 0.9693 - val_loss: 0.2820 - val_accuracy:
0.9389
Epoch 143/145
20/20 [==============================] - 31s 2s/step - loss: 0.1620 - accuracy: 0.9715 - val_loss: 0.1845 - val_accuracy:
0.9372
Epoch 144/145
20/20 [==============================] - 31s 2s/step - loss: 0.1563 - accuracy: 0.9765 - val_loss: 0.2120 - val_accuracy:
0.9422
Epoch 145/145
20/20 [==============================] - 31s 2s/step - loss: 0.1602 - accuracy: 0.9725 - val_loss: 0.2920 - val_accuracy:
0.9356
```

6.9 Testing Spectrograms and Probability of Each Language

Load the test generator for predictions

In [38]:

```
# Load the test generator for predictionstest_generator.reset()
pred = model.predict_generator(test_generator,steps=STEP_SIZE_TEST,verbose=1)

lang = np.argmax(pred,axis=1)
columns=["LanguageID_0", "LanguageID_1", "LanguageID_2"]
#columns should be the same order of y_col

results=pd.DataFrame(pred, columns=columns)
results["SpectrogramID"]=test_generator.filenames
ordered_cols=["SpectrogramID"]+columns
results=results[ordered_cols]#To get the same column order
results['lang'] = lang

file_id = results['SpectrogramID']
results = pd.concat([file_id, results[['LanguageID_0', 'LanguageID_1', 'LanguageID_2', 'lang']]], axis = 1)
results.head()
```

```
400/400 [==============================] - 4s 9ms/step
```

Out[38]:

	SpectrogramID	LanguageID_0	LanguageID_1	LanguageID_2	lang
0	Irish_0964.png	8.118259e-07	0.000000	0.999998	2
1	Irish_0707.png	0.000000e+00	0.000000	1.000000	2
2	Hindi_0799.png	8.443273e-04	0.998897	0.000265	1
3	Hindi_0824.png	6.894491e-02	0.986645	0.000007	1
4	Hindi_0830.png	3.543124e-02	0.988198	0.000034	1

6.10 Evaluations

Evaluation and Results

In [62]:

```
# Evaluation and Results# Evaluate classifier
from sklearn import metrics
from sklearn.metrics import accuracy_score, confusion_matrix

print('Classifier Metrics:')
y_test = shuffled_df[2698:3100].LanguageID
target_names = dict_genres.keys()
print(y_test.shape, results.lang.shape)
print(metrics.classification_report(y_test, results.lang,target_names=target_names))
print('Accuracy Score: {:.2%}'.format(metrics.accuracy_score(y_test, results.lang)))

print("\n Confusion Matrix:")
cm = confusion_matrix(y_test, results.lang)
print(cm)
df_cm = pd.DataFrame(cm, range(3),range(3))
plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size

```

```
Classifier Metrics:
(400,) (400,)
              precision    recall  f1-score   support

     English       0.94      0.88      0.91       137
       Hindi       0.92      0.96      0.94       135
       Irish       0.95      0.96      0.96       128

    accuracy                           0.94       400
   macro avg       0.94      0.94      0.94       400
weighted avg       0.94      0.94      0.93       400

Accuracy Score: 93.50%

 Confusion Matrix:
[[121  10   6]
 [  5 130   0]
 [  3   2 123]]
<Figure size 720x504 with 0 Axes>
```

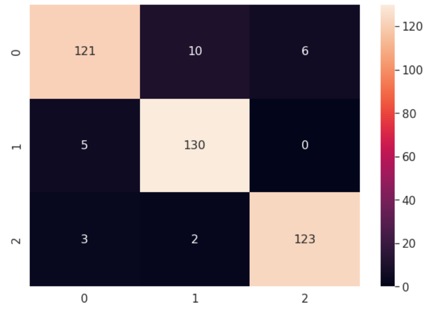


Figure 8: Confusion Matrix for CNN Model

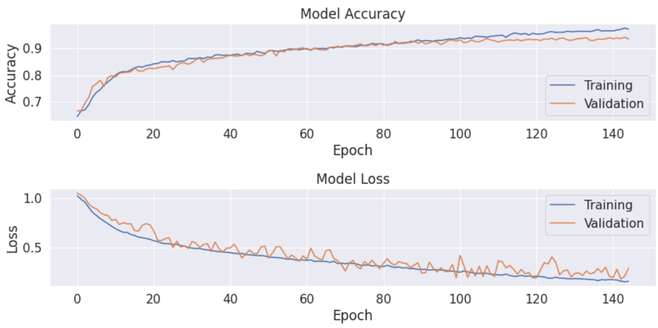


Figure 9: Accuracy and Loss Plot for CNN Model

6.11 Resnet Model Training

Model Training and Validation

In [24]:
```
STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
STEP_SIZE_VALID=valid_generator.n//valid_generator.batch_size
STEP_SIZE_TEST=test_generator.n//test_generator.batch_size

hist = model_resnet.fit_generator(generator=train_generator,
                    steps_per_epoch=STEP_SIZE_TRAIN,
                    validation_data=valid_generator,
                    validation_steps=STEP_SIZE_VALID,
                    epochs=50)
```

