EMOTIONAL FACIAL RECOGNITION AND CROWD COUNTING

Senior Project 2
Department of Information Systems & Technology College of Computer Sciences & Engineering

SUPERVISED BY:

Dr. Safaa Habibullah

OUR TEAM

Khlood Alamoudi
2115339

Asma Habadi
2112087

Rodina Bin Mahfouz
2110271

Jood Aljohani
2111644

Jood Fidah
2114874

TABLE OF CONTENT

01 INTRODUCTION

PROBLEM DEFINITION

Facial expression recognition technology analyzes facial images to understand emotions, often limited to identifying individuals based on their emotional state. Meanwhile, computer vision-based crowd counting is vital for measuring crowd density and ensuring public safety.

PROPOSED SOLUTION

The recommended solution is to use deep learning algorithms such as Convolution Neural Network (CNN). This method enhances human-computer interaction with facial expression recognition and counts the number of people, to measure customer satisfaction.

OBJECTIVES

01 Creating an advanced analytical approach to facial emotional detection using images

02 Creating a Model to Detect Number of People in Crowd Places.

DATA COLLECTION

Data collection is essential for solving various tasks and determining outcomes, with organizations using tools to predict future trends based on collected data.

DATA DESCRIPTION

Two datasets will be used in our project :

1- Facial emotional recognition.

FER dataset.

2- Crowd counting.

UIT dataset.

DATASET SAMPLES

FACIAL EMOTIONAL RECOGNITION DATASET

DATASET SAMPLES

CROWD COUNTING DATASET

02 MODEL BUILDING

EXPERIMENTAL SETUP

We build our models using Windows operating systems, Jupyter Notebook, and the Google Collab environment for programming software.

LIBRARIES USED

OS

Enables system-level operations like file and directory management, offering a portable, simplified interface to OS features.

Pandas

A powerful library for data manipulation, supporting tasks like cleaning, merging, and transforming datasets from various file formats.

NumPy

Optimized for large arrays and matrices, it supports mathematical functions for tasks like linear algebra and signal processing.

LIBRARIES USED

Tensorflow

A machine learning framework by Google for building and training complex models, reducing production time and costs.

Keras

A user-friendly deep learning API integrated with TensorFlow, designed for flexible and modular programming.

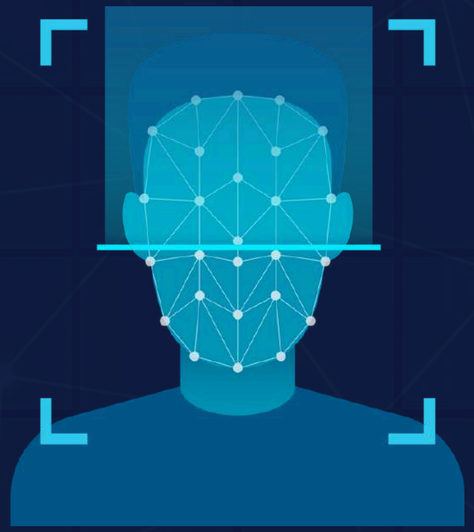
matplotlib.pyplot

A plotting library for creating static, animated, and interactive visualizations, offering tools to explore data visually.

LIBRARIES USED

Libraries	Extensions
import ResNet50V2, VGG16, MobileNetV2	tensorflow.keras.applications
Import Conv2D,MaxPooling2D	tensorflow.keras.layer
Import Models	keras.models
Import Input, Dense, Dropout,Flatten	tensorflow.keras.layers
Import img_to_array, load_img	keras.Preprocessing.image

FACIAL EMOTIONAL RECOGNITION MODELS



FACIAL EMOTIONAL RECOGNITION MODELS

MobileNetV2 Model

VGG16 Model

ResNet50V2 Model

CNN Model

INITIAL SETUP PARAMETERS

Hyperparameters							
Model	**Winner**	**Version**	**Epochs**	**Learning rate**	**Number of custom layers**	**Batch size**	**Accuracy**
MobileNetV3Small		1	30	1e-3	4 layers	32	11.44%
		2	30	1e-4	8 layers	32	25.46%
MobileNetV2		1	30	1e-5	5 layers	32	28.77%
		2	50	1e-5	5 layers	32	49.66%
		3	150	1e-5	5 layers	32	55.57%
		4	350	1e-5	5 layers	32	64.31%
	✓	5	50	1e-4	9 layers.	64	79.79%

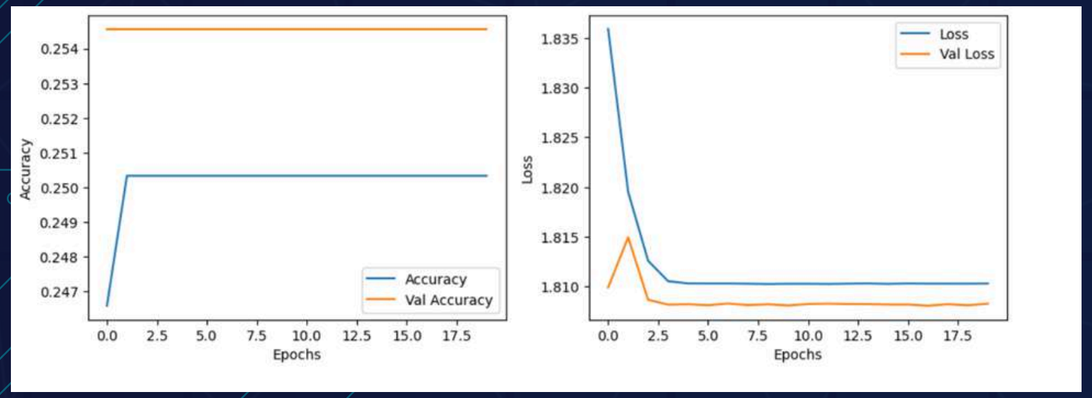
	Hyperparameters						
Model	**Winner**	**Version**	**Epochs**	**Learning rate**	**Number of layers**	**Batch size**	**Accuracy**
CNN		1	30	0.001	17 layers	64	14%
		2	50	0.001	10 layers	64	48.45%
	✓	3	150	0.0001	25 layers	64	92.00%

FACIAL EMOTIONAL RECOGNITION MODELS

MobileNetV2 Model

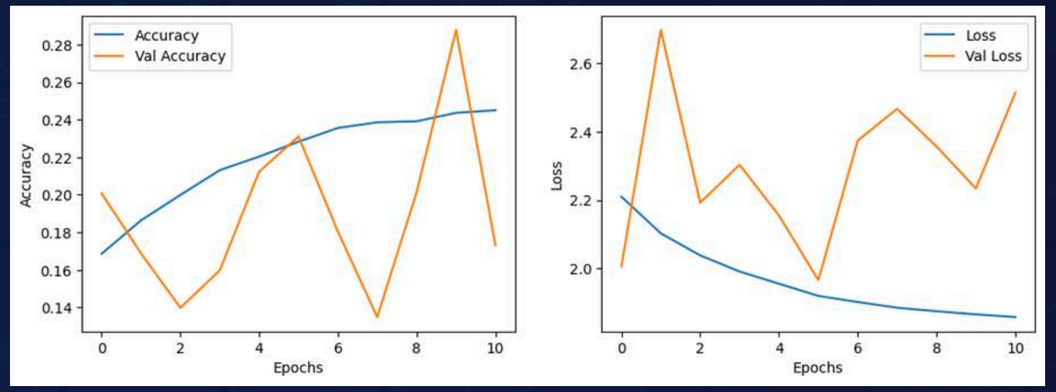
1 MobileNetV3Small-Vresion 1

```
base_model = MobileNetV3Small(input_shape=(224, 224, 3), weights="imagenet", include_top=False)
x = base_model.output
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
output_layer = Dense(7, activation='softmax')(x)
```



2 MobileNetV2-Vresion 2

```
[ ] base_model = MobileNetV2(input_shape=(224, 224, 3), weights="imagenet", include_top=False)
    x = base_model.output
    x = Flatten()(x)
    x = Dense(128, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)
    output_layer = Dense(7, activation='softmax')(x)
```

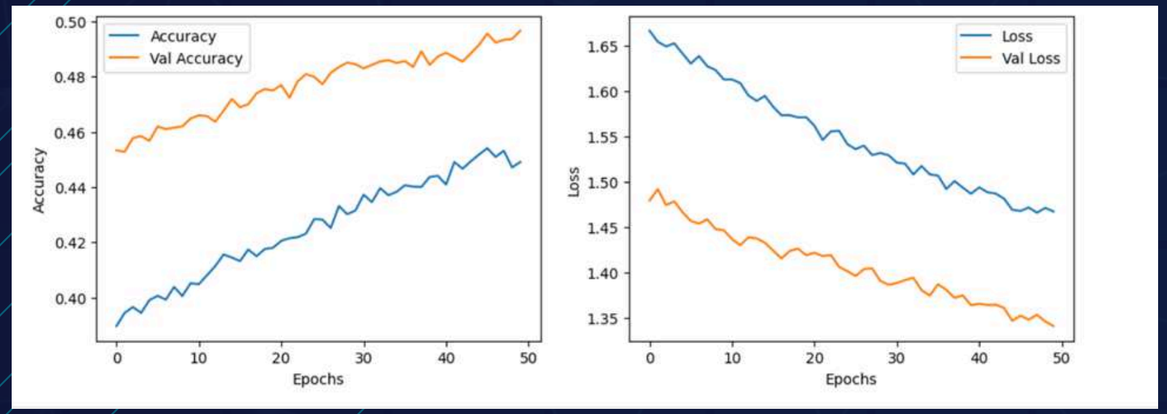


FACIAL EMOTIONAL RECOGNITION MODELS

MobileNetV2 Model

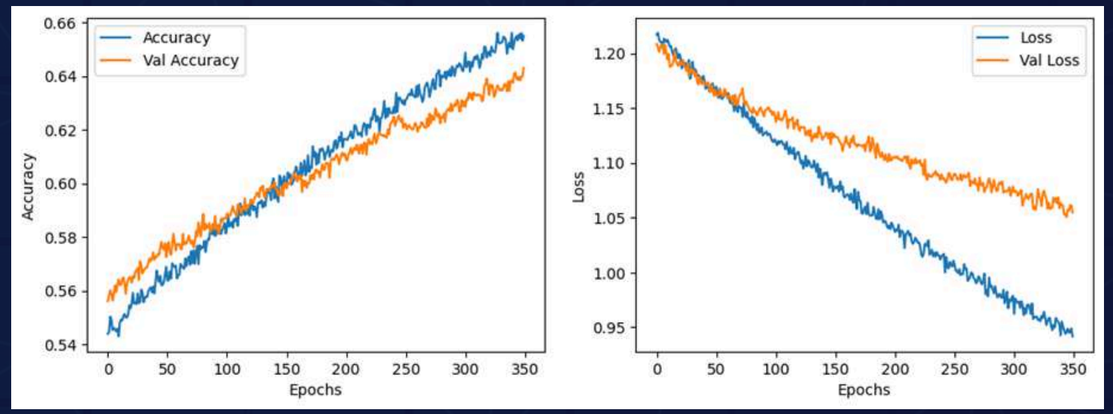
3 MobileNetV2-Vresion 3

```
base_model = MobileNetV2(input_shape=(224, 224, 3), weights="imagenet", include_top=False)
x = base_model.output
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
output_layer = Dense(7, activation='softmax')(x)
```



4 MobileNetV2-Vresion 4

```
base_model = MobileNetV2(input_shape=(224, 224, 3), weights="imagenet", include_top=False)
x = base_model.output
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
output_layer = Dense(7, activation='softmax')(x)
```



FACIAL EMOTIONAL RECOGNITION MODELS

MobileNetV2 Model with K-Fold Cross

```
Average Validation Accuracy across 5 folds: 74.37%
Standard Deviation: 5.47%
```

Classification Report:

	precision	recall	f1-score	support
neutral	0.59	0.68	0.63	3044
sad	0.64	0.53	0.58	2868
angry	0.61	0.59	0.60	2369
disgusted	0.78	0.66	0.72	256
happy	0.73	0.79	0.76	4313
fearful	0.63	0.57	0.60	2431
surprised	0.78	0.79	0.79	1954
accuracy			0.67	17235
macro avg	0.68	0.66	0.67	17235
weighted avg	0.67	0.67	0.66	17235

FACIAL EMOTIONAL RECOGNITION MODELS

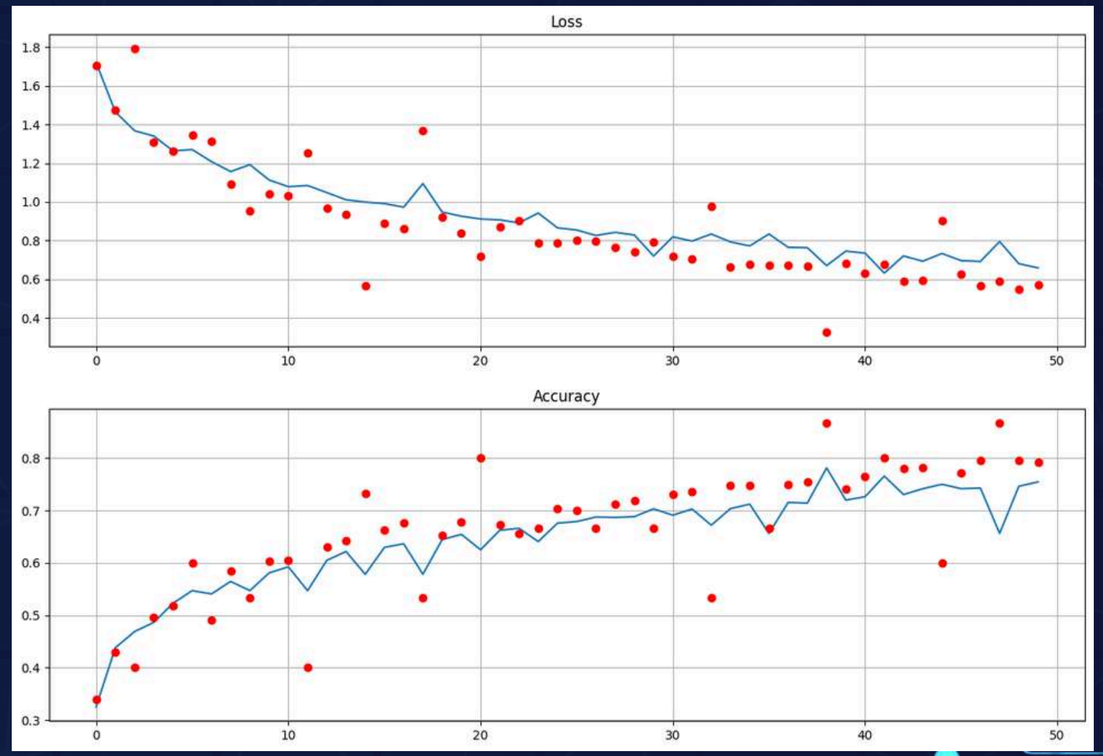
MobileNetV2 Model

```
input_shape = (224, 224, 3)

base_model = MobileNetV2(include_top=False, weights='imagenet', input_shape=input_shape)

model = Sequential([
    base_model,
    BatchNormalization(),
    GlobalAveragePooling2D(),
    Dense(512, activation='relu'),
    Dropout(0.1),
    Dense(256, activation='relu'),
    Dropout(0.1),
    Dense(128, activation='relu'),
    Dropout(0.1),
    Dense(7, activation='softmax')
])

optimizer = Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

FACIAL EMOTIONAL RECOGNITION RESULT

MobileNetV2 Model

