Import Libraries

In [79]:
```
# Import the necessary libraries
import os
import numpy as np
import cv2
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, precision_recall_curve
from sklearn.preprocessing import label_binarize
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, regularizers
from tensorflow.keras.applications import ResNet50
import matplotlib.pyplot as plt
```

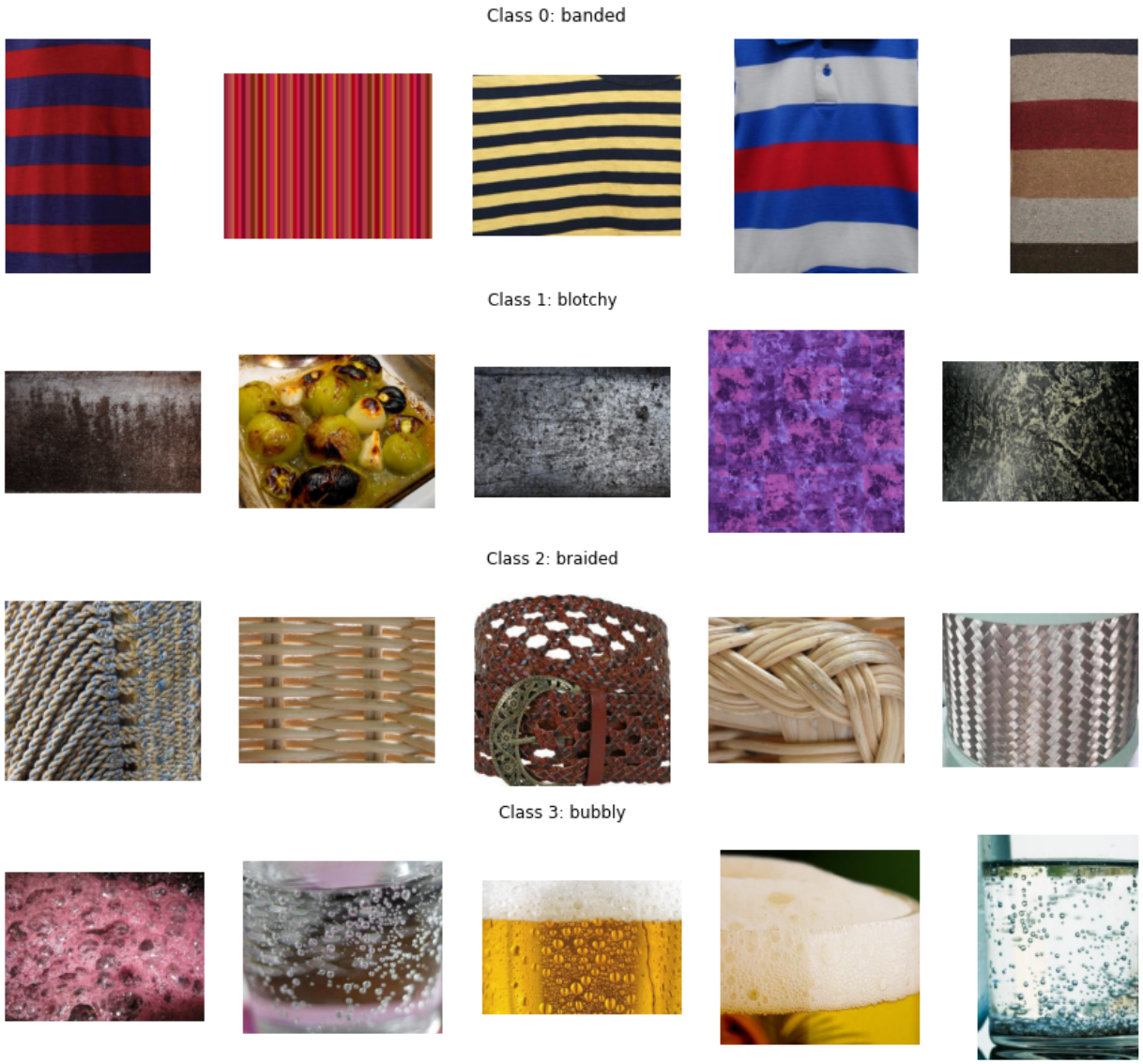
Load Dataset

In [80]:
```
# Load the image data
image_dir = './dtd-r1.0.1/dtd/images/'
class_names = os.listdir(image_dir)

X = []
y = []

for i, class_name in enumerate(class_names):
    class_dir = os.path.join(image_dir, class_name)
    for image_path in os.listdir(class_dir):
        image = keras.preprocessing.image.load_img(os.path.join(class_dir, image_path))
        image_array = keras.preprocessing.image.img_to_array(image)
        X.append(image_array)
        y.append(i)

# Convert the lists to numpy arrays
X = np.array(X)
y = np.array(y)
```

```
<ipython-input-80-ddfe85779b19>:17: VisibleDeprecationWarning: Creating an ndarray from ragged
nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengt
hs or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when cre
ating the ndarray.
  X = np.array(X)
```

Explore Dataset

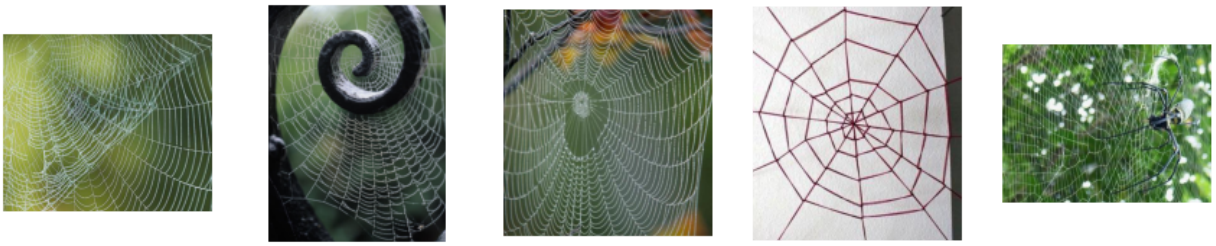
In [86]:
```
# Define the number of images to preview per class
num_images_per_class = 5

# Preview images for each class
for class_label, class_name in enumerate(class_names):
    # Find the indices of images belonging to the current class
    class_indices = np.where(y == class_label)[0]

    # Check if there are images available for the current class
    if len(class_indices) == 0:
        print(f"No images available for Class {class_label}: {class_name}")
        continue

    # Randomly select a subset of images to preview
    preview_indices = np.random.choice(class_indices, size=min(num_images_per_class, len(class_

    # Plot the preview images
    fig, axes = plt.subplots(1, len(preview_indices), figsize=(15, 3))
    for i, index in enumerate(preview_indices):
        image = X[index].astype(np.uint8)
        axes[i].imshow(image)
        axes[i].axis('off')

    # Set the title of the plot as the class name
    fig.suptitle(f'Class {class_label}: {class_name}')

    # Show the plot
    plt.show()
```



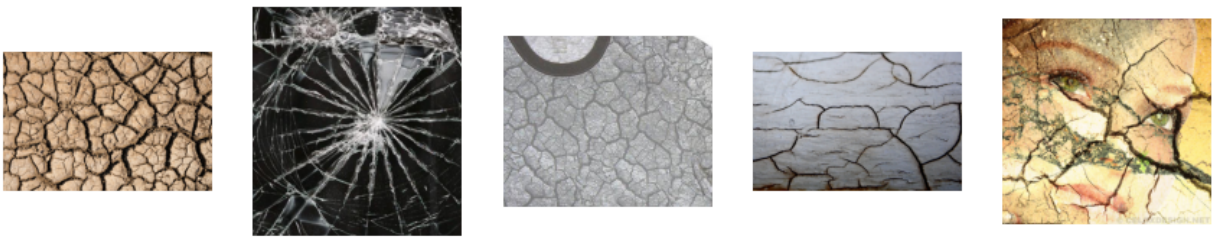
Class 4: bumpy

Class 5: chequered

Class 6: cobwebbed



Class 7: cracked



Class 8: crosshatched

Class 9: crystalline

Class 10: dotted

Class 11: fibrous

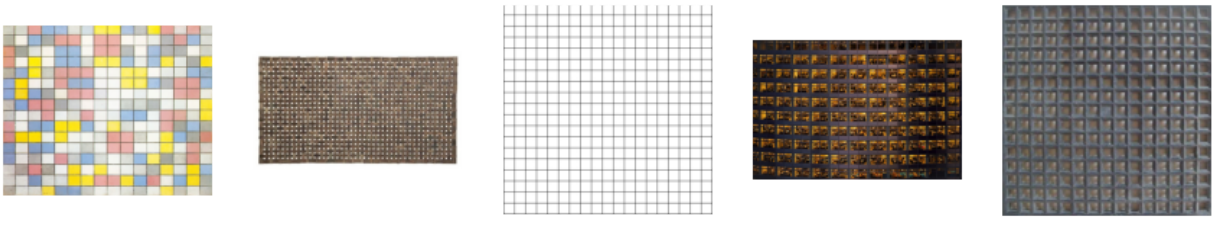
Class 12: flecked

Class 13: freckled

Class 14: frilly

Class 15: gauzy

Class 16: grid



Class 17: grooved

Class 18: honeycombed

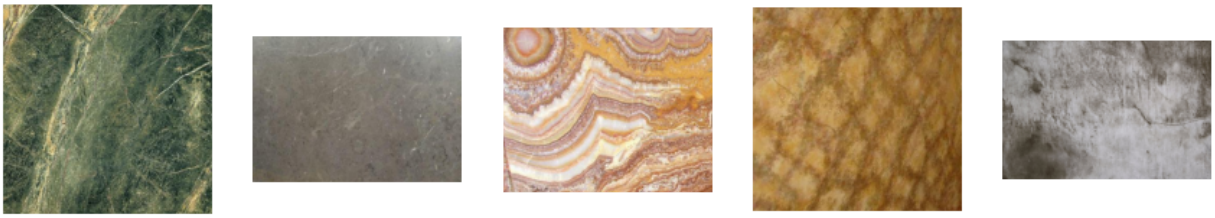
Class 19: interlaced

Class 20: knitted

Class 21: lacelike

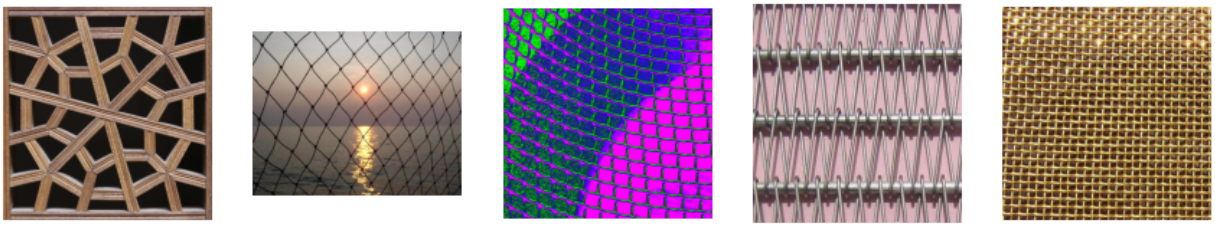
Class 22: lined

Class 23: marbled



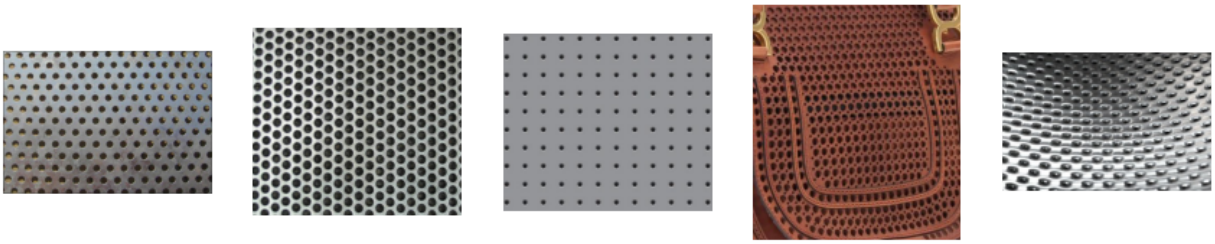
Class 24: matted

Class 25: meshed

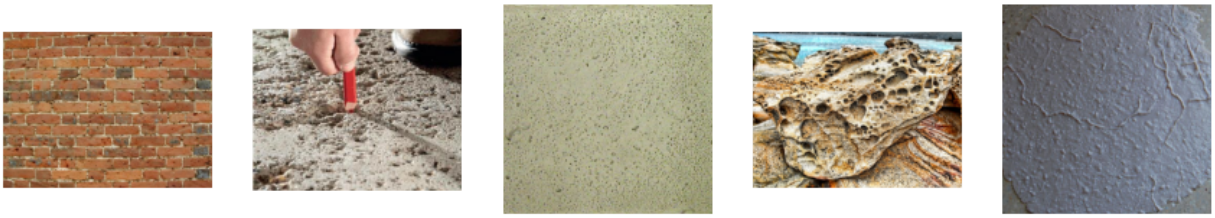


Class 26: paisley

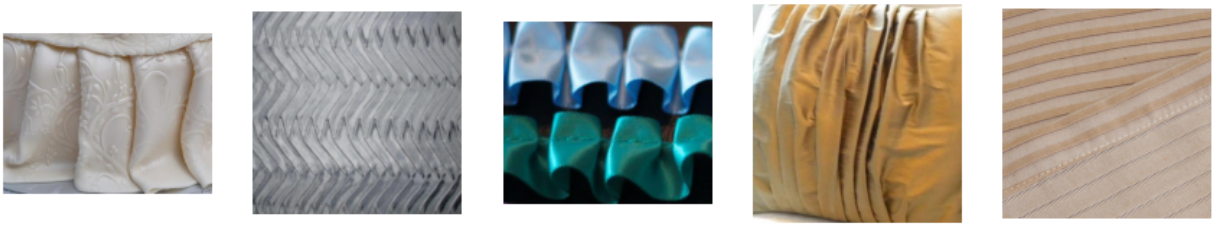
Class 27: perforated



Class 28: pitted



Class 29: pleated



Class 30: polka-dotted

Class 31: porous

Class 32: potholed

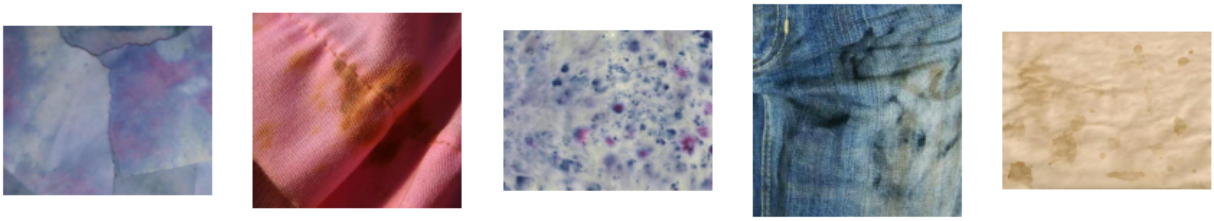
Class 33: scaly

Class 34: smeared

Class 35: spiralled

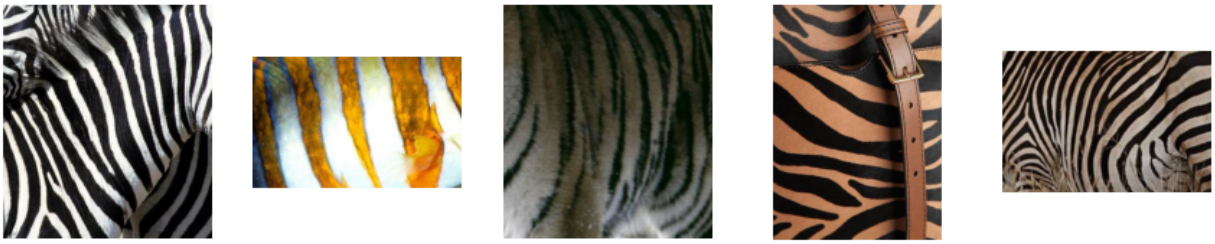
Class 36: sprinkled

Class 37: stained



Class 38: stratified

Class 39: striped



Class 40: studded

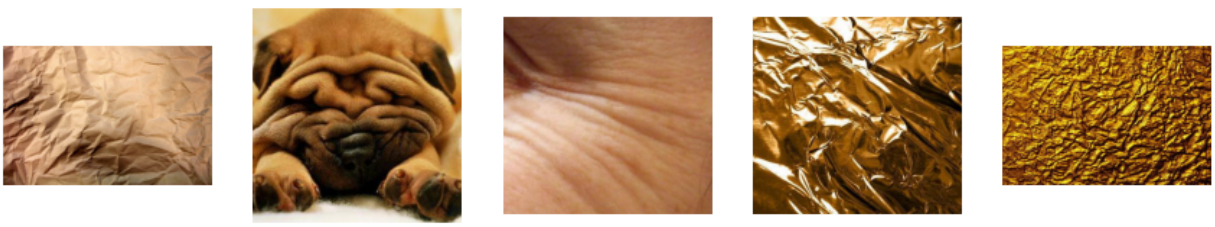
Class 41: swirly

Class 42: veined

Class 43: waffled

Class 44: woven

Class 45: wrinkled



In [87]:
```
# Count the total number of images
total_images = len(X)
print("Total number of images:", total_images)

# Determine the image size and type
image_size = X[0].shape
image_type = type(X[1][0][0][0])
print("Image size:", image_size)
print("Image type:", image_type)
```

```
Total number of images: 5640
Image size: (300, 300, 3)
Image type: <class 'numpy.float32'>
```

