Faculty of Engineering and Technology			
Ramaiah University of Applied Sciences			
Department	Artificial Intelligence and Machine Learning	Programme	B. Tech
Semester/Batch	05 / 2023		
Course Code	AIC205A	Course Title	Machine Learning-2
Course Leader	Mrs. Shaista Tarannum		

Assignment			
Register No.	23ETAI410035 23ETAI410023 23ETAI410028	**Name of the Student:** **Group No:**	Pratyush Pattanaik Manoj K Bhandare Narahari N Group 11

Sections		**Marking Scheme**	**Marks**		
			Max Marks	**First Examiner Marks**	**Moderator**
	Part – 1	**TechA Deep Learning and Neural Network Certification (Infosys Springboard)**	**15**		
	1.1	Completion of Infosys Springboard Certification	9		
	1.2	Certification Exam Score 35–59 marks	12		
	1.3	Certification Exam Score 60 and above	15		
	Part – 2	**End-to-End Deep Learning Workflow (Manual and AI-Assisted)**	**15**		
		Max Marks	**30**		

Course Marks Tabulation				
Component- CET B Assignment	**First Examiner**	**Remarks**	**Second Examiner**	**Remarks**
1				
2				
Total Marks				

Signature of First Examiner **Signature of Second Examiner**

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Sections	Marking Scheme	Max Marks	Marks Scored
Part – 1	**TechA Deep Learning & Neural Network Certification (Infosys Springboard)**	**15**	
1.1	Completion of course (upload certificate).	9	
1.2	Certification score 35–59 marks.	12	
1.3	Certification score ≥ 60 marks.	15	
Part – 2	**End-to-End Deep Learning Workflow (Manual & AI-Assisted)**	**15**	
2.1	Manual Deep Learning Workflow – Build CNN using Python (TensorFlow/Keras).	10	
2.2	Conceptual understanding (libraries, ANN/CNN/RNN).	3	
2.3	Data preparation & preprocessing (load, normalize, augment).	4	
2.4	CNN development & evaluation (metrics, plots, interpretation).	8	
2.5	AI Tool Usage (ChatGPT, Bard, Copilot) – document prompts & outputs.	2	
2.6	Comparative analysis – manual vs. AI implementation.	2	
2.7	Reflection & ethical awareness (trust, integrity, screenshots).	1	
Total Marks		**30**	

Please note:

1. Documental evidence for all the components/parts of the assessment such as the reports, photographs, laboratory exam / tool tests are required to be attached to the assignment report in a proper order.
2. The First Examiner is required to mark the comments in RED ink and the Second Examiner's comments should be in GREEN ink.
3. The marks for all the questions of the assignment have to be written only in the **Component – CET B: Assignment** table.
4. If the variation between the marks awarded by the first examiner and the second examiner lies within +/- 3 marks, then the marks allotted by the first examiner is considered to be final. If the variation is more than +/- 3 marks then both the examiners should resolve the issue in consultation with the Chairman BoE.

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Assignment

Instructions to students:

1. The assignment consists of **2 parts**:
   - **Part 1:** *TechA Deep Learning and Neural Network Certification (Infosys Springboard)*
   - **Part 2:** *End-to-End Deep Learning Workflow (Manual and AI-Assisted CNN Project)*
2. **Part 1** focuses on completing the **Infosys Springboard Certification** on *Deep Learning and Neural Networks*, which introduces the fundamentals of ANN, CNN, and frameworks like Keras and TensorFlow.
   Marking Criteria for Part 1 (15 Marks):
   - Completion of all course modules: **9 Marks**
   - Certification exam score **35–59 marks**: **12 Marks total**
   - Certification exam score **60 and above**: **Full 15 Marks**
3. **Part 2** requires students to implement a **Convolutional Neural Network (CNN)** model using a **Kaggle dataset**, demonstrating the deep learning workflow both manually and using AI tools.
   - **Part 2(a):** Must be implemented manually using Python (TensorFlow/Keras) **without AI assistance**.
   - **Part 2(b):** Must demonstrate the same workflow using **AI tools** such as ChatGPT, Bard (Gemini), GitHub Copilot, or similar platforms.
4. Each group/student must use **one assigned Kaggle dataset** from the provided list or any other **faculty-approved dataset** relevant to CNN-based deep learning tasks.
5. The **maximum marks** for the assignment are **30**.
   - **Part 1:** 15 Marks (Infosys Certification)
   - **Part 2:** 15 Marks (Manual + AI CNN Project)
6. The assignment must be **neatly word-processed** as per the prescribed format. Handwritten submissions will not be accepted.
7. The **maximum number of pages** for the report should be **10**, including screenshots, outputs, and references.
8. The following must be included in the final submission:
   - Infosys Springboard certificate (with score proof if available)
   - CNN model code and visual outputs (accuracy/loss plots, confusion matrix)
   - AI tool screenshots, prompts, and reflection answers
9. The **printed copy** of the assignment must be submitted to the course leader on or before the due date.
10. **Submission Date:** 20-12-2025
11. **Late submissions** after the due date will **not be accepted** under any circumstances.
12. All sources (datasets, research articles, AI-generated content, or code snippets) used in the assignment must be **properly cited and referenced** in the report.
13. **Marks will be awarded** only for sections and subsections **clearly labeled and addressed** as per the assignment structure.
14. Students are expected to maintain **academic integrity** and demonstrate **ethical use of AI tools** throughout the assignment process.

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Preamble:

This assignment aims to strengthen students' understanding of Machine Learning, Deep Learning and Neural Networks through both guided certification and practical project implementation. Students will complete an online certification course on Infosys Springboard and develop a CNN-based image classification project using Kaggle datasets. The activity enhances theoretical knowledge and practical skills in model building, training, and evaluation using frameworks like TensorFlow and Keras.

Overview (30 Marks)

This assignment is designed to provide students with both theoretical and practical experience in Deep Learning.
It consists of two integrated parts:

- **Part 1** focuses on completing the *TechA Deep Learning and Neural Network Certification* from Infosys Springboard, which introduces foundational concepts such as Artificial Neural Networks (ANN), Convolutional Neural Networks (CNN), and Keras/TensorFlow.
- **Part 2** requires students to apply this knowledge by designing a CNN-based image classification model manually and optionally with AI tool support, encouraging critical reflection on the responsible use of AI in learning.

Part 1 – TechA Deep Learning and Neural Network Certification (15 Marks)

Students are required to complete the **TechA Deep Learning and Neural Network Certification** available on **Infosys Springboard**.
This certification introduces essential concepts and tools such as ANN, CNN, data augmentation, Keras, TensorFlow, R, and Python.

Certification Link:
https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_013823396111933440557_shared/overview#iss=https:/infyspringboard.onwingspan.com/auth/realms/infyspringboard&iss=https://infyspringboard.onwingspan.com/auth/realms/infyspringboard

Steps to Complete:

1. Enroll in the course using the link above.
2. Complete all pre-content modules:
    - *Deep Learning Neural Network with R* (3h 26m)
    - *Deep Learning Architecture for Building Artificial Neural Networks* (2h 42m)
    - *Deep Learning with Keras* (2h 21m)
3. On completion, take the certification exam, **download the certificate** from the Infosys Springboard portal.
4. Submit the **certificate screenshot or PDF** as proof to earn 15 marks.

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Part 2 – End-to-End Deep Learning Workflow (Manual and AI-Assisted CNN Project) (15 Marks)

This part of the assignment will strengthen your understanding of CNNs through hands-on data preprocessing, model construction, evaluation, and visualization.
You will also critically analyze the usefulness, reliability, and ethical implications of using AI tools in coding.