```
1516/1516 ━━━━━━━━━━━━━━━━━━━━ 500s 330ms/step - accuracy: 0.7828 - loss: 0.5927
569/569 ━━━━━━━━━━━━━━━━━━━━ 26s 45ms/step - accuracy: 0.7750 - loss: 0.6311

Final Training Accuracy: 78.19%
Final Test Accuracy: 79.79%
```



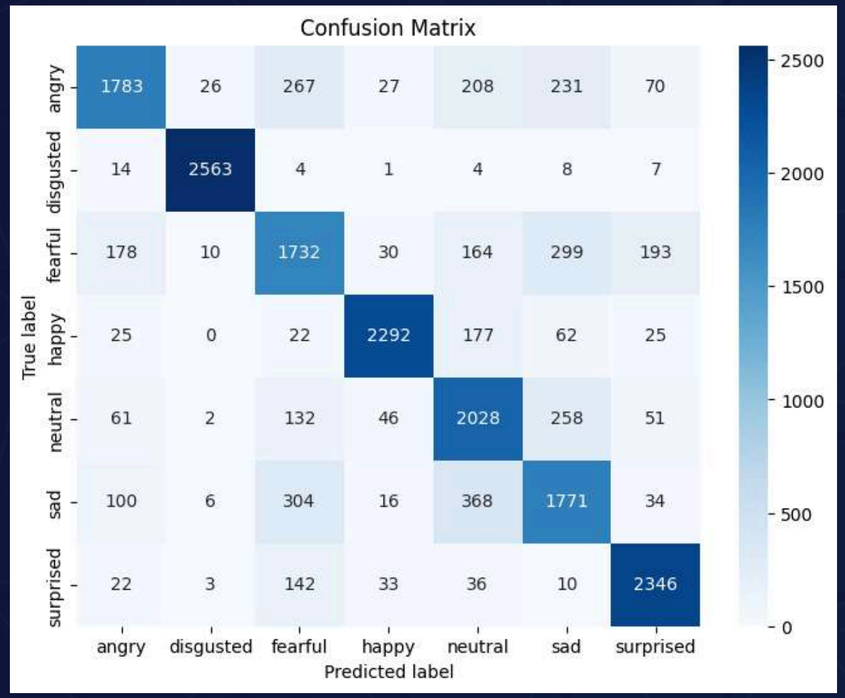
FACIAL EMOTIONAL RECOGNITION RESULT

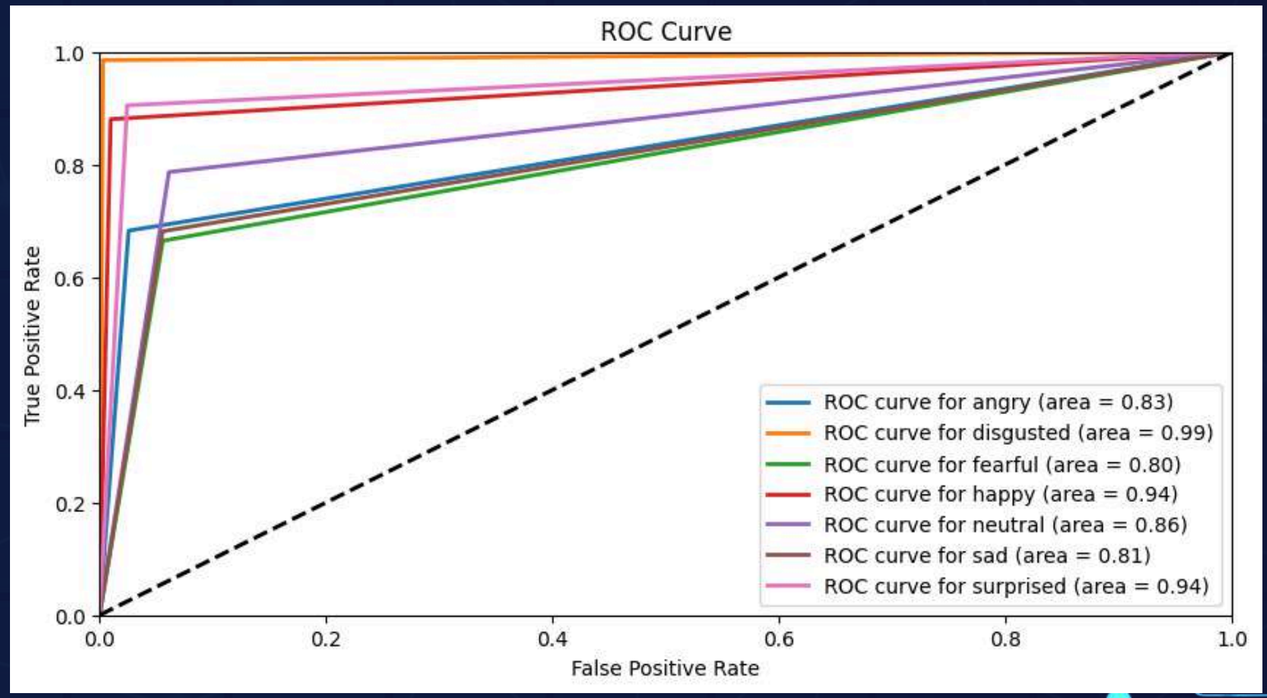
MobileNetV2 Model

	precision	recall	f1-score	support
0	0.82	0.68	0.74	2612
1	0.98	0.99	0.98	2601
2	0.67	0.66	0.67	2606
3	0.94	0.88	0.91	2603
4	0.68	0.79	0.73	2578
5	0.67	0.68	0.68	2599
6	0.86	0.91	0.88	2592
accuracy			0.80	18191
macro avg	0.80	0.80	0.80	18191
weighted avg	0.80	0.80	0.80	18191

FACIAL EMOTIONAL RECOGNITION RESULT

MobileNetV2 Model





FACIAL EMOTIONAL RECOGNITION RESULT

MobileNetV2 Model

Sample of Prediction

FACIAL EMOTIONAL RECOGNITION MODELS

VGG16 Model

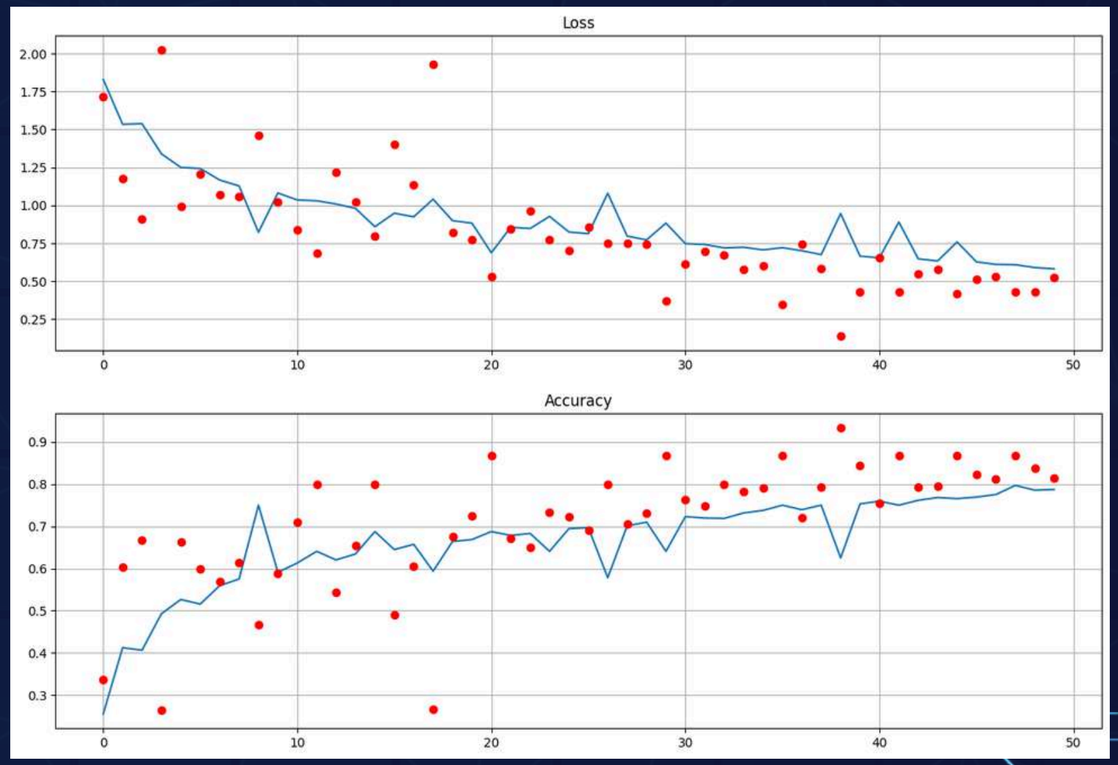
```
# VGG16 as the base model
vgg_base = VGG16(include_top=False, weights='imagenet', input_shape=(224, 224, 3))

# Building the model
vgg_model = Sequential([
    vgg_base,
    BatchNormalization(),
    GlobalAveragePooling2D(),
    Dense(512, activation='relu'),
    Dropout(0.1),
    Dense(256, activation='relu'),
    Dropout(0.1),
    Dense(128, activation='relu'),
    Dropout(0.1),
    Dense(7, activation='softmax')
])
```

FACIAL EMOTIONAL RECOGNITION RESULT

VGG16 Model

