Optimized CNN Model

An optimized Convolutional Neural Network (CNN) was built to improve accuracy significantly above 79% (targeting ~85–90%):

CNN Architecture (Optimized):

Conv2D (32 filters), BatchNormalization, MaxPooling2D, Dropout (0.3)

Conv2D (64 filters), BatchNormalization, MaxPooling2D, Dropout (0.4)

Conv2D (128 filters), BatchNormalization, MaxPooling2D, Dropout (0.4)

Conv2D (256 filters), BatchNormalization, MaxPooling2D, Dropout (0.4)

GlobalAveragePooling2D

Dense Layer (512 neurons), Dropout (0.5)

Dense Output Layer (10 neurons with Softmax activation)

Compilation & Training Parameters:

Optimizer: Adam (learning rate = 0.0001)

Loss: Sparse Categorical Crossentropy

Metrics: Accuracy

Callbacks: EarlyStopping (patience=20), ReduceLROnPlateau (factor=0.5)

```python
# Imports
import numpy as np
import matplotlib.pyplot as plt
import os
import json
import librosa

# Sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# TensorFlow
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential, save_model
from tensorflow.keras.layers import Dense, Dropout, Conv2D,
MaxPooling2D, Flatten, BatchNormalization
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
filepath = 'C:/Users/aksha/Downloads/DataMusic/data.json'
with open(filepath, "r") as fp:
    data = json.load(fp)

# Define X nd y
X = np.array(data["mfcc"])
y = np.array(data["genre_num"])

from sklearn.model_selection import train_test_split

# Split the data into train (80%), validation (10%), and test (10%)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.125, random_state=42, stratify=y_train)

# Shape of the splits after optimization
print(f"X training data shape: {X_train.shape}, y training data shape:
{y_train.shape}")
print(f"X validation data shape: {X_val.shape}, y validation data
shape: {y_val.shape}")
print(f"X test data shape: {X_test.shape}, y test data shape:
{y_test.shape}")
```

```
X training data shape: (6992, 132, 13), y training data shape: (6992,)
X validation data shape: (999, 132, 13), y validation data shape:
(999,)
X test data shape: (1998, 132, 13), y test data shape: (1998,)
```

```
# Ensure proper CNN reshaping:
X_train_cnn = X_train[..., np.newaxis] # (samples, 132, 13, 1)
X_val_cnn = X_val[..., np.newaxis]
X_test_cnn = X_test[..., np.newaxis]

input_shape = (132, 13, 1)

# Confirm final shape for CNN:
print("\nCNN input shapes:")
print(f"X_train_cnn: {X_train_cnn.shape}")
print(f"X_val_cnn: {X_val_cnn.shape}")
print(f"X_test_cnn: {X_test_cnn.shape}")
```

```

CNN input shapes:
X_train_cnn: (6992, 132, 13, 1)
X_val_cnn: (999, 132, 13, 1)
X_test_cnn: (1998, 132, 13, 1)
```

Build Model Here

Compilation & Training Parameters: Optimizer: Adam (learning rate = 0.0001) Loss: Sparse Categorical Crossentropy Metrics: Accuracy Callbacks: EarlyStopping (patience=20), ReduceLROnPlateau (factor=0.5)

Explanation of Activation Function Choice (ReLU):

In building our convolutional neural network (CNN), we chose the Rectified Linear Unit (ReLU) activation function for convolutional and dense layers. The ReLU function introduces non-linearity into the network, enabling it to learn complex mappings from the Mel-Frequency Cepstral Coefficients (MFCCs) inputs.

Alternative activation functions considered include:

Sigmoid produces outputs between 0 and 1. While useful for binary classification, it suffers from vanishing gradient issues in deep networks. Tanh outputs values between -1 and 1. It also has gradient problems and converges slower than ReLU. Leaky ReLU a variant of ReLU that mitigates the "dying ReLU" issue by allowing small negative gradients. Though it is beneficial, in some contexts, standard ReLU performs well without additional complexity. ReLU was chosen specifically because it is computationally efficient, speeds up convergence significantly, allowing for us to develop deeper architecture and build a more robust model.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
GlobalAveragePooling2D, Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau

# CNN Optimized (V3 - Better accuracy potential)
model_cnn_v3 = Sequential([
    Conv2D(64, (3, 3), activation='relu', input_shape=(132, 13, 1),
padding='same'),
    BatchNormalization(),
    MaxPooling2D((2, 2), padding='same'),
    Dropout(0.3),

    Conv2D(128, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D((2, 2), padding='same'),
    Dropout(0.4),

    Conv2D(256, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D((2, 2), padding='same'),
    Dropout(0.4),

    Conv2D(256, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D((2, 2), padding='same'),
    Dropout(0.4),

    GlobalAveragePooling2D(),

    Dense(512, activation='relu'),
    Dropout(0.5),

    Dense(10, activation='softmax')
])

# Compile the model
model_cnn_v3.compile(
    optimizer=Adam(learning_rate=1e-4),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# Model summary
model_cnn_v3.summary()
```

```
Model: "sequential_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ conv2d_8 (Conv2D)                │ (None, 132, 13, 64)    │
640 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ batch_normalization_8            │ (None, 132, 13, 64)    │
256 │
│ (BatchNormalization)             │                        │
│
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ max_pooling2d_8 (MaxPooling2D)   │ (None, 66, 7, 64)      │
0 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ dropout_9 (Dropout)              │ (None, 66, 7, 64)      │
0 │
├──────────────────────────────────┼────────────────────────┼──────────
─────┤
│ conv2d_9 (Conv2D)                │ (None, 66, 7, 128)     │
73,856 │
├──────────────────────────────────┼────────────────────────┼──────────
```

