Speech recognition

Assignment 3

Hazem Mohammed Abdullah	6723
Ahmed Ashraf Abdelkarim Hussein	6940
Amr Abdel Samee Yousef	7126

The dataset

CREMA-D (Crowd-Sourced Emotional Multimodal Actors Dataset) is a large-scale dataset designed for the purpose of emotion recognition and facial expression analysis. The dataset was constructed from the performances of 91 professional actors (48 female and 43 male) who were asked to portray 12 emotional states while performing various facial expressions and vocalizations, resulting in a total of 7,442 audio-visual clips.

The dataset was compiled using crowd-sourcing, with each clip being rated by multiple human raters for the intensity of facial expressions and vocalizations of the actors. The dataset includes annotations for six basic emotions (anger, disgust, fear, happiness, sadness, and surprise), as well as seven compound emotions (amusement, contempt, contentment, embarrassment, excitement, guilt, and pride).

The audio-visual clips in the dataset are captured at a resolution of 640x480 pixels and a frame rate of 30 frames per second. The audio is recorded at 44.1 kHz in 16-bit format. The dataset is split into training and testing sets, with 80% of the clips used for training and 20% used for testing.

The CREMA-D dataset has been widely used for research in the field of emotion recognition and facial expression analysis. It has been used as a benchmark dataset for evaluating the performance of various machine learning algorithms and deep learning models. The dataset has also been used to study the influence of factors such as age, gender, and culturalbackground on emotion recognition.

One of the advantages of the CREMA-D dataset is its large size, which allows for the development and evaluation of more complex and sophisticated models. The dataset also provides annotations for both facial expressions and vocalizations, which enables the study of multimodal emotion recognition.

However, there are also some limitations to the dataset. One limitation is that the actors in the dataset are all professional actors, which may not be representative of the general population. Additionally, the dataset only includes six basic emotions and seven compound emotions, which may not capture the full range of human emotions.

Despite its limitations, the CREMA-D dataset remains a valuable resource for researchers in the field of emotion recognition and facial expression analysis. The dataset can be accessed through the official website, and the use of the dataset is subject to the terms and conditions specified by the creators.

In conclusion, the CREMA-D dataset is a large-scale dataset designed for emotion recognition and facial expression analysis. Its availability and annotations have facilitated research in the field, and it remains an important benchmark for the evaluation of machine learning algorithms and deep learning models.

Data preprocessing

Data Augmentaion

we extended the size of the data by adding noise to the data and we also added some pitch shift to the data also we stretched the time of the ausios making it slower or faster. we combined all of these data so the new data set are 7442 x 3 = 22,326.

The new data gave better results than the original data we also tried to combine two or more of the ways we augment the data to one data set so we might add noise then stretch it in the time.

functions responsible for data augmentation:

```
def add_noise(audio, noise_factor=0.01):

def pitch_shift(audio, sample_rate, n_steps=5):

def time_stretch(audio, length, factor=0.9):
```

we call all of them in

```
def extend_data(audio, sample_rate):
```

Feature Extraction

We now have a data frame whci contain the pppaths for each track and its label.

we load each audio using librosa *library* and run some functions that extract the desired features. the features we used are:

```
def zero_crossing_rate(audio):

def energy(audio):

def mfcc(audio, sample_rate):# Mel-frequency Cepstral

def chroma(audio, sample_rate):

def contrast(audio, sample_rate):

def mel_spectrogram1D(audio, sample_rate):

def mel_spectrogram2D(audio, sample_rate):
```

and all of these function are called in:

```
def extract_features(audio, sample_rate):
```

We did extract the features for two models one uses 1D conv layers and the other uses 2d conv layers so we have two feature spaces.

the first one includes:

- `zero crossing rate`
- `energy`
- `mfcc`
- `chroma`
- `contrast`
- `mel_spectrogram1D`

the second feature space includes:

- `mel_spectrogram2D`

Data Prepareing

For all the labels we used one hot encoding to transform the data to a numerical form and at testing we reversed this step and ge the categorical data again.

We splited the data to train, test, and validation data

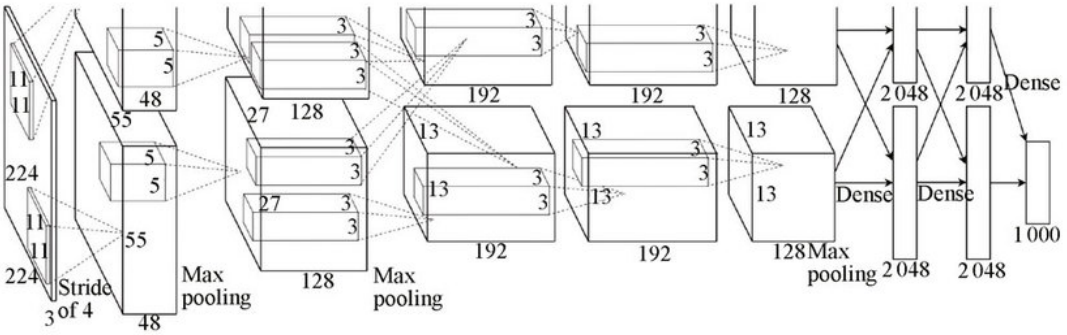
We splited the data into 70% training and validation and 30% testing. and we used 5% of the training and validation data for validation.

Architectures

1) AlexNet architecture with 1D convolutional layers

AlexNet is a convolutional neural network architecture that was introduced in 2012 and won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) that year. It was one of the first deep neural networks to achieve state-of-the-art results on this challenge and was instrumental in popularizing deep learning.

The original AlexNet architecture consists of five convolutional layers, followed by three fully connected layers. The first convolutional layer has 96 filters, the second and third convolutional layers have 256 filters each, and the fourth and fifth convolutional layers have 384 and 256 filters, respectively. The fully connected layers have 4096 neurons each, except for the last layer which has 1000 neurons corresponding to the 1000 classes in the ImageNet dataset.



While AlexNet was originally designed for 2D image data, it is possible to modify the architecture to work with 1D time series data by replacing the 2D convolutional layers with 1D convolutional layers. In this modified architecture, the input data is assumed to be a sequence of one-dimensional vectors, and the convolutional filters slide over this sequence in one dimension instead of two.

The modified AlexNet architecture with 1D convolutional layers typically consists of a series of convolutional layers followed by one or more fully connected layers. Each convolutional layer applies a set of filters to the input sequence, with the number of filters increasing as we move deeper into the network. The output of eachconvolutional layer is then passed through a non-linear activation function such as ReLU, and optionally followed by a pooling layer to reduce the spatial dimensionality of the output.

The final output of the convolutional layers is then flattened and passed through one or more fully connected layers, which perform a classification task on the input sequence. The number of neurons in the fully connected layers can be adjusted depending on the complexity of the task and the size of the input sequence.

One of the advantages of using 1D convolutional layers in the AlexNet architecture is that it allows for the processing of sequential data such as time series or audio signals. The use of 1D convolutional filters allows the network to capture local temporal patterns in the input sequence, while the use of multiple layers allows it to capture more complex temporal patterns

Another advantage of using 1D convolutional layers is that it reduces the number of parameters in the network compared to using fully connected layers. This reduces the risk of overfitting and allows the network to generalize better to new data.

However, there are also some limitations to using 1D convolutional layers. One limitation is that they may not capture long-term temporal patterns as effectively as recurrent neural networks (RNNs), which are specifically designed to handle sequential data. Additionally, 1D convolutional layers may not be as effective at capturing spatial patterns in multi-dimensional data as 2D or 3D convolutional layers.

In conclusion, the AlexNet architecturewith 1D convolutional layers is a modification of the original AlexNet architecture that allows for the processing of sequential data such as time series or audio signals. The use of 1D convolutional filters allows the network to capture local temporal patterns in the input sequence, while the use of multiple layers

allows it to capture more complex temporal patterns. While there are limitations to using 1D convolutional layers, they offer advantages such as reducing the number of parameters in the network and allowing for better generalization to new data. Overall, the modified AlexNet architecture with 1D convolutional layers is an effective tool for processing sequential data and has applications in various fields such as speech recognition, music analysis, and medical signal processing.