```
1516/1516 ━━━━━━━━━━━━━━━━━━━━ 559s 369ms/step - accuracy: 0.8051 - loss: 0.5316
569/569 ━━━━━━━━━━━━━━━━━━━━ 39s 68ms/step - accuracy: 0.8125 - loss: 0.5276
```

Final Training Accuracy: 80.51%
Final Test Accuracy: 82.44%



FACIAL EMOTIONAL RECOGNITION RESULT

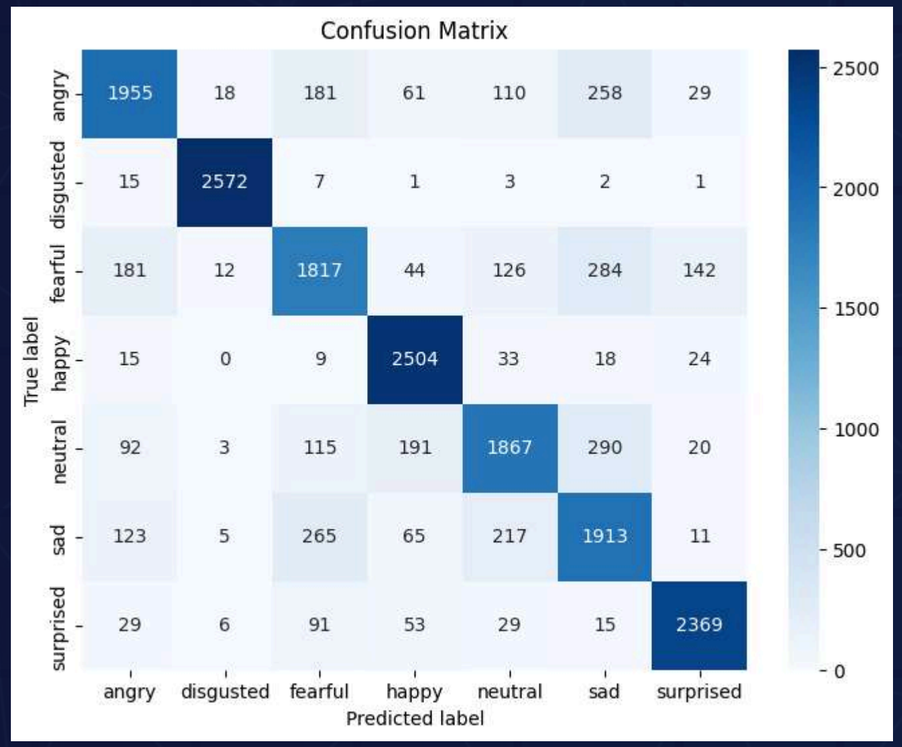
VGG16 Model

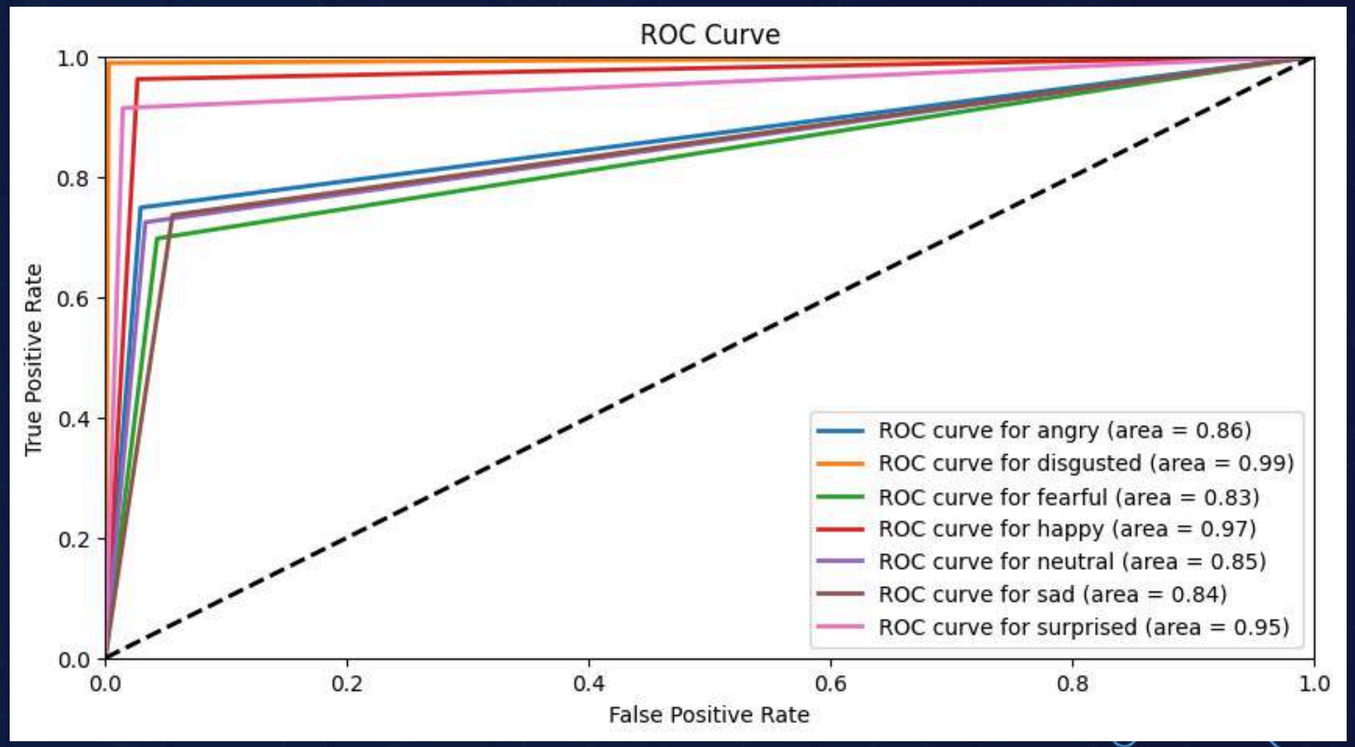
```
569/569 ━━━━━━━━━━━━━━━━━━━━ 34s 60ms/step
```

	precision	recall	f1-score	support
angry	0.81	0.75	0.78	2612
disgusted	0.98	0.99	0.99	2601
fearful	0.73	0.70	0.71	2606
happy	0.86	0.96	0.91	2603
neutral	0.78	0.72	0.75	2578
sad	0.69	0.74	0.71	2599
surprised	0.91	0.91	0.91	2592
accuracy			0.82	18191
macro avg	0.82	0.82	0.82	18191
weighted avg	0.82	0.82	0.82	18191

FACIAL EMOTIONAL RECOGNITION RESULT

VGG16 Model





FACIAL EMOTIONAL RECOGNITION RESULT

VGG16 Model

Sample of Prediction

FACIAL EMOTIONAL RECOGNITION MODELS

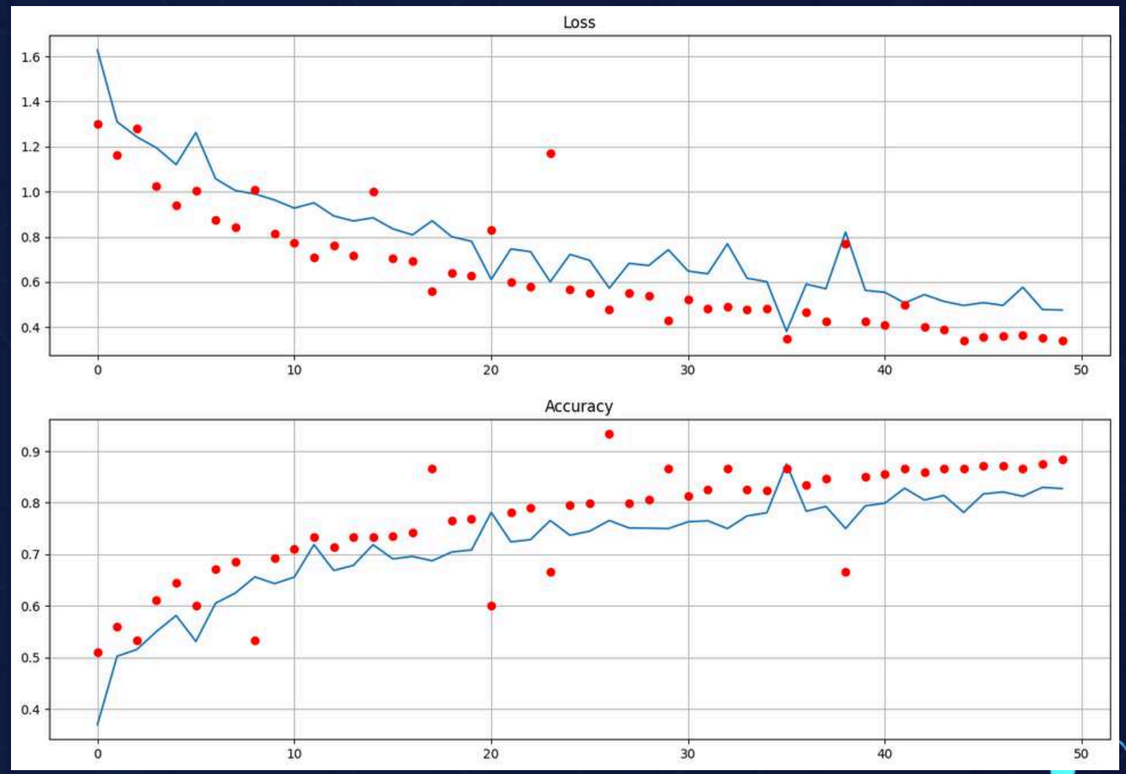
ResNet50V2 Model

```
input_shape = (224, 224, 3)

base_model = ResNet50V2(include_top=False, weights='imagenet', input_shape=input_shape)

model = Sequential([
    base_model,
    BatchNormalization(),
    GlobalAveragePooling2D(),
    Dense(512, activation='relu'),
    Dropout(0.1),
    Dense(256, activation='relu'),
    Dropout(0.1),
    Dense(128, activation='relu'),
    Dropout(0.1),
    Dense(7, activation='softmax')
])

optimizer = Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

FACIAL EMOTIONAL RECOGNITION RESULT

ResNet50V2 Model

```
1516/1516 ━━━━━━━━━━━━━━━━━━━━ 573s 378ms/step - accuracy: 0.8624 - loss: 0.3828
569/569 ━━━━━━━━━━━━━━━━━━━━ 39s 69ms/step - accuracy: 0.8813 - loss: 0.3453

Final Training Accuracy: 86.26%
Final Test Accuracy: 88.09%
```



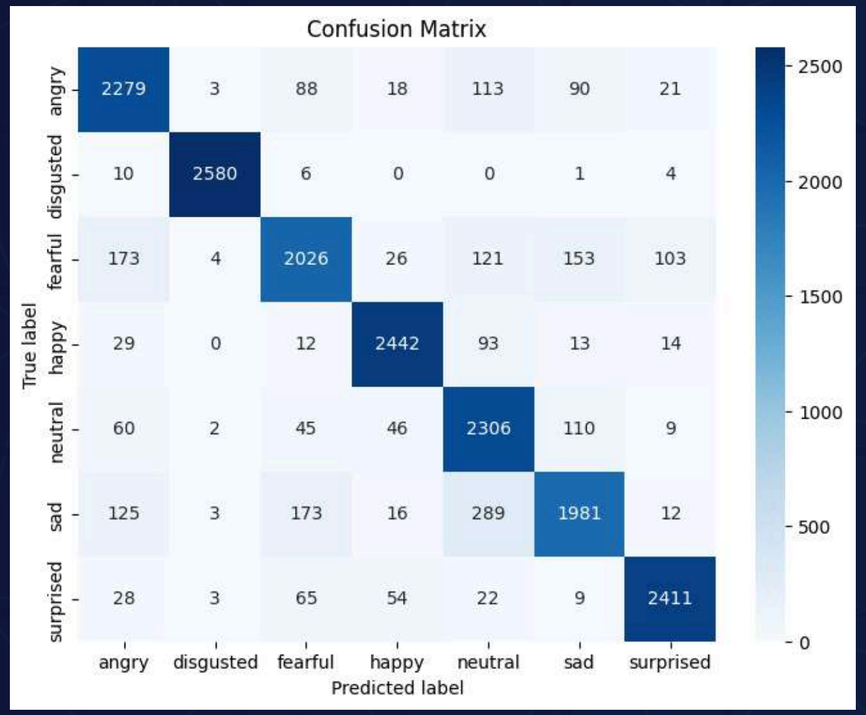
FACIAL EMOTIONAL RECOGNITION RESULT

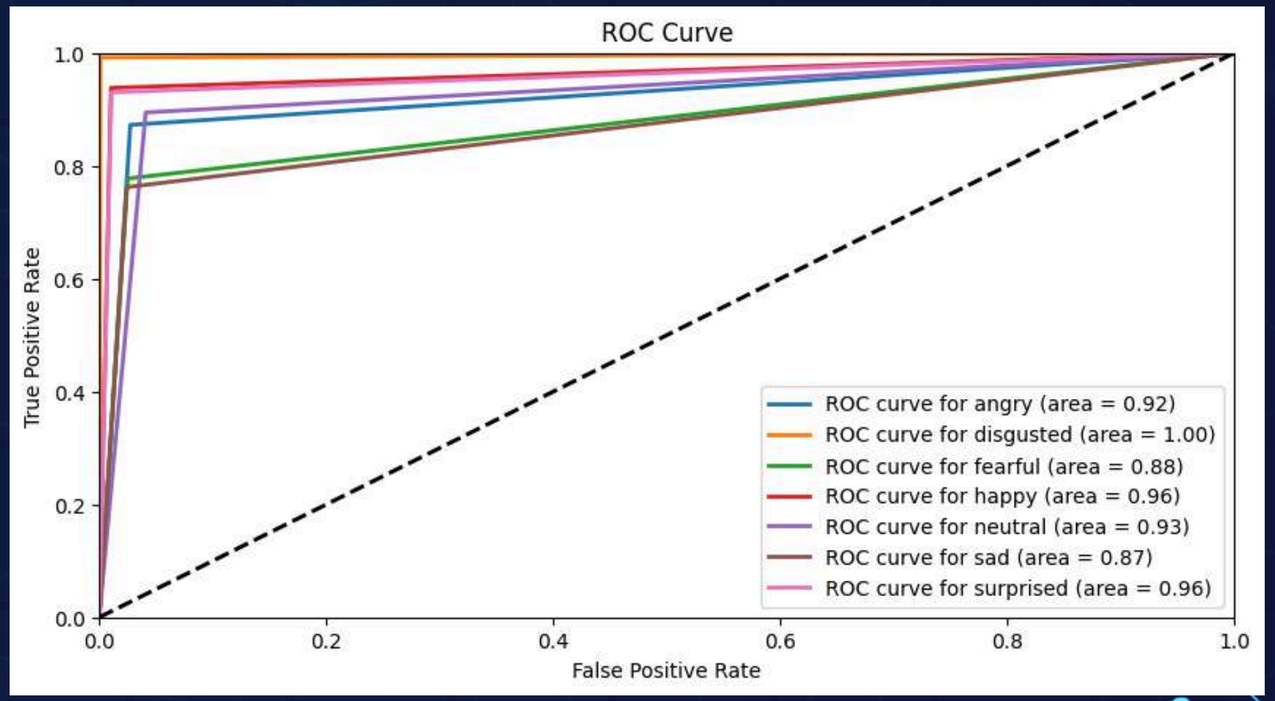
ResNet50V2 Model

	precision	recall	f1-score	support
0	0.84	0.87	0.86	2612
1	0.99	0.99	0.99	2601
2	0.84	0.78	0.81	2606
3	0.94	0.94	0.94	2603
4	0.78	0.89	0.84	2578
5	0.84	0.76	0.80	2599
6	0.94	0.93	0.93	2592
accuracy			0.88	18191
macro avg	0.88	0.88	0.88	18191
weighted avg	0.88	0.88	0.88	18191

FACIAL EMOTIONAL RECOGNITION RESULT

ResNet50V2 Model





FACIAL EMOTIONAL RECOGNITION RESULT

ResNet50V2 Model

Sample of Prediction

FACIAL EMOTIONAL RECOGNITION MODELS

CNN Model

1. CNN-version1
2. CNN-version2
3. CNN-version3

FACIAL EMOTIONAL RECOGNITION MODELS

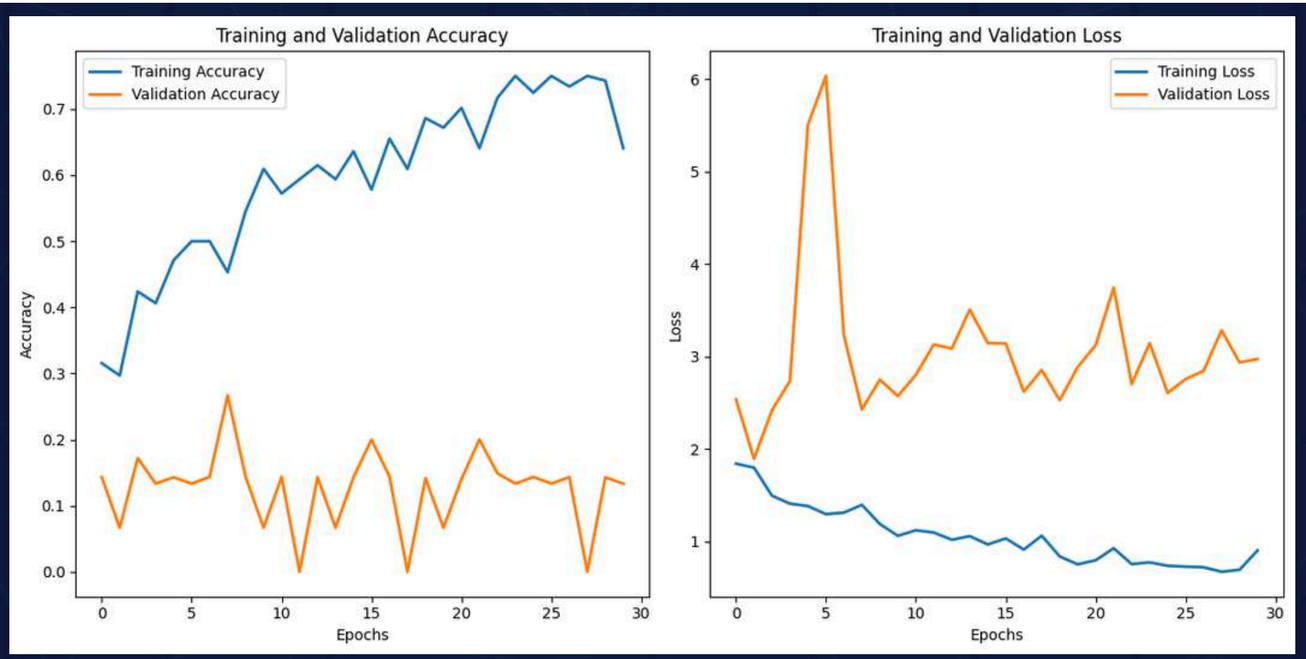
CNN-version1

```python
def create_emotion_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)),
        BatchNormalization(),
        Conv2D(64, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D(2, 2),
        Dropout(0.25),
        Conv2D(128, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D(2, 2),
        Conv2D(128, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D(2, 2),
        Dropout(0.25),
        Flatten(),
        Dense(1024, activation='relu'),
        Dropout(0.5),
        Dense(7, activation='softmax')
    ])
    return model_v1
```