In [95]:
```
# Check the shape of the data
print("X shape:", X.shape)
print("y shape:", y.shape)

# Check the number of classes
num_classes = len(class_names)
print("Number of classes:", num_classes)

# Check the distribution of classes
class_counts = np.bincount(y)
print("Class counts:", class_counts)
```

```
X shape: (5640,)
y shape: (5640,)
Number of classes: 47
Class counts: [120 120 120 120 120 120 120 120 120 120 120 120 120 120 120 120 120 120
 120 120 120 120 120 120 120 120 120 120 120 120 120 120 120 120 120 120
 120 120 120 120 120 120 120 120 120 120 120]
```

Split Dataset

In [89]:
```
# Split the data into training and combined set (testing + validation)
X_train, X_combined, y_train, y_combined = train_test_split(X, y, test_size=0.2, random_state=4

# Split the combined set into testing and validation sets
X_test, X_val, y_test, y_val = train_test_split(X_combined, y_combined, test_size=0.5, random_s

# Now there are X_train, X_test, X_val as the respective feature sets
# and y_train, y_test, y_val as the respective target sets
```

In [90]:
```
print("X_train shape:", X_train.shape)
print("X_val shape:", X_val.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_val shape:", y_val.shape)
print("y_test shape:", y_test.shape)
print("Number of samples in X_train:", len(X_train))
print("Number of samples in X_val:", len(X_val))
print("Number of samples in X_test:", len(X_test))
print("Number of labels in y_train:", len(y_train))
print("Number of labels in y_val:", len(y_val))
print("Number of labels in y_test:", len(y_test))
```

```
X_train shape: (4512,)
X_val shape: (564,)
X_test shape: (564,)
y_train shape: (4512,)
y_val shape: (564,)
y_test shape: (564,)
Number of samples in X_train: 4512
Number of samples in X_val: 564
Number of samples in X_test: 564
Number of labels in y_train: 4512
Number of labels in y_val: 564
Number of labels in y_test: 564
```

Pre-process Dataset

In [91]:
```
resized_X_train = []
resized_X_val = []
resized_X_test = []

# Resize images in the training set
for image in X_train:
    resized_image = cv2.resize(image, (224, 224))
    resized_X_train.append(resized_image)

# Resize images in the validation set
for image in X_val:
    resized_image = cv2.resize(image, (224, 224))
    resized_X_val.append(resized_image)

# Resize images in the testing set
for image in X_test:
    resized_image = cv2.resize(image, (224, 224))
    resized_X_test.append(resized_image)

# Convert the lists to numpy arrays
X_train = np.array(resized_X_train)
X_val = np.array(resized_X_val)
X_test = np.array(resized_X_test)

# Reshape the input data to the desired shape
X_train = X_train.reshape(X_train.shape[0], 224, 224, 3)
X_val = X_val.reshape(X_val.shape[0], 224, 224, 3)
X_test = X_test.reshape(X_test.shape[0], 224, 224, 3)
```

In [92]:
```
# Normalize X
X_train = X_train / 255.0
X_val = X_val / 255.0
X_test = X_test / 255.0
```

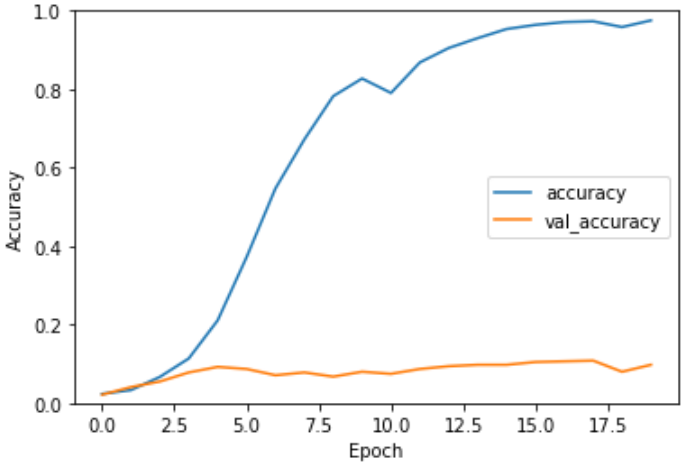
In [93]:
```
# Perform one-hot encoding on the labels
y_train = keras.utils.to_categorical(y_train, num_classes)
y_val = keras.utils.to_categorical(y_val, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

LeNet 5 Model

In [10]:
```
# Define the LeNet 5 model architecture
model = keras.Sequential([
    layers.Conv2D(6,kernel_size=(5,5),activation='relu',input_shape=(224,224,3)),
    layers.MaxPooling2D(pool_size=(2,2)),
    layers.Conv2D(16, kernel_size=(5,5), activation='relu'),
    layers.MaxPooling2D(pool_size=(2,2)),
    layers.Flatten(),
    layers.Dense(120, activation='relu'),
    layers.Dense(84, activation='relu'),
    layers.Dense(num_classes, activation='softmax')
])

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

# Train the model
history = model.fit(
    X_train, y_train,
    epochs=20,
    batch_size=128,
    verbose=1,
    validation_data=(X_val, y_val)
)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='center right')
```

```
Epoch 1/20
36/36 [==============================] - 61s 2s/step - loss: 4.0446 - accuracy: 0.0235 - val_l
oss: 3.8281 - val_accuracy: 0.0213
Epoch 2/20
36/36 [==============================] - 61s 2s/step - loss: 3.8046 - accuracy: 0.0330 - val_l
oss: 3.7937 - val_accuracy: 0.0408
Epoch 3/20
36/36 [==============================] - 66s 2s/step - loss: 3.7191 - accuracy: 0.0667 - val_l
oss: 3.7697 - val_accuracy: 0.0550
Epoch 4/20
36/36 [==============================] - 69s 2s/step - loss: 3.5514 - accuracy: 0.1137 - val_l
oss: 3.8276 - val_accuracy: 0.0780
Epoch 5/20
36/36 [==============================] - 69s 2s/step - loss: 3.1494 - accuracy: 0.2108 - val_l
oss: 3.8722 - val_accuracy: 0.0922
Epoch 6/20
36/36 [==============================] - 70s 2s/step - loss: 2.5277 - accuracy: 0.3717 - val_l
oss: 4.2708 - val_accuracy: 0.0869
Epoch 7/20
36/36 [==============================] - 70s 2s/step - loss: 1.8446 - accuracy: 0.5470 - val_l
oss: 5.3815 - val_accuracy: 0.0709
Epoch 8/20
36/36 [==============================] - 69s 2s/step - loss: 1.3209 - accuracy: 0.6718 - val_l
oss: 6.1650 - val_accuracy: 0.0780
Epoch 9/20
36/36 [==============================] - 69s 2s/step - loss: 0.9263 - accuracy: 0.7821 - val_l
oss: 7.1192 - val_accuracy: 0.0674
Epoch 10/20
36/36 [==============================] - 74s 2s/step - loss: 0.7416 - accuracy: 0.8271 - val_l
oss: 8.5597 - val_accuracy: 0.0798
Epoch 11/20
36/36 [==============================] - 74s 2s/step - loss: 0.9566 - accuracy: 0.7903 - val_l
oss: 8.1215 - val_accuracy: 0.0745
Epoch 12/20
36/36 [==============================] - 73s 2s/step - loss: 0.5742 - accuracy: 0.8681 - val_l
oss: 9.8621 - val_accuracy: 0.0869
Epoch 13/20
36/36 [==============================] - 72s 2s/step - loss: 0.4384 - accuracy: 0.9049 - val_l
oss: 9.6499 - val_accuracy: 0.0940
Epoch 14/20
36/36 [==============================] - 74s 2s/step - loss: 0.3155 - accuracy: 0.9300 - val_l
oss: 10.1538 - val_accuracy: 0.0975
Epoch 15/20
36/36 [==============================] - 75s 2s/step - loss: 0.2114 - accuracy: 0.9532 - val_l
oss: 10.5784 - val_accuracy: 0.0975
Epoch 16/20
36/36 [==============================] - 70s 2s/step - loss: 0.1713 - accuracy: 0.9639 - val_l
oss: 11.3988 - val_accuracy: 0.1046
Epoch 17/20
36/36 [==============================] - 70s 2s/step - loss: 0.1490 - accuracy: 0.9710 - val_l
oss: 11.6427 - val_accuracy: 0.1064
Epoch 18/20
36/36 [==============================] - 67s 2s/step - loss: 0.1480 - accuracy: 0.9730 - val_l
oss: 12.1059 - val_accuracy: 0.1082
Epoch 19/20
36/36 [==============================] - 66s 2s/step - loss: 0.2020 - accuracy: 0.9583 - val_l
oss: 11.6016 - val_accuracy: 0.0798
Epoch 20/20
36/36 [==============================] - 69s 2s/step - loss: 0.1253 - accuracy: 0.9752 - val_l
oss: 12.0452 - val_accuracy: 0.0975
```

Out[10]: <matplotlib.legend.Legend at 0x1db8038e790>

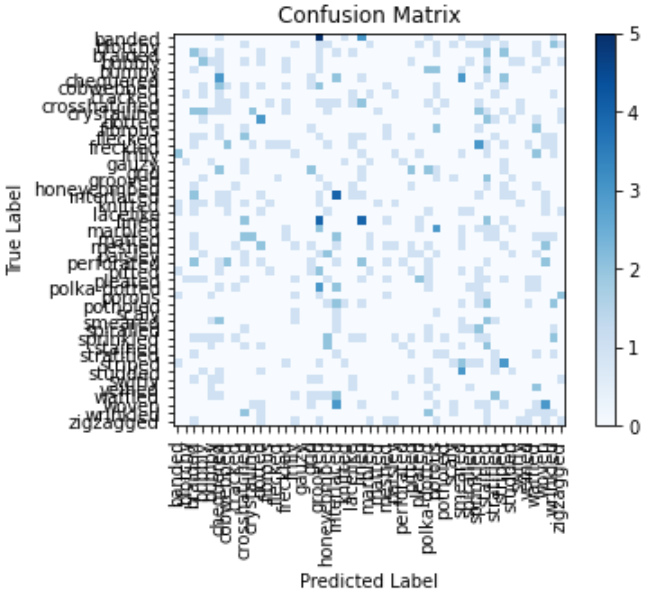


In [12]:
```
# Evaluate the model on the testing set
score = model.evaluate(X_test, y_test, verbose=2)
print('Test Loss:', score[0])
print('Test accuracy:', score[1])
```

```
18/18 - 2s - loss: 13.3695 - accuracy: 0.0851 - 2s/epoch - 115ms/step
Test Loss: 13.369486808776855
Test accuracy: 0.08510638028383255
```

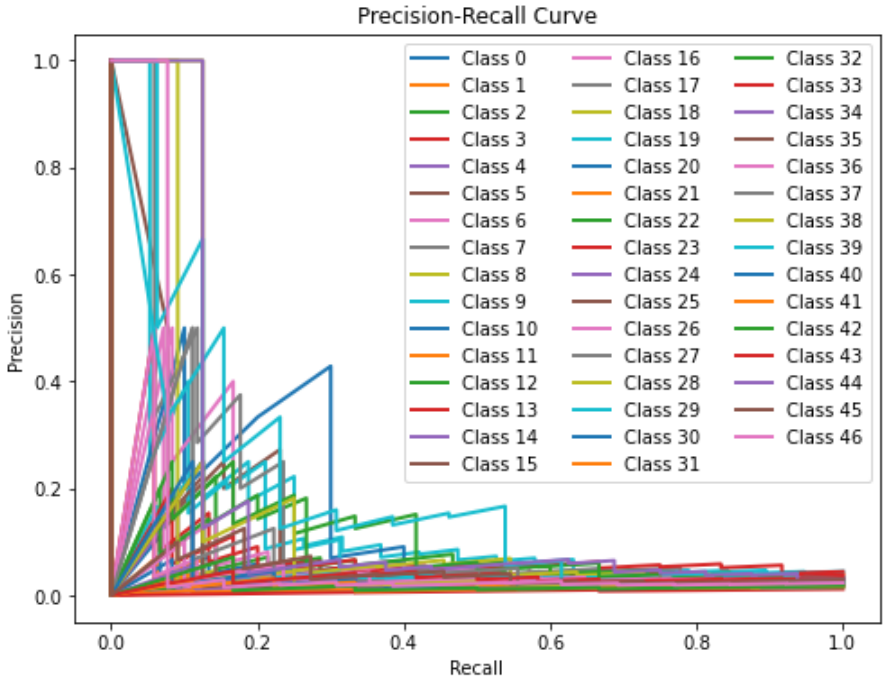
In [15]:
```
# Predict classes for the testing set
y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)

# Calculate precision, recall, and other metrics
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Calculate and plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.xticks(np.arange(len(class_names)), class_names, rotation=90)
plt.yticks(np.arange(len(class_names)), class_names)
plt.show()
```

```
18/18 [==============================] - 2s 107ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        15
           1       0.00      0.00      0.00        16
           2       0.10      0.14      0.11        14
           3       0.13      0.13      0.13        15
           4       0.00      0.00      0.00        12
           5       0.11      0.23      0.15        13
           6       0.10      0.07      0.08        14
           7       0.00      0.00      0.00        10
           8       0.00      0.00      0.00        14
           9       0.33      0.11      0.16        19
          10       0.19      0.30      0.23        10
          11       0.00      0.00      0.00         9
          12       0.00      0.00      0.00        13
          13       0.11      0.06      0.08        17
          14       0.11      0.09      0.10        11
          15       0.00      0.00      0.00         7
          16       0.18      0.17      0.17        12
          17       0.03      0.11      0.05         9
          18       0.07      0.12      0.09         8
          19       0.16      0.25      0.20        16
          20       0.00      0.00      0.00        12
          21       0.00      0.00      0.00        10
          22       0.19      0.27      0.22        15
          23       0.00      0.00      0.00        12
          24       0.00      0.00      0.00        15
          25       0.11      0.06      0.08        16
          26       0.00      0.00      0.00        10
          27       0.33      0.06      0.10        17
          28       0.09      0.09      0.09        11
          29       0.29      0.12      0.17        16
          30       0.05      0.08      0.06        12
          31       0.00      0.00      0.00         9
          32       0.00      0.00      0.00        12
          33       0.00      0.00      0.00         3
          34       0.05      0.11      0.07         9
          35       0.14      0.12      0.13         8
          36       0.05      0.06      0.05        17
          37       0.06      0.22      0.10         9
          38       0.08      0.12      0.10         8
          39       0.16      0.23      0.19        13
          40       0.14      0.11      0.12         9
          41       0.00      0.00      0.00         9
          42       0.00      0.00      0.00         6
          43       0.00      0.00      0.00        12
          44       0.19      0.19      0.19        16
          45       0.07      0.09      0.08        11
          46       0.11      0.08      0.09        13

    accuracy                           0.09       564
   macro avg       0.08      0.08      0.07       564
weighted avg       0.09      0.09      0.08       564