```
─────┤
│ batch_normalization_9           │ (None, 66, 7, 128)       │
512 │
│ (BatchNormalization)            │                          │
│
├─────────────────────────────────┼──────────────────────────┼───────────
─────┤
│ max_pooling2d_9 (MaxPooling2D)  │ (None, 33, 4, 128)       │
0 │
├─────────────────────────────────┼──────────────────────────┼───────────
─────┤
│ dropout_10 (Dropout)            │ (None, 33, 4, 128)       │
0 │
├─────────────────────────────────┼──────────────────────────┼───────────
─────┤
│ conv2d_10 (Conv2D)              │ (None, 33, 4, 256)       │
295,168 │
├─────────────────────────────────┼──────────────────────────┼───────────
─────┤
│ batch_normalization_10          │ (None, 33, 4, 256)       │
1,024 │
│ (BatchNormalization)            │                          │
│
├─────────────────────────────────┼──────────────────────────┼───────────
─────┤
│ max_pooling2d_10 (MaxPooling2D) │ (None, 17, 2, 256)       │
0 │
├─────────────────────────────────┼──────────────────────────┼───────────
─────┤
│ dropout_11 (Dropout)            │ (None, 17, 2, 256)       │
0 │
├─────────────────────────────────┼──────────────────────────┼───────────
─────┤
│ conv2d_11 (Conv2D)              │ (None, 17, 2, 256)       │
590,080 │
├─────────────────────────────────┼──────────────────────────┼───────────
─────┤
│ batch_normalization_11          │ (None, 17, 2, 256)       │
1,024 │
│ (BatchNormalization)            │                          │
│
├─────────────────────────────────┼──────────────────────────┼───────────
─────┤
│ max_pooling2d_11 (MaxPooling2D) │ (None, 9, 1, 256)        │
0 │
├─────────────────────────────────┼──────────────────────────┼───────────
─────┤
│ dropout_12 (Dropout)            │ (None, 9, 1, 256)        │
0 │
```

```
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
│ global_average_pooling2d_1       │ (None, 256)             │
0 │
│ (GlobalAveragePooling2D)         │                         │
│
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
│ dense_2 (Dense)                  │ (None, 512)             │
131,584 │
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
│ dropout_13 (Dropout)             │ (None, 512)             │
0 │
├──────────────────────────────────┼─────────────────────────┼──────────
──────┤
│ dense_3 (Dense)                  │ (None, 10)              │
5,130 │
└──────────────────────────────────┴─────────────────────────┴──────────
──────┘

 Total params: 1,099,274 (4.19 MB)

 Trainable params: 1,097,866 (4.19 MB)

 Non-trainable params: 1,408 (5.50 KB)
```

Compile method here (Runtime Approximately 45-60 minutes)

Can reduce number of Epochs for faster passthrough the

```
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau

early_stopping = EarlyStopping(monitor='val_loss', patience=25,
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=10, min_lr=1e-6, verbose=1)

history = model_cnn_v3.fit(
    X_train_cnn, y_train,
    validation_data=(X_val_cnn, y_val),
    epochs=150,
    batch_size=32,
    callbacks=[early_stopping, reduce_lr],
    verbose=1
)
```

```
Epoch 1/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 28s 112ms/step - accuracy: 0.2691 - loss:
2.1387 - val_accuracy: 0.3443 - val_loss: 1.8008 - learning_rate:
1.0000e-04
Epoch 2/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 108ms/step - accuracy: 0.4413 - loss:
1.5406 - val_accuracy: 0.3784 - val_loss: 1.8266 - learning_rate:
1.0000e-04
Epoch 3/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 112ms/step - accuracy: 0.4992 - loss:
1.3531 - val_accuracy: 0.3834 - val_loss: 1.8520 - learning_rate:
1.0000e-04
Epoch 4/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 110ms/step - accuracy: 0.5499 - loss:
1.2345 - val_accuracy: 0.4034 - val_loss: 1.9790 - learning_rate:
1.0000e-04
Epoch 5/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 112ms/step - accuracy: 0.5826 - loss:
1.1715 - val_accuracy: 0.4685 - val_loss: 1.6869 - learning_rate:
1.0000e-04
Epoch 6/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 112ms/step - accuracy: 0.6220 - loss:
1.0819 - val_accuracy: 0.5205 - val_loss: 1.5178 - learning_rate:
1.0000e-04
Epoch 7/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 113ms/step - accuracy: 0.6251 - loss:
1.0411 - val_accuracy: 0.5365 - val_loss: 1.4989 - learning_rate:
1.0000e-04
Epoch 8/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 115ms/step - accuracy: 0.6511 - loss:
0.9955 - val_accuracy: 0.5475 - val_loss: 1.4741 - learning_rate:
1.0000e-04
Epoch 9/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 118ms/step - accuracy: 0.6634 - loss:
0.9457 - val_accuracy: 0.5906 - val_loss: 1.2644 - learning_rate:
1.0000e-04
Epoch 10/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 29s 131ms/step - accuracy: 0.6843 - loss:
0.8978 - val_accuracy: 0.6276 - val_loss: 1.2109 - learning_rate:
1.0000e-04
Epoch 11/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 27s 124ms/step - accuracy: 0.6914 - loss:
0.8769 - val_accuracy: 0.6356 - val_loss: 1.1417 - learning_rate:
1.0000e-04
Epoch 12/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 27s 123ms/step - accuracy: 0.7052 - loss:
0.8382 - val_accuracy: 0.6216 - val_loss: 1.2768 - learning_rate:
1.0000e-04
Epoch 13/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 120ms/step - accuracy: 0.7170 - loss:
```

```
0.8220 - val_accuracy: 0.6406 - val_loss: 1.1876 - learning_rate:
1.0000e-04
Epoch 14/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 27s 124ms/step - accuracy: 0.7389 - loss:
0.7533 - val_accuracy: 0.6737 - val_loss: 1.0284 - learning_rate:
1.0000e-04
Epoch 15/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 28s 130ms/step - accuracy: 0.7463 - loss:
0.7277 - val_accuracy: 0.6957 - val_loss: 0.9591 - learning_rate:
1.0000e-04
Epoch 16/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 28s 128ms/step - accuracy: 0.7450 - loss:
0.7304 - val_accuracy: 0.6997 - val_loss: 0.9636 - learning_rate:
1.0000e-04
Epoch 17/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 27s 124ms/step - accuracy: 0.7598 - loss:
0.6966 - val_accuracy: 0.6937 - val_loss: 0.9436 - learning_rate:
1.0000e-04
Epoch 18/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 121ms/step - accuracy: 0.7614 - loss:
0.6719 - val_accuracy: 0.7487 - val_loss: 0.7666 - learning_rate:
1.0000e-04
Epoch 19/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 31s 140ms/step - accuracy: 0.7783 - loss:
0.6461 - val_accuracy: 0.7307 - val_loss: 0.8049 - learning_rate:
1.0000e-04
Epoch 20/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 28s 130ms/step - accuracy: 0.7834 - loss:
0.6280 - val_accuracy: 0.7497 - val_loss: 0.7756 - learning_rate:
1.0000e-04
Epoch 21/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 27s 123ms/step - accuracy: 0.7905 - loss:
0.6033 - val_accuracy: 0.7698 - val_loss: 0.6983 - learning_rate:
1.0000e-04
Epoch 22/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 116ms/step - accuracy: 0.8028 - loss:
0.5730 - val_accuracy: 0.7407 - val_loss: 0.8700 - learning_rate:
1.0000e-04
Epoch 23/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 115ms/step - accuracy: 0.8114 - loss:
0.5560 - val_accuracy: 0.7658 - val_loss: 0.7386 - learning_rate:
1.0000e-04
Epoch 24/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 116ms/step - accuracy: 0.8075 - loss:
0.5539 - val_accuracy: 0.7718 - val_loss: 0.7258 - learning_rate:
1.0000e-04
Epoch 25/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 107ms/step - accuracy: 0.8173 - loss:
0.5174 - val_accuracy: 0.8168 - val_loss: 0.5726 - learning_rate:
1.0000e-04
```

