In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from PIL import Image
import os
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical

from keras.models import Sequential, load_model
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
```

In [2]:

```
data = []
labels = []
classes = 6
cur_path = os.getcwd()
```

In [3]:

```
for i in range(classes):
    path = os.path.join('/content/drive/MyDrive/datasets/train', str(i))
    images = os.listdir(path)

    for a in images:
        try:
            image = Image.open(os.path.join(path, a))
            image = image.resize((30, 30))
            image_array = np.array(image)
            data.append(image_array)
            labels.append(i)
        except Exception as e:
            print("Error loading image:", e)

# Converting lists into numpy arrays
data = np.array(data)
labels = np.array(labels)

print("Data shape:", data.shape)
print("Labels shape:", labels.shape)
```

```
Data shape: (2740, 30, 30, 3)
Labels shape: (2740,)
```

In [85]:

```
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_
state=50)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

# Converting the labels into one hot encoding
y_train = to_categorical(y_train, 6)
y_test = to_categorical(y_test, 6)
```

```
(2192, 30, 30, 3) (548, 30, 30, 3) (2192,) (548,)
```

In [86]:

```
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import BatchNormalization
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense
from keras.optimizers import RMSprop
```

```python
# Data Augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

datagen.fit(X_train)

# Define the model
model = Sequential()
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', input_shape=X_train.shape[1:]))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))

model.add(Conv2D(filters=256, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))

model.add(Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(rate=0.5))
model.add(Dense(6, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer=RMSprop(lr=0.0001), metrics=['accuracy'])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the
legacy optimizer, e.g.,tf.keras.optimizers.legacy.RMSprop.
```

In [87]:

```python
epochs = 60
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test, y_test))
```

```
Epoch 1/60
69/69 [==============================] - 27s 375ms/step - loss: 2.3303 - accuracy: 0.2089
- val_loss: 1.6769 - val_accuracy: 0.2920
Epoch 2/60
69/69 [==============================] - 27s 398ms/step - loss: 1.8833 - accuracy: 0.2641
- val_loss: 1.8591 - val_accuracy: 0.3029
Epoch 3/60
69/69 [==============================] - 25s 361ms/step - loss: 1.8809 - accuracy: 0.2692
- val_loss: 4.9901 - val_accuracy: 0.1624
Epoch 4/60
69/69 [==============================] - 26s 375ms/step - loss: 1.7365 - accuracy: 0.3353
- val_loss: 1.9926 - val_accuracy: 0.2573
Epoch 5/60
69/69 [==============================] - 26s 371ms/step - loss: 1.5735 - accuracy: 0.3837
- val_loss: 1.3894 - val_accuracy: 0.4836
Epoch 6/60
69/69 [==============================] - 26s 377ms/step - loss: 1.4760 - accuracy: 0.4142
- val_loss: 1.6973 - val_accuracy: 0.3358
Epoch 7/60
69/69 [==============================] - 26s 374ms/step - loss: 1.4130 - accuracy: 0.4521
```

