3_model_compare_emotion_detection

November 24, 2025

Loads FER2013 from your folders:

LOCAL_TRAIN = 'data/train' LOCAL_VAL = 'data/val' LOCAL_TEST = 'data/test'

0.1 Trains three models:

VGG-Face-like → VGG16 backbone FaceNet-like → InceptionResNetV2 backbone ArcFace-like → ResNet50 backbone

0.2 Trains each model in two phases:

Phase 1 – backbone frozen (only head trains) Phase 2 – fine-tuning: unfreeze top layers of the backbone, smaller learning rate

Saves best weights (based on val_accuracy) with ModelCheckpoint and loads them before test evaluation Plots accuracy & loss curves Shows confusion matrices Compares test accuracy in a bar chart

0.3 Evaluates and compares them:

Training curves (accuracy + loss) Confusion matrix for each model Classification report for each model Bar chart of test accuracy comparison

0.4 Notes / Tweaks you can do

Increase EPOCHS once you see the script runs and GPU memory is fine.

For better performance: After initial training, unfreeze the last block of each backbone and fine-tune with a smaller LR. Add class-weighting if your FER2013 split is imbalanced.

A second fine-tuning phase (unfreeze last layers) for each model. Saving best weights and loading them before test evaluation.

[1]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf

tf.config.optimizer.set_jit(False)
```

```python
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    try:
        tf.config.experimental.set_memory_growth(gpu, True)
    except:
        pass

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16, InceptionResNetV2, ResNet50
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout,␣
↪BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.metrics import confusion_matrix, classification_report

# ============================
# 0. Paths & Global Settings
# ============================

LOCAL_TEST   = 'data/test'
LOCAL_TRAIN  = 'data/train'
LOCAL_VAL    = 'data/val'

IMG_SIZE = 128        # Resize 48x48 → 224x224 for pretrained backbones
BATCH_SIZE = 8
EPOCHS_FROZEN = 30      # initial training with frozen backbone
EPOCHS_FT = 10          # fine-tuning epochs
SEED = 42

np.random.seed(SEED)
tf.random.set_seed(SEED)
```

```
2025-11-20 14:55:17.839048: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:485] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2025-11-20 14:55:17.860274: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:8454] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2025-11-20 14:55:17.866515: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1452] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2025-11-20 14:55:17.882422: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
```

```
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
I0000 00:00:1763650519.832777   71308 cuda_executor.cc:1015] successful NUMA
node read from SysFS had negative value (-1), but there must be at least one
NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
I0000 00:00:1763650519.845509   71308 cuda_executor.cc:1015] successful NUMA
node read from SysFS had negative value (-1), but there must be at least one
NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
I0000 00:00:1763650519.848952   71308 cuda_executor.cc:1015] successful NUMA
node read from SysFS had negative value (-1), but there must be at least one
NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
```

[3]:
```python
# ============================
# 1. Data Generators
# ============================

def create_generators():
    # Explicit class ordering: MUST match directory names
    emotion_classes = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']

    train_datagen = ImageDataGenerator(
        rescale=1./255,
        horizontal_flip=True,
        rotation_range=10,
        width_shift_range=0.1,
        height_shift_range=0.1,
        zoom_range=0.1
    )

    val_test_datagen = ImageDataGenerator(rescale=1./255)

    train_gen = train_datagen.flow_from_directory(
        LOCAL_TRAIN,
        target_size=(IMG_SIZE, IMG_SIZE),
        color_mode='rgb',
        batch_size=BATCH_SIZE,
        class_mode='categorical',
        shuffle=True,
```

```python
        seed=SEED,
        classes=emotion_classes    #   force 7 classes
    )

    val_gen = val_test_datagen.flow_from_directory(
        LOCAL_VAL,
        target_size=(IMG_SIZE, IMG_SIZE),
        color_mode='rgb',
        batch_size=BATCH_SIZE,
        class_mode='categorical',
        shuffle=True,
        seed=SEED,
        classes=emotion_classes    #   same 7 classes, same order
    )

    test_gen = val_test_datagen.flow_from_directory(
        LOCAL_TEST,
        target_size=(IMG_SIZE, IMG_SIZE),
        color_mode='rgb',
        batch_size=BATCH_SIZE,
        class_mode='categorical',
        shuffle=False,
        classes=emotion_classes    #   same 7 classes, same order
    )

    class_names = emotion_classes
    print("[INFO] Classes:", class_names)
    return train_gen, val_gen, test_gen, class_names



# ============================
# 2. Model Builders
# ============================

def add_classification_head(base_model, num_classes, dropout_rate=0.5):
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = BatchNormalization()(x)
    x = Dropout(dropout_rate)(x)
    x = Dense(256, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dropout(dropout_rate)(x)
    outputs = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=outputs)
    return model
```

```python
def build_vgg_face_like(input_shape, num_classes):
    """
    VGG-Face-like model using VGG16 backbone (Imagenet weights).
    """
    base = VGG16(
        include_top=False,
        weights='imagenet',
        input_shape=input_shape
    )
    # Freeze all backbone layers initially
    for layer in base.layers:
        layer.trainable = False

    model = add_classification_head(base, num_classes, dropout_rate=0.5)
    model.name = "VGGFaceLike_VGG16"
    return model


def build_facenet_like(input_shape, num_classes):
    """
    FaceNet-like model using InceptionResNetV2 backbone.
    """
    base = InceptionResNetV2(
        include_top=False,
        weights='imagenet',
        input_shape=input_shape
    )
    for layer in base.layers:
        layer.trainable = False

    model = add_classification_head(base, num_classes, dropout_rate=0.5)
    model.name = "FaceNetLike_InceptionResNetV2"
    return model


def build_arcface_like(input_shape, num_classes):
    """
    ArcFace-like model using ResNet50 backbone.
    (We are not implementing the ArcFace loss here, but using the backbone and␣
 ↪a softmax head.)
    """
    base = ResNet50(
        include_top=False,
        weights='imagenet',
        input_shape=input_shape
    )
```

```python
    for layer in base.layers:
        layer.trainable = False

    model = add_classification_head(base, num_classes, dropout_rate=0.5)
    model.name = "ArcFaceLike_ResNet50"
    return model


# ============================
# 3. Train & Evaluate Utility
# ============================

def compile_model(model, lr=1e-4):
    model.compile(
        optimizer=Adam(learning_rate=lr),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    return model


class SimpleHistory:
    """Wrapper so we can store combined history and keep .history API."""
    def __init__(self, history_dict):
        self.history = history_dict


def train_and_evaluate(
    model,
    train_gen,
    val_gen,
    test_gen,
    class_names,
    epochs_frozen=EPOCHS_FROZEN,
    epochs_ft=EPOCHS_FT,
    fine_tune_at_layers=20
):
    print(f"\n[INFO] Training model (frozen backbone): {model.name}")
    model = compile_model(model, lr=1e-4)

    # ---- Phase 1: Frozen backbone ----
    es_frozen = EarlyStopping(
        monitor='val_accuracy',
        patience=3,
        restore_best_weights=True,
        mode='max',
        verbose=1
```

```python
    )

    hist_frozen = model.fit(
        train_gen,
        epochs=epochs_frozen,
        validation_data=val_gen,
        callbacks=[es_frozen],
        verbose=1
    )

