California State University, Fresno
Lyles College of Engineering
Electrical and Computer Engineering Department

PROJECT REPORT

Assignment:	**PROJECT 2**
Project Title:	**Classification of Spoken Digits using Deep Neural Networks**
Course Title:	**ECE 172 (Fundamentals of Machine Learning)**
Instructor:	**Dr. Hovannes Kulhandjian**

Prepared by: Aryan Singh, Omer Al Sumeri

Date Submitted: 04/28/2024

INSTRUCTOR SECTION

Comments: ____

____

____

____

____

____

____

Final Grade: ____________________

TABLE OF CONTENTS

LIST OF FIGURES

1. Objectives

The objective of this project is to perform accurate audio classification using different types of Artificial Neural Network computation models. In this project the audio files containing spoken digits from 0-9 are being classified using DCNN (Deep Convolutional Neural Networks), Mobilenet, ResNet50, Inception V3 and YOLO (You Only Look Once).

2. Theoretical Background

This project involves the use of various deep neural networks for audio classification as well as the preprocessing method of converting the .wav audio files into spectrograms. The different types of neural network models used were DCNN and then the results of this method were compared with the results of pretrained models like Mobilenet, ResNet-50, Inception V3 and YOLO. Python programming language was used for this project with imported deep learning libraries like TensorFlow, Keras and Scikit-learn and data manipulation libraries like Numpy and matplotlib.pyplot.

2.1 Preprocessing

In order to classify the audio files (.wav) using the various neural networks, they have to be converted into a form that could be spatially structured, similar to images. Therefore, the audio files have to be converted to spectrograms, which capture both the time and frequency domains of an audio signal. The Spectrograms, as shown in Figure 1, depict a 2D representation of how the signal's frequency changes over time with the X axis representing time and Y axis representing the frequency at that instant. Using spectrograms makes it easier to perform dimensionality reduction, therefore allowing processes like DCNN to process and extract meaningful features.

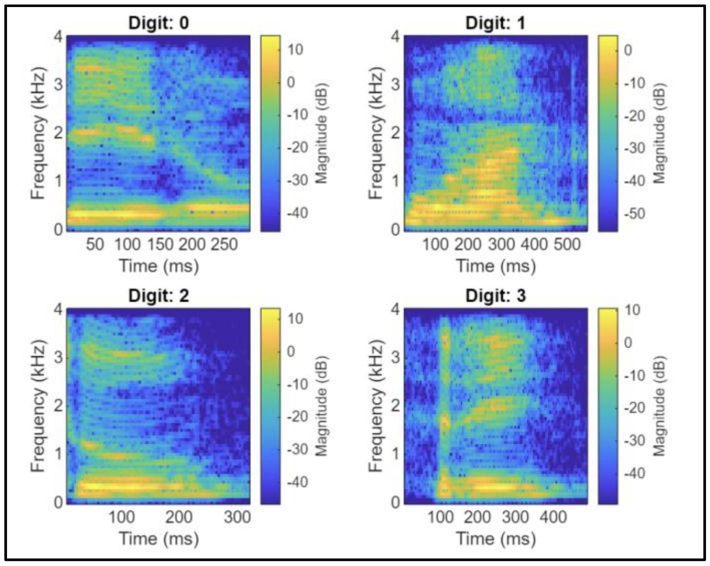


Figure 1: Spectrograms for different digits (0,1,2,3)

2.2 DCNN

Deep Convolutional Neural Networks are a type of Artificial Neural Network designed for processing and analyzing data such as images and audio/video files. They consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers.

The convolutional layers detect spatial patterns and features in the input data, which are spectrograms in our case, and produce feature maps using multiple filters. This process of extracting important features from the input data using filters is known as convolution. These filters/kernels slide over the input data and extract the local features. Each filter detects a specific pattern or feature such as edges, textures or shapes at different spatial locations of the input data (different frequencies at different instances in the case of spectrograms).

The second layer of DCNN are the pooling layers, which follow the convolutional layers. These layers reduce the spatial dimensions of feature maps while retaining the most important features. The Max-Pooling layers select the maximum values within each pooling window, therefore preserving the most dominant features of the feature maps. This layer is essential to reduce computational complexity, memory requirements and prevent overfitting by reducing the spatial dimensions of the data.

The final layer of the DCNN architecture is the fully connected layer or the dense layer. These layers perform data classification and regression tasks on the features extracted from the previous 2 layers. These layers connect each neuron from the previous layer to every neuron in the current layer to form a fully connected structure. These layers can perform decision making based on the features extracted from the previous layers which allows the network to make predictions or classifications. Functions like softmax (used by our team) are used by these layers to perform classification using probabilistic predictions i.e. converting raw scores into probabilities, therefore allowing the neural networks to make predictions for multi-class classifications, based on these probabilities.

Figure 2 shows the entire DCNN architecture, representing how these 3 layers work together and perform convolution and pooling, followed by classification.

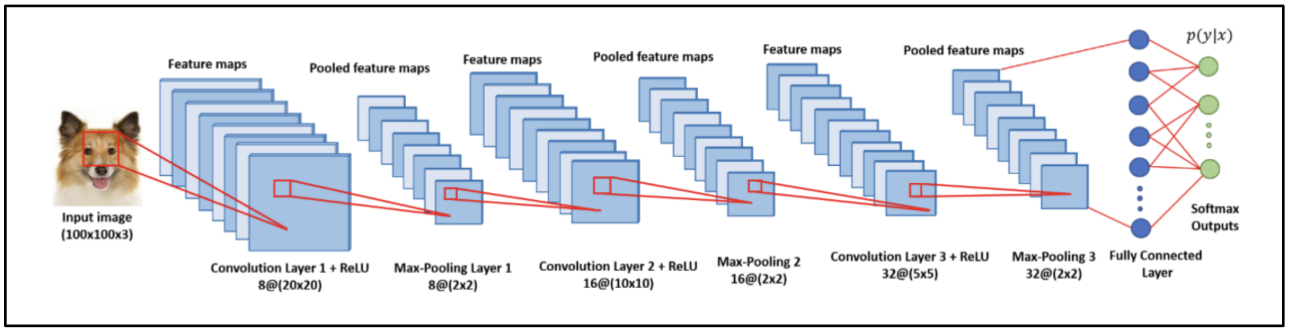


Figure 2: Deep Convolutional Neural Network Architecture

2.3 MobileNet

MobileNet is a type of Convolutional Neural Network (CNN) architecture that is designed to be lightweight and efficient for image classification on mobile or embedded devices. This type of architecture is designed for devices with limited computational capacity since it minimizes the computational resources required for classification while ensuring high accuracy. MobileNet employs depthwise separable convolution, which splits the standard convolution into 2 separate layers: depthwise convolution and pointwise convolution.