```
Epoch 26/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 110ms/step - accuracy: 0.8113 - loss:
0.5425 - val_accuracy: 0.8148 - val_loss: 0.5822 - learning_rate:
1.0000e-04
Epoch 27/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 23s 107ms/step - accuracy: 0.8252 - loss:
0.5084 - val_accuracy: 0.7898 - val_loss: 0.6545 - learning_rate:
1.0000e-04
Epoch 28/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 23s 107ms/step - accuracy: 0.8304 - loss:
0.4922 - val_accuracy: 0.8238 - val_loss: 0.5860 - learning_rate:
1.0000e-04
Epoch 29/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 23s 106ms/step - accuracy: 0.8404 - loss:
0.4654 - val_accuracy: 0.8148 - val_loss: 0.5887 - learning_rate:
1.0000e-04
Epoch 30/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 23s 106ms/step - accuracy: 0.8289 - loss:
0.4713 - val_accuracy: 0.8008 - val_loss: 0.6115 - learning_rate:
1.0000e-04
Epoch 31/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 23s 106ms/step - accuracy: 0.8312 - loss:
0.4584 - val_accuracy: 0.8138 - val_loss: 0.5936 - learning_rate:
1.0000e-04
Epoch 32/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 23s 107ms/step - accuracy: 0.8460 - loss:
0.4517 - val_accuracy: 0.8338 - val_loss: 0.5508 - learning_rate:
1.0000e-04
Epoch 33/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 108ms/step - accuracy: 0.8541 - loss:
0.4291 - val_accuracy: 0.8208 - val_loss: 0.5692 - learning_rate:
1.0000e-04
Epoch 34/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 23s 107ms/step - accuracy: 0.8558 - loss:
0.4070 - val_accuracy: 0.8388 - val_loss: 0.5179 - learning_rate:
1.0000e-04
Epoch 35/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 108ms/step - accuracy: 0.8638 - loss:
0.3925 - val_accuracy: 0.8448 - val_loss: 0.5211 - learning_rate:
1.0000e-04
Epoch 36/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 108ms/step - accuracy: 0.8688 - loss:
0.3822 - val_accuracy: 0.8509 - val_loss: 0.4967 - learning_rate:
1.0000e-04
Epoch 37/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 107ms/step - accuracy: 0.8705 - loss:
0.3708 - val_accuracy: 0.8468 - val_loss: 0.5132 - learning_rate:
1.0000e-04
Epoch 38/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 108ms/step - accuracy: 0.8741 - loss:
```

```
0.3475 - val_accuracy: 0.8719 - val_loss: 0.4309 - learning_rate:
1.0000e-04
Epoch 39/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 108ms/step - accuracy: 0.8699 - loss:
0.3641 - val_accuracy: 0.8519 - val_loss: 0.4635 - learning_rate:
1.0000e-04
Epoch 40/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 23s 105ms/step - accuracy: 0.8724 - loss:
0.3492 - val_accuracy: 0.8579 - val_loss: 0.4187 - learning_rate:
1.0000e-04
Epoch 41/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 108ms/step - accuracy: 0.8797 - loss:
0.3425 - val_accuracy: 0.8699 - val_loss: 0.4256 - learning_rate:
1.0000e-04
Epoch 42/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 23s 106ms/step - accuracy: 0.8828 - loss:
0.3345 - val_accuracy: 0.8789 - val_loss: 0.4052 - learning_rate:
1.0000e-04
Epoch 43/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 23s 103ms/step - accuracy: 0.8922 - loss:
0.3160 - val_accuracy: 0.8729 - val_loss: 0.4128 - learning_rate:
1.0000e-04
Epoch 44/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 108ms/step - accuracy: 0.8947 - loss:
0.3103 - val_accuracy: 0.8829 - val_loss: 0.3703 - learning_rate:
1.0000e-04
Epoch 45/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 23s 105ms/step - accuracy: 0.8877 - loss:
0.3098 - val_accuracy: 0.8799 - val_loss: 0.4008 - learning_rate:
1.0000e-04
Epoch 46/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 23s 105ms/step - accuracy: 0.8920 - loss:
0.3042 - val_accuracy: 0.8809 - val_loss: 0.3567 - learning_rate:
1.0000e-04
Epoch 47/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 23s 105ms/step - accuracy: 0.9029 - loss:
0.2711 - val_accuracy: 0.8839 - val_loss: 0.3815 - learning_rate:
1.0000e-04
Epoch 48/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 23s 103ms/step - accuracy: 0.9046 - loss:
0.2755 - val_accuracy: 0.8789 - val_loss: 0.3572 - learning_rate:
1.0000e-04
Epoch 49/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 22s 101ms/step - accuracy: 0.9023 - loss:
0.2687 - val_accuracy: 0.8879 - val_loss: 0.3361 - learning_rate:
1.0000e-04
Epoch 50/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 22s 100ms/step - accuracy: 0.9119 - loss:
0.2485 - val_accuracy: 0.8769 - val_loss: 0.4006 - learning_rate:
1.0000e-04
```