C:\Users\josep\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedM
etricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\josep\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedM
etricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\josep\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedM
etricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```



In [27]:
```
# Obtain the predicted probabilities for each class
y_pred_prob = model.predict(X_test)

# Binarize the true labels
y_test_bin = label_binarize(y_test, classes=np.unique(y_test))

# Calculate precision and recall for each class
precision = dict()
recall = dict()
for i in range(num_classes):
    precision[i], recall[i], _ = precision_recall_curve(y_test_bin[:, i], y_pred_prob[:, i])

# Plot precision-recall curve for each class
plt.figure(figsize=(8, 6))
for i in range(num_classes):
    plt.plot(recall[i], precision[i], lw=2, label=f'Class {i}')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='upper right', bbox_to_anchor=(1.0, 1.0), ncol=3)
plt.show()
```

```
18/18 [==============================] - 2s 118ms/step
```



CNN Model

CNN Model with 20 Epochs

In [28]:
```
# Define the CNN model architecture
model = keras.Sequential([
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(224, 224, 3)),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
    layers.Dropout(0.5),
    layers.Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='softmax')
])

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

# Train the model
history = model.fit(
    X_train, y_train,
    epochs=20,
    batch_size=128,
    verbose=1,
    validation_data=(X_val, y_val)
)
```

```
Epoch 1/20
36/36 [==============================] - 156s 4s/step - loss: 4.4337 - accuracy: 0.0224 - val_
loss: 4.1773 - val_accuracy: 0.0408
Epoch 2/20
36/36 [==============================] - 164s 5s/step - loss: 4.0893 - accuracy: 0.0268 - val_
loss: 4.0152 - val_accuracy: 0.0266
Epoch 3/20
36/36 [==============================] - 164s 5s/step - loss: 3.9661 - accuracy: 0.0299 - val_
loss: 3.9531 - val_accuracy: 0.0301
Epoch 4/20
36/36 [==============================] - 165s 5s/step - loss: 3.9176 - accuracy: 0.0370 - val_
loss: 3.9213 - val_accuracy: 0.0319
Epoch 5/20
36/36 [==============================] - 157s 4s/step - loss: 3.8951 - accuracy: 0.0397 - val_
loss: 3.9102 - val_accuracy: 0.0337
Epoch 6/20
36/36 [==============================] - 152s 4s/step - loss: 3.8761 - accuracy: 0.0401 - val_
loss: 3.9066 - val_accuracy: 0.0461
Epoch 7/20
36/36 [==============================] - 155s 4s/step - loss: 3.8623 - accuracy: 0.0468 - val_
loss: 3.8946 - val_accuracy: 0.0514
Epoch 8/20
36/36 [==============================] - 157s 4s/step - loss: 3.8446 - accuracy: 0.0543 - val_
loss: 3.8947 - val_accuracy: 0.0390
Epoch 9/20
36/36 [==============================] - 159s 4s/step - loss: 3.8133 - accuracy: 0.0516 - val_
loss: 3.8855 - val_accuracy: 0.0496
Epoch 10/20
36/36 [==============================] - 163s 5s/step - loss: 3.7857 - accuracy: 0.0678 - val_
loss: 3.8861 - val_accuracy: 0.0550
Epoch 11/20
36/36 [==============================] - 171s 5s/step - loss: 3.7723 - accuracy: 0.0731 - val_
loss: 3.8997 - val_accuracy: 0.0603
Epoch 12/20
36/36 [==============================] - 170s 5s/step - loss: 3.7425 - accuracy: 0.0889 - val_
loss: 3.9206 - val_accuracy: 0.0638
Epoch 13/20
36/36 [==============================] - 167s 5s/step - loss: 3.7133 - accuracy: 0.1053 - val_
loss: 3.9079 - val_accuracy: 0.0479
Epoch 14/20
36/36 [==============================] - 168s 5s/step - loss: 3.6562 - accuracy: 0.1197 - val_
loss: 3.9415 - val_accuracy: 0.0745
Epoch 15/20
36/36 [==============================] - 163s 5s/step - loss: 3.5938 - accuracy: 0.1463 - val_
loss: 3.9703 - val_accuracy: 0.0833
Epoch 16/20
36/36 [==============================] - 159s 4s/step - loss: 3.5603 - accuracy: 0.1653 - val_
loss: 3.9865 - val_accuracy: 0.0851
Epoch 17/20
36/36 [==============================] - 157s 4s/step - loss: 3.4730 - accuracy: 0.1886 - val_
loss: 4.0279 - val_accuracy: 0.0851
Epoch 18/20
36/36 [==============================] - 158s 4s/step - loss: 3.3745 - accuracy: 0.2152 - val_
loss: 4.0710 - val_accuracy: 0.0869
Epoch 19/20
36/36 [==============================] - 159s 4s/step - loss: 3.3252 - accuracy: 0.2431 - val_
loss: 4.0789 - val_accuracy: 0.0869
Epoch 20/20
36/36 [==============================] - 150s 4s/step - loss: 3.2689 - accuracy: 0.2538 - val_
loss: 4.0851 - val_accuracy: 0.1082
```

CNN Model with 50 Epochs

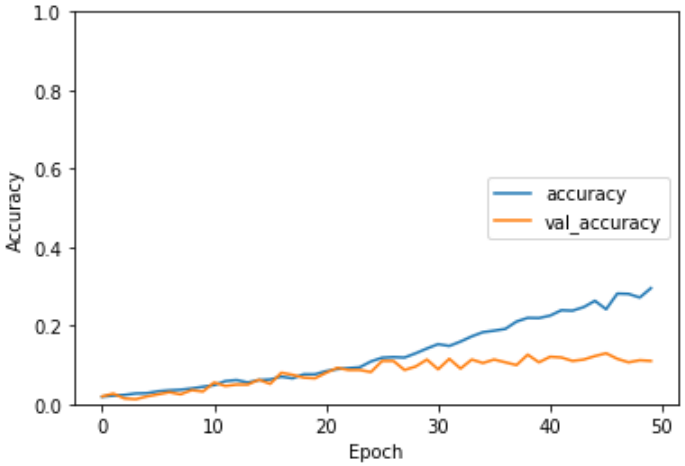
In [36]:
```
# Now with 50 epochs
model = keras.Sequential([
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(224, 224, 3)),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
    layers.Dropout(0.5),
    layers.Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='softmax')
])

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

# Train the model
history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=128,
    verbose=1,
    validation_data=(X_val, y_val)
)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='center right')
```

```
Epoch 1/50
36/36 [==============================] - 149s 4s/step - loss: 4.4766 - accuracy: 0.0188 - val_
loss: 4.1788 - val_accuracy: 0.0195
Epoch 2/50
36/36 [==============================] - 156s 4s/step - loss: 4.0969 - accuracy: 0.0215 - val_
loss: 4.0316 - val_accuracy: 0.0266
Epoch 3/50
36/36 [==============================] - 156s 4s/step - loss: 3.9966 - accuracy: 0.0233 - val_
loss: 3.9690 - val_accuracy: 0.0142
Epoch 4/50
36/36 [==============================] - 155s 4s/step - loss: 3.9480 - accuracy: 0.0270 - val_
loss: 3.9328 - val_accuracy: 0.0124
Epoch 5/50
36/36 [==============================] - 156s 4s/step - loss: 3.9111 - accuracy: 0.0277 - val_
loss: 3.8988 - val_accuracy: 0.0195
Epoch 6/50
36/36 [==============================] - 157s 4s/step - loss: 3.8857 - accuracy: 0.0326 - val_
loss: 3.8806 - val_accuracy: 0.0248
Epoch 7/50
36/36 [==============================] - 156s 4s/step - loss: 3.8592 - accuracy: 0.0350 - val_
loss: 3.8789 - val_accuracy: 0.0301
Epoch 8/50
36/36 [==============================] - 155s 4s/step - loss: 3.8459 - accuracy: 0.0366 - val_
loss: 3.8480 - val_accuracy: 0.0248
Epoch 9/50
36/36 [==============================] - 159s 4s/step - loss: 3.8149 - accuracy: 0.0399 - val_
loss: 3.8382 - val_accuracy: 0.0355
Epoch 10/50
36/36 [==============================] - 161s 4s/step - loss: 3.7899 - accuracy: 0.0441 - val_
loss: 3.8075 - val_accuracy: 0.0319
Epoch 11/50
36/36 [==============================] - 153s 4s/step - loss: 3.7721 - accuracy: 0.0481 - val_
loss: 3.7778 - val_accuracy: 0.0550
Epoch 12/50
36/36 [==============================] - 175s 5s/step - loss: 3.7468 - accuracy: 0.0585 - val_
loss: 3.7511 - val_accuracy: 0.0461
Epoch 13/50
36/36 [==============================] - 157s 4s/step - loss: 3.7195 - accuracy: 0.0612 - val_
loss: 3.7508 - val_accuracy: 0.0496
Epoch 14/50
36/36 [==============================] - 166s 5s/step - loss: 3.6933 - accuracy: 0.0550 - val_
loss: 3.7239 - val_accuracy: 0.0496
Epoch 15/50
36/36 [==============================] - 173s 5s/step - loss: 3.6571 - accuracy: 0.0614 - val_
loss: 3.7082 - val_accuracy: 0.0621
Epoch 16/50
36/36 [==============================] - 165s 5s/step - loss: 3.6413 - accuracy: 0.0627 - val_
loss: 3.6513 - val_accuracy: 0.0514
Epoch 17/50
36/36 [==============================] - 166s 5s/step - loss: 3.6190 - accuracy: 0.0698 - val_
loss: 3.6453 - val_accuracy: 0.0798
Epoch 18/50
36/36 [==============================] - 167s 5s/step - loss: 3.6045 - accuracy: 0.0656 - val_
loss: 3.6323 - val_accuracy: 0.0745
Epoch 19/50
36/36 [==============================] - 169s 5s/step - loss: 3.5804 - accuracy: 0.0758 - val_
loss: 3.7041 - val_accuracy: 0.0674
Epoch 20/50
36/36 [==============================] - 155s 4s/step - loss: 3.5598 - accuracy: 0.0758 - val_
loss: 3.6381 - val_accuracy: 0.0656
Epoch 21/50
36/36 [==============================] - 157s 4s/step - loss: 3.5369 - accuracy: 0.0840 - val_
loss: 3.6220 - val_accuracy: 0.0798
Epoch 22/50
36/36 [==============================] - 157s 4s/step - loss: 3.4940 - accuracy: 0.0891 - val_
loss: 3.6048 - val_accuracy: 0.0922
Epoch 23/50
36/36 [==============================] - 147s 4s/step - loss: 3.4633 - accuracy: 0.0915 - val_
```

```
loss: 3.6345 - val_accuracy: 0.0869
Epoch 24/50
36/36 [==============================] - 152s 4s/step - loss: 3.4180 - accuracy: 0.0938 - val_
loss: 3.6802 - val_accuracy: 0.0869
Epoch 25/50
36/36 [==============================] - 156s 4s/step - loss: 3.3909 - accuracy: 0.1086 - val_
loss: 3.6230 - val_accuracy: 0.0816
Epoch 26/50
36/36 [==============================] - 156s 4s/step - loss: 3.3713 - accuracy: 0.1181 - val_
loss: 3.6291 - val_accuracy: 0.1099
Epoch 27/50
36/36 [==============================] - 156s 4s/step - loss: 3.3405 - accuracy: 0.1197 - val_
loss: 3.6549 - val_accuracy: 0.1099
Epoch 28/50
36/36 [==============================] - 157s 4s/step - loss: 3.3247 - accuracy: 0.1186 - val_
loss: 3.6438 - val_accuracy: 0.0869
Epoch 29/50
36/36 [==============================] - 156s 4s/step - loss: 3.2885 - accuracy: 0.1294 - val_
loss: 3.6316 - val_accuracy: 0.0957
Epoch 30/50
36/36 [==============================] - 156s 4s/step - loss: 3.2517 - accuracy: 0.1416 - val_
loss: 3.6408 - val_accuracy: 0.1135
Epoch 31/50
36/36 [==============================] - 156s 4s/step - loss: 3.1863 - accuracy: 0.1527 - val_
loss: 3.7325 - val_accuracy: 0.0887
Epoch 32/50
36/36 [==============================] - 156s 4s/step - loss: 3.1833 - accuracy: 0.1480 - val_
loss: 3.6641 - val_accuracy: 0.1152
Epoch 33/50
36/36 [==============================] - 155s 4s/step - loss: 3.1235 - accuracy: 0.1594 - val_
loss: 3.6873 - val_accuracy: 0.0904
Epoch 34/50
36/36 [==============================] - 151s 4s/step - loss: 3.1272 - accuracy: 0.1724 - val_
loss: 3.7327 - val_accuracy: 0.1135
Epoch 35/50
36/36 [==============================] - 148s 4s/step - loss: 3.0488 - accuracy: 0.1833 - val_
loss: 3.7526 - val_accuracy: 0.1046
Epoch 36/50
36/36 [==============================] - 155s 4s/step - loss: 2.9972 - accuracy: 0.1871 - val_
loss: 3.8094 - val_accuracy: 0.1135
Epoch 37/50
36/36 [==============================] - 160s 4s/step - loss: 3.0107 - accuracy: 0.1915 - val_
loss: 3.6902 - val_accuracy: 0.1064
Epoch 38/50
36/36 [==============================] - 161s 4s/step - loss: 2.9664 - accuracy: 0.2103 - val_
loss: 3.7438 - val_accuracy: 0.0993
Epoch 39/50
36/36 [==============================] - 156s 4s/step - loss: 2.9222 - accuracy: 0.2196 - val_
loss: 3.7288 - val_accuracy: 0.1259
Epoch 40/50
36/36 [==============================] - 158s 4s/step - loss: 2.9081 - accuracy: 0.2192 - val_
loss: 3.6928 - val_accuracy: 0.1064
Epoch 41/50
36/36 [==============================] - 156s 4s/step - loss: 2.8655 - accuracy: 0.2254 - val_
loss: 3.8066 - val_accuracy: 0.1206
Epoch 42/50
36/36 [==============================] - 155s 4s/step - loss: 2.8092 - accuracy: 0.2391 - val_
loss: 3.8954 - val_accuracy: 0.1188
Epoch 43/50
36/36 [==============================] - 156s 4s/step - loss: 2.8246 - accuracy: 0.2380 - val_
loss: 3.8362 - val_accuracy: 0.1099
Epoch 44/50
36/36 [==============================] - 156s 4s/step - loss: 2.7875 - accuracy: 0.2471 - val_
loss: 3.8609 - val_accuracy: 0.1135
Epoch 45/50
36/36 [==============================] - 149s 4s/step - loss: 2.7892 - accuracy: 0.2631 - val_
loss: 3.8135 - val_accuracy: 0.1223
Epoch 46/50
```

```
36/36 [==============================] - 145s 4s/step - loss: 2.7752 - accuracy: 0.2416 - val_
loss: 3.9031 - val_accuracy: 0.1294
Epoch 47/50
36/36 [==============================] - 148s 4s/step - loss: 2.6770 - accuracy: 0.2817 - val_
loss: 3.9568 - val_accuracy: 0.1152
Epoch 48/50
36/36 [==============================] - 156s 4s/step - loss: 2.6802 - accuracy: 0.2808 - val_
loss: 3.9217 - val_accuracy: 0.1064
Epoch 49/50
36/36 [==============================] - 160s 4s/step - loss: 2.7268 - accuracy: 0.2715 - val_
loss: 4.0463 - val_accuracy: 0.1117
Epoch 50/50
36/36 [==============================] - 156s 4s/step - loss: 2.6147 - accuracy: 0.2954 - val_
loss: 3.9666 - val_accuracy: 0.1099
```

