Indonesian Speech Emotion Recognition - MFCC + BiLSTM + Standard Augmentation

CELL 1: Import Libraries

```
import os
import re
import random
import logging
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from glob import glob

# Audio processing
import librosa
import librosa.display

# Machine Learning
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split,
StratifiedShuffleSplit
from sklearn.metrics import (classification_report, confusion_matrix,
                             roc_curve, auc, precision_recall_curve,
                             accuracy_score, f1_score)

# Deep Learning
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout,
BatchNormalization, LSTM, Bidirectional
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,
ReduceLROnPlateau
from tensorflow.keras.regularizers import l2

# Set random seeds for reproducibility
np.random.seed(42)
tf.random.set_seed(42)
random.seed(42)

# Set up logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

print("Libraries imported successfully!")
```

```
WARNING:tensorflow:From c:\Users\LENOVO\AppData\Local\Programs\Python\
Python310\lib\site-packages\keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

Libraries imported successfully!
```

CELL 2: Dataset Loading and Exploration

```
DATASET_PATH = "datasets/IndoWaveSentiment/"
SAMPLE_RATE = 16000  # IndoWaveSentiment sample rate
N_MFCC = 40

def parse_filename(filename):
    """
    Parse IndoWaveSentiment filename format: XX-XX-XX-XX.wav
    Actors (01-10): odd=male, even=female
    Emotion Class: (01)neutral, (02)happy, (03)surprise, (04)disgust,
(05)disappointed
    Intensity: (01)normal, (02)strong
    Repetition: (01)first, (02)second, (03)third
    """
    parts = filename.replace('.wav', '').split('-')
    if len(parts) != 4:
        return None

    try:
        actor_id = int(parts[0])
        emotion_code = int(parts[1])
        intensity_code = int(parts[2])
        repetition = int(parts[3])

        # Gender classification
        gender = 'male' if actor_id % 2 == 1 else 'female'

        # Emotion mapping
        emotion_map = {1: 'neutral', 2: 'happy', 3: 'surprised', 4:
'disgusted', 5: 'disappointed'}
        emotion = emotion_map.get(emotion_code, 'unknown')

        # Intensity mapping
        intensity = 'normal' if intensity_code == 1 else 'strong'

        return {
            'filename': filename,
            'actor_id': actor_id,
            'gender': gender,
            'emotion': emotion,
            'emotion_code': emotion_code,
```

```
            'intensity': intensity,
            'repetition': repetition
        }
    except (ValueError, IndexError):
        return None

def load_dataset_metadata():
    """Load and parse all audio files in the dataset"""
    audio_files = glob(os.path.join(DATASET_PATH, "**", "*.wav"),
recursive=True)

    if not audio_files:
        raise FileNotFoundError(f"No WAV files found in
{DATASET_PATH}")

    metadata = []
    for file_path in audio_files:
        filename = os.path.basename(file_path)
        parsed = parse_filename(filename)

        if parsed:
            parsed['filepath'] = file_path
            metadata.append(parsed)
        else:
            logger.warning(f"Could not parse filename: {filename}")

    return pd.DataFrame(metadata)

# Load dataset
print("Loading IndoWaveSentiment dataset...")
df = load_dataset_metadata()

print(f"Dataset loaded: {len(df)} audio files")
print("\nDataset distribution:")
print("By Emotion:")
print(df['emotion'].value_counts())
print("\nBy Gender:")
print(df['gender'].value_counts())
print("\nBy Intensity:")
print(df['intensity'].value_counts())

# Display sample data
print("\nSample data:")
print(df.head())
```

```
Loading IndoWaveSentiment dataset...
Dataset loaded: 300 audio files

Dataset distribution:
By Emotion:
```

```
emotion
neutral         60
happy           60
surprised       60
disgusted       60
disappointed    60
Name: count, dtype: int64

By Gender:
gender
male      150
female    150
Name: count, dtype: int64

By Intensity:
intensity
normal    150
strong    150
Name: count, dtype: int64

Sample data:
          filename  actor_id gender  emotion  emotion_code
intensity  \
0  01-01-01-01.wav         1   male  neutral             1    normal

1  01-01-01-02.wav         1   male  neutral             1    normal

2  01-01-01-03.wav         1   male  neutral             1    normal

3  01-01-02-01.wav         1   male  neutral             1    strong

4  01-01-02-02.wav         1   male  neutral             1    strong


   repetition                                              filepath
0           1  datasets/IndoWaveSentiment\Actor_01\01-01-01-0...
1           2  datasets/IndoWaveSentiment\Actor_01\01-01-01-0...
2           3  datasets/IndoWaveSentiment\Actor_01\01-01-01-0...
3           1  datasets/IndoWaveSentiment\Actor_01\01-01-02-0...
4           2  datasets/IndoWaveSentiment\Actor_01\01-01-02-0...
```

CELL 3: Audio Preprocessing Functions

```python
def load_audio(filepath, sr=SAMPLE_RATE, duration=None):
    """Load and preprocess audio file"""
    try:
        audio, _ = librosa.load(filepath, sr=sr, duration=duration)
        return audio
    except Exception as e:
        logger.error(f"Error loading {filepath}: {e}")
```

```
        return None

def extract_mfcc_features(audio, sr=SAMPLE_RATE, n_mfcc=N_MFCC):
    """Extract MFCC features from audio"""
    try:
        # Extract MFCC features
        mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=n_mfcc)
        # Take mean across time axis
        mfcc_mean = np.mean(mfcc.T, axis=0)
        return mfcc_mean
    except Exception as e:
        logger.error(f"Error extracting MFCC: {e}")
        return None

def normalize_audio(audio):
    """Normalize audio to [-1, 1] range"""
    if np.max(np.abs(audio)) > 0:
        return audio / np.max(np.abs(audio))
    return audio

def remove_silence(audio, top_db=20):
    """Remove silence from audio"""
    try:
        audio_trimmed, _ = librosa.effects.trim(audio, top_db=top_db)
        return audio_trimmed if len(audio_trimmed) > 0 else audio
    except:
        return audio

print("Audio preprocessing functions defined!")
```

```
Audio preprocessing functions defined!
```

CELL 4: Standard Audio Augmentation Functions

```
def add_noise(audio, noise_factor=0.005):
    """Add random noise to audio"""
    noise = np.random.randn(len(audio))
    augmented_audio = audio + noise_factor * noise
    return normalize_audio(augmented_audio)

def time_stretch(audio, rate=None):
    """Apply time stretching to audio"""
    if rate is None:
        rate = np.random.uniform(0.8, 1.2)  # Random stretch rate
    try:
        return librosa.effects.time_stretch(audio, rate=rate)
    except:
        return audio

def pitch_shift(audio, sr, n_steps=None):
```

```
    """Apply pitch shifting to audio"""
    if n_steps is None:
        n_steps = np.random.uniform(-3, 3)  # Random pitch shift

    try:
        return librosa.effects.pitch_shift(audio, sr=sr,
n_steps=n_steps)
    except:
        return audio

def time_shift(audio, shift_max=0.2):
    """Apply time shifting (circular shift) to audio"""
    shift = int(np.random.uniform(-shift_max, shift_max) * len(audio))
    return np.roll(audio, shift)

def speed_change(audio, speed_factor=None):
    """Change speed of audio"""
    if speed_factor is None:
        speed_factor = np.random.uniform(0.9, 1.1)

    try:
        # Change speed by resampling
        indices = np.round(np.arange(0, len(audio),
speed_factor)).astype(int)
        indices = indices[indices < len(audio)]
        return audio[indices]
    except:
        return audio

def apply_augmentation(audio, sr, augmentation_type):
    """Apply specific augmentation technique"""
    if augmentation_type == 'noise':
        return add_noise(audio)
    elif augmentation_type == 'time_stretch':
        return time_stretch(audio)
    elif augmentation_type == 'pitch_shift':
        return pitch_shift(audio, sr)
    elif augmentation_type == 'time_shift':
        return time_shift(audio)
    elif augmentation_type == 'speed_change':
        return speed_change(audio)
    else:
        return audio

print("Standard augmentation functions defined!")
```

```
Standard augmentation functions defined!
```

CELL 5: Feature Extraction with Augmentation

```
def extract_features_with_standard_augmentation(df,
augmentation_ratio=2):
    """
    Extract MFCC features with standard augmentation techniques
    augmentation_ratio: number of augmented samples per original
sample
    """
    features = []
    labels = []
    genders = []
    actors = []
    intensities = []

    # Standard augmentation types
    augmentation_types = ['noise', 'time_stretch', 'pitch_shift',
'time_shift', 'speed_change']

    print("Extracting features with standard augmentation...")

    for idx, row in df.iterrows():
        # Load original audio
        audio = load_audio(row['filepath'])
        if audio is None:
            continue

        # Preprocess audio
        audio = remove_silence(audio)
        audio = normalize_audio(audio)

        # Extract original features
        mfcc_features = extract_mfcc_features(audio)
        if mfcc_features is not None:
            features.append(mfcc_features)
            labels.append(row['emotion'])
            genders.append(row['gender'])
            actors.append(row['actor_id'])
            intensities.append(row['intensity'])

            # Apply standard augmentation
            for i in range(augmentation_ratio):
                # Randomly select augmentation type
                aug_type = np.random.choice(augmentation_types)

                # Apply augmentation
                augmented_audio = apply_augmentation(audio,
SAMPLE_RATE, aug_type)

                # Extract features from augmented audio
```

```
                aug_mfcc_features =
extract_mfcc_features(augmented_audio)
                if aug_mfcc_features is not None:
                    features.append(aug_mfcc_features)
                    labels.append(row['emotion'])
                    genders.append(row['gender'])
                    actors.append(row['actor_id'])
                    intensities.append(row['intensity'])

        if (idx + 1) % 50 == 0:
            print(f"Processed {idx + 1}/{len(df)} files...")

    # Create final dataset
    feature_df = pd.DataFrame({
        'features': features,
        'emotion': labels,
        'gender': genders,
        'actor_id': actors,
        'intensity': intensities
    })

    print(f"Feature extraction completed!")
    print(f"Original samples: {len(df)}")
    print(f"Total samples (with augmentation): {len(feature_df)}")

    return feature_df

# Extract features
feature_df = extract_features_with_standard_augmentation(df,
augmentation_ratio=2)

# Display augmented dataset statistics
print("\nAugmented dataset distribution:")
print("By Emotion:")
print(feature_df['emotion'].value_counts())
print("\nBy Gender:")
print(feature_df['gender'].value_counts())
```

```
Extracting features with standard augmentation...
Processed 50/300 files...
Processed 100/300 files...
Processed 150/300 files...
Processed 200/300 files...
Processed 250/300 files...
Processed 300/300 files...
Feature extraction completed!
Original samples: 300
Total samples (with augmentation): 900

Augmented dataset distribution:
```

```
By Emotion:
emotion
neutral          180
happy            180
surprised        180
disgusted        180
disappointed     180
Name: count, dtype: int64

By Gender:
gender
male      450
female    450
Name: count, dtype: int64
```

