Parker Christenson Assignment 4

Building a Plant Classification Model Using Convolutional Neural Networks

`Objective`

- Your task for this assignment is to build a plant classification model using CNNs and the Flavia dataset.

`Dataset`

- You will be using the Flavia dataset Download Flavia dataset, which contains 1907 images of leaves from 32 different plant species. Each image is labeled with its corresponding species name.

`Note` : This zipped file is 931MB and -will take some time to download depending on your connection speed.

`Instructions`

> - Preprocess the data: You will need to preprocess the Flavia dataset by resizing the images to a fixed size, converting them to grayscale, and splitting them into training, validation, and test sets.
> - Build a CNN model: You will need to design and implement a CNN model architecture that can effectively classify plant leaves based on their images.
> - Train the model: You will need to train the CNN model on the preprocessed Flavia dataset using appropriate hyperparameters and regularization techniques.
> - Evaluate the model: You will need to evaluate the performance of the trained CNN model on the test set of the Flavia dataset using appropriate evaluation metrics such as accuracy, precision, recall, and F1 score.
> - Analyze the results: You will need to analyze the performance of the model and identify any potential areas for improvement. You can visualize the learned features of the model, plot confusion matrices, and perform other analysis techniques to gain insights into the model's behavior.

In [ ]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import tensorflow as tf
from PIL import Image
```

```
from tensorflow.keras import layers, models
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

In [ ]:
```
# so this is the labels for all of the images in this csv file, so we can use this
names_df = pd.read_csv(f"C:\\MSAAI_assignment_datasets\\flavia_dataset\\Leaves\\all
names_df.head()
```

Out[ ]:

	Unnamed: 0	id	y
0	0	1300.jpg	5
1	1	3152.jpg	23
2	2	1439.jpg	9
3	3	1243.jpg	4
4	4	1186.jpg	3

In [ ]:
```
# getting the counts of each label, which is the y column.

names_df['y'].value_counts()
```

Out[ ]:
```
y
16    77
4     73
3     72
19    66
2     65
23    65
13    65
29    64
1     63
11    63
17    62
6     62
18    61
20    60
14    60
0     59
9     59
28    57
15    56
31    56
5     56
27    55
22    55
8     55
21    55
24    54
30    53
26    53
25    52
7     52
12    52
10    50
Name: count, dtype: int64
```

So group 16 has the highest amount of images with 77 images. We might have to lower the count of the images in the 16 group to even out the data set for better results.

In [ ]:
```python
# we are going to write a function that pulls the images label from the id column i


def load_and_resize_images(csv_file, image_folder, target_size=(533, 400)):
    df = pd.read_csv(csv_file)

    images = []
    labels = []

    # iterate through the df rows
    for index, row in df.iterrows():
        image_id = row['id']
        label = row['y']

        # building the image pathway
```

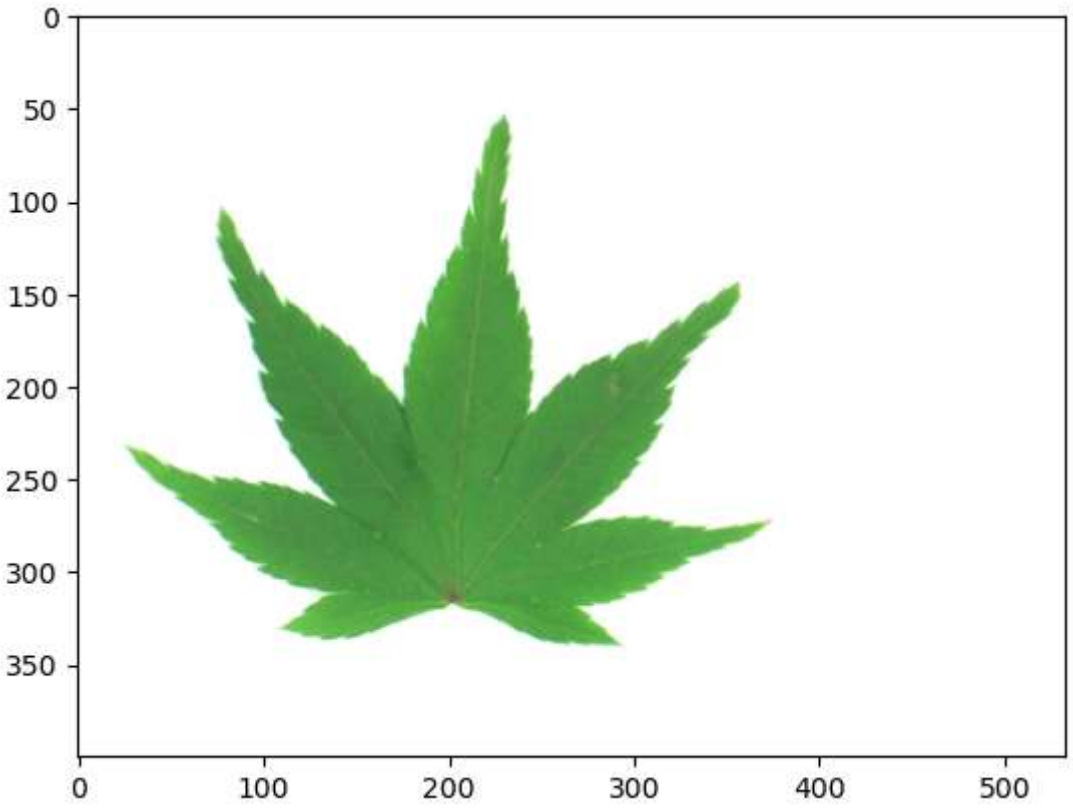
```
        image_path = os.path.join(image_folder, image_id)

        try:
            # open
            with Image.open(image_path) as img:
                # resize
                img_resized = img.resize(target_size)
                # convert the image to an array
                images.append(np.array(img_resized))
                labels.append(label)
        # error handling
        except Exception as e:
            print(f"Error loading image {image_id}: {e}")

    # convert lists into arrays
    images_array = np.array(images)
    labels_array = np.array(labels)

    return images_array, labels_array
```

In [ ]:
```
# running the function
csv_file_path = r"C:\MSAAI_assignment_datasets\flavia_dataset\Leaves\all.csv"
image_folder_path = r"C:\MSAAI_assignment_datasets\flavia_dataset\Leaves"

images, labels = load_and_resize_images(csv_file_path, image_folder_path)

print(f"loaded {len(images)} images, their shape is {images[0].shape}")
```

```
loaded 1907 images, their shape is (400, 533, 3)
```

