Task 2: Multi-layer Perceptron

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

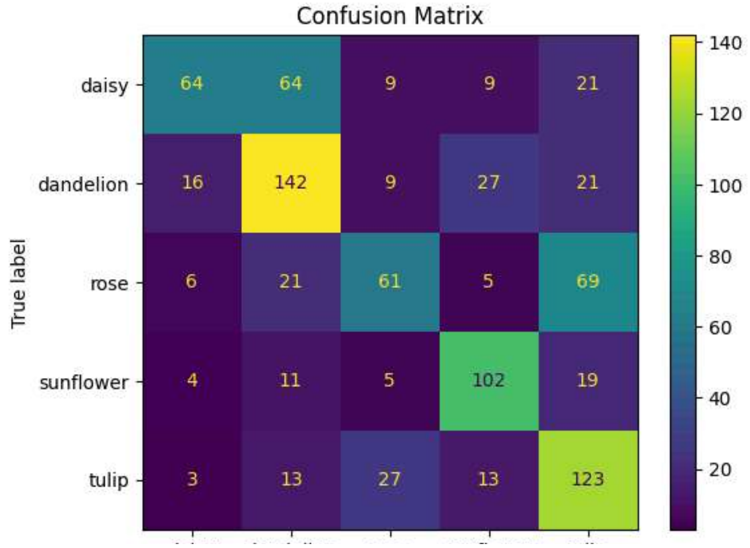
```
Mounted at /content/drive
```

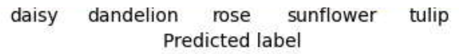
```
import os
import numpy as np
import matplotlib.pyplot as plt
import cv2
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matr
from tqdm import tqdm
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
import time

# Define global variables
LABELS = ["daisy", "dandelion", "rose", "sunflower", "tulip"]
BASE_PATH = "/content/drive/My Drive/Colab Notebooks/testing2/flowers"  # Update to your dataset pa
HIST_SIZE = 6  # Histogram size
EPOCHS = 10  # Default number of epochs
BATCH_SIZE = 32  # Default batch size

# ====================== Question 2.1: Preprocessing and Feature Extraction ======================
def preprocess_and_extract_histograms(base_path, labels, hist_size=6):
    """
    Preprocess images and extract color histogram features.
    """
    X_histograms = []
    X_original = []
    y = []
    for label in labels:
        folder_path = os.path.join(base_path, label)
        for img_name in tqdm(os.listdir(folder_path), desc=f"Processing {label}"):
            img_path = os.path.join(folder_path, img_name)
            image = cv2.imread(img_path, cv2.IMREAD_COLOR)
            if image is None:
                continue
            image_resized = cv2.resize(image, (150, 150))
            hist = cv2.calcHist([image_resized], [0, 1, 2], None, [hist_size, hist_size, hist_size]
            hist = cv2.normalize(hist, hist).flatten()
            X_histograms.append(hist)
            X_original.append(image_resized)  # Save original image
            y.append(label)
    return np.array(X_histograms), np.array(X_original), np.array(y)


# ====================== Question 2.2: Identify Optimal Network Structure ======================
def build_mlp(input_size, output_size, structure):
    """
    Build an MLP model with the given structure.
    """
    model = Sequential()
    model.add(Dense(structure[0], input_dim=input_size, activation='relu'))
    for units in structure[1:]:
        model.add(Dense(units, activation='relu'))
    model.add(Dense(output_size, activation='softmax'))
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model


def train_and_evaluate_structures(X_train, y_train, X_val, y_val, input_size, output_size, epochs=E
    """
    Train and evaluate nine different MLP structures.