CELL 5A: MFCC Visualization and Mapping

```
def save_mfcc_mapping(feature_df, save_dir='mfcc_outputs'):
    """Save MFCC features to various formats for analysis"""
    os.makedirs(save_dir, exist_ok=True)

    print("Saving MFCC mappings...")

    # 1. Save as CSV (most accessible format)
    mfcc_df = pd.DataFrame()
    for idx, row in feature_df.iterrows():
        sample_data = {
            'sample_id': idx,
            'emotion': row['emotion'],
            'gender': row['gender'],
            'actor_id': row['actor_id'],
            'intensity': row['intensity']
        }
        # Add MFCC coefficients as separate columns
        for i, coeff in enumerate(row['features']):
            sample_data[f'mfcc_{i+1}'] = coeff

        mfcc_df = pd.concat([mfcc_df, pd.DataFrame([sample_data])],
ignore_index=True)

    mfcc_df.to_csv(os.path.join(save_dir, 'mfcc_features.csv'),
index=False)

    # 2. Save statistics by emotion
    stats_by_emotion = {}
    for emotion in feature_df['emotion'].unique():
        emotion_data = feature_df[feature_df['emotion'] == emotion]
        emotion_features = np.array(emotion_data['features'].tolist())
```

```
        stats_by_emotion[emotion] = {
            'count': len(emotion_data),
            'mfcc_mean': np.mean(emotion_features, axis=0),
            'mfcc_std': np.std(emotion_features, axis=0)
        }

    # Save stats as CSV
    stats_df = pd.DataFrame()
    for emotion, stats in stats_by_emotion.items():
        for i, (mean_val, std_val) in
enumerate(zip(stats['mfcc_mean'], stats['mfcc_std'])):
            stats_df = pd.concat([stats_df, pd.DataFrame([{
                'emotion': emotion,
                'mfcc_coeff': i+1,
                'mean_value': mean_val,
                'std_value': std_val,
                'sample_count': stats['count']
            }])], ignore_index=True)

    stats_df.to_csv(os.path.join(save_dir, 'mfcc_statistics.csv'),
index=False)

    print(f"MFCC outputs saved to '{save_dir}/':")
    print(f"- mfcc_features.csv (all MFCC features)")
    print(f"- mfcc_statistics.csv (statistics by emotion)")

    return stats_by_emotion

def display_mfcc_samples(feature_df, n_samples=3):
    """Display MFCC feature samples in output"""
    print("\n" + "="*60)
    print("MFCC FEATURE SAMPLES")
    print("="*60)

    for emotion in feature_df['emotion'].unique():
        emotion_samples = feature_df[feature_df['emotion'] ==
emotion].head(n_samples)
        print(f"\n{emotion.upper()} Emotion Samples:")
        print("-" * 40)

        for idx, (_, row) in enumerate(emotion_samples.iterrows()):
            print(f"Sample {idx+1} (Gender: {row['gender']}, Actor:
{row['actor_id']}):")
            mfcc_features = row['features']
            print(f"  MFCC Shape: {mfcc_features.shape}")
            print(f"  First 10 coefficients: {mfcc_features[:10]}")
            print(f"  Mean: {np.mean(mfcc_features):.4f}, Std:
{np.std(mfcc_features):.4f}")
            print(f"  Min: {np.min(mfcc_features):.4f}, Max:
```

```
{np.max(mfcc_features):.4f}")
            print()

# Execute MFCC mapping and display
print("Creating MFCC outputs...")
mfcc_stats = save_mfcc_mapping(feature_df)
display_mfcc_samples(feature_df)

# Display MFCC statistics summary
print("\n" + "="*60)
print("MFCC STATISTICS BY EMOTION")
print("="*60)

for emotion, stats in mfcc_stats.items():
    print(f"\n{emotion.upper()}:")
    print(f"  Samples: {stats['count']}")
    print(f"  MFCC Mean Range: [{np.min(stats['mfcc_mean']):.4f},
{np.max(stats['mfcc_mean']):.4f}]")
    print(f"  MFCC Std Range: [{np.min(stats['mfcc_std']):.4f},
{np.max(stats['mfcc_std']):.4f}]")
    print(f"  First 5 Mean Coeffs: {stats['mfcc_mean'][:5]}")
```

```
Creating MFCC outputs...
Saving MFCC mappings...
MFCC outputs saved to 'mfcc_outputs/':
- mfcc_features.csv (all MFCC features)
- mfcc_statistics.csv (statistics by emotion)

============================================================
MFCC FEATURE SAMPLES
============================================================

NEUTRAL Emotion Samples:
----------------------------------------
Sample 1 (Gender: male, Actor: 1):
  MFCC Shape: (40,)
  First 10 coefficients: [-185.56456    86.4053     17.86474
24.316692  -35.4502     -5.204262
  -20.178936  -21.267418  -21.51963    -9.630535]
  Mean: -9.5411, Std: 33.3382
  Min: -185.5646, Max: 86.4053

Sample 2 (Gender: male, Actor: 1):
  MFCC Shape: (40,)
  First 10 coefficients: [-185.63596     86.928474    19.0367
24.553465   -34.234737
   -4.8290973  -19.652214   -20.605732   -21.832644    -9.769935 ]
  Mean: -9.4223, Std: 33.3793
  Min: -185.6360, Max: 86.9285
```

```
Sample 3 (Gender: male, Actor: 1):
  MFCC Shape: (40,)
  First 10 coefficients: [-195.72185    79.95727    18.900675
9.862856  -37.19016    -7.869899
  -25.791437  -25.114307  -15.944904  -12.101578]
  Mean: -9.4115, Std: 34.3084
  Min: -195.7218, Max: 79.9573


HAPPY Emotion Samples:
----------------------------------------
Sample 1 (Gender: male, Actor: 1):
  MFCC Shape: (40,)
  First 10 coefficients: [-156.06346    101.98871     -0.8017558
17.645277   -36.125103
  -10.113171   -21.215117   -18.954985   -29.216145    -5.1238384]
  Mean: -7.8536, Std: 30.7726
  Min: -156.0635, Max: 101.9887

Sample 2 (Gender: male, Actor: 1):
  MFCC Shape: (40,)
  First 10 coefficients: [-163.96078      95.55652      -0.29181892
13.07936     -35.72079
  -13.737822    -23.301655    -24.221306    -26.465809     -
6.874274  ]
  Mean: -7.9829, Std: 31.2413
  Min: -163.9608, Max: 95.5565

Sample 3 (Gender: male, Actor: 1):
  MFCC Shape: (40,)
  First 10 coefficients: [-168.32988    102.81209     -2.1989365
18.156324   -36.57644
  -10.765083   -22.598518   -20.229656   -30.687185    -5.1056023]
  Mean: -8.1414, Std: 32.4751
  Min: -168.3299, Max: 102.8121


SURPRISED Emotion Samples:
----------------------------------------
Sample 1 (Gender: male, Actor: 1):
  MFCC Shape: (40,)
  First 10 coefficients: [-181.66507     84.131226     3.3534398
6.0788026  -33.041767
   -6.4303784  -19.76532    -20.435219   -24.302235    -4.3640738]
  Mean: -7.9174, Std: 32.6497
  Min: -181.6651, Max: 84.1312

Sample 2 (Gender: male, Actor: 1):
  MFCC Shape: (40,)
  First 10 coefficients: [-181.50351     83.542114     3.3901145
```

```
6.244818   -32.967545
   -6.3734474  -19.758835   -20.200619   -24.291367    -4.415375 ]
  Mean: -7.9080, Std: 32.5829
  Min: -181.5035, Max: 83.5421

Sample 3 (Gender: male, Actor: 1):
  MFCC Shape: (40,)
  First 10 coefficients: [-177.84856     82.69569      2.6801858  -
10.49295    -34.00319
  -10.08428    -27.201967   -27.790997   -13.528613   -15.085135 ]
  Mean: -8.0459, Std: 32.1822
  Min: -177.8486, Max: 82.6957


DISGUSTED Emotion Samples:
----------------------------------------
Sample 1 (Gender: male, Actor: 1):
  MFCC Shape: (40,)
  First 10 coefficients: [-197.55086     85.91173      6.7082386
25.487738   -35.36378
   -9.344953   -17.799019   -16.418087   -28.946928    -5.1319156]
  Mean: -8.9252, Std: 35.1258
  Min: -197.5509, Max: 85.9117

Sample 2 (Gender: male, Actor: 1):
  MFCC Shape: (40,)
  First 10 coefficients: [-196.32515     85.20726      6.7631426
25.114285   -35.504894
   -8.823496   -17.307241   -15.219946   -28.93467     -4.8370852]
  Mean: -8.8195, Std: 34.8965
  Min: -196.3251, Max: 85.2073

Sample 3 (Gender: male, Actor: 1):
  MFCC Shape: (40,)
  First 10 coefficients: [-154.89849498   64.63605404   15.70743192
14.94511609  -18.2622295
   -8.46234307  -11.61639676  -15.72766223  -19.8957531    -
5.54956803]
  Mean: -6.1185, Std: 27.2511
  Min: -154.8985, Max: 64.6361


DISAPPOINTED Emotion Samples:
----------------------------------------
Sample 1 (Gender: male, Actor: 1):
  MFCC Shape: (40,)
  First 10 coefficients: [-195.95786     90.293       20.442495
25.692518   -32.956463
  -10.003607   -18.955706   -12.319283   -30.974562    -1.1222973]
  Mean: -9.4928, Std: 35.1644
```

```
  Min: -195.9579, Max: 90.2930

Sample 2 (Gender: male, Actor: 1):
  MFCC Shape: (40,)
  First 10 coefficients: [-166.88165     78.18361     21.279613
22.687151    -8.729934
  -10.337109   -11.4933815   -8.346552   -18.98071    -12.213678 ]
  Mean: -6.9200, Std: 29.7512
  Min: -166.8817, Max: 78.1836

Sample 3 (Gender: male, Actor: 1):
  MFCC Shape: (40,)
  First 10 coefficients: [-194.31282     89.95304     21.164362
26.443651   -32.04506
  -10.172462   -18.61726    -12.4805975  -30.196304    -1.0573113]
  Mean: -9.3325, Std: 34.9207
  Min: -194.3128, Max: 89.9530


============================================================
MFCC STATISTICS BY EMOTION
============================================================

NEUTRAL:
  Samples: 180
  MFCC Mean Range: [-183.7847, 87.3801]
  MFCC Std Range: [2.5286, 29.9326]
  First 5 Mean Coeffs: [-183.78467746   87.38014703   10.07731671
21.35604431  -26.22962454]

HAPPY:
  Samples: 180
  MFCC Mean Range: [-174.2080, 77.3357]
  MFCC Std Range: [2.8309, 26.6658]
  First 5 Mean Coeffs: [-174.20795893   77.33569242   -5.13498208
12.33752972  -29.46276147]

SURPRISED:
  Samples: 180
  MFCC Mean Range: [-166.3784, 70.8179]
  MFCC Std Range: [2.7033, 24.5647]
  First 5 Mean Coeffs: [-166.37835874   70.81794043   -8.78395847
4.72109242  -30.62239782]

DISGUSTED:
  Samples: 180
  MFCC Mean Range: [-218.1665, 76.4158]
  MFCC Std Range: [2.5917, 36.1406]
  First 5 Mean Coeffs: [-218.16646679   76.41577611    5.48720211
21.31633254  -25.52904955]
```

```
DISAPPOINTED:
  Samples: 180
  MFCC Mean Range: [-222.3169, 81.3142]
  MFCC Std Range: [3.0294, 37.6206]
  First 5 Mean Coeffs: [-222.31692318   81.31417978   13.08068908
20.09928601  -24.22950752]
```

CELL 6: Data Preparation for Deep Learning

```
# Encode labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(feature_df['emotion'])

# Prepare features
X = np.array(feature_df['features'].tolist())
y = y_encoded

# Create gender binary encoding
gender_encoder = LabelEncoder()
gender_encoded = gender_encoder.fit_transform(feature_df['gender'])

print(f"Feature matrix shape: {X.shape}")
print(f"Number of classes: {len(label_encoder.classes_)}")
print(f"Classes: {label_encoder.classes_}")

# Stratified train-test split to maintain class distribution
sss = StratifiedShuffleSplit(n_splits=1, test_size=0.2,
random_state=42)
train_idx, test_idx = next(sss.split(X, y))

X_train, X_test = X[train_idx], X[test_idx]
y_train, y_test = y[train_idx], y[test_idx]
gender_train = gender_encoded[train_idx]
gender_test = gender_encoded[test_idx]

# Reshape for LSTM input (samples, timesteps, features)
X_train = np.expand_dims(X_train, axis=2)
X_test = np.expand_dims(X_test, axis=2)

print(f"Training set shape: {X_train.shape}")
print(f"Test set shape: {X_test.shape}")

# Save class names for later use
os.makedirs('models_standard_aug', exist_ok=True)
class_names_df = pd.DataFrame({
    'class_name': label_encoder.classes_,
    'class_code': range(len(label_encoder.classes_))
})
```

```
class_names_df.to_csv('models_standard_aug/class_names.csv',
index=False)

Feature matrix shape: (900, 40)
Number of classes: 5
Classes: ['disappointed' 'disgusted' 'happy' 'neutral' 'surprised']
Training set shape: (720, 40, 1)
Test set shape: (180, 40, 1)
```

CELL 7: BiLSTM Model Architecture

```
def create_bilstm_model(input_shape, num_classes, dropout_rate=0.3):
    """Create BiLSTM model for emotion recognition"""

    model = Sequential([
        # First BiLSTM layer
        Bidirectional(LSTM(128, return_sequences=True,
                           dropout=dropout_rate,
recurrent_dropout=dropout_rate),
                      input_shape=input_shape),
        BatchNormalization(),

        # Second BiLSTM layer
        Bidirectional(LSTM(64, dropout=dropout_rate,
recurrent_dropout=dropout_rate)),
        BatchNormalization(),

        # Dense layers
        Dense(128, activation='relu', kernel_regularizer=l2(0.01)),
        Dropout(dropout_rate),
        BatchNormalization(),

        Dense(64, activation='relu', kernel_regularizer=l2(0.01)),
        Dropout(dropout_rate),
        BatchNormalization(),

        # Output layer
        Dense(num_classes, activation='softmax')
    ])

    return model

# Create model
input_shape = (X_train.shape[1], X_train.shape[2])  # (40, 1)
num_classes = len(label_encoder.classes_)

model = create_bilstm_model(input_shape, num_classes)

# Compile model
model.compile(
```

```
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

model.summary()
```

```
WARNING:tensorflow:From c:\Users\LENOVO\AppData\Local\Programs\Python\
Python310\lib\site-packages\keras\src\layers\rnn\lstm.py:148: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.


WARNING:tensorflow:From c:\Users\LENOVO\AppData\Local\Programs\Python\
Python310\lib\site-packages\keras\src\layers\rnn\lstm.py:148: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.


Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional (Bidirection  (None, 40, 256)           133120
 al)

 batch_normalization (Batch  (None, 40, 256)           1024
 Normalization)

 bidirectional_1 (Bidirecti  (None, 128)               164352
 onal)

 batch_normalization_1 (Bat  (None, 128)               512
 chNormalization)

 dense (Dense)               (None, 128)               16512

 dropout (Dropout)           (None, 128)               0

 batch_normalization_2 (Bat  (None, 128)               512
 chNormalization)

 dense_1 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 batch_normalization_3 (Bat  (None, 64)                256
 chNormalization)

 dense_2 (Dense)             (None, 5)                 325
```

```
=================================================================
Total params: 324869 (1.24 MB)
Trainable params: 323717 (1.23 MB)
Non-trainable params: 1152 (4.50 KB)
_________________________________________________________________
```