```
Epoch 51/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 22s 99ms/step - accuracy: 0.9090 - loss: 
0.2689 - val_accuracy: 0.8959 - val_loss: 0.3259 - learning_rate: 
1.0000e-04
Epoch 52/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 22s 101ms/step - accuracy: 0.9177 - loss: 
0.2334 - val_accuracy: 0.8859 - val_loss: 0.3623 - learning_rate: 
1.0000e-04
Epoch 53/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 22s 101ms/step - accuracy: 0.9165 - loss: 
0.2409 - val_accuracy: 0.8799 - val_loss: 0.3688 - learning_rate: 
1.0000e-04
Epoch 54/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 22s 100ms/step - accuracy: 0.9109 - loss: 
0.2526 - val_accuracy: 0.8929 - val_loss: 0.3231 - learning_rate: 
1.0000e-04
Epoch 55/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 22s 100ms/step - accuracy: 0.9245 - loss: 
0.2252 - val_accuracy: 0.8899 - val_loss: 0.3535 - learning_rate: 
1.0000e-04
Epoch 56/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 22s 101ms/step - accuracy: 0.9160 - loss: 
0.2311 - val_accuracy: 0.8819 - val_loss: 0.3893 - learning_rate: 
1.0000e-04
Epoch 57/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 22s 101ms/step - accuracy: 0.9251 - loss: 
0.2196 - val_accuracy: 0.9069 - val_loss: 0.3216 - learning_rate: 
1.0000e-04
Epoch 58/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 22s 100ms/step - accuracy: 0.9268 - loss: 
0.2138 - val_accuracy: 0.9089 - val_loss: 0.2998 - learning_rate: 
1.0000e-04
Epoch 59/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 22s 102ms/step - accuracy: 0.9278 - loss: 
0.2027 - val_accuracy: 0.9089 - val_loss: 0.2836 - learning_rate: 
1.0000e-04
Epoch 60/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 22s 101ms/step - accuracy: 0.9224 - loss: 
0.2047 - val_accuracy: 0.9069 - val_loss: 0.2979 - learning_rate: 
1.0000e-04
Epoch 61/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 114ms/step - accuracy: 0.9215 - loss: 
0.2139 - val_accuracy: 0.9139 - val_loss: 0.2815 - learning_rate: 
1.0000e-04
Epoch 62/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 108ms/step - accuracy: 0.9365 - loss: 
0.1818 - val_accuracy: 0.9169 - val_loss: 0.2847 - learning_rate: 
1.0000e-04
Epoch 63/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 113ms/step - accuracy: 0.9375 - loss: 
```

```
0.1823 - val_accuracy: 0.9159 - val_loss: 0.2730 - learning_rate:
1.0000e-04
Epoch 64/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 108ms/step - accuracy: 0.9378 - loss:
0.1710 - val_accuracy: 0.9079 - val_loss: 0.3043 - learning_rate:
1.0000e-04
Epoch 65/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 119ms/step - accuracy: 0.9396 - loss:
0.1755 - val_accuracy: 0.9189 - val_loss: 0.2594 - learning_rate:
1.0000e-04
Epoch 66/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 112ms/step - accuracy: 0.9368 - loss:
0.1802 - val_accuracy: 0.9199 - val_loss: 0.2564 - learning_rate:
1.0000e-04
Epoch 67/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 110ms/step - accuracy: 0.9390 - loss:
0.1714 - val_accuracy: 0.9039 - val_loss: 0.3142 - learning_rate:
1.0000e-04
Epoch 68/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 113ms/step - accuracy: 0.9339 - loss:
0.1803 - val_accuracy: 0.9199 - val_loss: 0.2570 - learning_rate:
1.0000e-04
Epoch 69/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 110ms/step - accuracy: 0.9457 - loss:
0.1673 - val_accuracy: 0.9169 - val_loss: 0.2594 - learning_rate:
1.0000e-04
Epoch 70/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 110ms/step - accuracy: 0.9394 - loss:
0.1601 - val_accuracy: 0.9189 - val_loss: 0.2339 - learning_rate:
1.0000e-04
Epoch 71/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 110ms/step - accuracy: 0.9452 - loss:
0.1517 - val_accuracy: 0.9299 - val_loss: 0.2480 - learning_rate:
1.0000e-04
Epoch 72/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 113ms/step - accuracy: 0.9432 - loss:
0.1539 - val_accuracy: 0.9109 - val_loss: 0.2726 - learning_rate:
1.0000e-04
Epoch 73/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 109ms/step - accuracy: 0.9495 - loss:
0.1499 - val_accuracy: 0.9149 - val_loss: 0.2318 - learning_rate:
1.0000e-04
Epoch 74/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 113ms/step - accuracy: 0.9526 - loss:
0.1410 - val_accuracy: 0.9159 - val_loss: 0.2345 - learning_rate:
1.0000e-04
Epoch 75/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 111ms/step - accuracy: 0.9528 - loss:
0.1418 - val_accuracy: 0.9159 - val_loss: 0.2388 - learning_rate:
1.0000e-04
```

```
Epoch 76/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 113ms/step - accuracy: 0.9463 - loss:
0.1472 - val_accuracy: 0.9179 - val_loss: 0.2401 - learning_rate:
1.0000e-04
Epoch 77/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 111ms/step - accuracy: 0.9492 - loss:
0.1422 - val_accuracy: 0.9119 - val_loss: 0.2470 - learning_rate:
1.0000e-04
Epoch 78/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 111ms/step - accuracy: 0.9552 - loss:
0.1259 - val_accuracy: 0.9169 - val_loss: 0.2472 - learning_rate:
1.0000e-04
Epoch 79/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 113ms/step - accuracy: 0.9558 - loss:
0.1217 - val_accuracy: 0.9209 - val_loss: 0.2525 - learning_rate:
1.0000e-04
Epoch 80/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 111ms/step - accuracy: 0.9563 - loss:
0.1288 - val_accuracy: 0.9219 - val_loss: 0.2491 - learning_rate:
1.0000e-04
Epoch 81/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 112ms/step - accuracy: 0.9617 - loss:
0.1193 - val_accuracy: 0.9219 - val_loss: 0.2491 - learning_rate:
1.0000e-04
Epoch 82/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 118ms/step - accuracy: 0.9612 - loss:
0.1175 - val_accuracy: 0.9249 - val_loss: 0.2534 - learning_rate:
1.0000e-04
Epoch 83/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step - accuracy: 0.9564 - loss:
0.1236
Epoch 83: ReduceLROnPlateau reducing learning rate to
4.999999873689376e-05.
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 112ms/step - accuracy: 0.9564 - loss:
0.1235 - val_accuracy: 0.9219 - val_loss: 0.2351 - learning_rate:
1.0000e-04
Epoch 84/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 27s 121ms/step - accuracy: 0.9635 - loss:
0.1133 - val_accuracy: 0.9269 - val_loss: 0.2205 - learning_rate:
5.0000e-05
Epoch 85/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 28s 128ms/step - accuracy: 0.9639 - loss:
0.1040 - val_accuracy: 0.9279 - val_loss: 0.2272 - learning_rate:
5.0000e-05
Epoch 86/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 28s 128ms/step - accuracy: 0.9668 - loss:
0.0904 - val_accuracy: 0.9329 - val_loss: 0.2162 - learning_rate:
5.0000e-05
Epoch 87/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 119ms/step - accuracy: 0.9714 - loss:
```