    """
    structures = [
        [int(2 / 3 * input_size + output_size)],  # Rule 1
        [input_size // 2],                       # Rule 2
        [input_size],                            # Rule 3
        [int(2 / 3 * input_size + output_size), input_size // 2],
        [input_size, output_size],
        [int(input_size / 2), input_size],
        [int(2 / 3 * input_size + output_size), input_size // 2, output_size],
        [input_size, input_size // 2, output_size],
        [input_size, int(input_size / 2), output_size]
    ]

    best_model = None
    best_structure = None
    best_accuracy = 0
    training_times = []

    print("\n=== Question 2.2: Training and Evaluating Nine MLP Structures ===")
    for structure in structures:
        print(f"\nTraining MLP Structure: {structure}")
        model = build_mlp(input_size, output_size, structure)
        start_time = time.time()
        history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=epochs, batch_
        training_time = time.time() - start_time
        training_times.append(training_time)
        val_accuracy = history.history['val_accuracy'][-1]

        print(f"Validation Accuracy for Structure {structure}: {val_accuracy:.4f}")

        # Update best model if current model performs better
        if val_accuracy > best_accuracy:
            best_accuracy = val_accuracy
            best_model = model
            best_structure = structure

    print(f"\nBest MLP Structure: {best_structure}")
    print(f"Best Validation Accuracy: {best_accuracy:.4f}")
    return best_model, best_structure, best_accuracy, training_times, structures


# ====================== Question 2.3: Evaluate the Best Model ======================
def evaluate_model(model, X_test, y_test, labels):
    """
    Evaluate the best MLP model on the test set.
    """
    y_test_pred = np.argmax(model.predict(X_test), axis=1)
    y_test_actual = np.argmax(y_test, axis=1)
    acc = accuracy_score(y_test_actual, y_test_pred)
    precision = precision_score(y_test_actual, y_test_pred, average='weighted')
    recall = recall_score(y_test_actual, y_test_pred, average='weighted')
    f1 = f1_score(y_test_actual, y_test_pred, average='weighted')
    cm = confusion_matrix(y_test_actual, y_test_pred)

    print(f"\nTest Accuracy: {acc:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")

    # Plot confusion matrix
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
    disp.plot()
    plt.title("Confusion Matrix")
    plt.show()

    return y_test_pred, y_test_actual


# ====================== Question 2.5: Visualize Predictions ======================
def visualize_predictions(X_test, y_true, y_pred, labels, correct=True, num_samples=5):
    """
    Visualize correctly and incorrectly classified images.
    """
    y_true_indices = np.argmax(y_true, axis=1)  # Convert one-hot encoding to class indices
    indices = np.where(y_true_indices == y_pred)[0] if correct else np.where(y_true_indices != y_pr