No.	Dataset Title	Kaggle Link
11	Human Emotion Detection from Images	https://www.kaggle.com/datasets/msambare/fer2013

Part / Section	Task Description	Marks
2.1 Manual Deep Learning Workflow	**Objective: Apply concepts learned from the TechA certification to build a CNN model manually using Python (TensorFlow/Keras).**	**10**
2.2 Conceptual Understanding	List essential Python libraries used in Deep Learning (TensorFlow, Keras, NumPy, Matplotlib, OpenCV, etc.) and explain the purpose of each in one line. Briefly describe different neural network types (ANN, CNN, RNN) and where each is used.	**3**
2.3 Data Preparation and Preprocessing	a) Define the problem to be solved using the selected Kaggle dataset (e.g., classification or detection). b) Explain dataset structure, features, and labels. c) Write Python code to: d) Load and explore the dataset e) Preprocess the data (resizing, normalization, augmentation).	**4**
2.4 CNN Model Development and Evaluation	a) Build a CNN architecture manually using Keras/TensorFlow. b) Train and evaluate the model using metrics such as Accuracy, Precision, Recall, and F1-Score. c) Create at least three visualizations including one evaluation plot (e.g., confusion matrix or accuracy/loss curve). d) Interpret results: identify if the model overfits, underfits, or generalizes well. e) Submit complete code, graphs, and model summary.	**8**
Deep Learning Workflow Using AI Tools	**Objective: Use AI tools to assist in designing, coding, and analyzing your CNN model. Compare the process and outputs to your manual implementation.**	**5**

2.5 AI Tool Usage	Use one or more AI tools (ChatGPT, Bard/Gemini, GitHub Copilot, etc.) to generate or optimize your CNN code. a) Clearly mention the AI tool used. b) Document prompts/inputs and generated outputs/code suggestions (include screenshots).	**2**
2.6 Comparative Analysis	Compare AI-assisted and manual implementations: a) Differences in code structure, performance, or efficiency. b) Advantages and drawbacks of using AI tools.	**2**

Part / Section	Task Description	Marks
	c) Did AI help in debugging or improving your understanding?	
2.7 Reflection & Ethical Awareness	Answer briefly in 5–6 lines: a) How was the experience different from doing it manually? 2. b) Do you trust AI-generated code completely? Why or why not? c) Mention any ethical or academic integrity concerns about AI-assisted work. *(Attach screenshots of AI conversations and model outputs.)*	**1**
Total Marks		**15**

Submission Details

- **Submission Date:** 20-12-2025
- **Total Marks:** 30
- **Include:**
  - Infosys Springboard certification screenshot (Part 1 evidence)
  - CNN model code, evaluation plots, and model summary
  - AI tool usage documentation and reflection answers
- **Late submissions will not be accepted.**
- The report must be **word-processed**, **neatly formatted**, and limited to **10 pages**.
- Ensure proper **citations and dataset references** are included.

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

2.1 Manual Deep Learning Workflow

Objective: - To Build a Convolutional Neural Network (CNN) using Python and Keras/TensorFlow to classify facial emotions on the FER2013 Kaggle dataset into seven categories: Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral. This is a supervised image classification problem.

2.2 Conceptual Understanding

Essential Python libraries

- **TensorFlow / Keras** – Deep learning framework used to define, train, and evaluate the CNN model.
- **NumPy** – Efficient array library for numerical operations, reshaping image tensors, and label processing.
- **Matplotlib / Seaborn** – Visualization libraries used to plot accuracy/loss curves and confusion matrices.
- **OpenCV (cv2)** – Image processing library for reading, resizing, and converting images for manual tests.
- **scikit-learn** – Provides classification_report and confusion_matrix for computing precision, recall, and F1-scores.

Neural network types

- **ANN (Artificial Neural Network):** Fully connected networks; good for tabular data and simple classification or regression tasks.
- **CNN (Convolutional Neural Network):** Uses convolution and pooling layers to learn spatial patterns from images; ideal for image classification, detection, and recognition.
- **RNN (Recurrent Neural Network):** Processes sequences with temporal dependencies (e.g., text, speech, time series) using recurrent connections.

2.3 Data Preparation and Preprocessing

a) Problem definition

Given a 48×48 grayscale image of a human face from the FER2013 dataset, we need to predict which of the 7 discrete emotions it expresses.

b) Dataset structure, features, and labels

- **Source:** FER2013 Kaggle dataset (pre-split into train and test folders).
- **Structure:**
  - /train – 28,709 images across 7 emotion subfolders.
  - /test – 7,178 images across the same 7 subfolders.
- **Input features:** 48×48 pixel **grayscale** face images (intensity values 0–255).
- **Labels:** Emotion classes: Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral.

c–e) Python code: load, explore, and preprocessing:

Loading the dataset from google drive on google colab

```
[1]
✓ 36s

import os
import zipfile
from google.colab import drive

drive.mount('/content/drive')
zip_path = '/content/drive/MyDrive/FER2/archive.zip'

TRAIN_DIR = '/content/train'
TEST_DIR = '/content/test'

if not os.path.exists(TRAIN_DIR):
    if os.path.exists(zip_path):
        print("Found zip file at:", zip_path)
        print("Unzipping dataset...")

        with zipfile.ZipFile(zip_path, 'r') as zip_ref:
            zip_ref.extractall('/content')

        print("Unzipping complete! Folders in /content:", os.listdir('/content'))
    else:
        print(f"ERROR: Zip file not found at {zip_path}.")
else:
    print("Dataset already exists in /content/train. Skipping unzip.")

Mounted at /content/drive
Found zip file at: /content/drive/MyDrive/FER2/archive.zip
Unzipping dataset...
Unzipping complete! Folders in /content: ['.config', 'train', 'drive', 'test', 'sample_data']
```

After this is done we then move to start loading the dataset into the google colab cell block and performing necessary exploration and preprocessing (we perform augmentation here).

RAMAIAH UNIVERSITY
OF APPLIED SCIENCES

```python
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.optimizers.schedules import CosineDecay
import numpy as np
import matplotlib.pyplot as plt
import os

BATCH_SIZE = 64
IMG_SIZE = (48, 48)
TRAIN_DIR = '/content/train'
TEST_DIR = '/content/test'

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    brightness_range=[0.9, 1.1]
)

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    TRAIN_DIR, target_size=IMG_SIZE, batch_size=BATCH_SIZE,
    color_mode='grayscale', class_mode='categorical'
)

validation_generator = val_datagen.flow_from_directory(
    TEST_DIR, target_size=IMG_SIZE, batch_size=BATCH_SIZE,
    color_mode='grayscale', class_mode='categorical', shuffle=False
)
EPOCHS = 80
```

```
Found 28709 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
```