```
- val_loss: 1.4712 - val_accuracy: 0.3905
Epoch 8/60
69/69 [==============================] - 25s 359ms/step - loss: 1.2969 - accuracy: 0.5114
- val_loss: 1.0980 - val_accuracy: 0.5712
Epoch 9/60
69/69 [==============================] - 25s 360ms/step - loss: 1.2541 - accuracy: 0.5105
- val_loss: 2.1193 - val_accuracy: 0.3120
Epoch 10/60
69/69 [==============================] - 26s 373ms/step - loss: 1.2235 - accuracy: 0.5406
- val_loss: 1.2774 - val_accuracy: 0.4982
Epoch 11/60
69/69 [==============================] - 26s 371ms/step - loss: 1.1180 - accuracy: 0.5598
- val_loss: 2.9232 - val_accuracy: 0.2883
Epoch 12/60
69/69 [==============================] - 26s 374ms/step - loss: 1.0671 - accuracy: 0.5953
- val_loss: 1.3222 - val_accuracy: 0.4872
Epoch 13/60
69/69 [==============================] - 24s 354ms/step - loss: 1.0062 - accuracy: 0.6223
- val_loss: 0.9935 - val_accuracy: 0.6478
Epoch 14/60
69/69 [==============================] - 25s 365ms/step - loss: 0.9606 - accuracy: 0.6296
- val_loss: 0.9726 - val_accuracy: 0.6150
Epoch 15/60
69/69 [==============================] - 27s 388ms/step - loss: 0.9464 - accuracy: 0.6478
- val_loss: 0.8716 - val_accuracy: 0.6825
Epoch 16/60
69/69 [==============================] - 26s 375ms/step - loss: 0.8212 - accuracy: 0.7035
- val_loss: 0.9037 - val_accuracy: 0.7044
Epoch 17/60
69/69 [==============================] - 25s 369ms/step - loss: 0.7656 - accuracy: 0.7181
- val_loss: 0.7270 - val_accuracy: 0.7646
Epoch 18/60
69/69 [==============================] - 25s 367ms/step - loss: 0.7764 - accuracy: 0.7226
- val_loss: 2.6612 - val_accuracy: 0.3011
Epoch 19/60
69/69 [==============================] - 24s 345ms/step - loss: 0.7217 - accuracy: 0.7359
- val_loss: 0.6306 - val_accuracy: 0.7883
Epoch 20/60
69/69 [==============================] - 25s 366ms/step - loss: 0.6401 - accuracy: 0.7669
- val_loss: 0.6655 - val_accuracy: 0.7518
Epoch 21/60
69/69 [==============================] - 26s 377ms/step - loss: 0.6525 - accuracy: 0.7728
- val_loss: 0.5532 - val_accuracy: 0.8193
Epoch 22/60
69/69 [==============================] - 26s 384ms/step - loss: 0.5704 - accuracy: 0.7993
- val_loss: 0.7034 - val_accuracy: 0.7518
Epoch 23/60
69/69 [==============================] - 26s 380ms/step - loss: 0.5582 - accuracy: 0.8047
- val_loss: 1.2598 - val_accuracy: 0.5821
Epoch 24/60
69/69 [==============================] - 25s 369ms/step - loss: 0.5585 - accuracy: 0.8043
- val_loss: 0.6514 - val_accuracy: 0.7792
Epoch 25/60
69/69 [==============================] - 25s 357ms/step - loss: 0.4903 - accuracy: 0.8353
- val_loss: 0.9205 - val_accuracy: 0.6843
Epoch 26/60
69/69 [==============================] - 25s 359ms/step - loss: 0.4693 - accuracy: 0.8326
- val_loss: 0.7547 - val_accuracy: 0.7482
Epoch 27/60
69/69 [==============================] - 26s 374ms/step - loss: 0.4200 - accuracy: 0.8467
- val_loss: 0.4880 - val_accuracy: 0.8157
Epoch 28/60
69/69 [==============================] - 33s 474ms/step - loss: 0.4731 - accuracy: 0.8344
- val_loss: 0.4782 - val_accuracy: 0.8449
Epoch 29/60
69/69 [==============================] - 26s 371ms/step - loss: 0.3620 - accuracy: 0.8755
- val_loss: 0.6214 - val_accuracy: 0.8175
Epoch 30/60
69/69 [==============================] - 26s 372ms/step - loss: 0.3701 - accuracy: 0.8718
- val_loss: 0.6055 - val_accuracy: 0.8011
Epoch 31/60
69/69 [==============================] - 26s 372ms/step - loss: 0.3519 - accuracy: 0.8764
```

