Exp 3

```
[6] from sklearn.datasets import fetch_olivetti_faces
    from sklearn.model_selection import train_test_split
    from keras.models import Sequential
    from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
    from keras.utils import to_categorical
    import numpy as np

    # 1. Load dataset
    faces = fetch_olivetti_faces()
    X, y_raw = faces.images, faces.target  # X: (400, 64, 64), y: (400,)

    # 2. Reshape and preprocess
    X = X.reshape(-1, 64, 64, 1).astype('float32')  # Already scaled 0-1
    y = to_categorical(y_raw, num_classes=40)       # One-hot encoding

    # 3. Train-test split with stratification
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, stratify=y_raw, random_state=42
    )

    # 4. Build CNN model
    model = Sequential([
        Conv2D(32, (3,3), activation='relu', input_shape=(64,64,1)),
        MaxPooling2D((2,2)),
        Dropout(0.25),

        Conv2D(64, (3,3), activation='relu'),
        MaxPooling2D((2,2)),
        Dropout(0.25),

        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(40, activation='softmax')  # 40 classes
    ])

    # 5. Compile model
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    # 6. Train model
    history = model.fit(
        X_train, y_train,
        epochs=20,
        batch_size=32,
        validation_data=(X_test, y_test),
        verbose=1
    )

    # 7. Evaluate model
    test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
```

```
)

# 7. Evaluate model
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
print(f"Test accuracy: {test_acc:.4f}")
```

Output:

```
Downloading Olivetti faces from https://ndownloader.figshare.com/files/5976027 to /root/scikit_learn_data
Epoch 1/20
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
10/10 ━━━━━━━━━━━━━━━━━━━━ 5s 338ms/step - accuracy: 0.0074 - loss: 3.8685 - val_accuracy: 0.0375 - val_loss: 3.6889
Epoch 2/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 4s 201ms/step - accuracy: 0.0310 - loss: 3.6892 - val_accuracy: 0.0250 - val_loss: 3.6887
Epoch 3/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 190ms/step - accuracy: 0.0384 - loss: 3.6887 - val_accuracy: 0.0250 - val_loss: 3.6887
Epoch 4/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 197ms/step - accuracy: 0.0494 - loss: 3.6867 - val_accuracy: 0.0250 - val_loss: 3.6887
Epoch 5/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 204ms/step - accuracy: 0.0280 - loss: 3.6862 - val_accuracy: 0.0500 - val_loss: 3.6870
Epoch 6/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 4s 312ms/step - accuracy: 0.0243 - loss: 3.6858 - val_accuracy: 0.0875 - val_loss: 3.6832
Epoch 7/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 4s 192ms/step - accuracy: 0.0613 - loss: 3.6820 - val_accuracy: 0.0625 - val_loss: 3.6843
Epoch 8/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 204ms/step - accuracy: 0.0621 - loss: 3.6716 - val_accuracy: 0.1625 - val_loss: 3.6605
Epoch 9/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 210ms/step - accuracy: 0.0465 - loss: 3.6398 - val_accuracy: 0.1500 - val_loss: 3.6432
Epoch 10/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 3s 272ms/step - accuracy: 0.0426 - loss: 3.6588 - val_accuracy: 0.2250 - val_loss: 3.6246
Epoch 11/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 4s 195ms/step - accuracy: 0.0861 - loss: 3.5832 - val_accuracy: 0.3000 - val_loss: 3.5714
Epoch 12/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 195ms/step - accuracy: 0.1690 - loss: 3.4971 - val_accuracy: 0.4000 - val_loss: 3.4504
Epoch 13/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 3s 267ms/step - accuracy: 0.2244 - loss: 3.3027 - val_accuracy: 0.4375 - val_loss: 3.1902
Epoch 14/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 5s 255ms/step - accuracy: 0.3483 - loss: 3.0130 - val_accuracy: 0.4875 - val_loss: 2.7733
Epoch 15/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 207ms/step - accuracy: 0.3689 - loss: 2.6668 - val_accuracy: 0.6125 - val_loss: 2.3907
Epoch 16/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 196ms/step - accuracy: 0.4959 - loss: 2.1070 - val_accuracy: 0.6500 - val_loss: 2.0148
Epoch 17/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 3s 205ms/step - accuracy: 0.5682 - loss: 1.7577 - val_accuracy: 0.7500 - val_loss: 1.4455
Epoch 18/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 3s 217ms/step - accuracy: 0.6255 - loss: 1.3586 - val_accuracy: 0.8125 - val_loss: 1.0426
Epoch 19/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 3s 258ms/step - accuracy: 0.7642 - loss: 0.9412 - val_accuracy: 0.8500 - val_loss: 0.8799
Epoch 20/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 195ms/step - accuracy: 0.7687 - loss: 0.8281 - val_accuracy: 0.8625 - val_loss: 0.6950
Test accuracy: 0.8625
```