```
cy: 0.9289
Epoch 45/50
100/100 [==============================] - 31s 306ms/step - loss: 0.0118 - accuracy: 0.9955 - val_loss: 1.0905 - val_accura
cy: 0.8656
Epoch 46/50
100/100 [==============================] - 31s 306ms/step - loss: 0.0135 - accuracy: 0.9960 - val_loss: 1.1073 - val_accura
cy: 0.8967
Epoch 47/50
100/100 [==============================] - 31s 305ms/step - loss: 0.0171 - accuracy: 0.9940 - val_loss: 0.0401 - val_accura
cy: 0.9256
Epoch 48/50
100/100 [==============================] - 30s 305ms/step - loss: 0.0154 - accuracy: 0.9967 - val_loss: 1.2497 - val_accura
cy: 0.9272
Epoch 49/50
100/100 [==============================] - 31s 307ms/step - loss: 0.0120 - accuracy: 0.9960 - val_loss: 1.8015 - val_accura
cy: 0.9211
Epoch 50/50
100/100 [==============================] - 31s 306ms/step - loss: 0.0148 - accuracy: 0.9963 - val_loss: 0.2000 - val_accura
cy: 0.9339
```

6.12 Inception Model Training

Model Training and Validation

```
In [9]:
STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
STEP_SIZE_VALID=valid_generator.n//valid_generator.batch_size
STEP_SIZE_TEST=test_generator.n//test_generator.batch_size

hist = model_Inception.fit_generator(generator=train_generator,
                    steps_per_epoch=STEP_SIZE_TRAIN,
                    validation_data=valid_generator,
                    validation_steps=STEP_SIZE_VALID,
                    epochs=50)
```

```
cy: 0.9567
Epoch 45/50
100/100 [==============================] - 23s 225ms/step - loss: 0.0232 - accuracy: 0.9972 - val_loss: 0.4121 - val_accura
cy: 0.9483
Epoch 46/50
100/100 [==============================] - 22s 224ms/step - loss: 0.0178 - accuracy: 0.9957 - val_loss: 0.3250 - val_accura
cy: 0.9328
Epoch 47/50
100/100 [==============================] - 23s 225ms/step - loss: 0.0067 - accuracy: 0.9980 - val_loss: 0.2219 - val_accura
cy: 0.9444
Epoch 48/50
100/100 [==============================] - 23s 225ms/step - loss: 0.0126 - accuracy: 0.9967 - val_loss: 0.4115 - val_accura
cy: 0.9289
Epoch 49/50
100/100 [==============================] - 22s 224ms/step - loss: 0.0118 - accuracy: 0.9973 - val_loss: 0.4610 - val_accura
cy: 0.9422
Epoch 50/50
100/100 [==============================] - 23s 226ms/step - loss: 0.0182 - accuracy: 0.9962 - val_loss: 0.5069 - val_accura
cy: 0.9306
```

6.13 EfficientNet Model Training

Model Training and Validation

```
In [10]:
STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
STEP_SIZE_VALID=valid_generator.n//valid_generator.batch_size
STEP_SIZE_TEST=test_generator.n//test_generator.batch_size
hist = model_EfficientNet.fit_generator(generator=train_generator,
                    steps_per_epoch=STEP_SIZE_TRAIN,
                    validation_data=valid_generator,
                    validation_steps=STEP_SIZE_VALID,
                    epochs=50)
```

```
cy: 0.9717
Epoch 45/50
100/100 [==============================] - 25s 251ms/step - loss: 0.0219 - accuracy: 0.9963 - val_loss: 2.6032 - val_accura
cy: 0.9661
Epoch 46/50
100/100 [==============================] - 25s 248ms/step - loss: 0.0065 - accuracy: 0.9988 - val_loss: 0.0060 - val_accura
cy: 0.9689
Epoch 47/50
100/100 [==============================] - 25s 249ms/step - loss: 0.0216 - accuracy: 0.9973 - val_loss: 0.2639 - val_accura
cy: 0.9667
Epoch 48/50
100/100 [==============================] - 25s 253ms/step - loss: 0.0313 - accuracy: 0.9963 - val_loss: 1.0887 - val_accura
cy: 0.9778
Epoch 49/50
100/100 [==============================] - 25s 247ms/step - loss: 0.0036 - accuracy: 0.9995 - val_loss: 0.2843 - val_accura
cy: 0.9683
Epoch 50/50
100/100 [==============================] - 25s 249ms/step - loss: 0.0166 - accuracy: 0.9980 - val_loss: 0.3988 - val_accura
cy: 0.9594
```