```
0.0901 - val_accuracy: 0.9399 - val_loss: 0.1928 - learning_rate:
5.0000e-05
Epoch 88/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 27s 124ms/step - accuracy: 0.9706 - loss:
0.0923 - val_accuracy: 0.9319 - val_loss: 0.2205 - learning_rate:
5.0000e-05
Epoch 89/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 27s 123ms/step - accuracy: 0.9690 - loss:
0.0845 - val_accuracy: 0.9339 - val_loss: 0.2146 - learning_rate:
5.0000e-05
Epoch 90/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 29s 131ms/step - accuracy: 0.9677 - loss:
0.0911 - val_accuracy: 0.9319 - val_loss: 0.2180 - learning_rate:
5.0000e-05
Epoch 91/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 29s 134ms/step - accuracy: 0.9627 - loss:
0.0944 - val_accuracy: 0.9289 - val_loss: 0.2112 - learning_rate:
5.0000e-05
Epoch 92/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 28s 127ms/step - accuracy: 0.9675 - loss:
0.0897 - val_accuracy: 0.9269 - val_loss: 0.2093 - learning_rate:
5.0000e-05
Epoch 93/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 119ms/step - accuracy: 0.9718 - loss:
0.0815 - val_accuracy: 0.9289 - val_loss: 0.2289 - learning_rate:
5.0000e-05
Epoch 94/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 118ms/step - accuracy: 0.9701 - loss:
0.0854 - val_accuracy: 0.9319 - val_loss: 0.2218 - learning_rate:
5.0000e-05
Epoch 95/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 114ms/step - accuracy: 0.9693 - loss:
0.0808 - val_accuracy: 0.9319 - val_loss: 0.2215 - learning_rate:
5.0000e-05
Epoch 96/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 118ms/step - accuracy: 0.9717 - loss:
0.0757 - val_accuracy: 0.9349 - val_loss: 0.2102 - learning_rate:
5.0000e-05
Epoch 97/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step - accuracy: 0.9703 - loss:
0.0852
Epoch 97: ReduceLROnPlateau reducing learning rate to
2.499999936844688e-05.
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 114ms/step - accuracy: 0.9703 - loss:
0.0852 - val_accuracy: 0.9339 - val_loss: 0.2198 - learning_rate:
5.0000e-05
Epoch 98/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 112ms/step - accuracy: 0.9736 - loss:
0.0818 - val_accuracy: 0.9329 - val_loss: 0.2081 - learning_rate:
2.5000e-05
```

```
Epoch 99/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 24s 112ms/step - accuracy: 0.9704 - loss:
0.0806 - val_accuracy: 0.9329 - val_loss: 0.2089 - learning_rate:
2.5000e-05
Epoch 100/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 112ms/step - accuracy: 0.9758 - loss:
0.0730 - val_accuracy: 0.9339 - val_loss: 0.1996 - learning_rate:
2.5000e-05
Epoch 101/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 118ms/step - accuracy: 0.9745 - loss:
0.0749 - val_accuracy: 0.9329 - val_loss: 0.2065 - learning_rate:
2.5000e-05
Epoch 102/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 27s 123ms/step - accuracy: 0.9749 - loss:
0.0707 - val_accuracy: 0.9329 - val_loss: 0.2046 - learning_rate:
2.5000e-05
Epoch 103/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 116ms/step - accuracy: 0.9738 - loss:
0.0772 - val_accuracy: 0.9349 - val_loss: 0.2048 - learning_rate:
2.5000e-05
Epoch 104/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 113ms/step - accuracy: 0.9723 - loss:
0.0795 - val_accuracy: 0.9319 - val_loss: 0.2076 - learning_rate:
2.5000e-05
Epoch 105/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 27s 123ms/step - accuracy: 0.9789 - loss:
0.0690 - val_accuracy: 0.9349 - val_loss: 0.2074 - learning_rate:
2.5000e-05
Epoch 106/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 117ms/step - accuracy: 0.9775 - loss:
0.0646 - val_accuracy: 0.9349 - val_loss: 0.2074 - learning_rate:
2.5000e-05
Epoch 107/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step - accuracy: 0.9746 - loss:
0.0726
Epoch 107: ReduceLROnPlateau reducing learning rate to
1.249999968422344e-05.
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 113ms/step - accuracy: 0.9746 - loss:
0.0726 - val_accuracy: 0.9339 - val_loss: 0.2062 - learning_rate:
2.5000e-05
Epoch 108/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 116ms/step - accuracy: 0.9787 - loss:
0.0648 - val_accuracy: 0.9349 - val_loss: 0.1937 - learning_rate:
1.2500e-05
Epoch 109/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 115ms/step - accuracy: 0.9774 - loss:
0.0684 - val_accuracy: 0.9349 - val_loss: 0.2017 - learning_rate:
1.2500e-05
Epoch 110/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 116ms/step - accuracy: 0.9783 - loss:
```