CELL 8: Model Training

```
# Callbacks
callbacks = [
    EarlyStopping(
        monitor='val_loss',
        patience=20,
        restore_best_weights=True,
        verbose=1
    ),
    ModelCheckpoint(
        filepath='models_standard_aug/best_model.keras',
        monitor='val_accuracy',
        save_best_only=True,
        verbose=1
    ),
    ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=10,
        min_lr=0.0001,
        verbose=1
    )
]

print("Starting model training...")

# Train model
history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=150,
    batch_size=32,
    callbacks=callbacks,
    verbose=1
)

print("Training completed!")
```

```
Starting model training...
Epoch 1/150
23/23 [==============================] - ETA: 0s - loss: 3.7988 -
accuracy: 0.2972
```

```
Epoch 1: val_accuracy improved from -inf to 0.30556, saving model to
models_standard_aug\best_model.keras
23/23 [==============================] - 2s 86ms/step - loss: 3.7988 -
accuracy: 0.2972 - val_loss: 3.5045 - val_accuracy: 0.3056 - lr:
0.0010
Epoch 2/150
23/23 [==============================] - ETA: 0s - loss: 3.6800 -
accuracy: 0.2931
Epoch 2: val_accuracy improved from 0.30556 to 0.31667, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 84ms/step - loss: 3.6800 -
accuracy: 0.2931 - val_loss: 3.3822 - val_accuracy: 0.3167 - lr:
0.0010
Epoch 3/150
23/23 [==============================] - ETA: 0s - loss: 3.5492 -
accuracy: 0.3167
Epoch 3: val_accuracy improved from 0.31667 to 0.32778, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 84ms/step - loss: 3.5492 -
accuracy: 0.3167 - val_loss: 3.2632 - val_accuracy: 0.3278 - lr:
0.0010
Epoch 4/150
23/23 [==============================] - ETA: 0s - loss: 3.4211 -
accuracy: 0.3292
Epoch 4: val_accuracy improved from 0.32778 to 0.36111, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 89ms/step - loss: 3.4211 -
accuracy: 0.3292 - val_loss: 3.1549 - val_accuracy: 0.3611 - lr:
0.0010
Epoch 5/150
23/23 [==============================] - ETA: 0s - loss: 3.2759 -
accuracy: 0.3569
Epoch 5: val_accuracy improved from 0.36111 to 0.41111, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 86ms/step - loss: 3.2759 -
accuracy: 0.3569 - val_loss: 3.0194 - val_accuracy: 0.4111 - lr:
0.0010
Epoch 6/150
23/23 [==============================] - ETA: 0s - loss: 3.1500 -
accuracy: 0.4000
Epoch 6: val_accuracy did not improve from 0.41111
23/23 [==============================] - 2s 79ms/step - loss: 3.1500 -
accuracy: 0.4000 - val_loss: 2.9316 - val_accuracy: 0.3556 - lr:
0.0010
Epoch 7/150
23/23 [==============================] - ETA: 0s - loss: 3.0253 -
accuracy: 0.3986
Epoch 7: val_accuracy improved from 0.41111 to 0.47778, saving model
to models_standard_aug\best_model.keras
```

```
23/23 [==============================] - 2s 79ms/step - loss: 3.0253 -
accuracy: 0.3986 - val_loss: 2.7755 - val_accuracy: 0.4778 - lr:
0.0010
Epoch 8/150
23/23 [==============================] - ETA: 0s - loss: 2.8991 -
accuracy: 0.4042
Epoch 8: val_accuracy improved from 0.47778 to 0.48889, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 79ms/step - loss: 2.8991 -
accuracy: 0.4042 - val_loss: 2.6819 - val_accuracy: 0.4889 - lr:
0.0010
Epoch 9/150
23/23 [==============================] - ETA: 0s - loss: 2.7960 -
accuracy: 0.4361
Epoch 9: val_accuracy improved from 0.48889 to 0.51111, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 82ms/step - loss: 2.7960 -
accuracy: 0.4361 - val_loss: 2.5874 - val_accuracy: 0.5111 - lr:
0.0010
Epoch 10/150
23/23 [==============================] - ETA: 0s - loss: 2.7361 -
accuracy: 0.4083
Epoch 10: val_accuracy did not improve from 0.51111
23/23 [==============================] - 2s 88ms/step - loss: 2.7361 -
accuracy: 0.4083 - val_loss: 2.4680 - val_accuracy: 0.4944 - lr:
0.0010
Epoch 11/150
23/23 [==============================] - ETA: 0s - loss: 2.6631 -
accuracy: 0.4056
Epoch 11: val_accuracy did not improve from 0.51111
23/23 [==============================] - 2s 82ms/step - loss: 2.6631 -
accuracy: 0.4056 - val_loss: 2.3523 - val_accuracy: 0.5000 - lr:
0.0010
Epoch 12/150
23/23 [==============================] - ETA: 0s - loss: 2.5606 -
accuracy: 0.4583
Epoch 12: val_accuracy improved from 0.51111 to 0.56667, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 83ms/step - loss: 2.5606 -
accuracy: 0.4583 - val_loss: 2.3511 - val_accuracy: 0.5667 - lr:
0.0010
Epoch 13/150
23/23 [==============================] - ETA: 0s - loss: 2.4648 -
accuracy: 0.4569
Epoch 13: val_accuracy did not improve from 0.56667
23/23 [==============================] - 2s 84ms/step - loss: 2.4648 -
accuracy: 0.4569 - val_loss: 2.2350 - val_accuracy: 0.5222 - lr:
0.0010
Epoch 14/150
```

```
23/23 [==============================] - ETA: 0s - loss: 2.3831 -
accuracy: 0.4847
Epoch 14: val_accuracy did not improve from 0.56667
23/23 [==============================] - 2s 78ms/step - loss: 2.3831 -
accuracy: 0.4847 - val_loss: 2.1935 - val_accuracy: 0.5167 - lr:
0.0010
Epoch 15/150
23/23 [==============================] - ETA: 0s - loss: 2.3215 -
accuracy: 0.4917
Epoch 15: val_accuracy improved from 0.56667 to 0.60556, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 86ms/step - loss: 2.3215 -
accuracy: 0.4917 - val_loss: 2.0380 - val_accuracy: 0.6056 - lr:
0.0010
Epoch 16/150
23/23 [==============================] - ETA: 0s - loss: 2.2882 -
accuracy: 0.4486
Epoch 16: val_accuracy improved from 0.60556 to 0.61667, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 87ms/step - loss: 2.2882 -
accuracy: 0.4486 - val_loss: 1.9526 - val_accuracy: 0.6167 - lr:
0.0010
Epoch 17/150
23/23 [==============================] - ETA: 0s - loss: 2.1452 -
accuracy: 0.5069
Epoch 17: val_accuracy did not improve from 0.61667
23/23 [==============================] - 2s 82ms/step - loss: 2.1452 -
accuracy: 0.5069 - val_loss: 1.8717 - val_accuracy: 0.6167 - lr:
0.0010
Epoch 18/150
23/23 [==============================] - ETA: 0s - loss: 2.0775 -
accuracy: 0.5278
Epoch 18: val_accuracy improved from 0.61667 to 0.62222, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 89ms/step - loss: 2.0775 -
accuracy: 0.5278 - val_loss: 1.8836 - val_accuracy: 0.6222 - lr:
0.0010
Epoch 19/150
23/23 [==============================] - ETA: 0s - loss: 1.9996 -
accuracy: 0.5319
Epoch 19: val_accuracy improved from 0.62222 to 0.66667, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 88ms/step - loss: 1.9996 -
accuracy: 0.5319 - val_loss: 1.7845 - val_accuracy: 0.6667 - lr:
0.0010
Epoch 20/150
23/23 [==============================] - ETA: 0s - loss: 1.9733 -
accuracy: 0.5403
Epoch 20: val_accuracy did not improve from 0.66667
```

```
23/23 [==============================] - 2s 86ms/step - loss: 1.9733 -
accuracy: 0.5403 - val_loss: 1.7218 - val_accuracy: 0.6500 - lr:
0.0010
Epoch 21/150
23/23 [==============================] - ETA: 0s - loss: 1.8893 -
accuracy: 0.5458
Epoch 21: val_accuracy did not improve from 0.66667
23/23 [==============================] - 2s 80ms/step - loss: 1.8893 -
accuracy: 0.5458 - val_loss: 1.7214 - val_accuracy: 0.6222 - lr:
0.0010
Epoch 22/150
23/23 [==============================] - ETA: 0s - loss: 1.8617 -
accuracy: 0.5444
Epoch 22: val_accuracy did not improve from 0.66667
23/23 [==============================] - 2s 81ms/step - loss: 1.8617 -
accuracy: 0.5444 - val_loss: 1.6205 - val_accuracy: 0.5833 - lr:
0.0010
Epoch 23/150
23/23 [==============================] - ETA: 0s - loss: 1.8048 -
accuracy: 0.5875
Epoch 23: val_accuracy did not improve from 0.66667
23/23 [==============================] - 2s 77ms/step - loss: 1.8048 -
accuracy: 0.5875 - val_loss: 1.5635 - val_accuracy: 0.6500 - lr:
0.0010
Epoch 24/150
23/23 [==============================] - ETA: 0s - loss: 1.7501 -
accuracy: 0.5875
Epoch 24: val_accuracy did not improve from 0.66667
23/23 [==============================] - 2s 73ms/step - loss: 1.7501 -
accuracy: 0.5875 - val_loss: 1.5545 - val_accuracy: 0.6556 - lr:
0.0010
Epoch 25/150
23/23 [==============================] - ETA: 0s - loss: 1.7210 -
accuracy: 0.5486
Epoch 25: val_accuracy improved from 0.66667 to 0.67222, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 81ms/step - loss: 1.7210 -
accuracy: 0.5486 - val_loss: 1.4554 - val_accuracy: 0.6722 - lr:
0.0010
Epoch 26/150
23/23 [==============================] - ETA: 0s - loss: 1.6318 -
accuracy: 0.5833
Epoch 26: val_accuracy did not improve from 0.67222
23/23 [==============================] - 2s 80ms/step - loss: 1.6318 -
accuracy: 0.5833 - val_loss: 1.5266 - val_accuracy: 0.5667 - lr:
0.0010
Epoch 27/150
23/23 [==============================] - ETA: 0s - loss: 1.6217 -
accuracy: 0.6181
```

```
Epoch 27: val_accuracy improved from 0.67222 to 0.67778, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 78ms/step - loss: 1.6217 -
accuracy: 0.6181 - val_loss: 1.4076 - val_accuracy: 0.6778 - lr:
0.0010
Epoch 28/150
23/23 [==============================] - ETA: 0s - loss: 1.5886 -
accuracy: 0.5764
Epoch 28: val_accuracy did not improve from 0.67778
23/23 [==============================] - 2s 71ms/step - loss: 1.5886 -
accuracy: 0.5764 - val_loss: 1.3594 - val_accuracy: 0.6611 - lr:
0.0010
Epoch 29/150
23/23 [==============================] - ETA: 0s - loss: 1.5274 -
accuracy: 0.5972
Epoch 29: val_accuracy did not improve from 0.67778
23/23 [==============================] - 2s 68ms/step - loss: 1.5274 -
accuracy: 0.5972 - val_loss: 1.3247 - val_accuracy: 0.6778 - lr:
0.0010
Epoch 30/150
23/23 [==============================] - ETA: 0s - loss: 1.4686 -
accuracy: 0.5972
Epoch 30: val_accuracy improved from 0.67778 to 0.69444, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 75ms/step - loss: 1.4686 -
accuracy: 0.5972 - val_loss: 1.3009 - val_accuracy: 0.6944 - lr:
0.0010
Epoch 31/150
23/23 [==============================] - ETA: 0s - loss: 1.4921 -
accuracy: 0.6111
Epoch 31: val_accuracy did not improve from 0.69444
23/23 [==============================] - 2s 80ms/step - loss: 1.4921 -
accuracy: 0.6111 - val_loss: 1.2887 - val_accuracy: 0.6611 - lr:
0.0010
Epoch 32/150
23/23 [==============================] - ETA: 0s - loss: 1.4125 -
accuracy: 0.6347
Epoch 32: val_accuracy improved from 0.69444 to 0.72222, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 79ms/step - loss: 1.4125 -
accuracy: 0.6347 - val_loss: 1.1732 - val_accuracy: 0.7222 - lr:
0.0010
Epoch 33/150
23/23 [==============================] - ETA: 0s - loss: 1.4060 -
accuracy: 0.6069
Epoch 33: val_accuracy did not improve from 0.72222
23/23 [==============================] - 2s 77ms/step - loss: 1.4060 -
accuracy: 0.6069 - val_loss: 1.3467 - val_accuracy: 0.6222 - lr:
0.0010
```

```
Epoch 34/150
23/23 [==============================] - ETA: 0s - loss: 1.3282 -
accuracy: 0.6500
Epoch 34: val_accuracy did not improve from 0.72222
23/23 [==============================] - 2s 74ms/step - loss: 1.3282 -
accuracy: 0.6500 - val_loss: 1.1807 - val_accuracy: 0.6556 - lr:
0.0010
Epoch 35/150
22/23 [===========================>..] - ETA: 0s - loss: 1.3052 -
accuracy: 0.6364
Epoch 35: val_accuracy did not improve from 0.72222
23/23 [==============================] - 2s 74ms/step - loss: 1.3030 -
accuracy: 0.6375 - val_loss: 1.1558 - val_accuracy: 0.6444 - lr:
0.0010
Epoch 36/150
23/23 [==============================] - ETA: 0s - loss: 1.2928 -
accuracy: 0.6361
Epoch 36: val_accuracy did not improve from 0.72222
23/23 [==============================] - 2s 80ms/step - loss: 1.2928 -
accuracy: 0.6361 - val_loss: 1.2323 - val_accuracy: 0.6556 - lr:
0.0010
Epoch 37/150
23/23 [==============================] - ETA: 0s - loss: 1.3010 -
accuracy: 0.6403
Epoch 37: val_accuracy did not improve from 0.72222
23/23 [==============================] - 2s 80ms/step - loss: 1.3010 -
accuracy: 0.6403 - val_loss: 1.1718 - val_accuracy: 0.6667 - lr:
0.0010
Epoch 38/150
23/23 [==============================] - ETA: 0s - loss: 1.2322 -
accuracy: 0.6375
Epoch 38: val_accuracy did not improve from 0.72222
23/23 [==============================] - 2s 79ms/step - loss: 1.2322 -
accuracy: 0.6375 - val_loss: 1.0795 - val_accuracy: 0.7056 - lr:
0.0010
Epoch 39/150
23/23 [==============================] - ETA: 0s - loss: 1.1850 -
accuracy: 0.6514
Epoch 39: val_accuracy did not improve from 0.72222
23/23 [==============================] - 2s 79ms/step - loss: 1.1850 -
accuracy: 0.6514 - val_loss: 1.1150 - val_accuracy: 0.6500 - lr:
0.0010
Epoch 40/150
23/23 [==============================] - ETA: 0s - loss: 1.2190 -
accuracy: 0.6208
Epoch 40: val_accuracy did not improve from 0.72222
23/23 [==============================] - 2s 80ms/step - loss: 1.2190 -
accuracy: 0.6208 - val_loss: 1.0724 - val_accuracy: 0.6722 - lr:
0.0010
```

```
Epoch 41/150
23/23 [==============================] - ETA: 0s - loss: 1.1551 -
accuracy: 0.6639
Epoch 41: val_accuracy improved from 0.72222 to 0.72778, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 76ms/step - loss: 1.1551 -
accuracy: 0.6639 - val_loss: 1.0368 - val_accuracy: 0.7278 - lr:
0.0010
Epoch 42/150
23/23 [==============================] - ETA: 0s - loss: 1.1545 -
accuracy: 0.6639
Epoch 42: val_accuracy improved from 0.72778 to 0.75000, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 75ms/step - loss: 1.1545 -
accuracy: 0.6639 - val_loss: 0.9657 - val_accuracy: 0.7500 - lr:
0.0010
Epoch 43/150
23/23 [==============================] - ETA: 0s - loss: 1.0937 -
accuracy: 0.6958
Epoch 43: val_accuracy did not improve from 0.75000
23/23 [==============================] - 2s 75ms/step - loss: 1.0937 -
accuracy: 0.6958 - val_loss: 0.9786 - val_accuracy: 0.7111 - lr:
0.0010
Epoch 44/150
23/23 [==============================] - ETA: 0s - loss: 1.1094 -
accuracy: 0.6694
Epoch 44: val_accuracy did not improve from 0.75000
23/23 [==============================] - 2s 74ms/step - loss: 1.1094 -
accuracy: 0.6694 - val_loss: 0.9395 - val_accuracy: 0.7389 - lr:
0.0010
Epoch 45/150
22/23 [===========================>..] - ETA: 0s - loss: 1.0695 -
accuracy: 0.6690
Epoch 45: val_accuracy improved from 0.75000 to 0.76111, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 78ms/step - loss: 1.0810 -
accuracy: 0.6694 - val_loss: 0.9046 - val_accuracy: 0.7611 - lr:
0.0010
Epoch 46/150
23/23 [==============================] - ETA: 0s - loss: 1.0463 -
accuracy: 0.6722
Epoch 46: val_accuracy did not improve from 0.76111
23/23 [==============================] - 2s 77ms/step - loss: 1.0463 -
accuracy: 0.6722 - val_loss: 0.9725 - val_accuracy: 0.6611 - lr:
0.0010
Epoch 47/150
23/23 [==============================] - ETA: 0s - loss: 1.0809 -
accuracy: 0.6611
Epoch 47: val_accuracy did not improve from 0.76111
```

```
23/23 [==============================] - 2s 75ms/step - loss: 1.0809 -
accuracy: 0.6611 - val_loss: 1.0012 - val_accuracy: 0.6833 - lr:
0.0010
Epoch 48/150
23/23 [==============================] - ETA: 0s - loss: 1.0107 -
accuracy: 0.7056
Epoch 48: val_accuracy did not improve from 0.76111
23/23 [==============================] - 2s 75ms/step - loss: 1.0107 -
accuracy: 0.7056 - val_loss: 0.8338 - val_accuracy: 0.7500 - lr:
0.0010
Epoch 49/150
23/23 [==============================] - ETA: 0s - loss: 0.9780 -
accuracy: 0.6597
Epoch 49: val_accuracy did not improve from 0.76111
23/23 [==============================] - 2s 73ms/step - loss: 0.9780 -
accuracy: 0.6597 - val_loss: 0.8627 - val_accuracy: 0.7111 - lr:
0.0010
Epoch 50/150
23/23 [==============================] - ETA: 0s - loss: 0.9410 -
accuracy: 0.7014
Epoch 50: val_accuracy improved from 0.76111 to 0.76667, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 75ms/step - loss: 0.9410 -
accuracy: 0.7014 - val_loss: 0.8017 - val_accuracy: 0.7667 - lr:
0.0010
Epoch 51/150
23/23 [==============================] - ETA: 0s - loss: 0.9805 -
accuracy: 0.6764
Epoch 51: val_accuracy did not improve from 0.76667
23/23 [==============================] - 2s 72ms/step - loss: 0.9805 -
accuracy: 0.6764 - val_loss: 0.8610 - val_accuracy: 0.7167 - lr:
0.0010
Epoch 52/150
23/23 [==============================] - ETA: 0s - loss: 0.9410 -
accuracy: 0.7000
Epoch 52: val_accuracy did not improve from 0.76667
23/23 [==============================] - 2s 77ms/step - loss: 0.9410 -
accuracy: 0.7000 - val_loss: 0.7743 - val_accuracy: 0.7667 - lr:
0.0010
Epoch 53/150
23/23 [==============================] - ETA: 0s - loss: 0.9318 -
accuracy: 0.7250
Epoch 53: val_accuracy did not improve from 0.76667
23/23 [==============================] - 2s 78ms/step - loss: 0.9318 -
accuracy: 0.7250 - val_loss: 0.8503 - val_accuracy: 0.7667 - lr:
0.0010
Epoch 54/150
23/23 [==============================] - ETA: 0s - loss: 0.9278 -
accuracy: 0.7278
```

```
Epoch 54: val_accuracy improved from 0.76667 to 0.78333, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 97ms/step - loss: 0.9278 -
accuracy: 0.7278 - val_loss: 0.7814 - val_accuracy: 0.7833 - lr:
0.0010
Epoch 55/150
23/23 [==============================] - ETA: 0s - loss: 0.9547 -
accuracy: 0.7097
Epoch 55: val_accuracy did not improve from 0.78333
23/23 [==============================] - 2s 89ms/step - loss: 0.9547 -
accuracy: 0.7097 - val_loss: 0.7999 - val_accuracy: 0.7778 - lr:
0.0010
Epoch 56/150
23/23 [==============================] - ETA: 0s - loss: 0.9398 -
accuracy: 0.7125
Epoch 56: val_accuracy did not improve from 0.78333
23/23 [==============================] - 2s 82ms/step - loss: 0.9398 -
accuracy: 0.7125 - val_loss: 0.8321 - val_accuracy: 0.7444 - lr:
0.0010
Epoch 57/150
23/23 [==============================] - ETA: 0s - loss: 0.8941 -
accuracy: 0.7069
Epoch 57: val_accuracy did not improve from 0.78333
23/23 [==============================] - 2s 81ms/step - loss: 0.8941 -
accuracy: 0.7069 - val_loss: 0.8039 - val_accuracy: 0.7611 - lr:
0.0010
Epoch 58/150
23/23 [==============================] - ETA: 0s - loss: 0.8970 -
accuracy: 0.7250
Epoch 58: val_accuracy did not improve from 0.78333
23/23 [==============================] - 2s 81ms/step - loss: 0.8970 -
accuracy: 0.7250 - val_loss: 0.8646 - val_accuracy: 0.7167 - lr:
0.0010
Epoch 59/150
23/23 [==============================] - ETA: 0s - loss: 0.8823 -
accuracy: 0.7222
Epoch 59: val_accuracy did not improve from 0.78333
23/23 [==============================] - 2s 80ms/step - loss: 0.8823 -
accuracy: 0.7222 - val_loss: 0.8503 - val_accuracy: 0.7056 - lr:
0.0010
Epoch 60/150
23/23 [==============================] - ETA: 0s - loss: 0.9255 -
accuracy: 0.7153
Epoch 60: val_accuracy did not improve from 0.78333
23/23 [==============================] - 2s 80ms/step - loss: 0.9255 -
accuracy: 0.7153 - val_loss: 0.7909 - val_accuracy: 0.7778 - lr:
0.0010
Epoch 61/150
23/23 [==============================] - ETA: 0s - loss: 0.9016 -
```

```
accuracy: 0.7153
Epoch 61: val_accuracy improved from 0.78333 to 0.79444, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 82ms/step - loss: 0.9016 -
accuracy: 0.7153 - val_loss: 0.7908 - val_accuracy: 0.7944 - lr:
0.0010
Epoch 62/150
23/23 [==============================] - ETA: 0s - loss: 0.8494 -
accuracy: 0.7347
Epoch 62: val_accuracy improved from 0.79444 to 0.80000, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 82ms/step - loss: 0.8494 -
accuracy: 0.7347 - val_loss: 0.7175 - val_accuracy: 0.8000 - lr:
0.0010
Epoch 63/150
23/23 [==============================] - ETA: 0s - loss: 0.8342 -
accuracy: 0.7333
Epoch 63: val_accuracy improved from 0.80000 to 0.81667, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 83ms/step - loss: 0.8342 -
accuracy: 0.7333 - val_loss: 0.6731 - val_accuracy: 0.8167 - lr:
0.0010
Epoch 64/150
23/23 [==============================] - ETA: 0s - loss: 0.8547 -
accuracy: 0.7458
Epoch 64: val_accuracy did not improve from 0.81667
23/23 [==============================] - 2s 75ms/step - loss: 0.8547 -
accuracy: 0.7458 - val_loss: 0.6722 - val_accuracy: 0.8111 - lr:
0.0010
Epoch 65/150
23/23 [==============================] - ETA: 0s - loss: 0.7728 -
accuracy: 0.7736
Epoch 65: val_accuracy did not improve from 0.81667
23/23 [==============================] - 2s 76ms/step - loss: 0.7728 -
accuracy: 0.7736 - val_loss: 0.6683 - val_accuracy: 0.7889 - lr:
0.0010
Epoch 66/150
23/23 [==============================] - ETA: 0s - loss: 0.8012 -
accuracy: 0.7556
Epoch 66: val_accuracy did not improve from 0.81667
23/23 [==============================] - 2s 76ms/step - loss: 0.8012 -
accuracy: 0.7556 - val_loss: 0.6842 - val_accuracy: 0.7778 - lr:
0.0010
Epoch 67/150
23/23 [==============================] - ETA: 0s - loss: 0.7792 -
accuracy: 0.7708
Epoch 67: val_accuracy did not improve from 0.81667
23/23 [==============================] - 2s 77ms/step - loss: 0.7792 -
accuracy: 0.7708 - val_loss: 0.6713 - val_accuracy: 0.8111 - lr:
0.0010
```

```
Epoch 68/150
23/23 [==============================] - ETA: 0s - loss: 0.7915 -
accuracy: 0.7500
Epoch 68: val_accuracy improved from 0.81667 to 0.82778, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 77ms/step - loss: 0.7915 -
accuracy: 0.7500 - val_loss: 0.6019 - val_accuracy: 0.8278 - lr:
0.0010
Epoch 69/150
23/23 [==============================] - ETA: 0s - loss: 0.7518 -
accuracy: 0.7806
Epoch 69: val_accuracy did not improve from 0.82778
23/23 [==============================] - 2s 76ms/step - loss: 0.7518 -
accuracy: 0.7806 - val_loss: 0.6533 - val_accuracy: 0.8278 - lr:
0.0010
Epoch 70/150
23/23 [==============================] - ETA: 0s - loss: 0.7191 -
accuracy: 0.7722
Epoch 70: val_accuracy did not improve from 0.82778
23/23 [==============================] - 2s 76ms/step - loss: 0.7191 -
accuracy: 0.7722 - val_loss: 0.7414 - val_accuracy: 0.7833 - lr:
0.0010
Epoch 71/150
23/23 [==============================] - ETA: 0s - loss: 0.6871 -
accuracy: 0.7972
Epoch 71: val_accuracy improved from 0.82778 to 0.83333, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 74ms/step - loss: 0.6871 -
accuracy: 0.7972 - val_loss: 0.5874 - val_accuracy: 0.8333 - lr:
0.0010
Epoch 72/150
23/23 [==============================] - ETA: 0s - loss: 0.8067 -
accuracy: 0.7417
Epoch 72: val_accuracy did not improve from 0.83333
23/23 [==============================] - 2s 71ms/step - loss: 0.8067 -
accuracy: 0.7417 - val_loss: 0.6968 - val_accuracy: 0.7722 - lr:
0.0010
Epoch 73/150
23/23 [==============================] - ETA: 0s - loss: 0.7251 -
accuracy: 0.7903
Epoch 73: val_accuracy did not improve from 0.83333
23/23 [==============================] - 2s 72ms/step - loss: 0.7251 -
accuracy: 0.7903 - val_loss: 0.6170 - val_accuracy: 0.8167 - lr:
0.0010
Epoch 74/150
23/23 [==============================] - ETA: 0s - loss: 0.7351 -
accuracy: 0.7556
Epoch 74: val_accuracy did not improve from 0.83333
23/23 [==============================] - 2s 73ms/step - loss: 0.7351 -
```

```
accuracy: 0.7556 - val_loss: 0.6273 - val_accuracy: 0.8056 - lr:
0.0010
Epoch 75/150
23/23 [==============================] - ETA: 0s - loss: 0.7471 -
accuracy: 0.7736
Epoch 75: val_accuracy did not improve from 0.83333
23/23 [==============================] - 2s 75ms/step - loss: 0.7471 -
accuracy: 0.7736 - val_loss: 0.6365 - val_accuracy: 0.7833 - lr:
0.0010
Epoch 76/150
23/23 [==============================] - ETA: 0s - loss: 0.6809 -
accuracy: 0.7875
Epoch 76: val_accuracy did not improve from 0.83333
23/23 [==============================] - 2s 75ms/step - loss: 0.6809 -
accuracy: 0.7875 - val_loss: 0.6297 - val_accuracy: 0.8111 - lr:
0.0010
Epoch 77/150
23/23 [==============================] - ETA: 0s - loss: 0.7128 -
accuracy: 0.7917
Epoch 77: val_accuracy did not improve from 0.83333
23/23 [==============================] - 2s 76ms/step - loss: 0.7128 -
accuracy: 0.7917 - val_loss: 0.6775 - val_accuracy: 0.7889 - lr:
0.0010
Epoch 78/150
23/23 [==============================] - ETA: 0s - loss: 0.7012 -
accuracy: 0.8000
Epoch 78: val_accuracy improved from 0.83333 to 0.86111, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 80ms/step - loss: 0.7012 -
accuracy: 0.8000 - val_loss: 0.5198 - val_accuracy: 0.8611 - lr:
0.0010
Epoch 79/150
23/23 [==============================] - ETA: 0s - loss: 0.6503 -
accuracy: 0.8111
Epoch 79: val_accuracy did not improve from 0.86111
23/23 [==============================] - 2s 80ms/step - loss: 0.6503 -
accuracy: 0.8111 - val_loss: 0.7025 - val_accuracy: 0.7667 - lr:
0.0010
Epoch 80/150
23/23 [==============================] - ETA: 0s - loss: 0.7236 -
accuracy: 0.7806
Epoch 80: val_accuracy improved from 0.86111 to 0.87222, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 79ms/step - loss: 0.7236 -
accuracy: 0.7806 - val_loss: 0.5326 - val_accuracy: 0.8722 - lr:
0.0010
Epoch 81/150
23/23 [==============================] - ETA: 0s - loss: 0.6215 -
accuracy: 0.8139
```

```
Epoch 81: val_accuracy did not improve from 0.87222
23/23 [==============================] - 2s 70ms/step - loss: 0.6215 -
accuracy: 0.8139 - val_loss: 0.5558 - val_accuracy: 0.8611 - lr:
0.0010
Epoch 82/150
23/23 [==============================] - ETA: 0s - loss: 0.6774 -
accuracy: 0.7931
Epoch 82: val_accuracy did not improve from 0.87222
23/23 [==============================] - 2s 78ms/step - loss: 0.6774 -
accuracy: 0.7931 - val_loss: 0.5716 - val_accuracy: 0.8222 - lr:
0.0010
Epoch 83/150
23/23 [==============================] - ETA: 0s - loss: 0.7240 -
accuracy: 0.7917
Epoch 83: val_accuracy did not improve from 0.87222
23/23 [==============================] - 2s 75ms/step - loss: 0.7240 -
accuracy: 0.7917 - val_loss: 0.7286 - val_accuracy: 0.8222 - lr:
0.0010
Epoch 84/150
23/23 [==============================] - ETA: 0s - loss: 0.6537 -
accuracy: 0.8042
Epoch 84: val_accuracy did not improve from 0.87222
23/23 [==============================] - 2s 75ms/step - loss: 0.6537 -
accuracy: 0.8042 - val_loss: 0.6819 - val_accuracy: 0.8000 - lr:
0.0010
Epoch 85/150
23/23 [==============================] - ETA: 0s - loss: 0.6777 -
accuracy: 0.7931
Epoch 85: val_accuracy did not improve from 0.87222
23/23 [==============================] - 2s 74ms/step - loss: 0.6777 -
accuracy: 0.7931 - val_loss: 0.6444 - val_accuracy: 0.8111 - lr:
0.0010
Epoch 86/150
23/23 [==============================] - ETA: 0s - loss: 0.5817 -
accuracy: 0.8389
Epoch 86: val_accuracy did not improve from 0.87222
23/23 [==============================] - 2s 74ms/step - loss: 0.5817 -
accuracy: 0.8389 - val_loss: 0.7116 - val_accuracy: 0.7722 - lr:
0.0010
Epoch 87/150
23/23 [==============================] - ETA: 0s - loss: 0.5939 -
accuracy: 0.8264
Epoch 87: val_accuracy did not improve from 0.87222
23/23 [==============================] - 2s 76ms/step - loss: 0.5939 -
accuracy: 0.8264 - val_loss: 0.6222 - val_accuracy: 0.8556 - lr:
0.0010
Epoch 88/150
22/23 [===========================>..] - ETA: 0s - loss: 0.6141 -
accuracy: 0.8310
```

```
Epoch 88: val_accuracy did not improve from 0.87222

Epoch 88: ReduceLROnPlateau reducing learning rate to
0.0005000000237487257.
23/23 [==============================] - 2s 72ms/step - loss: 0.6076 -
accuracy: 0.8347 - val_loss: 0.5770 - val_accuracy: 0.8278 - lr:
0.0010
Epoch 89/150
23/23 [==============================] - ETA: 0s - loss: 0.5643 -
accuracy: 0.8333
Epoch 89: val_accuracy improved from 0.87222 to 0.90556, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 77ms/step - loss: 0.5643 -
accuracy: 0.8333 - val_loss: 0.4936 - val_accuracy: 0.9056 - lr:
5.0000e-04
Epoch 90/150
23/23 [==============================] - ETA: 0s - loss: 0.5471 -
accuracy: 0.8500
Epoch 90: val_accuracy did not improve from 0.90556
23/23 [==============================] - 2s 74ms/step - loss: 0.5471 -
accuracy: 0.8500 - val_loss: 0.5963 - val_accuracy: 0.8389 - lr:
5.0000e-04
Epoch 91/150
23/23 [==============================] - ETA: 0s - loss: 0.5191 -
accuracy: 0.8486
Epoch 91: val_accuracy did not improve from 0.90556
23/23 [==============================] - 2s 74ms/step - loss: 0.5191 -
accuracy: 0.8486 - val_loss: 0.5039 - val_accuracy: 0.8611 - lr:
5.0000e-04
Epoch 92/150
23/23 [==============================] - ETA: 0s - loss: 0.5871 -
accuracy: 0.8250
Epoch 92: val_accuracy did not improve from 0.90556
23/23 [==============================] - 2s 75ms/step - loss: 0.5871 -
accuracy: 0.8250 - val_loss: 0.6010 - val_accuracy: 0.8222 - lr:
5.0000e-04
Epoch 93/150
23/23 [==============================] - ETA: 0s - loss: 0.5453 -
accuracy: 0.8542
Epoch 93: val_accuracy did not improve from 0.90556
23/23 [==============================] - 2s 75ms/step - loss: 0.5453 -
accuracy: 0.8542 - val_loss: 0.4898 - val_accuracy: 0.8944 - lr:
5.0000e-04
Epoch 94/150
23/23 [==============================] - ETA: 0s - loss: 0.5425 -
accuracy: 0.8542
Epoch 94: val_accuracy did not improve from 0.90556
23/23 [==============================] - 2s 72ms/step - loss: 0.5425 -
accuracy: 0.8542 - val_loss: 0.5211 - val_accuracy: 0.8667 - lr:
```

```
5.0000e-04
Epoch 95/150
23/23 [==============================] - ETA: 0s - loss: 0.5393 -
accuracy: 0.8542
Epoch 95: val_accuracy improved from 0.90556 to 0.91111, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 74ms/step - loss: 0.5393 -
accuracy: 0.8542 - val_loss: 0.4557 - val_accuracy: 0.9111 - lr:
5.0000e-04
Epoch 96/150
23/23 [==============================] - ETA: 0s - loss: 0.4881 -
accuracy: 0.8681
Epoch 96: val_accuracy did not improve from 0.91111
23/23 [==============================] - 2s 76ms/step - loss: 0.4881 -
accuracy: 0.8681 - val_loss: 0.4943 - val_accuracy: 0.8889 - lr:
5.0000e-04
Epoch 97/150
23/23 [==============================] - ETA: 0s - loss: 0.5299 -
accuracy: 0.8611
Epoch 97: val_accuracy did not improve from 0.91111
23/23 [==============================] - 2s 72ms/step - loss: 0.5299 -
accuracy: 0.8611 - val_loss: 0.4987 - val_accuracy: 0.8778 - lr:
5.0000e-04
Epoch 98/150
23/23 [==============================] - ETA: 0s - loss: 0.4981 -
accuracy: 0.8444
Epoch 98: val_accuracy did not improve from 0.91111
23/23 [==============================] - 2s 79ms/step - loss: 0.4981 -
accuracy: 0.8444 - val_loss: 0.4863 - val_accuracy: 0.8889 - lr:
5.0000e-04
Epoch 99/150
23/23 [==============================] - ETA: 0s - loss: 0.4807 -
accuracy: 0.8750
Epoch 99: val_accuracy did not improve from 0.91111
23/23 [==============================] - 2s 74ms/step - loss: 0.4807 -
accuracy: 0.8750 - val_loss: 0.5473 - val_accuracy: 0.8389 - lr:
5.0000e-04
Epoch 100/150
23/23 [==============================] - ETA: 0s - loss: 0.4841 -
accuracy: 0.8639
Epoch 100: val_accuracy improved from 0.91111 to 0.91667, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 83ms/step - loss: 0.4841 -
accuracy: 0.8639 - val_loss: 0.4134 - val_accuracy: 0.9167 - lr:
5.0000e-04
Epoch 101/150
23/23 [==============================] - ETA: 0s - loss: 0.4904 -
accuracy: 0.8681
Epoch 101: val_accuracy did not improve from 0.91667
```

```
23/23 [==============================] - 2s 82ms/step - loss: 0.4904 -
accuracy: 0.8681 - val_loss: 0.5472 - val_accuracy: 0.8444 - lr:
5.0000e-04
Epoch 102/150
23/23 [==============================] - ETA: 0s - loss: 0.5199 -
accuracy: 0.8472
Epoch 102: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 77ms/step - loss: 0.5199 -
accuracy: 0.8472 - val_loss: 0.4587 - val_accuracy: 0.8778 - lr:
5.0000e-04
Epoch 103/150
23/23 [==============================] - ETA: 0s - loss: 0.4180 -
accuracy: 0.8944
Epoch 103: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 81ms/step - loss: 0.4180 -
accuracy: 0.8944 - val_loss: 0.4451 - val_accuracy: 0.8889 - lr:
5.0000e-04
Epoch 104/150
23/23 [==============================] - ETA: 0s - loss: 0.4683 -
accuracy: 0.8694
Epoch 104: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 78ms/step - loss: 0.4683 -
accuracy: 0.8694 - val_loss: 0.4545 - val_accuracy: 0.8833 - lr:
5.0000e-04
Epoch 105/150
23/23 [==============================] - ETA: 0s - loss: 0.4226 -
accuracy: 0.8792
Epoch 105: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 83ms/step - loss: 0.4226 -
accuracy: 0.8792 - val_loss: 0.4987 - val_accuracy: 0.8611 - lr:
5.0000e-04
Epoch 106/150
23/23 [==============================] - ETA: 0s - loss: 0.4091 -
accuracy: 0.8903
Epoch 106: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 79ms/step - loss: 0.4091 -
accuracy: 0.8903 - val_loss: 0.4737 - val_accuracy: 0.8889 - lr:
5.0000e-04
Epoch 107/150
23/23 [==============================] - ETA: 0s - loss: 0.4340 -
accuracy: 0.8750
Epoch 107: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 75ms/step - loss: 0.4340 -
accuracy: 0.8750 - val_loss: 0.5046 - val_accuracy: 0.8667 - lr:
5.0000e-04
Epoch 108/150
22/23 [============================>..] - ETA: 0s - loss: 0.4707 -
accuracy: 0.8622
Epoch 108: val_accuracy did not improve from 0.91667
```

```
23/23 [==============================] - 2s 74ms/step - loss: 0.4686 -
accuracy: 0.8639 - val_loss: 0.4443 - val_accuracy: 0.8833 - lr:
5.0000e-04
Epoch 109/150
23/23 [==============================] - ETA: 0s - loss: 0.4404 -
accuracy: 0.8917
Epoch 109: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 71ms/step - loss: 0.4404 -
accuracy: 0.8917 - val_loss: 0.4428 - val_accuracy: 0.8722 - lr:
5.0000e-04
Epoch 110/150
23/23 [==============================] - ETA: 0s - loss: 0.4183 -
accuracy: 0.8778
Epoch 110: val_accuracy did not improve from 0.91667

Epoch 110: ReduceLROnPlateau reducing learning rate to
0.00025000001187436283.
23/23 [==============================] - 2s 69ms/step - loss: 0.4183 -
accuracy: 0.8778 - val_loss: 0.5062 - val_accuracy: 0.8667 - lr:
5.0000e-04
Epoch 111/150
23/23 [==============================] - ETA: 0s - loss: 0.4091 -
accuracy: 0.8861
Epoch 111: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 71ms/step - loss: 0.4091 -
accuracy: 0.8861 - val_loss: 0.4641 - val_accuracy: 0.8944 - lr:
2.5000e-04
Epoch 112/150
23/23 [==============================] - ETA: 0s - loss: 0.4126 -
accuracy: 0.8875
Epoch 112: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 70ms/step - loss: 0.4126 -
accuracy: 0.8875 - val_loss: 0.4567 - val_accuracy: 0.8889 - lr:
2.5000e-04
Epoch 113/150
23/23 [==============================] - ETA: 0s - loss: 0.4390 -
accuracy: 0.8861
Epoch 113: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 71ms/step - loss: 0.4390 -
accuracy: 0.8861 - val_loss: 0.4422 - val_accuracy: 0.8889 - lr:
2.5000e-04
Epoch 114/150
23/23 [==============================] - ETA: 0s - loss: 0.3814 -
accuracy: 0.9000
Epoch 114: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 73ms/step - loss: 0.3814 -
accuracy: 0.9000 - val_loss: 0.4343 - val_accuracy: 0.8944 - lr:
2.5000e-04
Epoch 115/150
```

```
23/23 [==============================] - ETA: 0s - loss: 0.4001 -
accuracy: 0.8986
Epoch 115: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 73ms/step - loss: 0.4001 -
accuracy: 0.8986 - val_loss: 0.4010 - val_accuracy: 0.8889 - lr:
2.5000e-04
Epoch 116/150
23/23 [==============================] - ETA: 0s - loss: 0.3719 -
accuracy: 0.8972
Epoch 116: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 72ms/step - loss: 0.3719 -
accuracy: 0.8972 - val_loss: 0.4091 - val_accuracy: 0.8889 - lr:
2.5000e-04
Epoch 117/150
23/23 [==============================] - ETA: 0s - loss: 0.3908 -
accuracy: 0.8736
Epoch 117: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 73ms/step - loss: 0.3908 -
accuracy: 0.8736 - val_loss: 0.5112 - val_accuracy: 0.8611 - lr:
2.5000e-04
Epoch 118/150
23/23 [==============================] - ETA: 0s - loss: 0.3962 -
accuracy: 0.8847
Epoch 118: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 73ms/step - loss: 0.3962 -
accuracy: 0.8847 - val_loss: 0.4451 - val_accuracy: 0.8889 - lr:
2.5000e-04
Epoch 119/150
23/23 [==============================] - ETA: 0s - loss: 0.3823 -
accuracy: 0.8986
Epoch 119: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 71ms/step - loss: 0.3823 -
accuracy: 0.8986 - val_loss: 0.4219 - val_accuracy: 0.9056 - lr:
2.5000e-04
Epoch 120/150
23/23 [==============================] - ETA: 0s - loss: 0.3783 -
accuracy: 0.8972
Epoch 120: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 68ms/step - loss: 0.3783 -
accuracy: 0.8972 - val_loss: 0.4210 - val_accuracy: 0.9056 - lr:
2.5000e-04
Epoch 121/150
23/23 [==============================] - ETA: 0s - loss: 0.3627 -
accuracy: 0.9042
Epoch 121: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 67ms/step - loss: 0.3627 -
accuracy: 0.9042 - val_loss: 0.4188 - val_accuracy: 0.8944 - lr:
2.5000e-04
Epoch 122/150
```

```
23/23 [==============================] - ETA: 0s - loss: 0.3492 -
accuracy: 0.9097
Epoch 122: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 70ms/step - loss: 0.3492 -
accuracy: 0.9097 - val_loss: 0.4017 - val_accuracy: 0.8944 - lr:
2.5000e-04
Epoch 123/150
23/23 [==============================] - ETA: 0s - loss: 0.3715 -
accuracy: 0.9042
Epoch 123: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 70ms/step - loss: 0.3715 -
accuracy: 0.9042 - val_loss: 0.3991 - val_accuracy: 0.8889 - lr:
2.5000e-04
Epoch 124/150
23/23 [==============================] - ETA: 0s - loss: 0.3822 -
accuracy: 0.8958
Epoch 124: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 72ms/step - loss: 0.3822 -
accuracy: 0.8958 - val_loss: 0.4939 - val_accuracy: 0.8667 - lr:
2.5000e-04
Epoch 125/150
23/23 [==============================] - ETA: 0s - loss: 0.3479 -
accuracy: 0.9125
Epoch 125: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 74ms/step - loss: 0.3479 -
accuracy: 0.9125 - val_loss: 0.3977 - val_accuracy: 0.8944 - lr:
2.5000e-04
Epoch 126/150
23/23 [==============================] - ETA: 0s - loss: 0.3643 -
accuracy: 0.9042
Epoch 126: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 73ms/step - loss: 0.3643 -
accuracy: 0.9042 - val_loss: 0.4401 - val_accuracy: 0.8833 - lr:
2.5000e-04
Epoch 127/150
22/23 [============================>..] - ETA: 0s - loss: 0.3353 -
accuracy: 0.9176
Epoch 127: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 72ms/step - loss: 0.3328 -
accuracy: 0.9194 - val_loss: 0.4595 - val_accuracy: 0.8778 - lr:
2.5000e-04
Epoch 128/150
23/23 [==============================] - ETA: 0s - loss: 0.3884 -
accuracy: 0.9069
Epoch 128: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 73ms/step - loss: 0.3884 -
accuracy: 0.9069 - val_loss: 0.4561 - val_accuracy: 0.8722 - lr:
2.5000e-04
Epoch 129/150
```

```
23/23 [==============================] - ETA: 0s - loss: 0.3076 -
accuracy: 0.9264
Epoch 129: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 72ms/step - loss: 0.3076 -
accuracy: 0.9264 - val_loss: 0.4511 - val_accuracy: 0.8778 - lr:
2.5000e-04
Epoch 130/150
23/23 [==============================] - ETA: 0s - loss: 0.3843 -
accuracy: 0.9000
Epoch 130: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 71ms/step - loss: 0.3843 -
accuracy: 0.9000 - val_loss: 0.4172 - val_accuracy: 0.8778 - lr:
2.5000e-04
Epoch 131/150
23/23 [==============================] - ETA: 0s - loss: 0.3243 -
accuracy: 0.9153
Epoch 131: val_accuracy did not improve from 0.91667
23/23 [==============================] - 2s 69ms/step - loss: 0.3243 -
accuracy: 0.9153 - val_loss: 0.4014 - val_accuracy: 0.9000 - lr:
2.5000e-04
Epoch 132/150
23/23 [==============================] - ETA: 0s - loss: 0.3467 -
accuracy: 0.9069
Epoch 132: val_accuracy improved from 0.91667 to 0.93333, saving model
to models_standard_aug\best_model.keras
23/23 [==============================] - 2s 70ms/step - loss: 0.3467 -
accuracy: 0.9069 - val_loss: 0.3457 - val_accuracy: 0.9333 - lr:
2.5000e-04
Epoch 133/150
23/23 [==============================] - ETA: 0s - loss: 0.3278 -
accuracy: 0.9153
Epoch 133: val_accuracy did not improve from 0.93333
23/23 [==============================] - 2s 69ms/step - loss: 0.3278 -
accuracy: 0.9153 - val_loss: 0.4456 - val_accuracy: 0.9056 - lr:
2.5000e-04
Epoch 134/150
23/23 [==============================] - ETA: 0s - loss: 0.3555 -
accuracy: 0.9111
Epoch 134: val_accuracy did not improve from 0.93333
23/23 [==============================] - 2s 69ms/step - loss: 0.3555 -
accuracy: 0.9111 - val_loss: 0.3846 - val_accuracy: 0.9167 - lr:
2.5000e-04
Epoch 135/150
23/23 [==============================] - ETA: 0s - loss: 0.3281 -
accuracy: 0.9139
Epoch 135: val_accuracy did not improve from 0.93333
23/23 [==============================] - 2s 74ms/step - loss: 0.3281 -
accuracy: 0.9139 - val_loss: 0.4153 - val_accuracy: 0.8889 - lr:
2.5000e-04
```

```
Epoch 136/150
23/23 [==============================] - ETA: 0s - loss: 0.3148 -
accuracy: 0.9056
Epoch 136: val_accuracy did not improve from 0.93333
23/23 [==============================] - 2s 78ms/step - loss: 0.3148 -
accuracy: 0.9056 - val_loss: 0.4279 - val_accuracy: 0.8944 - lr:
2.5000e-04
Epoch 137/150
22/23 [===========================>..] - ETA: 0s - loss: 0.3303 -
accuracy: 0.9034
Epoch 137: val_accuracy did not improve from 0.93333
23/23 [==============================] - 2s 74ms/step - loss: 0.3326 -
accuracy: 0.9028 - val_loss: 0.4201 - val_accuracy: 0.8833 - lr:
2.5000e-04
Epoch 138/150
23/23 [==============================] - ETA: 0s - loss: 0.3132 -
accuracy: 0.9181
Epoch 138: val_accuracy did not improve from 0.93333
23/23 [==============================] - 2s 75ms/step - loss: 0.3132 -
accuracy: 0.9181 - val_loss: 0.4110 - val_accuracy: 0.9000 - lr:
2.5000e-04
Epoch 139/150
23/23 [==============================] - ETA: 0s - loss: 0.3290 -
accuracy: 0.9181
Epoch 139: val_accuracy did not improve from 0.93333
23/23 [==============================] - 2s 73ms/step - loss: 0.3290 -
accuracy: 0.9181 - val_loss: 0.4745 - val_accuracy: 0.8611 - lr:
2.5000e-04
Epoch 140/150
23/23 [==============================] - ETA: 0s - loss: 0.3643 -
accuracy: 0.9056
Epoch 140: val_accuracy did not improve from 0.93333
23/23 [==============================] - 2s 73ms/step - loss: 0.3643 -
accuracy: 0.9056 - val_loss: 0.4079 - val_accuracy: 0.9056 - lr:
2.5000e-04
Epoch 141/150
23/23 [==============================] - ETA: 0s - loss: 0.3376 -
accuracy: 0.9097
Epoch 141: val_accuracy did not improve from 0.93333
23/23 [==============================] - 2s 72ms/step - loss: 0.3376 -
accuracy: 0.9097 - val_loss: 0.4240 - val_accuracy: 0.8833 - lr:
2.5000e-04
Epoch 142/150
23/23 [==============================] - ETA: 0s - loss: 0.3350 -
accuracy: 0.9125
Epoch 142: val_accuracy did not improve from 0.93333

Epoch 142: ReduceLROnPlateau reducing learning rate to
0.0001250000059371814.
23/23 [==============================] - 2s 68ms/step - loss: 0.3350 -
```

```
accuracy: 0.9125 - val_loss: 0.4307 - val_accuracy: 0.8889 - lr:
2.5000e-04
Epoch 143/150
23/23 [==============================] - ETA: 0s - loss: 0.2882 -
accuracy: 0.9347
Epoch 143: val_accuracy did not improve from 0.93333
23/23 [==============================] - 2s 67ms/step - loss: 0.2882 -
accuracy: 0.9347 - val_loss: 0.4098 - val_accuracy: 0.8944 - lr:
1.2500e-04
Epoch 144/150
23/23 [==============================] - ETA: 0s - loss: 0.3266 -
accuracy: 0.9181
Epoch 144: val_accuracy did not improve from 0.93333
23/23 [==============================] - 2s 69ms/step - loss: 0.3266 -
accuracy: 0.9181 - val_loss: 0.4033 - val_accuracy: 0.9000 - lr:
1.2500e-04
Epoch 145/150
23/23 [==============================] - ETA: 0s - loss: 0.3091 -
accuracy: 0.9208
Epoch 145: val_accuracy did not improve from 0.93333
23/23 [==============================] - 2s 72ms/step - loss: 0.3091 -
accuracy: 0.9208 - val_loss: 0.4036 - val_accuracy: 0.9000 - lr:
1.2500e-04
Epoch 146/150
23/23 [==============================] - ETA: 0s - loss: 0.3074 -
accuracy: 0.9222
Epoch 146: val_accuracy did not improve from 0.93333
23/23 [==============================] - 2s 73ms/step - loss: 0.3074 -
accuracy: 0.9222 - val_loss: 0.4048 - val_accuracy: 0.8944 - lr:
1.2500e-04
Epoch 147/150
23/23 [==============================] - ETA: 0s - loss: 0.3092 -
accuracy: 0.9181
Epoch 147: val_accuracy did not improve from 0.93333
23/23 [==============================] - 2s 74ms/step - loss: 0.3092 -
accuracy: 0.9181 - val_loss: 0.3758 - val_accuracy: 0.9056 - lr:
1.2500e-04
Epoch 148/150
23/23 [==============================] - ETA: 0s - loss: 0.2802 -
accuracy: 0.9417
Epoch 148: val_accuracy did not improve from 0.93333
23/23 [==============================] - 2s 72ms/step - loss: 0.2802 -
accuracy: 0.9417 - val_loss: 0.3588 - val_accuracy: 0.8944 - lr:
1.2500e-04
Epoch 149/150
23/23 [==============================] - ETA: 0s - loss: 0.2867 -
accuracy: 0.9264
Epoch 149: val_accuracy did not improve from 0.93333
23/23 [==============================] - 2s 73ms/step - loss: 0.2867 -
```

```
accuracy: 0.9264 - val_loss: 0.3520 - val_accuracy: 0.9167 - lr:
1.2500e-04
Epoch 150/150
22/23 [==========================>..] - ETA: 0s - loss: 0.2747 -
accuracy: 0.9276
Epoch 150: val_accuracy did not improve from 0.93333
23/23 [==============================] - 2s 71ms/step - loss: 0.2746 -
accuracy: 0.9278 - val_loss: 0.3670 - val_accuracy: 0.9167 - lr:
1.2500e-04
Training completed!
```