Depthwise convolution is a technique that applies a separate filter for each input channel, which means that there is a separate filter associated with each input for capturing spatial information specific to each channel.

Pointwise convolution takes the feature maps created by the depthwise convolution filters and uses a 1X1 filter to combine all of the outputs, thereby reducing computational complexity while preserving the most important features.

Figure 3 represents how Depthwise separable convolution is performed in MobileNet.

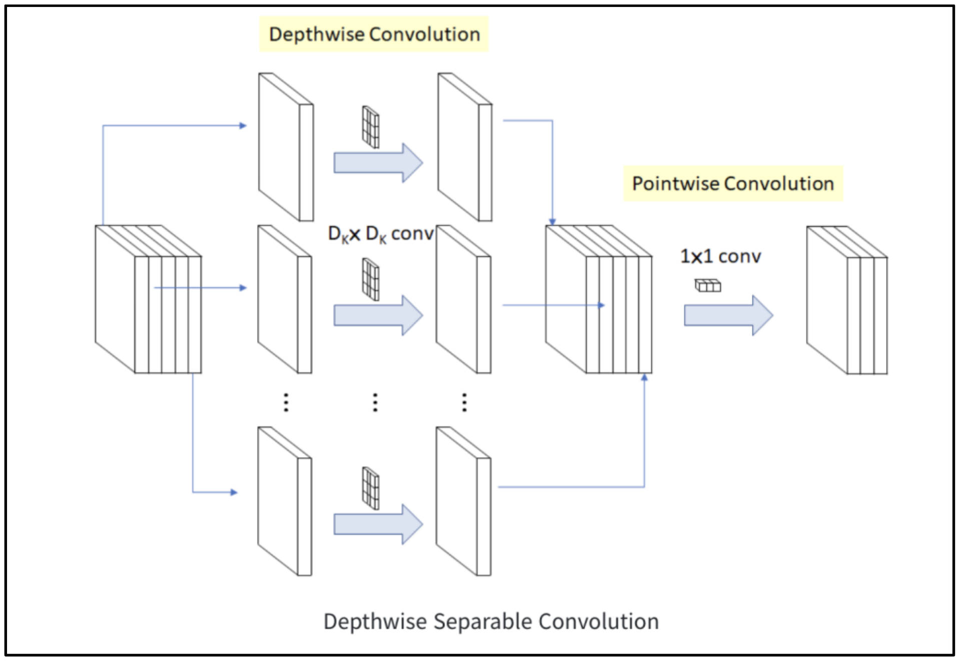


Figure 3: Depthwise Separable Convolution in MobileNet

2.4 ResNet-50

ResNet-50 is a variant of the ResNet (Residual Network) architecture, a deep convolutional neural network. It is a powerful image classification model which uses residual connections to learn deeper architectures more effectively. Residual connections allow the network to learn a set of residual mappings and bypass layers for faster and more effective propagation. The ResNet architecture is divided into 4 main sections: the convolutional layer, the identity and convolutional block, and the fully connected layers.

The convolutional layers are responsible for feature extraction from the input data. The feature maps are parsed on by the convolutional layers to the max pooling layers, which reduce the spatial dimensions of the feature maps while preserving the most important features.

The identity and convolutional blocks are responsible for processing these features and then the fully connected layers perform the final classification. The output of these layers goes into the softmax activation function which performs classification using probabilistic predictions for multi-class classification. Figure 4 represents the architecture of the ResNet-50 model.

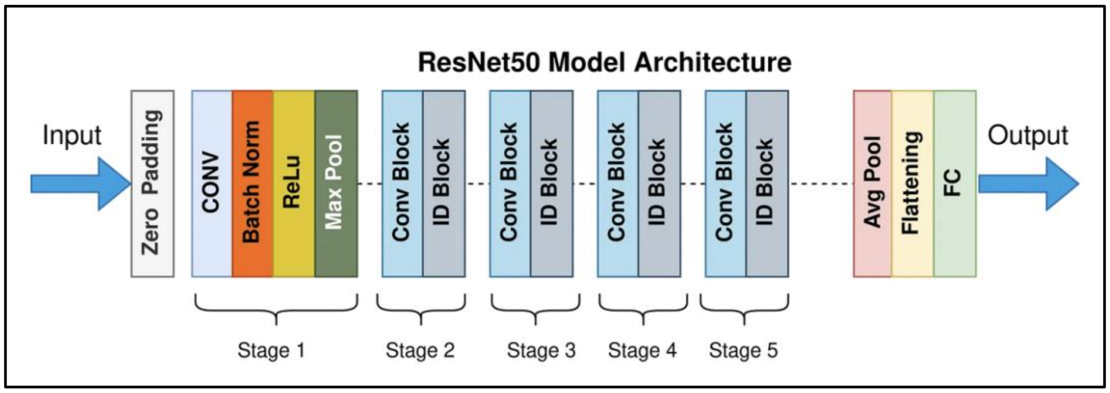


Figure 4: ResNet-50 Architecture

2.5 Inception V3

Inception V3 is a state-of-the-art convolutional neural network architecture designed for advanced image classification. This architecture provides the perfect balance between accuracy and computational efficiency, which is why it is a very popular choice for computer vision applications.

This model uses inception modules to capture multi-scale features efficiently by performing parallel convolutions with different filter sizes. The use of multiple filters and parallel convolution at each layer allows these modules to capture features at various levels of abstraction. This means that some filters are used to capture the finer details while others are used to capture the broader information, while working in parallel with each other. The feature maps processed by each parallel convolution layer are concatenated with each other and the final output has feature maps captured using different scales and filters, therefore representing information rich data.

This model also involves various factorization techniques like performing 1X1 convolutions for dimensionality reduction before each parallel convolution path. Through this the model is able to reduce complexity and allow more efficient processing by the inception modules. Along with these layers, the architecture also has pooling layers for downsizing the spatial dimensions of feature maps. Figure 5 represents the architecture of Inception V3.

