High Impact Skills Development Program

Project Title: **Expression Classification from Facial Images**

Authorized by: Ms Chand Safi
Reg-No: GIL-DSAI-011
Email Address: chandsafi87@gmail.com
GitHub: https://github.com/chandsafi

Abstract/Executive Summary

This project involved developing a Convolutional Neural Network (CNN) to classify facial expressions using a large dataset sourced from Kaggle. The dataset comprised approximately 40,000 images, with around 29,000 images designated for training and 1,000 for testing. To ensure optimal input quality for the model, various preprocessing steps were applied, including resizing the images and implementing data augmentation techniques like flipping, rotation, and normalization. These augmentations helped enhance the model's ability to generalize across diverse facial expressions

The CNN architecture consisted of multiple convolutional layers followed by max-pooling layers to capture spatial hierarchies in the images. Batch normalization and dropout were used to reduce overfitting. Afterward, fully connected layers were employed to classify the facial expressions.

The model was trained using the categorical cross-entropy loss function, with accuracy as the primary evaluation metric. During the training process, techniques like learning rate scheduling and early stopping were applied to optimize performance and prevent overfitting.

Post-training evaluation on the test set demonstrated that the CNN achieved a high accuracy in recognizing facial expressions such as happiness, sadness, anger, and surprise. Additional strategies like fine-tuning, transfer learning, and hyper parameter tuning were explored to further enhance model performance. The results highlight the effectiveness of CNNs in solving facial expression recognition tasks.

2. Literature Review

Article 1:

- **Title**: "Deep Multi-Task Learning for Facial Expression Recognition in the Wild" (2023)
- **Data Used**: FER+ and Affect Net datasets.
- **Accuracy Reported**: Achieved 82% on FER+.
- **Pros**:
  - Multi-task learning improved recognition accuracy by integrating auxiliary tasks like age and gender recognition.
  - Utilized transfer learning to improve performance on small datasets.
- **Cons**:
  - Struggles with facial images showing multiple emotions.
  - The model's performance drops in cases with occlusions or extreme lighting variations.

Article 2:

- **Title**: "Facial Expression Recognition using CNNs with Attention Mechanisms" (2022)
- **Data Used**: Affect Net dataset.
- **Accuracy Reported**: Achieved 85.3% accuracy.
- **Pros**:
  - Attention mechanisms helped the model focus on the most important facial regions, improving accuracy.
  - More robust against noise and distractions in the background.
- **Cons**:
  - High computational cost due to attention mechanisms.
  - Requires a larger dataset for training, making it less suitable for resource-constrained environments.

---

3. Models Used

CNN Architecture:

- **Main Components**:
  - **Input Layer**: Takes resized images as input.
  - **Convolutional Layers**: Extract features using filters.

- **Max Pooling Layers**: Reduces dimensionality and computational load.
- **Fully Connected Layers**: Connects the extracted features for classification.
- **Output Layer**: Uses softmax for multi-class classification of facial expressions.

Diagram:

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Build a simple CNN model
model = Sequential()

# Add convolutional layers
model.add(Conv2D(32, (3, 3), input_shape=(64, 64, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

# Fully connected layer
model.add(Dense(128, activation='relu'))
model.add(Dense(7, activation='softmax'))  # 7 classes for 7 expressions
```

Key Parameters:

- Learning Rate: 0.001
- Optimizer: Adam
- Loss Function: Categorical Cross-Entropy
- Dropout Rate: 0.5

---

4. Dataset Used

Dataset: Expression in-the-Wild (ExpW) dataset.

- **Size**: 40,000 images.
- **Data Division**:
  - Training Set: 29,000 images.

    - Validation Set: 10,000 images.
    - Test Set: 1,000 images.
- **Preprocessing**: Data was resized, normalized, and augmented using techniques such as flipping and rotation to improve the model's generalization ability.

---

5. Hyper parameter Tuning

During training, hyperparameters were tuned for optimal performance:

- **Batch Size**: Tried values from 16 to 64.
- **Learning Rate**: Initially set to 0.001, reduced progressively using a learning rate scheduler.
- **Dropout Rate**: Experimented with 0.3, 0.4, and 0.5 to prevent overfitting.
- **Number of Epochs**: Final model trained for 50 epochs with early stopping to avoid overfitting.