In [ ]:
```
# displaying the first image using matplotlib
plt.imshow(images[0])
```

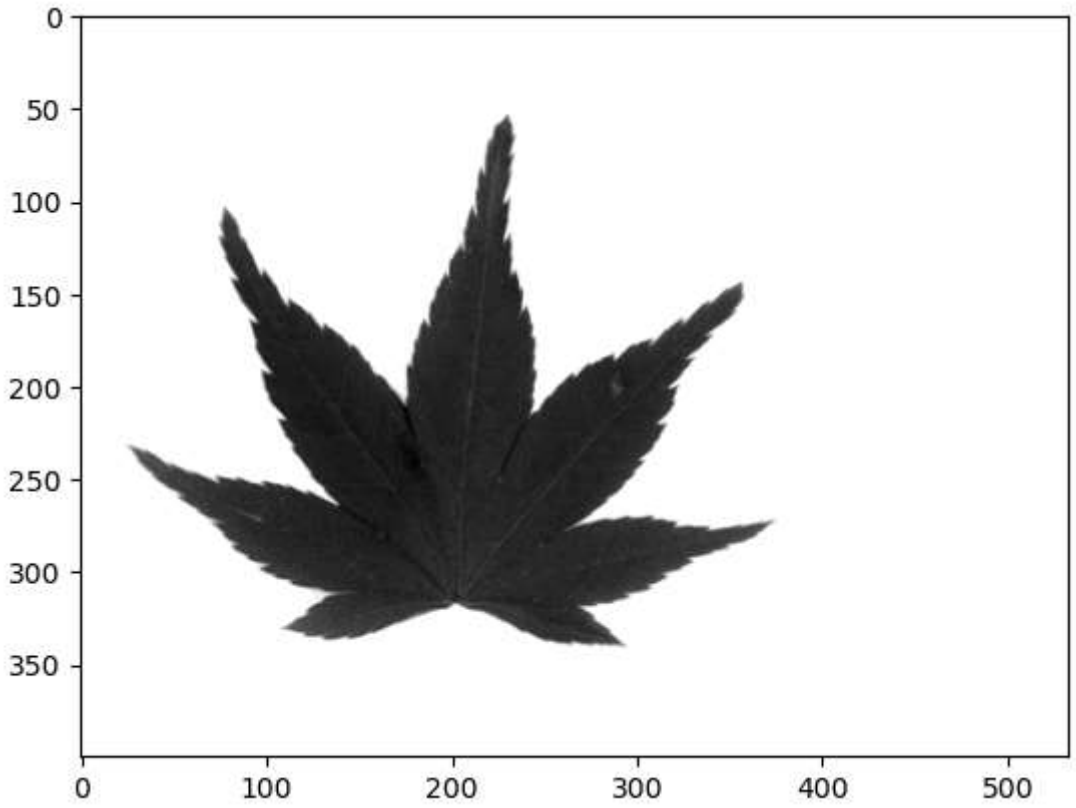
Out[ ]:
```
<matplotlib.image.AxesImage at 0x1174f4ba0a0>
```



In [ ]:
```
# load and convert to grayscale
images, labels = load_and_resize_images(csv_file_path, image_folder_path)
images_gray = np.mean(images, axis=-1, keepdims=True)
```

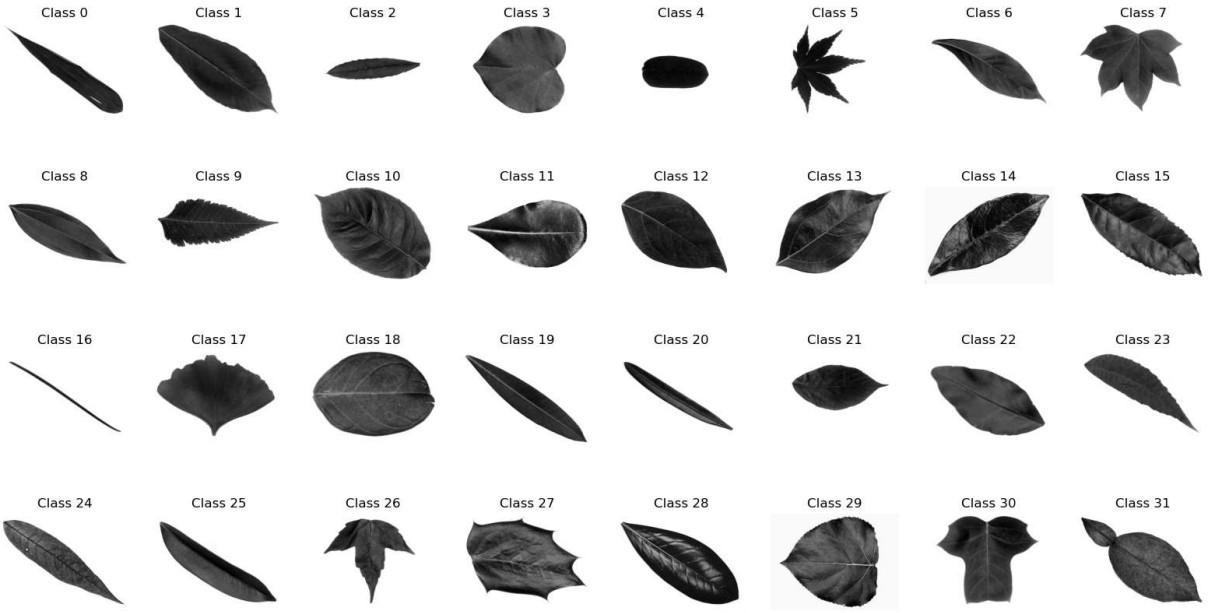
In [ ]:
```
# displaying the first image in grayscale
plt.imshow(images_gray[0].squeeze(), cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x1174e8832e0>
```



In [ ]:
```
labels_one_hot = to_categorical(labels, num_classes=32)
X_train, X_val, y_train, y_val = train_test_split(images_gray / 255.0, labels_one_h
```