Out[36]: `<matplotlib.legend.Legend at 0x1dd505dc820>`



In [78]:
```
# Evaluate the model on the testing set
score = model.evaluate(X_test, y_test, verbose=2)
print('Test Loss:', score[0])
print('Test accuracy:', score[1])
```

```
18/18 - 6s - loss: 3.5595 - accuracy: 0.0780 - 6s/epoch - 318ms/step
Test Loss: 3.5595102310180664
Test accuracy: 0.07801418751478195
```

AlexNet Model

AlexNet Model with 50 Epochs and Learning Rate 0.001

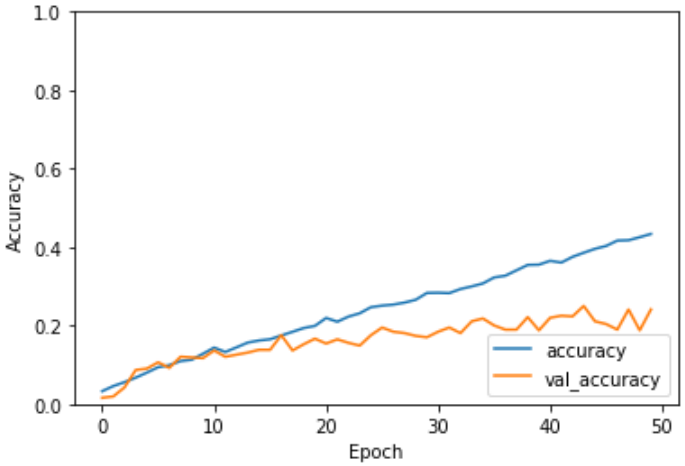
In [34]:
```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='relu', inpu
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu', paddi
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', paddi
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', paddi
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='relu', paddi
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Flatten(),
    keras.layers.Dense(4096, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4096, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(num_classes, activation='softmax')
])

model.compile(loss='categorical_crossentropy', optimizer=tf.optimizers.SGD(learning_rate=0.001)
model.summary()

# Train the model
history = model.fit(
    X_train, y_train,
    epochs=50,
    verbose=1,
    validation_data=(X_val, y_val)
)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 54, 54, 96)        34944

 batch_normalization_10 (Bat  (None, 54, 54, 96)       384
 chNormalization)

 max_pooling2d_6 (MaxPooling  (None, 26, 26, 96)       0
 2D)

 conv2d_11 (Conv2D)          (None, 26, 26, 256)       614656

 batch_normalization_11 (Bat  (None, 26, 26, 256)      1024
 chNormalization)

 max_pooling2d_7 (MaxPooling  (None, 12, 12, 256)      0
 2D)

 conv2d_12 (Conv2D)          (None, 12, 12, 384)       885120

 batch_normalization_12 (Bat  (None, 12, 12, 384)      1536
 chNormalization)

 conv2d_13 (Conv2D)          (None, 12, 12, 384)       1327488

 batch_normalization_13 (Bat  (None, 12, 12, 384)      1536
 chNormalization)

 conv2d_14 (Conv2D)          (None, 12, 12, 256)       884992

 batch_normalization_14 (Bat  (None, 12, 12, 256)      1024
 chNormalization)

 max_pooling2d_8 (MaxPooling  (None, 5, 5, 256)        0
 2D)

 flatten_2 (Flatten)         (None, 6400)              0

 dense_6 (Dense)             (None, 4096)              26218496

 dropout_4 (Dropout)         (None, 4096)              0

 dense_7 (Dense)             (None, 4096)              16781312

 dropout_5 (Dropout)         (None, 4096)              0

 dense_8 (Dense)             (None, 47)                192559

=================================================================
Total params: 46,945,071
Trainable params: 46,942,319
Non-trainable params: 2,752
_________________________________________________________________
Epoch 1/50
141/141 [==============================] - 214s 2s/step - loss: 5.2604 - accuracy: 0.0328 - va
l_loss: 3.9002 - val_accuracy: 0.0160
Epoch 2/50
141/141 [==============================] - 219s 2s/step - loss: 4.5072 - accuracy: 0.0459 - va
l_loss: 3.8646 - val_accuracy: 0.0195
Epoch 3/50
141/141 [==============================] - 218s 2s/step - loss: 4.1889 - accuracy: 0.0559 - va
l_loss: 3.7730 - val_accuracy: 0.0426
Epoch 4/50
141/141 [==============================] - 218s 2s/step - loss: 4.0252 - accuracy: 0.0674 - va
l_loss: 3.6270 - val_accuracy: 0.0869
Epoch 5/50
```

```
141/141 [==============================] - 220s 2s/step - loss: 3.8617 - accuracy: 0.0802 - va
l_loss: 3.6061 - val_accuracy: 0.0904
Epoch 6/50
141/141 [==============================] - 219s 2s/step - loss: 3.7654 - accuracy: 0.0940 - va
l_loss: 3.5801 - val_accuracy: 0.1064
Epoch 7/50
141/141 [==============================] - 218s 2s/step - loss: 3.6971 - accuracy: 0.0991 - va
l_loss: 3.6370 - val_accuracy: 0.0922
Epoch 8/50
141/141 [==============================] - 211s 1s/step - loss: 3.6016 - accuracy: 0.1095 - va
l_loss: 3.5480 - val_accuracy: 0.1206
Epoch 9/50
141/141 [==============================] - 204s 1s/step - loss: 3.5590 - accuracy: 0.1130 - va
l_loss: 3.4531 - val_accuracy: 0.1188
Epoch 10/50
141/141 [==============================] - 222s 2s/step - loss: 3.4700 - accuracy: 0.1281 - va
l_loss: 3.5021 - val_accuracy: 0.1170
Epoch 11/50
141/141 [==============================] - 223s 2s/step - loss: 3.4365 - accuracy: 0.1438 - va
l_loss: 3.4148 - val_accuracy: 0.1365
Epoch 12/50
141/141 [==============================] - 219s 2s/step - loss: 3.4060 - accuracy: 0.1328 - va
l_loss: 3.4566 - val_accuracy: 0.1206
Epoch 13/50
141/141 [==============================] - 219s 2s/step - loss: 3.3826 - accuracy: 0.1445 - va
l_loss: 3.4342 - val_accuracy: 0.1259
Epoch 14/50
141/141 [==============================] - 217s 2s/step - loss: 3.3205 - accuracy: 0.1567 - va
l_loss: 3.4636 - val_accuracy: 0.1312
Epoch 15/50
141/141 [==============================] - 220s 2s/step - loss: 3.2457 - accuracy: 0.1620 - va
l_loss: 3.4046 - val_accuracy: 0.1383
Epoch 16/50
141/141 [==============================] - 222s 2s/step - loss: 3.2403 - accuracy: 0.1651 - va
l_loss: 3.4926 - val_accuracy: 0.1383
Epoch 17/50
141/141 [==============================] - 207s 1s/step - loss: 3.2008 - accuracy: 0.1749 - va
l_loss: 3.2705 - val_accuracy: 0.1755
Epoch 18/50
141/141 [==============================] - 214s 2s/step - loss: 3.1591 - accuracy: 0.1842 - va
l_loss: 3.4299 - val_accuracy: 0.1365
Epoch 19/50
141/141 [==============================] - 218s 2s/step - loss: 3.1296 - accuracy: 0.1937 - va
l_loss: 3.3244 - val_accuracy: 0.1525
Epoch 20/50
141/141 [==============================] - 218s 2s/step - loss: 3.0948 - accuracy: 0.1992 - va
l_loss: 3.3670 - val_accuracy: 0.1667
Epoch 21/50
141/141 [==============================] - 220s 2s/step - loss: 3.0409 - accuracy: 0.2194 - va
l_loss: 3.3938 - val_accuracy: 0.1543
Epoch 22/50
141/141 [==============================] - 218s 2s/step - loss: 3.0080 - accuracy: 0.2099 - va
l_loss: 3.2298 - val_accuracy: 0.1649
Epoch 23/50
141/141 [==============================] - 218s 2s/step - loss: 2.9915 - accuracy: 0.2230 - va
l_loss: 3.3630 - val_accuracy: 0.1560
Epoch 24/50
141/141 [==============================] - 217s 2s/step - loss: 2.9364 - accuracy: 0.2314 - va
l_loss: 3.5324 - val_accuracy: 0.1489
Epoch 25/50
141/141 [==============================] - 204s 1s/step - loss: 2.9078 - accuracy: 0.2469 - va
l_loss: 3.2160 - val_accuracy: 0.1755
Epoch 26/50
141/141 [==============================] - 209s 1s/step - loss: 2.8707 - accuracy: 0.2509 - va
l_loss: 3.2589 - val_accuracy: 0.1950
Epoch 27/50
141/141 [==============================] - 217s 2s/step - loss: 2.8422 - accuracy: 0.2535 - va
l_loss: 3.1965 - val_accuracy: 0.1844
```

```
Epoch 28/50
141/141 [==============================] - 219s 2s/step - loss: 2.8176 - accuracy: 0.2586 - va
l_loss: 3.2422 - val_accuracy: 0.1809
Epoch 29/50
141/141 [==============================] - 217s 2s/step - loss: 2.7699 - accuracy: 0.2657 - va
l_loss: 3.2973 - val_accuracy: 0.1738
Epoch 30/50
141/141 [==============================] - 218s 2s/step - loss: 2.7338 - accuracy: 0.2832 - va
l_loss: 3.2957 - val_accuracy: 0.1702
Epoch 31/50
141/141 [==============================] - 220s 2s/step - loss: 2.6969 - accuracy: 0.2837 - va
l_loss: 3.3316 - val_accuracy: 0.1844
Epoch 32/50
141/141 [==============================] - 220s 2s/step - loss: 2.6938 - accuracy: 0.2828 - va
l_loss: 3.1074 - val_accuracy: 0.1950
Epoch 33/50
141/141 [==============================] - 210s 1s/step - loss: 2.6624 - accuracy: 0.2932 - va
l_loss: 3.2313 - val_accuracy: 0.1809
Epoch 34/50
141/141 [==============================] - 204s 1s/step - loss: 2.6107 - accuracy: 0.2999 - va
l_loss: 3.0846 - val_accuracy: 0.2110
Epoch 35/50
141/141 [==============================] - 218s 2s/step - loss: 2.5728 - accuracy: 0.3074 - va
l_loss: 3.1038 - val_accuracy: 0.2181
Epoch 36/50
141/141 [==============================] - 219s 2s/step - loss: 2.5344 - accuracy: 0.3227 - va
l_loss: 3.0639 - val_accuracy: 0.2004
Epoch 37/50
141/141 [==============================] - 219s 2s/step - loss: 2.5124 - accuracy: 0.3271 - va
l_loss: 3.2038 - val_accuracy: 0.1897
Epoch 38/50
141/141 [==============================] - 219s 2s/step - loss: 2.4633 - accuracy: 0.3409 - va
l_loss: 3.1867 - val_accuracy: 0.1897
Epoch 39/50
141/141 [==============================] - 218s 2s/step - loss: 2.4330 - accuracy: 0.3546 - va
l_loss: 3.0909 - val_accuracy: 0.2216
Epoch 40/50
141/141 [==============================] - 217s 2s/step - loss: 2.3823 - accuracy: 0.3553 - va
l_loss: 3.2403 - val_accuracy: 0.1879
Epoch 41/50
141/141 [==============================] - 217s 2s/step - loss: 2.3737 - accuracy: 0.3650 - va
l_loss: 3.0918 - val_accuracy: 0.2199
Epoch 42/50
141/141 [==============================] - 203s 1s/step - loss: 2.3417 - accuracy: 0.3606 - va
l_loss: 3.0464 - val_accuracy: 0.2252
Epoch 43/50
141/141 [==============================] - 212s 2s/step - loss: 2.2965 - accuracy: 0.3752 - va
l_loss: 3.0277 - val_accuracy: 0.2234
Epoch 44/50
141/141 [==============================] - 218s 2s/step - loss: 2.2651 - accuracy: 0.3856 - va
l_loss: 2.9942 - val_accuracy: 0.2500
Epoch 45/50
141/141 [==============================] - 217s 2s/step - loss: 2.2616 - accuracy: 0.3956 - va
l_loss: 3.1048 - val_accuracy: 0.2110
Epoch 46/50
141/141 [==============================] - 218s 2s/step - loss: 2.1995 - accuracy: 0.4029 - va
l_loss: 3.0981 - val_accuracy: 0.2039
Epoch 47/50
141/141 [==============================] - 217s 2s/step - loss: 2.1449 - accuracy: 0.4167 - va
l_loss: 3.0866 - val_accuracy: 0.1897
Epoch 48/50
141/141 [==============================] - 217s 2s/step - loss: 2.1445 - accuracy: 0.4176 - va
l_loss: 2.9823 - val_accuracy: 0.2411
Epoch 49/50
141/141 [==============================] - 218s 2s/step - loss: 2.0908 - accuracy: 0.4253 - va
l_loss: 3.5063 - val_accuracy: 0.1879
Epoch 50/50
```

```
141/141 [==============================] - 203s 1s/step - loss: 2.0701 - accuracy: 0.4335 - va
l_loss: 3.0149 - val_accuracy: 0.2411
```



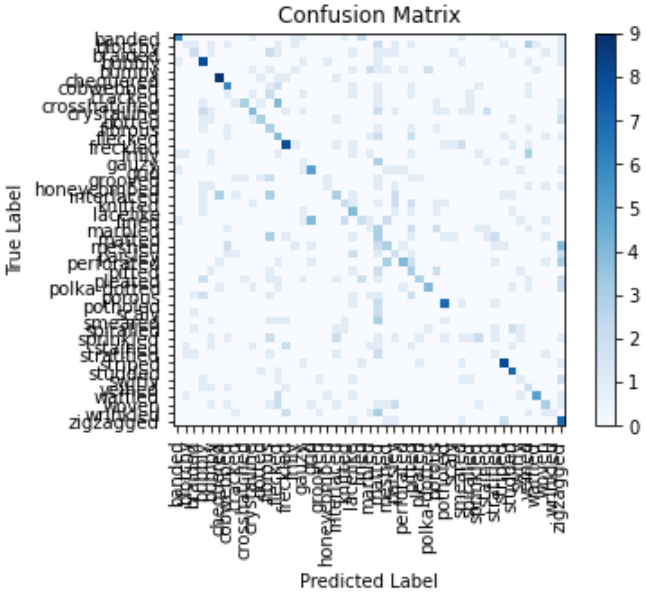
In [35]:
```
# Evaluate the model on the testing set
score = model.evaluate(X_test, y_test, verbose=2)
print('Test Loss:', score[0])
print('Test accuracy:', score[1])
```

```
18/18 - 6s - loss: 3.0014 - accuracy: 0.2518 - 6s/epoch - 331ms/step
Test Loss: 3.0014026165008545
Test accuracy: 0.25177305936813354
```

In [36]:
```
# Predict classes for the testing set
y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)

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

# Calculate precision, recall, and other metrics
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Calculate and plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.xticks(np.arange(len(class_names)), class_names, rotation=90)
plt.yticks(np.arange(len(class_names)), class_names)
plt.show()
```