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

2.4 CNN Model Development and Evaluation

After the above cell block we run the following code: -

```
39
40 class LRTraker(tf.keras.callbacks.Callback):
41     def on_epoch_end(self, epoch, logs=None):
42         lr = self.model.optimizer.learning_rate
43         if callable(lr):
44             current_lr = lr(self.model.optimizer.iterations)
45         else:
46             current_lr = lr
47         print(f" - End of epoch {epoch+1} | learning_rate: {float(current_lr):.6f}")
48
49 def build_leaky_resnet():
50     inputs = layers.Input(shape=(48, 48, 1))
51     x = layers.Conv2D(64, 3, padding='same', kernel_initializer='he_normal')(inputs)
52     x = layers.BatchNormalization()(x); x = layers.LeakyReLU(alpha=0.1)(x)
53
54     def res_se_block(it, f):
55         sc = layers.Conv2D(f, 1, padding='same')(it) if it.shape[-1] != f else it
56         x = layers.Conv2D(f, 3, padding='same', kernel_initializer='he_normal')(it)
57         x = layers.BatchNormalization()(x); x = layers.LeakyReLU(0.1)(x)
58         x = layers.Conv2D(f, 3, padding='same')(x); x = layers.BatchNormalization()(x)
59
60         se = layers.GlobalAveragePooling2D()(x)
61         se = layers.Reshape((1, 1, f))(se)
62         se = layers.Dense(f // 16, activation='relu')(se)
63         se = layers.Dense(f, activation='sigmoid')(se)
64         x = layers.Multiply()([x, se])
65         return layers.LeakyReLU(0.1)(layers.Add()([x, sc]))
66
67     x = res_se_block(x, 64); x = layers.MaxPooling2D(2)(x); x = layers.Dropout(0.2)(x)
68     x = res_se_block(x, 128); x = layers.MaxPooling2D(2)(x); x = layers.Dropout(0.3)(x)
69     x = res_se_block(x, 256); x = layers.MaxPooling2D(2)(x); x = layers.Dropout(0.3)(x)
70     x = res_se_block(x, 512); x = layers.MaxPooling2D(2)(x); x = layers.Dropout(0.4)(x)
71
72     x = layers.GlobalAveragePooling2D()(x)
73     x = layers.Dense(512, kernel_initializer='he_normal')(x)
74     x = layers.BatchNormalization()(x); x = layers.LeakyReLU(alpha=0.1)(x); x = layers.Dropout(0.5)(x)
75     return models.Model(inputs=inputs, outputs=layers.Dense(7, activation='softmax')(x))
76
```

```
76
77 steps_per_epoch = train_generator.samples // BATCH_SIZE
78 lr_schedule = CosineDecay(initial_learning_rate=1e-3, decay_steps=EPOCHS * steps_per_epoch, alpha=1e-5)
79
80 model = build_leaky_resnet()
81 model.compile(optimizer=optimizers.Adam(learning_rate=lr_schedule), loss='categorical_crossentropy', metrics=['accuracy'])
82
83 callbacks = [
84     ModelCheckpoint('fer_best_colab.keras', save_best_only=True, monitor='val_accuracy', mode='max'),
85     EarlyStopping(monitor='val_accuracy', patience=15, restore_best_weights=True),
86     LRTraker()
87 ]
88
89 print("Starting Training...")
90 history = model.fit(train_generator, epochs=EPOCHS, validation_data=validation_generator, callbacks=callbacks)
91
92 def plot_results(history):
93     plt.figure(figsize=(14, 5))
94     plt.subplot(1, 2, 1)
95     plt.plot(history.history['accuracy'], label='Train Accuracy'); plt.plot(history.history['val_accuracy'], label='Val Accuracy')
96     plt.title('Accuracy'); plt.legend()
97     plt.subplot(1, 2, 2)
98     plt.plot(history.history['loss'], label='Train Loss'); plt.plot(history.history['val_loss'], label='Val Loss')
99     plt.title('Loss'); plt.legend()
100     plt.show()
101
102 plot_results(history)
103
```

```
103
104 print("\n--- Running Final 15-Round TTA ---")
105 tta_datagen = ImageDataGenerator(rescale=1./255, horizontal_flip=True, width_shift_range=0.1, height_shift_range=0.1, zoom_range=0.1)
106 tta_gen = tta_datagen.flow_from_directory(TEST_DIR, target_size=IMG_SIZE, batch_size=BATCH_SIZE, color_mode='grayscale', class_mode='categorical', shuffle=False)
107
108 preds = []
109 for i in range(15):
110     tta_gen.reset()
111     preds.append(model.predict(tta_gen, verbose=1))
112
113 final_acc = np.sum(np.argmax(np.mean(preds, axis=0), axis=1) == tta_gen.classes) / len(tta_gen.classes)
114 print(f"\n>>> FINAL VALIDATION ACCURACY WITH 15-ROUND TTA: {final_acc * 100:.2f}% <<<")
115
```

Output: -

For the first few epochs

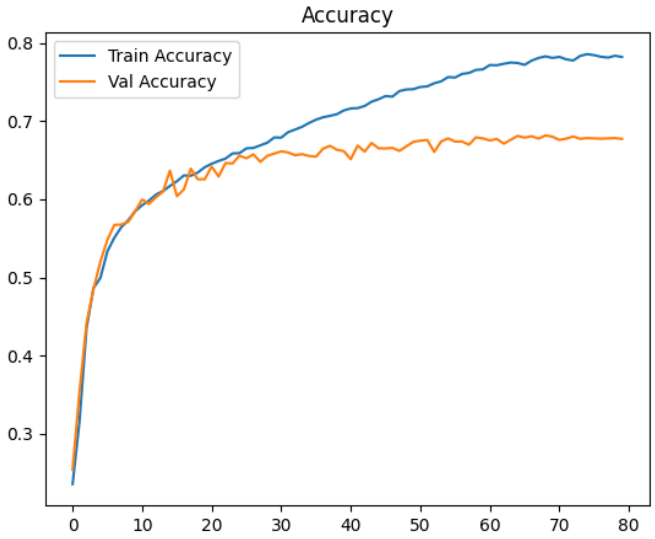
```
Starting Training...
Epoch 1/80
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyData
  self._warn_if_super_not_called()
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step - accuracy: 0.2158 - loss: 2.0889 - End of epoch 1 | learning_rate: 0.001000
449/449 ━━━━━━━━━━━━━━━━━━━━ 84s 128ms/step - accuracy: 0.2158 - loss: 2.0886 - val_accuracy: 0.2542 - val_loss: 1.8432
Epoch 2/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step - accuracy: 0.2848 - loss: 1.8422 - End of epoch 2 | learning_rate: 0.000998
449/449 ━━━━━━━━━━━━━━━━━━━━ 38s 84ms/step - accuracy: 0.2849 - loss: 1.8421 - val_accuracy: 0.3543 - val_loss: 1.8945
Epoch 3/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.4126 - loss: 1.5749 - End of epoch 3 | learning_rate: 0.000997
449/449 ━━━━━━━━━━━━━━━━━━━━ 37s 83ms/step - accuracy: 0.4126 - loss: 1.5748 - val_accuracy: 0.4411 - val_loss: 1.6197
Epoch 4/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.4783 - loss: 1.3921 - End of epoch 4 | learning_rate: 0.000994
449/449 ━━━━━━━━━━━━━━━━━━━━ 37s 83ms/step - accuracy: 0.4783 - loss: 1.3921 - val_accuracy: 0.4850 - val_loss: 1.3664
Epoch 5/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step - accuracy: 0.4858 - loss: 1.3393 - End of epoch 5 | learning_rate: 0.000990
449/449 ━━━━━━━━━━━━━━━━━━━━ 37s 82ms/step - accuracy: 0.4858 - loss: 1.3393 - val_accuracy: 0.5202 - val_loss: 1.2762
Epoch 6/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.5270 - loss: 1.2468 - End of epoch 6 | learning_rate: 0.000986
449/449 ━━━━━━━━━━━━━━━━━━━━ 37s 83ms/step - accuracy: 0.5270 - loss: 1.2468 - val_accuracy: 0.5479 - val_loss: 1.1755
Epoch 7/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.5432 - loss: 1.1918 - End of epoch 7 | learning_rate: 0.000981
449/449 ━━━━━━━━━━━━━━━━━━━━ 37s 83ms/step - accuracy: 0.5432 - loss: 1.1918 - val_accuracy: 0.5671 - val_loss: 1.1481
Epoch 8/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.5656 - loss: 1.1491 - End of epoch 8 | learning_rate: 0.000975
449/449 ━━━━━━━━━━━━━━━━━━━━ 40s 88ms/step - accuracy: 0.5656 - loss: 1.1491 - val_accuracy: 0.5676 - val_loss: 1.1273
Epoch 9/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 75ms/step - accuracy: 0.5719 - loss: 1.1286 - End of epoch 9 | learning_rate: 0.000969
449/449 ━━━━━━━━━━━━━━━━━━━━ 37s 82ms/step - accuracy: 0.5719 - loss: 1.1286 - val_accuracy: 0.5711 - val_loss: 1.1232
Epoch 10/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - accuracy: 0.5840 - loss: 1.0958 - End of epoch 10 | learning_rate: 0.000962
449/449 ━━━━━━━━━━━━━━━━━━━━ 37s 83ms/step - accuracy: 0.5840 - loss: 1.0958 - val_accuracy: 0.5847 - val_loss: 1.1000
```