In [ ]:
```
# now printing all pictures in one class to make sure they are the same
fig, axes = plt.subplots(4, 8, figsize=(20, 10))
axes = axes.ravel()

for i in range(32):
    axes[i].imshow(X_train[y_train[:, i] == 1][0].squeeze(), cmap='gray')
    axes[i].set_title(f"Class {i}")
    axes[i].axis('off')

plt.subplots_adjust(hspace=0.5)
```



In [ ]:
```
# now we are going to print the matrix of the first image
print(X_train[0])
```

```
[[[0.90718954]
  [0.9254902 ]
  [0.94379085]
  ...
  [0.98823529]
  [0.98823529]
  [0.98823529]]

 [[0.92026144]
  [0.9372549 ]
  [0.96078431]
  ...
  [0.98823529]
  [0.98823529]
  [0.98823529]]

 [[0.93464052]
  [0.9503268 ]
  [0.96732026]
  ...
  [0.98823529]
  [0.98823529]
  [0.98823529]]

 ...

 [[0.87581699]
  [0.89673203]
  [0.91111111]
  ...
  [0.98823529]
  [0.98823529]
  [0.98823529]]

 [[0.8627451 ]
  [0.87973856]
  [0.89542484]
  ...
  [0.98823529]
  [0.98823529]
  [0.98823529]]

 [[0.8627451 ]
  [0.87712418]
  [0.89411765]
  ...
  [0.98823529]
  [0.98823529]
  [0.98823529]]]
```

Making a "NEAT" CNN model for the Flavia dataset

`Explination of this method that I chose:`

So I hate the Idea of making a model, 5-6 times to make sure that I am able to accurately have the model be able to predict accurately. So I was watching youtube videos this week, and I stumbled across a video that was talking about the NEAT algorithm, but it was being applied to video games and how the AI was being trained on Donkey Kong. So I thought to myself, why not apply this to the CNN model that I am trying to make, so I dont have to spend hours sitting at my computer training models over and over again. So I am going to try to apply the NEAT algorithm to the CNN model that I am trying to make to classify the images.

Link to the video here: https://youtu.be/ovIykchkW5I?si=M9ZjAEfccFUjVrkB

In [ ]:
```python
# making a base model
def build_cnn_model(num_layers, num_filters, input_shape, num_classes=32):
    model = models.Sequential()
    model.add(layers.InputLayer(input_shape=input_shape))

    for _ in range(4):
        model.add(layers.Conv2D(num_filters, (3, 3), activation='relu'))
        model.add(layers.MaxPooling2D((2, 2)))

    for _ in range(num_layers - 4):
        model.add(layers.Conv2D(num_filters, (3, 3), activation='relu'))
        model.add(layers.MaxPooling2D((2, 2)))

    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(num_classes, activation='softmax'))

    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accu
    return model
```

Explaining `MetricsCallback` Class

`MetricsCallback`. This callback is designed to be used with a TensorFlow Keras model during training to compute and track evaluation metrics such as precision, recall, and F1 score on a validation dataset. It also stops the training process when a specified target accuracy is reached.

The *`MetricsCallback`* class inherits from the *`tf.keras.callbacks.Callback`* class, which is a base class for implementing callbacks in TensorFlow Keras. The *`__init__`* method of the *`MetricsCallback`* class initializes the callback object and takes three parameters: *`X_val`*, *`y_val`*, and *`target_accuracy`*. These represent the validation data features, validation data labels, and the desired target accuracy, respectively. These parameters are stored as instance variables for later use.

The `on_epoch_end` method is a callback method that is called at the end of each epoch during training. Inside this method, the callback computes the predicted labels for the validation data using the model's `predict` method. The `np.argmax` function is used to convert the predicted probabilities into class labels by selecting the index of the maximum value along the last axis.

Next, the true labels for the validation data are obtained by applying the `np.argmax` function to the `y_val` array. The callback then calculates the precision, recall, and F1 score using the `precision_score`, `recall_score`, and `f1_score` functions from the `sklearn.metrics` module. These functions require the true labels and predicted labels as inputs, along with optional parameters such as the averaging method.

The computed precision, recall, and F1 score are appended to separate lists `val_precisions`, `val_recalls`, and `val_f1s` to track the metrics over all epochs. The validation accuracy for the current epoch is also retrieved from the `logs` dictionary.

Finally, the callback prints the computed metrics using the `print` function, displaying the precision, recall, and F1 score. The callback also checks if the current validation accuracy exceeds the best accuracy observed so far, and updates the best accuracy if so. If the best accuracy meets or exceeds the target accuracy, the callback stops the training process by setting `self.model.stop_training` to `True`.

In [ ]:
```
# Metrics callback class

class MetricsCallback(tf.keras.callbacks.Callback):
    def __init__(self, X_val, y_val, target_accuracy):
        super(MetricsCallback, self).__init__()
        self.X_val = X_val
        self.y_val = y_val
        self.val_precisions = []
        self.val_recalls = []
        self.val_f1s = []
        self.target_accuracy = target_accuracy
        self.best_accuracy = 0

    def on_epoch_end(self, epoch, logs=None):
        val_predict = np.argmax(self.model.predict(self.X_val), axis=-1)
        val_true = np.argmax(self.y_val, axis=-1)
        precision = precision_score(val_true, val_predict, average='weighted')
        recall = recall_score(val_true, val_predict, average='weighted')
        f1 = f1_score(val_true, val_predict, average='weighted')
        accuracy = logs['val_accuracy']
        self.val_precisions.append(precision)
        self.val_recalls.append(recall)
        self.val_f1s.append(f1)
        print(f" - val_precision: {precision:.4f} - val_recall: {recall:.4f} - val_

        # checking to see if target accuracy is reached
        if accuracy > self.best_accuracy:
            self.best_accuracy = accuracy
```

```
        if self.best_accuracy >= self.target_accuracy:
            self.model.stop_training = True
```