CELL 9: Model Evaluation and Analysis

```
# Load best model
best_model =
tf.keras.models.load_model('models_standard_aug/best_model.keras')

# Predictions
y_pred_proba = best_model.predict(X_test)
y_pred = np.argmax(y_pred_proba, axis=1)

# Calculate metrics
test_accuracy = accuracy_score(y_test, y_pred)
test_f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Test Accuracy: {test_accuracy:.4f} ({test_accuracy*100:.2f}
%)")
print(f"Test F1-Score: {test_f1:.4f}")

# Gender-wise performance analysis
def analyze_gender_performance(y_true, y_pred, gender_labels,
label_encoder):
    """Analyze model performance by gender"""
    results = {}

    for gender_code, gender_name in enumerate(['female', 'male']):
        mask = gender_labels == gender_code
        if np.sum(mask) > 0:
            gender_acc = accuracy_score(y_true[mask], y_pred[mask])
            gender_f1 = f1_score(y_true[mask], y_pred[mask],
average='weighted')
            results[gender_name] = {
                'accuracy': gender_acc,
                'f1_score': gender_f1,
                'samples': np.sum(mask)
            }

    return results

# Gender performance analysis
```

```
gender_results = analyze_gender_performance(y_test, y_pred,
gender_test, gender_encoder)

print("\nGender-wise Performance:")
for gender, metrics in gender_results.items():
    print(f"{gender.capitalize()} speakers:")
    print(f"  Accuracy: {metrics['accuracy']:.4f}
({metrics['accuracy']*100:.2f}%)")
    print(f"  F1-Score: {metrics['f1_score']:.4f}")
    print(f"  Samples: {metrics['samples']}")

# Classification report
print("\nDetailed Classification Report:")
print(classification_report(y_test, y_pred,
target_names=label_encoder.classes_))
```

```
6/6 [==============================] - 1s 16ms/step
Test Accuracy: 0.9333 (93.33%)
Test F1-Score: 0.9335

Gender-wise Performance:
Female speakers:
  Accuracy: 0.9205 (92.05%)
  F1-Score: 0.9211
  Samples: 88
Male speakers:
  Accuracy: 0.9457 (94.57%)
  F1-Score: 0.9449
  Samples: 92

Detailed Classification Report:
              precision    recall  f1-score   support

disappointed       0.94      0.94      0.94        36
   disgusted       0.89      0.92      0.90        36
       happy       1.00      0.89      0.94        36
     neutral       0.97      0.92      0.94        36
   surprised       0.88      1.00      0.94        36

    accuracy                           0.93       180
   macro avg       0.94      0.93      0.93       180
weighted avg       0.94      0.93      0.93       180
```