    if len(indices) == 0:
        print("No samples to display.")
        return

    # Randomly sample indices
    sample_indices = np.random.choice(indices, min(num_samples, len(indices)), replace=False)

    plt.figure(figsize=(15, 10))
    for i, idx in enumerate(sample_indices):
        plt.subplot(1, num_samples, i + 1)
        img = X_test[idx]
        true_label = labels[y_true_indices[idx]]
        pred_label = labels[y_pred[idx]]
        title_color = "green" if correct else "red"
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        plt.title(f"True: {true_label}\nPred: {pred_label}", color=title_color)
        plt.axis('off')
    plt.tight_layout()
    plt.show()


# ====================== Main Code ======================
if __name__ == "__main__":
    # Validate dataset path and structure
    for label in LABELS:
        folder_path = os.path.join(BASE_PATH, label)
        if not os.path.exists(folder_path):
            print(f"Missing folder: {folder_path}")
            exit()

    print("\n--- Question 2.1: Preprocessing and Feature Extraction ---")
    print("\n=== Question 2.1: Preprocessing and Feature Extraction ===")
    X_histograms, X_original, y = preprocess_and_extract_histograms(BASE_PATH, LABELS, HIST_SIZE)
    print(f"Dataset loaded: {len(X_histograms)} samples")

    label_map = {label: idx for idx, label in enumerate(LABELS)}
    y_encoded = np.array([label_map[label] for label in y])
    y_one_hot = to_categorical(y_encoded, num_classes=len(LABELS))

    # Split dataset into training (60%), validation (20%), and test (20%) sets
    X_train, X_temp, y_train, y_temp, X_orig_train, X_orig_temp = train_test_split(
        X_histograms, y_one_hot, X_original, test_size=0.4, random_state=42
    )
    X_val, X_test, y_val, y_test, X_orig_val, X_orig_test = train_test_split(
        X_temp, y_temp, X_orig_temp, test_size=0.5, random_state=42
    )

    # Question 2.2: Train and evaluate MLP structures
    input_size = X_train.shape[1]
    output_size = len(LABELS)
    best_model, best_structure, best_accuracy, training_times, structures = train_and_evaluate_stru
        X_train, y_train, X_val, y_val, input_size, output_size
    )

    # Question 2.3: Evaluate the best model
    print("\n=== Question 2.3: Evaluate Best Model ===")
    y_test_pred, y_test_actual = evaluate_model(best_model, X_test, y_test, LABELS)

    # Question 2.4: Timing
    print("\n=== Question 2.4: Timing ===")
    start_time = time.time()
    best_model.predict(X_test)
    avg_inference_time = (time.time() - start_time) / len(X_test)
    print(f"Average Inference Time: {avg_inference_time:.4f} seconds")

    # Question 2.5: Visualize Predictions
    print("\n=== Question 2.5: Visualize Predictions ===")
    print("Correctly classified images:")
    visualize_predictions(X_orig_test, y_test, y_test_pred, LABELS, correct=True, num_samples=5)

    print("Incorrectly classified images:")
