

```
import os
import csv
import shutil
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
from sklearn.utils.class_weight import compute_class_weight
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, GlobalAveragePooling2D, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
import matplotlib.gridspec as gridspec
from tensorflow.keras.applications import VGG16, InceptionV3
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping, ReduceLRonPlateau
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
```

```
from google.colab import drive
drive.mount('/content/drive')
```

 Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

✓ Reading and Combining CSV Files


```
train_csv_path = '/content/drive/MyDrive/Indian Dance Forms/train.csv'
test_csv_path = '/content/drive/MyDrive/Indian Dance Forms/test.csv'

train_df_1 = pd.read_csv(train_csv_path)
test_df_1 = pd.read_csv(test_csv_path)


df = pd.concat([train_df_1, test_df_1], ignore_index=True)
df.reset_index(drop=True, inplace=True)
df.to_csv('/content/drive/MyDrive/Indian Dance Forms/data.csv', index=False)
```

✓ Checking Data Integrity


```
train_df_1.info()
```

 `<class 'pandas.core.frame.DataFrame'>`
 RangeIndex: 364 entries, 0 to 363
 Data columns (total 2 columns):
 # Column Non-Null Count Dtype
 --- ---
 0 Image 364 non-null object
 1 target 364 non-null object
 dtypes: object(2)
 memory usage: 5.8+ KB

```
test_df_1.info()
```

 `<class 'pandas.core.frame.DataFrame'>`
 RangeIndex: 156 entries, 0 to 155
 Data columns (total 1 columns):
 # Column Non-Null Count Dtype
 --- ---
 0 Image 156 non-null object
 dtypes: object(1)
 memory usage: 1.3+ KB

```
df.info()
```

 `<class 'pandas.core.frame.DataFrame'>`
 RangeIndex: 520 entries, 0 to 519
 Data columns (total 2 columns):
 # Column Non-Null Count Dtype
 --- ---

```

0   Image    520 non-null    object
1   target   364 non-null    object
dtypes: object(2)
memory usage: 8.3+ KB

```

✧ Copying Images to a Unified Dataset Folder

```

source_folders = [
    '/content/drive/MyDrive/Indian Dance Forms/train',
    '/content/drive/MyDrive/Indian Dance Forms/test'
]
output_folder = '/content/drive/MyDrive/Indian Dance Forms/dataset'

os.makedirs(output_folder, exist_ok=True)
for folder in source_folders:
    for img_file in os.listdir(folder):
        src = os.path.join(folder, img_file)
        dst = os.path.join(output_folder, img_file)
        if not os.path.exists(dst):
            shutil.copy(src, dst)

def count_images_in_folder(folder, extensions=['.jpg', '.jpeg', '.png']):
    return len([f for f in os.listdir(folder) if os.path.splitext(f)[1].lower() in extensions])

train_folder = "/content/drive/MyDrive/Indian Dance Forms/train"
test_folder = "/content/drive/MyDrive/Indian Dance Forms/test"

num_train = count_images_in_folder(train_folder)
num_test = count_images_in_folder(test_folder)

print(f"Number of images in train folder: {num_train}")
print(f"Number of images in test folder: {num_test}")

```

```

➡ Number of images in train folder: 364
   Number of images in test folder: 156

```

✧ Filtering CSVs to Match Available Images

This code filters a CSV file so it only keeps rows for images that actually exist in the dataset folder. It checks each image name from the CSV and writes that row to a new CSV only if the image file is present in the folder. The result is a CSV that matches the images available in dataset folder, preventing errors from missing files during training.

```

def filter_csv_by_existing_images(csv_file, image_folder, output_csv):
    with open(csv_file, 'r') as csvfile, open(output_csv, 'w', newline='') as outputfile:
        csv_reader = csv.reader(csvfile)
        csv_writer = csv.writer(outputfile)
        header = next(csv_reader)
        csv_writer.writerow(header)
        for row in csv_reader:
            image_name = row[0]
            image_path = os.path.join(image_folder, image_name)
            if os.path.exists(image_path):
                csv_writer.writerow(row)

filter_csv_by_existing_images(
    '/content/drive/MyDrive/Indian Dance Forms/train.csv',
    '/content/drive/MyDrive/Indian Dance Forms/train',
    '/content/drive/MyDrive/Indian Dance Forms/filtered_train.csv')
filter_csv_by_existing_images(
    '/content/drive/MyDrive/Indian Dance Forms/test.csv',
    '/content/drive/MyDrive/Indian Dance Forms/test',
    '/content/drive/MyDrive/Indian Dance Forms/filtered_test.csv' )

```

```

filtered_train=pd.read_csv('/content/drive/MyDrive/Indian Dance Forms/filtered_train.csv')
filtered_test=pd.read_csv('/content/drive/MyDrive/Indian Dance Forms/filtered_test.csv')
print(filtered_train.info())
print(filtered_test.info())

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 364 entries, 0 to 363
Data columns (total 2 columns):
#   Column  Non-Null Count  Dtype
---  ---
0   Image    364 non-null      object
1   target   364 non-null      object
dtypes: object(2)
memory usage: 5.8+ KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 156 entries, 0 to 155
Data columns (total 1 columns):
#   Column  Non-Null Count  Dtype
---  ---
0   Image    156 non-null      object
dtypes: object(1)
memory usage: 1.3+ KB
None

```

✓ Train-Test Split

```

train_data, test_data = train_test_split(filtered_train, test_size=0.2,
                                         random_state=42, stratify=filtered_train['target'])
dataset_dir = '/content/drive/MyDrive/Indian Dance Forms/dataset/'
IMAGE_WIDTH, IMAGE_HEIGHT = 224, 224
BATCH_SIZE = 32
EPOCHS = 20