    # ---- Phase 2: Fine-tuning ----
    print(f"\n[INFO] Fine-tuning top {fine_tune_at_layers} layers of {model.name}")

    # Unfreeze last N layers (except BatchNorm, usually safer to keep them frozen)
    trainable_count = 0
    for layer in model.layers[-fine_tune_at_layers:]:
        if not isinstance(layer, BatchNormalization):
            layer.trainable = True
            trainable_count += 1
    print(f"[INFO] Unfroze {trainable_count} layers in {model.name}")

    model = compile_model(model, lr=1e-5)

    checkpoint_path = f"best_{model.name}_finetune.keras"
    ckpt = ModelCheckpoint(
        checkpoint_path,
        monitor='val_accuracy',
        save_best_only=True,
        mode='max',
        verbose=1
    )
    es_ft = EarlyStopping(
        monitor='val_accuracy',
        patience=3,
        restore_best_weights=True,
        mode='max',
        verbose=1
    )

    hist_ft = model.fit(
        train_gen,
        epochs=epochs_ft,
        validation_data=val_gen,
        callbacks=[ckpt, es_ft],
        verbose=1
```

```
    )

    # ---- Combine histories for plotting ----
    combined_history = {
        'accuracy': hist_frozen.history['accuracy'] + hist_ft.
↪history['accuracy'],
        'val_accuracy': hist_frozen.history['val_accuracy'] + hist_ft.
↪history['val_accuracy'],
        'loss': hist_frozen.history['loss'] + hist_ft.history['loss'],
        'val_loss': hist_frozen.history['val_loss'] + hist_ft.
↪history['val_loss'],
    }
    history = SimpleHistory(combined_history)

    # ---- Load best weights from fine-tuning phase ----
    if os.path.exists(checkpoint_path):
        print(f"[INFO] Loading best weights from: {checkpoint_path}")
        model.load_weights(checkpoint_path)

    # ---- Evaluate on test set ----
    print(f"\n[INFO] Evaluating model: {model.name} on test set")
    test_gen.reset()
    test_loss, test_acc = model.evaluate(test_gen, verbose=1)
    print(f"[RESULT] {model.name} - Test Loss: {test_loss:.4f}, Test Accuracy:␣
↪{test_acc:.4f}")

    # Predictions for confusion matrix
    test_gen.reset()
    y_prob = model.predict(test_gen, verbose=1)
    y_pred = np.argmax(y_prob, axis=1)
    y_true = test_gen.classes

    # Classification report
    print(f"\n[CLASSIFICATION REPORT] {model.name}")
    print(classification_report(y_true, y_pred, target_names=class_names))

    full_model_path = f"{model.name}_best_full.keras"
    print(f"[INFO] Saving full model to {full_model_path}")
    model.save(full_model_path)

    return history, test_loss, test_acc, y_true, y_pred, full_model_path


# ============================
# 4. Plotting Functions
# ============================
```

```python
def plot_training_histories(histories_dict):
    """
    histories_dict: {model_name: history_like}
    history_like: object with .history dict
    """
    plt.figure(figsize=(12, 5))

    # Accuracy
    plt.subplot(1, 2, 1)
    for name, hist in histories_dict.items():
        plt.plot(hist.history['accuracy'], label=f'{name} Train')
        plt.plot(hist.history['val_accuracy'], linestyle='--', label=f'{name}␣
↪Val')
    plt.title('Training & Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    # Loss
    plt.subplot(1, 2, 2)
    for name, hist in histories_dict.items():
        plt.plot(hist.history['loss'], label=f'{name} Train')
        plt.plot(hist.history['val_loss'], linestyle='--', label=f'{name} Val')
    plt.title('Training & Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()


def plot_confusion_matrix(y_true, y_pred, class_names, title):
    cm = confusion_matrix(y_true, y_pred)
    cm_norm = cm.astype('float') / cm.sum(axis=1, keepdims=True)

    plt.figure(figsize=(7, 6))
    sns.heatmap(cm_norm, annot=True, fmt=".2f",
                xticklabels=class_names, yticklabels=class_names,
                cmap='Blues')
    plt.title(f'Confusion Matrix - {title}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.tight_layout()
    plt.show()
```

```python
def plot_accuracy_bar(test_results):
    """
    test_results: {model_name: test_accuracy}
    """
    names = list(test_results.keys())
    accs = [test_results[n] for n in names]

    plt.figure(figsize=(8, 5))
    plt.bar(names, accs)
    plt.ylim(0, 1.0)
    plt.title('Test Accuracy Comparison')
    plt.ylabel('Accuracy')
    for i, v in enumerate(accs):
        plt.text(i, v + 0.01, f"{v:.3f}", ha='center')
    plt.xticks(rotation=20)
    plt.tight_layout()
    plt.show()


# ============================
# 5. Main
# ============================

def main():
    # 1) Create generators
    train_gen, val_gen, test_gen, class_names = create_generators()
    num_classes = len(class_names)
    input_shape = (IMG_SIZE, IMG_SIZE, 3)

    # 2) Build models
    vgg_model = build_vgg_face_like(input_shape, num_classes)
    facenet_model = build_facenet_like(input_shape, num_classes)
    arcface_model = build_arcface_like(input_shape, num_classes)

    histories = {}
    test_results = {}
    predictions = {}  # store y_true & y_pred for each model

    vgg_hist, vgg_loss, vgg_acc, vgg_y_true, vgg_y_pred, vgg_path = ␣
↪train_and_evaluate(
        vgg_model, train_gen, val_gen, test_gen, class_names
    )
    histories['VGG16'] = vgg_hist
    test_results['VGG16'] = vgg_acc
    predictions['VGG16'] = (vgg_y_true, vgg_y_pred)
```

```python
    model_paths = {'VGG16': vgg_path}

    # 4) Train & evaluate FaceNet-like
    facenet_hist, facenet_loss, facenet_acc, fn_y_true, fn_y_pred, fn_path = train_and_evaluate(
        facenet_model, train_gen, val_gen, test_gen, class_names
    )
    histories['InceptionResNetV2'] = facenet_hist
    test_results['InceptionResNetV2'] = facenet_acc
    predictions['InceptionResNetV2'] = (fn_y_true, fn_y_pred)
    model_paths['InceptionResNetV2'] = fn_path

    # 5) Train & evaluate ArcFace-like
    arc_hist, arc_loss, arc_acc, arc_y_true, arc_y_pred, arc_path = train_and_evaluate(
        arcface_model, train_gen, val_gen, test_gen, class_names
    )
    histories['ResNet50'] = arc_hist
    test_results['ResNet50'] = arc_acc
    predictions['ResNet50'] = (arc_y_true, arc_y_pred)
    model_paths['ResNet50'] = arc_path

    # 6) Plot training curves
    plot_training_histories(histories)

    # 7) Plot confusion matrices
    for model_name, (y_true, y_pred) in predictions.items():
        plot_confusion_matrix(y_true, y_pred, class_names, title=model_name)

    # 8) Bar chart comparison
    plot_accuracy_bar(test_results)

    # 9) Pick best model and save its path + class names for inference
    best_model_name = max(test_results, key=test_results.get)
    best_model_path = model_paths[best_model_name]
    print(f"\n[INFO] Best model: {best_model_name} (acc={test_results[best_model_name]:.4f})")
    print(f"[INFO] Best model file: {best_model_path}")

    # Save a unified copy for inference
    import shutil
    shutil.copy(best_model_path, "best_emotion_model.keras")
    print("[INFO] Copied best model to best_emotion_model.keras")

    # Save class names in correct order
    np.save("class_names.npy", np.array(class_names))
    print("[INFO] Saved class names to class_names.npy:", class_names)
```