    visualize_predictions(X_orig_test, y_test, y_test_pred, LABELS, correct=False, num_samples=5)
```

```
=== Question 2.1: Preprocessing and Feature Extraction ===
Processing daisy: 100%|██████████| 764/764 [00:14<00:00, 52.15it/s]
Processing dandelion: 100%|██████████| 1052/1052 [00:19<00:00, 54.11it/s]
Processing rose: 100%|██████████| 784/784 [00:15<00:00, 50.23it/s]
Processing sunflower: 100%|██████████| 733/733 [00:14<00:00, 49.17it/s]
Processing tulip: 100%|██████████| 984/984 [00:18<00:00, 52.43it/s]
Dataset loaded: 4317 samples

=== Question 2.2: Training and Evaluating Nine MLP Structures ===

Training MLP Structure: [149]
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
81/81 - 2s - 20ms/step - accuracy: 0.3842 - loss: 1.5175 - val_accuracy: 0.4368 - val_loss: 1.3930
Epoch 2/10
81/81 - 0s - 4ms/step - accuracy: 0.4699 - loss: 1.3050 - val_accuracy: 0.5145 - val_loss: 1.2311
Epoch 3/10
81/81 - 0s - 4ms/step - accuracy: 0.5340 - loss: 1.1915 - val_accuracy: 0.5330 - val_loss: 1.1585
Epoch 4/10
81/81 - 0s - 5ms/step - accuracy: 0.5541 - loss: 1.1374 - val_accuracy: 0.5562 - val_loss: 1.1230
Epoch 5/10
81/81 - 1s - 10ms/step - accuracy: 0.5718 - loss: 1.0978 - val_accuracy: 0.5620 - val_loss: 1.1055
Epoch 6/10
81/81 - 0s - 5ms/step - accuracy: 0.5838 - loss: 1.0666 - val_accuracy: 0.5655 - val_loss: 1.0889
Epoch 7/10
81/81 - 0s - 4ms/step - accuracy: 0.5958 - loss: 1.0414 - val_accuracy: 0.5678 - val_loss: 1.0790
Epoch 8/10
81/81 - 1s - 7ms/step - accuracy: 0.6042 - loss: 1.0201 - val_accuracy: 0.5863 - val_loss: 1.0659
Epoch 9/10
81/81 - 1s - 8ms/step - accuracy: 0.6104 - loss: 0.9995 - val_accuracy: 0.5933 - val_loss: 1.0609
Epoch 10/10
81/81 - 1s - 9ms/step - accuracy: 0.6174 - loss: 0.9824 - val_accuracy: 0.5898 - val_loss: 1.0578
Validation Accuracy for Structure [149]: 0.5898

Training MLP Structure: [108]
Epoch 1/10
81/81 - 2s - 25ms/step - accuracy: 0.3602 - loss: 1.5306 - val_accuracy: 0.4171 - val_loss: 1.4314
Epoch 2/10
81/81 - 1s - 12ms/step - accuracy: 0.4722 - loss: 1.3427 - val_accuracy: 0.5064 - val_loss: 1.2618
Epoch 3/10
81/81 - 1s - 8ms/step - accuracy: 0.5216 - loss: 1.2190 - val_accuracy: 0.5342 - val_loss: 1.1837
Epoch 4/10
81/81 - 1s - 7ms/step - accuracy: 0.5421 - loss: 1.1576 - val_accuracy: 0.5469 - val_loss: 1.1431
Epoch 5/10
81/81 - 1s - 7ms/step - accuracy: 0.5710 - loss: 1.1183 - val_accuracy: 0.5620 - val_loss: 1.1188
Epoch 6/10
81/81 - 0s - 6ms/step - accuracy: 0.5656 - loss: 1.0866 - val_accuracy: 0.5481 - val_loss: 1.1062
Epoch 7/10
81/81 - 1s - 10ms/step - accuracy: 0.5884 - loss: 1.0615 - val_accuracy: 0.5759 - val_loss: 1.0870
Epoch 8/10
81/81 - 1s - 7ms/step - accuracy: 0.5954 - loss: 1.0415 - val_accuracy: 0.5747 - val_loss: 1.0864
Epoch 9/10
81/81 - 1s - 7ms/step - accuracy: 0.6058 - loss: 1.0228 - val_accuracy: 0.5782 - val_loss: 1.0707
Epoch 10/10
81/81 - 1s - 9ms/step - accuracy: 0.6131 - loss: 1.0046 - val_accuracy: 0.5759 - val_loss: 1.0679