```

✓ Visualizing Class Distribution

```

def display_class_images(df, image_folder):
    class_images = {}
    image_paths = df['Image'].tolist()
    class_labels = df['target'].tolist()
    # Group image paths by class label
    for image_path, class_label in zip(image_paths, class_labels):
        if class_label not in class_images:
            class_images[class_label] = []
        class_images[class_label].append(image_path)
    # Display one image per class
    fig = plt.figure(figsize=(16, 4))
    gs = gridspec.GridSpec(1, len(class_images))
    for i, (class_label, images) in enumerate(class_images.items()):
        ax = plt.subplot(gs[0, i])
        ax.set_title(class_label)
        img = Image.open(os.path.join(image_folder, images[0])) # First image of each class
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()

```

```
display_class_images(train_data, dataset_dir)
```

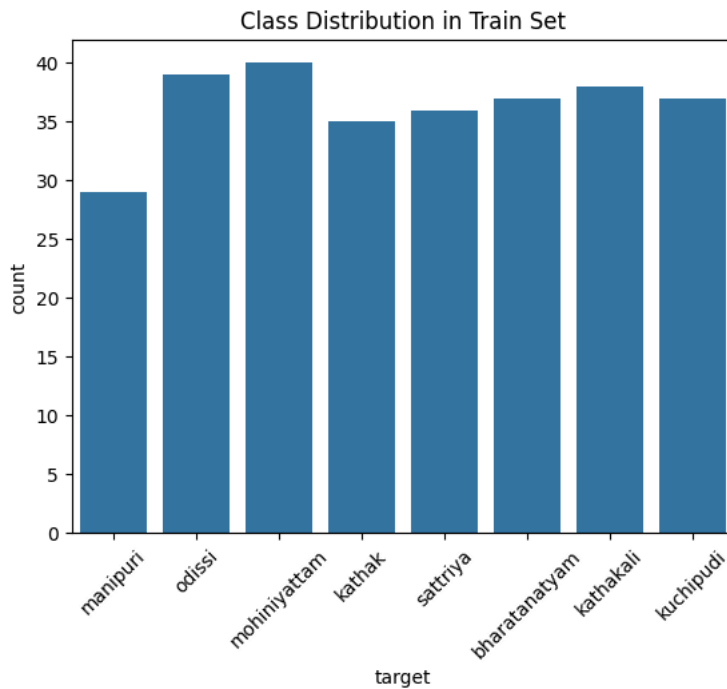


```

sns.countplot(data=train_data, x='target')
plt.xticks(rotation=45)

```

```
plt.title("Class Distribution in Train Set")
plt.show()
```



Data Augmentation and Generators

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_dataframe(
    dataframe=train_data,
    directory=dataset_dir,
    x_col="Image",
    y_col="target",
    target_size=(IMAGE_WIDTH, IMAGE_HEIGHT),
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)
test_generator = test_datagen.flow_from_dataframe(
    dataframe=test_data,
    directory=dataset_dir,
    x_col="Image",
    y_col="target",
    target_size=(IMAGE_WIDTH, IMAGE_HEIGHT),
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=False
)
```



Found 291 validated image filenames belonging to 8 classes.
Found 73 validated image filenames belonging to 8 classes.

Handling Class Imbalance - Computing Class Weights

- Higher weight → class is underrepresented

- Lower weight → class is overrepresented

```
from sklearn.utils.class_weight import compute_class_weight
import numpy as np
```

```
class_labels = train_data['target'].unique()
label_to_index = train_generator.class_indices
index_to_label = {v: k for k, v in label_to_index.items()}
y_train_integers = train_data['target'].map(label_to_index)
```

```
# Compute class weights
```

```
class_weights_array = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(y_train_integers),
    y=y_train_integers
)
```

```
class_weights = {
    class_idx: weight for class_idx, weight in zip(np.unique(y_train_integers), class_weights_array)
}
```

```
print("Class Weights:", class_weights)
```

```
➦ Class Weights: {np.int64(0): np.float64(0.9831081081081081), np.int64(1): np.float64(1.0392857142857144), np.int64(2): np.float64(0.9572
```

- Compute class weights to handle class imbalance.
- This ensures that underrepresented classes have a higher weight, so the model pays more attention to them during training.

✓ CNN Model

```
cnn_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(8, activation='softmax')
])
```

```
cnn_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
#Early stopping
```

```
early_stop = EarlyStopping(
    monitor='val_loss', patience=5, restore_best_weights=True)
history=cnn_model.fit(
    train_generator,epochs=EPOCHS,
    validation_data=test_generator,
    class_weight=class_weights,callbacks=[early_stop])
```

```
➦ ackages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
larizer=activity_regularizer, **kwargs)
ackages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**k
())
```

```
10s/step - accuracy: 0.0974 - loss: 2.1582 - val_accuracy: 0.0959 - val_loss: 2.0946
```

```
3ms/step - accuracy: 0.0959 - loss: 2.0736 - val_accuracy: 0.1644 - val_loss: 2.0772
```

```
0ms/step - accuracy: 0.1523 - loss: 2.0763 - val_accuracy: 0.1370 - val_loss: 2.0561
```

```
4ms/step - accuracy: 0.1315 - loss: 2.0339 - val_accuracy: 0.1781 - val_loss: 2.0209
```

```
9ms/step - accuracy: 0.2231 - loss: 2.0081 - val_accuracy: 0.1370 - val_loss: 2.0175
```

```
3ms/step - accuracy: 0.1871 - loss: 2.0046 - val_accuracy: 0.2877 - val_loss: 1.9845
```