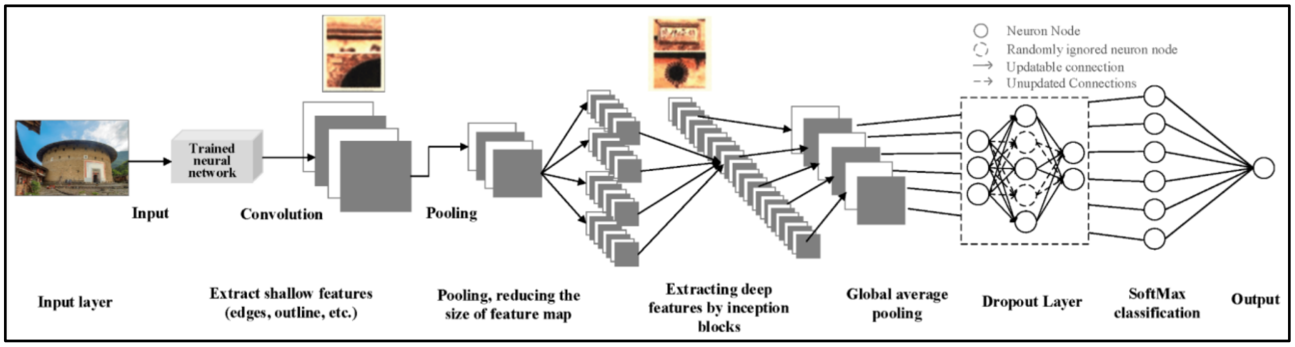


Figure 5: Inception V3 Architecture

2.6 YOLO

YOLO (You Only Look Once) is a state-of-the-art object detection algorithm known for its incredible speed and accuracy with which it classifies real time objects in images. The reason why YOLO is much quicker than other object detection algorithms is because it performs object detection in a single pass through an image rather than the multi-stage approach that traditional methods take. In a single pass through a neural network, this model predicts the bounding boxes and class probabilities from the entire image.

The learning approach that YOLO takes is that it divides the input image into a grid of cells and then predicts the bounding boxes and class probabilities for the objects that lie inside each cell. This network outputs a probability distribution over all possible classes for each bounding box.

After all predictions are made for all cells, this architecture uses a post processing method called non-max suppression (NMS) to remove redundant or overlapping bounding boxes. This is done by NMS by selecting the most confident bounding boxes based on the predictions made and the rest of the overlapping boxes are removed to avoid redundancy.

Figure 6 shows the architecture of YOLO, showing all the convolutional layers followed by the bounding boxes and class probability maps. Combining these 2 give the final detection results

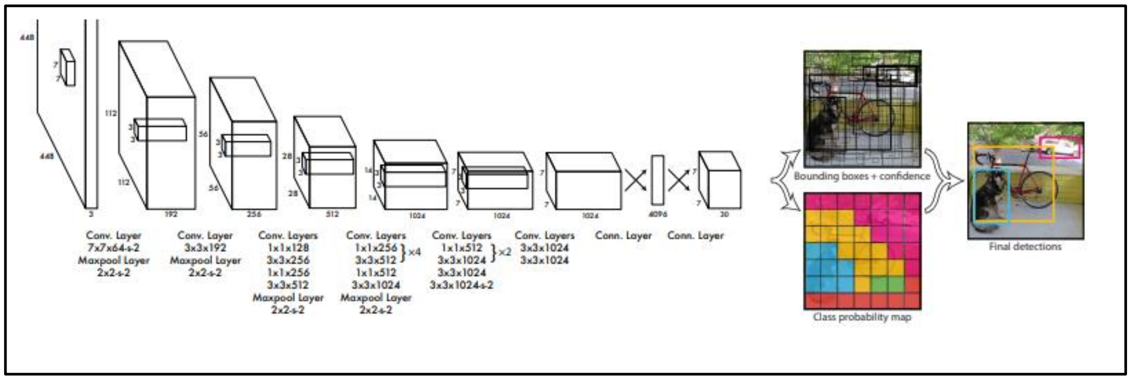

Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 ×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 ×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer
Bounding boxes + confidence
Class probability map
Final detections

Figure 6: YOLO Architecture

3. Procedure

3.1 DCNN

The development of the DCNN architecture was done with the use of various python libraries like "os" for file operations, "librosa" for audio processing, "numpy" for numerical computations, "tensorflow" and "keras" for machine learning and "matplotlib.pyplot" for data visualization.

After all the libraries were specified, the preprocessing task of converting the audio files to spectrograms was performed. For this task all the parameters for the spectrograms and their labels were declared. Each audio file was read, and its corresponding spectrogram was created using Short Term Fourier Transform (STFT), followed by truncation or padding to ensure a fixed length for each spectrogram.

Once the input data was converted into spectrograms, the DCNN architecture was defined and the convolutional layers (3 in total) along with ReLU (Rectified Linear Unit) activation unit, and the max pooling layers were introduced. Finally, the output of the max pooling layer was parsed onto the dense layers with softmax activation function for accurate classification. Therefore, the model was designed and the Adam optimizer was used for compiling.

The data was then split into training and validation sets, where 80% of data was reserved for training and 20% for validation, followed by training being done for 32 batch size and 20 epochs. Once initial training was done, the code was optimized and a smaller learning rate of 0.0001 was defined to retrain the data.

Once the training was done the accuracy vs epoch graphs are plotted, showing the model's performance on the validation set. The confusion matrix was also generated to further visualize DCNN's classification on the validation set.

3.2 MobileNet

The development of the MobileNet architecture also involves the use of all the same libraries as the ones used for DCNN architecture development. Once all the libraries were loaded in, the preprocessing of the raw audio files was done by converting them to spectrograms. This was done by using Short Term Fourier Transform (STFT), followed by truncation, or padding to ensure a fixed length for each spectrogram, also similar to DCNN design. Once the preprocessing was completed, it then splits the preprocessed data (spectrograms) into training and validation sets (80% for training and 20% for validation)
The MobileNet architecture was then loaded using the Keras library and the different convolutional, pooling, and dense layers were added. The first dense layer had 128 units with ReLU activation, and the final one had 10 units with softmax activation for multi class classification (all 10 digits). The model was then compiled with the Adam optimizer and the training was done for 20 epochs with the validation being performed based on the validation set of the model. The results of training and validation were then represented on a graph, showing the accuracy of the model with increasing epochs.
Finally, the confusion matrix was also generated to represent the performance of the model for each class (each digit in our case). Different classes were shown, with true positives (along the diagonal), false positives, true negatives, and false negatives.

3.3 ResNet-50

For the development of the ResNet-50 architecture, all similar libraries were used for loading audio files, processing them, and then performing machine learning operations. The spectrograms were then generated using short term Fourier transform and padding was done to ensure fixed length for each spectrogram. Once the preprocessing was performed, the spectrograms were then split into training and validations sets (80% for training and 20% for validation).
The architecture for ResNet-50 was loaded from the Keras library to include the convolutional, pooling, and dense layers. Then the model was compiled using the Adam optimizer and trained on the training set for 20 epochs while performing validation on the validation set. The results were then plotted on a graph to show the accuracy of the model with each increasing epoch.
Finally, the confusion matrix was generated to represent the accuracy of the model in classifying each class (digit) in the validation set.