Validation Accuracy for Structure [108]: 0.5759

Training MLP Structure: [216]
Epoch 1/10
81/81 - 2s - 24ms/step - accuracy: 0.3761 - loss: 1.4971 - val_accuracy: 0.4496 - val_loss: 1.3679
Epoch 2/10
81/81 - 0s - 4ms/step - accuracy: 0.5035 - loss: 1.2756 - val_accuracy: 0.5342 - val_loss: 1.2001
Epoch 3/10
81/81 - 0s - 4ms/step - accuracy: 0.5413 - loss: 1.1707 - val_accuracy: 0.5516 - val_loss: 1.1432
Epoch 4/10
81/81 - 0s - 5ms/step - accuracy: 0.5625 - loss: 1.1143 - val_accuracy: 0.5678 - val_loss: 1.1105
Epoch 5/10
81/81 - 0s - 4ms/step - accuracy: 0.5780 - loss: 1.0758 - val_accuracy: 0.5724 - val_loss: 1.0940
Epoch 6/10
81/81 - 0s - 5ms/step - accuracy: 0.5977 - loss: 1.0465 - val_accuracy: 0.5771 - val_loss: 1.0831
Epoch 7/10
81/81 - 1s - 8ms/step - accuracy: 0.6066 - loss: 1.0202 - val_accuracy: 0.5771 - val_loss: 1.0727
Epoch 8/10
81/81 - 1s - 7ms/step - accuracy: 0.6185 - loss: 0.9970 - val_accuracy: 0.5794 - val_loss: 1.0617
Epoch 9/10
81/81 - 1s - 10ms/step - accuracy: 0.6274 - loss: 0.9744 - val_accuracy: 0.5829 - val_loss: 1.0572
Epoch 10/10
81/81 - 1s - 8ms/step - accuracy: 0.6297 - loss: 0.9566 - val_accuracy: 0.5979 - val_loss: 1.0485
Validation Accuracy for Structure [216]: 0.5979

Training MLP Structure: [149, 108]
Epoch 1/10
```

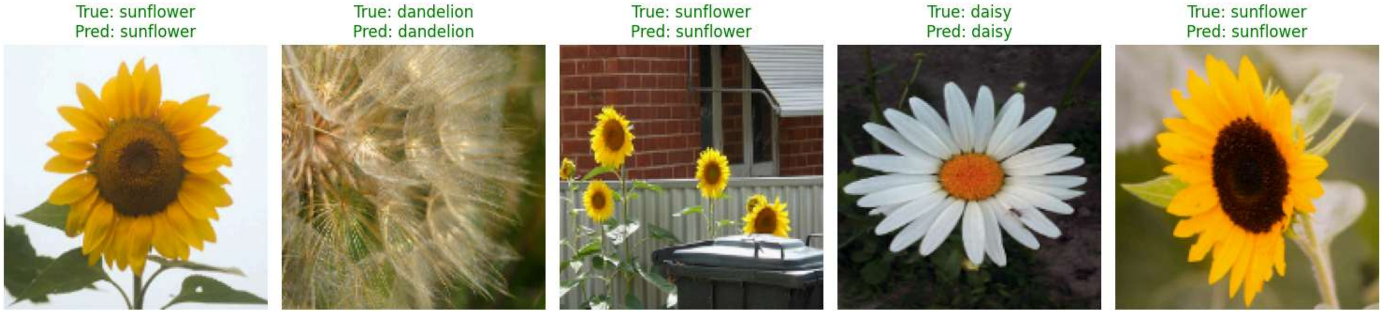
```
81/81 - 3s - 34ms/step - accuracy: 0.3710 - loss: 1.4674 - val_accuracy: 0.4600 - val_loss: 1.2669
Epoch 2/10
81/81 - 1s - 10ms/step - accuracy: 0.5166 - loss: 1.1856 - val_accuracy: 0.5492 - val_loss: 1.1310
Epoch 3/10
81/81 - 1s - 9ms/step - accuracy: 0.5695 - loss: 1.0855 - val_accuracy: 0.5782 - val_loss: 1.0777
Epoch 4/10
81/81 - 1s - 7ms/step - accuracy: 0.5988 - loss: 1.0234 - val_accuracy: 0.5805 - val_loss: 1.0532
Epoch 5/10
81/81 - 1s - 7ms/step - accuracy: 0.6174 - loss: 0.9831 - val_accuracy: 0.5782 - val_loss: 1.0561
Epoch 6/10
81/81 - 0s - 6ms/step - accuracy: 0.6340 - loss: 0.9409 - val_accuracy: 0.5643 - val_loss: 1.0606
Epoch 7/10
81/81 - 0s - 5ms/step - accuracy: 0.6398 - loss: 0.9121 - val_accuracy: 0.5979 - val_loss: 1.0317
Epoch 8/10
81/81 - 1s - 7ms/step - accuracy: 0.6595 - loss: 0.8743 - val_accuracy: 0.5886 - val_loss: 1.0316
Epoch 9/10
81/81 - 0s - 5ms/step - accuracy: 0.6691 - loss: 0.8511 - val_accuracy: 0.6072 - val_loss: 1.0118
Epoch 10/10
81/81 - 1s - 7ms/step - accuracy: 0.6846 - loss: 0.8167 - val_accuracy: 0.6130 - val_loss: 1.0116
Validation Accuracy for Structure [149, 108]: 0.6130

Training MLP Structure: [216, 5]
Epoch 1/10
81/81 - 2s - 24ms/step - accuracy: 0.2614 - loss: 1.5374 - val_accuracy: 0.3569 - val_loss: 1.4496
Epoch 2/10
81/81 - 1s - 13ms/step - accuracy: 0.4147 - loss: 1.3883 - val_accuracy: 0.4519 - val_loss: 1.3460
Epoch 3/10
81/81 - 0s - 6ms/step - accuracy: 0.4795 - loss: 1.2969 - val_accuracy: 0.4832 - val_loss: 1.2840
Epoch 4/10
81/81 - 1s - 10ms/step - accuracy: 0.5189 - loss: 1.2326 - val_accuracy: 0.5145 - val_loss: 1.2313
Epoch 5/10