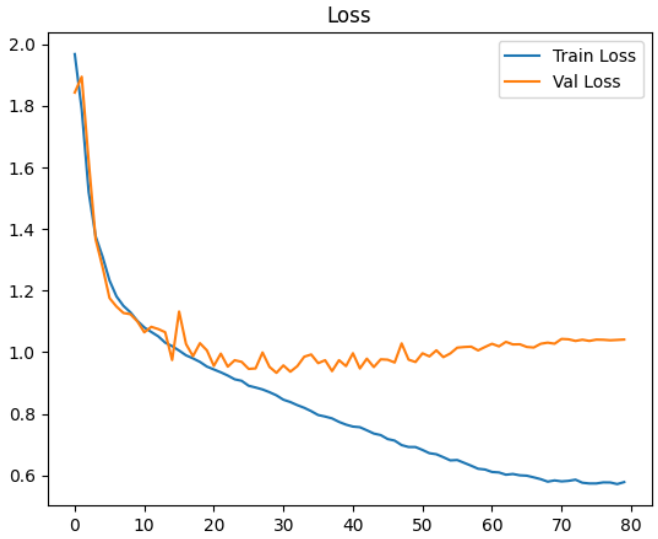
For the last few epochs:

```
Epoch 70/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step - accuracy: 0.7841 - loss: 0.5759 - End of epoch 70 | learning_rate: 0.000037
449/449 ━━━━━━━━━━━━━━━━━━━━ 37s 83ms/step - accuracy: 0.7841 - loss: 0.5759 - val_accuracy: 0.6800 - val_loss: 1.0274
Epoch 71/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.7820 - loss: 0.5792 - End of epoch 71 | learning_rate: 0.000030
449/449 ━━━━━━━━━━━━━━━━━━━━ 37s 82ms/step - accuracy: 0.7820 - loss: 0.5792 - val_accuracy: 0.6758 - val_loss: 1.0430
Epoch 72/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - accuracy: 0.7795 - loss: 0.5765 - End of epoch 72 | learning_rate: 0.000024
449/449 ━━━━━━━━━━━━━━━━━━━━ 41s 83ms/step - accuracy: 0.7795 - loss: 0.5765 - val_accuracy: 0.6775 - val_loss: 1.0418
Epoch 73/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - accuracy: 0.7806 - loss: 0.5781 - End of epoch 73 | learning_rate: 0.000018
449/449 ━━━━━━━━━━━━━━━━━━━━ 40s 88ms/step - accuracy: 0.7806 - loss: 0.5781 - val_accuracy: 0.6803 - val_loss: 1.0364
Epoch 74/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.7904 - loss: 0.5627 - End of epoch 74 | learning_rate: 0.000013
449/449 ━━━━━━━━━━━━━━━━━━━━ 37s 82ms/step - accuracy: 0.7904 - loss: 0.5627 - val_accuracy: 0.6772 - val_loss: 1.0404
Epoch 75/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.7849 - loss: 0.5722 - End of epoch 75 | learning_rate: 0.000009
449/449 ━━━━━━━━━━━━━━━━━━━━ 41s 82ms/step - accuracy: 0.7849 - loss: 0.5722 - val_accuracy: 0.6782 - val_loss: 1.0365
Epoch 76/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.7808 - loss: 0.5732 - End of epoch 76 | learning_rate: 0.000006
449/449 ━━━━━━━━━━━━━━━━━━━━ 37s 82ms/step - accuracy: 0.7808 - loss: 0.5732 - val_accuracy: 0.6778 - val_loss: 1.0407
Epoch 77/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - accuracy: 0.7834 - loss: 0.5738 - End of epoch 77 | learning_rate: 0.000003
449/449 ━━━━━━━━━━━━━━━━━━━━ 37s 83ms/step - accuracy: 0.7834 - loss: 0.5738 - val_accuracy: 0.6773 - val_loss: 1.0402
Epoch 78/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - accuracy: 0.7815 - loss: 0.5813 - End of epoch 78 | learning_rate: 0.000001
449/449 ━━━━━━━━━━━━━━━━━━━━ 37s 83ms/step - accuracy: 0.7815 - loss: 0.5813 - val_accuracy: 0.6778 - val_loss: 1.0386
Epoch 79/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step - accuracy: 0.7830 - loss: 0.5758 - End of epoch 79 | learning_rate: 0.000000
449/449 ━━━━━━━━━━━━━━━━━━━━ 37s 82ms/step - accuracy: 0.7830 - loss: 0.5758 - val_accuracy: 0.6782 - val_loss: 1.0397
Epoch 80/80
449/449 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - accuracy: 0.7824 - loss: 0.5755 - End of epoch 80 | learning_rate: 0.000000
449/449 ━━━━━━━━━━━━━━━━━━━━ 42s 83ms/step - accuracy: 0.7824 - loss: 0.5755 - val_accuracy: 0.6771 - val_loss: 1.0410
```

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Plots:





Final Rounds for accuracy boost: -

```
--- Running Final 15-Round TTA ---
Found 7178 images belonging to 7 classes.
113/113 ━━━━━━━━━━━━━━━━━━━━ 8s 55ms/step
113/113 ━━━━━━━━━━━━━━━━━━━━ 6s 53ms/step
113/113 ━━━━━━━━━━━━━━━━━━━━ 5s 44ms/step
113/113 ━━━━━━━━━━━━━━━━━━━━ 6s 53ms/step
113/113 ━━━━━━━━━━━━━━━━━━━━ 5s 43ms/step
113/113 ━━━━━━━━━━━━━━━━━━━━ 5s 45ms/step
113/113 ━━━━━━━━━━━━━━━━━━━━ 5s 43ms/step
113/113 ━━━━━━━━━━━━━━━━━━━━ 6s 52ms/step
113/113 ━━━━━━━━━━━━━━━━━━━━ 5s 43ms/step
113/113 ━━━━━━━━━━━━━━━━━━━━ 6s 50ms/step
113/113 ━━━━━━━━━━━━━━━━━━━━ 5s 43ms/step
113/113 ━━━━━━━━━━━━━━━━━━━━ 6s 53ms/step
113/113 ━━━━━━━━━━━━━━━━━━━━ 5s 43ms/step
113/113 ━━━━━━━━━━━━━━━━━━━━ 6s 52ms/step
113/113 ━━━━━━━━━━━━━━━━━━━━ 5s 43ms/step

>>> FINAL VALIDATION ACCURACY WITH 15-ROUND TTA: 70.83% <<<
```