In [ ]:
```
# training loop for the model

def train_evolving_cnn(X_train, y_train, X_val, y_val, target_accuracy=0.90, max_ep
    input_shape = X_train.shape[1:]
    num_layers = 4  # Start with 4 layers
    num_filters = 32
    best_accuracy = 0
    best_model = None

    train_accuracies = []
    val_accuracies = []
    val_precisions = []
    val_recalls = []
    val_f1s = []

    while best_accuracy < target_accuracy:
        print(f"Training model with {num_layers} layers and {num_filters} filters p

        model = build_cnn_model(num_layers, num_filters, input_shape, num_classes)

        metrics_callback = MetricsCallback(X_val, y_val, target_accuracy)

        history = model.fit(X_train, y_train, epochs=max_epochs_per_stage, validati

        train_accuracies.extend(history.history['accuracy'])
        val_accuracies.extend(history.history['val_accuracy'])
        val_precisions.extend(metrics_callback.val_precisions)
        val_recalls.extend(metrics_callback.val_recalls)
        val_f1s.extend(metrics_callback.val_f1s)

        best_epoch_accuracy = metrics_callback.best_accuracy

        if best_epoch_accuracy > best_accuracy:
            best_accuracy = best_epoch_accuracy
            best_model = model
            print(f"New best accuracy: {best_accuracy}")

        if best_accuracy >= target_accuracy:
            break

        num_layers += 1
        num_filters += 16

    return best_model, best_accuracy, train_accuracies, val_accuracies, val_precisi
```