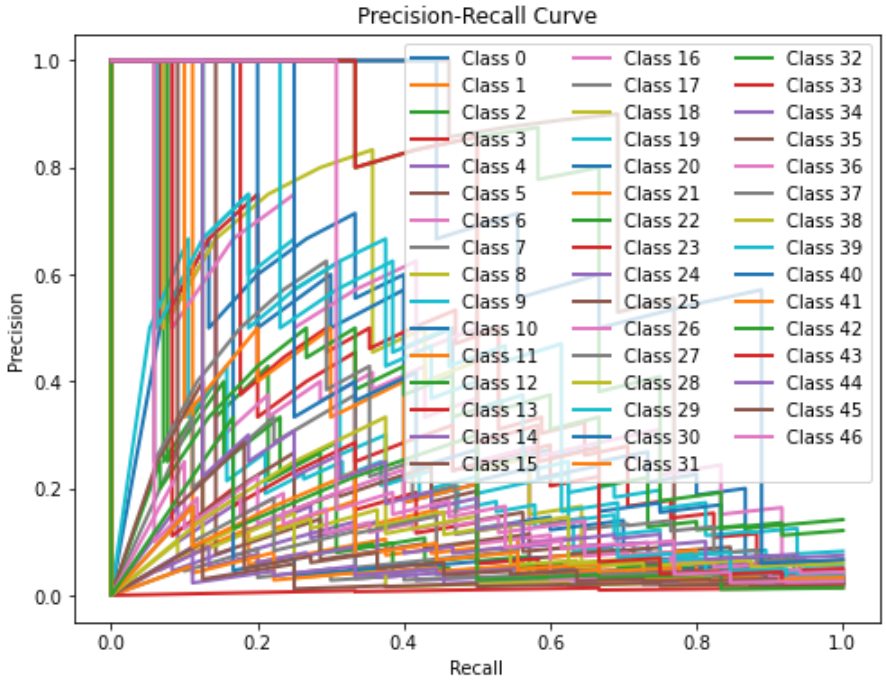
```
18/18 [==============================] - 6s 323ms/step
Classification Report:
```

	precision	recall	f1-score	support
0	0.60	0.40	0.48	15
1	0.20	0.06	0.10	16
2	0.29	0.14	0.19	14
3	0.35	0.53	0.42	15
4	0.00	0.00	0.00	12
5	0.64	0.69	0.67	13
6	0.35	0.43	0.39	14
7	0.00	0.00	0.00	10
8	0.60	0.21	0.32	14
9	0.36	0.21	0.27	19
10	0.43	0.30	0.35	10
11	0.12	0.33	0.18	9
12	0.17	0.31	0.22	13
13	0.47	0.47	0.47	17
14	0.00	0.00	0.00	11
15	0.10	0.14	0.12	7
16	0.42	0.42	0.42	12
17	0.00	0.00	0.00	9
18	0.25	0.12	0.17	8
19	0.25	0.19	0.21	16
20	0.29	0.17	0.21	12
21	0.19	0.40	0.26	10
22	0.29	0.13	0.18	15
23	0.11	0.08	0.10	12
24	0.06	0.20	0.10	15
25	0.23	0.19	0.21	16
26	0.12	0.20	0.15	10
27	0.50	0.24	0.32	17
28	0.13	0.27	0.18	11
29	0.50	0.19	0.27	16
30	0.57	0.33	0.42	12
31	0.00	0.00	0.00	9
32	0.54	0.58	0.56	12
33	0.00	0.00	0.00	3
34	0.00	0.00	0.00	9
35	0.00	0.00	0.00	8
36	0.33	0.12	0.17	17
37	0.00	0.00	0.00	9
38	0.11	0.12	0.12	8
39	0.53	0.62	0.57	13
40	0.44	0.78	0.56	9
41	0.14	0.11	0.12	9
42	0.11	0.33	0.17	6
43	0.71	0.42	0.53	12
44	0.27	0.19	0.22	16
45	1.00	0.09	0.17	11
46	0.19	0.54	0.28	13
accuracy			0.25	564
macro avg	0.28	0.24	0.23	564
weighted avg	0.30	0.25	0.25	564



In [37]:
```
# Obtain the predicted probabilities for each class
y_pred_prob = model.predict(X_test)

# Binarize the true labels
y_test_bin = label_binarize(y_test, classes=np.unique(y_test))

# Calculate precision and recall for each class
precision = dict()
recall = dict()
for i in range(num_classes):
    precision[i], recall[i], _ = precision_recall_curve(y_test_bin[:, i], y_pred_prob[:, i])

# Plot precision-recall curve for each class
plt.figure(figsize=(8, 6))
for i in range(num_classes):
    plt.plot(recall[i], precision[i], lw=2, label=f'Class {i}')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='upper right', bbox_to_anchor=(1.0, 1.0), ncol=3)
plt.show()
```

```
18/18 [==============================] - 6s 329ms/step
```



AlexNet Model with 50 Epochs and Learning Rate 0.01

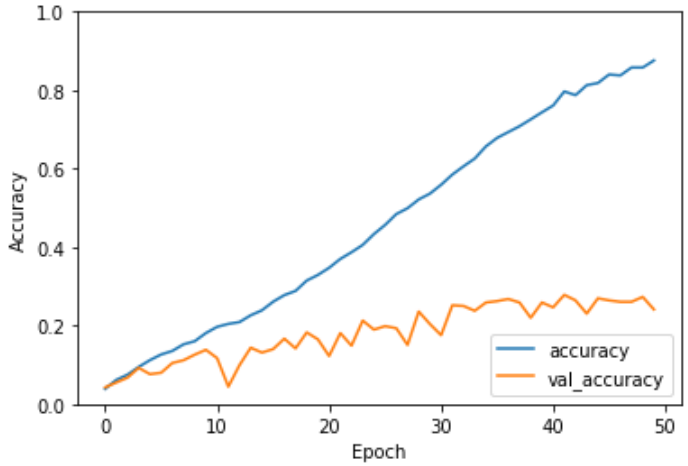
In [47]:
```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='relu', inpu
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu', paddi
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', paddi
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', paddi
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='relu', paddi
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Flatten(),
    keras.layers.Dense(4096, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4096, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(num_classes, activation='softmax')
])

model.compile(loss='categorical_crossentropy', optimizer=tf.optimizers.SGD(learning_rate=0.01),
model.summary()

# Train the model
history = model.fit(
    X_train, y_train,
    epochs=50,
    verbose=1,
    validation_data=(X_val, y_val)
)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_15 (Conv2D)          (None, 54, 54, 96)        34944

 batch_normalization_15 (Bat  (None, 54, 54, 96)       384
 chNormalization)

 max_pooling2d_9 (MaxPooling  (None, 26, 26, 96)       0
 2D)

 conv2d_16 (Conv2D)          (None, 26, 26, 256)       614656

 batch_normalization_16 (Bat  (None, 26, 26, 256)      1024
 chNormalization)

 max_pooling2d_10 (MaxPoolin  (None, 12, 12, 256)      0
 g2D)

 conv2d_17 (Conv2D)          (None, 12, 12, 384)       885120

 batch_normalization_17 (Bat  (None, 12, 12, 384)      1536
 chNormalization)

 conv2d_18 (Conv2D)          (None, 12, 12, 384)       1327488

 batch_normalization_18 (Bat  (None, 12, 12, 384)      1536
 chNormalization)

 conv2d_19 (Conv2D)          (None, 12, 12, 256)       884992

 batch_normalization_19 (Bat  (None, 12, 12, 256)      1024
 chNormalization)

 max_pooling2d_11 (MaxPoolin  (None, 5, 5, 256)        0
 g2D)

 flatten_3 (Flatten)         (None, 6400)              0

 dense_9 (Dense)             (None, 4096)              26218496

 dropout_6 (Dropout)         (None, 4096)              0

 dense_10 (Dense)            (None, 4096)              16781312

 dropout_7 (Dropout)         (None, 4096)              0

 dense_11 (Dense)            (None, 47)                192559

=================================================================
Total params: 46,945,071
Trainable params: 46,942,319
Non-trainable params: 2,752
_________________________________________________________________
Epoch 1/50
141/141 [==============================] - 215s 2s/step - loss: 4.8836 - accuracy: 0.0388 - va
l_loss: 3.8321 - val_accuracy: 0.0426
Epoch 2/50
141/141 [==============================] - 220s 2s/step - loss: 4.0172 - accuracy: 0.0616 - va
l_loss: 3.8202 - val_accuracy: 0.0550
Epoch 3/50
141/141 [==============================] - 219s 2s/step - loss: 3.8124 - accuracy: 0.0749 - va
l_loss: 3.7001 - val_accuracy: 0.0674
Epoch 4/50
141/141 [==============================] - 218s 2s/step - loss: 3.6461 - accuracy: 0.0946 - va
l_loss: 3.5628 - val_accuracy: 0.0922
Epoch 5/50
```

```
141/141 [==============================] - 220s 2s/step - loss: 3.5394 - accuracy: 0.1119 - va
l_loss: 3.6397 - val_accuracy: 0.0762
Epoch 6/50
141/141 [==============================] - 219s 2s/step - loss: 3.4770 - accuracy: 0.1263 - va
l_loss: 3.6056 - val_accuracy: 0.0798
Epoch 7/50
141/141 [==============================] - 218s 2s/step - loss: 3.3913 - accuracy: 0.1354 - va
l_loss: 3.4952 - val_accuracy: 0.1046
Epoch 8/50
141/141 [==============================] - 208s 1s/step - loss: 3.3078 - accuracy: 0.1523 - va
l_loss: 3.4411 - val_accuracy: 0.1117
Epoch 9/50
141/141 [==============================] - 203s 1s/step - loss: 3.2560 - accuracy: 0.1602 - va
l_loss: 3.3436 - val_accuracy: 0.1259
Epoch 10/50
141/141 [==============================] - 218s 2s/step - loss: 3.1938 - accuracy: 0.1811 - va
l_loss: 3.4025 - val_accuracy: 0.1383
Epoch 11/50
141/141 [==============================] - 218s 2s/step - loss: 3.0845 - accuracy: 0.1968 - va
l_loss: 3.7430 - val_accuracy: 0.1170
Epoch 12/50
141/141 [==============================] - 218s 2s/step - loss: 3.0342 - accuracy: 0.2043 - va
l_loss: 4.2336 - val_accuracy: 0.0443
Epoch 13/50
141/141 [==============================] - 219s 2s/step - loss: 3.0092 - accuracy: 0.2090 - va
l_loss: 3.8172 - val_accuracy: 0.0993
Epoch 14/50
141/141 [==============================] - 218s 2s/step - loss: 2.9281 - accuracy: 0.2263 - va
l_loss: 3.4043 - val_accuracy: 0.1436
Epoch 15/50
141/141 [==============================] - 219s 2s/step - loss: 2.8614 - accuracy: 0.2389 - va
l_loss: 3.3307 - val_accuracy: 0.1312
Epoch 16/50
141/141 [==============================] - 216s 2s/step - loss: 2.7742 - accuracy: 0.2609 - va
l_loss: 3.3919 - val_accuracy: 0.1401
Epoch 17/50
141/141 [==============================] - 204s 1s/step - loss: 2.7013 - accuracy: 0.2777 - va
l_loss: 3.2417 - val_accuracy: 0.1667
Epoch 18/50
141/141 [==============================] - 213s 2s/step - loss: 2.6280 - accuracy: 0.2883 - va
l_loss: 3.3154 - val_accuracy: 0.1418
Epoch 19/50
141/141 [==============================] - 218s 2s/step - loss: 2.5648 - accuracy: 0.3147 - va
l_loss: 3.2357 - val_accuracy: 0.1826
Epoch 20/50
141/141 [==============================] - 218s 2s/step - loss: 2.4679 - accuracy: 0.3291 - va
l_loss: 3.3177 - val_accuracy: 0.1649
Epoch 21/50
141/141 [==============================] - 221s 2s/step - loss: 2.3797 - accuracy: 0.3473 - va
l_loss: 3.6015 - val_accuracy: 0.1223
Epoch 22/50
141/141 [==============================] - 218s 2s/step - loss: 2.2998 - accuracy: 0.3699 - va
l_loss: 3.3148 - val_accuracy: 0.1809
Epoch 23/50
141/141 [==============================] - 218s 2s/step - loss: 2.2309 - accuracy: 0.3870 - va
l_loss: 3.6513 - val_accuracy: 0.1489
Epoch 24/50
141/141 [==============================] - 220s 2s/step - loss: 2.1293 - accuracy: 0.4056 - va
l_loss: 3.1309 - val_accuracy: 0.2128
Epoch 25/50
141/141 [==============================] - 205s 1s/step - loss: 2.0382 - accuracy: 0.4333 - va
l_loss: 3.2695 - val_accuracy: 0.1897
Epoch 26/50
141/141 [==============================] - 207s 1s/step - loss: 1.9455 - accuracy: 0.4566 - va
l_loss: 3.2975 - val_accuracy: 0.1986
Epoch 27/50
141/141 [==============================] - 217s 2s/step - loss: 1.8819 - accuracy: 0.4843 - va
l_loss: 3.5896 - val_accuracy: 0.1933
```

```
Epoch 28/50
141/141 [==============================] - 218s 2s/step - loss: 1.7704 - accuracy: 0.4991 - va
l_loss: 3.6628 - val_accuracy: 0.1507
Epoch 29/50
141/141 [==============================] - 218s 2s/step - loss: 1.6962 - accuracy: 0.5215 - va
l_loss: 3.1028 - val_accuracy: 0.2358
Epoch 30/50
141/141 [==============================] - 218s 2s/step - loss: 1.6295 - accuracy: 0.5363 - va
l_loss: 3.4030 - val_accuracy: 0.2039
Epoch 31/50
141/141 [==============================] - 218s 2s/step - loss: 1.5362 - accuracy: 0.5590 - va
l_loss: 3.7357 - val_accuracy: 0.1755
Epoch 32/50
141/141 [==============================] - 218s 2s/step - loss: 1.4562 - accuracy: 0.5849 - va
l_loss: 3.1535 - val_accuracy: 0.2518
Epoch 33/50
141/141 [==============================] - 209s 1s/step - loss: 1.3643 - accuracy: 0.6059 - va
l_loss: 3.1617 - val_accuracy: 0.2500
Epoch 34/50
141/141 [==============================] - 204s 1s/step - loss: 1.2824 - accuracy: 0.6257 - va
l_loss: 3.2588 - val_accuracy: 0.2376
Epoch 35/50
141/141 [==============================] - 217s 2s/step - loss: 1.1926 - accuracy: 0.6567 - va
l_loss: 3.2759 - val_accuracy: 0.2589
Epoch 36/50
141/141 [==============================] - 218s 2s/step - loss: 1.1101 - accuracy: 0.6784 - va
l_loss: 3.2158 - val_accuracy: 0.2624
Epoch 37/50
141/141 [==============================] - 218s 2s/step - loss: 1.0726 - accuracy: 0.6933 - va
l_loss: 3.3322 - val_accuracy: 0.2677
Epoch 38/50
141/141 [==============================] - 218s 2s/step - loss: 1.0134 - accuracy: 0.7081 - va
l_loss: 3.2056 - val_accuracy: 0.2589
Epoch 39/50
141/141 [==============================] - 217s 2s/step - loss: 0.9505 - accuracy: 0.7256 - va
l_loss: 3.7358 - val_accuracy: 0.2199
Epoch 40/50
141/141 [==============================] - 218s 2s/step - loss: 0.8755 - accuracy: 0.7436 - va
l_loss: 3.3627 - val_accuracy: 0.2589
Epoch 41/50
141/141 [==============================] - 214s 2s/step - loss: 0.8179 - accuracy: 0.7609 - va
l_loss: 3.5969 - val_accuracy: 0.2465
Epoch 42/50
141/141 [==============================] - 203s 1s/step - loss: 0.7254 - accuracy: 0.7968 - va
l_loss: 3.4931 - val_accuracy: 0.2784
Epoch 43/50
141/141 [==============================] - 210s 1s/step - loss: 0.7010 - accuracy: 0.7872 - va
l_loss: 3.3895 - val_accuracy: 0.2642
Epoch 44/50
141/141 [==============================] - 217s 2s/step - loss: 0.6523 - accuracy: 0.8127 - va
l_loss: 4.2434 - val_accuracy: 0.2305
Epoch 45/50
141/141 [==============================] - 218s 2s/step - loss: 0.6263 - accuracy: 0.8183 - va
l_loss: 3.5627 - val_accuracy: 0.2695
Epoch 46/50
141/141 [==============================] - 218s 2s/step - loss: 0.5521 - accuracy: 0.8400 - va
l_loss: 3.5264 - val_accuracy: 0.2642
Epoch 47/50
141/141 [==============================] - 218s 2s/step - loss: 0.5510 - accuracy: 0.8375 - va
l_loss: 3.6248 - val_accuracy: 0.2606
Epoch 48/50
141/141 [==============================] - 217s 2s/step - loss: 0.4987 - accuracy: 0.8577 - va
l_loss: 3.5673 - val_accuracy: 0.2606
Epoch 49/50
141/141 [==============================] - 219s 2s/step - loss: 0.4955 - accuracy: 0.8577 - va
l_loss: 3.7836 - val_accuracy: 0.2730
Epoch 50/50
```

```
141/141 [==============================] - 207s 1s/step - loss: 0.4411 - accuracy: 0.8757 - va
l_loss: 3.6587 - val_accuracy: 0.2411
```