3.4 Inception V3

For developing the Inception V3 architecture, all the same libraries for loading audio files, array manipulation, deep learning and preprocessing of data were included. Similar to previous architectures, the audio files were loaded and converted to spectrograms using short term fourier transform, followed by padding or truncating them to ensure fixed length for each spectrogram.

The data was then split into training and validations sets, with 80% data being reserved for training and 20% for validation.
The Keras library was then used to load in the pretrained Inception V3 model, along with the convolutional, pooling, and dense layers. After the model was loaded, it was compiled using the Adam optimizer and trained on the training set for 20 epochs while performing validation on the validation set. The results were then plotted on a graph to show the accuracy of the model with each increasing epoch. The confusion matrix was then generated to represent the accuracy of the model in classifying each digit in the validation set.

4. Analysis and Discussion of Results

4.1 DCNN

For the custom DCNN model, two experiments were performed. One with the Adam optimizer and the other with a custom learning rate. Both models will be trained with a batch size of 32 and 20 epochs.
Figure 7 contains an Accuracy vs Epoch graph for the custom DCNN model created with the Adam optimizer. After 3 epochs, the training and validation accuracy are around the 90% mark. After 6 epochs, the training and validation accuracy stabilize near the 95% range. The final validation accuracy is 95% for the custom DCNN model using the Adam optimizer.

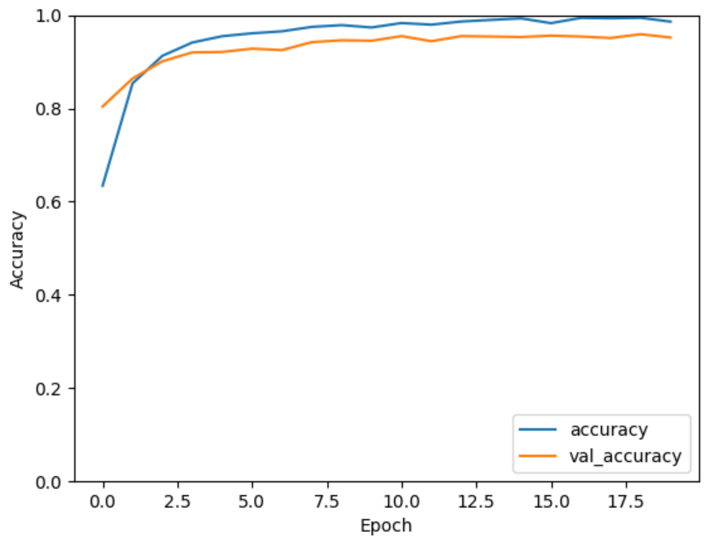


Figure 7: Accuracy vs Epoch graph for DCNN using Adam optimizer

In Figure 8, the Accuracy vs Epoch graph for the custom DCNN model with a 0.0001 learning rate. The smaller learning rate tells the model to be more thorough with the dataset in each epoch. The training accuracy and validation accuracy is above 90% after the first epoch and states that way.

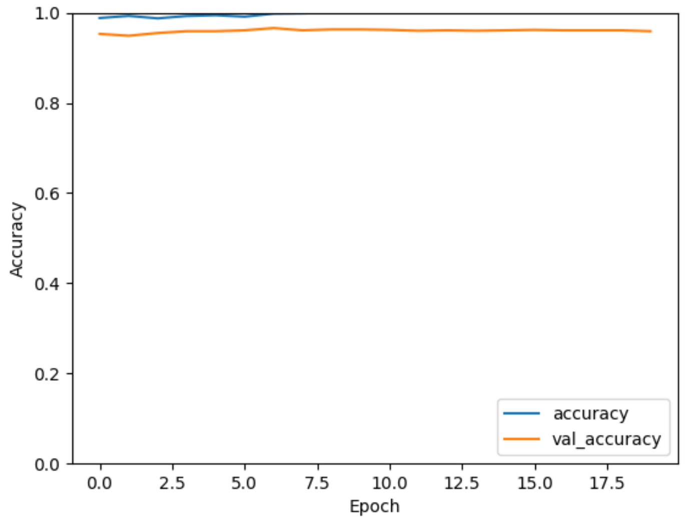
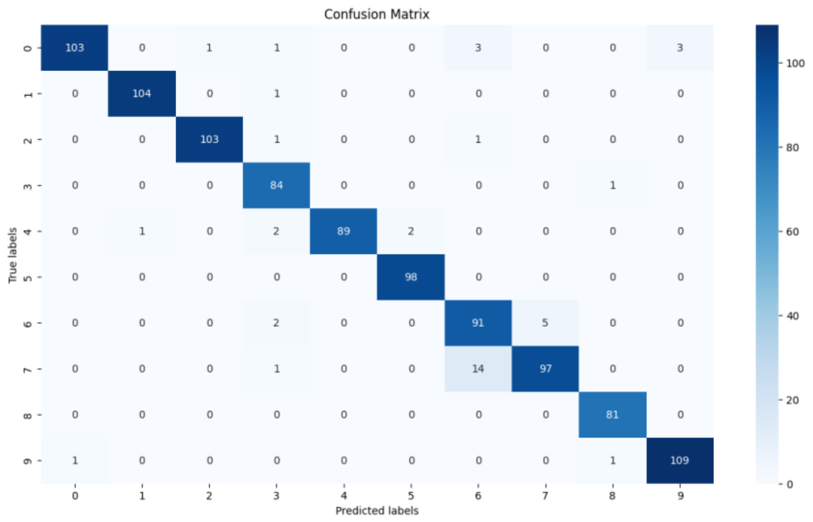


Figure 8: Accuracy vs Epoch graph for DCNN using smaller learning rate

Figure 9 contains the confusion matrix of the DCNN model with the smaller learning rate. The diagonal line of the matrix shows which classifications were predicted correctly. All classifications had at least above 80% accuracy. The worst performing class was the spoken digit 8, with an accuracy rate 81%.



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	103	0	1	1	0	0	3	0	0	3
1	0	104	0	1	0	0	0	0	0	0
2	0	0	103	1	0	0	1	0	0	0
3	0	0	0	84	0	0	0	0	1	0
4	0	1	0	2	89	2	0	0	0	0
5	0	0	0	0	0	98	0	0	0	0
6	0	0	0	2	0	0	91	5	0	0
7	0	0	0	1	0	0	14	97	0	0
8	0	0	0	0	0	0	0	0	81	0
9	1	0	0	0	0	0	0	0	1	109

Figure 9: Confusion matrix of DCNN model

4.2 MobileNet

For the low-end model, MobileNet will be the model. The pretrained MobileNet model from tensorflow with the Adam optimizer will be used. One experiment will be conducted training the model using Mobile Net. The model will be trained with a batch size of 32 and 20 epochs.