```
5ms/step - accuracy: 0.2208 - loss: 1.9771 - val_accuracy: 0.2329 - val_loss: 2.1568
5ms/step - accuracy: 0.2622 - loss: 1.9167 - val_accuracy: 0.1370 - val_loss: 2.2781
2ms/step - accuracy: 0.3251 - loss: 1.8437 - val_accuracy: 0.2603 - val_loss: 2.1132
1ms/step - accuracy: 0.3274 - loss: 1.7735 - val_accuracy: 0.3425 - val_loss: 1.8716
8ms/step - accuracy: 0.3280 - loss: 1.7848 - val_accuracy: 0.3699 - val_loss: 1.7596
3ms/step - accuracy: 0.2804 - loss: 1.7942 - val_accuracy: 0.3151 - val_loss: 1.7151
8ms/step - accuracy: 0.3452 - loss: 1.7577 - val_accuracy: 0.3425 - val_loss: 2.0098
8ms/step - accuracy: 0.3306 - loss: 1.7751 - val_accuracy: 0.3973 - val_loss: 1.8002
8ms/step - accuracy: 0.3969 - loss: 1.6536 - val_accuracy: 0.2466 - val_loss: 2.0785
4ms/step - accuracy: 0.3763 - loss: 1.6883 - val_accuracy: 0.3836 - val_loss: 1.6873
1ms/step - accuracy: 0.4124 - loss: 1.6880 - val_accuracy: 0.2877 - val_loss: 2.2233
3ms/step - accuracy: 0.3997 - loss: 1.6142 - val_accuracy: 0.2877 - val_loss: 2.3659
4ms/step - accuracy: 0.4200 - loss: 1.4730 - val_accuracy: 0.2603 - val_loss: 2.0537
0ms/step - accuracy: 0.3921 - loss: 1.5722 - val_accuracy: 0.3562 - val_loss: 2.0414
```

▼ CNN Evaluation

```
loss, accuracy = cnn_model.evaluate(test_generator)
print("CNN Test Loss:", loss)
print("CNN Test Accuracy:", accuracy)

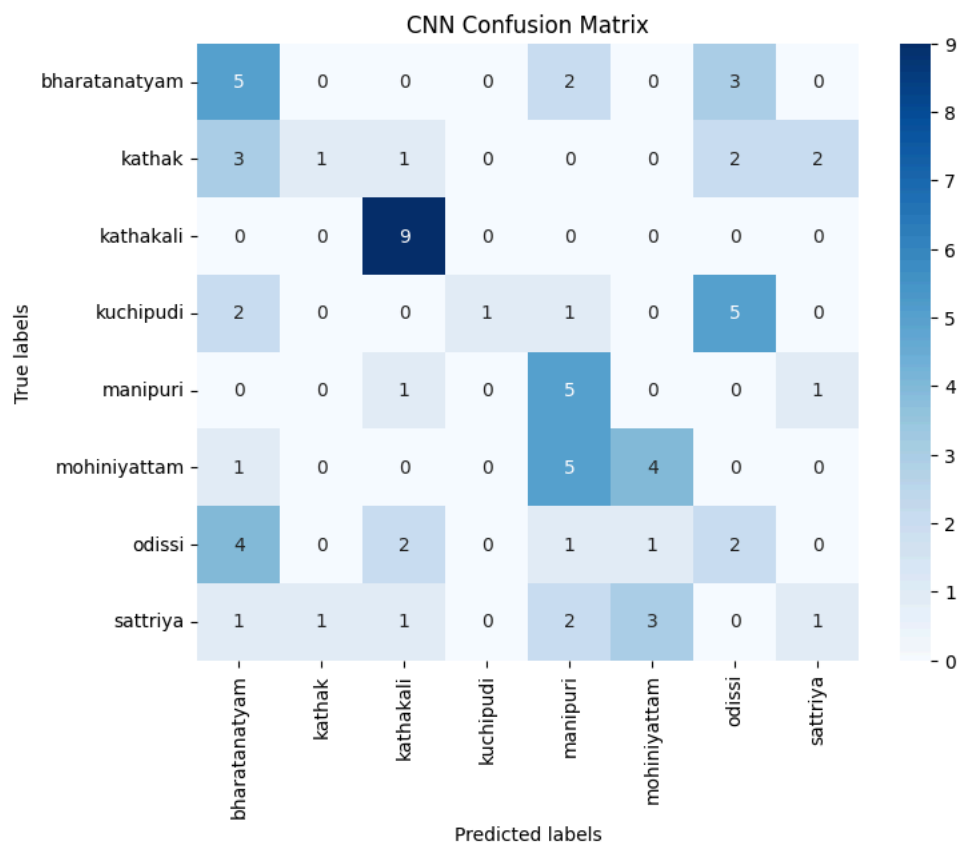
predictions = cnn_model.predict(test_generator)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = test_generator.classes
class_labels = list(test_generator.class_indices.keys())

print("CNN Accuracy:", accuracy_score(true_classes, predicted_classes))
print("CNN Classification Report:\n", classification_report(true_classes, predicted_classes, target_names=class_labels))