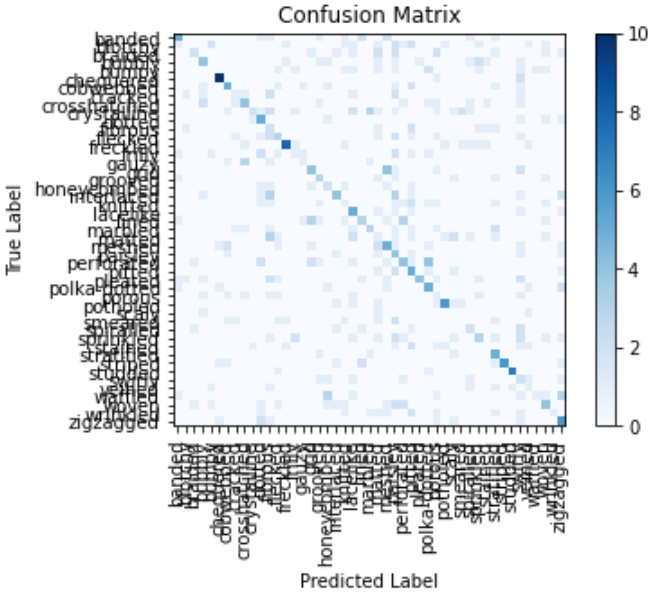
In [48]:
```
# Evaluate the model on the testing set
score = model.evaluate(X_test, y_test, verbose=2)
print('Test Loss:', score[0])
print('Test accuracy:', score[1])
```

```
18/18 - 6s - loss: 3.6180 - accuracy: 0.2677 - 6s/epoch - 319ms/step
Test Loss: 3.6179747581481934
Test accuracy: 0.2677305042743683
```

In [49]:
```
# Predict classes for the testing set
y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)

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

# Calculate precision, recall, and other metrics
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Calculate and plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.xticks(np.arange(len(class_names)), class_names, rotation=90)
plt.yticks(np.arange(len(class_names)), class_names)
plt.show()
```

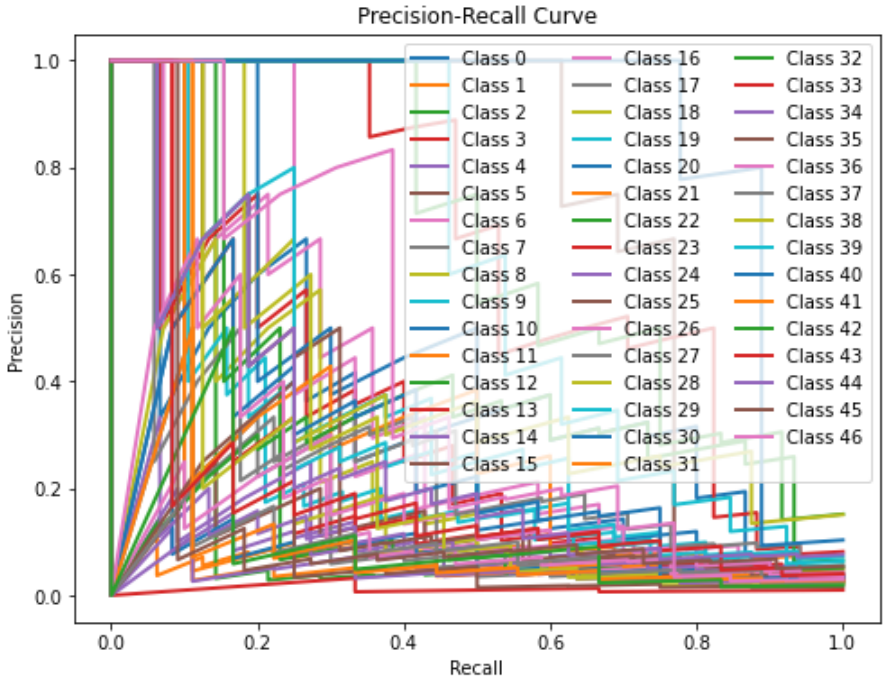
```
18/18 [==============================] - 6s 303ms/step
Classification Report:
```

	precision	recall	f1-score	support
0	0.42	0.33	0.37	15
1	0.00	0.00	0.00	16
2	0.40	0.14	0.21	14
3	0.33	0.27	0.30	15
4	0.00	0.00	0.00	12
5	0.71	0.77	0.74	13
6	0.38	0.36	0.37	14
7	0.20	0.10	0.13	10
8	0.36	0.29	0.32	14
9	0.50	0.11	0.17	19
10	0.26	0.50	0.34	10
11	0.08	0.22	0.12	9
12	0.23	0.23	0.23	13
13	0.67	0.47	0.55	17
14	0.17	0.09	0.12	11
15	0.25	0.14	0.18	7
16	0.33	0.33	0.33	12
17	0.27	0.33	0.30	9
18	0.18	0.25	0.21	8
19	0.40	0.25	0.31	16
20	0.10	0.08	0.09	12
21	0.25	0.50	0.33	10
22	0.33	0.20	0.25	15
23	0.33	0.25	0.29	12
24	0.05	0.07	0.06	15
25	0.28	0.31	0.29	16
26	0.11	0.40	0.17	10
27	0.29	0.24	0.26	17
28	0.19	0.45	0.26	11
29	0.80	0.25	0.38	16
30	0.29	0.42	0.34	12
31	0.06	0.11	0.08	9
32	0.60	0.50	0.55	12
33	0.00	0.00	0.00	3
34	0.00	0.00	0.00	9
35	0.20	0.25	0.22	8
36	0.27	0.18	0.21	17
37	0.00	0.00	0.00	9
38	0.26	0.62	0.37	8
39	1.00	0.46	0.63	13
40	0.88	0.78	0.82	9
41	0.07	0.22	0.11	9
42	0.14	0.17	0.15	6
43	0.50	0.08	0.14	12
44	0.29	0.25	0.27	16
45	1.00	0.09	0.17	11
46	0.27	0.46	0.34	13
accuracy			0.27	564
macro avg	0.31	0.27	0.26	564
weighted avg	0.34	0.27	0.27	564



In [50]:
```
# Obtain the predicted probabilities for each class
y_pred_prob = model.predict(X_test)

# Binarize the true labels
y_test_bin = label_binarize(y_test, classes=np.unique(y_test))

# Calculate precision and recall for each class
precision = dict()
recall = dict()
for i in range(num_classes):
    precision[i], recall[i], _ = precision_recall_curve(y_test_bin[:, i], y_pred_prob[:, i])

# Plot precision-recall curve for each class
plt.figure(figsize=(8, 6))
for i in range(num_classes):
    plt.plot(recall[i], precision[i], lw=2, label=f'Class {i}')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='upper right', bbox_to_anchor=(1.0, 1.0), ncol=3)
plt.show()
```

```
18/18 [==============================] - 5s 295ms/step
```

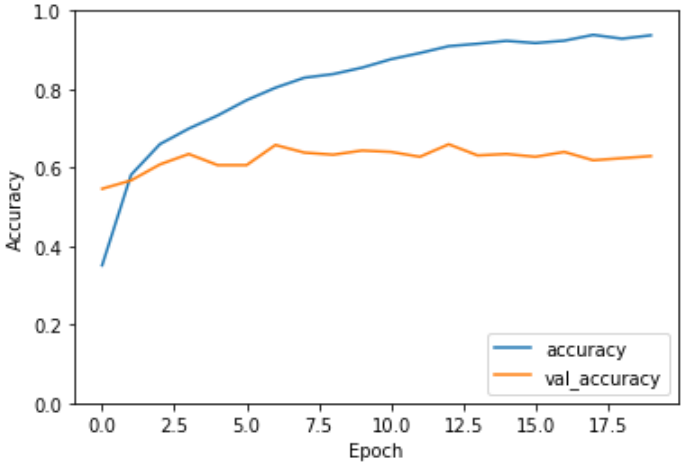


Inception V4 Model

Inception V4 Model using Adam Optimizer

In [61]:
```
# Load the pre-trained InceptionV4 model without the top layer
inceptionv4 = keras.applications.InceptionV3(
    include_top=False,
    weights='imagenet',
    input_shape=(224, 224, 3),
    pooling=None
)

# Add a new top layer for classification
x = layers.GlobalAveragePooling2D()(inceptionv4.output)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.5)(x)
predictions = layers.Dense(num_classes, activation='softmax')(x)

# Define the new model
model = keras.Model(inputs=inceptionv4.input, outputs=predictions)

# Freeze the pre-trained layers
for layer in inceptionv4.layers:
    layer.trainable = False

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

# Train the model
history = model.fit(
    X_train, y_train,
    epochs=20,
    batch_size=32,
    verbose=1,
    validation_data=(X_val, y_val)
)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

```
Epoch 1/20
141/141 [==============================] - 195s 1s/step - loss: 2.6797 - accuracy: 0.3511 - va
l_loss: 1.7677 - val_accuracy: 0.5461
Epoch 2/20
141/141 [==============================] - 191s 1s/step - loss: 1.5153 - accuracy: 0.5816 - va
l_loss: 1.5624 - val_accuracy: 0.5674
Epoch 3/20
141/141 [==============================] - 194s 1s/step - loss: 1.2100 - accuracy: 0.6598 - va
l_loss: 1.4754 - val_accuracy: 0.6082
Epoch 4/20
141/141 [==============================] - 195s 1s/step - loss: 1.0368 - accuracy: 0.6995 - va
l_loss: 1.3463 - val_accuracy: 0.6348
Epoch 5/20
141/141 [==============================] - 188s 1s/step - loss: 0.8762 - accuracy: 0.7329 - va
l_loss: 1.4532 - val_accuracy: 0.6064
Epoch 6/20
141/141 [==============================] - 188s 1s/step - loss: 0.7406 - accuracy: 0.7719 - va
l_loss: 1.4581 - val_accuracy: 0.6064
Epoch 7/20
141/141 [==============================] - 188s 1s/step - loss: 0.6212 - accuracy: 0.8036 - va
l_loss: 1.4069 - val_accuracy: 0.6578
Epoch 8/20
141/141 [==============================] - 178s 1s/step - loss: 0.5506 - accuracy: 0.8291 - va
l_loss: 1.4487 - val_accuracy: 0.6383
Epoch 9/20
141/141 [==============================] - 180s 1s/step - loss: 0.5058 - accuracy: 0.8384 - va
l_loss: 1.4913 - val_accuracy: 0.6330
Epoch 10/20
141/141 [==============================] - 188s 1s/step - loss: 0.4467 - accuracy: 0.8548 - va
l_loss: 1.4454 - val_accuracy: 0.6436
Epoch 11/20
141/141 [==============================] - 188s 1s/step - loss: 0.3833 - accuracy: 0.8763 - va
l_loss: 1.5249 - val_accuracy: 0.6401
Epoch 12/20
141/141 [==============================] - 188s 1s/step - loss: 0.3378 - accuracy: 0.8921 - va
l_loss: 1.4971 - val_accuracy: 0.6277
Epoch 13/20
141/141 [==============================] - 190s 1s/step - loss: 0.2922 - accuracy: 0.9096 - va
l_loss: 1.5887 - val_accuracy: 0.6596
Epoch 14/20
141/141 [==============================] - 188s 1s/step - loss: 0.2661 - accuracy: 0.9158 - va
l_loss: 1.6276 - val_accuracy: 0.6312
Epoch 15/20
141/141 [==============================] - 190s 1s/step - loss: 0.2389 - accuracy: 0.9231 - va
l_loss: 1.6050 - val_accuracy: 0.6348
Epoch 16/20
141/141 [==============================] - 188s 1s/step - loss: 0.2436 - accuracy: 0.9176 - va
l_loss: 1.7046 - val_accuracy: 0.6277
Epoch 17/20
141/141 [==============================] - 187s 1s/step - loss: 0.2356 - accuracy: 0.9235 - va
l_loss: 1.7589 - val_accuracy: 0.6401
Epoch 18/20
141/141 [==============================] - 174s 1s/step - loss: 0.1963 - accuracy: 0.9384 - va
l_loss: 1.7652 - val_accuracy: 0.6188
Epoch 19/20
141/141 [==============================] - 185s 1s/step - loss: 0.2159 - accuracy: 0.9286 - va
l_loss: 1.7723 - val_accuracy: 0.6241
Epoch 20/20
141/141 [==============================] - 188s 1s/step - loss: 0.1978 - accuracy: 0.9373 - va
l_loss: 1.8616 - val_accuracy: 0.6294
```



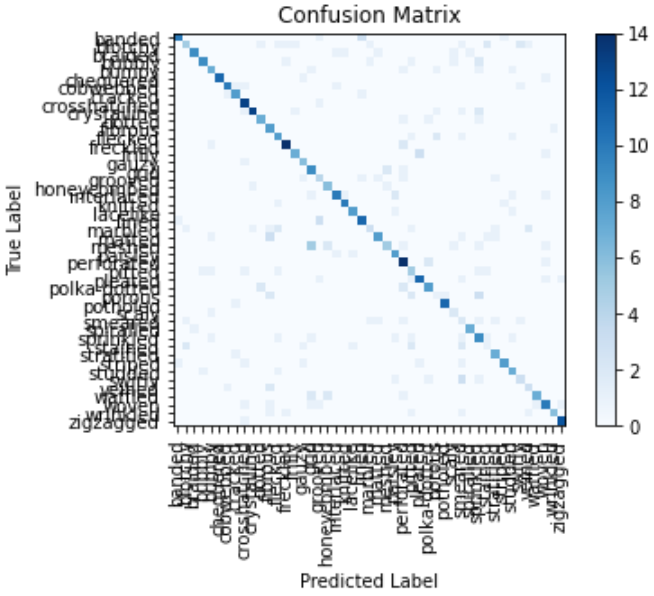
In [62]:
```
# Evaluate the model on the testing set
score = model.evaluate(X_test, y_test, verbose=2)
print('Test Loss:', score[0])
print('Test accuracy:', score[1])
```

```
18/18 - 21s - loss: 1.7414 - accuracy: 0.6401 - 21s/epoch - 1s/step
Test Loss: 1.7413880825042725
Test accuracy: 0.640070915222168
```

In [63]:
```
# Predict classes for the testing set
y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)