```python
if __name__ == "__main__":
    main()
```

```
Found 29008 images belonging to 7 classes.
Found 6216 images belonging to 7 classes.
Found 6216 images belonging to 7 classes.
[INFO] Classes: ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad',
'surprise']

[INFO] Training model (frozen backbone): VGGFaceLike_VGG16
Epoch 1/30
3626/3626          128s 34ms/step
- accuracy: 0.2572 - loss: 2.3776 - val_accuracy: 0.3975 - val_loss: 1.6079
Epoch 2/30
3626/3626          125s 34ms/step
- accuracy: 0.3088 - loss: 1.9410 - val_accuracy: 0.4305 - val_loss: 1.4962
Epoch 3/30
3626/3626          124s 34ms/step
- accuracy: 0.3361 - loss: 1.7666 - val_accuracy: 0.4376 - val_loss: 1.4708
Epoch 4/30
3626/3626          126s 35ms/step
- accuracy: 0.3548 - loss: 1.6867 - val_accuracy: 0.4397 - val_loss: 1.4504
Epoch 5/30
3626/3626          125s 34ms/step
- accuracy: 0.3651 - loss: 1.6374 - val_accuracy: 0.4453 - val_loss: 1.4410
Epoch 6/30
3626/3626          124s 34ms/step
- accuracy: 0.3674 - loss: 1.6144 - val_accuracy: 0.4495 - val_loss: 1.4341
Epoch 7/30
3626/3626          123s 34ms/step
- accuracy: 0.3793 - loss: 1.5965 - val_accuracy: 0.4529 - val_loss: 1.4254
Epoch 8/30
3626/3626          124s 34ms/step
- accuracy: 0.3807 - loss: 1.5826 - val_accuracy: 0.4545 - val_loss: 1.4119
Epoch 9/30
3626/3626          123s 34ms/step
- accuracy: 0.3884 - loss: 1.5735 - val_accuracy: 0.4625 - val_loss: 1.4088
Epoch 10/30
3626/3626          124s 34ms/step
- accuracy: 0.3924 - loss: 1.5706 - val_accuracy: 0.4574 - val_loss: 1.4060
Epoch 11/30
3626/3626          123s 34ms/step
- accuracy: 0.3889 - loss: 1.5711 - val_accuracy: 0.4546 - val_loss: 1.4100
Epoch 12/30
3626/3626          123s 34ms/step
- accuracy: 0.3895 - loss: 1.5633 - val_accuracy: 0.4579 - val_loss: 1.4025
Epoch 12: early stopping
```

```
Restoring model weights from the end of the best epoch: 9.

[INFO] Fine-tuning top 20 layers of VGGFaceLike_VGG16
[INFO] Unfroze 18 layers in VGGFaceLike_VGG16
Epoch 1/10
3625/3626              0s 46ms/step -
accuracy: 0.4287 - loss: 1.4817
Epoch 1: val_accuracy improved from None to 0.54424, saving model to
best_VGGFaceLike_VGG16_finetune.keras
3626/3626              190s 50ms/step
- accuracy: 0.4631 - loss: 1.4019 - val_accuracy: 0.5442 - val_loss: 1.2018
Epoch 2/10
3626/3626              0s 46ms/step -
accuracy: 0.5355 - loss: 1.2508
Epoch 2: val_accuracy improved from 0.54424 to 0.57384, saving model to
best_VGGFaceLike_VGG16_finetune.keras
3626/3626              181s 50ms/step
- accuracy: 0.5486 - loss: 1.2201 - val_accuracy: 0.5738 - val_loss: 1.1134
Epoch 3/10
3626/3626              0s 46ms/step -
accuracy: 0.5721 - loss: 1.1493
Epoch 3: val_accuracy improved from 0.57384 to 0.61052, saving model to
best_VGGFaceLike_VGG16_finetune.keras
3626/3626              181s 50ms/step
- accuracy: 0.5799 - loss: 1.1311 - val_accuracy: 0.6105 - val_loss: 1.0221
Epoch 4/10
3626/3626              0s 46ms/step -
accuracy: 0.6103 - loss: 1.0703
Epoch 4: val_accuracy improved from 0.61052 to 0.63530, saving model to
best_VGGFaceLike_VGG16_finetune.keras
3626/3626              181s 50ms/step
- accuracy: 0.6150 - loss: 1.0583 - val_accuracy: 0.6353 - val_loss: 0.9888
Epoch 5/10
3626/3626              0s 46ms/step -
accuracy: 0.6371 - loss: 1.0075
Epoch 5: val_accuracy improved from 0.63530 to 0.65878, saving model to
best_VGGFaceLike_VGG16_finetune.keras
3626/3626              181s 50ms/step
- accuracy: 0.6355 - loss: 1.0029 - val_accuracy: 0.6588 - val_loss: 0.9144
Epoch 6/10
3626/3626              0s 46ms/step -
accuracy: 0.6526 - loss: 0.9503
Epoch 6: val_accuracy improved from 0.65878 to 0.67278, saving model to
best_VGGFaceLike_VGG16_finetune.keras
3626/3626              181s 50ms/step
- accuracy: 0.6544 - loss: 0.9520 - val_accuracy: 0.6728 - val_loss: 0.8605
Epoch 7/10
3626/3626              0s 46ms/step -
```

```
accuracy: 0.6699 - loss: 0.9104
Epoch 7: val_accuracy improved from 0.67278 to 0.70640, saving model to
best_VGGFaceLike_VGG16_finetune.keras
3626/3626            181s 50ms/step
- accuracy: 0.6698 - loss: 0.9159 - val_accuracy: 0.7064 - val_loss: 0.8141
Epoch 8/10
3625/3626            0s 46ms/step -
accuracy: 0.6898 - loss: 0.8754
Epoch 8: val_accuracy did not improve from 0.70640
3626/3626            179s 49ms/step
- accuracy: 0.6891 - loss: 0.8746 - val_accuracy: 0.6968 - val_loss: 0.8134
Epoch 9/10
3625/3626            0s 46ms/step -
accuracy: 0.7073 - loss: 0.8338
Epoch 9: val_accuracy did not improve from 0.70640
3626/3626            180s 50ms/step
- accuracy: 0.7044 - loss: 0.8391 - val_accuracy: 0.7001 - val_loss: 0.8140
Epoch 10/10
3626/3626            0s 46ms/step -
accuracy: 0.7181 - loss: 0.8046
Epoch 10: val_accuracy did not improve from 0.70640
3626/3626            179s 49ms/step
- accuracy: 0.7182 - loss: 0.8037 - val_accuracy: 0.7001 - val_loss: 0.8573
Epoch 10: early stopping
Restoring model weights from the end of the best epoch: 7.
[INFO] Loading best weights from: best_VGGFaceLike_VGG16_finetune.keras

[INFO] Evaluating model: VGGFaceLike_VGG16 on test set
777/777            13s 17ms/step -
accuracy: 0.6840 - loss: 0.8581
[RESULT] VGGFaceLike_VGG16 - Test Loss: 0.8581, Test Accuracy: 0.6840
777/777            14s 17ms/step

[CLASSIFICATION REPORT] VGGFaceLike_VGG16
              precision    recall  f1-score   support

       angry       0.65      0.53      0.59       888
     disgust       0.81      0.83      0.82       888
        fear       0.61      0.39      0.47       888
       happy       0.77      0.90      0.83       888
     neutral       0.58      0.70      0.63       888
         sad       0.55      0.62      0.58       888
    surprise       0.78      0.82      0.80       888

    accuracy                           0.68      6216
   macro avg       0.68      0.68      0.68      6216
weighted avg       0.68      0.68      0.68      6216
```