FACIAL EMOTIONAL RECOGNITION RESULT

CNN-version1

```
758/758 ━━━━━━━━━━━━━━━━━━━━ 17s 22ms/step - accuracy: 0.8643 - loss: 0.4854
285/285 ━━━━━━━━━━━━━━━━━━━━ 7s 25ms/step - accuracy: 0.1449 - loss: 2.9572
Training Accuracy: 86.63%
Validation Accuracy: 14.28%
```



FACIAL EMOTIONAL RECOGNITION RESULT

CNN-version1

Classification Report for Custom CNN Model:

	precision	recall	f1-score	support
angry	0.15	0.01	0.01	2612
disgusted	0.00	0.00	0.00	2601
fearful	0.00	0.00	0.00	2606
happy	1.00	0.00	0.00	2603
neutral	0.00	0.00	0.00	2578
sad	0.00	0.00	0.00	2599
surprised	0.14	0.99	0.25	2592
accuracy			0.14	18191
macro avg	0.18	0.14	0.04	18191
weighted avg	0.18	0.14	0.04	18191

FACIAL EMOTIONAL RECOGNITION RESULT

CNN-version1

Hyperparameter Tuning

We use random search to optimize the hyperparameters of our predictive model. By randomly sampling combinations of parameters such as learning rate, batch size, and the number of hidden layers, we efficiently explored the hyperparameter space.

```
Trial 10 Complete [00h 08m 49s]
val_accuracy: 0.2666666805744171

Best val_accuracy So Far: 0.4666666865348816
Total elapsed time: 01h 24m 53s
```

FACIAL EMOTIONAL RECOGNITION MODELS

CNN-version2

```
def create_cnn_model():

    model = Sequential([

        Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)),
        MaxPooling2D(2, 2),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(7, activation='softmax')

    ])

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

    return model
```

FACIAL EMOTIONAL RECOGNITION RESULT

CNN-version2

Grid Search

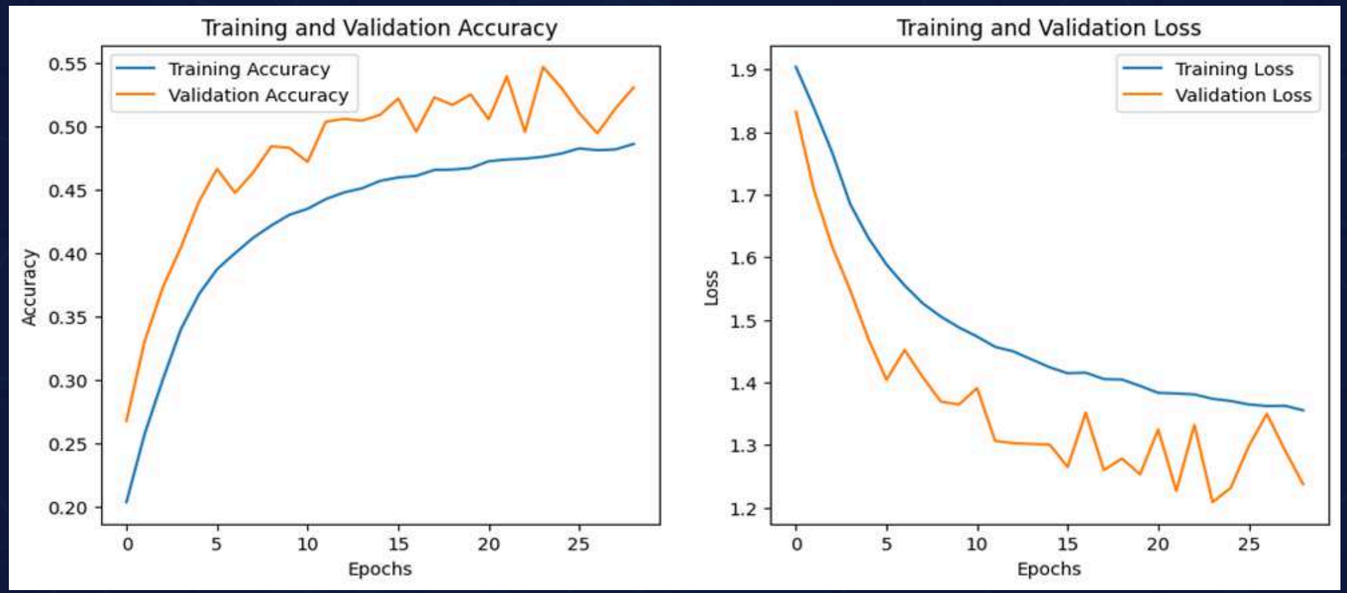
Index	Batch Size	Learning Rate	Dropout Rate	Validation Accuracy
0	32	0.001	0.3	0.5160
1	32	0.001	0.5	0.5337
2	32	0.0001	0.3	0.4978
3	32	0.0001	0.5	0.4037
4	32	1e-05	0.3	0.2753
5	32	1e-05	0.5	0.2779
6	64	0.001	0.3	0.4596
7	64	0.001	0.5	0.5419
8	64	0.0001	0.3	0.4040
9	64	0.0001	0.5	0.4665
10	64	1e-05	0.3	0.2864
11	64	1e-05	0.5	0.2730

Best parameters: Batch Size=64, Learning Rate=0.001, Dropout Rate=0.5, Validation Accuracy=0.5419

FACIAL EMOTIONAL RECOGNITION RESULT

CNN-version2

```
553/553 ━━━━━━━━━━━━━━━━━━━━ 25s 44ms/step - accuracy: 0.4597 - loss: 1.4132
119/119 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - accuracy: 0.4331 - loss: 1.4355
Training Accuracy: 45.80%
Test Accuracy: 48.45%
```



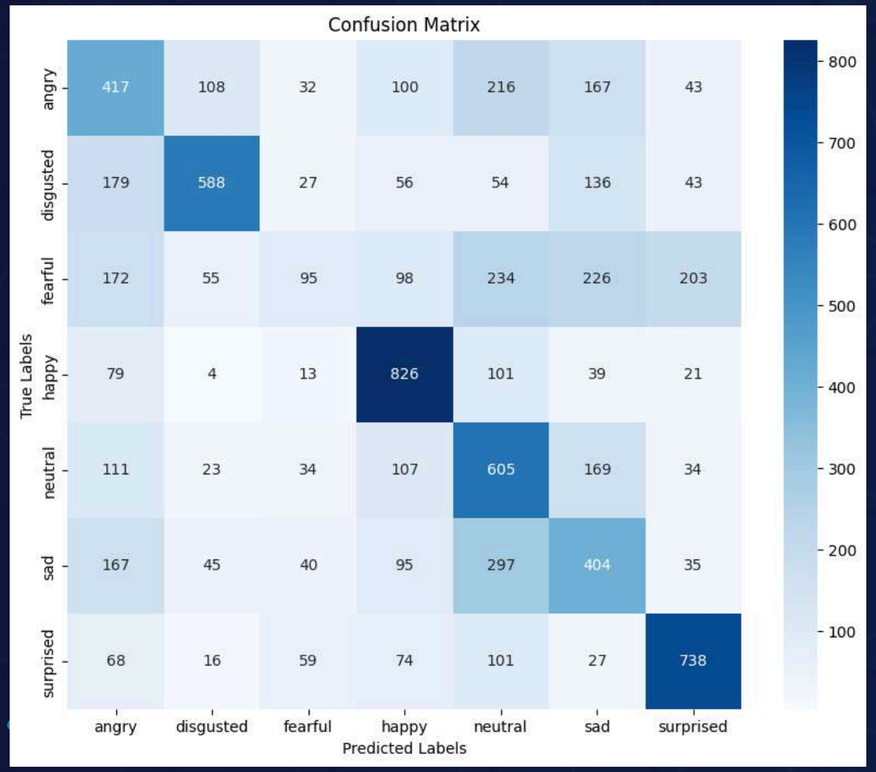
FACIAL EMOTIONAL RECOGNITION RESULT

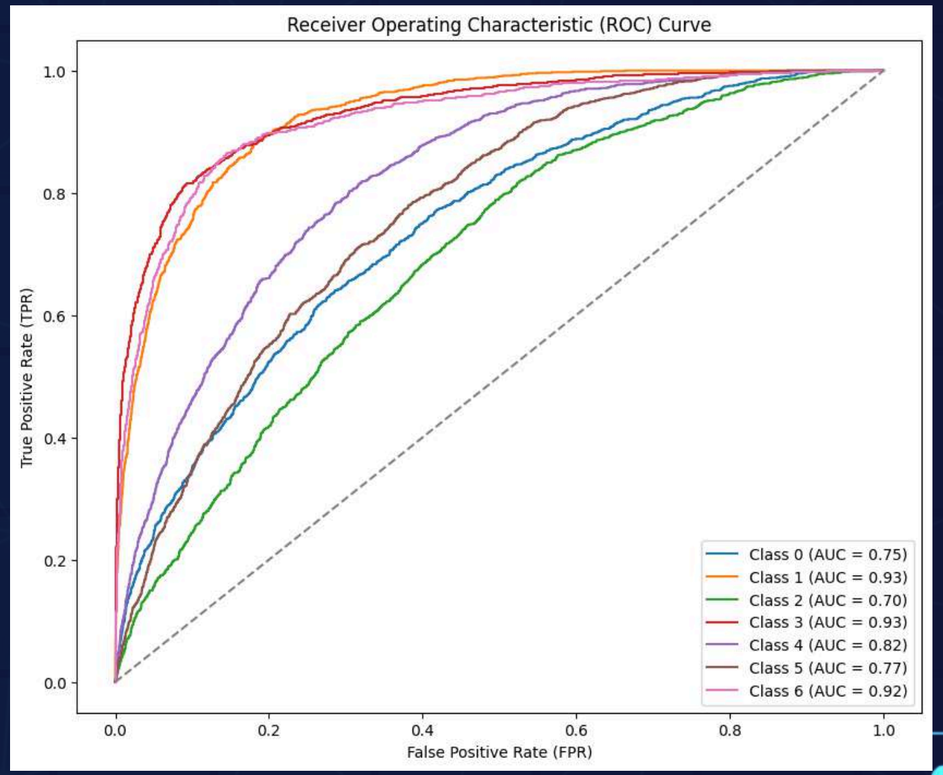
CNN-version2

	precision	recall	f1-score	support
angry	0.35	0.39	0.37	1083
disgusted	0.70	0.54	0.61	1083
fearful	0.32	0.09	0.14	1083
happy	0.61	0.76	0.68	1083
neutral	0.38	0.56	0.45	1083
sad	0.35	0.37	0.36	1083
surprised	0.66	0.68	0.67	1083
accuracy			0.48	7581
macro avg	0.48	0.48	0.47	7581
weighted avg	0.48	0.48	0.47	7581

FACIAL EMOTIONAL RECOGNITION RESULT

CNN-version2





FACIAL EMOTIONAL RECOGNITION RESULT

CNN-version2

Sample of Prediction

FACIAL EMOTIONAL RECOGNITION MODELS

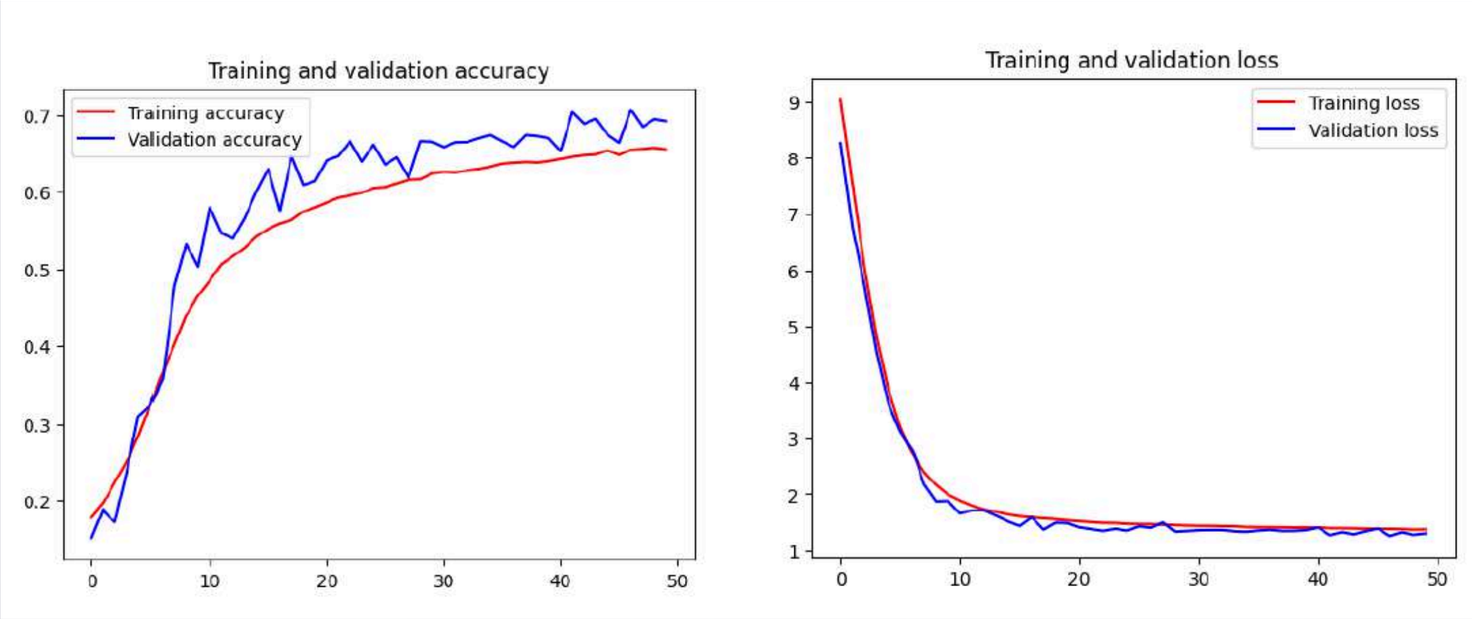
CNN-version3

```python
model = Sequential([

    Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)),
    Conv2D(64, (3, 3), padding='same', activation='relu'),
    BatchNormalization(),
    MaxPooling2D(2, 2),
    Dropout(0.25),
    Conv2D(128, (5, 5), padding='same', activation='relu'),
    BatchNormalization(),
    MaxPooling2D(2, 2),
    Dropout(0.25),
    Conv2D(512, (3, 3), padding='same', activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01)),
    BatchNormalization(),
    MaxPooling2D(2, 2),
    Dropout(0.25),
    Conv2D(512, (3, 3), padding='same', activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01)),
    BatchNormalization(),
    MaxPooling2D(2, 2),
    Dropout(0.25),
    # Flatten layer
    Flatten(),
    Dense(256, activation='relu'),
    BatchNormalization(),
    Dropout(0.25),
    Dense(512, activation='relu'),
    BatchNormalization(),
    Dropout(0.25),
    # output layer
    Dense(7, activation='softmax')
])
```