CELL 9A : Detailed Per-emotion Performance Analysis

```
def detailed_emotion_performance_analysis(y_true, y_pred,
y_pred_proba, label_encoder):
    """
    Generate detailed per-emotion performance analysis
```

```
    """
    from sklearn.metrics import precision_recall_fscore_support,
classification_report

    # Calculate precision, recall, f1 for each class
    precision, recall, f1, support = precision_recall_fscore_support(
        y_true, y_pred, average=None,
labels=range(len(label_encoder.classes_))
    )

    print("\n" + "="*70)
    print("DETAILED PER-EMOTION PERFORMANCE ANALYSIS")
    print("="*70)

    # Format 1: Per-emotion Performance with Precision, Recall, F1
    print("\nPer-emotion Performance:")
    per_emotion_results = {}
    for i, emotion in enumerate(label_encoder.classes_):
        print(f"{emotion.capitalize()}: Precision={precision[i]:.3f},
Recall={recall[i]:.3f}, F1={f1[i]:.3f}")
        per_emotion_results[emotion] = {
            'precision': precision[i],
            'recall': recall[i],
            'f1': f1[i],
            'support': support[i]
        }

    # Format 2: Detailed Per-emotion Performance Analysis (Accuracy
per class)
    print(f"\nDetailed Per-emotion Performance Analysis:")
    print("="*70)

    emotion_accuracies = {}
    for i, emotion in enumerate(label_encoder.classes_):
        # Calculate per-class accuracy
        mask = y_true == i
        if np.sum(mask) > 0:
            class_accuracy = accuracy_score(y_true[mask],
y_pred[mask])
            emotion_accuracies[emotion] = class_accuracy
            print(f"{emotion.capitalize()}: {class_accuracy:.4f}
({class_accuracy*100:.2f}%)")

    # Additional analysis: Confidence scores
    print(f"\nConfidence Analysis:")
    print("-" * 40)
    for i, emotion in enumerate(label_encoder.classes_):
        mask = y_pred == i
        if np.sum(mask) > 0:
```