```
[INFO] Saving full model to VGGFaceLike_VGG16_best_full.keras

[INFO] Training model (frozen backbone): FaceNetLike_InceptionResNetV2
Epoch 1/30
3626/3626              175s 41ms/step
- accuracy: 0.2693 - loss: 2.3507 - val_accuracy: 0.4176 - val_loss: 1.5521
Epoch 2/30
3626/3626              143s 39ms/step
- accuracy: 0.3116 - loss: 1.9418 - val_accuracy: 0.4363 - val_loss: 1.4864
Epoch 3/30
3626/3626              143s 39ms/step
- accuracy: 0.3429 - loss: 1.7669 - val_accuracy: 0.4479 - val_loss: 1.4577
Epoch 4/30
3626/3626              142s 39ms/step
- accuracy: 0.3587 - loss: 1.6741 - val_accuracy: 0.4564 - val_loss: 1.4423
Epoch 5/30
3626/3626              140s 39ms/step
- accuracy: 0.3715 - loss: 1.6287 - val_accuracy: 0.4546 - val_loss: 1.4350
Epoch 6/30
3626/3626              141s 39ms/step
- accuracy: 0.3806 - loss: 1.6004 - val_accuracy: 0.4583 - val_loss: 1.4281
Epoch 7/30
3626/3626              141s 39ms/step
- accuracy: 0.3881 - loss: 1.5837 - val_accuracy: 0.4604 - val_loss: 1.4190
Epoch 8/30
3626/3626              140s 39ms/step
- accuracy: 0.3922 - loss: 1.5670 - val_accuracy: 0.4598 - val_loss: 1.4172
Epoch 9/30
3626/3626              141s 39ms/step
- accuracy: 0.3984 - loss: 1.5623 - val_accuracy: 0.4648 - val_loss: 1.4104
Epoch 10/30
3626/3626              142s 39ms/step
- accuracy: 0.4025 - loss: 1.5548 - val_accuracy: 0.4640 - val_loss: 1.4028
Epoch 11/30
3626/3626              140s 39ms/step
- accuracy: 0.4008 - loss: 1.5494 - val_accuracy: 0.4649 - val_loss: 1.3991
Epoch 12/30
3626/3626              140s 39ms/step
- accuracy: 0.4044 - loss: 1.5450 - val_accuracy: 0.4653 - val_loss: 1.3952
Epoch 13/30
3626/3626              141s 39ms/step
- accuracy: 0.4031 - loss: 1.5475 - val_accuracy: 0.4754 - val_loss: 1.3904
Epoch 14/30
3626/3626              140s 38ms/step
- accuracy: 0.4043 - loss: 1.5427 - val_accuracy: 0.4688 - val_loss: 1.3918
Epoch 15/30
3626/3626              143s 39ms/step
- accuracy: 0.4066 - loss: 1.5373 - val_accuracy: 0.4686 - val_loss: 1.3841
```

```
Epoch 16/30
3626/3626            142s 39ms/step
- accuracy: 0.4097 - loss: 1.5346 - val_accuracy: 0.4757 - val_loss: 1.3806
Epoch 17/30
3626/3626            139s 38ms/step
- accuracy: 0.4100 - loss: 1.5310 - val_accuracy: 0.4706 - val_loss: 1.3817
Epoch 18/30
3626/3626            139s 38ms/step
- accuracy: 0.4052 - loss: 1.5350 - val_accuracy: 0.4770 - val_loss: 1.3710
Epoch 19/30
3626/3626            140s 39ms/step
- accuracy: 0.4112 - loss: 1.5298 - val_accuracy: 0.4796 - val_loss: 1.3796
Epoch 20/30
3626/3626            141s 39ms/step
- accuracy: 0.4112 - loss: 1.5244 - val_accuracy: 0.4801 - val_loss: 1.3727
Epoch 21/30
3626/3626            139s 38ms/step
- accuracy: 0.4105 - loss: 1.5264 - val_accuracy: 0.4780 - val_loss: 1.3763
Epoch 22/30
3626/3626            139s 38ms/step
- accuracy: 0.4094 - loss: 1.5230 - val_accuracy: 0.4796 - val_loss: 1.3709
Epoch 23/30
3626/3626            139s 38ms/step
- accuracy: 0.4128 - loss: 1.5230 - val_accuracy: 0.4820 - val_loss: 1.3694
Epoch 24/30
3626/3626            139s 38ms/step
- accuracy: 0.4145 - loss: 1.5198 - val_accuracy: 0.4797 - val_loss: 1.3706
Epoch 25/30
3626/3626            138s 38ms/step
- accuracy: 0.4134 - loss: 1.5207 - val_accuracy: 0.4807 - val_loss: 1.3641
Epoch 26/30
3626/3626            139s 38ms/step
- accuracy: 0.4164 - loss: 1.5185 - val_accuracy: 0.4789 - val_loss: 1.3619
Epoch 26: early stopping
Restoring model weights from the end of the best epoch: 23.