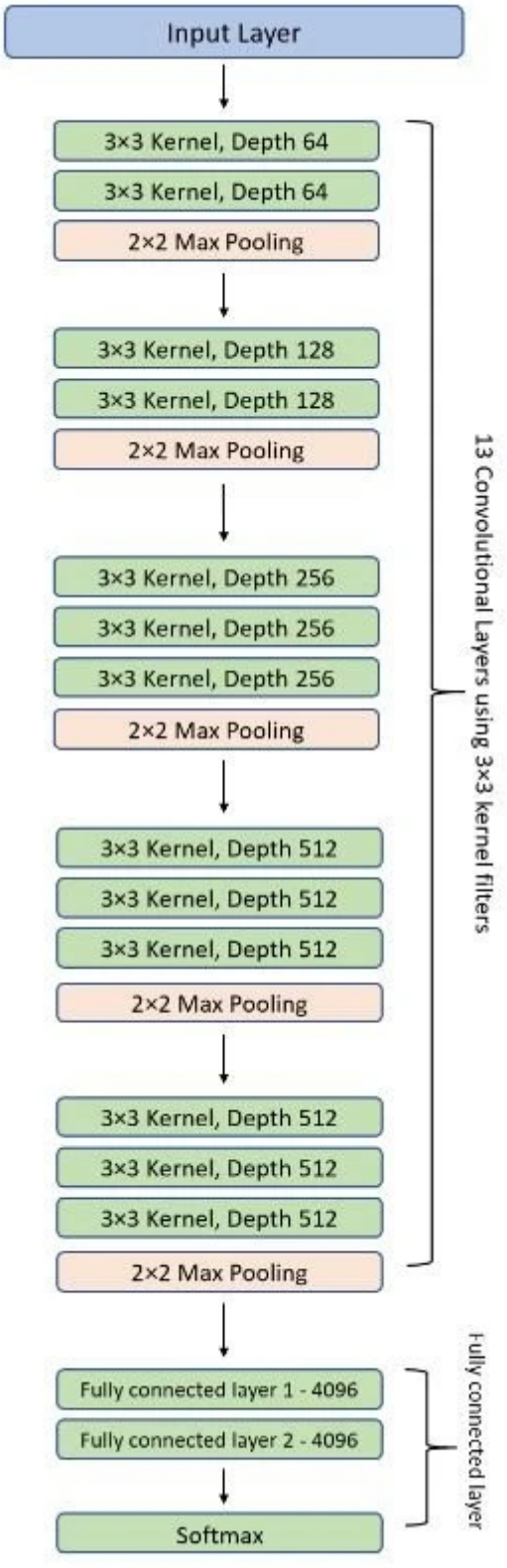
2) VGG-16:

VGG-16 is a convolutional neural network architecture that was introduced in 2014 by the Visual Geometry Group (VGG) at the University of Oxford. It is one of the most popular and influential deep learning architectures, and has been widely used for image recognition, object detection, and other computer vision tasks.

The VGG-16 architecture consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. The first 13 layers are convolutional layers, with filters of size 3x3 and a stride of 1 pixel. The number of filters in each layer increases as we move deeper into the network, from 64 in the first layer to 512 in the last layer. Each convolutional layer is followed by a rectified linear unit (ReLU) activation function and a 2x2 max pooling layer, which reduces the spatial dimensionality of the output.

The final 3 layers are fully connected layers with 4096 neurons each, followed by a softmax layer that outputs the probability distribution over the classes. The number of neurons in the fully connected layers can be adjusted depending on the complexity of the task and the size of the input images.

One of the key features of the VGG-16 architecture is its simplicity and uniformity. All convolutional layers use the same filter size and stride, and all max pooling layers use the same size and stride. This makes the architecture easy to understand and implement, and allows for easy transfer of knowledgebetween tasks and datasets.



Another advantage of the VGG-16 architecture is its high accuracy on image recognition tasks. It achieved state-of-the-art performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2014, with a top-5 error rate of 7.3%. This was a significant improvement over previous state-of-the-art models, and helped to establish deep learning as the dominant approach in computer vision.

The VGG-16 architecture has also been widely used as a feature extractor for transfer learning. By removing the fully connected layers and using the convolutional layers to extract features from images, the network can be used as a starting point for training on new datasets with limited labeled data. This has been particularly useful for tasks such as object detection and segmentation, where labeled data is often scarce.

However, there are also some limitations to the VGG-16 architecture. One limitation is its high computational cost, due to the large number of parameters in the fully connected layers. This can make it difficult to train the network on large datasets or with limited computational resources.

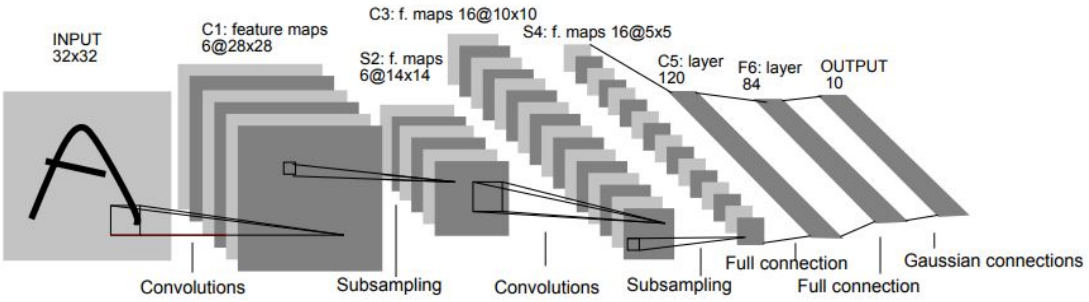
Another limitation is its lack of attention to spatial hierarchy. The architecture assumes that all features are equally important, regardless of their spatial location in the image. This can limit its ability to capture spatial relationships between objects and parts of objects.

In conclusion, the VGG-16 architecture is a highly influential and effective deep learning architecture for image recognition and other computer vision tasks. Its simplicity and uniformity make iteasy to understand and implement, and its high accuracy on image recognition tasks has made it a popular choice in the field. Its use as a feature extractor for transfer learning has also been valuable for tasks with limited labeled data. However, its high computational cost and lack of attention to spatial hierarchy are limitations that should be considered when choosing an architecture for a specific task. Overall, the VGG-16 architecture has had a significant impact on the field of deep learning and computer vision, and will likely continue to be an important benchmark for future research.

3) LeNet-5:

LeNet-5 is a convolutional neural network architecture that was introduced in 1998 by Yann LeCun and his colleagues. It was one of the first successful deep learning architectures for image recognition, and has been widely used as a benchmark and starting point for other deep learning models.

The LeNet-5 architecture was originally designed for handwritten digit recognition, and consists of 7 layers, including 2 convolutional layers, 2 subsampling layers, and 3 fully connected layers. The first convolutional layer has 6 filters of size 5x5, and the second convolutional layer has 16 filters of size 5x5. Each convolutional layer is followed by a subsampling layer, which reduces the spatial dimensionality of the output by a factor of 2.



The output of the second subsampling layer is then flattened and passed through 3 fully connected layers, with 120, 84, and 10 neurons, respectively. The final layer has 10 neurons corresponding to the 10 possible digits (0-9) in the MNIST dataset.

One of the key features of the LeNet-5 architecture is its use of convolutional and subsampling layers. The convolutional layers apply a set of filters to the input image, which allows the network to capture local spatial patterns in the image. The subsampling layers then reduce the spatial dimensionality of the output, which helps to reduce the number of parameters in the network and prevent overfitting.

Another advantage of the LeNet-5 architecture is its simplicity and efficiency. The network has relatively few parameters compared to modern deep learning architectures, which makes it easier to train and deploy on resource-constrained devices. Its use of convolutional and subsampling layers also makes it well-suited for tasks with spatially correlated inputs, such as image recognition and computer vision.

The LeNet-5 architecture has been widely used as a benchmark for evaluating the performance of other deep learning models on image recognition tasks. Its high accuracy on the MNIST dataset, which is commonly used as a benchmark for handwritten digit recognition, has made it a popular starting point for other deep learning models.

However, there are also some limitations to the LeNet-5 architecture. One limitation is its limited capacity to capture complex spatial patterns in images. The network is relatively shallow compared to modern deep learning architectures, which can limit its ability to capture high-level features and relationships between objects in images.