```
0.0631 - val_accuracy: 0.9389 - val_loss: 0.1918 - learning_rate:
1.2500e-05
Epoch 111/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 118ms/step - accuracy: 0.9771 - loss:
0.0664 - val_accuracy: 0.9429 - val_loss: 0.1961 - learning_rate:
1.2500e-05
Epoch 112/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 27s 123ms/step - accuracy: 0.9758 - loss:
0.0720 - val_accuracy: 0.9419 - val_loss: 0.1910 - learning_rate:
1.2500e-05
Epoch 113/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 117ms/step - accuracy: 0.9809 - loss:
0.0580 - val_accuracy: 0.9369 - val_loss: 0.2003 - learning_rate:
1.2500e-05
Epoch 114/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 117ms/step - accuracy: 0.9798 - loss:
0.0602 - val_accuracy: 0.9369 - val_loss: 0.1976 - learning_rate:
1.2500e-05
Epoch 115/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 114ms/step - accuracy: 0.9824 - loss:
0.0541 - val_accuracy: 0.9369 - val_loss: 0.1991 - learning_rate:
1.2500e-05
Epoch 116/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 114ms/step - accuracy: 0.9754 - loss:
0.0656 - val_accuracy: 0.9399 - val_loss: 0.1876 - learning_rate:
1.2500e-05
Epoch 117/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 118ms/step - accuracy: 0.9785 - loss:
0.0676 - val_accuracy: 0.9389 - val_loss: 0.1903 - learning_rate:
1.2500e-05
Epoch 118/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 114ms/step - accuracy: 0.9803 - loss:
0.0604 - val_accuracy: 0.9379 - val_loss: 0.1941 - learning_rate:
1.2500e-05
Epoch 119/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 119ms/step - accuracy: 0.9794 - loss:
0.0622 - val_accuracy: 0.9399 - val_loss: 0.1945 - learning_rate:
1.2500e-05
Epoch 120/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 116ms/step - accuracy: 0.9789 - loss:
0.0544 - val_accuracy: 0.9369 - val_loss: 0.1916 - learning_rate:
1.2500e-05
Epoch 121/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 112ms/step - accuracy: 0.9766 - loss:
0.0701 - val_accuracy: 0.9389 - val_loss: 0.2005 - learning_rate:
1.2500e-05
Epoch 122/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 117ms/step - accuracy: 0.9816 - loss:
0.0599 - val_accuracy: 0.9389 - val_loss: 0.1919 - learning_rate:
1.2500e-05
```

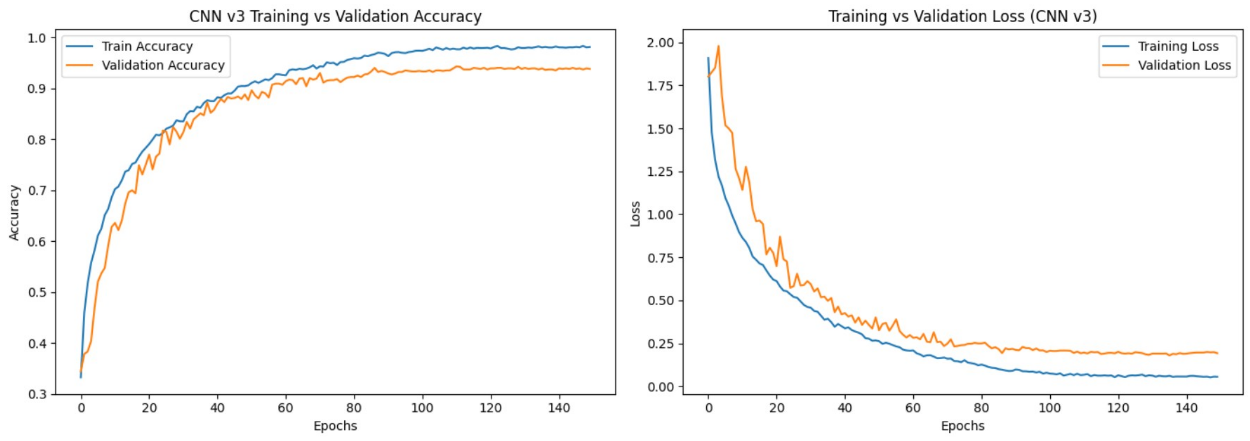
```
Epoch 123/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 120ms/step - accuracy: 0.9820 - loss:
0.0510 - val_accuracy: 0.9399 - val_loss: 0.1898 - learning_rate:
1.2500e-05
Epoch 124/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 118ms/step - accuracy: 0.9768 - loss:
0.0619 - val_accuracy: 0.9399 - val_loss: 0.1921 - learning_rate:
1.2500e-05
Epoch 125/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 113ms/step - accuracy: 0.9722 - loss:
0.0798 - val_accuracy: 0.9379 - val_loss: 0.1893 - learning_rate:
1.2500e-05
Epoch 126/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 119ms/step - accuracy: 0.9795 - loss:
0.0568
Epoch 126: ReduceLROnPlateau reducing learning rate to
6.24999984211172e-06.
219/219 ━━━━━━━━━━━━━━━━━━━━ 27s 123ms/step - accuracy: 0.9795 - loss:
0.0569 - val_accuracy: 0.9389 - val_loss: 0.1981 - learning_rate:
1.2500e-05
Epoch 127/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 114ms/step - accuracy: 0.9773 - loss:
0.0602 - val_accuracy: 0.9389 - val_loss: 0.1958 - learning_rate:
6.2500e-06
Epoch 128/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 119ms/step - accuracy: 0.9755 - loss:
0.0733 - val_accuracy: 0.9379 - val_loss: 0.1929 - learning_rate:
6.2500e-06
Epoch 129/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 113ms/step - accuracy: 0.9801 - loss:
0.0577 - val_accuracy: 0.9419 - val_loss: 0.1858 - learning_rate:
6.2500e-06
Epoch 130/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 117ms/step - accuracy: 0.9793 - loss:
0.0634 - val_accuracy: 0.9379 - val_loss: 0.1828 - learning_rate:
6.2500e-06
Epoch 131/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 115ms/step - accuracy: 0.9806 - loss:
0.0617 - val_accuracy: 0.9399 - val_loss: 0.1900 - learning_rate:
6.2500e-06
Epoch 132/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 116ms/step - accuracy: 0.9792 - loss:
0.0528 - val_accuracy: 0.9379 - val_loss: 0.1902 - learning_rate:
6.2500e-06
Epoch 133/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 27s 123ms/step - accuracy: 0.9777 - loss:
0.0615 - val_accuracy: 0.9389 - val_loss: 0.1899 - learning_rate:
6.2500e-06
Epoch 134/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 114ms/step - accuracy: 0.9812 - loss:
```