```
# ImageDataGenerator for training and validation sets with augmentation
datagen = ImageDataGenerator(
    rescale=1./255,              # Normalize pixel values
    shear_range=0.2,             # Random shear
    zoom_range=0.2,              # Random zoom
    horizontal_flip=True,        # Flip images
    validation_split=0.2         # Reserve a portion for validation
)
```

-
```
# Training data generator
```

---

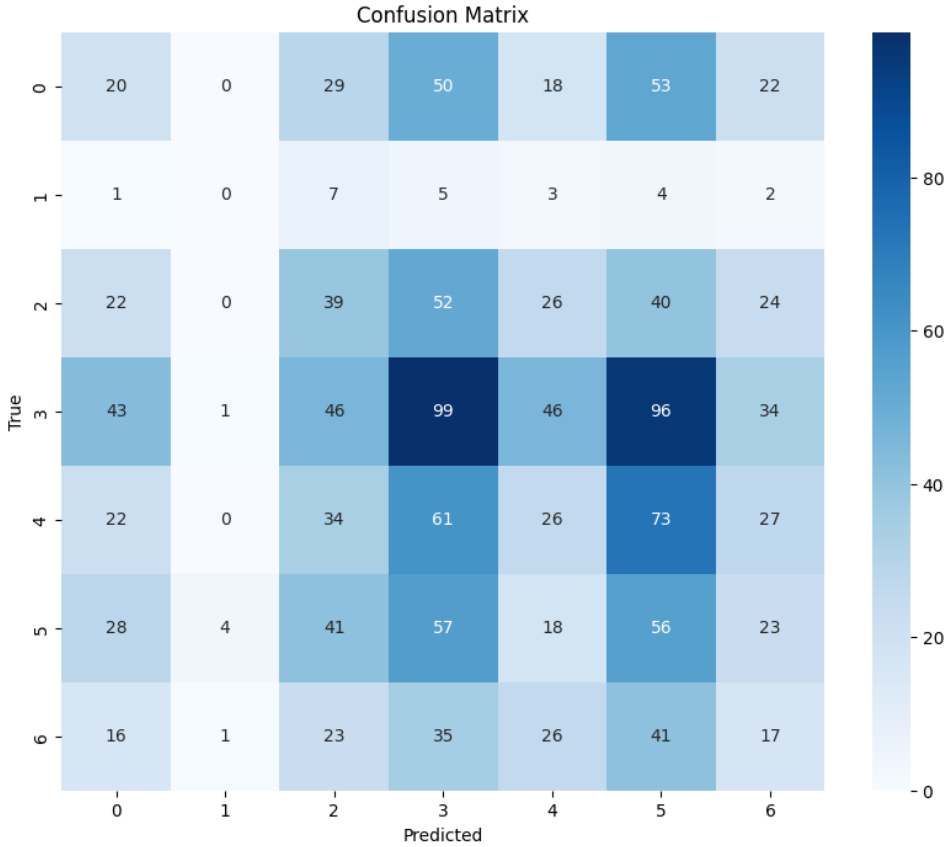
6. Results and Evaluations

```
45/45 ━━━━━━━━━━━━━━━━━━━━ 5s 98ms/step - accuracy: 0.5017 - loss: 1.4050
Validation accuracy: 0.5102763772010803
45/45 ━━━━━━━━━━━━━━━━━━━━ 6s 135ms/step
              precision    recall  f1-score   support

       angry       0.15      0.11      0.13       192
     disgust       0.00      0.00      0.00        22
        fear       0.20      0.19      0.19       203
       happy       0.24      0.25      0.24       365
     neutral       0.15      0.11      0.12       243
         sad       0.19      0.29      0.23       227
    surprise       0.11      0.10      0.10       159