Figure 10 is a plot of the Accuracy vs Epoch graph. After 5 epochs, the training and validation accuracy is above 85%. After 8 epochs, the accuracies stabilize out around the 85% mark. The final accuracy rate for the MobileNet model is 93%.

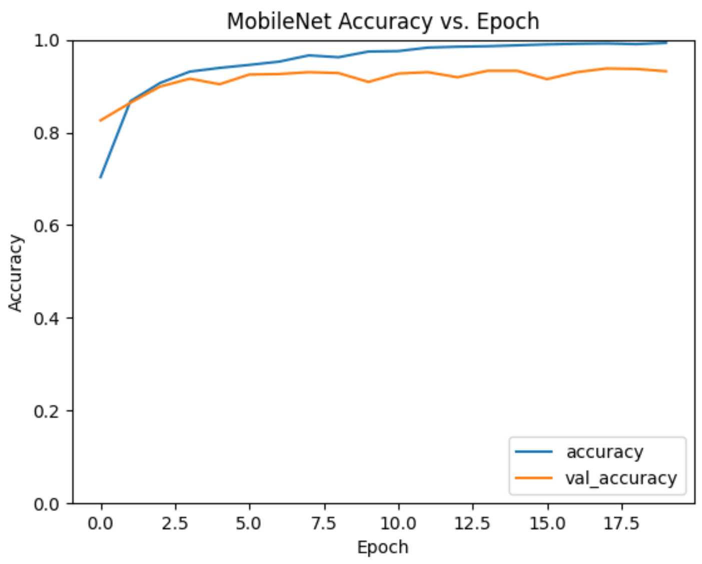
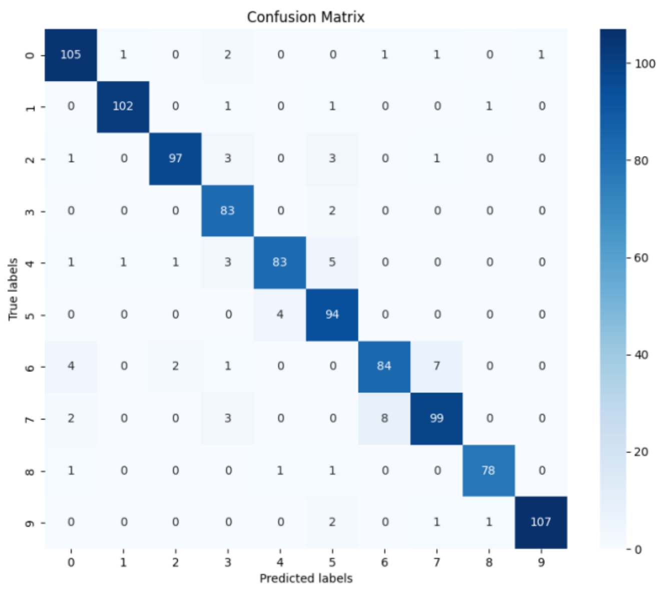


Figure 10: Accuracy vs Epoch graph for MobileNet

Figure 11 contains the confusion matrix of the MobileNet model. The diagonal line of the matrix shows which classifications were predicted correctly. All classifications had at least above 75% accuracy. The worst performing class was the spoken digit 8, with an accuracy rate 78%.



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	105	1	0	2	0	0	1	1	0	1
1	0	102	0	1	0	1	0	0	1	0
2	1	0	97	3	0	3	0	1	0	0
3	0	0	0	83	0	2	0	0	0	0
4	1	1	1	3	83	5	0	0	0	0
5	0	0	0	0	4	94	0	0	0	0
6	4	0	2	1	0	0	84	7	0	0
7	2	0	0	3	0	0	8	99	0	0
8	1	0	0	0	1	1	0	0	78	0
9	0	0	0	0	0	2	0	1	1	107

Figure 11: Confusion matrix of MobileNet model

4.3 Resnet-50

For the mid-tier model, Resnet50 will be the model. The pretrained ResNet-50 model from tensorflow with the Adam optimizer will be used. One experiment will be conducted training the model using Mobile Net. The model will be trained with a batch size of 32 and 20 epochs.

Figure 12 contains the Accuracy vs Epoch plot for the ResNet-50 training model. After 3 epochs, the training accuracy is high, above 90%. The validation accuracy is consistently below 20% until 7 epochs. From there, the validation accuracy is volatile. The reason for this could be overfitting, where the model tries to train the data too closely. This makes sense as Resnet50 is a mid-tier model. At the end, the final validation accuracy is 93%.

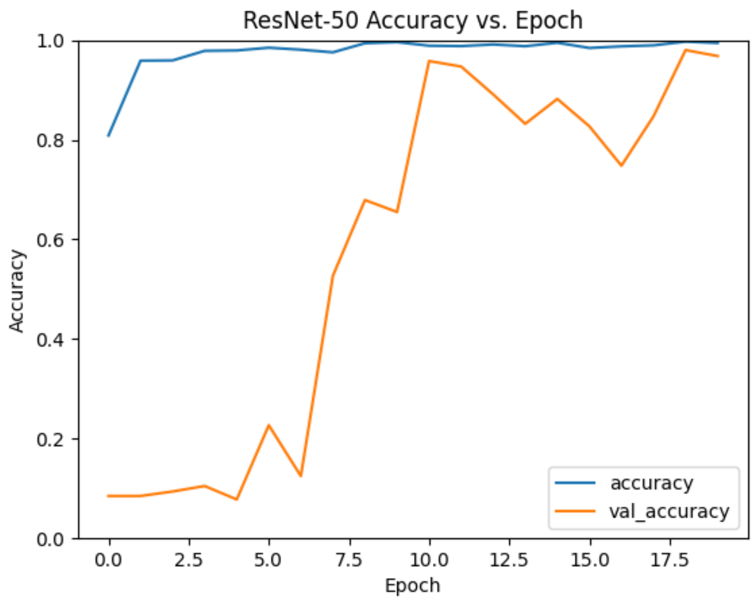


Figure 12: Accuracy vs Epoch graph for ResNet-50

Figure 13 contains the confusion matrix of the ResNe-t50 model. The diagonal line of the matrix shows which classifications were predicted correctly. All classifications had at least above 75% accuracy. The worst performing class was the spoken digit 8, with an accuracy rate 79%. The reason why the validation accuracy in Figure 4.6 is volatile but the confusion matrix has good results classifying individual samples but difficulties in general performance.