```
0.0559 - val_accuracy: 0.9389 - val_loss: 0.1895 - learning_rate:
6.2500e-06
Epoch 135/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 114ms/step - accuracy: 0.9821 - loss:
0.0565 - val_accuracy: 0.9369 - val_loss: 0.1909 - learning_rate:
6.2500e-06
Epoch 136/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 27s 121ms/step - accuracy: 0.9794 - loss:
0.0614 - val_accuracy: 0.9389 - val_loss: 0.1793 - learning_rate:
6.2500e-06
Epoch 137/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 115ms/step - accuracy: 0.9805 - loss:
0.0546 - val_accuracy: 0.9359 - val_loss: 0.1886 - learning_rate:
6.2500e-06
Epoch 138/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 115ms/step - accuracy: 0.9810 - loss:
0.0547 - val_accuracy: 0.9369 - val_loss: 0.1864 - learning_rate:
6.2500e-06
Epoch 139/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 117ms/step - accuracy: 0.9798 - loss:
0.0581 - val_accuracy: 0.9369 - val_loss: 0.1929 - learning_rate:
6.2500e-06
Epoch 140/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 114ms/step - accuracy: 0.9824 - loss:
0.0575 - val_accuracy: 0.9349 - val_loss: 0.1886 - learning_rate:
6.2500e-06
Epoch 141/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 115ms/step - accuracy: 0.9801 - loss:
0.0587 - val_accuracy: 0.9389 - val_loss: 0.1895 - learning_rate:
6.2500e-06
Epoch 142/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 118ms/step - accuracy: 0.9833 - loss:
0.0523 - val_accuracy: 0.9379 - val_loss: 0.1928 - learning_rate:
6.2500e-06
Epoch 143/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 116ms/step - accuracy: 0.9777 - loss:
0.0668 - val_accuracy: 0.9389 - val_loss: 0.1948 - learning_rate:
6.2500e-06
Epoch 144/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 27s 121ms/step - accuracy: 0.9788 - loss:
0.0568 - val_accuracy: 0.9379 - val_loss: 0.1961 - learning_rate:
6.2500e-06
Epoch 145/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 118ms/step - accuracy: 0.9781 - loss:
0.0568 - val_accuracy: 0.9399 - val_loss: 0.1963 - learning_rate:
6.2500e-06
Epoch 146/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 0s 110ms/step - accuracy: 0.9849 - loss:
0.0449
```

```
Epoch 146: ReduceLROnPlateau reducing learning rate to
3.12499992105586e-06.
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 114ms/step - accuracy: 0.9849 - loss:
0.0449 - val_accuracy: 0.9379 - val_loss: 0.1961 - learning_rate:
6.2500e-06
Epoch 147/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 28s 129ms/step - accuracy: 0.9797 - loss:
0.0601 - val_accuracy: 0.9389 - val_loss: 0.1995 - learning_rate:
3.1250e-06
Epoch 148/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 116ms/step - accuracy: 0.9834 - loss:
0.0505 - val_accuracy: 0.9369 - val_loss: 0.1973 - learning_rate:
3.1250e-06
Epoch 149/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 26s 118ms/step - accuracy: 0.9826 - loss:
0.0542 - val_accuracy: 0.9389 - val_loss: 0.1985 - learning_rate:
3.1250e-06
Epoch 150/150
219/219 ━━━━━━━━━━━━━━━━━━━━ 25s 115ms/step - accuracy: 0.9782 - loss:
0.0599 - val_accuracy: 0.9379 - val_loss: 0.1925 - learning_rate:
3.1250e-06
```

Uncomment this code for per-epoch training and validation accuracy,

```
# import matplotlib.pyplot as plt

# # Plot accuracy
# plt.figure(figsize=(14, 5))
# plt.subplot(1, 2, 1)
# plt.plot(history.history['accuracy'], label='Train Accuracy')
# plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
# plt.title('CNN v3 Training vs Validation Accuracy')
# plt.xlabel('Epochs')
# plt.ylabel('Accuracy')
# plt.legend()

# # Plot loss
# plt.subplot(1, 2, 2)
# plt.plot(history.history['loss'], label='Training Loss')
# plt.plot(history.history['val_loss'], label='Validation Loss')
# plt.title('Training vs Validation Loss (CNN v3)')
# plt.xlabel('Epochs')
# plt.ylabel('Loss')
# plt.legend()

# plt.tight_layout()
# plt.show()
# # Evaluate final accuracy on test set
# test_loss, test_acc = model_cnn_v3.evaluate(X_test_cnn, y_test,
verbose=2)
# print(f"\nCNN v3 Test Accuracy: {test_acc:.4f}")
```



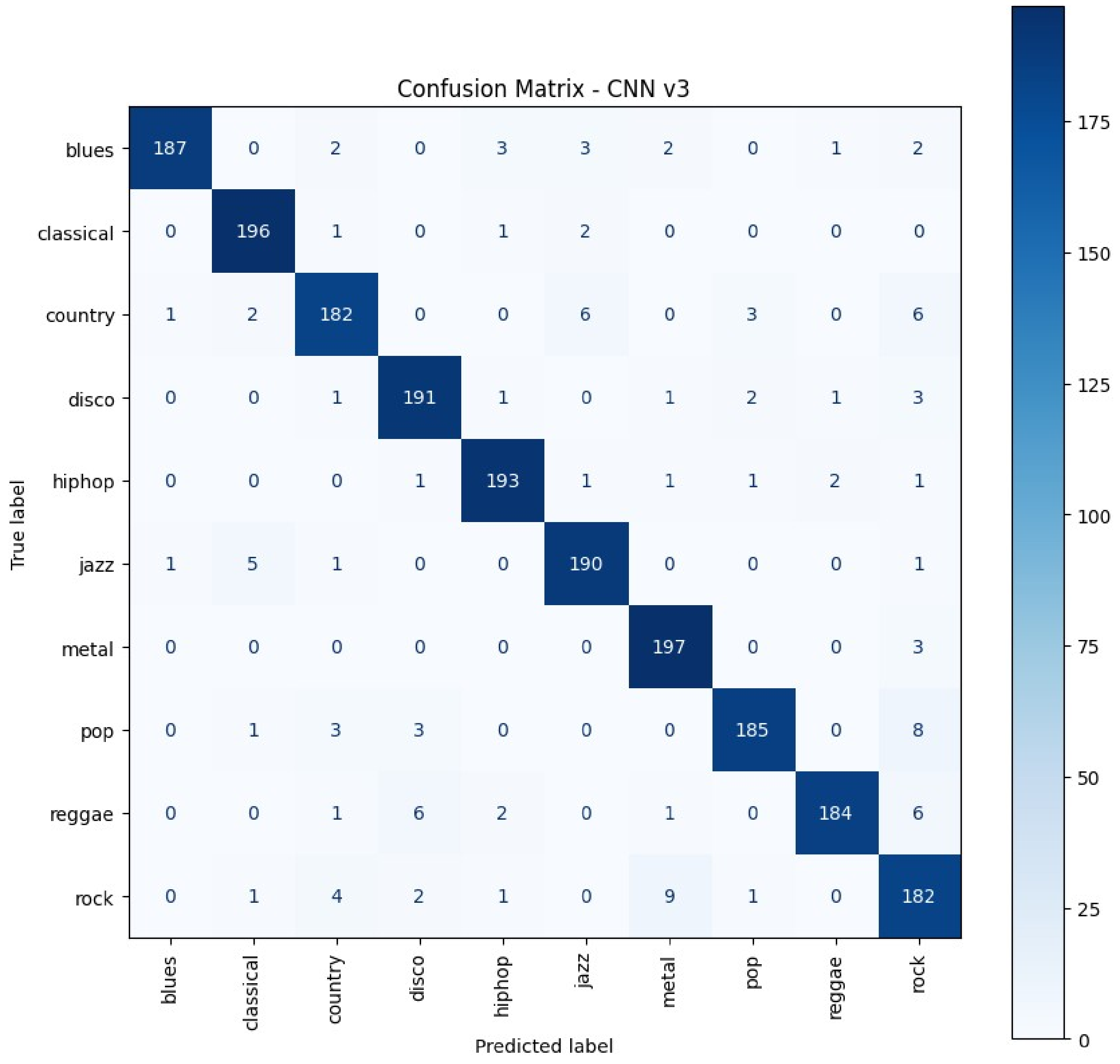
```
63/63 - 2s - 35ms/step - accuracy: 0.9444 - loss: 0.2026