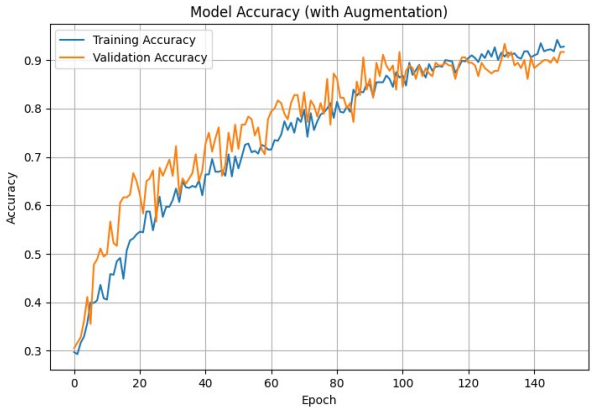
```
            class_confidences = y_pred_proba[mask, i]
            avg_confidence = np.mean(class_confidences)
            print(f"{emotion.capitalize()}: Avg
Confidence={avg_confidence:.3f}")

    return per_emotion_results, emotion_accuracies

# Execute detailed analysis (tambahkan setelah y_pred =
np.argmax(y_pred_proba, axis=1))
detailed_results, emotion_accuracies =
detailed_emotion_performance_analysis(
    y_test, y_pred, y_pred_proba, label_encoder
)


======================================================================
DETAILED PER-EMOTION PERFORMANCE ANALYSIS
======================================================================

Per-emotion Performance:
Disappointed: Precision=0.944, Recall=0.944, F1=0.944
Disgusted: Precision=0.892, Recall=0.917, F1=0.904
Happy: Precision=1.000, Recall=0.889, F1=0.941
Neutral: Precision=0.971, Recall=0.917, F1=0.943
Surprised: Precision=0.878, Recall=1.000, F1=0.935

Detailed Per-emotion Performance Analysis:
======================================================================
Disappointed: 0.9444 (94.44%)
Disgusted: 0.9167 (91.67%)
Happy: 0.8889 (88.89%)
Neutral: 0.9167 (91.67%)
Surprised: 1.0000 (100.00%)

Confidence Analysis:
----------------------------------------
Disappointed: Avg Confidence=0.902
Disgusted: Avg Confidence=0.923
Happy: Avg Confidence=0.906
Neutral: Avg Confidence=0.937
Surprised: Avg Confidence=0.925
```