```
- val_loss: 0.8811 - val_accuracy: 0.7482
Epoch 32/60
69/69 [==============================] - 25s 368ms/step - loss: 0.3312 - accuracy: 0.8859
- val_loss: 0.9219 - val_accuracy: 0.7026
Epoch 33/60
69/69 [==============================] - 25s 357ms/step - loss: 0.3123 - accuracy: 0.8855
- val_loss: 0.5370 - val_accuracy: 0.8303
Epoch 34/60
69/69 [==============================] - 25s 367ms/step - loss: 0.3196 - accuracy: 0.8996
- val_loss: 0.6825 - val_accuracy: 0.7901
Epoch 35/60
69/69 [==============================] - 25s 367ms/step - loss: 0.2582 - accuracy: 0.9069
- val_loss: 0.7597 - val_accuracy: 0.7701
Epoch 36/60
69/69 [==============================] - 25s 364ms/step - loss: 0.2876 - accuracy: 0.9001
- val_loss: 0.5695 - val_accuracy: 0.8321
Epoch 37/60
69/69 [==============================] - 25s 364ms/step - loss: 0.2463 - accuracy: 0.9179
- val_loss: 0.4855 - val_accuracy: 0.8595
Epoch 38/60
69/69 [==============================] - 24s 349ms/step - loss: 0.2238 - accuracy: 0.9224
- val_loss: 0.7956 - val_accuracy: 0.7938
Epoch 39/60
69/69 [==============================] - 26s 378ms/step - loss: 0.2410 - accuracy: 0.9256
- val_loss: 0.5300 - val_accuracy: 0.8230
Epoch 40/60
69/69 [==============================] - 25s 368ms/step - loss: 0.2491 - accuracy: 0.9115
- val_loss: 0.4590 - val_accuracy: 0.8704
Epoch 41/60
69/69 [==============================] - 27s 397ms/step - loss: 0.2225 - accuracy: 0.9302
- val_loss: 0.5543 - val_accuracy: 0.8449
Epoch 42/60
69/69 [==============================] - 26s 370ms/step - loss: 0.2250 - accuracy: 0.9266
- val_loss: 0.7156 - val_accuracy: 0.7828
Epoch 43/60
69/69 [==============================] - 24s 352ms/step - loss: 0.1863 - accuracy: 0.9366
- val_loss: 0.7106 - val_accuracy: 0.7956
Epoch 44/60
69/69 [==============================] - 26s 371ms/step - loss: 0.2259 - accuracy: 0.9302
- val_loss: 0.5069 - val_accuracy: 0.8485
Epoch 45/60
69/69 [==============================] - 26s 372ms/step - loss: 0.1814 - accuracy: 0.9484
- val_loss: 0.7871 - val_accuracy: 0.7774
Epoch 46/60
69/69 [==============================] - 25s 366ms/step - loss: 0.2045 - accuracy: 0.9348
- val_loss: 0.6298 - val_accuracy: 0.8120
Epoch 47/60
69/69 [==============================] - 25s 364ms/step - loss: 0.2063 - accuracy: 0.9357
- val_loss: 0.6957 - val_accuracy: 0.8175
Epoch 48/60
69/69 [==============================] - 25s 367ms/step - loss: 0.1718 - accuracy: 0.9443
- val_loss: 0.7217 - val_accuracy: 0.7938
Epoch 49/60
69/69 [==============================] - 25s 361ms/step - loss: 0.1641 - accuracy: 0.9457
- val_loss: 0.7285 - val_accuracy: 0.7938
Epoch 50/60
69/69 [==============================] - 26s 378ms/step - loss: 0.1687 - accuracy: 0.9475
- val_loss: 0.5529 - val_accuracy: 0.8394
Epoch 51/60
69/69 [==============================] - 26s 372ms/step - loss: 0.1490 - accuracy: 0.9557
- val_loss: 0.6367 - val_accuracy: 0.8376
Epoch 52/60
69/69 [==============================] - 26s 375ms/step - loss: 0.1829 - accuracy: 0.9443
- val_loss: 0.4992 - val_accuracy: 0.8504
Epoch 53/60
69/69 [==============================] - 26s 382ms/step - loss: 0.1475 - accuracy: 0.9557
- val_loss: 0.7660 - val_accuracy: 0.8011
Epoch 54/60
69/69 [==============================] - 26s 383ms/step - loss: 0.1681 - accuracy: 0.9439
- val_loss: 1.0501 - val_accuracy: 0.7500
Epoch 55/60
69/69 [==============================] - 24s 345ms/step - loss: 0.1389 - accuracy: 0.9562
```