CNN v3 Test Accuracy: 0.9444
```

Model Saved and loaded

Uncomment to save after running model

```
# from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# # Save your model
# model_cnn_v3.save('cnn_music_genre_model_v3.h5')
# print("□ Model saved as 'cnn_music_genre_model_v3.h5'")
# genres = ['blues', 'classical', 'country', 'disco', 'hiphop',
#           'jazz', 'metal', 'pop', 'reggae', 'rock']
# # Prediction Function
# def make_prediction_v3(model, X):
#     preds_num = []
#     preds_name = []
#     for X_current in X:
#         X_current = X_current[np.newaxis, ...]  # Add batch
dimension
#         pred = model.predict(X_current, verbose=0)
#         pred_idx = np.argmax(pred, axis=1)[0]  # Predicted genre
index
#         preds_num.append(pred_idx)
#         preds_name.append(genres[pred_idx])
#     return preds_num, preds_name
# # Make predictions
# preds_num, preds_name = make_prediction_v3(model_cnn_v3, X_test_cnn)
# print("□ Predictions completed!")
# # Plot confusion matrix
# cm = confusion_matrix(y_test, preds_num)
# disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=genres)
# fig, ax = plt.subplots(figsize=(10,10))
# disp.plot(ax=ax, cmap='Blues', xticks_rotation='vertical')
# plt.title('Confusion Matrix - CNN v3')
# plt.show()

from tensorflow.keras.models import load_model
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Load the previously saved model correctly
model_cnn_v3 = load_model('cnn_music_genre_model_v3.h5')
print("□ Model loaded successfully from
'cnn_music_genre_model_v3.h5'")

# Define genre labels
genres = ['blues', 'classical', 'country', 'disco', 'hiphop',
          'jazz', 'metal', 'pop', 'reggae', 'rock']

# Prediction function (optimized for faster inference)
def make_prediction_v3(model, X):
    preds = model.predict(X, verbose=0)
    preds_num = np.argmax(preds, axis=1)
    preds_name = [genres[i] for i in preds_num]
    return preds_num, preds_name

# Run predictions on the test dataset
preds_num, preds_name = make_prediction_v3(model_cnn_v3, X_test_cnn)
print("□ Predictions completed successfully!")

# Compute and display confusion matrix
cm = confusion_matrix(y_test, preds_num)
fig, ax = plt.subplots(figsize=(10,10))
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=genres)
disp.plot(ax=ax, cmap='Blues', xticks_rotation='vertical')
plt.title('Confusion Matrix - CNN v3')
plt.show()

# Evaluate the loaded model to confirm accuracy
test_loss, test_acc = model_cnn_v3.evaluate(X_test_cnn, y_test,
verbose=2)
print(f"\n□ CNN v3 Test Accuracy: {test_acc:.4f}")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have
yet to be built. `model.compile_metrics` will be empty until you train
or evaluate the model.

□ Model loaded successfully from 'cnn_music_genre_model_v3.h5'
□ Predictions completed successfully!
```



```
63/63 - 2s - 27ms/step - accuracy: 0.9444 - loss: 0.2026

□ CNN v3 Test Accuracy: 0.9444
```

Why CNN Outperforms Logistic Regression and Other Traditional Methods

Our CNN is particularly effective for music genre classification compared to simpler models such as logistic regression or even traditional machine learning algorithms like random forests or support vector machines. This superiority arises from CNNs' inherent ability to:

Capture spatial hierarchies: CNNs leverage convolutional layers that capture local patterns in the MFCCs (frequency patterns), extracting meaningful hierarchical features crucial for distinguishing genres. Robust feature extraction: CNNs automatically learn the most salient features directly from input data, whereas traditional methods require manual feature engineering. Generalization: With regularization techniques like dropout, batch normalization, and global average pooling, CNNs generalize better to unseen data, achieving higher accuracy on validation and test sets. In summary, ReLU activation and CNN architecture combine powerful non-linear representation with automatic, deep feature extraction capability, significantly outperforming traditional classification methods in accuracy and robustness.

How ML4SIP AI Buddy enhanced optimization and coding in this project:

Using AI, we were able to significantly enhance the efficiency and accuracy of the CNN model developed in this project. Initially, a basic Convolutional Neural Network (CNN) was created manually; however, we ran into errors, such as input dimension errors, incorrect data reshaping, and suboptimal hyperparameters. By prompting the AI assistant, debugging was streamlined, pinpointing dimensionality issues immediately. The AI guided the proper reshaping of the MFCC inputs, explained necessary modifications clearly, and suggested to implement techniques like batch normalization, dropout, and global average pooling, dramatically reducing overfitting and improving generalization.

The AI also provided recommendations to use early stopping and learning rate schedulers (ReduceLROnPlateau), enhancing convergence stability and training efficiency. Through these insights, hyperparameters such as the learning rate, batch size, convolutional filters, and kernel sizes were carefully tuned to suit the specific MFCC data shape and achieve maximal accuracy.

Overall, the AI buddy made the debugging process faster and optimized model architecture effectively, allowing for us to develop a quicker and more robust model compared to traditional manual experimentation.