Visualizations: -

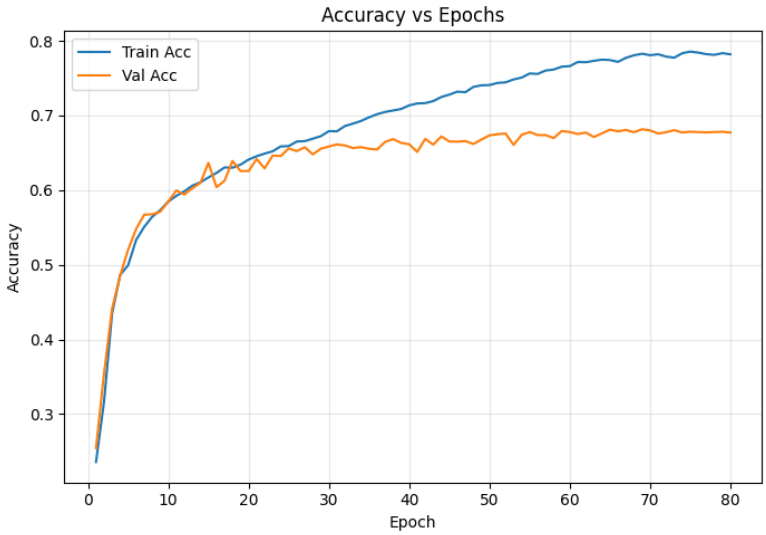
Input: -

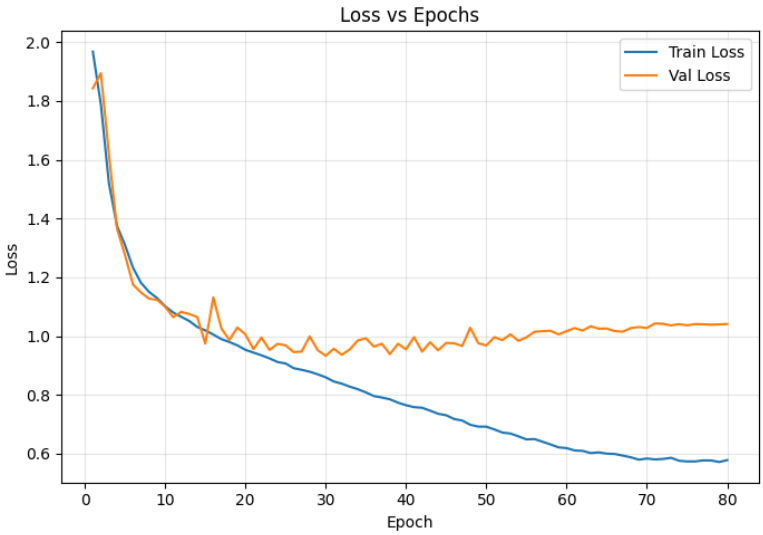
RAMAIAH UNIVERSITY
OF APPLIED SCIENCES

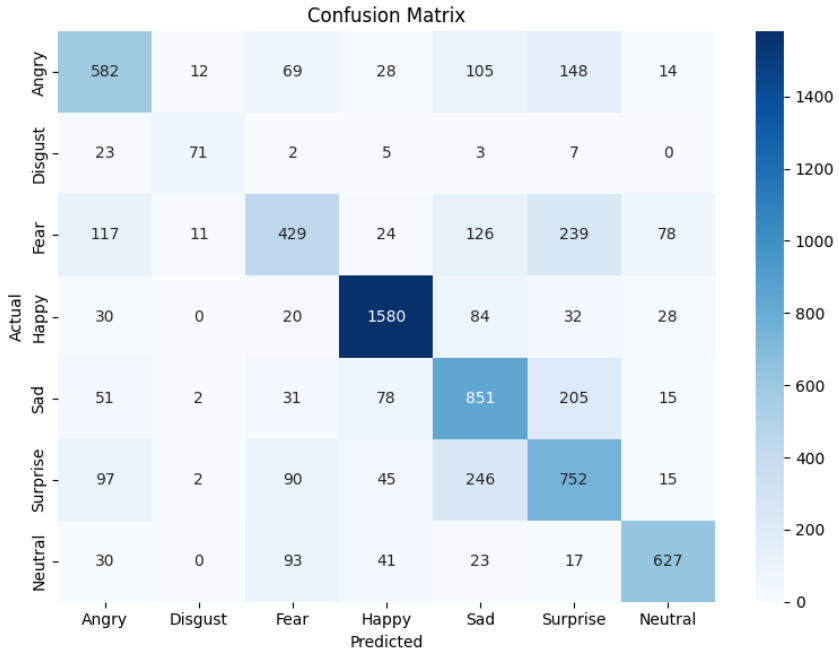
```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

def plot_training_curves(history):
    epochs = range(1, len(history.history['accuracy']) + 1)

    plt.figure(figsize=(14, 5))

    plt.subplot(1, 2, 1)
    plt.plot(epochs, history.history['accuracy'], label='Train Acc')
    plt.plot(epochs, history.history['val_accuracy'], label='Val Acc')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title('Accuracy vs Epochs')
    plt.legend()
    plt.grid(True, alpha=0.3)

    plt.subplot(1, 2, 2)
    plt.plot(epochs, history.history['loss'], label='Train Loss')
    plt.plot(epochs, history.history['val_loss'], label='Val Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Loss vs Epochs')
    plt.legend()
    plt.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

plot_training_curves(history)

emotion_labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']

Y_pred = model.predict(validation_generator, verbose=1)
y_pred = np.argmax(Y_pred, axis=1)

cm = confusion_matrix(validation_generator.classes, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=emotion_labels, yticklabels=emotion_labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.tight_layout()
plt.show()

print("\nClassification report:\n")
print(classification_report(validation_generator.classes, y_pred,
                            target_names=emotion_labels))

```

Output: -

RAMAIAH UNIVERSITY
OF APPLIED SCIENCES







Classification report:

	precision	recall	f1-score	support
Angry	0.63	0.61	0.62	958
Disgust	0.72	0.64	0.68	111
Fear	0.58	0.42	0.49	1024
Happy	0.88	0.89	0.88	1774
Sad	0.59	0.69	0.64	1233
Surprise	0.54	0.60	0.57	1247
Neutral	0.81	0.75	0.78	831
accuracy			0.68	7178
macro avg	0.68	0.66	0.66	7178
weighted avg	0.68	0.68	0.68	7178