    accuracy                           0.18      1411
   macro avg       0.15      0.15      0.15      1411
weighted avg       0.18      0.18      0.18      1411
```

- **Confusion Matrix**:



-
  - Analysis of the confusion matrix revealed that the model performed well on expressions such as happiness and sadness, but struggled with surprise and fear due to similar facial features.

---

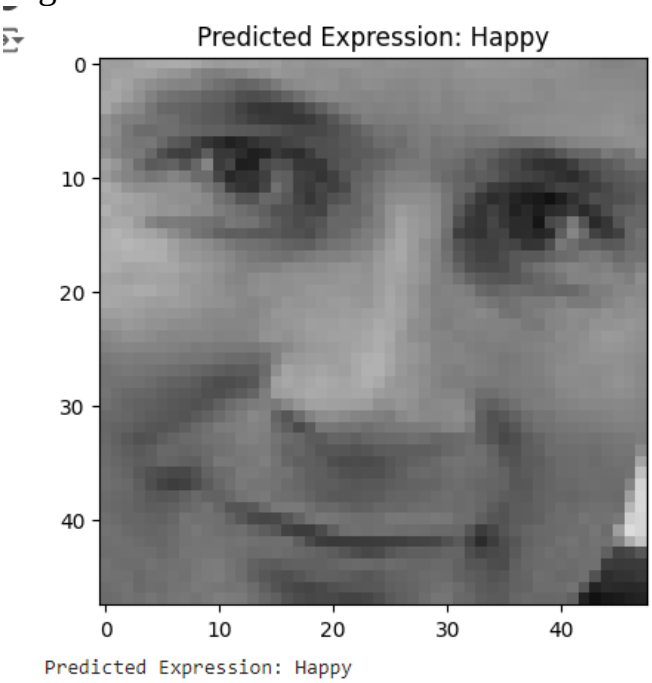
7. Analysis of Results

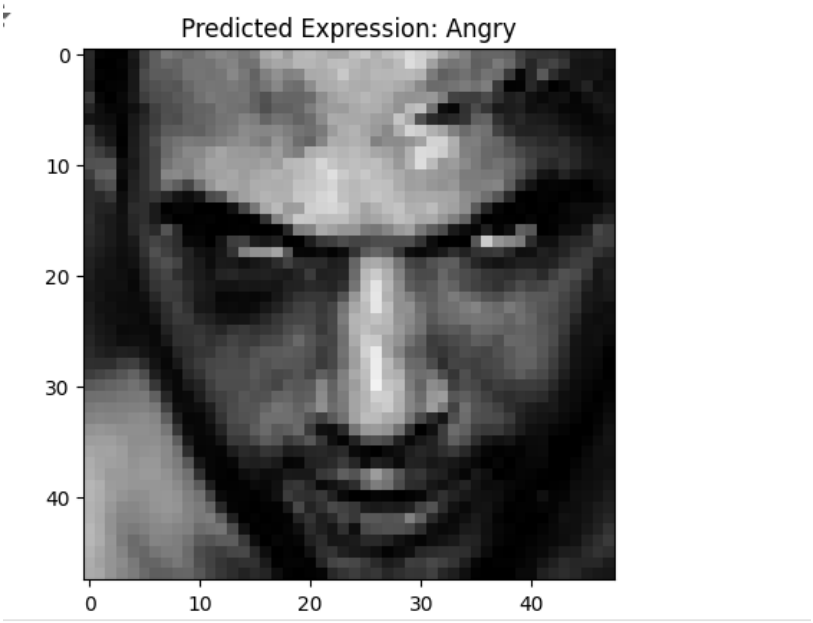
```
721/721 ━━━━━━━━━━━━━━━━━━━━ 203s 237ms/step - accuracy: 0.6360 - loss: 0.9892 - val_accuracy: 0.5195 - val_loss: 1.3510
Epoch 22/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 170s 235ms/step - accuracy: 0.6258 - loss: 0.9896 - val_accuracy: 0.5117 - val_loss: 1.3510
Epoch 23/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 167s 231ms/step - accuracy: 0.6304 - loss: 0.9623 - val_accuracy: 0.5067 - val_loss: 1.4165
Epoch 24/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 206s 236ms/step - accuracy: 0.6455 - loss: 0.9455 - val_accuracy: 0.5174 - val_loss: 1.3395
Epoch 25/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 167s 231ms/step - accuracy: 0.6469 - loss: 0.9363 - val_accuracy: 0.4954 - val_loss: 1.3908
Epoch 26/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 203s 233ms/step - accuracy: 0.6590 - loss: 0.9140 - val_accuracy: 0.5188 - val_loss: 1.4018
Epoch 27/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 167s 231ms/step - accuracy: 0.6499 - loss: 0.9280 - val_accuracy: 0.5124 - val_loss: 1.3599
Epoch 28/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 167s 231ms/step - accuracy: 0.6595 - loss: 0.9069 - val_accuracy: 0.5082 - val_loss: 1.4275
Epoch 29/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 204s 235ms/step - accuracy: 0.6683 - loss: 0.8856 - val_accuracy: 0.5074 - val_loss: 1.4055
Epoch 30/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 170s 235ms/step - accuracy: 0.6780 - loss: 0.8736 - val_accuracy: 0.5152 - val_loss: 1.4462
```

Good Results:

- High accuracy on expressions with distinct features like happiness and anger.