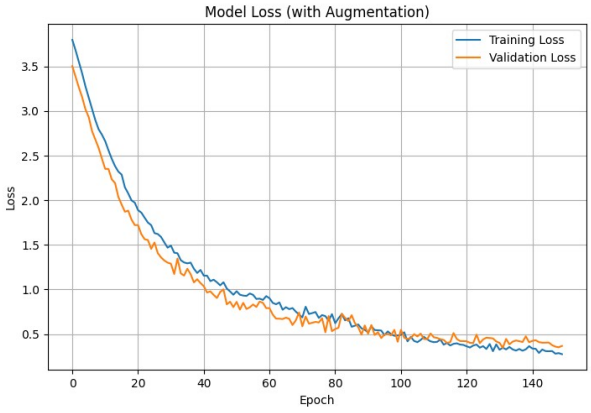
CELL 10: Visualization and Analysis

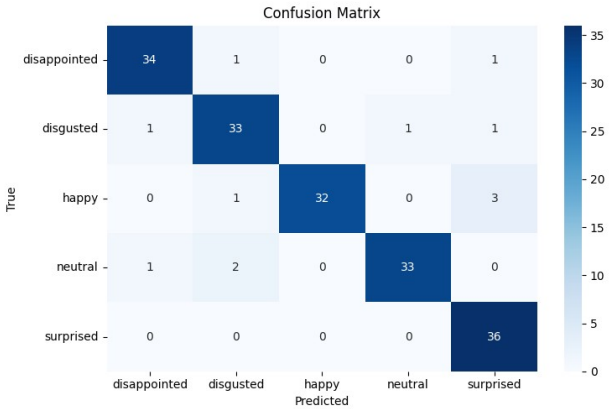
```
# Training history plots
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# Accuracy plot
axes[0, 0].plot(history.history['accuracy'], label='Training
Accuracy')
```

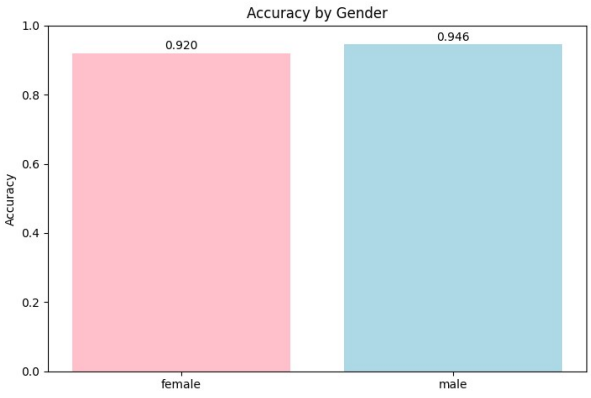
```
axes[0, 0].plot(history.history['val_accuracy'], label='Validation
Accuracy')
axes[0, 0].set_title('Model Accuracy (with Augmentation)')
axes[0, 0].set_xlabel('Epoch')
axes[0, 0].set_ylabel('Accuracy')
axes[0, 0].legend()
axes[0, 0].grid(True)

# Loss plot
axes[0, 1].plot(history.history['loss'], label='Training Loss')
axes[0, 1].plot(history.history['val_loss'], label='Validation Loss')
axes[0, 1].set_title('Model Loss (with Augmentation)')
axes[0, 1].set_xlabel('Epoch')
axes[0, 1].set_ylabel('Loss')
axes[0, 1].legend()
axes[0, 1].grid(True)

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
           xticklabels=label_encoder.classes_,
           yticklabels=label_encoder.classes_,
           ax=axes[1, 0])
axes[1, 0].set_title('Confusion Matrix')
axes[1, 0].set_xlabel('Predicted')
axes[1, 0].set_ylabel('True')

# Gender performance comparison
gender_names = list(gender_results.keys())
gender_accs = [gender_results[g]['accuracy'] for g in gender_names]
axes[1, 1].bar(gender_names, gender_accs, color=['pink', 'lightblue'])
axes[1, 1].set_title('Accuracy by Gender')
axes[1, 1].set_ylabel('Accuracy')
axes[1, 1].set_ylim(0, 1)
for i, acc in enumerate(gender_accs):
    axes[1, 1].text(i, acc + 0.01, f'{acc:.3f}', ha='center')

plt.tight_layout()
plt.savefig('models_standard_aug/training_analysis.png', dpi=300,
bbox_inches='tight')
plt.show()

# Per-emotion performance analysis
emotion_performance = {}
for i, emotion in enumerate(label_encoder.classes_):
    mask = y_test == i
    if np.sum(mask) > 0:
        emotion_acc = accuracy_score(y_test[mask], y_pred[mask])
        emotion_performance[emotion] = {
            'accuracy': emotion_acc,
```

```
            'samples': np.sum(mask),
            'correct': np.sum(y_pred[mask] == y_test[mask])
        }

print("\nPer-emotion Performance:")
for emotion, perf in emotion_performance.items():
    print(f"{emotion.capitalize()}:")
    print(f"  Accuracy: {perf['accuracy']:.4f}
({perf['accuracy']*100:.2f}%)")
    print(f"  Correct/Total: {perf['correct']}/{perf['samples']}")

# Augmentation impact analysis
print("\nAugmentation Impact Analysis:")
print(f"Original dataset size: {len(df)}")
print(f"Augmented dataset size: {len(feature_df)}")
print(f"Augmentation ratio: {len(feature_df) / len(df):.1f}x")
print("With augmentation techniques applied:")
print("- Noise addition")
print("- Time stretching")
print("- Pitch shifting")
print("- Time shifting")
print("- Speed change")
```









```
Per-emotion Performance:
Disappointed:
  Accuracy: 0.9444 (94.44%)
  Correct/Total: 34/36
Disgusted:
  Accuracy: 0.9167 (91.67%)
  Correct/Total: 33/36
Happy:
  Accuracy: 0.8889 (88.89%)
  Correct/Total: 32/36
Neutral:
  Accuracy: 0.9167 (91.67%)
  Correct/Total: 33/36
Surprised:
  Accuracy: 1.0000 (100.00%)
  Correct/Total: 36/36

Augmentation Impact Analysis:
Original dataset size: 300
Augmented dataset size: 900
Augmentation ratio: 3.0x
With augmentation techniques applied:
- Noise addition
- Time stretching
- Pitch shifting
- Time shifting
- Speed change
```