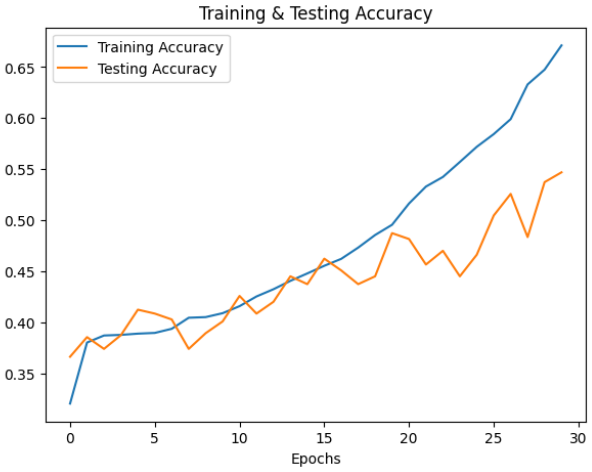
Another limitation is its lack of flexibility in handling inputs of different sizes and aspect ratios. The network was originally designed for the MNIST dataset, which has fixed-size grayscale images of 28x28 pixels. Adapting the network to handle inputs of different sizes and aspect ratios requires significant modifications to the architecture.

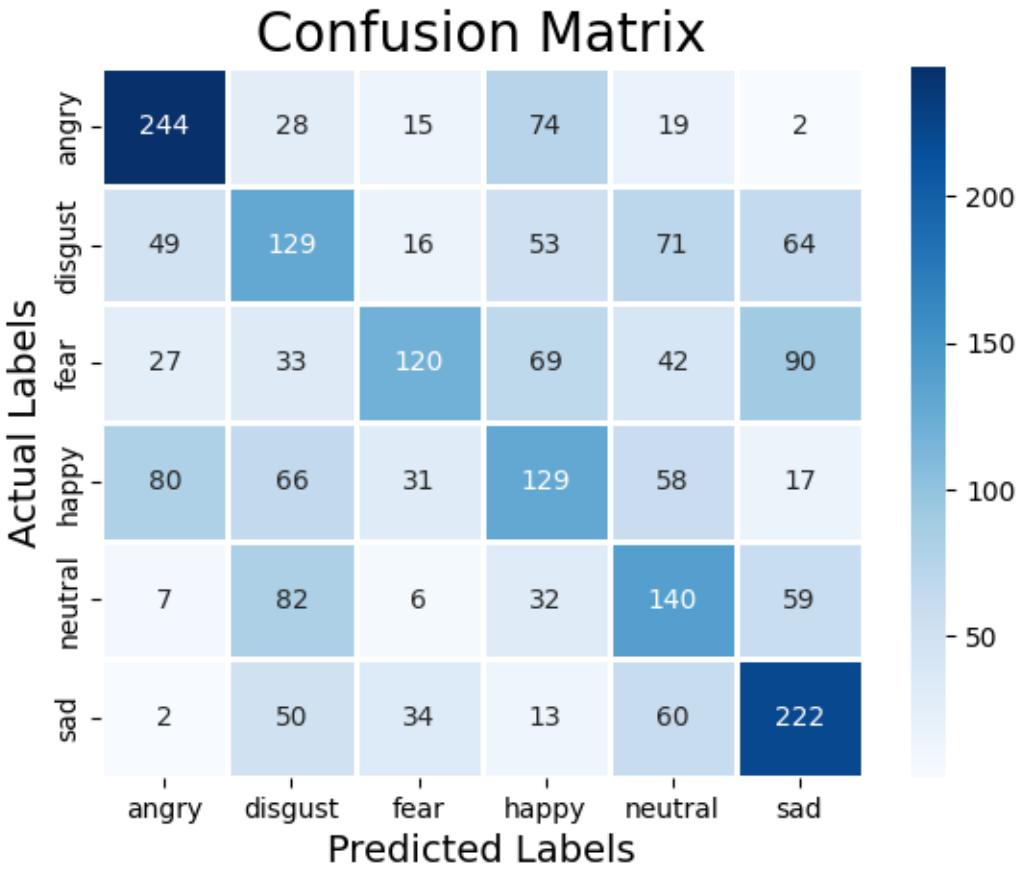
In conclusion, the LeNet-5 architecture is a seminal deep learning architecture for image recognition, and has had a significant impact on the field of deep learning. Its use of convolutional and subsampling layers has proven to beeffective in capturing local spatial patterns in images, and its simplicity and efficiency make it a popular starting point for other deep learning models. However, its limited capacity to capture complex spatial patterns and lack of flexibility in handling inputs of different sizes and aspect ratios are limitations that should be considered when choosing an architecture for a specific task. Overall, the LeNet-5 architecture has played an important role in the development of deep learning for image recognition, and its legacy can still be seen in modern deep learning architectures.