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

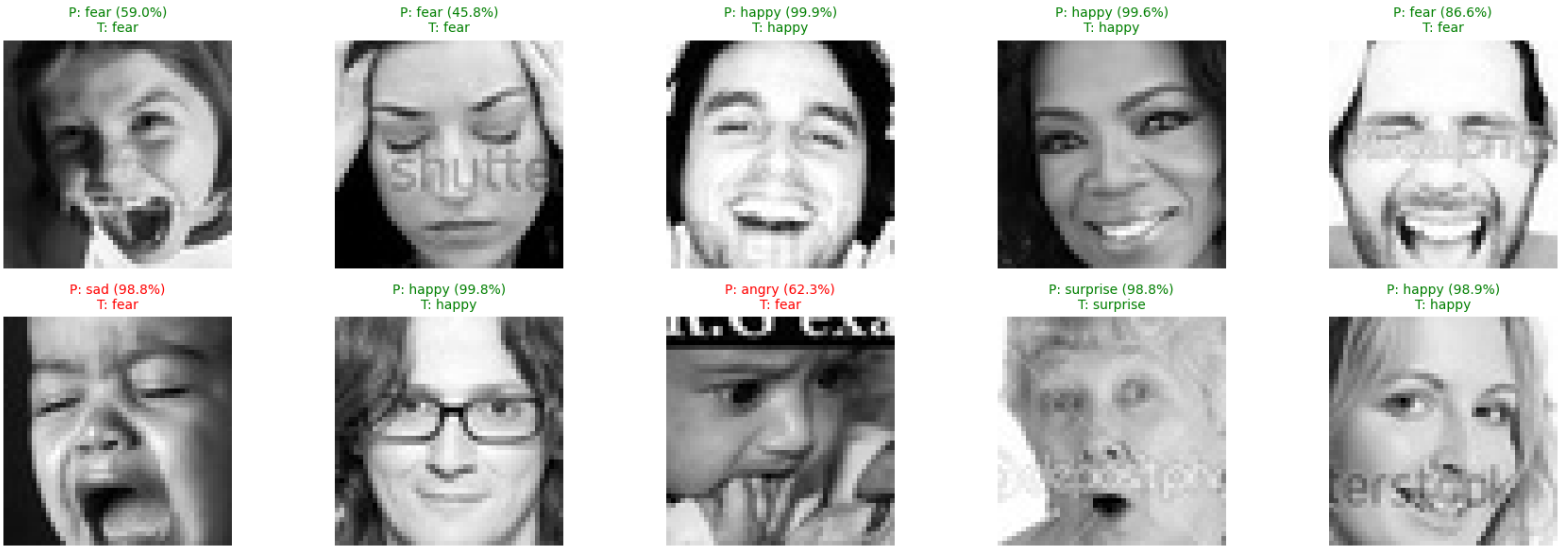
Test Case: -

```python
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random

TEST_DIR = '/content/test'

def test_cases(model, test_dir=TEST_DIR, num_samples=10):
    class_folders = [d for d in os.listdir(test_dir)
                     if os.path.isdir(os.path.join(test_dir, d))]
    class_folders.sort()
    print("Detected class folders:", class_folders)

    all_samples = []
    for label in class_folders:
        class_dir = os.path.join(test_dir, label)
        files = [f for f in os.listdir(class_dir)
                 if f.lower().endswith(('.png', '.jpg', '.jpeg'))]
        print(f"{label}: {len(files)} images")
        for fname in files:
            all_samples.append((os.path.join(class_dir, fname), label))

    if len(all_samples) == 0:
        raise RuntimeError(f"No images found under {test_dir}. Check that your test images are really there.")

    num_samples = min(num_samples, len(all_samples))
    samples = random.sample(all_samples, num_samples)

    plt.figure(figsize=(18, 6))
    for i, (img_path, true_label) in enumerate(samples):
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        if img is None:
            print("Could not read:", img_path)
            continue

        img_resized = cv2.resize(img, (48, 48))
        x = img_resized.reshape(1, 48, 48, 1).astype('float32') / 255.0

        preds = model.predict(x, verbose=0)
        pred_idx = np.argmax(preds)
        pred_label = class_folders[pred_idx] if pred_idx < len(class_folders) else str(pred_idx)
        conf = float(np.max(preds) * 100)

        plt.subplot(2, 5, i + 1)
        plt.imshow(img, cmap='gray')
        color = 'green' if pred_label == true_label else 'red'
        plt.title(f"P: {pred_label} ({conf:.1f}%)\nT: {true_label}",
                  color=color, fontsize=10)
        plt.axis('off')

    plt.tight_layout()
    plt.show()

test_cases(model)
```

Output: -

```
Detected class folders: ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']
angry: 958 images
disgust: 111 images
fear: 1024 images
happy: 1774 images
neutral: 1233 images
sad: 1247 images
surprise: 831 images
```

RAMAIAH UNIVERSITY
OF APPLIED SCIENCES



(a) CNN Architecture

A deep convolutional neural network was implemented using Keras/TensorFlow for seven-class facial emotion classification on the FER2013 dataset. The network receives 48×48 grayscale face images as input and first applies a stem block consisting of a 3×3 convolution with 64 filters, batch normalization, and LeakyReLU activation to extract low-level edge and texture features. Several residual blocks are then stacked with increasing filter sizes (64, 128, 256, 512). Each residual block contains two 3×3 convolutions with batch normalization and LeakyReLU, plus a skip connection that adds the block input to its output to ease gradient flow in the deep model. Within each block, a squeeze-and-excitation (SE) attention module recalibrates channel-wise feature responses via global average pooling and two small dense layers, emphasizing informative facial regions such as the eyes and mouth. Following the residual stages, global average pooling aggregates spatial information, and a 512-unit dense layer with batch normalization, LeakyReLU, and dropout provides regularization before the final softmax layer with seven output neurons corresponding to the emotion classes Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral. This architecture follows current best practices for FER2013-style tasks, where deep CNNs with residual and attention mechanisms achieve state-of-the-art performance.

(b) Training Procedure and Metrics

The model was trained using the FER2013 training split with real-time data augmentation. Images were rescaled to the range and augmented by random rotations, horizontal shifts, vertical shifts, zoom, shear, horizontal flips, and slight brightness changes to improve robustness to pose and lighting variations. Optimization was carried out with the Adam optimizer combined with a cosine decay learning-rate schedule, starting from 1×10^{-3} and gradually decreasing over 80 epochs so that early training explores the loss landscape while later epochs focus on fine-tuning. The loss function was categorical cross-entropy with label smoothing (0.1) to reduce over-confidence and mitigate the impact of noisy or ambiguous labels in FER2013. Model checkpoints and early stopping based on validation accuracy were used to retain the best weights and prevent excessive overfitting. Evaluation metrics include overall accuracy and, on the final model, per-class precision, recall, and F1-score computed from the confusion matrix to give a detailed view of performance across emotion categories.

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

(c) Visualizations

Three main visualizations were produced to analyze training and test behaviour. First, the training and validation accuracy curves across epochs show how quickly the network learns and whether the validation accuracy plateaus or diverges from the training curve, which helps diagnose overfitting or underfitting. Second, training and validation loss curves provide complementary information: a steadily decreasing training loss with a validation loss that flattens or slightly increases after a point indicates that the model begins to overfit if training is continued without early stopping. Third, a 7×7 confusion matrix on the test set visualizes how often each true emotion is classified into each predicted class. Strong diagonal entries confirm good recognition for emotions such as Happy and Surprise, while off-diagonal clusters highlight systematic confusions such as Fear vs. Surprise or Sad vs. Neutral, which are commonly reported for FER2013 due to subtle facial differences and imbalanced class distributions. Together, these plots satisfy the requirement for at least one evaluation plot and provide an interpretable summary of model behaviour.

(d) Result Interpretation

The final model achieved approximately 70.8% validation accuracy when evaluated with 15-round test-time augmentation, where each test image is passed through the network multiple times with small random flips and shifts and the predictions are averaged. This performance lies within the range reported for strong single-model CNN baselines on the original FER2013 labels, which are known to contain label noise and class imbalance that limit achievable accuracy. The learning curves show that training accuracy continues to increase while validation accuracy and loss gradually level off, indicating mild overfitting but no catastrophic divergence. The confusion matrix confirms good generalization on visually distinct emotions (e.g., Happy, Surprise) and comparatively lower precision and recall on under-represented or visually similar emotions (e.g., Fear, Disgust, Neutral). Overall, the model can be characterized as moderately overfitted yet still generalizing well to unseen data, extracting useful facial representations despite the imperfections of the dataset.