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

# Calculate precision, recall, and other metrics
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Calculate and plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.xticks(np.arange(len(class_names)), class_names, rotation=90)
plt.yticks(np.arange(len(class_names)), class_names)
plt.show()
```

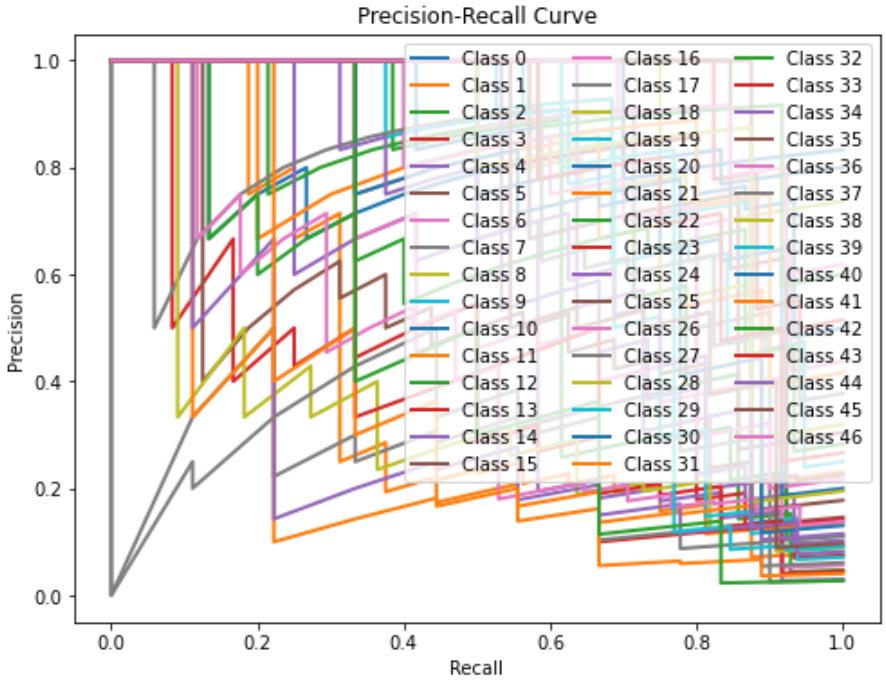
```
18/18 [==============================] - 23s 1s/step
Classification Report:
```

	precision	recall	f1-score	support
0	0.83	0.67	0.74	15
1	0.56	0.31	0.40	16
2	0.90	0.64	0.75	14
3	0.82	0.60	0.69	15
4	0.58	0.58	0.58	12
5	1.00	0.85	0.92	13
6	0.83	0.71	0.77	14
7	0.73	0.80	0.76	10
8	0.65	0.93	0.76	14
9	0.87	0.68	0.76	19
10	0.70	0.70	0.70	10
11	0.44	0.89	0.59	9
12	0.53	0.62	0.57	13
13	0.82	0.82	0.82	17
14	0.78	0.64	0.70	11
15	0.60	0.86	0.71	7
16	0.45	0.75	0.56	12
17	0.36	0.44	0.40	9
18	0.60	0.75	0.67	8
19	0.77	0.62	0.69	16
20	0.83	0.83	0.83	12
21	0.80	0.80	0.80	10
22	0.69	0.73	0.71	15
23	0.50	0.25	0.33	12
24	0.73	0.53	0.62	15
25	0.62	0.31	0.42	16
26	0.62	0.80	0.70	10
27	0.56	0.82	0.67	17
28	0.33	0.45	0.38	11
29	0.73	0.69	0.71	16
30	0.67	0.67	0.67	12
31	0.20	0.11	0.14	9
32	0.92	0.92	0.92	12
33	0.40	0.67	0.50	3
34	0.13	0.22	0.17	9
35	0.88	0.88	0.88	8
36	0.41	0.53	0.46	17
37	0.17	0.11	0.13	9
38	0.64	0.88	0.74	8
39	0.80	0.62	0.70	13
40	0.58	0.78	0.67	9
41	0.43	0.33	0.38	9
42	0.43	0.50	0.46	6
43	0.88	0.58	0.70	12
44	0.62	0.62	0.62	16
45	0.86	0.55	0.67	11
46	0.86	0.92	0.89	13
accuracy			0.64	564
macro avg	0.64	0.64	0.63	564
weighted avg	0.66	0.64	0.64	564



In [64]:
```
# Obtain the predicted probabilities for each class
y_pred_prob = model.predict(X_test)

# Binarize the true labels
y_test_bin = label_binarize(y_test, classes=np.unique(y_test))

# Calculate precision and recall for each class
precision = dict()
recall = dict()
for i in range(num_classes):
    precision[i], recall[i], _ = precision_recall_curve(y_test_bin[:, i], y_pred_prob[:, i])

# Plot precision-recall curve for each class
plt.figure(figsize=(8, 6))
for i in range(num_classes):
    plt.plot(recall[i], precision[i], lw=2, label=f'Class {i}')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='upper right', bbox_to_anchor=(1.0, 1.0), ncol=3)
plt.show()
```

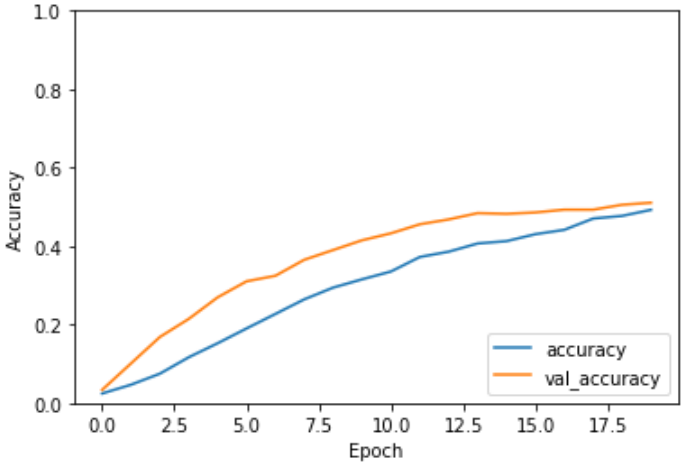
```
18/18 [==============================] - 22s 1s/step
```



Inception V4 Model using SGD Optimizer with 20 Epochs

In [74]:
```
# Load the pre-trained InceptionV4 model without the top layer
inceptionv4 = keras.applications.InceptionV3(
    include_top=False,
    weights='imagenet',
    input_shape=(224, 224, 3),
    pooling=None
)

# Add a new top layer for classification
x = layers.GlobalAveragePooling2D()(inceptionv4.output)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.5)(x)
predictions = layers.Dense(num_classes, activation='softmax')(x)

# Define the new model
model = keras.Model(inputs=inceptionv4.input, outputs=predictions)

# Freeze the pre-trained layers
for layer in inceptionv4.layers:
    layer.trainable = False

# Compile the model
model.compile(
    optimizer=tf.optimizers.SGD(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Train the model
history = model.fit(
    X_train, y_train,
    epochs=20,
    batch_size=32,
    verbose=1,
    validation_data=(X_val, y_val)
)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

```
Epoch 1/20
141/141 [==============================] - 194s 1s/step - loss: 4.2191 - accuracy: 0.0244 - va
l_loss: 3.8667 - val_accuracy: 0.0337
Epoch 2/20
141/141 [==============================] - 189s 1s/step - loss: 3.9424 - accuracy: 0.0468 - va
l_loss: 3.6944 - val_accuracy: 0.1011
Epoch 3/20
141/141 [==============================] - 181s 1s/step - loss: 3.7428 - accuracy: 0.0749 - va
l_loss: 3.5489 - val_accuracy: 0.1684
Epoch 4/20
141/141 [==============================] - 175s 1s/step - loss: 3.5939 - accuracy: 0.1168 - va
l_loss: 3.4218 - val_accuracy: 0.2145
Epoch 5/20
141/141 [==============================] - 174s 1s/step - loss: 3.4284 - accuracy: 0.1525 - va
l_loss: 3.3053 - val_accuracy: 0.2695
Epoch 6/20
141/141 [==============================] - 174s 1s/step - loss: 3.2982 - accuracy: 0.1902 - va
l_loss: 3.1970 - val_accuracy: 0.3103
Epoch 7/20
141/141 [==============================] - 186s 1s/step - loss: 3.1808 - accuracy: 0.2272 - va
l_loss: 3.0935 - val_accuracy: 0.3245
Epoch 8/20
141/141 [==============================] - 189s 1s/step - loss: 3.0475 - accuracy: 0.2644 - va
l_loss: 2.9918 - val_accuracy: 0.3652
Epoch 9/20
141/141 [==============================] - 189s 1s/step - loss: 2.9452 - accuracy: 0.2943 - va
l_loss: 2.8995 - val_accuracy: 0.3901
Epoch 10/20
141/141 [==============================] - 189s 1s/step - loss: 2.8348 - accuracy: 0.3154 - va
l_loss: 2.8088 - val_accuracy: 0.4149
Epoch 11/20
141/141 [==============================] - 189s 1s/step - loss: 2.7562 - accuracy: 0.3353 - va
l_loss: 2.7235 - val_accuracy: 0.4326
Epoch 12/20
141/141 [==============================] - 190s 1s/step - loss: 2.6477 - accuracy: 0.3723 - va
l_loss: 2.6470 - val_accuracy: 0.4557
Epoch 13/20
141/141 [==============================] - 188s 1s/step - loss: 2.5714 - accuracy: 0.3859 - va
l_loss: 2.5732 - val_accuracy: 0.4681
Epoch 14/20
141/141 [==============================] - 189s 1s/step - loss: 2.4900 - accuracy: 0.4067 - va
l_loss: 2.5009 - val_accuracy: 0.4840
Epoch 15/20
141/141 [==============================] - 186s 1s/step - loss: 2.4252 - accuracy: 0.4127 - va
l_loss: 2.4344 - val_accuracy: 0.4823
Epoch 16/20
141/141 [==============================] - 178s 1s/step - loss: 2.3608 - accuracy: 0.4304 - va
l_loss: 2.3747 - val_accuracy: 0.4858
Epoch 17/20
141/141 [==============================] - 243s 2s/step - loss: 2.2846 - accuracy: 0.4415 - va
l_loss: 2.3161 - val_accuracy: 0.4929
Epoch 18/20
141/141 [==============================] - 26649s 190s/step - loss: 2.2088 - accuracy: 0.4703
- val_loss: 2.2647 - val_accuracy: 0.4929
Epoch 19/20
141/141 [==============================] - 290s 2s/step - loss: 2.1726 - accuracy: 0.4770 - va
l_loss: 2.2161 - val_accuracy: 0.5053
Epoch 20/20
141/141 [==============================] - 295s 2s/step - loss: 2.0925 - accuracy: 0.4925 - va
l_loss: 2.1675 - val_accuracy: 0.5106
```



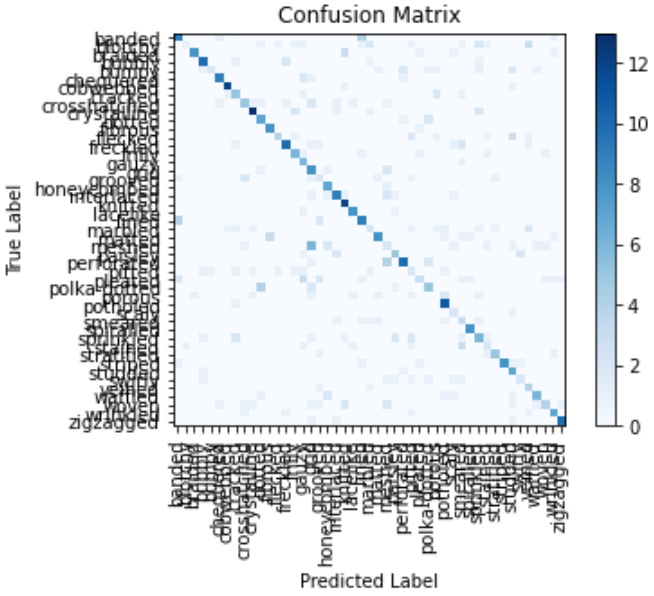
In [75]:
```
# Evaluate the model on the testing set
score = model.evaluate(X_test, y_test, verbose=2)
print('Test Loss:', score[0])
print('Test accuracy:', score[1])
```

```
18/18 - 32s - loss: 2.0861 - accuracy: 0.5177 - 32s/epoch - 2s/step
Test Loss: 2.0860631465911865
Test accuracy: 0.5177304744720459
```

In [76]:
```
# Predict classes for the testing set
y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)

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

# Calculate precision, recall, and other metrics
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Calculate and plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.xticks(np.arange(len(class_names)), class_names, rotation=90)
plt.yticks(np.arange(len(class_names)), class_names)
plt.show()
```

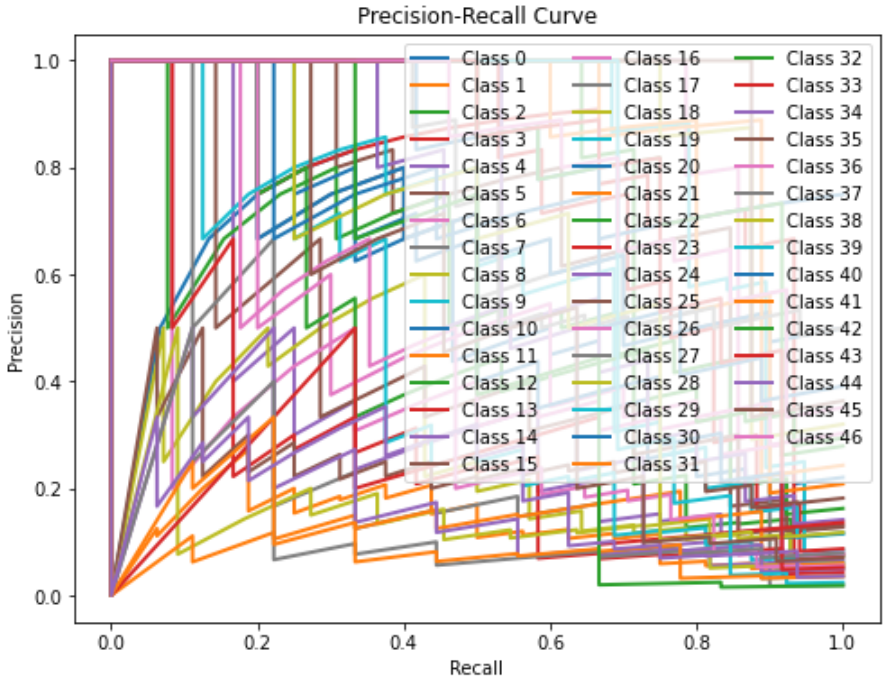
```
18/18 [==============================] - 39s 2s/step
Classification Report:
```

	precision	recall	f1-score	support
0	0.53	0.60	0.56	15
1	0.50	0.06	0.11	16
2	0.89	0.57	0.70	14
3	0.67	0.67	0.67	15
4	0.67	0.17	0.27	12
5	0.75	0.69	0.72	13
6	0.86	0.86	0.86	14
7	0.42	0.50	0.45	10
8	0.50	0.36	0.42	14
9	0.72	0.68	0.70	19
10	0.47	0.70	0.56	10
11	0.73	0.89	0.80	9
12	0.67	0.31	0.42	13
13	0.71	0.59	0.65	17
14	0.55	0.55	0.55	11
15	0.33	0.86	0.48	7
16	0.30	0.67	0.41	12
17	0.20	0.22	0.21	9
18	0.54	0.88	0.67	8
19	0.60	0.56	0.58	16
20	0.60	1.00	0.75	12
21	0.73	0.80	0.76	10
22	0.50	0.60	0.55	15
23	0.22	0.17	0.19	12
24	0.67	0.53	0.59	15
25	0.13	0.12	0.13	16
26	0.45	0.50	0.48	10
27	0.67	0.59	0.62	17
28	0.15	0.18	0.17	11
29	0.43	0.19	0.26	16
30	0.45	0.42	0.43	12
31	0.00	0.00	0.00	9
32	0.61	0.92	0.73	12
33	0.25	0.67	0.36	3
34	0.20	0.33	0.25	9
35	0.80	1.00	0.89	8
36	0.40	0.35	0.38	17
37	0.25	0.22	0.24	9
38	0.71	0.62	0.67	8
39	1.00	0.62	0.76	13
40	0.37	0.78	0.50	9
41	0.40	0.22	0.29	9
42	0.33	0.67	0.44	6
43	0.67	0.50	0.57	12
44	0.57	0.25	0.35	16
45	0.58	0.64	0.61	11
46	0.83	0.77	0.80	13
accuracy			0.52	564
macro avg	0.52	0.53	0.50	564
weighted avg	0.54	0.52	0.51	564



In [77]:
```
# Obtain the predicted probabilities for each class
y_pred_prob = model.predict(X_test)

# Binarize the true labels
y_test_bin = label_binarize(y_test, classes=np.unique(y_test))

# Calculate precision and recall for each class
precision = dict()
recall = dict()
for i in range(num_classes):
    precision[i], recall[i], _ = precision_recall_curve(y_test_bin[:, i], y_pred_prob[:, i])

# Plot precision-recall curve for each class
plt.figure(figsize=(8, 6))
for i in range(num_classes):
    plt.plot(recall[i], precision[i], lw=2, label=f'Class {i}')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='upper right', bbox_to_anchor=(1.0, 1.0), ncol=3)
plt.show()
```

```
18/18 [==============================] - 32s 2s/step
```