conf_matrix = confusion_matrix(true_classes, predicted_classes)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('CNN Confusion Matrix')
plt.show()
```

3/3 0s 87ms/step - accuracy: 0.3871 - loss: 1.6452
 CNN Test Loss: 1.6873048543930054
 CNN Test Accuracy: 0.3835616409778595
 3/3 1s 194ms/step
 CNN Accuracy: 0.3835616438356164
 CNN Classification Report:

	precision	recall	f1-score	support
bharatanatyam	0.31	0.50	0.38	10
kathak	0.50	0.11	0.18	9
kathakali	0.64	1.00	0.78	9
kuchipudi	1.00	0.11	0.20	9
manipuri	0.31	0.71	0.43	7
mohiniyattam	0.50	0.40	0.44	10
odissi	0.17	0.20	0.18	10
sattriya	0.25	0.11	0.15	9
accuracy			0.38	73
macro avg	0.46	0.39	0.35	73
weighted avg	0.46	0.38	0.34	73



Results

- The model achieved about 37% accuracy on the test set, indicating limited ability to distinguish between dance forms.
- Training loss: Decreased from 2.21 to 1.67, showing the model was learning to fit the training data.
- Validation loss: Fluctuated and remained high (around 1.85–2.13), the model struggled to generalize.

Classification Report

- Precision: High for some classes (e.g., kathak), but this is misleading due to very low recall.
- Recall: Highest for kathakali (0.89), lowest for kathak (0.11).
- F1-score: Indicates poor balance between precision and recall for most classes.

Confusion Martix

- The confusion matrix shows the model frequently confuses similar dance forms.
- Some classes (like kathakali) are predicted well, but others (like kathak, odissi, sattriya) are often misclassified.
- The model tends to predict certain classes more often, indicating bias or confusion due to visual similarity.

Reason for low performance

- **Model Complexity:** The CNN architecture was too shallow for the complexity of distinguishing fine-grained dance forms.
- **Data Size:** The dataset was relatively small, limiting the model's ability to learn diverse features.
- **Class Imbalance:** Some dance forms had fewer samples, making it harder for the model to learn those classes.
- **Visual Similarity:** Many Indian classical dances have similar costumes and poses, increasing class confusion.
- **Overfitting Risk:** Although data augmentation and class weights were used, the model still struggled to generalize.

Steps Taken to Address Issues

- **Data Augmentation:** Applied rotations, shifts, zooms, and flips to increase data diversity and reduce overfitting.
- **Class Weights:** Assigned higher weights to minority classes to help the model pay more attention to them.
- **Careful Data Cleaning:** Ensured only existing images were used, preventing missing file errors.
- **Early Stopping:** Used to prevent overfitting by stopping training when validation loss stopped improving.

✓ VGG16 Model

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, 3))
for layer in base_model.layers[-4:]:
    layer.trainable = True
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
predictions = Dense(8, activation='softmax')(x)

tl_model = Model(inputs=base_model.input, outputs=predictions)
tl_model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
history_vgg16=tl_model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=test_generator,
    validation_steps=test_generator.samples // BATCH_SIZE,
    class_weight=class_weights,
    callbacks=[early_stop]
)
```

```
Epoch 1/20
9/9 ----- 20s 1s/step - accuracy: 0.1282 - loss: 2.1749 - val_accuracy: 0.1719 - val_loss: 2.0655
Epoch 2/20
1/9 ----- 3s 423ms/step - accuracy: 0.0625 - loss: 2.0536 - val_accuracy: 0.1719 - val_loss: 2.0671
self._interrupted_warning()
9/9 ----- 1s 58ms/step - accuracy: 0.0625 - loss: 2.0536 - val_accuracy: 0.1719 - val_loss: 2.0671
Epoch 3/20
9/9 ----- 21s 2s/step - accuracy: 0.1295 - loss: 2.0717 - val_accuracy: 0.2031 - val_loss: 1.9718
Epoch 4/20
9/9 ----- 1s 96ms/step - accuracy: 0.0625 - loss: 2.0818 - val_accuracy: 0.2812 - val_loss: 1.9592
Epoch 5/20
9/9 ----- 16s 1s/step - accuracy: 0.2375 - loss: 2.0055 - val_accuracy: 0.1562 - val_loss: 1.9947
Epoch 6/20
9/9 ----- 1s 100ms/step - accuracy: 0.0625 - loss: 2.2574 - val_accuracy: 0.2344 - val_loss: 1.9538
Epoch 7/20
9/9 ----- 17s 889ms/step - accuracy: 0.2556 - loss: 1.9492 - val_accuracy: 0.2969 - val_loss: 1.8723
Epoch 8/20
9/9 ----- 1s 62ms/step - accuracy: 0.3438 - loss: 1.8700 - val_accuracy: 0.3125 - val_loss: 1.7769
Epoch 9/20
9/9 ----- 9s 827ms/step - accuracy: 0.3825 - loss: 1.7053 - val_accuracy: 0.2812 - val_loss: 1.6497
Epoch 10/20
9/9 ----- 1s 165ms/step - accuracy: 0.0000e+00 - loss: 2.0955 - val_accuracy: 0.3281 - val_loss: 1.7489
Epoch 11/20
9/9 ----- 8s 667ms/step - accuracy: 0.4330 - loss: 1.6218 - val_accuracy: 0.2969 - val_loss: 1.6941
Epoch 12/20
9/9 ----- 1s 100ms/step - accuracy: 0.1250 - loss: 2.0714 - val_accuracy: 0.3125 - val_loss: 1.6730
Epoch 13/20
9/9 ----- 10s 804ms/step - accuracy: 0.3797 - loss: 1.6336 - val_accuracy: 0.4688 - val_loss: 1.4862
Epoch 14/20
9/9 ----- 1s 104ms/step - accuracy: 0.5312 - loss: 1.2198 - val_accuracy: 0.4688 - val_loss: 1.5112
Epoch 15/20
9/9 ----- 10s 972ms/step - accuracy: 0.4864 - loss: 1.3466 - val_accuracy: 0.4844 - val_loss: 1.2435
Epoch 16/20
9/9 ----- 1s 58ms/step - accuracy: 0.6562 - loss: 1.0731 - val_accuracy: 0.4531 - val_loss: 1.3100
Epoch 17/20
9/9 ----- 10s 1s/step - accuracy: 0.4853 - loss: 1.3473 - val_accuracy: 0.3750 - val_loss: 1.6073
Epoch 18/20
```



```
9/9 ————— 1s 59ms/step - accuracy: 1.0000 - loss: 1.2069 - val_accuracy: 0.3906 - val_loss: 1.6425
Epoch 19/20
9/9 ————— 18s 739ms/step - accuracy: 0.5297 - loss: 1.3307 - val_accuracy: 0.4219 - val_loss: 1.6230
Epoch 20/20
9/9 ————— 1s 71ms/step - accuracy: 0.6250 - loss: 1.0091 - val_accuracy: 0.4375 - val_loss: 1.4419
```