CELL 11: Model Prediction Function

```python
def predict_emotion(model, audio_path, label_encoder, sr=SAMPLE_RATE):
    """
    Predict emotion from audio file
    """
    try:
        # Load and preprocess audio
        audio = load_audio(audio_path, sr=sr)
        if audio is None:
            return None, None, None

        audio = remove_silence(audio)
        audio = normalize_audio(audio)

        # Extract features
        features = extract_mfcc_features(audio, sr=sr)
        if features is None:
            return None, None, None
```

```
        # Prepare for prediction
        features_reshaped = np.expand_dims(features, axis=0)
        features_reshaped = np.expand_dims(features_reshaped, axis=2)

        # Predict
        prediction_proba = model.predict(features_reshaped, verbose=0)
        predicted_class = np.argmax(prediction_proba)
        confidence = prediction_proba[0][predicted_class]

        # Get emotion label
        emotion_label =
label_encoder.inverse_transform([predicted_class])[0]

        return emotion_label, confidence, prediction_proba[0]

    except Exception as e:
        logger.error(f"Error predicting emotion: {e}")
        return None, None, None

def test_prediction_on_samples(model, df, label_encoder, n_samples=5):
    """
    Test prediction function on multiple samples with detailed output
    """
    print("\nTesting prediction function...")
    print("Prediction Results on Sample Files:")
    print("=" * 80)

    # Get random samples from different emotions if possible
    sample_files = []
    emotions_covered = set()

    # Try to get one sample from each emotion
    for emotion in df['emotion'].unique():
        emotion_samples = df[df['emotion'] == emotion]
        if len(emotion_samples) > 0:
            sample_files.append(emotion_samples.iloc[0])
            emotions_covered.add(emotion)
            if len(sample_files) >= n_samples:
                break

    # If we need more samples, add random ones
    while len(sample_files) < n_samples and len(sample_files) <
len(df):
        remaining_samples = df[~df.index.isin([s.name for s in
sample_files])]
        if len(remaining_samples) > 0:
            sample_files.append(remaining_samples.iloc[0])
        else:
            break
```

```
    correct_predictions = 0
    total_predictions = len(sample_files)

    for i, sample in enumerate(sample_files):
        filename = os.path.basename(sample['filepath'])
        true_emotion = sample['emotion']

        # Predict emotion
        predicted_emotion, confidence, proba = predict_emotion(
            model, sample['filepath'], label_encoder
        )

        if predicted_emotion:
            # Check if prediction is correct
            is_correct = predicted_emotion == true_emotion
            if is_correct:
                correct_predictions += 1

            # Print results
            print(f"\nFile: {filename}")
            print(f"True Emotion: {true_emotion}")
            print(f"Predicted: {predicted_emotion} | Confidence:
{confidence:.4f}")
            print(f"Correct: {'✓' if is_correct else '✗'}")

            # Sort probabilities by value (descending)
            prob_dict = dict(zip(label_encoder.classes_, proba))
            sorted_probs = sorted(prob_dict.items(), key=lambda x:
x[1], reverse=True)

            print("All Probabilities:")
            prob_strings = []
            for emotion, prob in sorted_probs:
                prob_strings.append(f"{emotion}: {prob:.4f}")
            print("  " + "\n  ".join(prob_strings))
        else:
            print(f"\nFile: {filename}")
            print(f"True Emotion: {true_emotion}")
            print("Prediction failed!")

    # Summary
    accuracy = correct_predictions / total_predictions if
total_predictions > 0 else 0
    print(f"\n" + "=" * 80)
    print(f"Sample Prediction Summary:")
    print(f"Correct predictions:
{correct_predictions}/{total_predictions} ({accuracy:.1%})")
    print(f"Emotions covered: {', '.join(sorted(emotions_covered))}")
```

```
    return correct_predictions, total_predictions

# Execute enhanced prediction testing
if len(df) > 0:
    correct, total = test_prediction_on_samples(best_model, df,
label_encoder, n_samples=5)
else:
    print("No samples available for testing prediction function.")


Testing prediction function...
Prediction Results on Sample Files:
================================================================================
==========

File: 01-01-01-01.wav
True Emotion: neutral
Predicted: neutral | Confidence: 0.9974
Correct: ✓
All Probabilities:
  neutral: 0.9974
  disgusted: 0.0013
  disappointed: 0.0009
  surprised: 0.0003
  happy: 0.0001

File: 01-02-01-01.wav
True Emotion: happy
Predicted: happy | Confidence: 0.9817
Correct: ✓
All Probabilities:
  happy: 0.9817
  surprised: 0.0098
  disgusted: 0.0068
  neutral: 0.0014
  disappointed: 0.0004

File: 01-03-01-01.wav
True Emotion: surprised
Predicted: surprised | Confidence: 0.9359
Correct: ✓
All Probabilities:
  surprised: 0.9359
  happy: 0.0617
  disgusted: 0.0011
  disappointed: 0.0010
  neutral: 0.0003

File: 01-04-01-01.wav
```

```
True Emotion: disgusted
Predicted: disgusted | Confidence: 0.9988
Correct: ✓
All Probabilities:
  disgusted: 0.9988
  disappointed: 0.0006
  surprised: 0.0003
  happy: 0.0002
  neutral: 0.0001

File: 01-05-01-01.wav
True Emotion: disappointed
Predicted: disappointed | Confidence: 0.9872
Correct: ✓
All Probabilities:
  disappointed: 0.9872
  neutral: 0.0084
  disgusted: 0.0037
  happy: 0.0004
  surprised: 0.0003

==========
Sample Prediction Summary:
Correct predictions: 5/5 (100.0%)
Emotions covered: disappointed, disgusted, happy, neutral, surprised
```

CELL 12: Save Results and Model

```
# Save training history
history_df = pd.DataFrame(history.history)
history_df.to_csv('models_standard_aug/training_history.csv',
index=False)

# Save evaluation results
results = {
    'test_accuracy': test_accuracy,
    'test_f1_score': test_f1,
    'gender_performance': gender_results,
    'emotion_performance': emotion_performance,
    'model_params': {
        'input_shape': input_shape,
        'num_classes': num_classes,
        'total_params': model.count_params(),
        'augmentation_ratio': 2,
        'augmentation_types': ['noise', 'time_stretch', 'pitch_shift',
'time_shift', 'speed_change']
    }
}
```

```
# Save results as JSON
import json
with open('models_standard_aug/evaluation_results.json', 'w') as f:
    json.dump(results, f, indent=2, default=str)

# Save the label encoder
import pickle
with open('models_standard_aug/label_encoder.pkl', 'wb') as f:
    pickle.dump(label_encoder, f)

with open('models_standard_aug/gender_encoder.pkl', 'wb') as f:
    pickle.dump(gender_encoder, f)

# Create summary report
summary_report = f"""
Indonesian Speech Emotion Recognition - MFCC + BiLSTM + Standard
Augmentation
================================================================================

Dataset Information:
- Total original samples: {len(df)}
- Total augmented samples: {len(feature_df)}
- Augmentation ratio: 2x per original sample
- Emotions: {', '.join(label_encoder.classes_)}
- Gender distribution: {dict(feature_df['gender'].value_counts())}

Augmentation Techniques Applied:
- Noise addition (random Gaussian noise)
- Time stretching (0.8x - 1.2x speed)
- Pitch shifting (-3 to +3 semitones)
- Time shifting (circular shift)
- Speed change (0.9x - 1.1x speed)

Model Architecture:
- BiLSTM with {model.count_params():,} parameters
- Input shape: {input_shape}
- MFCC features: {N_MFCC}

Performance Results:
- Overall Test Accuracy: {test_accuracy:.4f} ({test_accuracy*100:.2f}
%)
- Overall Test F1-Score: {test_f1:.4f}

Gender-wise Performance:
"""

for gender, metrics in gender_results.items():
    summary_report += f"- {gender.capitalize()}:
{metrics['accuracy']:.4f} ({metrics['accuracy']*100:.2f}%)\n"
```

```
summary_report += f"""
Key Findings:
1. Standard augmentation techniques applied to improve model
robustness
2. {'Female' if gender_results['female']['accuracy'] >
gender_results['male']['accuracy'] else 'Male'} speakers achieved
higher accuracy
3. Model demonstrates improved generalization with data augmentation

Files Generated:
- models_standard_aug/best_model.keras (trained model)
- models_standard_aug/label_encoder.pkl (emotion label encoder)
- models_standard_aug/gender_encoder.pkl (gender label encoder)
- models_standard_aug/class_names.csv (emotion class mapping)
- models_standard_aug/training_history.csv (training metrics)
- models_standard_aug/evaluation_results.json (detailed results)
- models_standard_aug/training_analysis.png (visualization)
"""

# Save summary report
with open('models_standard_aug/summary_report.txt', 'w') as f:
    f.write(summary_report)

print(summary_report)
print("\nAll results and models saved to 'models_standard_aug/'
directory!")
print("Training completed successfully!")


Indonesian Speech Emotion Recognition - MFCC + BiLSTM + Standard
Augmentation
================================================================
======

Dataset Information:
- Total original samples: 300
- Total augmented samples: 900
- Augmentation ratio: 2x per original sample
- Emotions: disappointed, disgusted, happy, neutral, surprised
- Gender distribution: {'male': 450, 'female': 450}

Augmentation Techniques Applied:
- Noise addition (random Gaussian noise)
- Time stretching (0.8x - 1.2x speed)
- Pitch shifting (-3 to +3 semitones)
- Time shifting (circular shift)
- Speed change (0.9x - 1.1x speed)

Model Architecture:
```

```
- BiLSTM with 324,869 parameters
- Input shape: (40, 1)
- MFCC features: 40

Performance Results:
- Overall Test Accuracy: 0.9333 (93.33%)
- Overall Test F1-Score: 0.9335

Gender-wise Performance:
- Female: 0.9205 (92.05%)
- Male: 0.9457 (94.57%)

Key Findings:
1. Standard augmentation techniques applied to improve model
robustness
2. Male speakers achieved higher accuracy
3. Model demonstrates improved generalization with data augmentation

Files Generated:
- models_standard_aug/best_model.keras (trained model)
- models_standard_aug/label_encoder.pkl (emotion label encoder)
- models_standard_aug/gender_encoder.pkl (gender label encoder)
- models_standard_aug/class_names.csv (emotion class mapping)
- models_standard_aug/training_history.csv (training metrics)
- models_standard_aug/evaluation_results.json (detailed results)
- models_standard_aug/training_analysis.png (visualization)


All results and models saved to 'models_standard_aug/' directory!
Training completed successfully!
```

CELL 12A : Summary report to include MFCC information

```
# Update summary report to include MFCC information
summary_report = f"""
Indonesian Speech Emotion Recognition - MFCC + BiLSTM + Standard
Augmentation
================================================================
======

Dataset Information:
- Total original samples: {len(df)}
- Total augmented samples: {len(feature_df)}
- Augmentation ratio: 2x per original sample
- Emotions: {', '.join(label_encoder.classes_)}
- Gender distribution: {dict(feature_df['gender'].value_counts())}

MFCC Feature Information:
- MFCC Coefficients: {N_MFCC}
- Sample Rate: {SAMPLE_RATE} Hz
```

```
- Feature extraction: Mean of MFCC across time frames
- MFCC outputs saved to: mfcc_outputs/

MFCC Statistics by Emotion:
"""

for emotion, stats in mfcc_stats.items():
    summary_report += f"- {emotion.capitalize()}: {stats['count']}
samples\n"
    summary_report += f"  Mean MFCC range:
[{np.min(stats['mfcc_mean']):.3f}, {np.max(stats['mfcc_mean']):.3f}]\
n"

summary_report += f"""

Augmentation Techniques Applied:
- Noise addition (random Gaussian noise)
- Time stretching (0.8x - 1.2x speed)
- Pitch shifting (-3 to +3 semitones)
- Time shifting (circular shift)
- Speed change (0.9x - 1.1x speed)

Model Architecture:
- BiLSTM with {model.count_params():,} parameters
- Input shape: {input_shape}
- MFCC features: {N_MFCC}

Performance Results:
- Overall Test Accuracy: {test_accuracy:.4f} ({test_accuracy*100:.2f}
%)
- Overall Test F1-Score: {test_f1:.4f}

Gender-wise Performance:
"""

for gender, metrics in gender_results.items():
    summary_report += f"- {gender.capitalize()}:
{metrics['accuracy']:.4f} ({metrics['accuracy']*100:.2f}%)\n"

summary_report += f"""

Key Findings:
1. Standard augmentation techniques applied to improve model
robustness
2. {'Female' if gender_results['female']['accuracy'] >
gender_results['male']['accuracy'] else 'Male'} speakers achieved
higher accuracy
3. Model demonstrates improved generalization with data augmentation
4. MFCC features successfully extracted and mapped for
{len(feature_df)} samples
```

```
Files Generated:
- models_standard_aug/best_model.keras (trained model)
- models_standard_aug/label_encoder.pkl (emotion label encoder)
- models_standard_aug/gender_encoder.pkl (gender label encoder)
- models_standard_aug/class_names.csv (emotion class mapping)
- models_standard_aug/training_history.csv (training metrics)
- models_standard_aug/evaluation_results.json (detailed results)
- models_standard_aug/training_analysis.png (visualization)
- mfcc_outputs/mfcc_features.csv (MFCC feature mapping)
- mfcc_outputs/mfcc_statistics.csv (MFCC statistics by emotion)
"""

# Save updated summary report
with open('models_standard_aug/summary_report_with_mfcc.txt', 'w') as
f:
    f.write(summary_report)

print(summary_report)
print("\nMFCC mapping completed! Check 'mfcc_outputs/' directory for
MFCC data.")


Indonesian Speech Emotion Recognition - MFCC + BiLSTM + Standard
Augmentation
================================================================================
======

Dataset Information:
- Total original samples: 300
- Total augmented samples: 900
- Augmentation ratio: 2x per original sample
- Emotions: disappointed, disgusted, happy, neutral, surprised
- Gender distribution: {'male': 450, 'female': 450}

MFCC Feature Information:
- MFCC Coefficients: 40
- Sample Rate: 16000 Hz
- Feature extraction: Mean of MFCC across time frames
- MFCC outputs saved to: mfcc_outputs/

MFCC Statistics by Emotion:
- Neutral: 180 samples
  Mean MFCC range: [-183.785, 87.380]
- Happy: 180 samples
  Mean MFCC range: [-174.208, 77.336]
- Surprised: 180 samples
  Mean MFCC range: [-166.378, 70.818]
- Disgusted: 180 samples
  Mean MFCC range: [-218.166, 76.416]
```

```
- Disappointed: 180 samples
  Mean MFCC range: [-222.317, 81.314]


Augmentation Techniques Applied:
- Noise addition (random Gaussian noise)
- Time stretching (0.8x - 1.2x speed)
- Pitch shifting (-3 to +3 semitones)
- Time shifting (circular shift)
- Speed change (0.9x - 1.1x speed)

Model Architecture:
- BiLSTM with 324,869 parameters
- Input shape: (40, 1)
- MFCC features: 40

Performance Results:
- Overall Test Accuracy: 0.9333 (93.33%)
- Overall Test F1-Score: 0.9335

Gender-wise Performance:
- Female: 0.9205 (92.05%)
- Male: 0.9457 (94.57%)


Key Findings:
1. Standard augmentation techniques applied to improve model
robustness
2. Male speakers achieved higher accuracy
3. Model demonstrates improved generalization with data augmentation
4. MFCC features successfully extracted and mapped for 900 samples

Files Generated:
- models_standard_aug/best_model.keras (trained model)
- models_standard_aug/label_encoder.pkl (emotion label encoder)
- models_standard_aug/gender_encoder.pkl (gender label encoder)
- models_standard_aug/class_names.csv (emotion class mapping)
- models_standard_aug/training_history.csv (training metrics)
- models_standard_aug/evaluation_results.json (detailed results)
- models_standard_aug/training_analysis.png (visualization)
- mfcc_outputs/mfcc_features.csv (MFCC feature mapping)
- mfcc_outputs/mfcc_statistics.csv (MFCC statistics by emotion)


MFCC mapping completed! Check 'mfcc_outputs/' directory for MFCC data.
```