In [ ]:
```
# run the custom training loop
best_model, best_accuracy, train_accuracies, val_accuracies, val_precisions, val_re

# final print
print(f"TRAINING DONE! THE BEST VAL ACCURACY WAS: {best_accuracy}")
```

```
Training model with 4 layers and 32 filters per layer
Epoch 1/10
12/12 [==============================] - 3s 215ms/steps: 3
 - val_precision: 0.1640 - val_recall: 0.2461 - val_f1: 0.1644
48/48 [==============================] - 40s 830ms/step - loss: 3.3133 - accuracy:
0.0931 - val_loss: 2.6936 - val_accuracy: 0.2461
Epoch 2/10
c:\Users\tehwh\anaconda3\envs\ml\lib\site-packages\sklearn\metrics\_classification.p
y:1469: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this behavio
r.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
12/12 [==============================] - 3s 216ms/steps: 1
 - val_precision: 0.6895 - val_recall: 0.6466 - val_f1: 0.6310
48/48 [==============================] - 41s 852ms/step - loss: 1.9938 - accuracy:
0.4426 - val_loss: 1.3083 - val_accuracy: 0.6466
Epoch 3/10
12/12 [==============================] - 3s 210ms/steps: 0
 - val_precision: 0.7757 - val_recall: 0.7382 - val_f1: 0.7312
48/48 [==============================] - 40s 846ms/step - loss: 0.9877 - accuracy:
0.7331 - val_loss: 0.9826 - val_accuracy: 0.7382
Epoch 4/10
12/12 [==============================] - 2s 195ms/steps: 0
 - val_precision: 0.8106 - val_recall: 0.8010 - val_f1: 0.7948
48/48 [==============================] - 40s 843ms/step - loss: 0.5649 - accuracy:
0.8334 - val_loss: 0.8013 - val_accuracy: 0.8010
Epoch 5/10
12/12 [==============================] - 2s 191ms/steps: 0
 - val_precision: 0.8266 - val_recall: 0.8220 - val_f1: 0.8163
48/48 [==============================] - 40s 826ms/step - loss: 0.2801 - accuracy:
0.9134 - val_loss: 0.7790 - val_accuracy: 0.8220
Epoch 6/10
12/12 [==============================] - 3s 212ms/steps: 0
 - val_precision: 0.8469 - val_recall: 0.8168 - val_f1: 0.8185
48/48 [==============================] - 40s 841ms/step - loss: 0.2024 - accuracy:
0.9423 - val_loss: 0.7857 - val_accuracy: 0.8168
Epoch 7/10
12/12 [==============================] - 2s 203ms/steps: 0
 - val_precision: 0.8250 - val_recall: 0.7958 - val_f1: 0.7903
48/48 [==============================] - 40s 844ms/step - loss: 0.1511 - accuracy:
0.9534 - val_loss: 1.0017 - val_accuracy: 0.7958
Epoch 8/10
12/12 [==============================] - 3s 213ms/steps: 0
 - val_precision: 0.8376 - val_recall: 0.8194 - val_f1: 0.8105
48/48 [==============================] - 40s 834ms/step - loss: 0.1618 - accuracy:
0.9528 - val_loss: 0.9378 - val_accuracy: 0.8194
Epoch 9/10
12/12 [==============================] - 2s 204ms/steps: 0
 - val_precision: 0.8504 - val_recall: 0.8351 - val_f1: 0.8325
48/48 [==============================] - 40s 840ms/step - loss: 0.0893 - accuracy:
0.9744 - val_loss: 0.8874 - val_accuracy: 0.8351
Epoch 10/10
12/12 [==============================] - 3s 219ms/steps: 0
 - val_precision: 0.8615 - val_recall: 0.8560 - val_f1: 0.8535
48/48 [==============================] - 41s 853ms/step - loss: 0.0254 - accuracy:
0.9954 - val_loss: 0.7993 - val_accuracy: 0.8560
New best accuracy: 0.8560209274291992
Training model with 5 layers and 48 filters per layer
Epoch 1/10
12/12 [==============================] - 5s 383ms/steps: 3
 - val_precision: 0.3963 - val_recall: 0.3194 - val_f1: 0.2669
48/48 [==============================] - 80s 2s/step - loss: 3.0723 - accuracy: 0.16
79 - val_loss: 2.2451 - val_accuracy: 0.3194
Epoch 2/10
```

```
c:\Users\tehwh\anaconda3\envs\ml\lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
12/12 [==============================] - 5s 377ms/steps: 1
 - val_precision: 0.7674 - val_recall: 0.7094 - val_f1: 0.7076
48/48 [==============================] - 79s 2s/step - loss: 1.3930 - accuracy: 0.62
43 - val_loss: 1.0286 - val_accuracy: 0.7094
Epoch 3/10
12/12 [==============================] - 5s 386ms/steps: 0
 - val_precision: 0.8099 - val_recall: 0.7801 - val_f1: 0.7719
48/48 [==============================] - 79s 2s/step - loss: 0.6907 - accuracy: 0.79
48 - val_loss: 0.7894 - val_accuracy: 0.7801
Epoch 4/10
12/12 [==============================] - 4s 358ms/steps: 0
 - val_precision: 0.8143 - val_recall: 0.7853 - val_f1: 0.7778
48/48 [==============================] - 78s 2s/step - loss: 0.3794 - accuracy: 0.88
79 - val_loss: 0.7679 - val_accuracy: 0.7853
Epoch 5/10
12/12 [==============================] - 4s 370ms/steps: 0
 - val_precision: 0.8400 - val_recall: 0.8194 - val_f1: 0.8071
48/48 [==============================] - 79s 2s/step - loss: 0.3096 - accuracy: 0.90
10 - val_loss: 0.7001 - val_accuracy: 0.8194
Epoch 6/10
12/12 [==============================] - 5s 381ms/steps: 0
 - val_precision: 0.8653 - val_recall: 0.8455 - val_f1: 0.8439
48/48 [==============================] - 79s 2s/step - loss: 0.2301 - accuracy: 0.93
25 - val_loss: 0.6727 - val_accuracy: 0.8455
Epoch 7/10
12/12 [==============================] - 4s 373ms/steps: 0
 - val_precision: 0.8522 - val_recall: 0.8168 - val_f1: 0.8183
48/48 [==============================] - 80s 2s/step - loss: 0.2058 - accuracy: 0.93
97 - val_loss: 0.7038 - val_accuracy: 0.8168
Epoch 8/10
12/12 [==============================] - 5s 382ms/steps: 0
 - val_precision: 0.8701 - val_recall: 0.8534 - val_f1: 0.8498
48/48 [==============================] - 79s 2s/step - loss: 0.1157 - accuracy: 0.96
20 - val_loss: 0.6584 - val_accuracy: 0.8534
Epoch 9/10
12/12 [==============================] - 4s 370ms/steps: 0
 - val_precision: 0.8734 - val_recall: 0.8717 - val_f1: 0.8666
48/48 [==============================] - 79s 2s/step - loss: 0.0990 - accuracy: 0.96
72 - val_loss: 0.5493 - val_accuracy: 0.8717
Epoch 10/10
12/12 [==============================] - 4s 359ms/steps: 0
 - val_precision: 0.8684 - val_recall: 0.8613 - val_f1: 0.8589
48/48 [==============================] - 79s 2s/step - loss: 0.0827 - accuracy: 0.97
38 - val_loss: 0.6108 - val_accuracy: 0.8613
New best accuracy: 0.8717277646064758
Training model with 6 layers and 64 filters per layer
Epoch 1/10
12/12 [==============================] - 7s 587ms/steps: 3
 - val_precision: 0.1127 - val_recall: 0.1885 - val_f1: 0.0957
48/48 [==============================] - 121s 3s/step - loss: 3.3941 - accuracy: 0.0
584 - val_loss: 2.9090 - val_accuracy: 0.1885
Epoch 2/10
```

```
c:\Users\tehwh\anaconda3\envs\ml\lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
12/12 [==============================] - 7s 583ms/steps: 2
 - val_precision: 0.6603 - val_recall: 0.6414 - val_f1: 0.6096