▼ VGG16 Evaluation

```
loss, accuracy = tl_model.evaluate(test_generator)
print("VGG16 Test Loss:", loss)
print("VGG16 Test Accuracy:", accuracy)

y_pred_prob = tl_model.predict(test_generator)
y_pred = np.argmax(y_pred_prob, axis=1)
y_true = test_generator.classes

print("VGG16 Accuracy:", accuracy_score(y_true, y_pred))
print("VGG16 Classification Report:\n", classification_report(y_true, y_pred, target_names=class_labels))

conf_matrix = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('VGG16 Confusion Matrix')
plt.show()
```



```

3/3 ----- 1s 442ms/step - accuracy: 0.4936 - loss: 1.2580
VGG16 Test Loss: 1.2877321243286133
VGG16 Test Accuracy: 0.4794520437717438
WARNING:tensorflow:5 out of the last 7 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed
2/3 ----- 0s 155ms/stepWARNING:tensorflow:6 out of the last 9 calls to <function TensorFlowTrainer.make_predict_function.
3/3 ----- 2s 388ms/step
VGG16 Accuracy: 0.4794520547945205
VGG16 Classification Report:

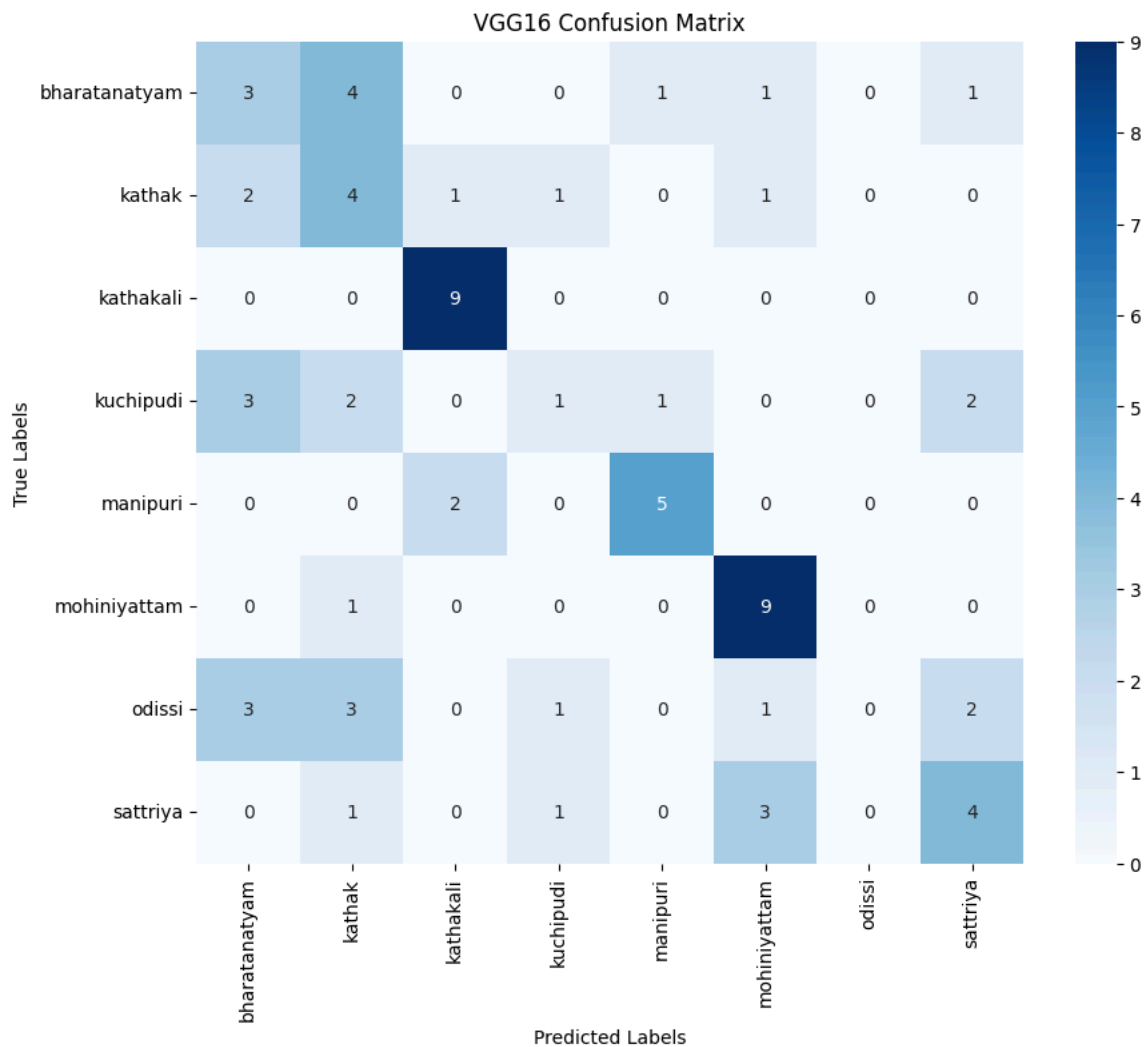
```

	precision	recall	f1-score	support
bharatanatyam	0.27	0.30	0.29	10
kathak	0.27	0.44	0.33	9
kathakali	0.75	1.00	0.86	9
kuchipudi	0.25	0.11	0.15	9
manipuri	0.71	0.71	0.71	7
mohiniyattam	0.60	0.90	0.72	10
odissi	0.00	0.00	0.00	10
sattriya	0.44	0.44	0.44	9
accuracy			0.48	73
macro avg	0.41	0.49	0.44	73
weighted avg	0.40	0.48	0.43	73

```

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and be
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and be
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and be
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

```



Results

- The model achieved about 48% accuracy on the test set, showing clear improvement over the baseline CNN but still leaving room for further gains.
- Training loss: Decreased from 2.17 to 1.07, indicating the model was able to fit the training data.