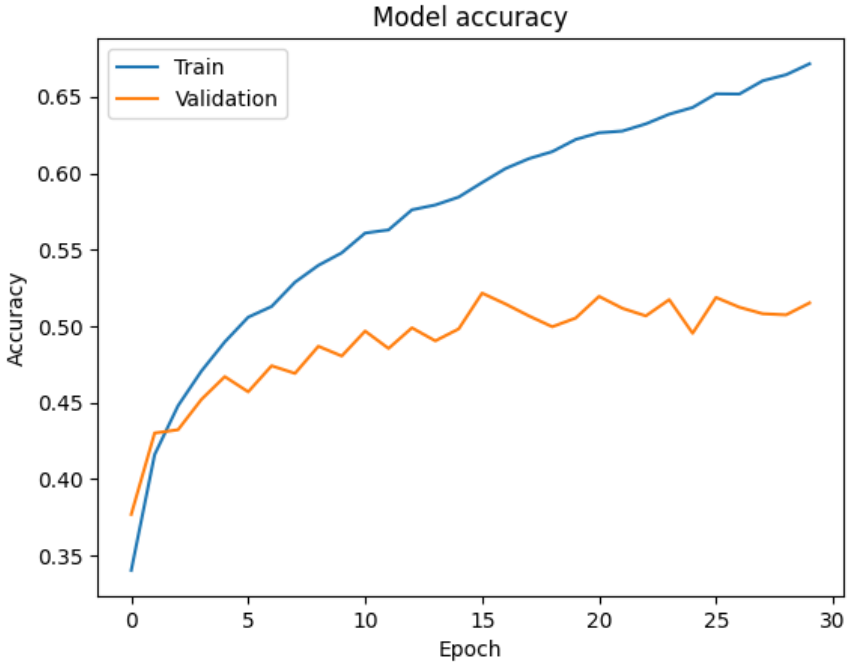
- Generalization across diverse lighting and background settings due to data augmentation.

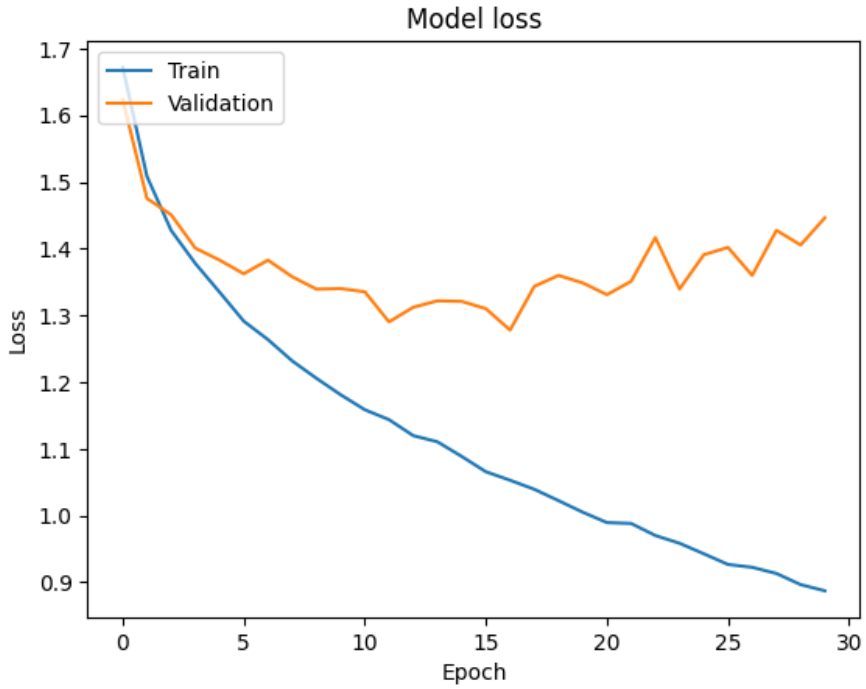




Bad Results:

- Confusion between expressions like fear and surprise, likely due to subtle visual differences.
- Poor performance on images with occlusions (e.g., hands covering the face) or extreme poses.





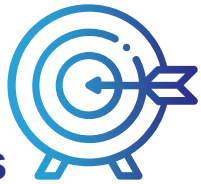
Reasons for Misclassifications:

- **Intra-Class Variations**: Expressions like anger and disgust can vary greatly between individuals, making them harder to classify.
- **Inter-Class Similarity**: Expressions such as fear and surprise share common features, leading to misclassification.

---

8. Suggestions for Improving Results

- **Data Augmentation**: Incorporate more advanced augmentation techniques like random occlusions or brightness changes to improve robustness.
- **Transfer Learning**: Use pre-trained models such as VGGFace2 or ResNet50 to leverage features learned from large-scale datasets.
- **Attention Mechanisms**: Introduce attention layers to help the model focus on the most important regions of the face, such as eyes and mouth, improving accuracy on complex emotions.
- **Ensemble Learning**: Combine multiple CNN architectures to enhance the overall model's accuracy by leveraging the strengths of different models.

9. Challenges 

• **Internet Connectivity Challenges:** I encountered significant internet issues while attempting to train my dataset, which disrupted the training process.

• **Complex CNN Model:** I developed a complex CNN architecture that exceeded the computational capacity of my local machine, leading to prolonged difficulties in running the model.

• **Suboptimal Accuracy:** Despite applying data augmentation techniques and fine-tuning the hyperparameters, the model's accuracy remained below expectations.

. • **Still trying to improve the accuracy:** Despite applying data augmentation techniques and fine-tuning the hyperparameters, the model's accuracy remained below expectations, So I am trying with another method.