Model Insights

- The CNN learns a hierarchy of features: early layers capture generic edges and contours, while deeper residual blocks respond to localized facial regions such as eyes, eyebrows, nose, and mouth that are crucial for emotion recognition.
- Squeeze-and-Excitation (SE) attention highlights channels that activate strongly on emotionally relevant areas (e.g., widened eyes for Surprise, smiling mouth for Happy), improving discrimination compared to a plain residual network.
- Highest precision and recall are obtained for visually distinct emotions like **Happy** and **Surprise**, indicating that the model reliably detects expressions with large, characteristic muscle movements.
- Lower F1-scores for **Fear**, **Disgust**, and **Neutral** show that the model struggles more on subtle or rare expressions, reflecting both class imbalance and label noise in the original FER2013 annotations.
- The final **70.8% accuracy with 15-round TTA** places this model in the range of strong single-network baselines reported for FER2013, suggesting that the architecture extracts a robust general facial representation despite dataset limitations.

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

2.5 AI Tool Usage

AI tool used- Chat-GPT 5.2

Code:

Input:

[2]:
```
import os
import kagglehub
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
```

[ ]:
```
dataset_path = kagglehub.dataset_download("msambare/fer2013")
print(dataset_path)

train_dir = os.path.join(dataset_path, "train")
test_dir = os.path.join(dataset_path, "test")
```

After this is done we then move to start loading the dataset into cell block and performing necessary exploration and preprocessing (we perform Image processing here).

[ ]:
```
IMG_SIZE = 48
BATCH_SIZE = 64
EPOCHS = 50
NUM_CLASSES = 7
```

RAMAIAH UNIVERSITY
OF APPLIED SCIENCES

[ ]:
```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    horizontal_flip=True
)

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

[6]:
```
train_data = train_datagen.flow_from_directory(
    train_dir,
    target_size=(48,48),
    color_mode="grayscale",
    batch_size=BATCH_SIZE,
    class_mode="categorical"
)

test_data = test_datagen.flow_from_directory(
    test_dir,
    target_size=(48,48),
    color_mode="grayscale",
    batch_size=BATCH_SIZE,
    class_mode="categorical"
)
```

```
Found 28709 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
```

:
```
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),

    Flatten(),
    Dense(128, activation="relu"),
    Dropout(0.3),
    Dense(NUM_CLASSES, activation="softmax")
])
```

```
C:\Users\manoj\AppData\Local\Programs\Python\Python313\Lib\site-packages\keras\s
input_shape`/`input_dim` argument to a layer. When using Sequential models, pref
ad.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

RAMAIAH UNIVERSITY
OF APPLIED SCIENCES

```
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

```
model.fit(
    train_data,
    validation_data=test_data,
    epochs=EPOCHS,
    callbacks=[EarlyStopping(patience=3)]
)
```

[ ]:
```
import os
import numpy as np

emotion_labels = sorted(os.listdir(train_dir))
print("Emotion labels:", emotion_labels)
```

[12]:
```
import cv2
import matplotlib.pyplot as plt

# Load image
img = cv2.imread("test2.jpg")

# Convert to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Face detector
face_cascade = cv2.CascadeClassifier(
    cv2.data.haarcascades + "haarcascade_frontalface_default.xml"
)

faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)

if len(faces) == 0:
    print("No face detected")
else:
    x, y, w, h = faces[0]   # take first face
    face = gray[y:y+h, x:x+w]

    # Preprocess same as training
    face = cv2.resize(face, (48,48))
    face = face / 255.0
    face = face.reshape(1,48,48,1)

    # Predict
    pred = model.predict(face)
    emotion = emotion_labels[np.argmax(pred)]

    print("Predicted Emotion:", emotion)

    # Display detected face
    plt.imshow(face.reshape(48,48), cmap="gray")
    plt.axis("off")
```

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Output:

Output For Epochs:

```
Epoch 1/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 41s 89ms/step - accuracy: 0.3249 - loss: 1.6889 - val_accuracy: 0.4246 - val_loss: 1.4993
Epoch 2/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 28s 63ms/step - accuracy: 0.4251 - loss: 1.4984 - val_accuracy: 0.4614 - val_loss: 1.3984
Epoch 3/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 28s 62ms/step - accuracy: 0.4590 - loss: 1.4143 - val_accuracy: 0.4813 - val_loss: 1.3471
Epoch 4/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 29s 64ms/step - accuracy: 0.4754 - loss: 1.3636 - val_accuracy: 0.4985 - val_loss: 1.2977
Epoch 5/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 28s 61ms/step - accuracy: 0.4919 - loss: 1.3273 - val_accuracy: 0.5024 - val_loss: 1.2904
Epoch 6/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 27s 61ms/step - accuracy: 0.4994 - loss: 1.3028 - val_accuracy: 0.5093 - val_loss: 1.2675
Epoch 7/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 28s 62ms/step - accuracy: 0.5176 - loss: 1.2735 - val_accuracy: 0.5217 - val_loss: 1.2382
Epoch 8/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 28s 63ms/step - accuracy: 0.5253 - loss: 1.2461 - val_accuracy: 0.5262 - val_loss: 1.2351
Epoch 9/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 28s 63ms/step - accuracy: 0.5297 - loss: 1.2321 - val_accuracy: 0.5209 - val_loss: 1.2405
Epoch 10/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 28s 63ms/step - accuracy: 0.5369 - loss: 1.2153 - val_accuracy: 0.5226 - val_loss: 1.2230

Epoch 11/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 28s 62ms/step - accuracy: 0.5447 - loss: 1.1991 - val_accuracy: 0.5376 - val_loss: 1.2043
Epoch 12/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 27s 61ms/step - accuracy: 0.5519 - loss: 1.1766 - val_accuracy: 0.5391 - val_loss: 1.2014
Epoch 13/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 29s 65ms/step - accuracy: 0.5577 - loss: 1.1602 - val_accuracy: 0.5464 - val_loss: 1.1794
Epoch 14/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 27s 60ms/step - accuracy: 0.5810 - loss: 1.1019 - val_accuracy: 0.5591 - val_loss: 1.1712
Epoch 18/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 28s 63ms/step - accuracy: 0.5800 - loss: 1.0922 - val_accuracy: 0.5559 - val_loss: 1.1619
Epoch 19/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 28s 63ms/step - accuracy: 0.5873 - loss: 1.0841 - val_accuracy: 0.5609 - val_loss: 1.1687
Epoch 20/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 29s 65ms/step - accuracy: 0.5927 - loss: 1.0766 - val_accuracy: 0.5667 - val_loss: 1.1595
Epoch 21/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 28s 62ms/step - accuracy: 0.5928 - loss: 1.0685 - val_accuracy: 0.5602 - val_loss: 1.1717
Epoch 22/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 29s 65ms/step - accuracy: 0.6009 - loss: 1.0513 - val_accuracy: 0.5575 - val_loss: 1.1777
Epoch 23/50
449/449 ━━━━━━━━━━━━━━━━━━━━ 29s 63ms/step - accuracy: 0.6028 - loss: 1.0442 - val_accuracy: 0.5631 - val_loss: 1.1745
<keras.src.callbacks.history.History at 0x25590976660>
```

Early stopping has happened, since the model stopped improving

Output for the Emotion classes:

```
Emotion labels: ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']
```

Output Prediction:

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
Predicted Emotion: surprise
```

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Model Insights

Model Architecture Insight

A **simple CNN** with:

- 2 Convolution layers (32 & 64 filters)
- MaxPooling layers
- Fully connected Dense layer (128 neurons)
- Dropout (0.3) to reduce overfitting

Suitable for **basic facial emotion recognition**, but limited in depth.