[INFO] Fine-tuning top 20 layers of FaceNetLike_InceptionResNetV2
[INFO] Unfroze 15 layers in FaceNetLike_InceptionResNetV2
Epoch 1/10
3626/3626            0s 33ms/step -
accuracy: 0.4220 - loss: 1.5130
Epoch 1: val_accuracy improved from None to 0.49823, saving model to
best_FaceNetLike_InceptionResNetV2_finetune.keras
3626/3626            178s 42ms/step
- accuracy: 0.4259 - loss: 1.4957 - val_accuracy: 0.4982 - val_loss: 1.3152
Epoch 2/10
3625/3626            0s 33ms/step -
accuracy: 0.4397 - loss: 1.4534
```

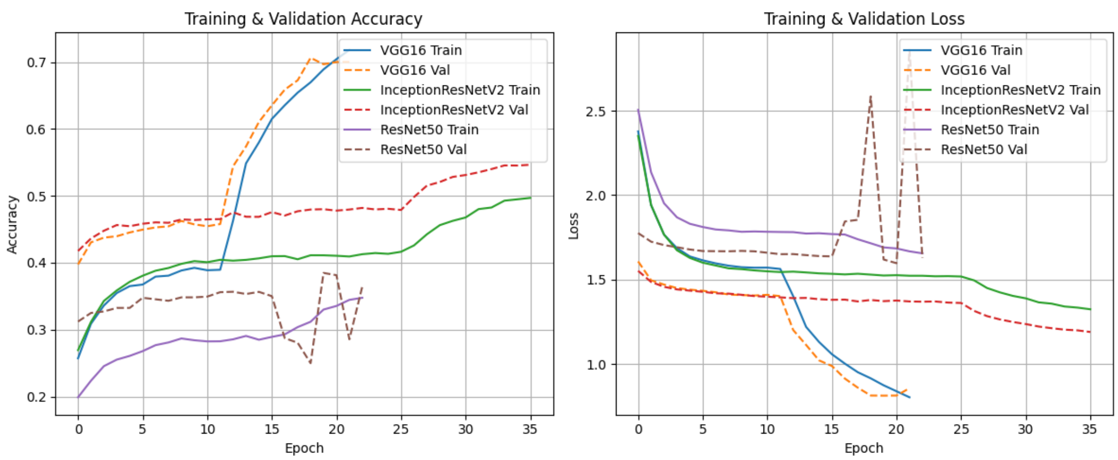
```
Epoch 2: val_accuracy improved from 0.49823 to 0.51528, saving model to
best_FaceNetLike_InceptionResNetV2_finetune.keras
3626/3626            145s 40ms/step
- accuracy: 0.4426 - loss: 1.4506 - val_accuracy: 0.5153 - val_loss: 1.2842
Epoch 3/10
3625/3626            0s 33ms/step -
accuracy: 0.4549 - loss: 1.4303
Epoch 3: val_accuracy improved from 0.51528 to 0.52075, saving model to
best_FaceNetLike_InceptionResNetV2_finetune.keras
3626/3626            145s 40ms/step
- accuracy: 0.4562 - loss: 1.4246 - val_accuracy: 0.5208 - val_loss: 1.2644
Epoch 4/10
3625/3626            0s 33ms/step -
accuracy: 0.4579 - loss: 1.4123
Epoch 4: val_accuracy improved from 0.52075 to 0.52831, saving model to
best_FaceNetLike_InceptionResNetV2_finetune.keras
3626/3626            144s 40ms/step
- accuracy: 0.4627 - loss: 1.4039 - val_accuracy: 0.5283 - val_loss: 1.2495
Epoch 5/10
3625/3626            0s 33ms/step -
accuracy: 0.4687 - loss: 1.3874
Epoch 5: val_accuracy improved from 0.52831 to 0.53121, saving model to
best_FaceNetLike_InceptionResNetV2_finetune.keras
3626/3626            144s 40ms/step
- accuracy: 0.4677 - loss: 1.3893 - val_accuracy: 0.5312 - val_loss: 1.2372
Epoch 6/10
3625/3626            0s 33ms/step -
accuracy: 0.4844 - loss: 1.3598
Epoch 6: val_accuracy improved from 0.53121 to 0.53523, saving model to
best_FaceNetLike_InceptionResNetV2_finetune.keras
3626/3626            145s 40ms/step
- accuracy: 0.4802 - loss: 1.3656 - val_accuracy: 0.5352 - val_loss: 1.2225
Epoch 7/10
3626/3626            0s 33ms/step -
accuracy: 0.4871 - loss: 1.3596
Epoch 7: val_accuracy improved from 0.53523 to 0.54006, saving model to
best_FaceNetLike_InceptionResNetV2_finetune.keras
3626/3626            143s 39ms/step
- accuracy: 0.4826 - loss: 1.3579 - val_accuracy: 0.5401 - val_loss: 1.2126
Epoch 8/10
3625/3626            0s 33ms/step -
accuracy: 0.4881 - loss: 1.3447
Epoch 8: val_accuracy improved from 0.54006 to 0.54553, saving model to
best_FaceNetLike_InceptionResNetV2_finetune.keras
3626/3626            145s 40ms/step
- accuracy: 0.4928 - loss: 1.3411 - val_accuracy: 0.5455 - val_loss: 1.2040
Epoch 9/10
3625/3626            0s 34ms/step -
```

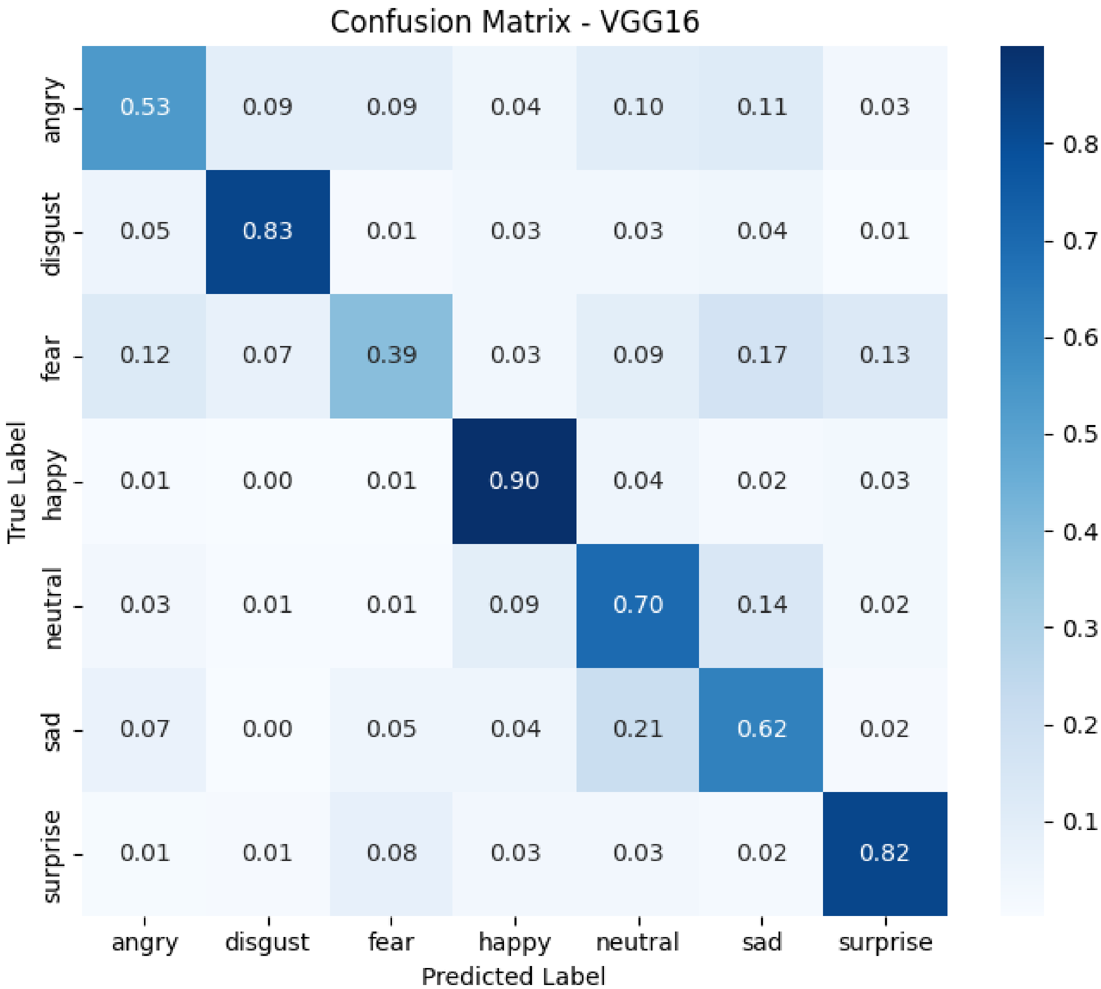
```
accuracy: 0.4960 - loss: 1.3303
Epoch 9: val_accuracy did not improve from 0.54553
3626/3626             143s 39ms/step
- accuracy: 0.4949 - loss: 1.3346 - val_accuracy: 0.5454 - val_loss: 1.1999
Epoch 10/10
3626/3626             0s 34ms/step -
accuracy: 0.4959 - loss: 1.3275
Epoch 10: val_accuracy improved from 0.54553 to 0.54649, saving model to
best_FaceNetLike_InceptionResNetV2_finetune.keras
3626/3626             147s 41ms/step
- accuracy: 0.4969 - loss: 1.3245 - val_accuracy: 0.5465 - val_loss: 1.1902
Restoring model weights from the end of the best epoch: 10.
[INFO] Loading best weights from:
best_FaceNetLike_InceptionResNetV2_finetune.keras