FACIAL EMOTIONAL RECOGNITION RESULT

CNN-version3

```
607/607 ━━━━━━━━━━━━━━━━━━━━ 51s 84ms/step - accuracy: 0.6964 - loss: 1.2679
143/143 ━━━━━━━━━━━━━━━━━━━━ 8s 54ms/step - accuracy: 0.6715 - loss: 1.3745
final train accuracy = 69.74 , validation accuracy = 69.21
```

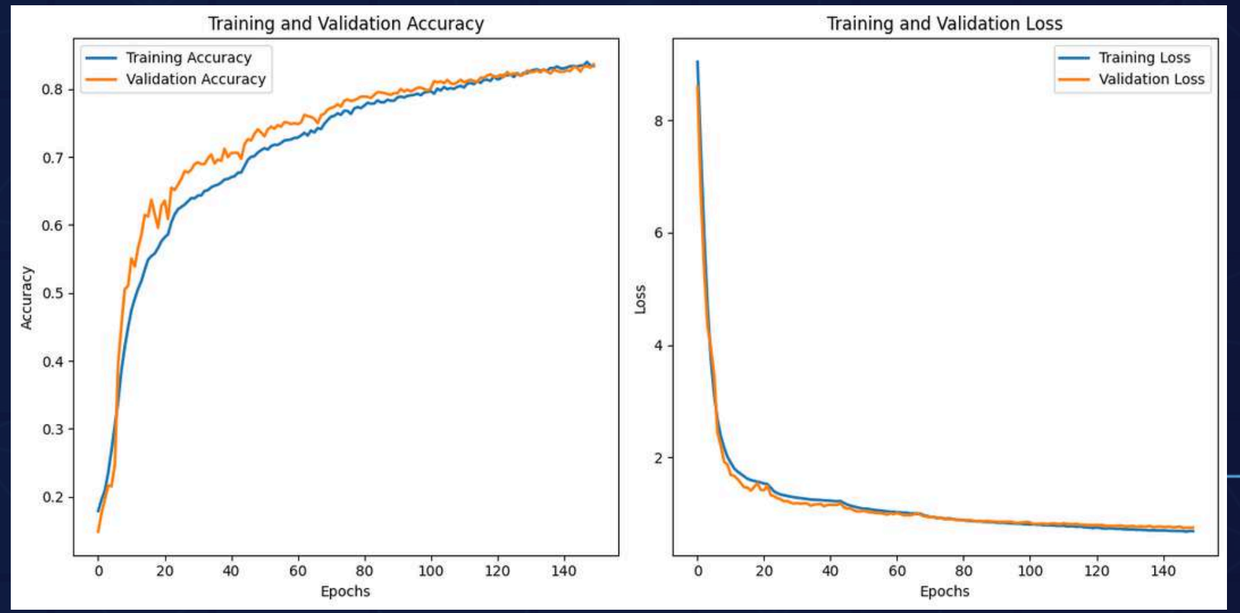


FACIAL EMOTIONAL RECOGNITION RESULT

CNN-version3

```
607/607 ━━━━━━━━━━━━━━━━━━━━ 223s 368ms/step - accuracy: 0.9441 - loss: 0.4107
228/228 ━━━━━━━━━━━━━━━━━━━━ 83s 365ms/step - accuracy: 0.9242 - loss: 0.4806
Training Accuracy: 94.37%
Test Accuracy: 92.00%
```

After Optimize Model



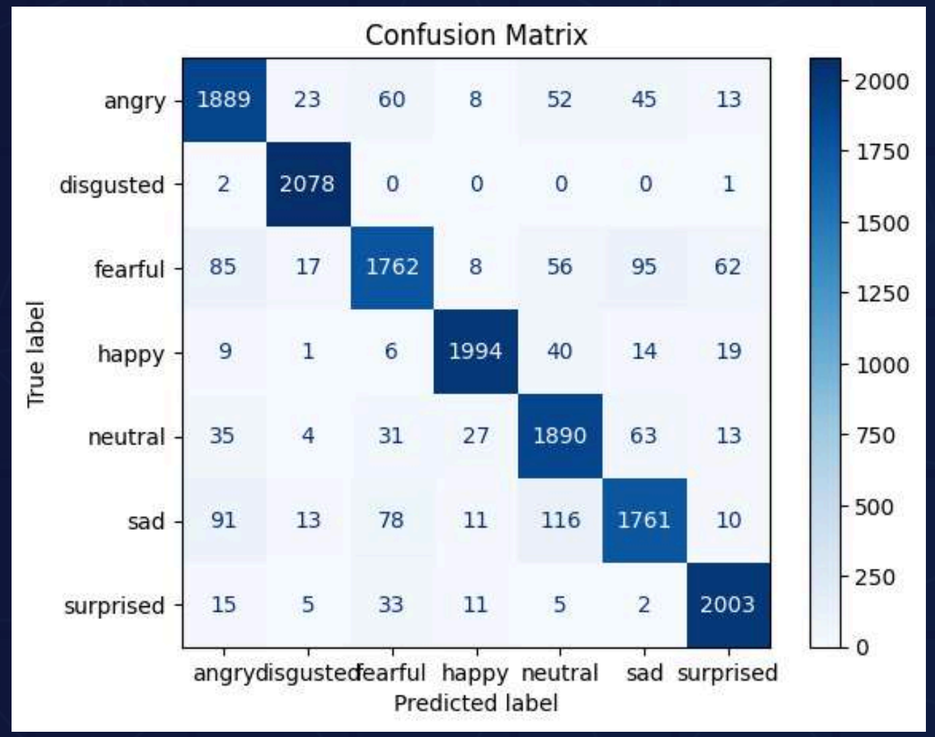
FACIAL EMOTIONAL RECOGNITION RESULT

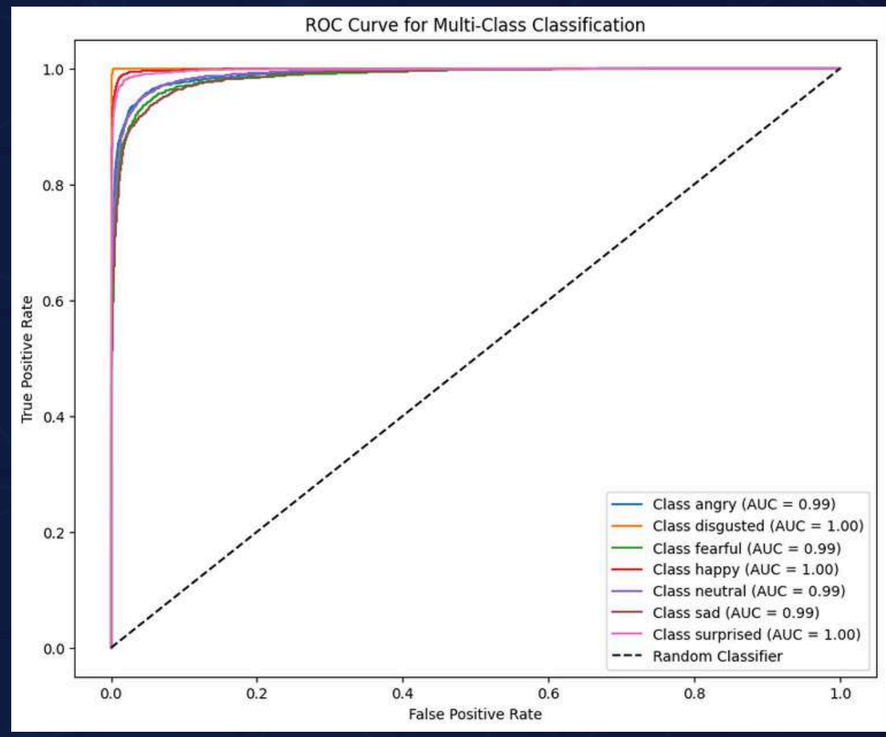
CNN-version3

	precision	recall	f1-score	support
angry	0.89	0.91	0.90	2090
disgusted	0.97	1.00	0.99	2081
fearful	0.91	0.85	0.88	2085
happy	0.97	0.96	0.96	2083
neutral	0.87	0.91	0.89	2063
sad	0.88	0.85	0.87	2080
surprised	0.94	0.97	0.95	2074
accuracy			0.92	14556
macro avg	0.92	0.92	0.92	14556
weighted avg	0.92	0.92	0.92	14556

FACIAL EMOTIONAL RECOGNITION RESULT

CNN-version3

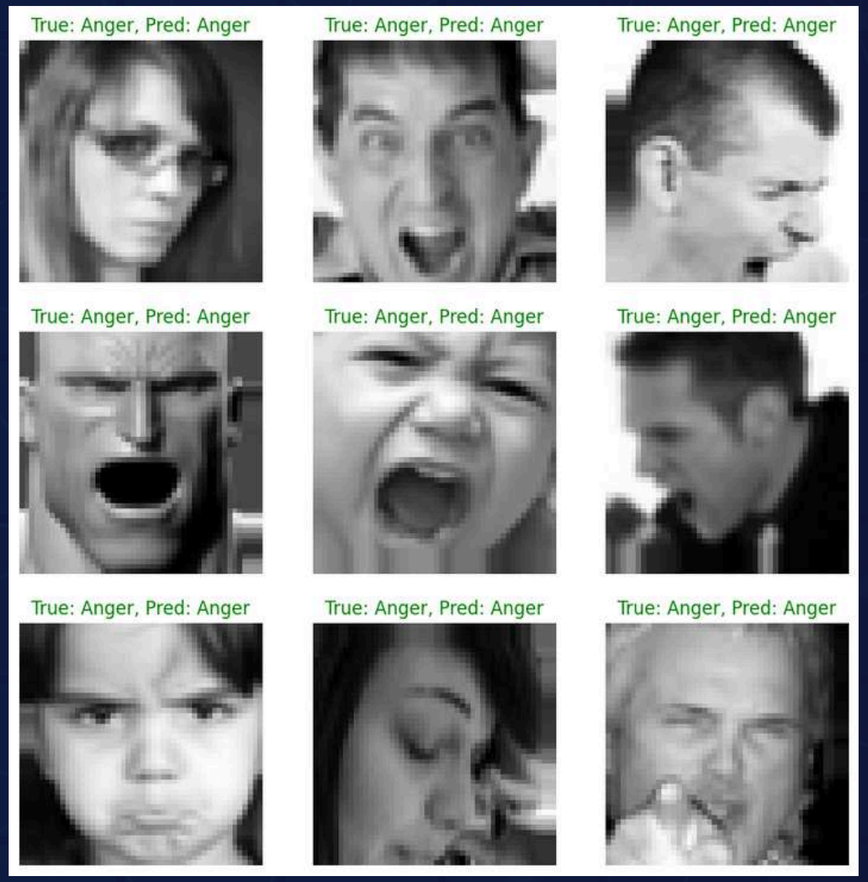




FACIAL EMOTIONAL RECOGNITION RESULT

CNN-version3

Sample of Prediction



CROWD COUNTING MODELS

CROWD COUNTING MODELS

MCNN Model

1. MCNN-version1
2. MCNN-version2
3. MCNN-version3
4. MCNN-version4

INITIAL SETUP PARAMETERS

Hyperparameters							
Model	**Version**	**Epochs**	**Batch Size**	**Learning Rate**	**filters_col1_1**	**filters_col2_1**	**filters_col3_1**
MCNN	1	100	8	0.0005	8	48	72
	2	100	16	0.001	32	64	96
	3	100	16	0.005	8	48	72
	4	150	16	0.001	16	64	24