81/81 - 1s - 6ms/step - accuracy: 0.5293 - loss: 1.1780 - val_accuracy: 0.5261 - val_loss: 1.1971
Epoch 6/10
81/81 - 1s - 7ms/step - accuracy: 0.5533 - loss: 1.1292 - val_accuracy: 0.5469 - val_loss: 1.1577
Epoch 7/10
81/81 - 1s - 9ms/step - accuracy: 0.5737 - loss: 1.0866 - val_accuracy: 0.5504 - val_loss: 1.1297
Epoch 8/10
81/81 - 1s - 8ms/step - accuracy: 0.5853 - loss: 1.0505 - val_accuracy: 0.5574 - val_loss: 1.1097
Epoch 9/10
81/81 - 1s - 7ms/step - accuracy: 0.5992 - loss: 1.0204 - val_accuracy: 0.5585 - val_loss: 1.0983
Epoch 10/10
81/81 - 1s - 8ms/step - accuracy: 0.6093 - loss: 0.9911 - val_accuracy: 0.5701 - val_loss: 1.0767
Validation Accuracy for Structure [216, 5]: 0.5701

Training MLP Structure: [108, 216]
Epoch 1/10
81/81 - 2s - 30ms/step - accuracy: 0.3668 - loss: 1.4697 - val_accuracy: 0.4751 - val_loss: 1.2466
Epoch 2/10
81/81 - 0s - 6ms/step - accuracy: 0.5124 - loss: 1.1820 - val_accuracy: 0.5365 - val_loss: 1.1148
Epoch 3/10
81/81 - 1s - 7ms/step - accuracy: 0.5533 - loss: 1.0881 - val_accuracy: 0.5678 - val_loss: 1.0853
Epoch 4/10
81/81 - 1s - 10ms/step - accuracy: 0.5942 - loss: 1.0279 - val_accuracy: 0.5689 - val_loss: 1.0659
Epoch 5/10
81/81 - 0s - 6ms/step - accuracy: 0.6093 - loss: 0.9811 - val_accuracy: 0.5782 - val_loss: 1.0636
Epoch 6/10
81/81 - 1s - 7ms/step - accuracy: 0.6336 - loss: 0.9366 - val_accuracy: 0.5689 - val_loss: 1.0697
Epoch 7/10
81/81 - 1s - 7ms/step - accuracy: 0.6506 - loss: 0.9025 - val_accuracy: 0.5991 - val_loss: 1.0195
Epoch 8/10
81/81 - 0s - 5ms/step - accuracy: 0.6633 - loss: 0.8720 - val_accuracy: 0.5852 - val_loss: 1.0267
Epoch 9/10
81/81 - 1s - 7ms/step - accuracy: 0.6784 - loss: 0.8326 - val_accuracy: 0.5991 - val_loss: 1.0261
Epoch 10/10
81/81 - 1s - 7ms/step - accuracy: 0.6973 - loss: 0.8026 - val_accuracy: 0.6176 - val_loss: 1.0298
Validation Accuracy for Structure [108, 216]: 0.6176

Training MLP Structure: [149, 108, 5]
Epoch 1/10
81/81 - 2s - 27ms/step - accuracy: 0.3278 - loss: 1.5263 - val_accuracy: 0.4264 - val_loss: 1.4144
Epoch 2/10
81/81 - 0s - 5ms/step - accuracy: 0.4591 - loss: 1.3031 - val_accuracy: 0.4670 - val_loss: 1.2538
Epoch 3/10
81/81 - 0s - 5ms/step - accuracy: 0.4985 - loss: 1.1764 - val_accuracy: 0.4878 - val_loss: 1.1736
Epoch 4/10
81/81 - 1s - 10ms/step - accuracy: 0.5274 - loss: 1.1003 - val_accuracy: 0.5272 - val_loss: 1.1273
Epoch 5/10
81/81 - 1s - 8ms/step - accuracy: 0.5622 - loss: 1.0472 - val_accuracy: 0.5597 - val_loss: 1.0879
Epoch 6/10
81/81 - 1s - 7ms/step - accuracy: 0.6019 - loss: 0.9972 - val_accuracy: 0.5886 - val_loss: 1.0495
Epoch 7/10
81/81 - 1s - 8ms/step - accuracy: 0.6274 - loss: 0.9474 - val_accuracy: 0.6049 - val_loss: 1.0413
Epoch 8/10
81/81 - 1s - 7ms/step - accuracy: 0.6386 - loss: 0.9109 - val_accuracy: 0.5991 - val_loss: 1.0281
Epoch 9/10
```

```
81/81 - 1s - 8ms/step - accuracy: 0.6583 - loss: 0.8828 - val_accuracy: 0.6072 - val_loss: 1.0309
Epoch 10/10
81/81 - 1s - 7ms/step - accuracy: 0.6780 - loss: 0.8425 - val_accuracy: 0.6141 - val_loss: 1.0069