Results

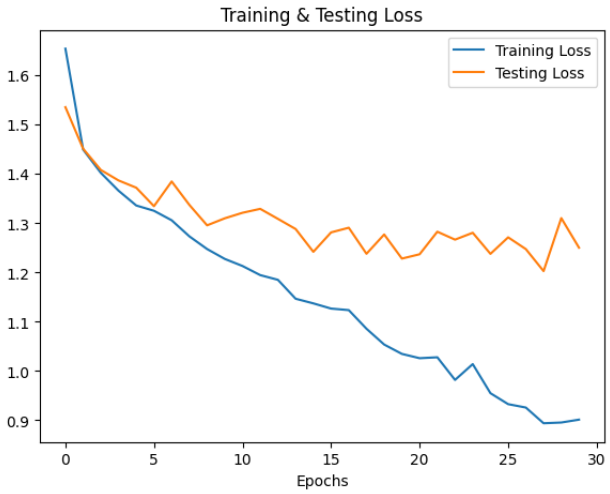
1D MODELS

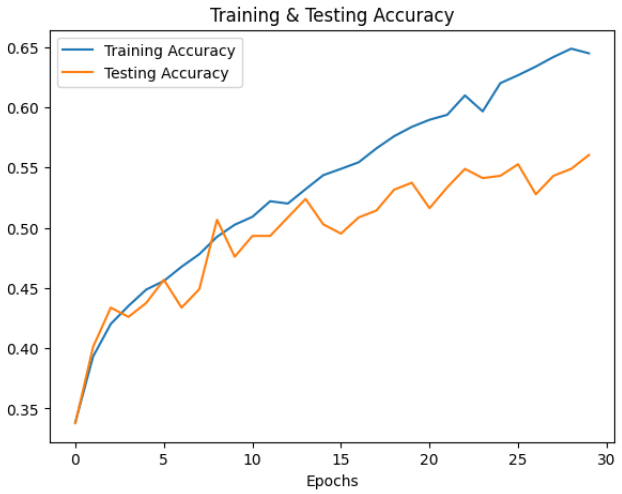
VGG16

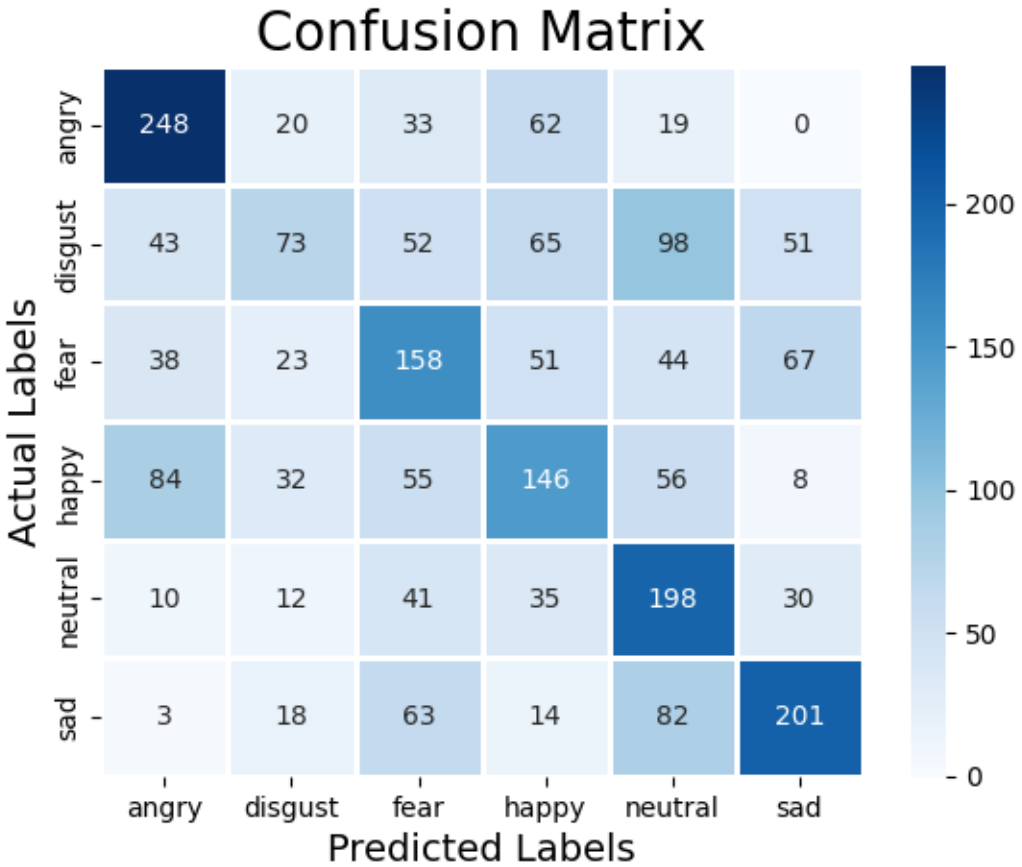




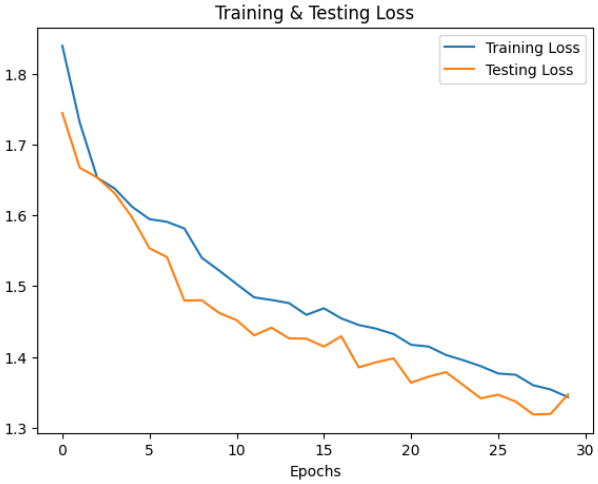
Letnet5

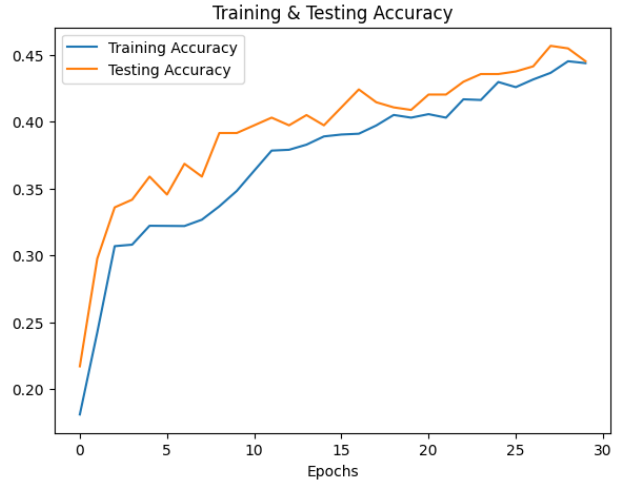

Training & Testing Loss
Training Loss
Testing Loss
Epochs

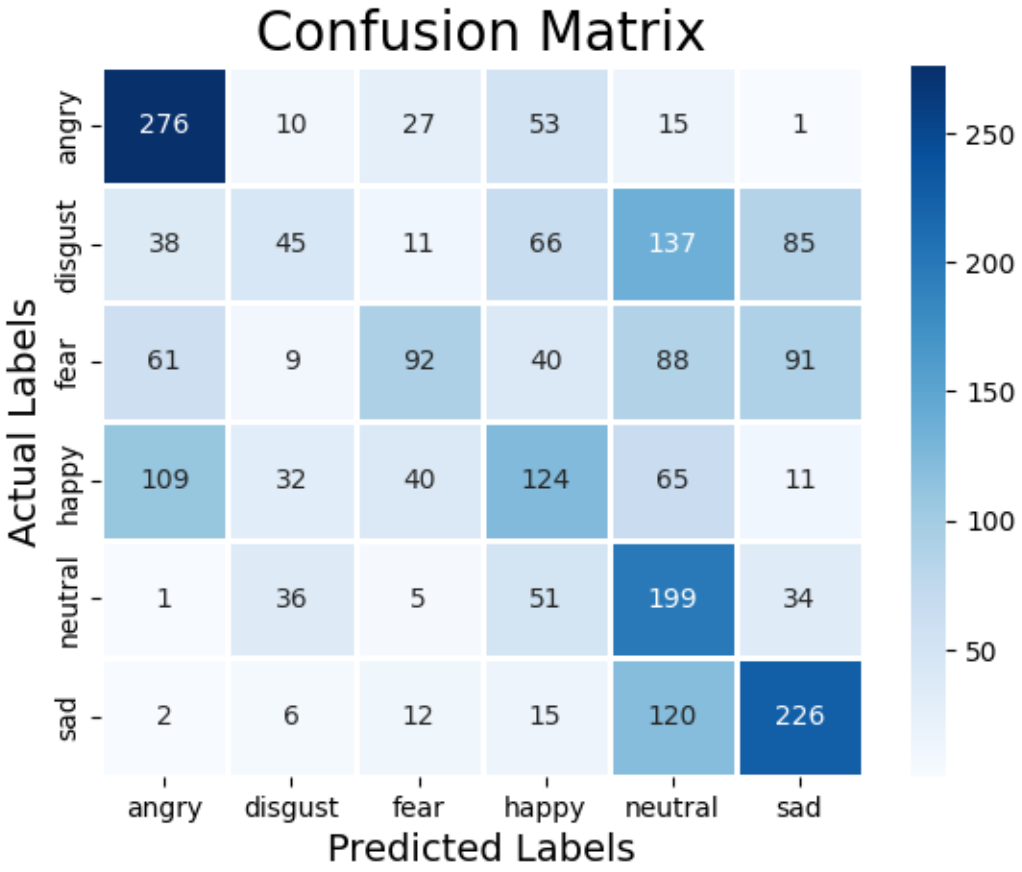

Training & Testing Accuracy
Training Accuracy
Testing Accuracy
Epochs


Confusion Matrix
Actual Labels
angry
disgust
fear
happy
neutral
sad
248 20 33 62 19 0
43 73 52 65 98 51
38 23 158 51 44 67
84 32 55 146 56 8
10 12 41 35 198 30
3 18 63 14 82 201
Predicted Labels
200
150
100
50
0

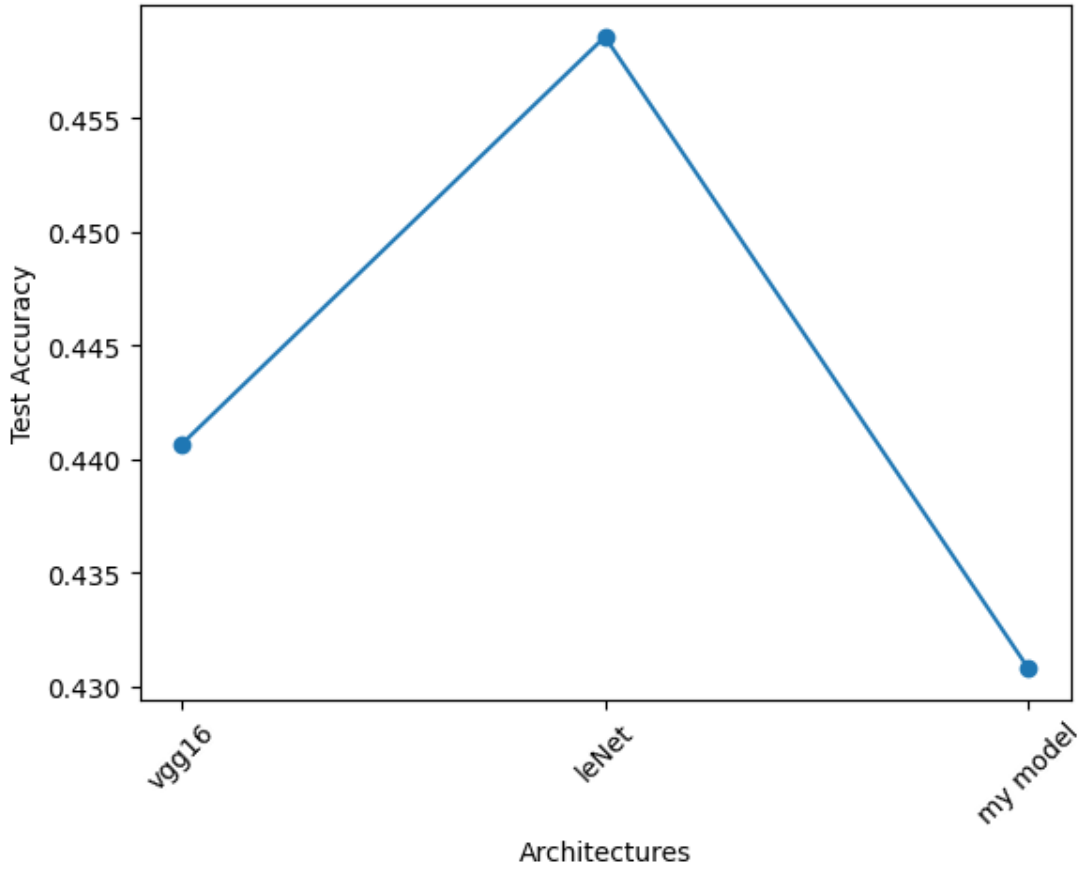
My model


Training & Testing Loss
Training Loss
Testing Loss
1.8
1.7
1.6
1.5
1.4
1.3
0
5
10
15
20
25
30
Epochs