- Validation loss: Decreased from 2.06 to 1.24, but fluctuated in later epochs, suggesting some instability or overfitting.

Classification Report

- Precision: High for kathakali (0.75), manipuri (0.71), and mohiniyattam (0.60), but 0.00 for odissi (model never predicted this class).
- Recall: Highest for kathakali (1.00), mohiniyattam (0.90), and manipuri (0.71); lowest for kuchipudi (0.11) and odissi (0.00).
- F1-score: Indicates a few classes (kathakali, mohiniyattam, manipuri) are recognized well, but others are not.

Confusion Martix

- kathakali and mohiniyattam are predicted very well (most samples correctly classified).
- odissi is never predicted correctly (all predictions are wrong).
- kuchipudi and bharatanatyam are often misclassified as each other or as other classes.
- Some improvement in distinguishing between classes compared to the baseline, but confusion remains, especially for visually similar or underrepresented classes.

Reason for low performance

- **Model Complexity:** VGG16, with its deep layers and pretrained ImageNet weights, extracts more meaningful features than a simple CNN, which is why accuracy improved.
- **Transfer Learning:** Leveraging pretrained weights helped the model generalize better, especially with limited data.
- **Class Weights:** Helped the model pay more attention to minority classes, but not enough to fully resolve class imbalance.
- **Data Augmentation:** Increased effective dataset size and diversity, reducing overfitting.
- **Overfitting Risk:** Despite early stopping and augmentation, validation loss fluctuated, indicating some overfitting or instability, likely due to the small dataset.

Steps Taken to Address Issues

- **Data Augmentation:** Applied rotations, shifts, zooms, and flips to increase data diversity and reduce overfitting.
- **Class Weights:** Assigned higher weights to minority classes to help the model pay more attention to them.
- **Careful Data Cleaning:** Ensured only existing images were used, preventing missing file errors.
- **Early Stopping:** Used to prevent overfitting by stopping training when validation loss stopped improving.
- **Transfer Learning:** Used VGG16 pretrained on ImageNet to leverage learned features.verage learned general features.

✓ InceptionV3 Model

```
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, 3))
for layer in base_model.layers:
    layer.trainable = False
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
predictions = Dense(8, activation='softmax')(x)
tf_model2 = Model(inputs=base_model.input, outputs=predictions)
tf_model2.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
#Early stopping
early_stop = EarlyStopping(
    monitor='val_loss', patience=5, restore_best_weights=True)
history_v3=tf_model2.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=test_generator,
    validation_steps=test_generator.samples // BATCH_SIZE,
    class_weight=class_weights,callbacks=[early_stop]
)
```

```
📄 Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_
87910968/87910968 ————— 0s 0us/step
Epoch 1/20
9/9 ————— 34s 2s/step - accuracy: 0.1651 - loss: 2.1519 - val_accuracy: 0.2188 - val_loss: 1.8799
Epoch 2/20
1/9 ————— 0s 77ms/step - accuracy: 0.2500 - loss: 2.0966/usr/local/lib/python3.11/dist-packages/keras/src/trainers/epoch_
self._interrupted_warning()
9/9 ————— 2s 209ms/step - accuracy: 0.2500 - loss: 2.0966 - val_accuracy: 0.2656 - val_loss: 1.8463
Epoch 3/20
```

```

9/9 ————— 18s 693ms/step - accuracy: 0.2891 - loss: 1.8872 - val_accuracy: 0.3594 - val_loss: 1.6629
Epoch 4/20
9/9 ————— 1s 68ms/step - accuracy: 0.0000e+00 - loss: 2.1402 - val_accuracy: 0.4219 - val_loss: 1.6481
Epoch 5/20
9/9 ————— 10s 1s/step - accuracy: 0.4248 - loss: 1.5781 - val_accuracy: 0.3438 - val_loss: 1.5966
Epoch 6/20
9/9 ————— 1s 135ms/step - accuracy: 0.5625 - loss: 1.5073 - val_accuracy: 0.3438 - val_loss: 1.5921
Epoch 7/20
9/9 ————— 5s 507ms/step - accuracy: 0.5385 - loss: 1.4153 - val_accuracy: 0.4688 - val_loss: 1.4803
Epoch 8/20
9/9 ————— 1s 67ms/step - accuracy: 0.6250 - loss: 1.2134 - val_accuracy: 0.4375 - val_loss: 1.4769
Epoch 9/20
9/9 ————— 5s 562ms/step - accuracy: 0.5227 - loss: 1.3051 - val_accuracy: 0.4688 - val_loss: 1.4156
Epoch 10/20
9/9 ————— 1s 78ms/step - accuracy: 0.4688 - loss: 1.3793 - val_accuracy: 0.5156 - val_loss: 1.3952
Epoch 11/20
9/9 ————— 5s 538ms/step - accuracy: 0.6200 - loss: 1.1949 - val_accuracy: 0.5625 - val_loss: 1.3422
Epoch 12/20
9/9 ————— 0s 45ms/step - accuracy: 0.7188 - loss: 1.2496 - val_accuracy: 0.5156 - val_loss: 1.3508
Epoch 13/20
9/9 ————— 14s 1s/step - accuracy: 0.5659 - loss: 1.1555 - val_accuracy: 0.4844 - val_loss: 1.2932
Epoch 14/20
9/9 ————— 1s 65ms/step - accuracy: 0.5938 - loss: 1.1092 - val_accuracy: 0.4844 - val_loss: 1.2902
Epoch 15/20
9/9 ————— 5s 533ms/step - accuracy: 0.6268 - loss: 1.1006 - val_accuracy: 0.5781 - val_loss: 1.2603
Epoch 16/20
9/9 ————— 1s 56ms/step - accuracy: 0.5938 - loss: 1.0827 - val_accuracy: 0.5781 - val_loss: 1.2694
Epoch 17/20
9/9 ————— 10s 552ms/step - accuracy: 0.6387 - loss: 1.0396 - val_accuracy: 0.5938 - val_loss: 1.1784
Epoch 18/20
9/9 ————— 0s 44ms/step - accuracy: 0.3333 - loss: 1.8105 - val_accuracy: 0.5938 - val_loss: 1.1999
Epoch 19/20
9/9 ————— 9s 506ms/step - accuracy: 0.6721 - loss: 0.9866 - val_accuracy: 0.6250 - val_loss: 1.1676
Epoch 20/20
9/9 ————— 1s 66ms/step - accuracy: 0.7188 - loss: 1.0133 - val_accuracy: 0.6562 - val_loss: 1.1547