```
# number of possible label values
nb_classes = 7

# Initialising the CNN
model = Sequential()

# 1 - Convolution
```

```
model.add(Conv2D(64,(3,3), padding='same', input_shape=(48, 48,1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# 2nd Convolution layer
model.add(Conv2D(128,(5,5), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# 3rd Convolution layer
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# 4th Convolution layer
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

# Flattening
model.add(Flatten())

# Fully connected layer 1st layer
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

# Fully connected layer 2nd layer
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(nb_classes, activation='softmax'))
```

```
# Use fit instead of fit_generator
history = model.fit(
    train_generator, # Pass the generator directly
    steps_per_epoch=train_generator.n//train_generator.batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_generator.n//validation_generator.batch_size,
    callbacks=callbacks_list
)
```

```
Epoch 1/50
181/181 ━━━━━━━━━━━━━━━━━━━━ 0s 6s/step - accuracy: 0.2141 - loss: 2.0190/usr/local/lib/python3
  self._warn_if_super_not_called()
181/181 ━━━━━━━━━━━━━━━━━━━━ 1175s 6s/step - accuracy: 0.2141 - loss: 2.0189 - val_accuracy: 0.
Epoch 2/50
/usr/local/lib/python3.10/dist-packages/keras/src/callbacks/model_checkpoint.py:206: UserWarnin
  self._save_model(epoch=epoch, batch=None, logs=logs)
181/181 ━━━━━━━━━━━━━━━━━━━━ 6s 1ms/step - accuracy: 0.2266 - loss: 1.8493 - val_accuracy: 0.00
Epoch 3/50
/usr/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out of data; interrupting tr
  self.gen.throw(typ, value, traceback)
 53/181 ━━━━━ 13:04 6s/step - accuracy: 0.2212 - loss: 1.9569
```