48/48 [==============================] - 120s 2s/step - loss: 2.0762 - accuracy: 0.4157 - val_loss: 1.1600 - val_accuracy: 0.6414
Epoch 3/10
c:\Users\tehwh\anaconda3\envs\ml\lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
12/12 [==============================] - 7s 588ms/steps: 1
 - val_precision: 0.7823 - val_recall: 0.7173 - val_f1: 0.7121
48/48 [==============================] - 120s 3s/step - loss: 1.0179 - accuracy: 0.6
997 - val_loss: 0.9445 - val_accuracy: 0.7173
Epoch 4/10
12/12 [==============================] - 7s 583ms/steps: 0
 - val_precision: 0.8323 - val_recall: 0.7984 - val_f1: 0.7952
48/48 [==============================] - 120s 2s/step - loss: 0.5918 - accuracy: 0.8
249 - val_loss: 0.6781 - val_accuracy: 0.7984
Epoch 5/10
12/12 [==============================] - 7s 604ms/steps: 0
 - val_precision: 0.8665 - val_recall: 0.8482 - val_f1: 0.8416
48/48 [==============================] - 121s 3s/step - loss: 0.4445 - accuracy: 0.8
570 - val_loss: 0.5495 - val_accuracy: 0.8482
Epoch 6/10
12/12 [==============================] - 7s 585ms/steps: 0
 - val_precision: 0.8423 - val_recall: 0.8377 - val_f1: 0.8326
48/48 [==============================] - 121s 3s/step - loss: 0.2876 - accuracy: 0.9
095 - val_loss: 0.5994 - val_accuracy: 0.8377
Epoch 7/10
12/12 [==============================] - 7s 581ms/steps: 0
 - val_precision: 0.8584 - val_recall: 0.8455 - val_f1: 0.8429
48/48 [==============================] - 120s 3s/step - loss: 0.2136 - accuracy: 0.9
298 - val_loss: 0.6358 - val_accuracy: 0.8455
Epoch 8/10
12/12 [==============================] - 7s 613ms/steps: 0
 - val_precision: 0.8709 - val_recall: 0.8613 - val_f1: 0.8557
48/48 [==============================] - 122s 3s/step - loss: 0.1512 - accuracy: 0.9
469 - val_loss: 0.6126 - val_accuracy: 0.8613
Epoch 9/10
12/12 [==============================] - 7s 577ms/steps: 0
 - val_precision: 0.8684 - val_recall: 0.8298 - val_f1: 0.8247
48/48 [==============================] - 122s 3s/step - loss: 0.1753 - accuracy: 0.9
508 - val_loss: 0.8307 - val_accuracy: 0.8298
Epoch 10/10
12/12 [==============================] - 7s 592ms/steps: 0
 - val_precision: 0.8945 - val_recall: 0.8796 - val_f1: 0.8753
48/48 [==============================] - 120s 3s/step - loss: 0.1213 - accuracy: 0.9
659 - val_loss: 0.5635 - val_accuracy: 0.8796
New best accuracy: 0.8795811533927917
Training model with 7 layers and 80 filters per layer
Epoch 1/10
12/12 [==============================] - 10s 843ms/step: 3.
 - val_precision: 0.0004 - val_recall: 0.0183 - val_f1: 0.0008
48/48 [==============================] - 176s 4s/step - loss: 3.4689 - accuracy: 0.0
380 - val_loss: 3.4604 - val_accuracy: 0.0183
Epoch 2/10
c:\Users\tehwh\anaconda3\envs\ml\lib\site-packages\sklearn\metrics\_classification.p
y:1469: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this behavio
r.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
12/12 [==============================] - 10s 817ms/step: 2
 - val_precision: 0.4684 - val_recall: 0.4974 - val_f1: 0.4383
48/48 [==============================] - 173s 4s/step - loss: 2.7318 - accuracy: 0.2
328 - val_loss: 1.6920 - val_accuracy: 0.4974
Epoch 3/10
c:\Users\tehwh\anaconda3\envs\ml\lib\site-packages\sklearn\metrics\_classification.p
y:1469: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this behavio
r.
  _warn_prf(average, modifier, msg_start, len(result))
12/12 [==============================] - 10s 815ms/step: 1
 - val_precision: 0.7044 - val_recall: 0.6440 - val_f1: 0.6201
48/48 [==============================] - 173s 4s/step - loss: 1.3258 - accuracy: 0.5
902 - val_loss: 1.0969 - val_accuracy: 0.6440
Epoch 4/10
c:\Users\tehwh\anaconda3\envs\ml\lib\site-packages\sklearn\metrics\_classification.p
y:1469: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this behavio
r.
  _warn_prf(average, modifier, msg_start, len(result))
19/48 [==========>...................] - ETA: 1:34 - loss: 0.8495 - accuracy: 0.7237
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[143], line 2
      1 # run the custom training loop
----> 2 best_model, best_accuracy, train_accuracies, val_accuracies, val_precisions, val_recalls, val_f1s = train_evolving_cnn(X_train, y_train, X_val, y_val, target_accuracy=0.90)
      4 # final print
      5 print(f"TRAINING DONE! THE BEST VAL ACCURACY WAS: {best_accuracy}")

Cell In[142], line 23, in train_evolving_cnn(X_train, y_train, X_val, y_val, target_accuracy, max_epochs_per_stage, num_classes)
     19 model = build_cnn_model(num_layers, num_filters, input_shape, num_classes)
     21 metrics_callback = MetricsCallback(X_val, y_val, target_accuracy)
---> 23 history = model.fit(X_train, y_train, epochs=max_epochs_per_stage, validation_data=(X_val, y_val), verbose=1, callbacks=[metrics_callback])
     25 train_accuracies.extend(history.history['accuracy'])
     26 val_accuracies.extend(history.history['val_accuracy'])

File c:\Users\tehwh\anaconda3\envs\ml\lib\site-packages\keras\utils\traceback_utils.py:65, in filter_traceback.<locals>.error_handler(*args, **kwargs)
     63 filtered_tb = None
     64 try:
---> 65     return fn(*args, **kwargs)
     66 except Exception as e:
     67     filtered_tb = _process_traceback_frames(e.__traceback__)

File c:\Users\tehwh\anaconda3\envs\ml\lib\site-packages\keras\engine\training.py:1564, in Model.fit(self, x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_data, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, validation_steps, validation_batch_size, validation_freq, max_queue_size, workers, use_multiprocessing)
   1556 with tf.profiler.experimental.Trace(
   1557     "train",
   1558     epoch_num=epoch,
   (...)
   1561     _r=1,
   1562 ):
   1563     callbacks.on_train_batch_begin(step)
-> 1564     tmp_logs = self.train_function(iterator)
   1565     if data_handler.should_sync:
   1566         context.async_wait()

File c:\Users\tehwh\anaconda3\envs\ml\lib\site-packages\tensorflow\python\util\traceback_utils.py:150, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    148 filtered_tb = None
    149 try:
--> 150   return fn(*args, **kwargs)
    151 except Exception as e:
    152   filtered_tb = _process_traceback_frames(e.__traceback__)

File c:\Users\tehwh\anaconda3\envs\ml\lib\site-packages\tensorflow\python\eager\def_function.py:915, in Function.__call__(self, *args, **kwds)
    912 compiler = "xla" if self._jit_compile else "nonXla"
    914 with OptionalXlaContext(self._jit_compile):
--> 915   result = self._call(*args, **kwds)
```