Training & Testing Accuracy
Training Accuracy
Testing Accuracy
0.45
0.40
0.35
0.30
0.25
0.20
0
5
10
15
20
25
30
Epochs

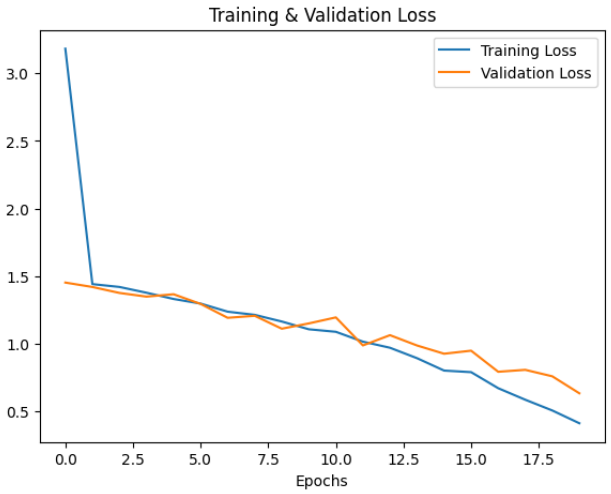

Confusion Matrix
Actual Labels
angry
disgust
fear
happy
neutral
sad
276 10 27 53 15 1
38 45 11 66 137 85
61 9 92 40 88 91
109 32 40 124 65 11
1 36 5 51 199 34
2 6 12 15 120 226
250
200
150
100
50
angry
disgust
fear
happy
neutral
sad
Predicted Labels

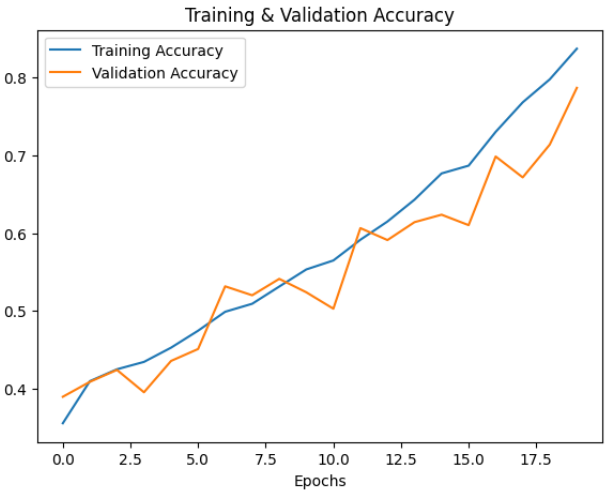
Comparisons among models:

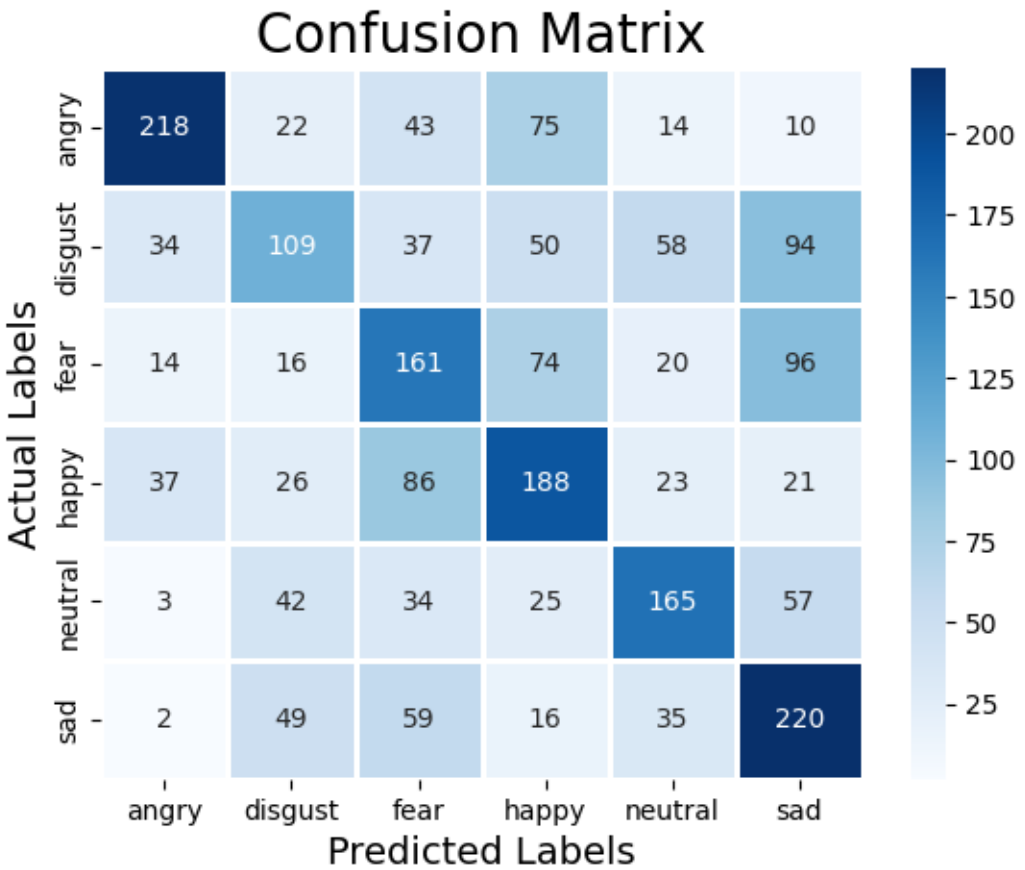


2D Models

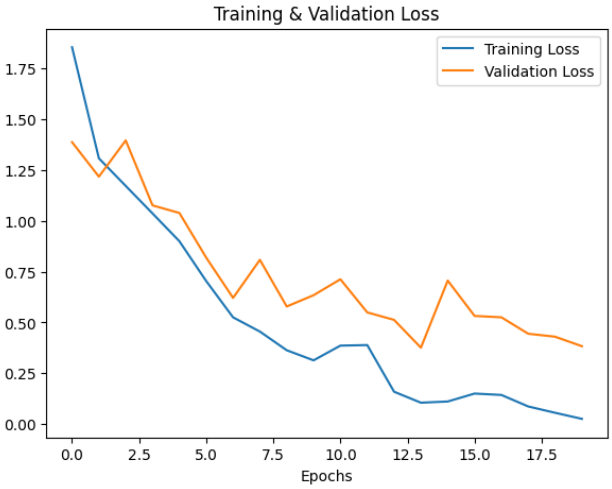
VGG16

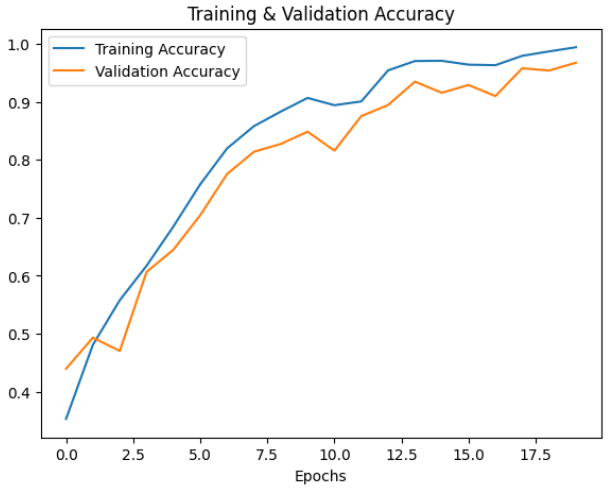


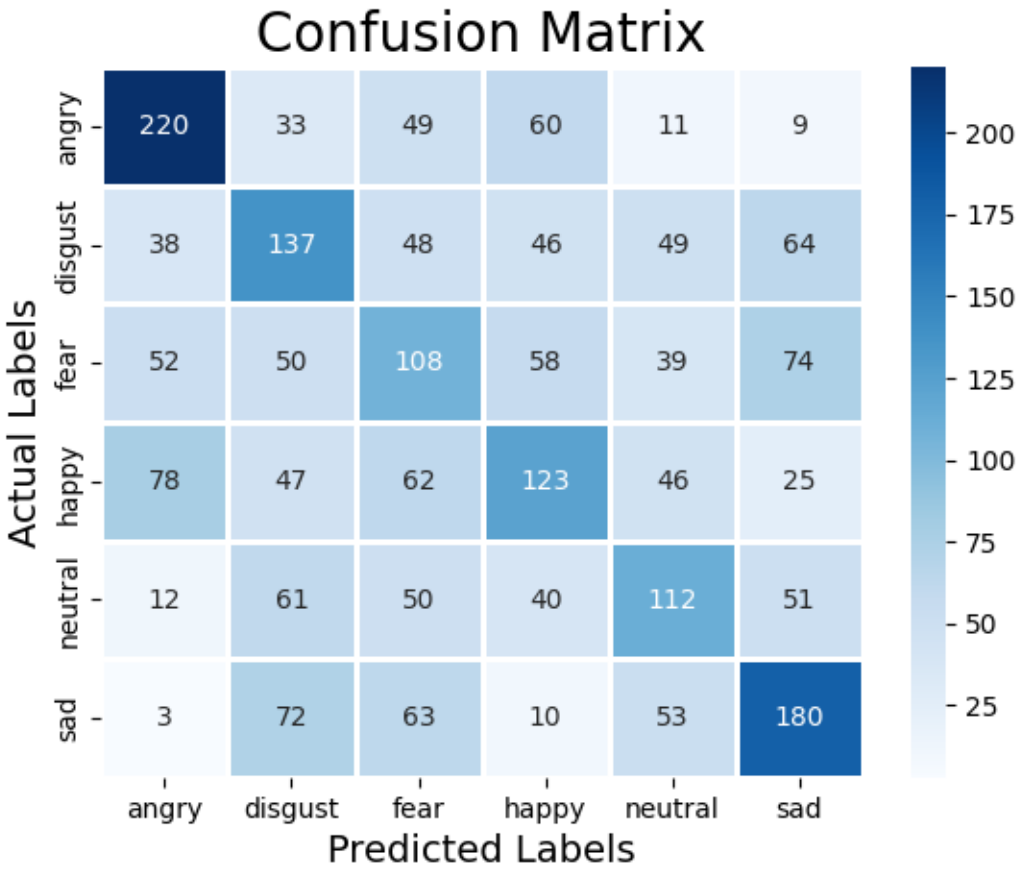




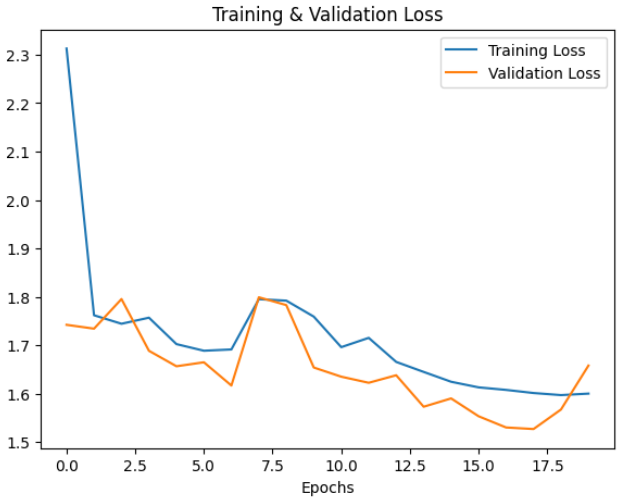
Letnet 5

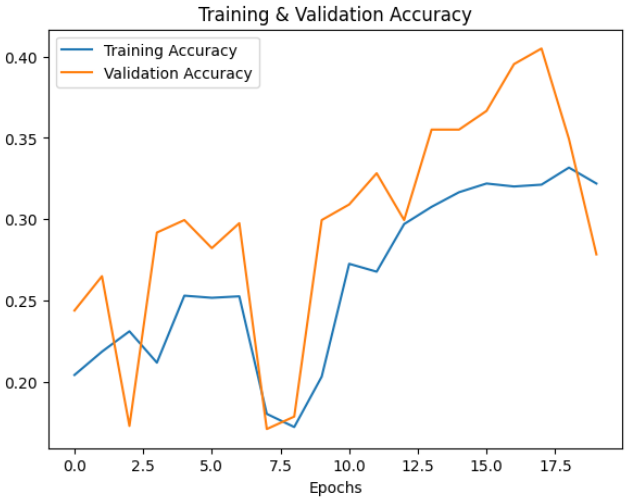

Training & Validation Loss
Training Loss
Validation Loss
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs

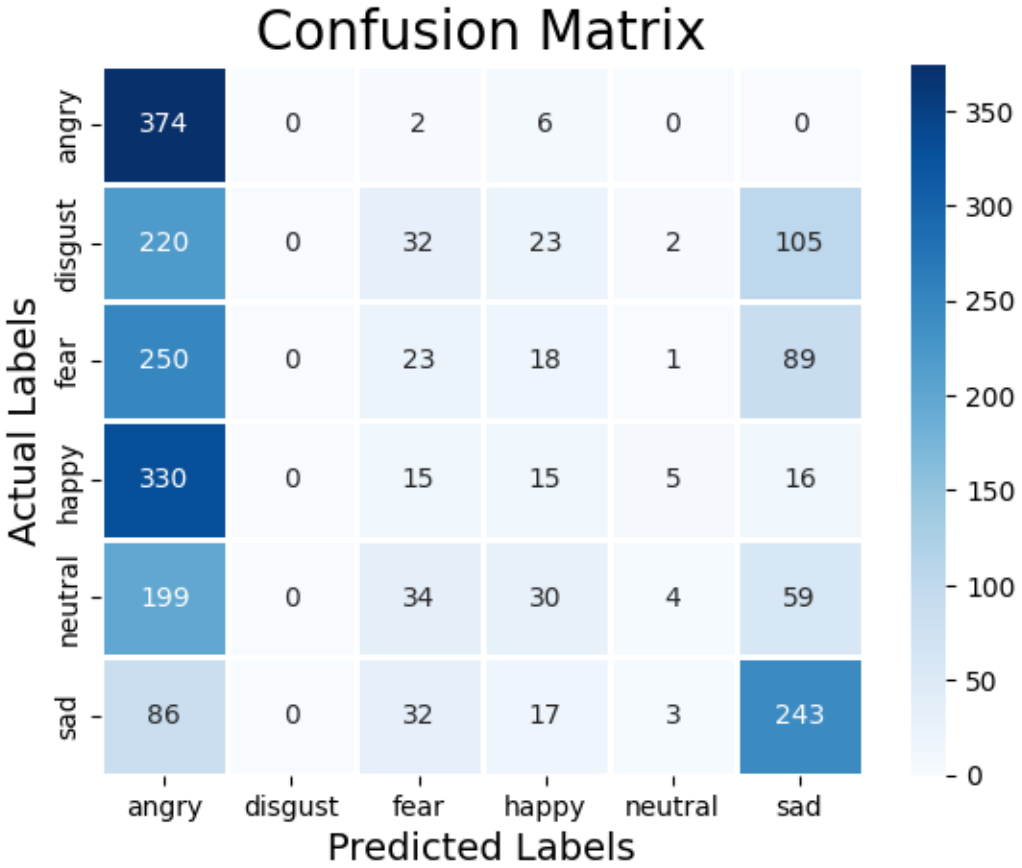

Training & Validation Accuracy
Training Accuracy
Validation Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs


Confusion Matrix
Actual Labels
angry
disgust
fear
happy
neutral
sad
220 33 49 60 11 9
38 137 48 46 49 64
52 50 108 58 39 74
78 47 62 123 46 25
12 61 50 40 112 51
3 72 63 10 53 180
Predicted Labels
200
175
150
125
100
75
50
25