CROWD COUNTING MODELS

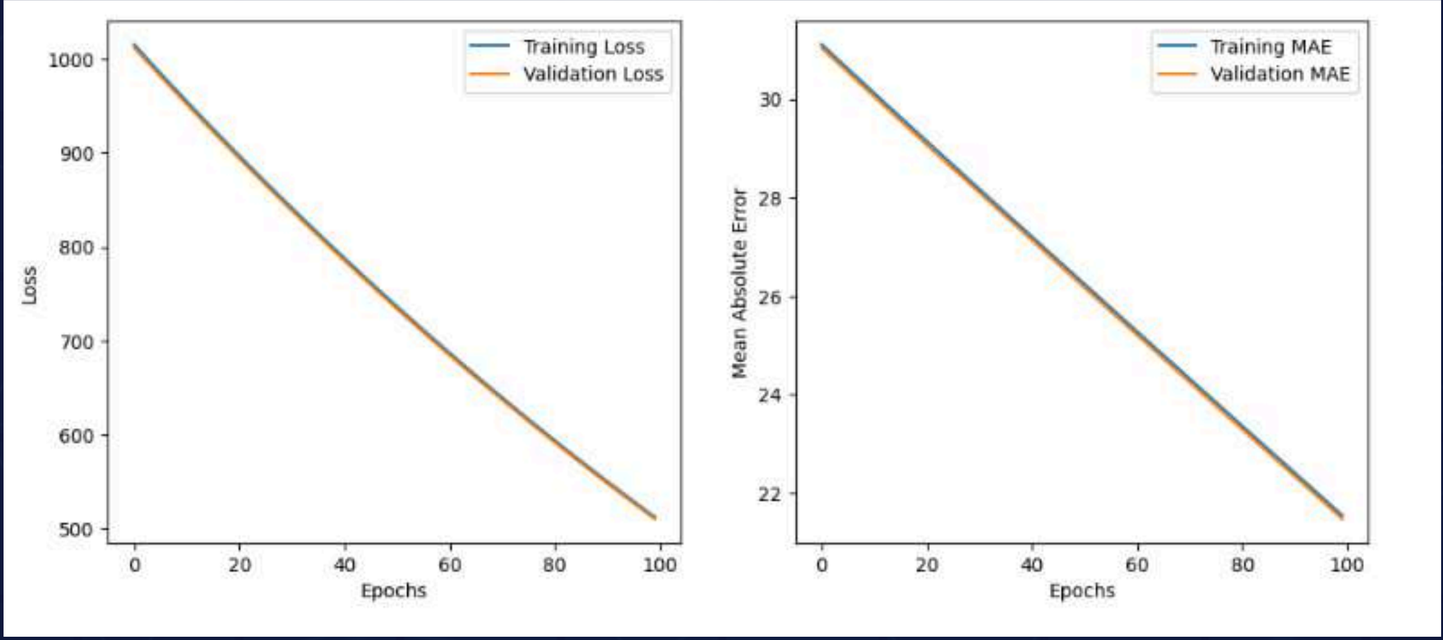
MCNN-version1

```python
# MCNN Model Definition
def create_mcnn_model(input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)):
    input_img = Input(shape=input_shape)

    # Column 1 - Small receptive field
    x1 = layers.Conv2D(16, (9, 9), activation='relu', padding='same')(input_img)
    x1 = layers.MaxPooling2D((2, 2))(x1)
    x1 = layers.Conv2D(32, (7, 7), activation='relu', padding='same')(x1)
    x1 = layers.MaxPooling2D((2, 2))(x1)
    x1 = layers.Conv2D(16, (5, 5), activation='relu', padding='same')(x1)
    x1 = layers.MaxPooling2D((2, 2))(x1)
    x1 = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x1)

    # Column 2 - Medium receptive field
    x2 = layers.Conv2D(20, (7, 7), activation='relu', padding='same')(input_img)
    x2 = layers.MaxPooling2D((2, 2))(x2)
    x2 = layers.Conv2D(40, (5, 5), activation='relu', padding='same')(x2)
    x2 = layers.MaxPooling2D((2, 2))(x2)
    x2 = layers.Conv2D(20, (3, 3), activation='relu', padding='same')(x2)
    x2 = layers.MaxPooling2D((2, 2))(x2)

    # Column 3 - Large receptive field
    x3 = layers.Conv2D(24, (5, 5), activation='relu', padding='same')(input_img)
    x3 = layers.MaxPooling2D((2, 2))(x3)
    x3 = layers.Conv2D(48, (3, 3), activation='relu', padding='same')(x3)
    x3 = layers.MaxPooling2D((2, 2))(x3)
    x3 = layers.Conv2D(24, (3, 3), activation='relu', padding='same')(x3)
    x3 = layers.MaxPooling2D((2, 2))(x3)

    # Concatenate the outputs of all three columns
    x = layers.Concatenate()([x1, x2, x3])
    x = layers.Conv2D(1, (1, 1), activation='relu', padding='same')(x)
    x = layers.GlobalAveragePooling2D()(x)

    # Output layer for crowd count
    output = layers.Dense(1, activation='linear')(x)

    # Build and compile model
    model = models.Model(inputs=input_img, outputs=output)
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error', metrics=['mae'])
    return model
```

CROWD COUNTING RESULT

MCNN-version1

```
Epoch 99/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 51ms/step - loss: 519.2307 - mae: 21.7344 - val_loss: 513.9863 - val_mae: 21.5622
Epoch 100/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 52ms/step - loss: 515.0912 - mae: 21.6389 - val_loss: 509.8804 - val_mae: 21.4668
```



CROWD COUNTING RESULT

MCNN-version1

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 3s 55ms/step
Test MAE: 21.47
```

CROWD COUNTING MODELS

MCNN-version2

```python
# Define the MCNN model
def create_mcnn_model(input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)):
    input_img = Input(shape=input_shape)

    # Column 1
    x1 = layers.Conv2D(16, (9, 9), activation='relu', padding='same')(input_img)
    x1 = layers.MaxPooling2D((2, 2))(x1)
    x1 = layers.Conv2D(32, (7, 7), activation='relu', padding='same')(x1)
    x1 = layers.MaxPooling2D((2, 2))(x1)
    x1 = layers.Conv2D(16, (5, 5), activation='relu', padding='same')(x1)
    x1 = layers.MaxPooling2D((2, 2))(x1)
    x1 = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x1)

    # Column 2
    x2 = layers.Conv2D(20, (7, 7), activation='relu', padding='same')(input_img)
    x2 = layers.MaxPooling2D((2, 2))(x2)
    x2 = layers.Conv2D(40, (5, 5), activation='relu', padding='same')(x2)
    x2 = layers.MaxPooling2D((2, 2))(x2)
    x2 = layers.Conv2D(20, (3, 3), activation='relu', padding='same')(x2)
    x2 = layers.MaxPooling2D((2, 2))(x2)

    # Column 3
    x3 = layers.Conv2D(24, (5, 5), activation='relu', padding='same')(input_img)
    x3 = layers.MaxPooling2D((2, 2))(x3)
    x3 = layers.Conv2D(48, (3, 3), activation='relu', padding='same')(x3)
    x3 = layers.MaxPooling2D((2, 2))(x3)
    x3 = layers.Conv2D(24, (3, 3), activation='relu', padding='same')(x3)
    x3 = layers.MaxPooling2D((2, 2))(x3)

    # Concatenate the outputs of all three columns
    x = layers.Concatenate()([x1, x2, x3])
    x = layers.Conv2D(1, (1, 1), activation='relu', padding='same')(x)
    x = layers.GlobalAveragePooling2D()(x)

    # Output layer
    output = layers.Dense(1, activation='linear')(x)

    # Build and compile model
    model = models.Model(inputs=input_img, outputs=output)
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error', metrics=['mae'])
    return model
```

CROWD COUNTING RESULT

MCNN-version2

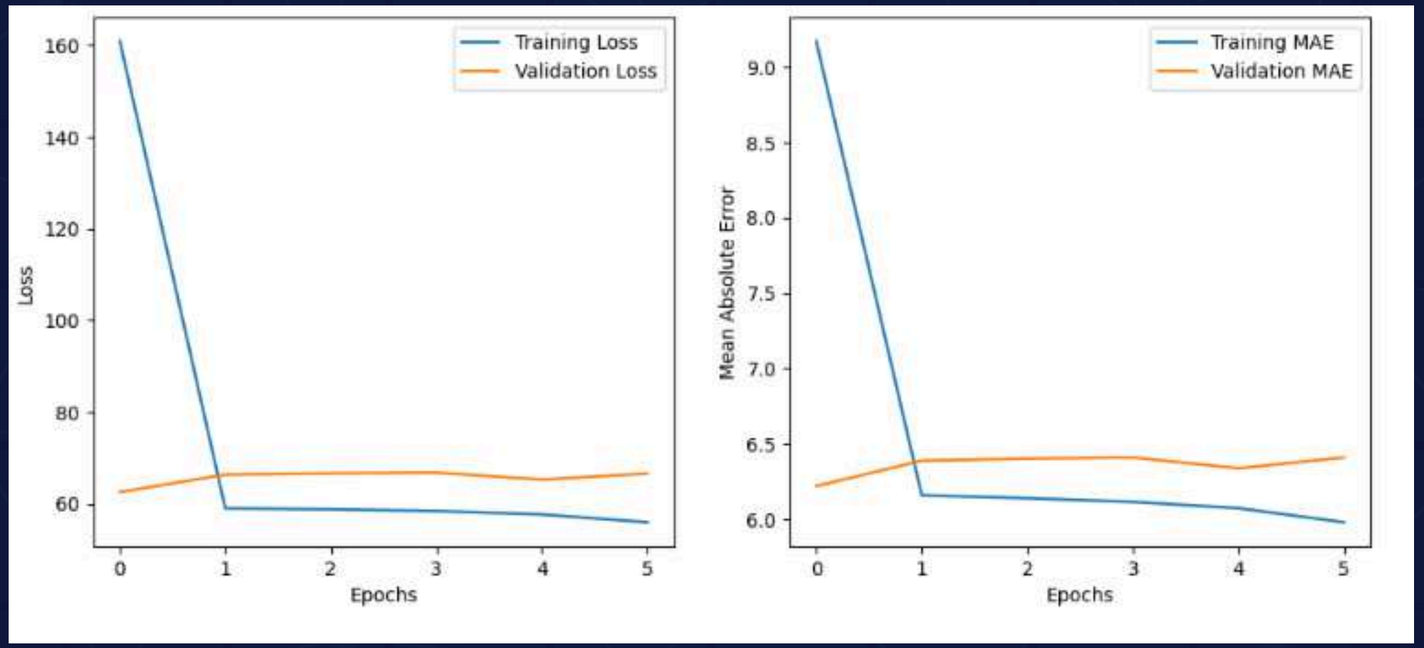
Train with Different Batch Sizes

```
Best Batch Size: 16 with Validation MAE: 6.129660606384277
```

CROWD COUNTING RESULT

MCNN-version2

```
Epoch 5/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 48ms/step - loss: 55.8029 - mae: 6.0067 - val_loss: 65.2022 - val_mae: 6.3383
Epoch 6/100
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 52ms/step - loss: 54.0557 - mae: 5.9048 - val_loss: 66.5136 - val_mae: 6.4102
```



CROWD COUNTING RESULT

MCNN-version2

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 56ms/step
Test MAE with Best Batch Size 16: 6.22
```

CROWD COUNTING MODELS

MCNN-version3

```python
# Define HyperModel for MCNN
def build_mcnn_model(hp):
    input_img = Input(shape=(IMG_HEIGHT, IMG_WIDTH, 3))

    # Column 1 - Small receptive field with tunable filters
    x1 = layers.Conv2D(hp.Int('filters_col1_1', 8, 32, step=8), (9, 9), activation='relu', padding='same')(input_img)
    x1 = layers.MaxPooling2D((2, 2))(x1)
    x1 = layers.Conv2D(hp.Int('filters_col1_2', 16, 64, step=16), (7, 7), activation='relu', padding='same')(x1)
    x1 = layers.MaxPooling2D((2, 2))(x1)
    x1 = layers.Conv2D(16, (5, 5), activation='relu', padding='same')(x1)
    x1 = layers.MaxPooling2D((2, 2))(x1)
    x1 = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x1)

    # Column 2 - Medium receptive field with tunable filters
    x2 = layers.Conv2D(hp.Int('filters_col2_1', 16, 64, step=16), (7, 7), activation='relu', padding='same')(input_img)
    x2 = layers.MaxPooling2D((2, 2))(x2)
    x2 = layers.Conv2D(hp.Int('filters_col2_2', 32, 128, step=32), (5, 5), activation='relu', padding='same')(x2)
    x2 = layers.MaxPooling2D((2, 2))(x2)
    x2 = layers.Conv2D(20, (3, 3), activation='relu', padding='same')(x2)
    x2 = layers.MaxPooling2D((2, 2))(x2)

    # Column 3 - Large receptive field with tunable filters
    x3 = layers.Conv2D(hp.Int('filters_col3_1', 24, 96, step=24), (5, 5), activation='relu', padding='same')(input_img)
    x3 = layers.MaxPooling2D((2, 2))(x3)
    x3 = layers.Conv2D(hp.Int('filters_col3_2', 48, 192, step=48), (3, 3), activation='relu', padding='same')(x3)
    x3 = layers.MaxPooling2D((2, 2))(x3)
    x3 = layers.Conv2D(24, (3, 3), activation='relu', padding='same')(x3)
    x3 = layers.MaxPooling2D((2, 2))(x3)

    # Concatenate the outputs of all three columns
    x = layers.Concatenate()([x1, x2, x3])
    x = layers.Conv2D(1, (1, 1), activation='relu', padding='same')(x)
    x = layers.GlobalAveragePooling2D()(x)

    # Output layer for crowd count
    output = layers.Dense(1, activation='linear')(x)

    # Build and compile model
    model = models.Model(inputs=input_img, outputs=output)
    model.compile(
        optimizer=Adam(learning_rate=hp.Choice('learning_rate', [1e-3, 5e-4, 1e-4])),
        loss='mean_squared_error',
        metrics=['mae']
    )
    return model
```

CROWD COUNTING RESULT

MCNN-version3

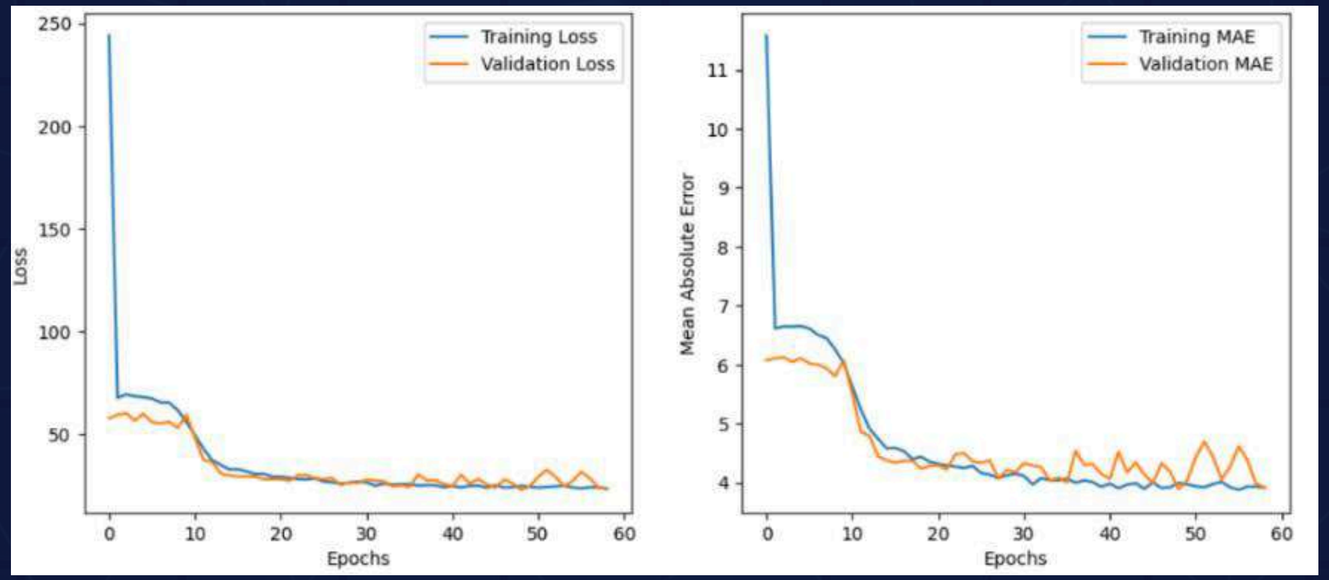
Hyperparameter Tuning

```
Trial 5 Complete [00h 22m 58s]
val_mae: 26.260219573974 61

Best val_mae So Far: 4.1174921989440 92
Total elapsed time: 01h 42m 05s
```

CROWD COUNTING RESULT

MCNN-version3

```
Best Training MAE: 4.0176
Best Validation MAE: 4.3431
```



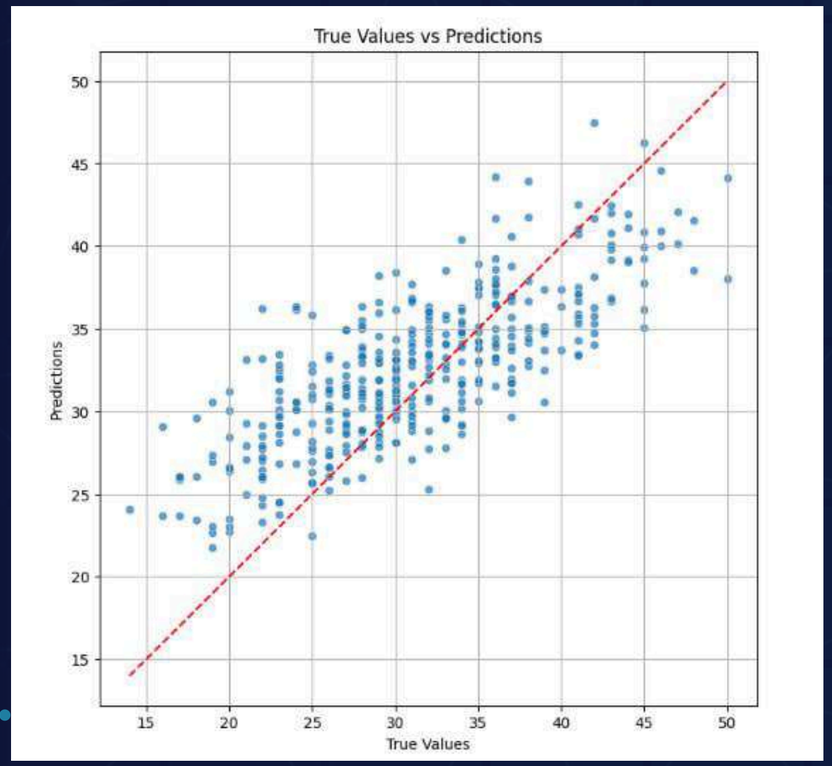
CROWD COUNTING RESULT

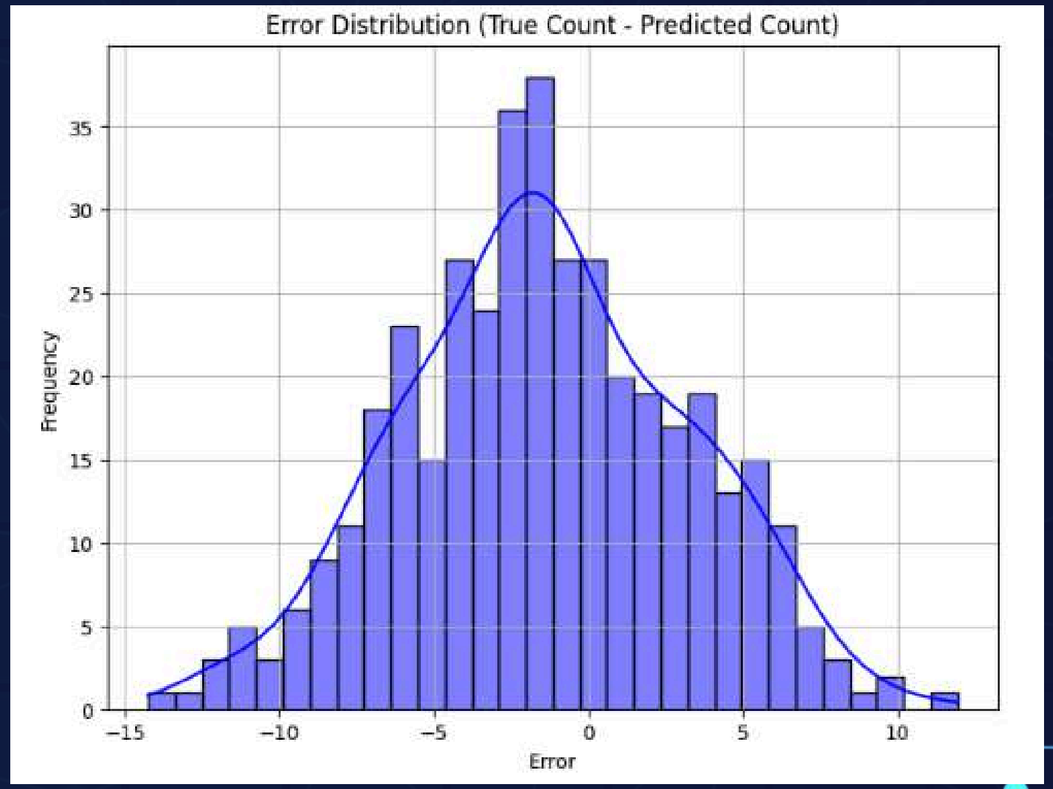
MCNN-version3

```
Model Performance Report
========================
1. Model Architecture:
   - Columns with tunable filters.
   - Optimized hyperparameters:
     - Learning Rate: 0.0005
     - Filters (Column 1, Layer 1): 8
     - Filters (Column 2, Layer 1): 48
     - Filters (Column 3, Layer 1): 96

2. Training Summary:
   - Final Training Loss: 23.6506
   - Final Validation Loss: 23.2556
   - Final Training MAE: 3.9102
   - Final Validation MAE: 3.9135

3. Test Metrics:
   - Mean Absolute Error (MAE): 3.8858
   - Mean Squared Error (MSE): 22.9771
   - R^2 Score: 0.5316
```

CROWD COUNTING RESULT

MCNN-version3





CROWD COUNTING MODELS

MCNN-version4

```python
# Define HyperModel for MCNN
def build_mcnn_model(hp):

    input_img = Input(shape=(IMG_HEIGHT, IMG_WIDTH, 3))

    # Column 1 - Small receptive field with tunable filters
    x1 = layers.Conv2D(hp.Int('filters_col1_1', 8, 32, step=8), (9, 9), activation='relu', padding='same')(input_img)
    x1 = layers.MaxPooling2D((2, 2))(x1)
    x1 = layers.Conv2D(hp.Int('filters_col1_2', 16, 64, step=16), (7, 7), activation='relu', padding='same')(x1)
    x1 = layers.MaxPooling2D((2, 2))(x1)
    x1 = layers.Conv2D(16, (5, 5), activation='relu', padding='same')(x1)
    x1 = layers.MaxPooling2D((2, 2))(x1)
    x1 = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x1)

    # Column 2 - Medium receptive field with tunable filters
    x2 = layers.Conv2D(hp.Int('filters_col2_1', 16, 64, step=16), (7, 7), activation='relu', padding='same')(input_img)
    x2 = layers.MaxPooling2D((2, 2))(x2)
    x2 = layers.Conv2D(hp.Int('filters_col2_2', 32, 128, step=32), (5, 5), activation='relu', padding='same')(x2)
    x2 = layers.MaxPooling2D((2, 2))(x2)
    x2 = layers.Conv2D(20, (3, 3), activation='relu', padding='same')(x2)
    x2 = layers.MaxPooling2D((2, 2))(x2)

    # Column 3 - Large receptive field with tunable filters
    x3 = layers.Conv2D(hp.Int('filters_col3_1', 24, 96, step=24), (5, 5), activation='relu', padding='same')(input_img)
    x3 = layers.MaxPooling2D((2, 2))(x3)
    x3 = layers.Conv2D(hp.Int('filters_col3_2', 48, 192, step=48), (3, 3), activation='relu', padding='same')(x3)
    x3 = layers.MaxPooling2D((2, 2))(x3)
    x3 = layers.Conv2D(24, (3, 3), activation='relu', padding='same')(x3)
    x3 = layers.MaxPooling2D((2, 2))(x3)

    # Concatenate the outputs of all three columns
    x = layers.Concatenate()([x1, x2, x3])
    x = layers.Conv2D(1, (1, 1), activation='relu', padding='same')(x)
    x = layers.GlobalAveragePooling2D()(x)

    # Output layer for crowd count
    output = layers.Dense(1, activation='linear')(x)

    # Build and compile model
    model = models.Model(inputs=input_img, outputs=output)
    model.compile(
        optimizer=Adam(learning_rate=hp.Choice('learning_rate', [1e-3, 5e-4, 1e-4])),
        loss='mean_squared_error',
        metrics=['mae']
    )
    return model
```

CROWD COUNTING RESULT

MCNN-version4

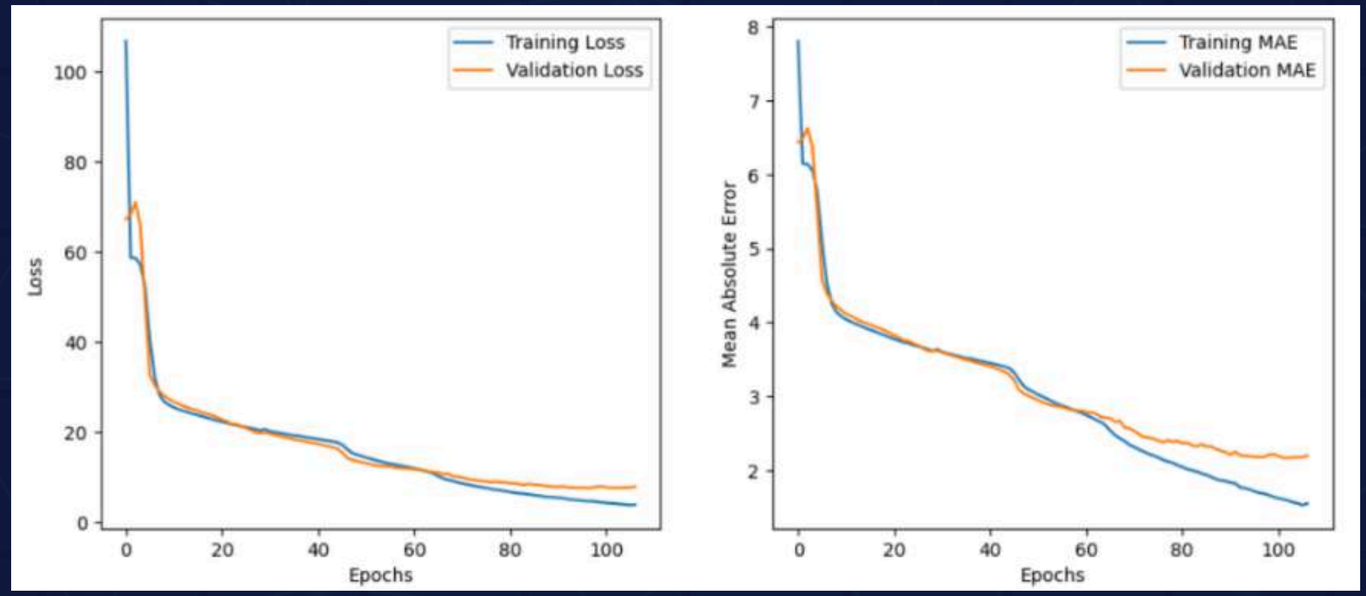
Hyperparameter Tuning

```
Trial 5 Complete [00h 02m 47s]
val_mae: 3.965034246444702

Best val_mae So Far: 3.686537504196167
Total elapsed time: 00h 26m 32s
```

CROWD COUNTING RESULT

MCNN-version4

```
Best Training MAE: 1.5293
Best Validation MAE: 2.1674
```



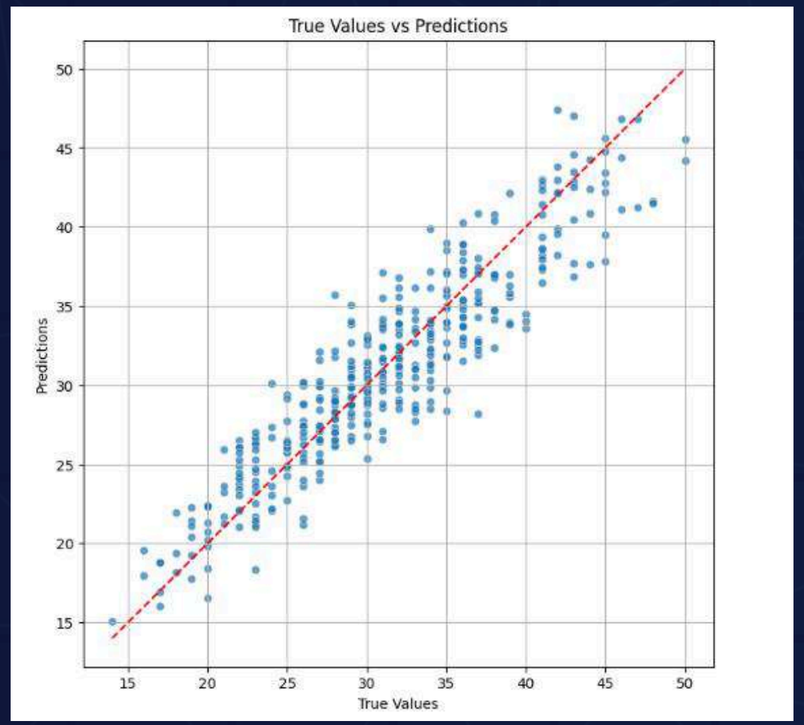
CROWD COUNTING RESULT

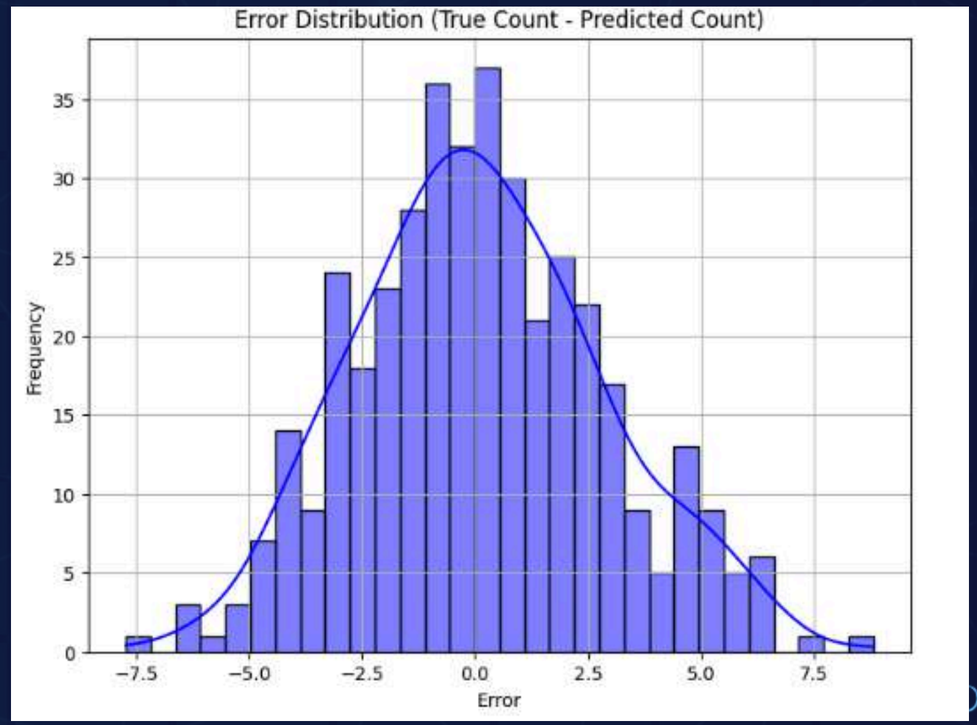
MCNN-version4

```
Model Performance Report
========================
1. Model Architecture:
   - Columns with tunable filters.
   - Optimized hyperparameters:
     - Learning Rate: 0.001
     - Filters (Column 1, Layer 1): 16
     - Filters (Column 2, Layer 1): 64
     - Filters (Column 3, Layer 1): 24

2. Training Summary:
   - Final Training Loss: 3.8355
   - Final Validation Loss: 7.8120
   - Final Training MAE: 1.5505
   - Final Validation MAE: 2.1976

3. Test Metrics:
   - Mean Absolute Error (MAE): 2.1793
   - Mean Squared Error (MSE): 7.5225
   - R^2 Score: 0.8467
```

CROWD COUNTING RESULT

MCNN-version4





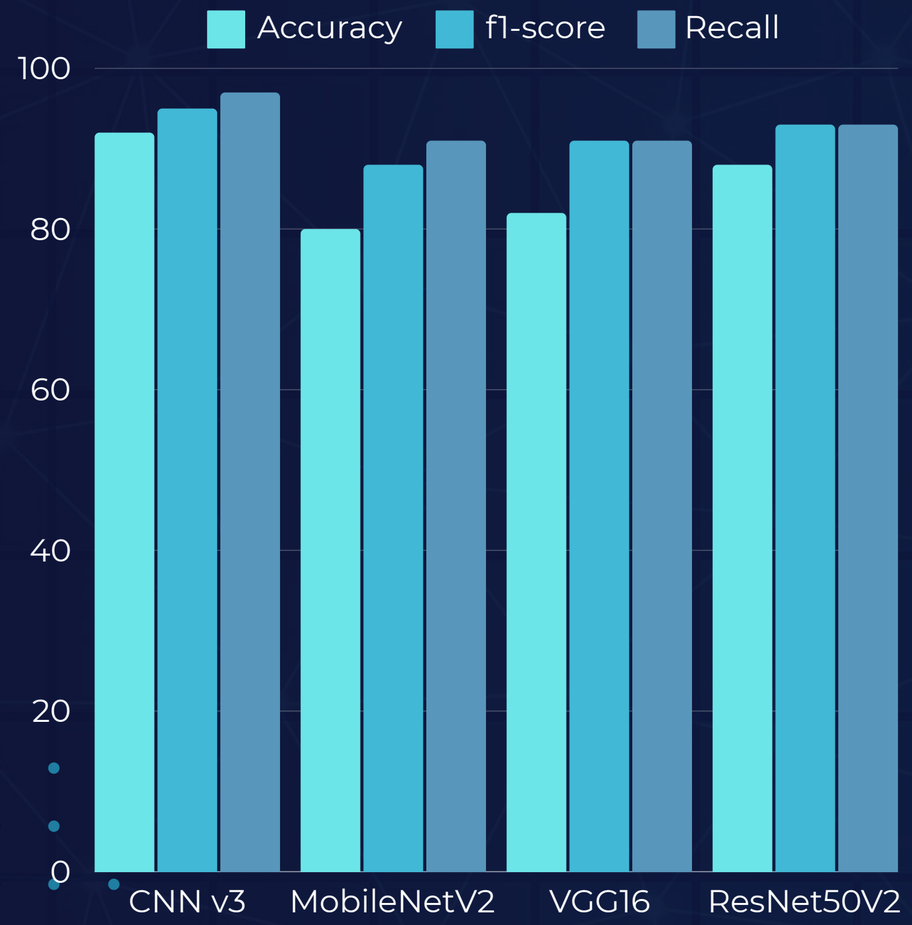
CROWD COUNTING RESULT

MCNN-version4

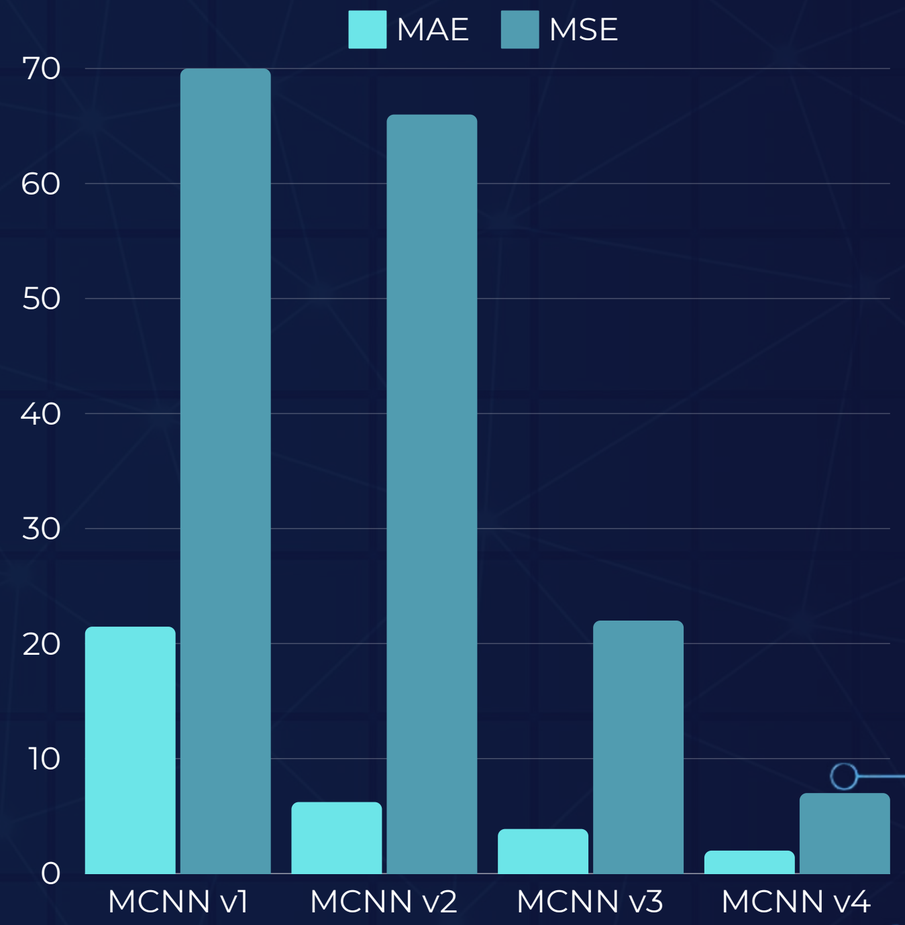
03 RESULTS AND DISCUSSIONS

RESULTS

Facial Emotion Detection



Crowd Counting



DISCUSSION

- The results of experiments using CNN, ResNet50, VGG16, MobileNetv2, and MCNN, showed that CNN outperformed all other models by 92% in Facial Emotional Detection.

- The Multi-Column Convolutional Neural Network (MCNN) model in crowd counting in Version 4 showed an 84% for R2 and 2.17 for MAE.

OBJECTIVE ACHIEVED

After extensive research and comparison between different deep learning models, we succeeded in:

1. Building a model for facial emotion detection
2. Building a model for crowd counting
3. Integrating the two models and displaying their results in a dashboard

04 CONCLUSION & FUTURE WORK

DIFFICULTIES & LIMITATIONS

Data Quality and Diversity	Limited dataset variability (e.g., age, ethnicity, lighting) hindered the model's ability to generalize to real-world scenarios, emphasizing the need for more representative data.
Model Optimization:	Fine-tuning models to prevent overfitting was slow and iterative but crucial for balancing accuracy and reliability.
Crowd Detection	Dynamic environments with movement, overlapping objects, and behavior variations reduced crowd recognition accuracy.
Time Constraints	Tight deadlines limited opportunities for dataset expansion, broader testing, and advanced optimization methods.

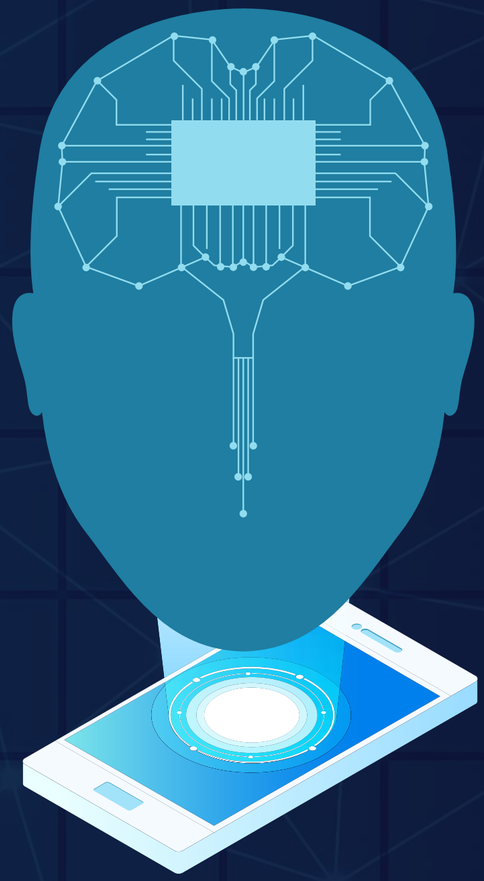
CONCLUSION

FUTURE WORK

Multi-Modal Integration	Combine facial expressions with audio and text data to improve emotion recognition accuracy.
Advanced Algorithms	Explore 3D imaging or thermal cameras for enhanced crowd detection in complex environments.
Real-World Testing	Validate model performance in diverse conditions, such as public events or urban settings.
Ethical Considerations	Address privacy and security concerns as facial recognition and crowd detection technologies expand.
Enhanced Applicability	Focus on improving the effectiveness and practical use of these technologies in real-world scenarios.

THANK YOU

Do you have questions?



وفي ختام رحلتنا في هذا المشروع، نحتفي بنجاحنا كخطوة أولى نحو تحقيق أحلامنا، مستندين إلى شغفنا بالعلم وعزمنا على الإبداع والتطوير. فالنهايات ليست إلا بدايات جديدة.

REFERENCE

[1] N. e. a. Kumar HN, “Automatic facial expression recognition combining texture and shape features from prominent facial regions.,” IET Image Processing, 2023.

[2] A. e. a. Patwal, “Crowd counting analysis using deep learning: A critical review.,” Procedia Computer Science, 2023.

[3] S. Sakib, N. Ahmed, A. Kabir, J. Ahmed and H. Ahmed, “An overview of convolutional neural network: Its architecture and applications.,” p. 6, 2019.

[4] Y. Khaireddin and Z. Chen, “Facial emotion recognition: State of the art performance on FER2013.,” arXiv, 2021.

[5] C. Dalvi, M. Rathod, S. Patil, S. Gite and K. Kotecha, “A survey of ai-based facial emotion recognition: Features, ml & dl techniques, age-wise datasets and future directions.,” Ieee Access., 2021.

REFERENCE

[6] A. Chen, W. Su and Z. Wang, “Crowd counting with crowd attention convolutional neural network,” Neurocomputing, vol. 382, pp. 210-220, 2020.

[7] M. Sivaram, V. Porkodi, A. S. Mohammed and V. Manikandan , “DETECTION OF ACCURATE FACIAL DETECTION USING HYBRID DEEP,” ICTACT Journal on Soft Computing, 2019.

[8] S. S. Mohamed, W. A. Mohamed, . A. T. Khalil and A. S. Mohra, “Deep Learning Face Detection and Recognition,” International journal of electronics and telecommunications, 2019.

[9] K. Sarvakar, . R. Senkamalavalli , . S. Raghavendra, . J. Santosh Kumar and R. Manjunat, “Facial emotion recognition using convolutional neural networks,” Proceedings, pp. 3560-3564, 2023.

[10] G. Gao, . J. Gao, Q. Liu, Q. Wang and Y. Wang, “CNN-based Density Estimation and Crowd,” arXiv preprint arXiv, 2020.

REFERENCE

11] D. S. Shuji, Face expression image detection and recognition based on big data technology, International Journal of Intelligent Networks 4, 2023.

[12] L. Yuting , Z. Wang, M. Shi, S. Sato, Q. Zhao and H. Yang, Discovering regression-detection bi-knowledge transfer for unsupervised cross-domain crowd counting., Neurocomputing 494, 2022.

[13] V. A. Sindagi and V. M. Patel, “A survey of recent advances in CNN-based single image crowd counting and density estimation,” Pattern Recognition Letters, vol. 107, pp. 3-16, 2018.

[14] Javatpoint, "Python OS Module," 2023. [Online]. Available: https://www.javatpoint.com/python-os-module

[15] The Pandas Development Team, "pandas," PyPI, Sep. 20, 2024. [Online]. Available: https://pypi.org/project/pandas/.
[ww.w3schools.com/python/numpy/numpy_intro.asp.

REFERENCE

[16] W3Schools, "Introduction to NumPy," 2023. [Online]. Available: https://www.w3schools.com/python/numpy/numpy_intro.asp.

[17] TensorFlow, "TensorFlow," 2023. [Online]. Available: https://www.tensorflow.org/?hl=ar.

[18] TensorFlow, "Keras: The high-level API for TensorFlow," 2023. [Online]. Available: https://www.tensorflow.org/guide/keras.

[19] ActiveState, "What Is Pyplot In Matplotlib," 15-Jan-2021. [Online]. Available: https://www.activestate.com/resources/quick-reads/what-is-pyplot-in-matplotlib/.

[20] Pathmind, "Confusion Matrix, Precision, Recall, and F1-Score Explained," 2024. [Online]. Available: https://wiki.pathmind.com.

[21] GeeksforGeeks, "Performance Metrics in Machine Learning," 2024. [Online]. Available: https://www.geeksforgeeks.org.