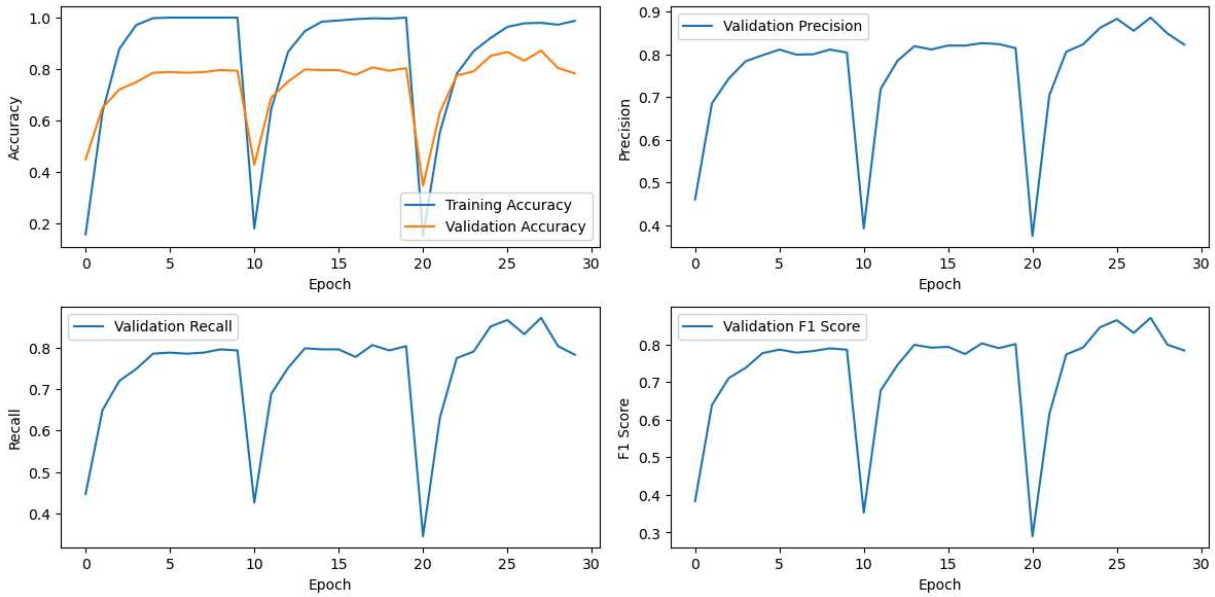
```
    917 new_tracing_count = self.experimental_get_tracing_count()
    918 without_tracing = (tracing_count == new_tracing_count)

File c:\Users\tehwh\anaconda3\envs\ml\lib\site-packages\tensorflow\python\eager\def_
function.py:947, in Function._call(self, *args, **kwds)
    944   self._lock.release()
    945   # In this case we have created variables on the first call, so we run the
    946   # defunned version which is guaranteed to never create variables.
--> 947   return self._stateless_fn(*args, **kwds)  # pylint: disable=not-callable
    948 elif self._stateful_fn is not None:
    949   # Release the lock early so that multiple threads can perform the call
    950   # in parallel.
    951   self._lock.release()

File c:\Users\tehwh\anaconda3\envs\ml\lib\site-packages\tensorflow\python\eager\func
tion.py:2496, in Function.__call__(self, *args, **kwargs)
   2493 with self._lock:
   2494   (graph_function,
   2495    filtered_flat_args) = self._maybe_define_function(args, kwargs)
-> 2496 return graph_function._call_flat(
   2497     filtered_flat_args, captured_inputs=graph_function.captured_inputs)

File c:\Users\tehwh\anaconda3\envs\ml\lib\site-packages\tensorflow\python\eager\func
tion.py:1862, in ConcreteFunction._call_flat(self, args, captured_inputs, cancellati
on_manager)
   1858 possible_gradient_type = gradients_util.PossibleTapeGradientTypes(args)
   1859 if (possible_gradient_type == gradients_util.POSSIBLE_GRADIENT_TYPES_NONE
   1860     and executing_eagerly):
   1861   # No tape is watching; skip to running the function.
-> 1862   return self._build_call_outputs(self._inference_function.call(
   1863       ctx, args, cancellation_manager=cancellation_manager))
   1864 forward_backward = self._select_forward_and_backward_functions(
   1865     args,
   1866     possible_gradient_type,
   1867     executing_eagerly)
   1868 forward_function, args_with_tangents = forward_backward.forward()

File c:\Users\tehwh\anaconda3\envs\ml\lib\site-packages\tensorflow\python\eager\func
tion.py:499, in _EagerDefinedFunction.call(self, ctx, args, cancellation_manager)
    497 with _InterpolateFunctionError(self):
    498   if cancellation_manager is None:
--> 499     outputs = execute.execute(
    500         str(self.signature.name),
    501         num_outputs=self._num_outputs,
    502         inputs=args,
    503         attrs=attrs,
    504         ctx=ctx)
    505   else:
    506     outputs = execute.execute_with_cancellation(
    507         str(self.signature.name),
    508         num_outputs=self._num_outputs,
   (...)
    511         ctx=ctx,
    512         cancellation_manager=cancellation_manager)

File c:\Users\tehwh\anaconda3\envs\ml\lib\site-packages\tensorflow\python\eager\exec
```

```
ute.py:54, in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     52 try:
     53   ctx.ensure_initialized()
---> 54   tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     55                                       inputs, attrs, num_outputs)
     56 except core._NotOkStatusException as e:
     57   if name is not None:

KeyboardInterrupt:
```