```

✓ InceptionV3 Evaluation

```

loss, accuracy = tf_model2.evaluate(test_generator)
print("InceptionV3 Test Loss:", loss)
print("InceptionV3 Test Accuracy:", accuracy)

y_pred_prob = tf_model2.predict(test_generator)
y_pred = np.argmax(y_pred_prob, axis=1)
y_true = test_generator.classes

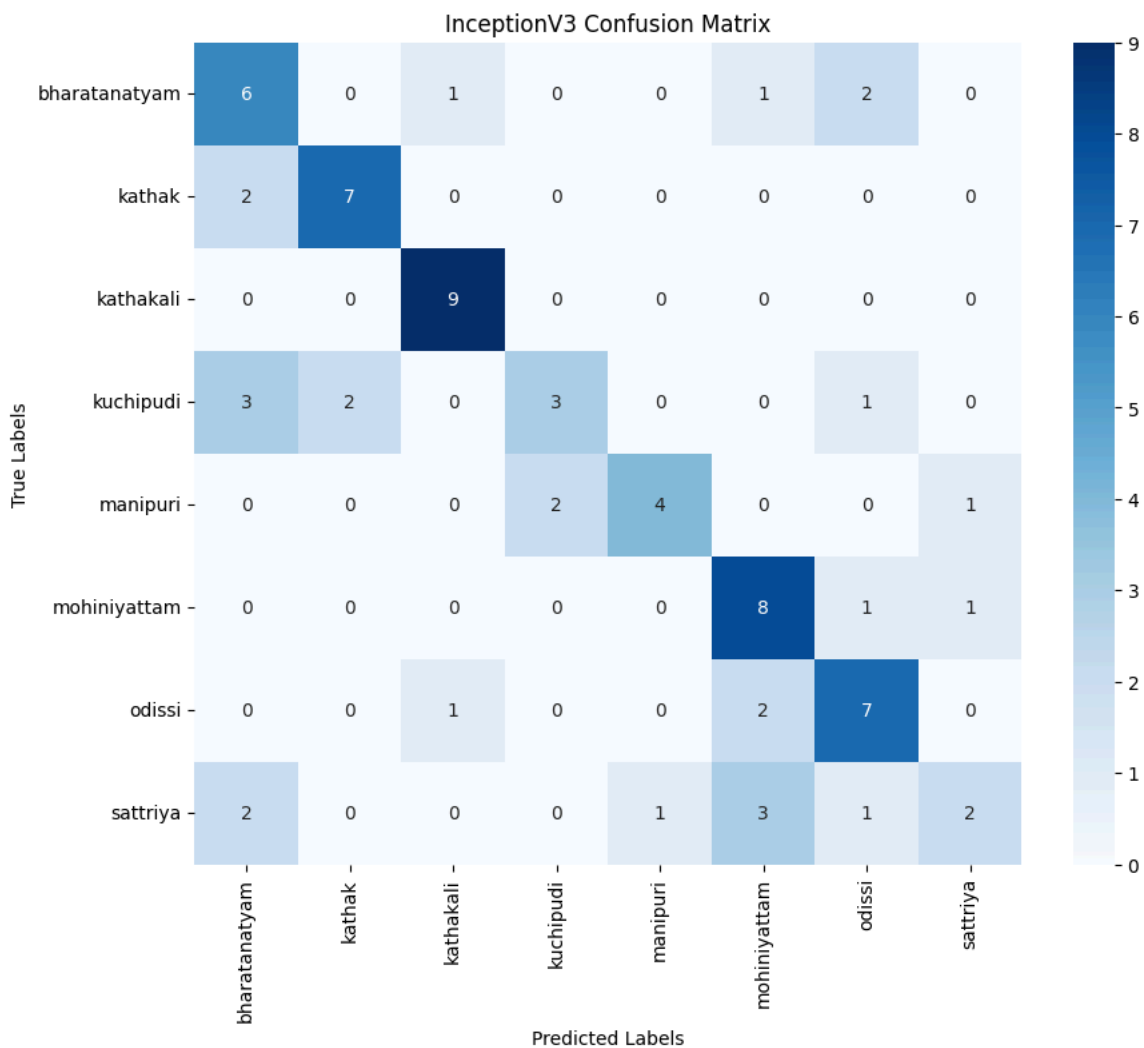
print("InceptionV3 Accuracy:", accuracy_score(y_true, y_pred))
print("InceptionV3 Classification Report:\n", classification_report(y_true, y_pred, target_names=class_labels))

conf_matrix = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('InceptionV3 Confusion Matrix')
plt.show()