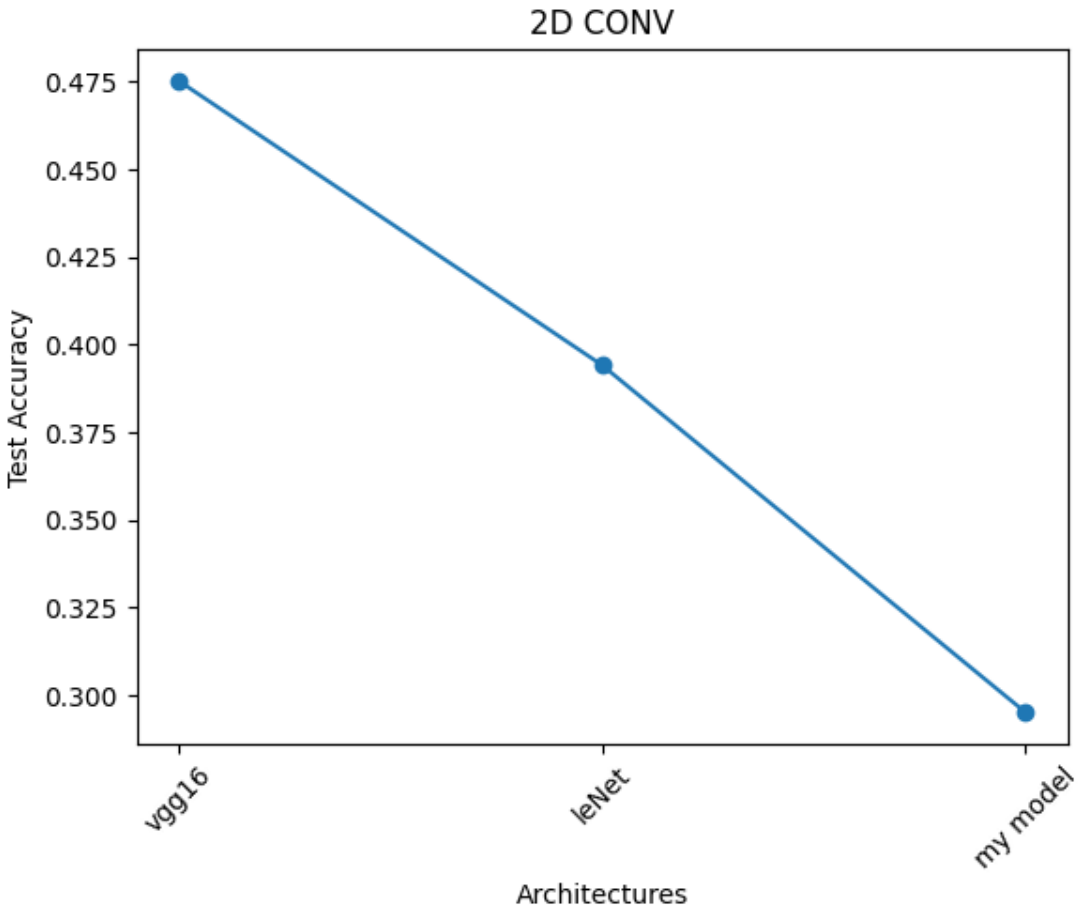
My model


Training & Validation Loss
Training Loss
Validation Loss
Epochs


Training & Validation Accuracy
Training Accuracy
Validation Accuracy
Epochs


Confusion Matrix
Actual Labels
Predicted Labels
angry
disgust
fear
happy
neutral
sad
374 0 2 6 0 0
220 0 32 23 2 105
250 0 23 18 1 89
330 0 15 15 5 16
199 0 34 30 4 59
86 0 32 17 3 243

Comparisons among models



Bonus: SepTr: Separable Transformer for Audio Spectrogram Processing

Introduction:

SepTr (Separable Transformer), an innovative architecture designed for efficient and effective processing of audio spectrograms. Spectrograms, which represent audio signals in the frequency and time domains, are widely used in various audio-related tasks such as speech recognition, music classification, and sound event detection. SepTr addresses the limitations of applying standard transformers to spectrograms by proposing a separable approach that separates the attention dedicated to each axis. This report provides an in-depth analysis of the SepTr project, including its purpose, architecture, code explanation, applications, and conclusions.

Architecture of SepTr:

The architecture of the model described in the given text is called SepTr (Separable Transformer). It is an architecture designed for audio spectrogram processing. SepTr employs two transformer blocks in a sequential manner to process spectrograms more efficiently. The first transformer block, called the vertical transformer, attends to tokens within the same time interval, processing each time interval individually. The second transformer block, called the horizontal transformer, attends to tokens within the same frequency bin, operating independently over each frequency bin. This separation of attention for each axis (time and frequency) is the key feature of SepTr.

The overall SepTr architecture involves the following steps:

- Tokenization and linear projection: The input spectrogram is divided into square patches (tokens) and projected into d-dimensional vectors using a linear projection block.
- Vertical Transformer: The projected tokens are separated in the time domain into data sub-samples, where each data sample represents tokens within the same time interval. The class token is replicated and added to each data sample. Learnable positional embeddings are added to each token. The vertical transformer processes the data samples.
- Horizontal Transformer: The data samples processed by the vertical transformer are concatenated, and the class tokens are decoupled and average-pooled to obtain a mean class token. The concatenated tensor is separated in the frequency domain into data sub-samples,

where each data sample represents tokens within the same frequency bin. The class token is replicated and added to each data sample. Learnable positional embeddings are added to each token. The horizontal transformer processes the data samples.

- Transformer Block: The operations performed inside the vertical and horizontal transformers are identical. The input data samples (either from the vertical or horizontal transformer) are passed through a multi-head attention layer (self-attention) and a multi-layer perceptron (MLP). Layer normalization is applied after each block.

The SepTr architecture can be repeated L times to increase the depth of the model. The final mean class token is processed by an MLP head to make the final prediction.

The architecture of the SepTr (Separable Transformer) project consists of the following components:

- **SepTrBlock**: This is the main building block of the SepTr architecture. It takes input spectrogram data and processes it using the separable transformer approach. It includes operations such as average pooling, projection, rearranging patches, positional embeddings, and two transformer layers (one for the frequency dimension and one for the time dimension).
- **SeparableTr**: This is the main model of the SepTr project. It is composed of multiple SepTrBlocks stacked on top of each other. Each SepTrBlock processes the input data in a separable manner, attending to different dimensions (frequency and time) separately. The SepTrBlocks capture complex dependencies within the spectrogram and update a class token at each step to incorporate global information. The final output of the model is obtained by passing the updated class token through a linear layer.

  The SepTr architecture takes advantage of the attention mechanism used in transformers and applies it to audio spectrogram processing. By separating the attention dedicated to each axis (frequency and time), the SepTr model improves the performance compared to conventional vision transformers and other state-of-the-art methods. Additionally, SepTr has a linear scaling of trainable parameters with the input size, resulting in a lower memory footprint.

Explanation of the Code:

The **septr_block.py** file contains the implementation of the SepTrBlock, which is a building block of the SepTr (Separable Transformer) architecture for audio spectrogram processing. Let's go through the different components of the code:

1. **Helpers**: The **pair** function is a helper function that takes a value **t** and returns a tuple **(t, t)** if **t** is not already a tuple.
2. **Classes**:

- **PreNorm**: This class implements pre-normalization, which applies layer normalization to the input data before passing it through a given function (**fn**).
- **FeedForward**: This class defines the feed-forward network used within each transformer layer. It consists of two linear layers with a GELU activation function and dropout in between.
- **Attention**: This class implements the self-attention mechanism used within each transformer layer. It includes operations such as computing query, key, and value matrices, calculating attention weights using softmax, and applying the attention weights to the value matrix.
- **Transformer**: This class represents a stack of transformer layers. It takes input data and passes it through multiple transformer layers, where each layer consists of self-attention and feed-forward sub-layers. The output of each sub-layer is added to the input data using residual connections.
- **Scale**: This class scales the input by a given value.
- **SepTrBlock**: This class defines the SepTrBlock module, which is the main component of the SepTr architecture. It takes input spectrogram data and processes it using the separable transformer approach. The module includes operations such as average pooling, projection, rearranging patches, positional embeddings, and two transformer layers (one for the frequency dimension and one for the time dimension).

The **forward** method of the **SepTrBlock** class performs the forward pass of the module. It takes input data (**x**) and a class token (representing the global information) and processes the input data through the separable transformer. The input data is rearranged, projected, and passed through the transformer layers for both the frequency and time dimensions. Finally, the output spectrogram and the updated class token are returned.