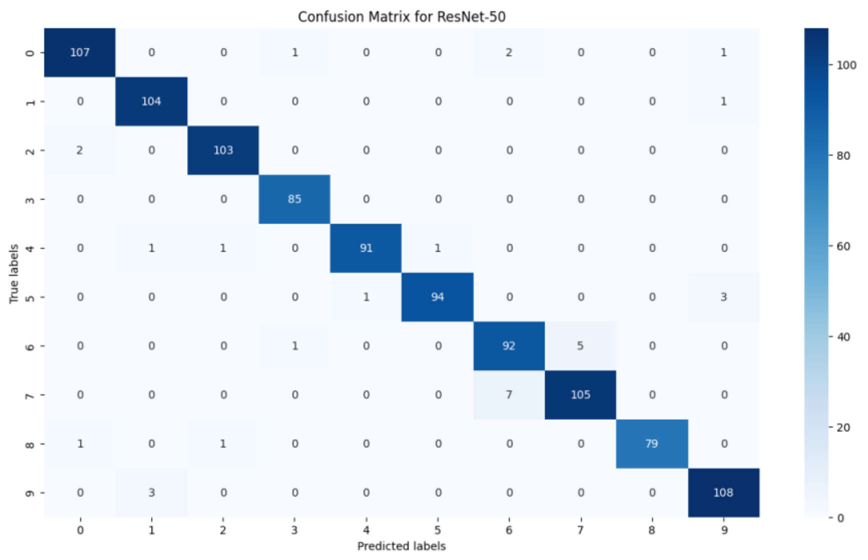


Figure 13: Confusion matrix of ResNet-50 model

4.4 InceptionV3

For the high-end model, InceptionV3 is the model. The pretrained InceptionV3 model from tensorflow with the Adam optimizer will be used. One experiment will be conducted training the model using Mobile Net. The model will be trained with a batch size of 32 and 20 epochs.

Figure 14 contains the Accuracy vs Epoch plot for the InceptionV3 training model Since Inception V3 is a high-end model, the validation accuracy is volatile. Up until 5 epochs, the validation accuracy is around 60%. The training accuracy is just above around 70%. For both the training and volatile accuracy were trending upward. The final validation accuracy is 90%.

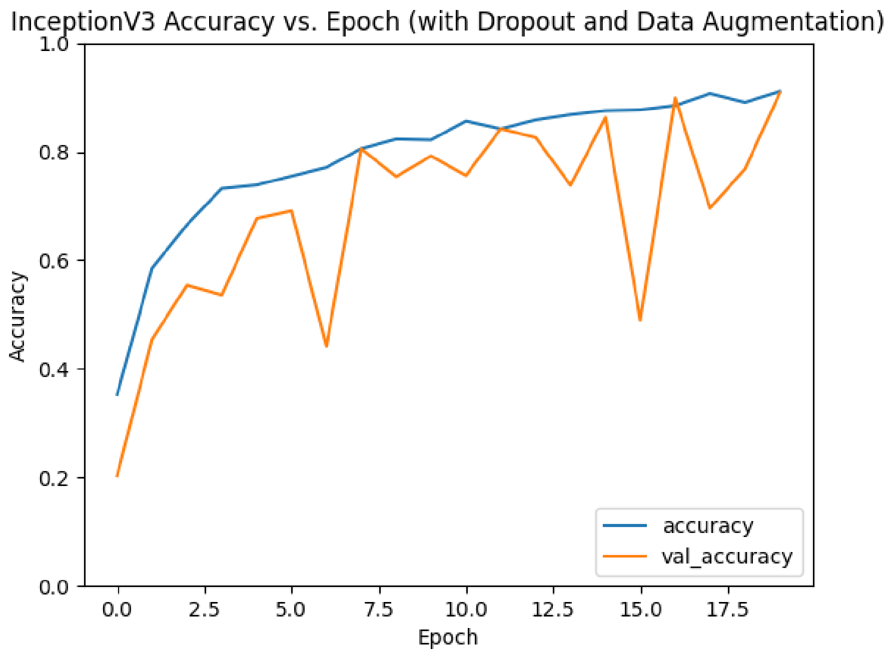


Figure 14: Accuracy vs Epoch graph for InceptionV3

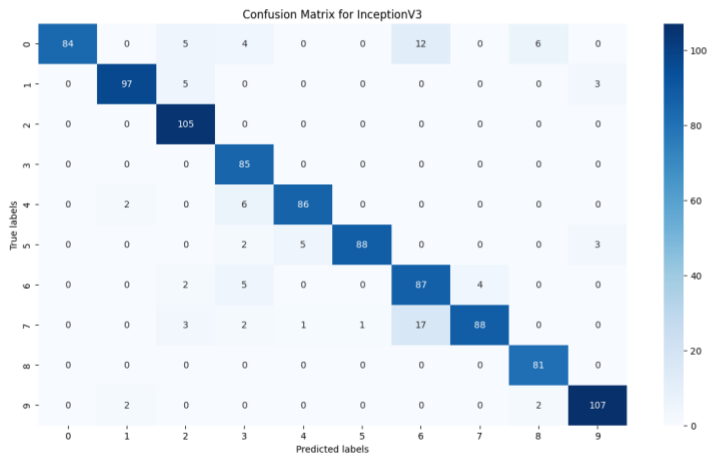


Figure 15: Confusion matrix of InceptionV3 model

Figure 15 contains the confusion matrix of the InceptionV3 model. The diagonal line of the matrix shows which classifications were predicted correctly. All classifications had at least above 80% accuracy. Most classifications hover around the 80% range. The worst performing class was the spoken digit 8, with an accuracy rate 81%. One interesting observation about this confusion matrix is that spoken digit 7 was misclassified as spoken digit 6, 17 times.

4.5 YOLO

Results for YOLO were not achievable with the current hardware that was used to complete the training for the other models. Due to classifying .wav files, the .wav files need to be converted into spectrograms; the spectrograms would then be converted into a picture format. Even if the .wav file might be the same size, when they are converted into images, they are not the same dimensions.

For future work, the team would like to convert all .wav files into spectrograms, and the spectrograms will be converted to an image file. Once all the images are gathered, the largest image would be found and that would be the set size. All other images would be filled with black boxes. The team does not know how the accuracy will be affected; images of the same class will have different amounts of black boxes.

4.6 Analyzing all classification models

Table 1 contains the Validation accuracy and time to train for each classification model. The best performing classification model is the DCNN with a validation accuracy of 95%. The worst performing classification is InceptionV3 with a validation accuracy of 90%.

The classification model that took the shortest to train is DCNN, only taking 10 minutes. The classification model that took the longest is InceptionV3, taking 3.45 hours to train. The time to train for all the algorithms falls in line with the number of layers each has. DCNN only had 3 layers. Each algorithm has more layers then the last so the time to train is longer.