[INFO] Evaluating model: FaceNetLike_InceptionResNetV2 on test set
777/777             21s 27ms/step -
accuracy: 0.5489 - loss: 1.2012
[RESULT] FaceNetLike_InceptionResNetV2 - Test Loss: 1.2012, Test Accuracy:
0.5489
777/777             29s 26ms/step

[CLASSIFICATION REPORT] FaceNetLike_InceptionResNetV2
              precision    recall  f1-score   support

       angry       0.47      0.37      0.41       888
     disgust       0.65      0.78      0.71       888
        fear       0.44      0.29      0.35       888
       happy       0.67      0.71      0.69       888
     neutral       0.47      0.54      0.50       888
         sad       0.44      0.48      0.46       888
    surprise       0.63      0.66      0.65       888

    accuracy                           0.55      6216
   macro avg       0.54      0.55      0.54      6216
weighted avg       0.54      0.55      0.54      6216

[INFO] Saving full model to FaceNetLike_InceptionResNetV2_best_full.keras

[INFO] Training model (frozen backbone): ArcFaceLike_ResNet50
Epoch 1/30
3626/3626             137s 35ms/step
- accuracy: 0.1989 - loss: 2.5050 - val_accuracy: 0.3121 - val_loss: 1.7755
Epoch 2/30
3626/3626             122s 34ms/step
- accuracy: 0.2237 - loss: 2.1353 - val_accuracy: 0.3250 - val_loss: 1.7248
Epoch 3/30
3626/3626             122s 34ms/step
```

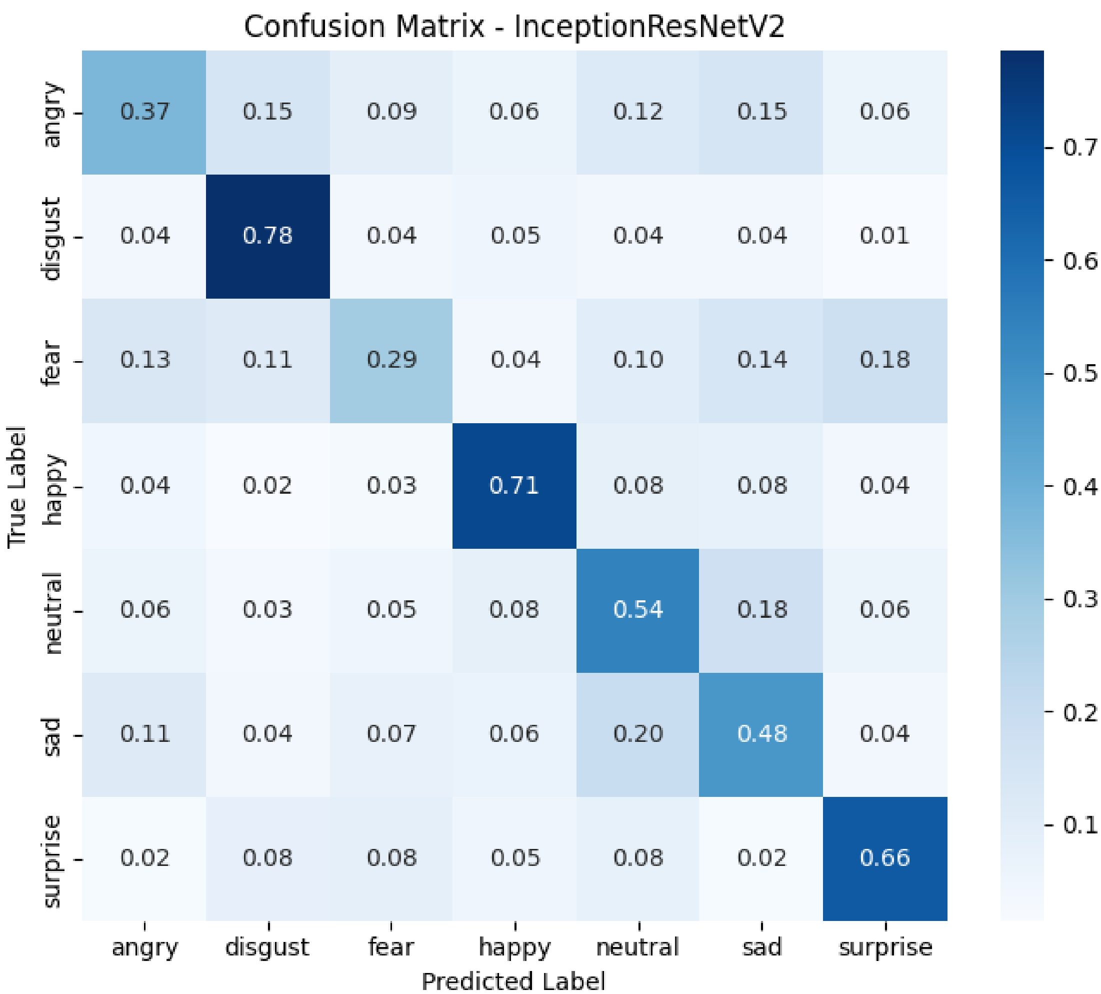
```
- accuracy: 0.2454 - loss: 1.9522 - val_accuracy: 0.3272 - val_loss: 1.7047
Epoch 4/30
3626/3626              122s 34ms/step
- accuracy: 0.2552 - loss: 1.8691 - val_accuracy: 0.3325 - val_loss: 1.6918
Epoch 5/30
3626/3626              122s 34ms/step
- accuracy: 0.2608 - loss: 1.8306 - val_accuracy: 0.3325 - val_loss: 1.6784
Epoch 6/30
3626/3626              122s 34ms/step
- accuracy: 0.2679 - loss: 1.8116 - val_accuracy: 0.3477 - val_loss: 1.6693
Epoch 7/30
3626/3626              121s 33ms/step
- accuracy: 0.2769 - loss: 1.7968 - val_accuracy: 0.3454 - val_loss: 1.6685
Epoch 8/30
3626/3626              121s 33ms/step
- accuracy: 0.2809 - loss: 1.7915 - val_accuracy: 0.3431 - val_loss: 1.6679
Epoch 9/30
3626/3626              122s 34ms/step
- accuracy: 0.2869 - loss: 1.7832 - val_accuracy: 0.3483 - val_loss: 1.6701
Epoch 10/30
3626/3626              121s 33ms/step
- accuracy: 0.2842 - loss: 1.7851 - val_accuracy: 0.3483 - val_loss: 1.6677
Epoch 11/30
3626/3626              121s 33ms/step
- accuracy: 0.2825 - loss: 1.7833 - val_accuracy: 0.3494 - val_loss: 1.6593
Epoch 12/30
3626/3626              121s 33ms/step
- accuracy: 0.2826 - loss: 1.7820 - val_accuracy: 0.3560 - val_loss: 1.6508
Epoch 13/30
3626/3626              122s 34ms/step
- accuracy: 0.2856 - loss: 1.7811 - val_accuracy: 0.3567 - val_loss: 1.6513
Epoch 14/30
3626/3626              122s 34ms/step
- accuracy: 0.2906 - loss: 1.7729 - val_accuracy: 0.3533 - val_loss: 1.6451
Epoch 15/30
3626/3626              122s 34ms/step
- accuracy: 0.2850 - loss: 1.7746 - val_accuracy: 0.3565 - val_loss: 1.6395
Epoch 16/30
3626/3626              122s 34ms/step
- accuracy: 0.2892 - loss: 1.7697 - val_accuracy: 0.3501 - val_loss: 1.6378
Epoch 16: early stopping
Restoring model weights from the end of the best epoch: 13.