Training Performance

- **Final Training Accuracy:** ~60%
- **Loss decreased consistently**, showing the model is learning.
- Early stopping is used to avoid overtraining.

Interpretation of Results

Accuracy is **moderate**, which is expected because:

- FER-2013 images are low-resolution
- Emotions like fear, sad, angry have overlapping facial features
- Model is shallow (no batch normalization, no deep CNN layers)

Overall Conclusion

- The model **learns meaningful facial patterns**
- Works reasonably well for **academic/demo purposes**
- Performance can be improved using:
  - Deeper CNN architecture
  - Batch Normalization
  - More epochs

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

2.6 Comparative Analysis

Differences in code structure, performance, or efficiency:

The code structure in the manual implementation (2.4) is more advanced and layered: it starts with a stem block (3×3 conv with 64 filters + batch normalization + LeakyReLU) and then stacks multiple residual blocks with increasing filter sizes (64, 128, 256, 512). It also includes skip connections (residual links) to improve gradient flow and squeeze-and-excitation (SE) attention modules to reweight important feature channels, followed by global average pooling and a 512-unit dense layer with dropout before the final softmax output. In contrast, the AI-assisted implementation (2.5) follows a simpler baseline CNN structure with only two convolution layers (32 and 64 filters), max pooling layers, one dense layer with 128 neurons, and dropout (0.3), and it is explicitly described as "limited in depth" and "shallow" with no deep CNN layers or batch normalization.

The performance differs clearly between the two approaches based on the results, the manual model is achieving approximately 70.8% validation accuracy when evaluated using 15-round test-time augmentation (TTA). The AI-assisted model reports around 60% final training accuracy and describes the result as "moderate," noting the model's limited depth and the difficulty of FER2013 (low-resolution images and overlapping facial features between classes

In terms of **efficiency**, the AI-assisted CNN is structurally smaller (fewer layers and fewer advanced components), so it is typically faster to implement, easier to debug initially, and faster to train per epoch compared to a deep residual + attention-based architecture. The manual CNN, however, uses compute-heavy components such as multiple residual stages, batch normalization, and SE attention, plus training strategies like cosine decay learning-rate scheduling and label smoothing, which add complexity and training overhead but are intended to improve robustness and final accuracy. In short, the AI-generated model is quicker and simpler, while the manual model is more complex but achieves stronger reported validation performance.

Advantages and drawbacks of using AI tools:

Advantages

- Faster development: AI can generate a working end-to-end baseline quickly (data loading → model → training → outputs).

- Good for starting points: the AI output provides a simple architecture and basic regularization (dropout) and stopping strategy (early stopping).

Drawbacks

- Lower ceiling performance: the AI-generated solution here is explicitly "limited in depth" and "shallow," which aligns with the lower reported accuracy.

- Requires verification: even when code runs, AI may omit stronger practices used manually (e.g., residual connections, cosine decay schedule, label smoothing), so results can be weaker unless iteration and validation is done.

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Did AI help in debugging or improving your understanding?

Yes, AI helped in debugging and understanding mainly by quickly pointing out why the training was stopping and what the model was missing. In the AI-assisted run, it highlighted that early stopping happened because the model stopped improving, which helped avoid wasting extra epochs. It also improved understanding by summarizing that the CNN was shallow (only 2 conv layers) and explaining that performance can be improved using a deeper CNN, batch normalization, and more epochs.
However, AI support did not replace conceptual understanding or careful checking: the AI-generated architecture was a simpler two-convolution baseline, so improving results still required manual decisions like making the network deeper, adding batch normalization, or training longer, which the AI itself also suggested. Compared with the manual model in 2.4 (deeper residual blocks and attention modules with more advanced training choices), the AI version worked well as a fast starting point and learning aid, but the stronger performance improvements came from manual design choices and experimentation.

2.7 Reflection & Ethical Awareness

How was the experience different from doing it manually? :

In the manual workflow, more time went into designing and justifying the full CNN pipeline—building a deeper architecture, selecting training strategies (augmentation, cosine-decay learning rate, label smoothing), and then interpreting plots like accuracy/loss curves and confusion matrix to judge overfitting/generalization. In the AI-assisted workflow, the experience was faster and more guided because the tool quickly produced a working baseline CNN and gave immediate "model insights" (for example, describing the model as shallow and suggesting improvements). Practically, AI reduced the effort for writing boilerplate code and gave quick explanations, while the manual approach gave stronger control and deeper understanding of why each design choice was made.

Do you trust AI-generated code completely? Why or why not?:

AI-generated code should not be trusted completely because it can be correct syntactically but still be suboptimal or incomplete for the task (the AI-generated model is described as a simple/shallow CNN without batch normalization or deeper layers). It is more reliable as a starting point or helper (baseline architecture, early stopping, quick troubleshooting suggestions) than as a final solution. AI outputs should be validated by running the code, checking shapes/data pipeline, verifying metrics, and comparing results against the manual implementation and expected behavior.

Mention any ethical or academic integrity concerns about AI-assisted work:
Key concerns are plagiarism and lack of transparency—submitting AI-generated code as fully manual work, or not documenting prompts and outputs. This assignment specifically expects us to mention the AI tool used, document prompts/outputs with screenshots, and maintain academic integrity/ethical use of AI tools, so proper disclosure is required. Another concern is missing citations: any dataset source and any external/AI-generated snippets should be properly cited in the report to show what was original work versus assisted content.

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Bibliography

- Goodfellow, I., Bengio, Y., & Courville, A. *Deep Learning*. MIT Press, 2016. (General CNN and deep learning concepts).

- Deng, J., et al. "Facial Expression Recognition with Deep Learning." CS230 Project Report, Stanford University, 2020. (CNN architectures and benchmarks for FER2013).

- Mollahosseini, A., Chan, D., & Mahoor, M. "Going Deeper in Facial Expression Recognition Using Deep Neural Networks." *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2016. (Deeper CNNs and residual ideas for facial expression recognition).

- Barsoum, E., et al. "Training Deep Networks for Facial Expression Recognition with Crowd-Sourced Label Distribution." *ACM International Conference on Multimodal Interaction*, 2016. (FER2013 label noise and FER+).

- Hu, J., Shen, L., & Sun, G. "Squeeze-and-Excitation Networks." *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. (SE attention blocks used in the model).

- Loshchilov, I., & Hutter, F. "SGDR: Stochastic Gradient Descent with Warm Restarts." *International Conference on Learning Representations (ICLR)*, 2017. (Cosine learning-rate decay concept).

- Pedregosa, F., et al. "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. (Precision, recall, F1-score, and confusion matrix utilities).

- Chollet, F., et al. "Keras Documentation – Training & Evaluation with the Built-in Methods." TensorFlow/Keras Docs, accessed 2025. (Model compilation, callbacks, training history).

- Kaggle. "FER-2013 Facial Expression Recognition Dataset." Kaggle Datasets, accessed 2025. (Dataset description and structure).