Validation Accuracy for Structure [149, 108, 5]: 0.6141

Training MLP Structure: [216, 108, 5]
Epoch 1/10
81/81 - 2s - 29ms/step - accuracy: 0.3846 - loss: 1.4948 - val_accuracy: 0.4670 - val_loss: 1.3224
Epoch 2/10
81/81 - 0s - 5ms/step - accuracy: 0.4927 - loss: 1.2392 - val_accuracy: 0.4948 - val_loss: 1.1940
Epoch 3/10
81/81 - 0s - 5ms/step - accuracy: 0.5201 - loss: 1.1358 - val_accuracy: 0.5191 - val_loss: 1.1348
Epoch 4/10
81/81 - 0s - 5ms/step - accuracy: 0.5456 - loss: 1.0685 - val_accuracy: 0.5284 - val_loss: 1.1054
Epoch 5/10
81/81 - 0s - 6ms/step - accuracy: 0.5757 - loss: 1.0113 - val_accuracy: 0.5388 - val_loss: 1.0867
Epoch 6/10
81/81 - 1s - 7ms/step - accuracy: 0.6031 - loss: 0.9572 - val_accuracy: 0.5620 - val_loss: 1.0614
Epoch 7/10
81/81 - 0s - 5ms/step - accuracy: 0.6305 - loss: 0.9100 - val_accuracy: 0.5829 - val_loss: 1.0330
Epoch 8/10
81/81 - 0s - 6ms/step - accuracy: 0.6521 - loss: 0.8700 - val_accuracy: 0.5898 - val_loss: 1.0210
Epoch 9/10
81/81 - 0s - 4ms/step - accuracy: 0.6718 - loss: 0.8307 - val_accuracy: 0.5875 - val_loss: 1.0397
Epoch 10/10
81/81 - 1s - 8ms/step - accuracy: 0.6880 - loss: 0.7983 - val_accuracy: 0.6188 - val_loss: 1.0135
Validation Accuracy for Structure [216, 108, 5]: 0.6188

Training MLP Structure: [216, 108, 5]
Epoch 1/10
81/81 - 2s - 26ms/step - accuracy: 0.2961 - loss: 1.5197 - val_accuracy: 0.4148 - val_loss: 1.4279
Epoch 2/10
81/81 - 0s - 6ms/step - accuracy: 0.4452 - loss: 1.3696 - val_accuracy: 0.4461 - val_loss: 1.3300
Epoch 3/10
81/81 - 1s - 7ms/step - accuracy: 0.4795 - loss: 1.2585 - val_accuracy: 0.4797 - val_loss: 1.2345
Epoch 4/10
81/81 - 1s - 9ms/step - accuracy: 0.5274 - loss: 1.1604 - val_accuracy: 0.5214 - val_loss: 1.1718
Epoch 5/10
81/81 - 1s - 8ms/step - accuracy: 0.5591 - loss: 1.0945 - val_accuracy: 0.5284 - val_loss: 1.1463
Epoch 6/10
81/81 - 1s - 8ms/step - accuracy: 0.5795 - loss: 1.0323 - val_accuracy: 0.5342 - val_loss: 1.1377
Epoch 7/10
81/81 - 1s - 15ms/step - accuracy: 0.5911 - loss: 0.9892 - val_accuracy: 0.5435 - val_loss: 1.1046
Epoch 8/10
81/81 - 1s - 8ms/step - accuracy: 0.6127 - loss: 0.9468 - val_accuracy: 0.5527 - val_loss: 1.1044
Epoch 9/10
81/81 - 1s - 15ms/step - accuracy: 0.6236 - loss: 0.9089 - val_accuracy: 0.5632 - val_loss: 1.0867
Epoch 10/10
81/81 - 1s - 7ms/step - accuracy: 0.6398 - loss: 0.8779 - val_accuracy: 0.5504 - val_loss: 1.1252
Validation Accuracy for Structure [216, 108, 5]: 0.5504

Best MLP Structure: [216, 108, 5]
Best Validation Accuracy: 0.6188

=== Question 2.3: Evaluate Best Model ===
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step

Test Accuracy: 0.5694
Precision: 0.5843
Recall: 0.5694
F1 Score: 0.5607
```





```
=== Question 2.4: Timing ===
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Average Inference Time: 0.0002 seconds

=== Question 2.5: Visualize Predictions ===
Correctly classified images:
```



```
Incorrectly classified images:
```