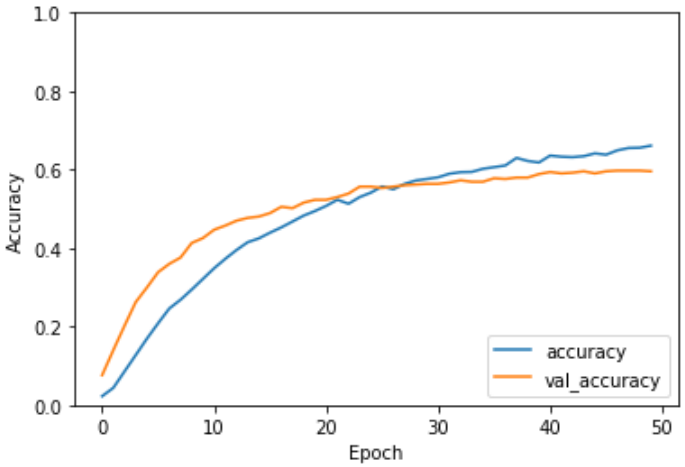
Inception V4 Model using SGD Optimizer with 50 Epochs

In [94]:
```
# Load the pre-trained InceptionV4 model without the top layer
inceptionv4 = keras.applications.InceptionV3(
    include_top=False,
    weights='imagenet',
    input_shape=(224, 224, 3),
    pooling=None
)

# Add a new top layer for classification
x = layers.GlobalAveragePooling2D()(inceptionv4.output)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.5)(x)
predictions = layers.Dense(num_classes, activation='softmax')(x)

# Define the new model
model = keras.Model(inputs=inceptionv4.input, outputs=predictions)

# Freeze the pre-trained layers
for layer in inceptionv4.layers:
    layer.trainable = False

# Compile the model
model.compile(
    optimizer=tf.optimizers.SGD(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Train the model
history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=32,
    verbose=1,
    validation_data=(X_val, y_val)
)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

```
Epoch 1/50
141/141 [==============================] - 308s 2s/step - loss: 4.2053 - accuracy: 0.0226 - va
l_loss: 3.7669 - val_accuracy: 0.0762
Epoch 2/50
141/141 [==============================] - 273s 2s/step - loss: 3.8973 - accuracy: 0.0439 - va
l_loss: 3.6132 - val_accuracy: 0.1401
Epoch 3/50
141/141 [==============================] - 248s 2s/step - loss: 3.7045 - accuracy: 0.0858 - va
l_loss: 3.4800 - val_accuracy: 0.2021
Epoch 4/50
141/141 [==============================] - 189s 1s/step - loss: 3.5298 - accuracy: 0.1274 - va
l_loss: 3.3581 - val_accuracy: 0.2624
Epoch 5/50
141/141 [==============================] - 207s 1s/step - loss: 3.3801 - accuracy: 0.1693 - va
l_loss: 3.2417 - val_accuracy: 0.2996
Epoch 6/50
141/141 [==============================] - 202s 1s/step - loss: 3.2572 - accuracy: 0.2086 - va
l_loss: 3.1301 - val_accuracy: 0.3387
Epoch 7/50
141/141 [==============================] - 210s 1s/step - loss: 3.1176 - accuracy: 0.2462 - va
l_loss: 3.0272 - val_accuracy: 0.3599
Epoch 8/50
141/141 [==============================] - 204s 1s/step - loss: 3.0153 - accuracy: 0.2684 - va
l_loss: 2.9276 - val_accuracy: 0.3759
Epoch 9/50
141/141 [==============================] - 200s 1s/step - loss: 2.9054 - accuracy: 0.2943 - va
l_loss: 2.8334 - val_accuracy: 0.4131
Epoch 10/50
141/141 [==============================] - 196s 1s/step - loss: 2.7934 - accuracy: 0.3216 - va
l_loss: 2.7439 - val_accuracy: 0.4255
Epoch 11/50
141/141 [==============================] - 199s 1s/step - loss: 2.7249 - accuracy: 0.3486 - va
l_loss: 2.6631 - val_accuracy: 0.4468
Epoch 12/50
141/141 [==============================] - 212s 2s/step - loss: 2.6170 - accuracy: 0.3728 - va
l_loss: 2.5830 - val_accuracy: 0.4574
Epoch 13/50
141/141 [==============================] - 194s 1s/step - loss: 2.5321 - accuracy: 0.3952 - va
l_loss: 2.5070 - val_accuracy: 0.4699
Epoch 14/50
141/141 [==============================] - 192s 1s/step - loss: 2.4507 - accuracy: 0.4151 - va
l_loss: 2.4347 - val_accuracy: 0.4770
Epoch 15/50
141/141 [==============================] - 180s 1s/step - loss: 2.3984 - accuracy: 0.4249 - va
l_loss: 2.3754 - val_accuracy: 0.4805
Epoch 16/50
141/141 [==============================] - 188s 1s/step - loss: 2.3223 - accuracy: 0.4395 - va
l_loss: 2.3174 - val_accuracy: 0.4894
Epoch 17/50
141/141 [==============================] - 183s 1s/step - loss: 2.2603 - accuracy: 0.4528 - va
l_loss: 2.2599 - val_accuracy: 0.5053
Epoch 18/50
141/141 [==============================] - 177s 1s/step - loss: 2.1867 - accuracy: 0.4679 - va
l_loss: 2.2037 - val_accuracy: 0.5018
Epoch 19/50
141/141 [==============================] - 178s 1s/step - loss: 2.1221 - accuracy: 0.4829 - va
l_loss: 2.1542 - val_accuracy: 0.5160
Epoch 20/50
141/141 [==============================] - 187s 1s/step - loss: 2.0734 - accuracy: 0.4945 - va
l_loss: 2.1104 - val_accuracy: 0.5230
Epoch 21/50
141/141 [==============================] - 190s 1s/step - loss: 2.0360 - accuracy: 0.5075 - va
l_loss: 2.0699 - val_accuracy: 0.5230
Epoch 22/50
141/141 [==============================] - 196s 1s/step - loss: 1.9736 - accuracy: 0.5233 - va
l_loss: 2.0335 - val_accuracy: 0.5301
Epoch 23/50
141/141 [==============================] - 192s 1s/step - loss: 1.9474 - accuracy: 0.5131 - va
```

```
l_loss: 1.9941 - val_accuracy: 0.5390
Epoch 24/50
141/141 [==============================] - 192s 1s/step - loss: 1.9078 - accuracy: 0.5301 - va
l_loss: 1.9589 - val_accuracy: 0.5567
Epoch 25/50
141/141 [==============================] - 191s 1s/step - loss: 1.8622 - accuracy: 0.5414 - va
l_loss: 1.9273 - val_accuracy: 0.5567
Epoch 26/50
141/141 [==============================] - 7267s 52s/step - loss: 1.8251 - accuracy: 0.5572 -
val_loss: 1.9022 - val_accuracy: 0.5532
Epoch 27/50
141/141 [==============================] - 254s 2s/step - loss: 1.7921 - accuracy: 0.5499 - va
l_loss: 1.8730 - val_accuracy: 0.5567
Epoch 28/50
141/141 [==============================] - 261s 2s/step - loss: 1.7471 - accuracy: 0.5636 - va
l_loss: 1.8494 - val_accuracy: 0.5603
Epoch 29/50
141/141 [==============================] - 252s 2s/step - loss: 1.7119 - accuracy: 0.5723 - va
l_loss: 1.8235 - val_accuracy: 0.5621
Epoch 30/50
141/141 [==============================] - 244s 2s/step - loss: 1.6806 - accuracy: 0.5762 - va
l_loss: 1.8010 - val_accuracy: 0.5638
Epoch 31/50
141/141 [==============================] - 253s 2s/step - loss: 1.6706 - accuracy: 0.5805 - va
l_loss: 1.7796 - val_accuracy: 0.5638
Epoch 32/50
141/141 [==============================] - 260s 2s/step - loss: 1.6390 - accuracy: 0.5895 - va
l_loss: 1.7600 - val_accuracy: 0.5674
Epoch 33/50
141/141 [==============================] - 285s 2s/step - loss: 1.6114 - accuracy: 0.5935 - va
l_loss: 1.7415 - val_accuracy: 0.5727
Epoch 34/50
141/141 [==============================] - 274s 2s/step - loss: 1.6093 - accuracy: 0.5942 - va
l_loss: 1.7247 - val_accuracy: 0.5691
Epoch 35/50
141/141 [==============================] - 259s 2s/step - loss: 1.5563 - accuracy: 0.6022 - va
l_loss: 1.7094 - val_accuracy: 0.5691
Epoch 36/50
141/141 [==============================] - 277s 2s/step - loss: 1.5416 - accuracy: 0.6064 - va
l_loss: 1.6944 - val_accuracy: 0.5780
Epoch 37/50
141/141 [==============================] - 345s 2s/step - loss: 1.5320 - accuracy: 0.6106 - va
l_loss: 1.6802 - val_accuracy: 0.5762
Epoch 38/50
141/141 [==============================] - 194s 1s/step - loss: 1.4944 - accuracy: 0.6299 - va
l_loss: 1.6654 - val_accuracy: 0.5798
Epoch 39/50
141/141 [==============================] - 199s 1s/step - loss: 1.4747 - accuracy: 0.6221 - va
l_loss: 1.6516 - val_accuracy: 0.5798
Epoch 40/50
141/141 [==============================] - 203s 1s/step - loss: 1.4760 - accuracy: 0.6181 - va
l_loss: 1.6415 - val_accuracy: 0.5887
Epoch 41/50
141/141 [==============================] - 204s 1s/step - loss: 1.4365 - accuracy: 0.6359 - va
l_loss: 1.6288 - val_accuracy: 0.5940
Epoch 42/50
141/141 [==============================] - 197s 1s/step - loss: 1.4207 - accuracy: 0.6330 - va
l_loss: 1.6179 - val_accuracy: 0.5904
Epoch 43/50
141/141 [==============================] - 192s 1s/step - loss: 1.4128 - accuracy: 0.6319 - va
l_loss: 1.6073 - val_accuracy: 0.5922
Epoch 44/50
141/141 [==============================] - 182s 1s/step - loss: 1.3996 - accuracy: 0.6343 - va
l_loss: 1.5994 - val_accuracy: 0.5957
Epoch 45/50
141/141 [==============================] - 178s 1s/step - loss: 1.3721 - accuracy: 0.6414 - va
l_loss: 1.5907 - val_accuracy: 0.5904
Epoch 46/50
```

```
141/141 [==============================] - 182s 1s/step - loss: 1.3627 - accuracy: 0.6381 - va
l_loss: 1.5764 - val_accuracy: 0.5957
Epoch 47/50
141/141 [==============================] - 182s 1s/step - loss: 1.3490 - accuracy: 0.6492 - va
l_loss: 1.5667 - val_accuracy: 0.5975
Epoch 48/50
141/141 [==============================] - 192s 1s/step - loss: 1.3264 - accuracy: 0.6551 - va
l_loss: 1.5623 - val_accuracy: 0.5975
Epoch 49/50
141/141 [==============================] - 202s 1s/step - loss: 1.3078 - accuracy: 0.6558 - va
l_loss: 1.5541 - val_accuracy: 0.5975
Epoch 50/50
141/141 [==============================] - 190s 1s/step - loss: 1.2964 - accuracy: 0.6611 - va
l_loss: 1.5475 - val_accuracy: 0.5957
```



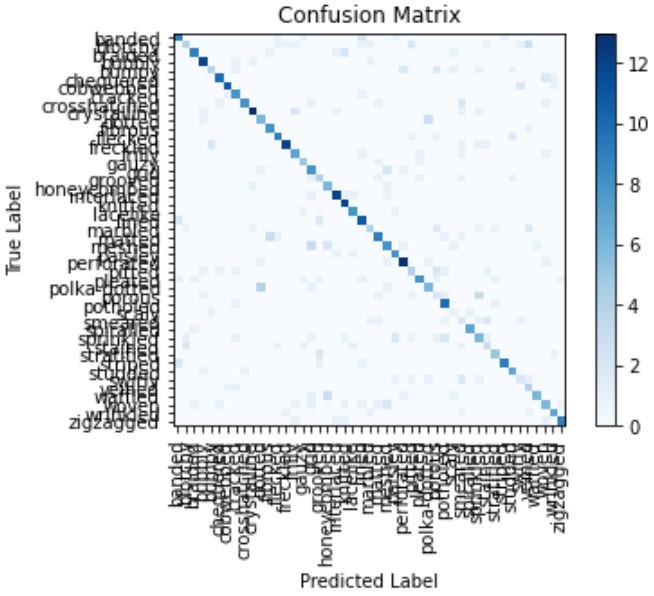
In [96]:
```
# Evaluate the model on the testing set
score = model.evaluate(X_test, y_test, verbose=2)
print('Test Loss:', score[0])
print('Test accuracy:', score[1])
```

```
18/18 - 21s - loss: 1.5318 - accuracy: 0.6064 - 21s/epoch - 1s/step
Test Loss: 1.5318270921707153
Test accuracy: 0.6063829660415649
```

In [97]:
```
# Predict classes for the testing set
y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)

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

# Calculate precision, recall, and other metrics
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Calculate and plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.xticks(np.arange(len(class_names)), class_names, rotation=90)
plt.yticks(np.arange(len(class_names)), class_names)
plt.show()
```

```
18/18 [==============================] - 24s 1s/step
Classification Report:
```

	precision	recall	f1-score	support
0	0.64	0.60	0.62	15
1	0.50	0.25	0.33	16
2	0.75	0.64	0.69	14
3	0.71	0.80	0.75	15
4	0.57	0.33	0.42	12
5	0.77	0.77	0.77	13
6	0.85	0.79	0.81	14
7	0.57	0.80	0.67	10
8	0.89	0.57	0.70	14
9	0.81	0.68	0.74	19
10	0.50	0.60	0.55	10
11	0.67	0.89	0.76	9
12	0.69	0.69	0.69	13
13	0.86	0.71	0.77	17
14	0.58	0.64	0.61	11
15	0.56	0.71	0.63	7
16	0.44	0.67	0.53	12
17	0.40	0.44	0.42	9
18	0.55	0.75	0.63	8
19	0.75	0.75	0.75	16
20	0.67	1.00	0.80	12
21	0.73	0.80	0.76	10
22	0.58	0.73	0.65	15
23	0.50	0.33	0.40	12
24	0.60	0.60	0.60	15
25	0.50	0.50	0.50	16
26	0.62	0.80	0.70	10
27	0.72	0.76	0.74	17
28	0.40	0.36	0.38	11
29	0.73	0.50	0.59	16
30	0.55	0.50	0.52	12
31	0.18	0.22	0.20	9
32	0.62	0.83	0.71	12
33	0.20	0.33	0.25	3
34	0.15	0.22	0.18	9
35	0.88	0.88	0.88	8
36	0.40	0.35	0.38	17
37	0.33	0.33	0.33	9
38	0.62	0.62	0.62	8
39	0.90	0.69	0.78	13
40	0.64	0.78	0.70	9
41	0.33	0.22	0.27	9
42	0.33	0.67	0.44	6
43	0.86	0.50	0.63	12
44	0.50	0.38	0.43	16
45	0.70	0.64	0.67	11
46	0.82	0.69	0.75	13
accuracy			0.61	564
macro avg	0.60	0.60	0.59	564
weighted avg	0.62	0.61	0.60	564



In [98]:
```
# Obtain the predicted probabilities for each class
y_pred_prob = model.predict(X_test)

# Binarize the true labels
y_test_bin = label_binarize(y_test, classes=np.unique(y_test))

# Calculate precision and recall for each class
precision = dict()
recall = dict()
for i in range(num_classes):
    precision[i], recall[i], _ = precision_recall_curve(y_test_bin[:, i], y_pred_prob[:, i])

# Plot precision-recall curve for each class
plt.figure(figsize=(8, 6))
for i in range(num_classes):
    plt.plot(recall[i], precision[i], lw=2, label=f'Class {i}')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='upper right', bbox_to_anchor=(1.0, 1.0), ncol=3)
plt.show()
```

```
18/18 [==============================] - 21s 1s/step
```