[INFO] Fine-tuning top 20 layers of ArcFaceLike_ResNet50
[INFO] Unfroze 14 layers in ArcFaceLike_ResNet50
Epoch 1/10
3625/3626              0s 31ms/step -
accuracy: 0.2871 - loss: 1.7795
```

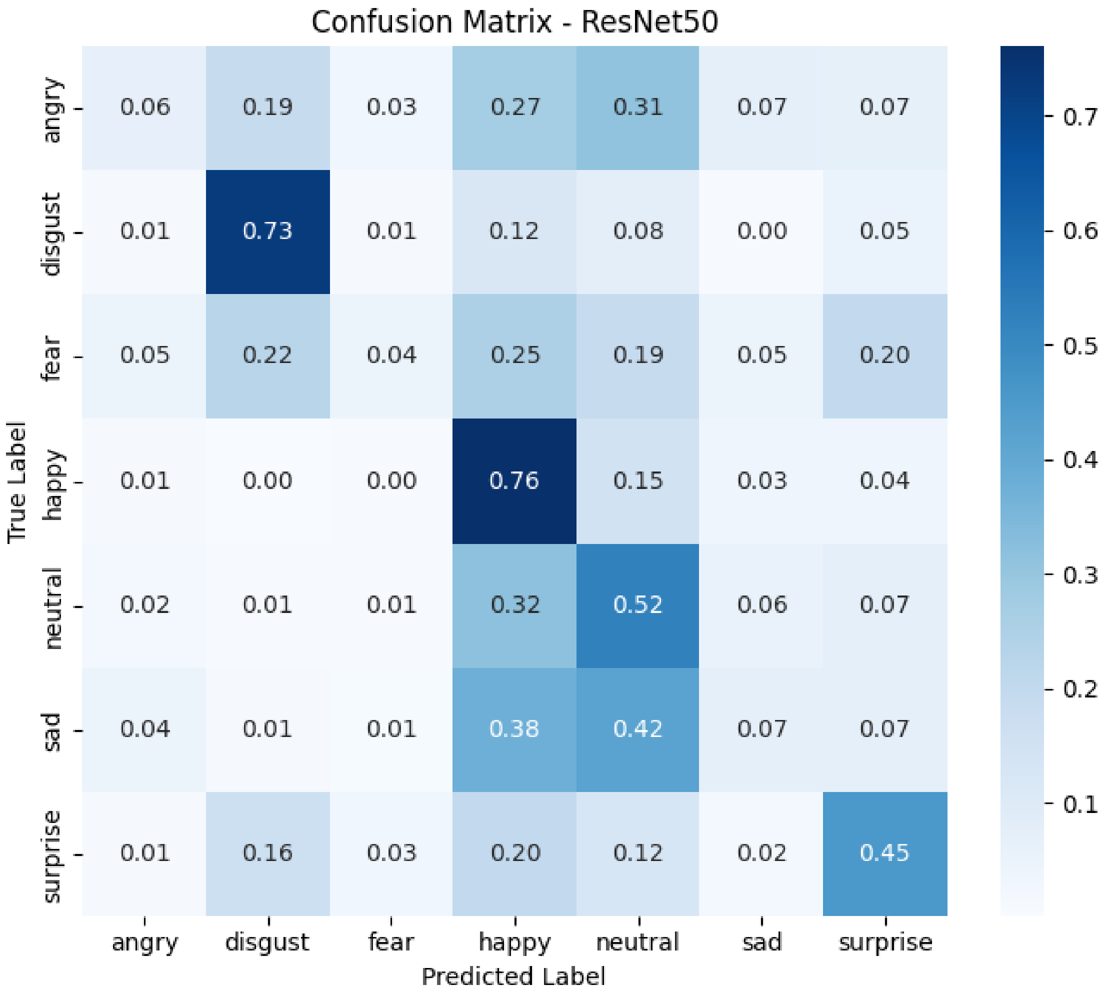
```
Epoch 1: val_accuracy improved from None to 0.28732, saving model to
best_ArcFaceLike_ResNet50_finetune.keras
3626/3626              137s 35ms/step
- accuracy: 0.2930 - loss: 1.7671 - val_accuracy: 0.2873 - val_loss: 1.8444
Epoch 2/10
3625/3626              0s 31ms/step -
accuracy: 0.2939 - loss: 1.7536
Epoch 2: val_accuracy did not improve from 0.28732
3626/3626              123s 34ms/step
- accuracy: 0.3038 - loss: 1.7389 - val_accuracy: 0.2804 - val_loss: 1.8541
Epoch 3/10
3625/3626              0s 31ms/step -
accuracy: 0.3016 - loss: 1.7291
Epoch 3: val_accuracy did not improve from 0.28732
3626/3626              123s 34ms/step
- accuracy: 0.3118 - loss: 1.7153 - val_accuracy: 0.2498 - val_loss: 2.5845
Epoch 4/10
3625/3626              0s 31ms/step -
accuracy: 0.3299 - loss: 1.6888
Epoch 4: val_accuracy improved from 0.28732 to 0.38481, saving model to
best_ArcFaceLike_ResNet50_finetune.keras
3626/3626              123s 34ms/step
- accuracy: 0.3300 - loss: 1.6908 - val_accuracy: 0.3848 - val_loss: 1.6192
Epoch 5/10
3625/3626              0s 31ms/step -
accuracy: 0.3345 - loss: 1.6901
Epoch 5: val_accuracy did not improve from 0.38481
3626/3626              122s 34ms/step
- accuracy: 0.3355 - loss: 1.6845 - val_accuracy: 0.3814 - val_loss: 1.5972
Epoch 6/10
3625/3626              0s 31ms/step -
accuracy: 0.3450 - loss: 1.6692
Epoch 6: val_accuracy did not improve from 0.38481
3626/3626              123s 34ms/step
- accuracy: 0.3445 - loss: 1.6675 - val_accuracy: 0.2856 - val_loss: 2.8597
Epoch 7/10
3625/3626              0s 31ms/step -
accuracy: 0.3407 - loss: 1.6675
Epoch 7: val_accuracy did not improve from 0.38481
3626/3626              123s 34ms/step
- accuracy: 0.3477 - loss: 1.6558 - val_accuracy: 0.3645 - val_loss: 1.6293
Epoch 7: early stopping
Restoring model weights from the end of the best epoch: 4.
[INFO] Loading best weights from: best_ArcFaceLike_ResNet50_finetune.keras