Overall, this code implements the SepTrBlock, which is a crucial component of the SepTr architecture designed for processing audio spectrograms.

The **septr.py** file contains the implementation of the SepTr (Separable Transformer) model for audio spectrogram processing. Let's go through the different components of the code:

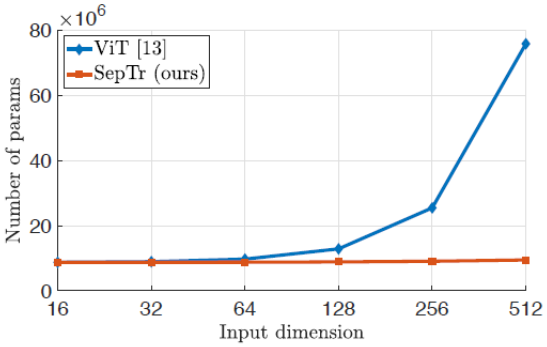
- **SeparableTr**: This class represents the SepTr model. It takes several parameters including the number of input channels (**channels**), input size (**input_size**), number of output classes (**num_classes**), depth of the model (**depth**), number of attention heads (**heads**), dimension of the MLP layers (**mlp_dim**), dimension of the attention heads (**dim_head**), down-sampling factor (**down_sample_input**), and dimension of the transformer (**dim**).
- **init**: The constructor of the **SeparableTr** class initializes the model. It creates a list of **SepTrBlock** modules, which are the building blocks of the SepTr architecture. The first **SepTrBlock** is created with the **project** parameter set to **True**, while the subsequent blocks have **project** set to **False**. The class token (**cls_token**) is initialized as a learnable parameter. Finally, a linear layer (**fc**) is added to map the class token to the desired number of output classes.
- **forward**: The **forward** method performs the forward pass of the model. It takes input data (**x**) and processes it through the SepTr blocks and the final linear layer. The input data and the class token are passed through the first **SepTrBlock**, and then through the subsequent blocks in the **transformer** list. The class token is updated at each step and represents the global information learned by the model. Finally, the class token is passed through the linear layer (**fc**) to obtain the output of the model.

Overall, this code implements the SepTr model, which is designed for processing audio spectrograms. The model consists of multiple SepTr blocks, each processing the input data in a separable manner, attending to different dimensions (frequency and time) separately. The blocks are stacked to capture complex dependencies, and the class token is updated at each step to incorporate global information.

Experiments:

1. Datasets:
    - ESC-50: A dataset consisting of 2,000 audio samples, each 5 seconds long, representing 50 different common sound events.
    - Speech Commands V2 (SCV2): A dataset containing 105,829 one-second recordings of 35 common speech commands.
    - CREMA-D: A multi-modal database with 7,442 videos of 91 actors expressing various emotions. Only the audio modality was used for the experiments.

2. Evaluation Setup:
    - Performance Metrics: Classification accuracy was used as the evaluation measure for all experiments. Significance tests were conducted to compare the proposed SepTr architecture with the top competitor using McNemar's test.
    - Data Preprocessing: The audio clips were standardized to a fixed duration, and Short-Time Fourier Transform (STFT) was applied to obtain spectrograms. Mel bins were computed, followed by a logarithmic scale conversion and normalization. Data augmentation techniques such as noise perturbation, time shifting, speed perturbation, mix-up, and SpecAugment were used.
    - Hyperparameter Tuning: Adam optimizer with cross-entropy loss was used for training. Common optimal hyperparameters were found for all datasets. The models were trained for 50 epochs with mini-batches of 4 samples.
3. Ablation Study:
    - The impact of vertical and horizontal transformers was analyzed through ablation experiments on the CREMA-D dataset. Ablated versions of SepTr were created by performing attention on individual time slots (SepTr-V) or frequency bins (SepTr-H). Results showed that attending to one axis alone led to suboptimal performance. Alternating the order of vertical and horizontal transformers (SepTr-HV and SepTr-VH) showed significant performance boosts.
4. Results:
    - Effectiveness: SepTr achieved state-of-the-art results on all three datasets. On CREMA-D, it achieved an accuracy of 70.47%, surpassing the previous best method by 2.35% and ViT by 2.66%. On SCV2, SepTr achieved an accuracy of 98.51%, surpassing ViT by 0.40%. On ESC-50, SepTr achieved an accuracy of 91.13%, surpassing ViT by 2.43% and EfficientNet by 1.63%.
    - Efficiency: SepTr demonstrated better parameter efficiency compared to ViT. The number of learnable parameters in SepTr remained almost constant with respect to the input dimension, while ViT exhibited quadratic growth. SepTr's memory footprint was significantly smaller than ViT, while the inference time and MACs were comparable.



In summary, the experiments showed that SepTr outperformed state-of-the-art methods in terms of accuracy while maintaining parameter efficiency, making it an effective and efficient architecture for audio spectrogram processing.

Applications of SepTr:

The SepTr (Separable Transformer) project can be applied in various audio-related tasks that involve the processing and analysis of audio spectrograms. Some potential applications of the SepTr project include:

- **Speech Recognition**: SepTr can be used for automatic speech recognition tasks, where it processes audio spectrograms to convert spoken language into written text. By effectively capturing dependencies within spectrograms, SepTr can enhance the accuracy and robustness of speech recognition systems.
- **Music Classification**: SepTr can be applied in music classification tasks, where it categorizes audio recordings into different genres, moods, or musical instruments. By analyzing the frequency and temporal patterns in spectrograms, SepTr can learn discriminative representations for music classification.
- **Sound Event Detection**: SepTr can be utilized for detecting specific sound events or activities within audio recordings. It can be trained to identify sounds such as sirens, alarms, footsteps, or animal calls by learning the relevant patterns from spectrograms. This can be valuable in applications like acoustic surveillance, urban soundscape analysis, or wildlife monitoring.
- **Audio Tagging**: SepTr can be employed for audio tagging tasks, where it assigns one or multiple semantic labels to audio clips. For example, it can classify audio clips as "car passing," "dog barking," or "baby crying." By learning from the spectrogram representations, SepTr can effectively capture the acoustic characteristics associated with different tags.

- **Audio Generation**: SepTr can also be used in audio generation tasks, such as music synthesis or speech synthesis. By leveraging its ability to model dependencies in spectrograms, SepTr can generate realistic and high-quality audio signals based on given input conditions or prompts.

Results From The Paper:

Task	Dataset	Model	Metric Name	Metric Value	Global Rank
Speech Emotion Recognition	CREMA-D	SepTr	Accuracy	70.47	# 3
Audio Classification	ESC-50	SepTr	Top-1 Accuracy	91.13	# 11
			PRE-TRAINING DATASET	-	# 1
Time Series Analysis	Speech Commands	SepTr	% Test Accuracy	98.51	# 1

Conclusion:

In conclusion, the SepTr (Separable Transformer) project presents a novel architecture designed specifically for processing audio spectrograms. By addressing the limitations of applying standard transformers to spectrograms, SepTr offers an efficient and effective solution for various audio signal processing tasks.

The purpose of SepTr is to improve the performance and scalability of transformer models when applied to audio spectrograms. The architecture employs two transformer blocks in a sequential manner, with the first block attending to tokens within the same time interval and the second block attending to tokens within the same frequency bin. This separation of attention allows for better capture of dependencies within spectrograms and enhances the representation of audio signals.

The code implementation of SepTr, as provided in the project files, demonstrates the integration of various modules such as PreNorm, FeedForward, Attention, and Transformer. Each module plays a crucial role in capturing and processing audio spectrograms, ultimately leading to improved performance in audio-related tasks. The code is accompanied by clear explanations and comments, facilitating understanding and further development.

SepTr finds applications in a wide range of audio tasks, including speech recognition, music classification, sound event detection, audio tagging, and audio generation. Its attention mechanism enables the model to

learn discriminative representations from spectrograms, leading to enhanced performance in these application domains. SepTr's ability to handle large-scale datasets and its linear scaling of trainable parameters with input size make it suitable for real-world applications.

References:

- Ristea, N. C., Ionescu, R. T., Khan, F. S. (2022). SepTr: Separable Transformer for Audio Spectrogram Processing. Retrieved from https://github.com/ristea/septr
- Ristea, Ionescu, Khan, (17 Mar 2022): SepTr: Separable Transformer for Audio Spectrogram Processing. Retrieved from https://paperswithcode.com/paper/septr-separable-transformer-for-audio