Classification Model	Validation accuracy	Time to train
DCNN	95%	10 minutes
MobileNet	93%	10.66 minutes
ResNet-50	93%	2.22 hours
InceptionV3	90%	3.45 hours

Table 1: Validation accuracy and time to train of classification models

5. Conclusion

In conclusion, this project was able to provide us with great insight into how spoken digits in audio files are classified using different deep neural networks like DCNN, MobileNet, ResNet-50, Inception V3 and YOLO. Along with learning about these models and their implementation, we were also able to understand which model works best for our dataset and gives the most accuracy and which model does not work well with spectrograms.

6. Appendix

6.1 Python Code for DCNN

```
import os
import numpy as np
import librosa
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Rescaling, Conv2D, MaxPool2D, Flatten, Dropout,
Dense, Activation


# Step 1: Data Preparation
data_dir = 'C:/1/ECE 172/Project/Project 2/Data'
spectrograms = []
labels = []
# parameters for spectrogram computation

window_size = 256  # Window size for spectrogram computation
overlap = 128  # Number of samples overlap between segments
n_fft = 512  # Number of points used in the FFT
fixed_length = 128  # Define a fixed length for spectrograms

# Loop through each folder (digit) in the data directory
digit_folders = [folder for folder in os.listdir(data_dir) if
os.path.isdir(os.path.join(data_dir, folder))]
for digit_folder in digit_folders:
    digit_path = os.path.join(data_dir, digit_folder)
    # Loop through each audio file in the digit folder
    audio_files = [file for file in os.listdir(digit_path) if file.endswith('.wav')]
    for audio_file in audio_files:
        audio_path = os.path.join(digit_path, audio_file)
        # Read audio file
        audio, fs = librosa.load(audio_path, sr=48000)
        # Compute spectrogram and take absolute value
        S = np.abs(librosa.stft(audio, n_fft=n_fft, hop_length=overlap,
win_length=window_size))

        # Ensure fixed length by padding or truncating
        if S.shape[1] < fixed_length:
            pad_width = fixed_length - S.shape[1]
            S = np.pad(S, ((0, 0), (0, pad_width)), mode='constant',
constant_values=0)
        elif S.shape[1] > fixed_length:
            S = S[:, :fixed_length]

        spectrograms.append(S)
        labels.append(int(digit_folder))

spectrograms = np.array(spectrograms)
```

```
labels = np.array(labels)

print('Size of spectrograms array:', spectrograms.shape)
print('Size of labels array:', labels.shape)

# Step 2: Model Architecture

# Define the input shape
input_shape = spectrograms[0].shape

model = Sequential()
model.add(Conv2D(filters=8, kernel_size=(20, 20), activation='relu',
input_shape=input_shape))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters=16, kernel_size=(10, 10), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(filters=32, kernel_size=(5, 5), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(units=10, activation='softmax'))

# Step 3: Training

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Display the model summary
model.summary()

X_train, X_val, y_train, y_val = train_test_split(spectrograms, labels, test_size=0.2,
random_state=42)

# Reshape the input data to match the model's input shape
X_train = X_train.reshape(-1, 257, 128, 1)
X_val = X_val.reshape(-1, 257, 128, 1)


# Step 4: Use at least 10 iterations per Epoch at least 20 Epochs.
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_val, y_val))

# Plot training history before optimize
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()

# Step 5: Adjust the Learning Rate and Optimizer

val_loss, val_acc = model.evaluate(X_val, y_val)
print(f'Validation Accuracy: {val_acc:.4f}')

learning_rate = 0.0001
optimizer = Adam(learning_rate=learning_rate)

model.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=20, validation_data=(X_val, y_val))
```

```
# Step 6: Graph
# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()


# Step 7: Confusion Matrix

y_pred = model.predict(X_val)
y_pred_classes = np.argmax(y_pred, axis=1)
conf_matrix = confusion_matrix(y_val, y_pred_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=range(10),
yticklabels=range(10))
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

6.2 Python Code for MobileNet

```
import os
import numpy as np
import librosa
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix


# Function to load audio files from a folder and generate spectrograms
def load_and_preprocess_audio(folder_path):
    spectrograms = []
    labels = []

    # Define parameters for spectrogram computation
    window_size = 256  # Window size for spectrogram computation
    overlap = 128  # Number of samples overlap between segments
    n_fft = 512  # Number of points used in the FFT
    fixed_length = 128  # Define a fixed length for spectrograms

    # Loop through each folder (digit) in the data directory
    digit_folders = [folder for folder in os.listdir(folder_path) if
os.path.isdir(os.path.join(folder_path, folder))]

    for digit_folder in digit_folders:
        digit_path = os.path.join(folder_path, digit_folder)

        # Loop through each audio file in the digit folder
        audio_files = [file for file in os.listdir(digit_path) if
file.endswith('.wav')]

        for audio_file in audio_files:
            audio_path = os.path.join(digit_path, audio_file)

        # Read audio file
            audio, fs = librosa.load(audio_path, sr=48000)

            # Compute spectrogram and take absolute value
            S = np.abs(librosa.stft(audio, n_fft=n_fft, hop_length=overlap, 
win_length=window_size))

            # Ensure fixed length by padding or truncating
            if S.shape[1] < fixed_length:
                pad_width = fixed_length - S.shape[1]
                S = np.pad(S, ((0, 0), (0, pad_width)), mode='constant', 
constant_values=0)
        elif S.shape[1] > fixed_length:
                S = S[:, :fixed_length]

            # Convert to RGB image by duplicating the single channel across three 
channels
            S_rgb = np.stack((S,) * 3, axis=-1)

            spectrograms.append(S_rgb)
            labels.append(int(digit_folder))

    spectrograms = np.array(spectrograms)
    labels = np.array(labels)

    return spectrograms, labels


# Load audio data and preprocess
data_dir = 'C:/1/ECE 172/Project/Project 2/Data'
spectrograms, labels = load_and_preprocess_audio(data_dir)

input_shape = spectrograms[0].shape

X_train, X_val, y_train, y_val = train_test_split(spectrograms, labels, test_size=0.2, 
random_state=42)


def create_mobilenet_model(input_shape):
    base_model = tf.keras.applications.MobileNet(include_top=False, 
input_shape=input_shape, weights='imagenet')
    base_model.trainable = False  # Freeze the MobileNet base layers

    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dense(128, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])

    return model


model = create_mobilenet_model(input_shape)

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=20, validation_data=(X_val, y_val))
```

```
# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.title('MobileNet Accuracy vs. Epoch')
plt.show()

val_loss, val_acc = model.evaluate(X_val, y_val)
print(f'Validation Accuracy: {val_acc:.4f}')

y_pred = model.predict(X_val)
y_pred_classes = np.argmax(y_pred, axis=1)

# Compute confusion matrix
conf_matrix = confusion_matrix(y_val, y_pred_classes)

# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=range(10),
yticklabels=range(10))
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