I did end the training early, as there was a bug in the code, not ending the cell after the accuracy was met.

In [ ]:

```python
# custom plotting to plot the accuracy, precision, recall and f1 score

epochs = range(len(train_accuracies))

plt.figure(figsize=(12, 6))
plt.subplot(2, 2, 1)
plt.plot(epochs, train_accuracies, label='Training Accuracy')
plt.plot(epochs, val_accuracies, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(2, 2, 2)
plt.plot(epochs, val_precisions, label='Validation Precision')
plt.xlabel('Epoch')
plt.ylabel('Precision')
plt.legend()

plt.subplot(2, 2, 3)
plt.plot(epochs, val_recalls, label='Validation Recall')
plt.xlabel('Epoch')
plt.ylabel('Recall')
plt.legend()

plt.subplot(2, 2, 4)
plt.plot(epochs, val_f1s, label='Validation F1 Score')
plt.xlabel('Epoch')
plt.ylabel('F1 Score')
plt.legend()

plt.tight_layout()
plt.show()
```



In [ ]:
```
# getting the best model that we have trained

best_model.summary()
```

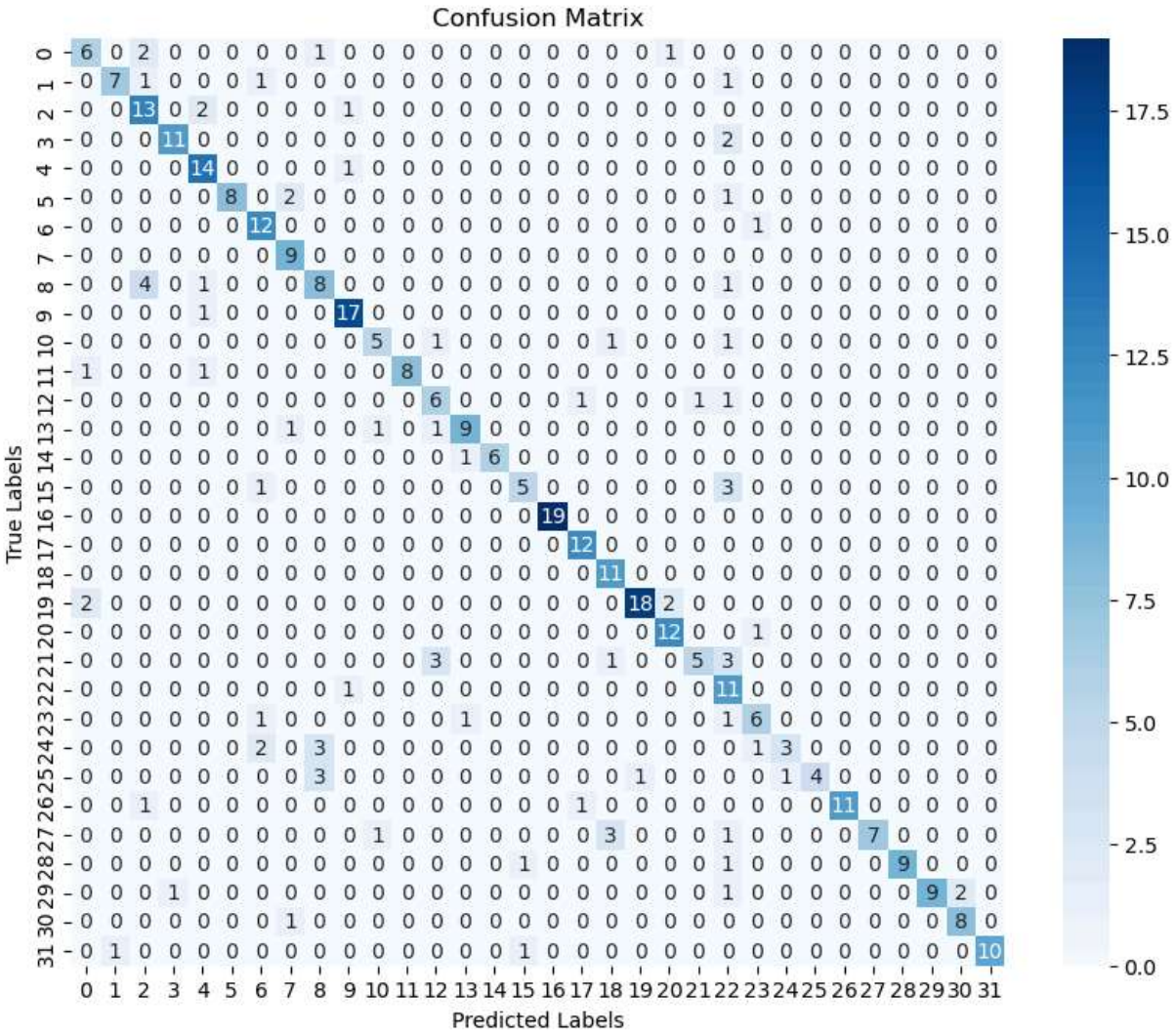
```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_44 (Conv2D)          (None, 398, 531, 64)      640

 max_pooling2d_44 (MaxPoolin  (None, 199, 265, 64)     0
 g2D)

 conv2d_45 (Conv2D)          (None, 197, 263, 64)      36928

 max_pooling2d_45 (MaxPoolin  (None, 98, 131, 64)      0
 g2D)

 conv2d_46 (Conv2D)          (None, 96, 129, 64)       36928

 max_pooling2d_46 (MaxPoolin  (None, 48, 64, 64)       0
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_44 (Conv2D)          (None, 398, 531, 64)      640

 max_pooling2d_44 (MaxPoolin  (None, 199, 265, 64)     0
 g2D)

 conv2d_45 (Conv2D)          (None, 197, 263, 64)      36928

 max_pooling2d_45 (MaxPoolin  (None, 98, 131, 64)      0
 g2D)

 conv2d_46 (Conv2D)          (None, 96, 129, 64)       36928

 max_pooling2d_46 (MaxPoolin  (None, 48, 64, 64)       0
 g2D)

 conv2d_47 (Conv2D)          (None, 46, 62, 64)        36928

 max_pooling2d_47 (MaxPoolin  (None, 23, 31, 64)       0
 g2D)

 flatten_15 (Flatten)        (None, 45632)             0

 dense_30 (Dense)            (None, 64)                2920512

 dense_31 (Dense)            (None, 32)                2080

=================================================================
Total params: 3,034,016
Trainable params: 3,034,016
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
# Make predictions on the validation set
val_predictions = best_model.predict(X_val)

# Convert predictions and true labels from one-hot encoded format to original class
```

```
val_predictions_labels = np.argmax(val_predictions, axis=1)
val_true_labels = np.argmax(y_val, axis=1)

# Generate the confusion matrix
conf_matrix = confusion_matrix(val_true_labels, val_predictions_labels)

# Plotting the confusion matrix using seaborn heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=range(32),
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

```
12/12 [==============================] - 6s 515ms/step
```



In [ ]:
```
# Now we are going to look at how it tested for each of the classes using the class

from sklearn.metrics import classification_report

print(classification_report(val_true_labels, val_predictions_labels))
```

	precision	recall	f1-score	support
0	0.67	0.60	0.63	10
1	0.88	0.70	0.78	10
2	0.62	0.81	0.70	16
3	0.92	0.85	0.88	13
4	0.74	0.93	0.82	15
5	1.00	0.73	0.84	11
6	0.71	0.92	0.80	13
7	0.69	1.00	0.82	9
8	0.53	0.57	0.55	14
9	0.85	0.94	0.89	18
10	0.71	0.62	0.67	8
11	1.00	0.80	0.89	10
12	0.55	0.67	0.60	9
13	0.82	0.75	0.78	12
14	1.00	0.86	0.92	7
15	0.71	0.56	0.63	9
16	1.00	1.00	1.00	19
17	0.86	1.00	0.92	12
18	0.69	1.00	0.81	11
19	0.95	0.82	0.88	22
20	0.80	0.92	0.86	13
21	0.83	0.42	0.56	12
22	0.39	0.92	0.55	12
23	0.67	0.67	0.67	9
24	0.75	0.33	0.46	9
25	1.00	0.44	0.62	9
26	1.00	0.85	0.92	13
27	1.00	0.58	0.74	12
28	1.00	0.82	0.90	11
29	1.00	0.69	0.82	13
30	0.80	0.89	0.84	9
31	1.00	0.83	0.91	12
accuracy			0.78	382
macro avg	0.82	0.77	0.77	382
weighted avg	0.82	0.78	0.78	382

Results

Strong classes

- 1, 3, 4, 9, 11, 14, 16, 17, 19, 20, 26, and 28, have high precision, recall, and F1-scores, meaning that our model was able to predict these classes with high accuracy. These classes are likely to be well-separated from the other classes in the feature space, making them easier to classify.

Weak Classes

- 8, 21, 22, and 24, have lower precision, recall, and F1-scores which means our model was not really that good at predicting these classes. Which my assumption is that these

classes are very similar to eachother, thus making it harder for the model to predict these classes. What I dont think helped, was that we downsized the pixles in the pictures, which if we used higher resolution images then the model could have picked up on some of the finer details.

Conclusion

Considering the fact that we had a very small dataset, and that we were able to get a 0.78 overall accuracy, I would say that this model is pretty good. I would say that if we had a larger dataset, and we were able to use higher resolution images, then we would have been able to get a higher accuracy. But considering that we had some classes that were predicted very high, such as class 16, which had a 1.0 precision, recall, and F1-score, I find it very impressive. But, looking at the confusion matrix, there are some classes that do need more help, and I think that we could add more weights to those classes to help the model predict those classes better.