[INFO] Evaluating model: ArcFaceLike_ResNet50 on test set
777/777              9s 12ms/step -
accuracy: 0.3774 - loss: 1.6248
```

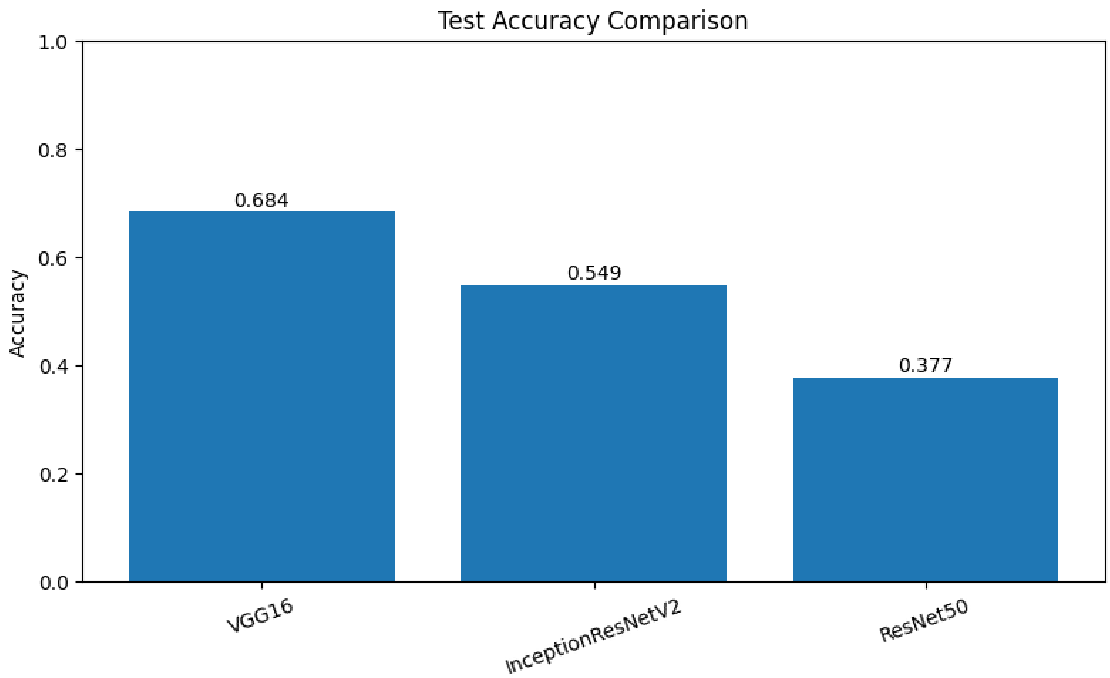
```
[RESULT] ArcFaceLike_ResNet50 - Test Loss: 1.6248, Test Accuracy: 0.3774
777/777              12s 12ms/step

[CLASSIFICATION REPORT] ArcFaceLike_ResNet50
              precision    recall  f1-score   support

       angry       0.31      0.06      0.11       888
     disgust       0.55      0.73      0.63       888
        fear       0.33      0.04      0.08       888
       happy       0.33      0.76      0.46       888
     neutral       0.29      0.52      0.37       888
         sad       0.23      0.07      0.11       888
    surprise       0.48      0.45      0.47       888

    accuracy                           0.38      6216
   macro avg       0.36      0.38      0.32      6216
weighted avg       0.36      0.38      0.32      6216

[INFO] Saving full model to ArcFaceLike_ResNet50_best_full.keras
```




Confusion Matrix - VGG16
True Label
Predicted Label
angry
disgust
fear
happy
neutral
sad
surprise
0.53 0.09 0.09 0.04 0.10 0.11 0.03
0.05 0.83 0.01 0.03 0.03 0.04 0.01
0.12 0.07 0.39 0.03 0.09 0.17 0.13
0.01 0.00 0.01 0.90 0.04 0.02 0.03
0.03 0.01 0.01 0.09 0.70 0.14 0.02
0.07 0.00 0.05 0.04 0.21 0.62 0.02
0.01 0.01 0.08 0.03 0.03 0.02 0.82
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1


Confusion Matrix - InceptionResNetV2
True Label
Predicted Label
angry
disgust
fear
happy
neutral
sad
surprise
0.37 0.15 0.09 0.06 0.12 0.15 0.06
0.04 0.78 0.04 0.05 0.04 0.04 0.01
0.13 0.11 0.29 0.04 0.10 0.14 0.18
0.04 0.02 0.03 0.71 0.08 0.08 0.04
0.06 0.03 0.05 0.08 0.54 0.18 0.06
0.11 0.04 0.07 0.06 0.20 0.48 0.04
0.02 0.08 0.08 0.05 0.08 0.02 0.66
0.7
0.6
0.5
0.4
0.3
0.2
0.1


Confusion Matrix - ResNet50
True Label
angry
disgust
fear
happy
neutral
sad
surprise
0.06 0.19 0.03 0.27 0.31 0.07 0.07
0.01 0.73 0.01 0.12 0.08 0.00 0.05
0.05 0.22 0.04 0.25 0.19 0.05 0.20
0.01 0.00 0.00 0.76 0.15 0.03 0.04
0.02 0.01 0.01 0.32 0.52 0.06 0.07
0.04 0.01 0.01 0.38 0.42 0.07 0.07
0.01 0.16 0.03 0.20 0.12 0.02 0.45
angry
disgust
fear
happy
neutral
sad
surprise
Predicted Label
0.7
0.6
0.5
0.4
0.3
0.2
0.1




```
[INFO] Best model: VGG16 (acc=0.6840)
[INFO] Best model file: VGGFaceLike_VGG16_best_full.keras
[INFO] Copied best model to best_emotion_model.keras
[INFO] Saved class names to class_names.npy: ['angry', 'disgust', 'fear',
'happy', 'neutral', 'sad', 'surprise']
```

[ ]:

Summary of Model Comparison Results (FER2013 Emotion Recognition)

You trained and evaluated three deep-learning models for 7-class facial-emotion recognition (angry, disgust, fear, happy, neutral, sad, surprise) using the FER2013 dataset split into:

29,008 training images

6,216 validation images

6,216 test images

All models were trained in two phases:

Frozen backbone (only classifier head trains)

Fine-tuning (unfreezing top layers with a low learning rate)

The three architectures tested:

VGGFaceLike – VGG16 backbone

FaceNetLike – InceptionResNetV2 backbone

ArcFaceLike – ResNet50 backbone

Overall Results (Test Accuracy) Model Test Accuracy Notes VGGFaceLike (VGG16) 68.4% Best model, strong across all classes FaceNetLike (InceptionResNetV2) 54.9% Moderate results, weaker on difficult emotions ArcFaceLike (ResNet50) 37.7% Struggled significantly Detailed Model Insights 1. VGGFaceLike (VGG16) — Best Model

Final test accuracy: 68.4%

Fine-tuning greatly improved performance (from ~45% → 70% val accuracy).

Strongest predictions:

Happy (F1   0.83)

Disgust (F1   0.82)

Surprise (F1   0.80)

Weaker but acceptable:

Fear (F1 = 0.47) is still difficult for the model.

Most-balanced confusion matrix among all models.

Conclusion: VGG16 learned emotion features best and generalizes well. It is the recommended model for deployment.

2. FaceNetLike (InceptionResNetV2)

Test accuracy: 54.9%

Improved with fine-tuning but still far behind VGG16.

Good on:

Disgust (F1 = 0.71)

Happy (F1 = 0.69)

Poor on:

Fear (F1 = 0.35)

Angry (F1 = 0.41)

Conclusion: Bigger/more complex backbone did not help for FER2013. Overfits more and struggles with subtle emotions.

3. ArcFaceLike (ResNet50)

Test accuracy: 37.7%

Worst model in all evaluations.

Major difficulties with:

Angry (F1 = 0.11)

Fear (F1 = 0.08)

Sad (F1 = 0.11)

Despite fine-tuning, accuracy stayed low.

Conclusion: The ArcFace-style ResNet50 approach is not suitable here without a true ArcFace loss or embedding-based pipeline.

Final Conclusion:

VGGFaceLike (VGG16):

Best test accuracy: 68.4% Best overall F1-scores and stability Saved as: best_emotion_model.keras

This model should be used for your webcam real-time emotion detection feature.