```

3/3 ————— 6s 3s/step - accuracy: 0.6276 - loss: 1.1837
 InceptionV3 Test Loss: 1.1821907758712769
 InceptionV3 Test Accuracy: 0.6301369667053223
 3/3 ————— 12s 3s/step
 InceptionV3 Accuracy: 0.6301369863013698
 InceptionV3 Classification Report:

	precision	recall	f1-score	support
bharatanatyam	0.46	0.60	0.52	10
kathak	0.78	0.78	0.78	9
katakali	0.82	1.00	0.90	9
kuchipudi	0.60	0.33	0.43	9
manipuri	0.80	0.57	0.67	7
mohiniyattam	0.57	0.80	0.67	10
odissi	0.58	0.70	0.64	10
sattriya	0.50	0.22	0.31	9
accuracy			0.63	73
macro avg	0.64	0.63	0.61	73
weighted avg	0.63	0.63	0.61	73



- Results
- The model achieved about 63% accuracy on the test set, a significant improvement over the baseline CNN and VGG16 models.
- Training loss: Decreased steadily from 2.15 to 1.01, indicating the model was fitting the training data well.
- Validation loss: Decreased from 1.88 to 1.15, showing improved generalization and less overfitting compared to previous models.
- Classification Report
- Precision: Highest for kathakali (0.82) and kathak (0.78), lowest for bharatanatyam (0.46).
 - Recall: Highest for kathakali (1.00), mohiniyattam (0.80), and kathak (0.78); lowest for kuchipudi (0.33) and sattriya (0.22).
 - F1-score: Indicates a good balance between precision and recall for most classes, especially kathakali and kathak.

Confusion Matrix

- Most classes are predicted correctly, especially kathakali, kathak, odiss, and mohiniyattam.
- There is still some confusion between visually similar dance forms, e.g.:
 - Kuchipudi is sometimes predicted as bharatanatyam or manipuri.
 - Sattriya is confused with bharatanatyam and mohiniyattam
 - Manipuri predictions are spread over several classes.
- Kathak and mohiniyattam have a few misclassifications, but overall, the model distinguishes most classes well.

Reason for Improved Performance

- **Model Complexity:** InceptionV3 is a much deeper and more sophisticated architecture than a simple CNN or VGG16, capable of extracting complex features from images.
- **Transfer Learning:** Using pretrained weights from ImageNet allowed the model to leverage learned visual features, which is especially beneficial with a small dataset.
- **Class Weights:** Helped the model focus on underrepresented classes, reducing bias toward majority classes.
- **Data Augmentation:** Increased the effective size and diversity of the training set, helping the model generalize better.
- **Early Stopping:** Prevented overfitting by halting training when validation loss stopped improving.

Steps Taken to Address Issues

- **Data Augmentation:** Used rotations, shifts, zooms, and flips to increase data diversity and reduce overfitting.
- **Class Weights:** Assigned higher weights to minority classes to help the model pay more attention to them.
- **Data Cleaning:** Ensured only existing images were used, preventing missing file errors.
- **Early Stopping:** Used to prevent overfitting by stopping training when validation loss stopped improving.
- **Transfer Learning with InceptionV3:** Leveraged a powerful, pretrained model to extract rich features from the images.

✓ Predicting on Unseen Images (All Models)

```
def predict_dance_form(image_path, model, train_generator):
    IMG_DIM = (IMAGE_WIDTH, IMAGE_HEIGHT)
    orig_im = image.load_img(image_path, target_size=IMG_DIM)
    plt.imshow(orig_im)
    plt.axis('off')
    plt.show()
    custom_im = image.img_to_array(orig_im)
    custom_im_scaled = custom_im.astype('float32') / 255
    custom_im_scaled = custom_im_scaled.reshape((1,) + custom_im_scaled.shape)
    predictions = model.predict(custom_im_scaled)
    predicted_class_index = np.argmax(predictions[0])
    class_labels = train_generator.class_indices
    class_labels = {v: k for k, v in class_labels.items()}
    predicted_class = class_labels[predicted_class_index]
    print("Predicted Dance Form:", predicted_class)

uploaded_images_folder = "/content/drive/MyDrive/Indian Dance Forms/TEST_data/"
uploaded_image_files = os.listdir(uploaded_images_folder)

print("CNN Predictions:")
for image_file in uploaded_image_files:
    image_path = os.path.join(uploaded_images_folder, image_file)
    predict_dance_form(image_path, cnn_model, train_generator)
```

→ CNN Predictions:



1/1 ————— 1s 810ms/step
Predicted Dance Form: manipuri



1/1 ————— 0s 32ms/step
Predicted Dance Form: odissi



1/1 ————— 0s 44ms/step
Predicted Dance Form: mohiniyattam





1/1 — 0s 34ms/step
Predicted Dance Form: kathak



1/1 — 0s 35ms/step
Predicted Dance Form: manipuri



1/1 — 0s 35ms/step
Predicted Dance Form: bharatanatyam



1/1 ————— 0s 35ms/step
Predicted Dance Form: kathakali



1/1 ————— 0s 36ms/step
Predicted Dance Form: bharatanatyam