6.3 Python Code for ResNet-50

```
import os
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import seaborn as sns
import librosa
from tensorflow.keras import layers, models
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.optimizers import Adam


# Function to load and preprocess audio data
def load_and_preprocess_audio(folder_path):
    spectrograms = []
    labels = []

    # Define parameters for spectrogram computation
    window_size = 256  # Window size for spectrogram computation
    overlap = 128  # Number of samples overlap between segments
    n_fft = 512  # Number of points used in the FFT
    fixed_length = 128  # Define a fixed length for spectrograms

    # Loop through each folder (digit) in the data directory
    digit_folders = [folder for folder in os.listdir(folder_path) if
os.path.isdir(os.path.join(folder_path, folder))]

    for digit_folder in digit_folders:
        digit_path = os.path.join(folder_path, digit_folder)

        # Loop through each audio file in the digit folder
        audio_files = [file for file in os.listdir(digit_path) if
file.endswith('.wav')]

        for audio_file in audio_files:
            audio_path = os.path.join(digit_path, audio_file)

            # Read audio file
            audio, fs = librosa.load(audio_path, sr=48000)

            # Compute spectrogram and take absolute value
            S = np.abs(librosa.stft(audio, n_fft=n_fft, hop_length=overlap,
win_length=window_size))

            # Ensure fixed length by padding or truncating
            if S.shape[1] < fixed_length:
                pad_width = fixed_length - S.shape[1]
                S = np.pad(S, ((0, 0), (0, pad_width)), mode='constant',
constant_values=0)
            elif S.shape[1] > fixed_length:
                S = S[:, :fixed_length]

            # Convert to RGB image by duplicating the single channel across three
channels
            S_rgb = np.stack((S,) * 3, axis=-1)

            spectrograms.append(S_rgb)
            labels.append(int(digit_folder))

    spectrograms = np.array(spectrograms)
    labels = np.array(labels)

    return spectrograms, labels


# Load and preprocess audio data
data_folder = 'C:/1/ECE 172/Project/Project 2/Data'
spectrograms, labels = load_and_preprocess_audio(data_folder)

X_train, X_val, y_train, y_val = train_test_split(spectrograms, labels, test_size=0.2,
random_state=42)

input_shape = spectrograms[0].shape


def create_resnet50_model(input_shape):
    base_model = ResNet50(include_top=False, input_shape=input_shape,
weights='imagenet')

    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dense(128, activation='relu'),
        layers.Dense(10, activation='softmax')  # Output layer for 10 classes
    ])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    return model


resnet50_model = create_resnet50_model(input_shape)
```

```
history = resnet50_model.fit(X_train, y_train, epochs=20, validation_data=(X_val,
y_val))

# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.title('ResNet-50 Accuracy vs. Epoch')
plt.show()

val_loss, val_acc = resnet50_model.evaluate(X_val, y_val)
print(f'Validation Accuracy for ResNet-50: {val_acc:.4f}')

y_pred = np.argmax(resnet50_model.predict(X_val), axis=-1)

conf_matrix = confusion_matrix(y_val, y_pred)

# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=range(10),
yticklabels=range(10))
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for ResNet-50')
plt.show()
```

6.4 Python Code for Inception V3

```
import os
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import seaborn as sns
import librosa
from tensorflow.keras import layers, models
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Function to load and preprocess audio data
def load_and_preprocess_audio(folder_path):
    spectrograms = []
    labels = []

    # parameters for spectrogram computation
    window_size = 256  # Window size for spectrogram computation
    overlap = 128  # Number of samples overlap between segments
    n_fft = 512  # Number of points used in the FFT
    fixed_length = 128  # Define a fixed length for spectrograms

    # Loop through each folder (digit) in the data directory
    digit_folders = [folder for folder in os.listdir(folder_path) if
os.path.isdir(os.path.join(folder_path, folder))]

    for digit_folder in digit_folders:
        digit_path = os.path.join(folder_path, digit_folder)
        # Loop through each audio file in the digit folder
        audio_files = [file for file in os.listdir(digit_path) if
file.endswith('.wav')]

        for audio_file in audio_files:
            audio_path = os.path.join(digit_path, audio_file)

            # Read audio file
            audio, fs = librosa.load(audio_path, sr=48000)

            # Compute spectrogram and take absolute value
            S = np.abs(librosa.stft(audio, n_fft=n_fft, hop_length=overlap,
win_length=window_size))

            # Ensure fixed length by padding or truncating
            if S.shape[1] < fixed_length:
                pad_width = fixed_length - S.shape[1]
                S = np.pad(S, ((0, 0), (0, pad_width)), mode='constant',
constant_values=0)
            elif S.shape[1] > fixed_length:
                S = S[:, :fixed_length]

            # Convert to RGB image by duplicating the single channel across three
channels
            S_rgb = np.stack((S,) * 3, axis=-1)

            spectrograms.append(S_rgb)
            labels.append(int(digit_folder))

    spectrograms = np.array(spectrograms)
    labels = np.array(labels)

    return spectrograms, labels


# Load and preprocess audio data
data_folder = 'C:/1/ECE 172/Project/Project 2/Data'
spectrograms, labels = load_and_preprocess_audio(data_folder)


X_train, X_val, y_train, y_val = train_test_split(spectrograms, labels, test_size=0.2,
random_state=42)

input_shape = spectrograms[0].shape # RGB spectrogram

#  dropout rate to prevent overfitting
dropout_rate = 0.5

def create_inceptionv3_model(input_shape, dropout_rate):
    base_model = InceptionV3(include_top=False, input_shape=input_shape,
weights='imagenet')

    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(dropout_rate),
        layers.Dense(10, activation='softmax')  # Output layer for 10 classes
    ])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    return model

inceptionv3_model = create_inceptionv3_model(input_shape, dropout_rate)

# Data augmentation
datagen = ImageDataGenerator(rotation_range=10, width_shift_range=0.1,
height_shift_range=0.1, shear_range=0.2, zoom_range=0.2, horizontal_flip=True)

history = inceptionv3_model.fit(datagen.flow(X_train, y_train, batch_size=32),
epochs=20, validation_data=(X_val, y_val))

# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.title('InceptionV3 Accuracy vs. Epoch (with Dropout and Data Augmentation)')
plt.show()

val_loss, val_acc = inceptionv3_model.evaluate(X_val, y_val)
print(f'Validation Accuracy for InceptionV3: {val_acc:.4f}')

y_pred = np.argmax(inceptionv3_model.predict(X_val), axis=-1)

conf_matrix = confusion_matrix(y_val, y_pred)

# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=range(10),
yticklabels=range(10))
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for InceptionV3')
plt.show()
```

6.5 Link to the Dataset Used

https://www.kaggle.com/datasets/sripaadsrinivasan/audio-mnist?resource=download