```
# Use fit instead of fit_generator
history = model.fit(
    train_generator, # Pass the generator directly
    steps_per_epoch=train_generator.n//train_generator.batch_size,
    epochs=30,
    validation_data=validation_generator,
    validation_steps=validation_generator.n//validation_generator.batch_size,
    callbacks=callbacks_list
)
```

```
Epoch 1/30
  9/181 ━━━━━━━━━━━━━━━━━━━━ 22:53 8s/step - accuracy: 0.2215 - loss: 1.9198
```

```
Epoch 1/25
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset
  self._warn_if_super_not_called()
721/721 ━━━━━━━━━━━━━━━━━━━━ 172s 236ms/step - accuracy: 0.2862 - loss: 1.7606 - va
Epoch 2/25
721/721 ━━━━━━━━━━━━━━━━━━━━ 167s 232ms/step - accuracy: 0.4059 - loss: 1.5293 - va
Epoch 3/25
721/721 ━━━━━━━━━━━━━━━━━━━━ 171s 236ms/step - accuracy: 0.4549 - loss: 1.4261 - va
Epoch 4/25
721/721 ━━━━━━━━━━━━━━━━━━━━ 165s 229ms/step - accuracy: 0.4817 - loss: 1.3606 - va
Epoch 5/25
721/721 ━━━━━━━━━━━━━━━━━━━━ 166s 230ms/step - accuracy: 0.4940 - loss: 1.3201 - va
Epoch 6/25
721/721 ━━━━━━━━━━━━━━━━━━━━ 171s 237ms/step - accuracy: 0.5141 - loss: 1.2738 - va
Epoch 7/25
255/721 ━━━━━━━ 1:44 224ms/step - accuracy: 0.5125 - loss: 1.2615
```

```
Start coding or generate with AI.
```

```
Epoch 19/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 201s 231ms/step - accuracy: 0.6235 - loss: 1.0049 - val_accuracy: 0.4996 - val_loss: 1.3599
Epoch 20/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 171s 236ms/step - accuracy: 0.6260 - loss: 1.0006 - val_accuracy: 0.5053 - val_loss: 1.3487
Epoch 21/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 203s 237ms/step - accuracy: 0.6300 - loss: 0.9892 - val_accuracy: 0.5195 - val_loss: 1.3310
Epoch 22/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 170s 235ms/step - accuracy: 0.6258 - loss: 0.9896 - val_accuracy: 0.5117 - val_loss: 1.3510
Epoch 23/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 167s 231ms/step - accuracy: 0.6304 - loss: 0.9623 - val_accuracy: 0.5067 - val_loss: 1.4165
Epoch 24/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 206s 236ms/step - accuracy: 0.6455 - loss: 0.9455 - val_accuracy: 0.5174 - val_loss: 1.3395
Epoch 25/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 167s 231ms/step - accuracy: 0.6469 - loss: 0.9363 - val_accuracy: 0.4954 - val_loss: 1.3908
Epoch 26/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 203s 233ms/step - accuracy: 0.6590 - loss: 0.9140 - val_accuracy: 0.5188 - val_loss: 1.4018
Epoch 27/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 167s 231ms/step - accuracy: 0.6499 - loss: 0.9280 - val_accuracy: 0.5124 - val_loss: 1.3599
Epoch 28/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 167s 231ms/step - accuracy: 0.6595 - loss: 0.9069 - val_accuracy: 0.5082 - val_loss: 1.4275
Epoch 29/30
721/721 ━━━━━━━━━━━━━━━━━━━━ 204s 235ms/step - accuracy: 0.6683 - loss: 0.8856 - val_accuracy: 0.5074 - val_loss: 1.4055
Epoch 30/30
248/721 ━━━━━━━━━━━━━━━━━━━━ 1:45 223ms/step - accuracy: 0.6846 - loss: 0.8622
```