```
- val_loss: 1.0402 - val_accuracy: 0.7883
Epoch 56/60
69/69 [==============================] - 25s 361ms/step - loss: 0.1459 - accuracy: 0.9526
- val_loss: 1.0213 - val_accuracy: 0.7464
Epoch 57/60
69/69 [==============================] - 25s 366ms/step - loss: 0.1568 - accuracy: 0.9526
- val_loss: 0.5815 - val_accuracy: 0.8449
Epoch 58/60
69/69 [==============================] - 26s 378ms/step - loss: 0.1447 - accuracy: 0.9535
- val_loss: 0.8342 - val_accuracy: 0.7938
Epoch 59/60
69/69 [==============================] - 25s 367ms/step - loss: 0.1570 - accuracy: 0.9494
- val_loss: 0.4687 - val_accuracy: 0.8796
Epoch 60/60
69/69 [==============================] - 25s 362ms/step - loss: 0.1414 - accuracy: 0.9612
- val_loss: 0.5257 - val_accuracy: 0.8723
```

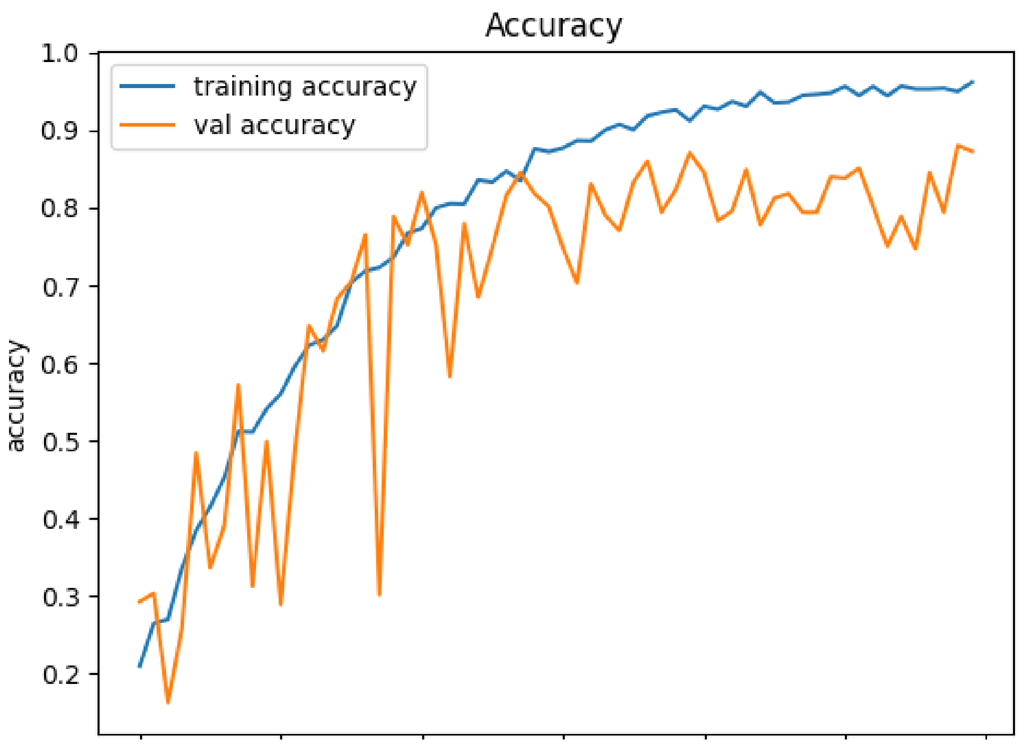
In [111]:

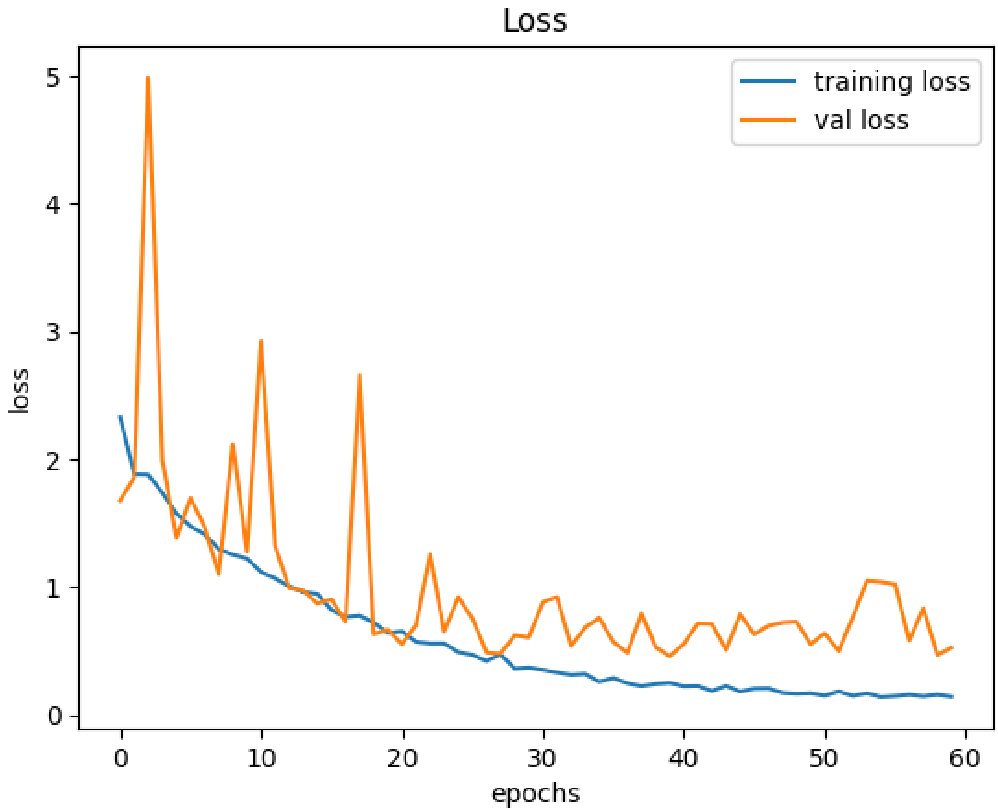
```
model.save("model2.h5")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: Y
ou are saving your model as an HDF5 file via `model.save()`. This file format is consider
ed legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model
.keras')`.
  saving_api.save_model(
```

In [88]:

```
# Plotting graphs for accuracy
plt.figure(0)
plt.plot(history.history['accuracy'], label='training accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()

plt.figure(1)
plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.title('Loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```





In [99]:

```
sample_img = Image.open('/content/WhatsApp Image 2024-04-03 at 9.03.12 PM.jpeg')
plt.imshow(sample_img)
plt.title("Sample Image (Before Preprocessing)\nLabel: ")
plt.axis('off')
plt.show()

# Preprocess the sample image
sample_img = sample_img.resize((30, 30))
sample_img_array = np.array(sample_img)
sample_img_array = np.expand_dims(sample_img_array, axis=0)
```

Sample Image (Before Preprocessing)
Label:

In [100]:

```
class_labels = {0: '10 rupee', 1: '20 rupee', 2: '50 rupee', 4: '100 rupee', 5: '200 rup
ee', 6: '500 rupee' }

# Predict the class of the sample image
predicted_class = np.argmax(model.predict(sample_img_array))
```

```
predicted_label = class_labels[predicted_class]

print("Predicted Label:", predicted_label)
```

```
1/1 [==============================] - 0s 33ms/step
Predicted Label: 20 rupee
```

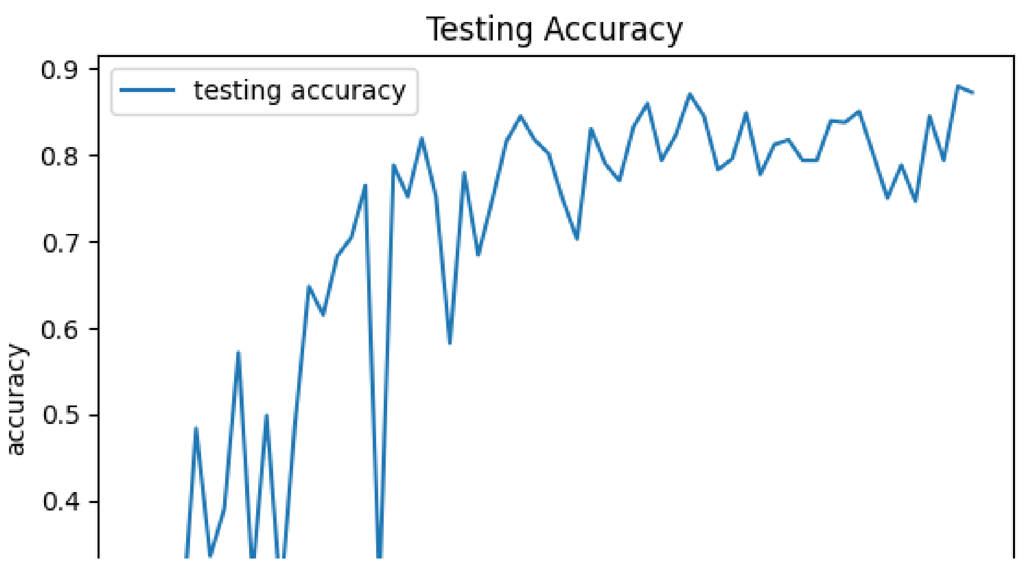
In [96]:

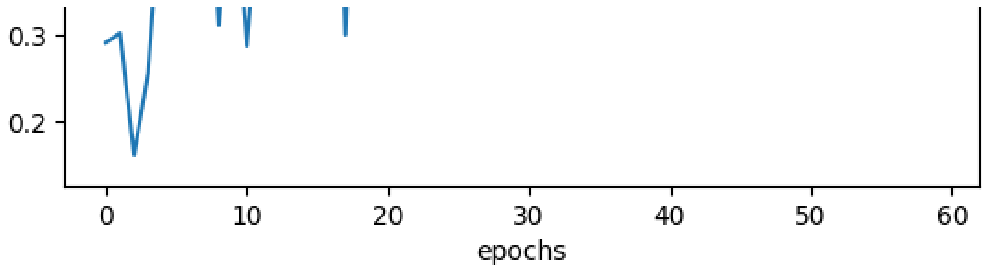
```
#Display the preprocessed image and its predicted class
plt.imshow(sample_img_array[0])
plt.title("Sample Image (Preprocessed)\nPredicted Class: " + str(predicted_class) + "\nP
redicted Label: " + predicted_label)
plt.axis('off')
plt.show()
```

Sample Image (Preprocessed)
Predicted Class: 1
Predicted Label: 20 rupee

In [101]:

```
# Plotting graph for testing accuracy
plt.plot(history.history['val_accuracy'], label='testing accuracy')
plt.title('Testing Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()
```





In [102]:

```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test)

# Print the test loss and accuracy
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
18/18 [==============================] - 1s 31ms/step - loss: 0.5257 - accuracy: 0.8723
Test Loss: 0.5257259011268616
Test Accuracy: 0.8722627758979797
```

In [105]:

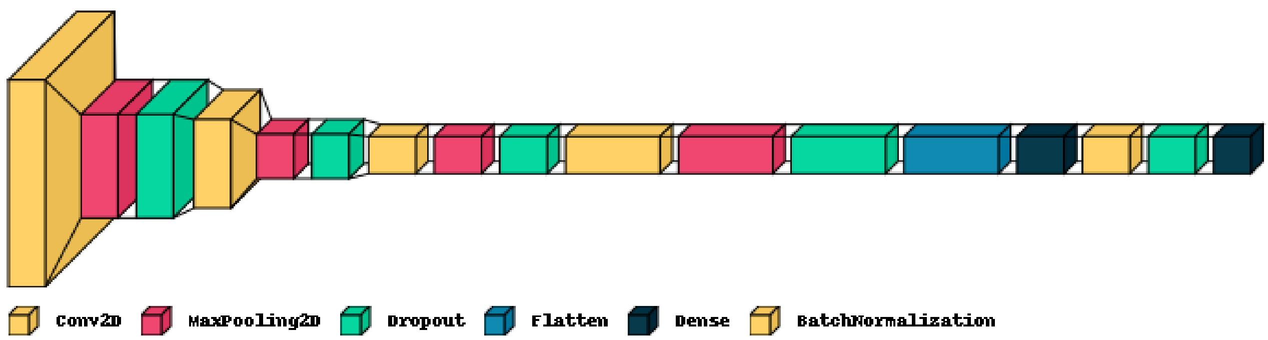
```
pip install visualkeras
```

```
Collecting visualkeras
  Downloading visualkeras-0.0.2-py3-none-any.whl (12 kB)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (
from visualkeras) (9.4.0)
Requirement already satisfied: numpy>=1.18.1 in /usr/local/lib/python3.10/dist-packages (
from visualkeras) (1.25.2)
Collecting aggdraw>=1.3.11 (from visualkeras)
  Downloading aggdraw-1.3.18.post0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64
.whl (993 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 993.8/993.8 kB 18.0 MB/s eta 0:00:00
Installing collected packages: aggdraw, visualkeras
Successfully installed aggdraw-1.3.18.post0 visualkeras-0.0.2
```

In [106]:

```
import visualkeras
from PIL import ImageFont
visualkeras.layered_view(model, legend=True)
```

Out[106]:



In [108]:

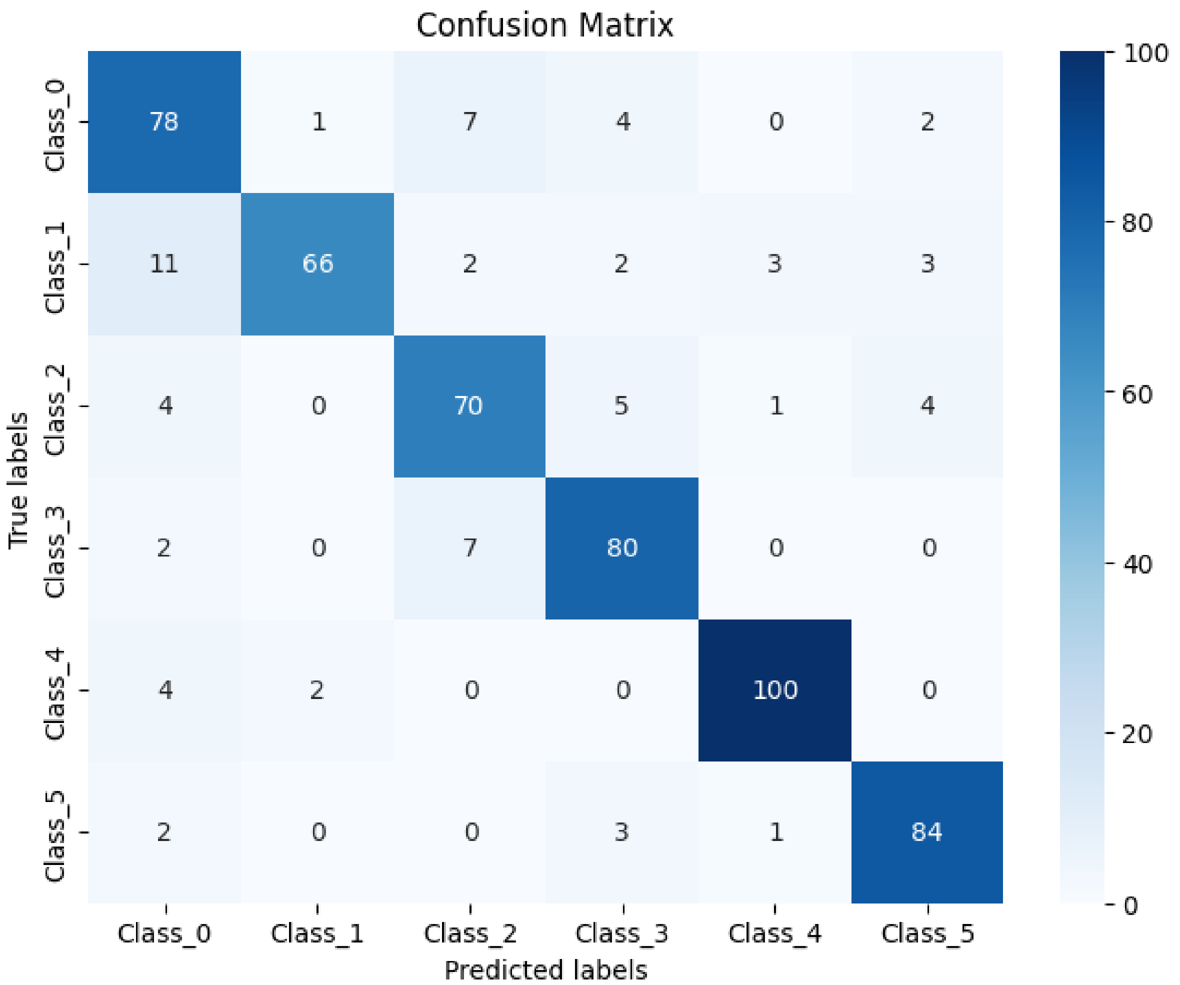
```
evaluation = model.evaluate(X_test, y_test)

# Print the evaluation metrics
print("Evaluation Loss:", evaluation[0])
print("Evaluation Accuracy:", evaluation[1])
```

```
18/18 [==============================] - 1s 33ms/step - loss: 0.5257 - accuracy: 0.8723
Evaluation Loss: 0.5257259011268616
Evaluation Accuracy: 0.8722627758979797
```

In [110]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Predict probabilities for each class for the test set
y_pred_probs = model.predict(X_test)

# Convert probabilities to class labels
y_pred = np.argmax(y_pred_probs, axis=1)

# Generate confusion matrix
conf_matrix = confusion_matrix(np.argmax(y_test, axis=1), y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=["Class_0", "Cla
ss_1", "Class_2", "Class_3", "Class_4", "Class_5"], yticklabels=["Class_0", "Class_1", "
Class_2", "Class_3", "Class_4", "Class_5"])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

```
18/18 [==============================] - 2s 112ms/step
```



In [ ]: