## Upload the dataset for creation of CNN, and check charasteristics

# to donwload the file you need to have you kaggle api token istalled the path should be ~/.kaggle/kaggle.json  
# The kaggle.json file should look like this:  
# {  
# "username": "your\_username",  
# "key": "your\_key"  
# }  
from kaggle.api.kaggle\_api\_extended import KaggleApi  
import os  
  
# Initialize the Kaggle API  
api = KaggleApi()  
api.authenticate()  
  
# Specify the competition name (plant-seedlings-classification)  
competition\_name = 'plant-seedlings-classification'  
  
# Specify the download path  
download\_path = './plant\_seedlings'  
  
# Download the competition dataset  
api.competition\_download\_files(competition\_name, path=download\_path)  
  
print(f"Dataset downloaded to {download\_path}")

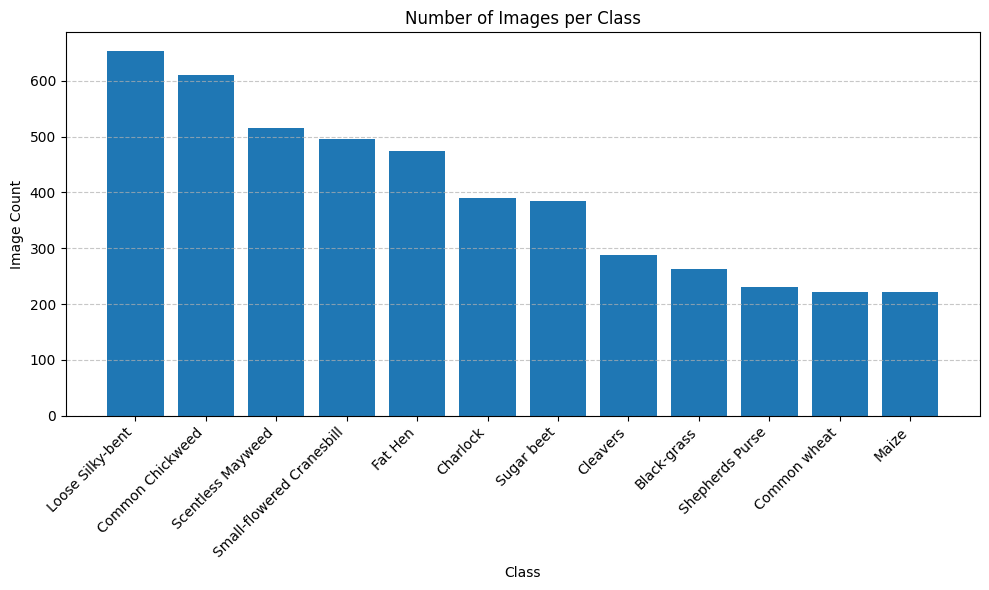
---------------------------------------------------------------------------  
  
OSError Traceback (most recent call last)  
  
Cell In[9], line 12  
 10 # Initialize the Kaggle API  
 11 api = KaggleApi()  
---> 12 api.authenticate()  
 14 # Specify the competition name (plant-seedlings-classification)  
 15 competition\_name = 'plant-seedlings-classification'  
  
  
File ~/Documents/APROF/CNN/venv/lib/python3.8/site-packages/kaggle/api/kaggle\_api\_extended.py:434, in KaggleApi.authenticate(self)  
 432 return  
 433 else:  
--> 434 raise IOError('Could not find {}. Make sure it\'s located in'  
 435 ' {}. Or use the environment method. See setup'  
 436 ' instructions at'  
 437 ' https://github.com/Kaggle/kaggle-api/'.format(  
 438 self.config\_file, self.config\_dir))  
 440 # Step 3: load into configuration!  
 441 self.\_load\_config(config\_data)  
  
  
OSError: Could not find kaggle.json. Make sure it's located in /Users/filiporlikowski/kaggle\_api. Or use the environment method. See setup instructions at https://github.com/Kaggle/kaggle-api/

import zipfile  
  
# Unzip the dataset  
with zipfile.ZipFile(f'{download\_path}/{competition\_name}.zip', 'r') as zip\_ref:  
 zip\_ref.extractall(download\_path)  
  
print("Dataset extracted.")

---------------------------------------------------------------------------  
  
NameError Traceback (most recent call last)  
  
Cell In[8], line 4  
 1 import zipfile  
 3 # Unzip the dataset  
----> 4 with zipfile.ZipFile(f'{download\_path}/{competition\_name}.zip', 'r') as zip\_ref:  
 5 zip\_ref.extractall(download\_path)  
 7 print("Dataset extracted.")  
  
  
NameError: name 'download\_path' is not defined

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import matplotlib.image as mpimg  
import os  
import random  
from PIL import Image

train\_dir = "./plant\_seedlings/train"  
test\_dir = "./plant\_seedlings/test"  
  
# Get list of subdirectories (classes) and count images  
class\_counts = {}  
for cls\_name in os.listdir(train\_dir):  
 cls\_path = os.path.join(train\_dir, cls\_name)  
 if os.path.isdir(cls\_path):  
 image\_files = [f for f in os.listdir(cls\_path) if os.path.isfile(os.path.join(cls\_path, f))]  
 class\_counts[cls\_name] = len(image\_files)  
  
# Create DataFrame and sort by count  
df\_counts = pd.DataFrame(list(class\_counts.items()), columns=['Class', 'Image Count'])  
df\_counts.sort\_values(by='Image Count', ascending=False, inplace=True)  
  
# Plotting  
fig, ax = plt.subplots(figsize=(10, 6))  
ax.bar(df\_counts['Class'], df\_counts['Image Count'])  
ax.set\_title("Number of Images per Class")  
ax.set\_xlabel("Class")  
ax.set\_ylabel("Image Count")  
plt.xticks(rotation=45, ha='right', fontsize=10)  
ax.grid(axis='y', linestyle='--', alpha=0.7)  
plt.tight\_layout()  
plt.show()



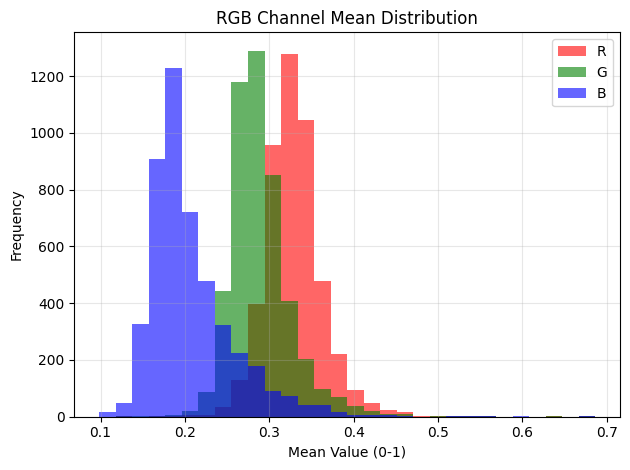
png

import os  
import random  
from PIL import Image  
import pandas as pd  
import plotly.subplots as sp  
import plotly.graph\_objects as go  
import nbformat as nb  
import IPython  
  
# Create DataFrame of image counts per class  
df\_counts = pd.DataFrame(list(class\_counts.items()), columns=['Class', 'Image Count'])  
df\_counts.sort\_values(by='Image Count', ascending=False, inplace=True)  
  
# Get class names  
class\_names = df\_counts['Class'].tolist()  
  
# Grid size (customize as needed)  
n\_rows, n\_cols = 3, 4  
max\_images = n\_rows \* n\_cols  
  
# Initialize Plotly subplot figure  
fig = sp.make\_subplots(rows=n\_rows, cols=n\_cols,  
 subplot\_titles=class\_names[:max\_images],  
 vertical\_spacing=0.05, horizontal\_spacing=0.03)  
  
# Load and add images  
img\_idx = 0  
for row in range(1, n\_rows + 1):  
 for col in range(1, n\_cols + 1):  
 if img\_idx >= len(class\_names):  
 break  
 cls = class\_names[img\_idx]  
 cls\_path = os.path.join(train\_dir, cls)  
 image\_files = [f for f in os.listdir(cls\_path) if os.path.isfile(os.path.join(cls\_path, f))]  
 if image\_files:  
 img\_file = random.choice(image\_files)  
 img\_path = os.path.join(cls\_path, img\_file)  
 try:  
 with Image.open(img\_path) as img:  
 fig.add\_trace(  
 go.Image(z=img),  
 row=row, col=col  
 )  
 except Exception as e:  
 print(f"Error loading image: {img\_path} ({e})")  
 img\_idx += 1  
  
# Update layout for cleaner visuals  
fig.update\_layout(  
 height=800, width=1200,  
 title\_text="Example Images from Each Class (Interactive)",  
 showlegend=False,  
 title\_x=0.5,  
 margin=dict(l=10, r=10, t=60, b=10)  
)  
  
# Remove axes for all subplots  
for i in range(1, n\_rows \* n\_cols + 1):  
 fig['layout'][f'xaxis{i}']['visible'] = False  
 fig['layout'][f'yaxis{i}']['visible'] = False  
  
fig.show()

We will also check balance of rgb colors, by plotting: RGB Channel Mean Distribution

import numpy as np  
import os  
from PIL import Image  
import pandas as pd  
import matplotlib.pyplot as plt  
  
# Function to compute RGB means  
def compute\_rgb\_means(image\_path):  
 try:  
 img = Image.open(image\_path).convert('RGB')  
 img\_np = np.array(img) / 255.0 # Normalize pixel values to [0, 1]  
 r\_mean = np.mean(img\_np[:, :, 0])  
 g\_mean = np.mean(img\_np[:, :, 1])  
 b\_mean = np.mean(img\_np[:, :, 2])  
 return r\_mean, g\_mean, b\_mean  
 except:  
 return None  
  
# Collect the RGB means for all images  
rgb\_means = []  
  
for cls in os.listdir(train\_dir):  
 cls\_dir = os.path.join(train\_dir, cls)  
 for img\_name in os.listdir(cls\_dir):  
 img\_path = os.path.join(cls\_dir, img\_name)  
 means = compute\_rgb\_means(img\_path)  
 if means:  
 rgb\_means.append(means)  
  
# Convert to DataFrame  
df\_rgb = pd.DataFrame(rgb\_means, columns=['R', 'G', 'B'])  
  
# Check the mean and std dev of each color channel  
mean\_rgb = df\_rgb.mean()  
std\_rgb = df\_rgb.std()  
  
print(f"Mean values per channel:\n{mean\_rgb}")  
print(f"Standard deviation per channel:\n{std\_rgb}")  
  
# Check correlations between channels (to see if any channel is too similar or too different)  
correlation\_matrix = df\_rgb.corr()  
print(f"Correlation between RGB channels:\n{correlation\_matrix}")  
  
# Plot histograms of R, G, B with matching colors  
plt.figure(figsize=(10, 6))  
df\_rgb.plot(kind='hist', bins=30, alpha=0.6, color=['red', 'green', 'blue'])  
plt.title('RGB Channel Mean Distribution')  
plt.xlabel("Mean Value (0-1)")  
plt.grid(alpha=0.3)  
plt.tight\_layout()  
plt.show()

Mean values per channel:  
R 0.328815  
G 0.289346  
B 0.207304  
dtype: float64  
Standard deviation per channel:  
R 0.034725  
G 0.037108  
B 0.050703  
dtype: float64  
Correlation between RGB channels:  
 R G B  
R 1.000000 0.882091 0.764380  
G 0.882091 1.000000 0.858662  
B 0.764380 0.858662 1.000000  
  
  
  
<Figure size 1000x600 with 0 Axes>



png

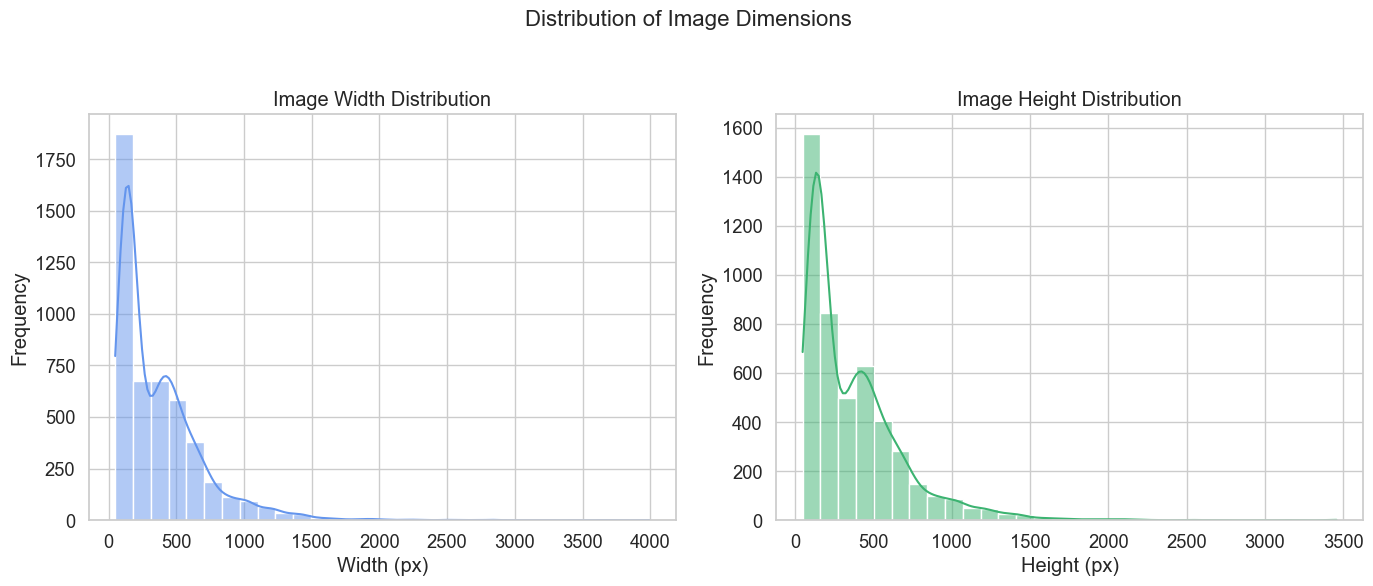
From the graph we see that we should not encounter grayscale pictures, rgb colors are distinclty distributed with red channel resulting in a mean value of 0.32 we can definately call it a dominating color, then we have green 0.29, pictures presumably present plants, and blue 0.21 which is a minority among the rgb set of colors. But to be shure we will check for greenscale pictures, they should have similar values of rgb colors in them. We will also identify grayscale images in ‘L’ mode. But mostly we will focus on checking which images can not be opened as RGB and then converted to RGB.

import os  
from PIL import Image  
  
corrupted\_images = []  
  
for cls in os.listdir(train\_dir):  
 cls\_dir = os.path.join(train\_dir, cls)  
 for img\_name in os.listdir(cls\_dir):  
 img\_path = os.path.join(cls\_dir, img\_name)  
 try:  
 img = Image.open(img\_path).convert('RGB') # Try to open and convert to RGB  
 except Exception as e:  
 # If there's an error opening the image, add it to the corrupted list  
 corrupted\_images.append(img\_path)  
 print(f"Error opening {img\_path}: {e}")  
  
# Output the list of corrupted images  
print(f"\nFound {len(corrupted\_images)} corrupted or unreadable images.")  
for img in corrupted\_images:  
 print(img)

Found 0 corrupted or unreadable images.

We will also check the sizes of the images

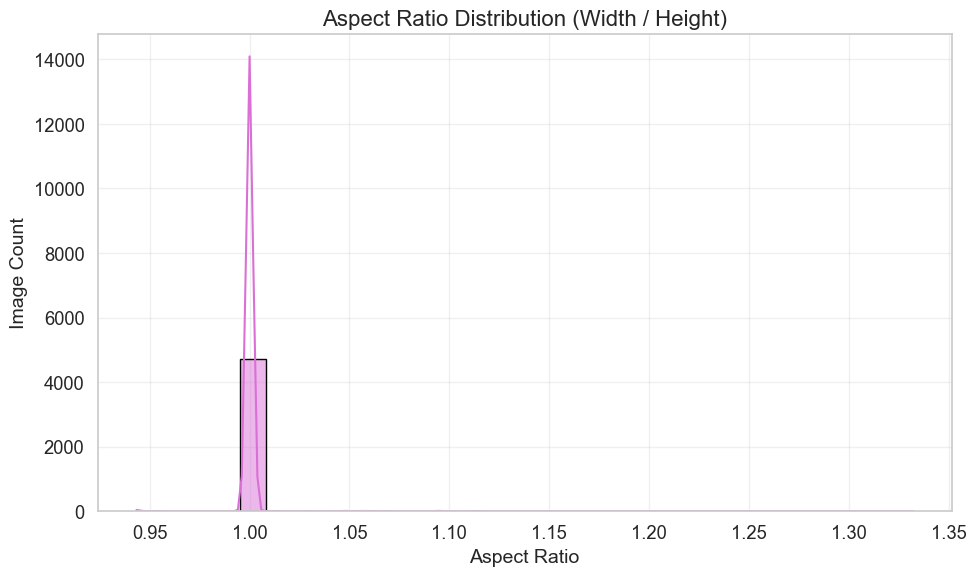
import os  
from PIL import Image  
import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
# Set Seaborn theme for cleaner aesthetics  
sns.set(style="whitegrid", palette="pastel", font\_scale=1.2)  
  
def collect\_image\_shapes(directory):  
 shapes = []  
 for class\_name in os.listdir(directory):  
 class\_path = os.path.join(directory, class\_name)  
 if not os.path.isdir(class\_path):  
 continue  
 for image\_file in os.listdir(class\_path):  
 image\_path = os.path.join(class\_path, image\_file)  
 try:  
 with Image.open(image\_path) as img:  
 shapes.append(img.size) # (width, height)  
 except Exception as e:  
 print(f"Could not open image: {image\_path} ({e})")  
 return shapes  
  
# Collect image dimensions and create DataFrame  
image\_shapes = collect\_image\_shapes(train\_dir)  
df\_shapes = pd.DataFrame(image\_shapes, columns=['width', 'height'])  
  
# Create figure with Seaborn plots  
fig, axs = plt.subplots(1, 2, figsize=(14, 6))  
  
sns.histplot(data=df\_shapes, x='width', bins=30, ax=axs[0], kde=True, color='cornflowerblue')  
axs[0].set\_title('Image Width Distribution')  
axs[0].set\_xlabel('Width (px)')  
axs[0].set\_ylabel('Frequency')  
  
sns.histplot(data=df\_shapes, x='height', bins=30, ax=axs[1], kde=True, color='mediumseagreen')  
axs[1].set\_title('Image Height Distribution')  
axs[1].set\_xlabel('Height (px)')  
axs[1].set\_ylabel('Frequency')  
  
plt.suptitle('Distribution of Image Dimensions', fontsize=16)  
plt.tight\_layout(rect=[0, 0, 1, 0.95])  
plt.show()



png

Histograms show that most images range from 100 to 600 pixels in width and height, with a few large outliers exceeding 3000 pixels. This size variability suggests that resizing or normalization is needed before using the data in a CNN. We will also try to visualize the aspect ratio of images so that we know their shape.

import seaborn as sns  
import matplotlib.pyplot as plt  
  
# Calculate Aspect Ratio  
df\_shapes['aspect\_ratio'] = df\_shapes['width'] / df\_shapes['height']  
  
# Set Seaborn style for improved visuals  
sns.set(style="whitegrid", palette="muted", font\_scale=1.2)  
  
# Create a figure with custom size  
plt.figure(figsize=(10, 6))  
  
# Plot histogram of aspect ratios using Seaborn's histplot  
sns.histplot(df\_shapes['aspect\_ratio'], bins=30, color='orchid', kde=True, edgecolor='black')  
  
# Customize plot with titles and labels  
plt.title("Aspect Ratio Distribution (Width / Height)", fontsize=16)  
plt.xlabel("Aspect Ratio", fontsize=14)  
plt.ylabel("Image Count", fontsize=14)  
  
# Show grid with a soft transparency  
plt.grid(alpha=0.3)  
  
# Tight layout for better spacing  
plt.tight\_layout()  
  
# Show the plot  
plt.show()



png

A significant majority of the images have an aspect ratio close to 1, meaning that most of the images are square or nearly square. So knowing that the dataset primarily consists of square images, it is okay to standardize all images to a square resolution when preprocessing. Mostly because there are slim chances of changing the content of the picture.

## Preprocessing of the images

We will convert all images to RGB and resize them

from tensorflow.keras.preprocessing.image import img\_to\_array  
import os  
  
# Parameters  
IMG\_SIZE = 96  
DATASET\_DIR = "./plant\_seedlings/train"  
  
# Containers  
images = []  
labels = []  
class\_names = sorted([d for d in os.listdir(DATASET\_DIR) if os.path.isdir(os.path.join(DATASET\_DIR, d))])  
  
# Iterate through class folders and load images  
for class\_name in class\_names:  
 class\_dir = os.path.join(DATASET\_DIR, class\_name)  
 for filename in os.listdir(class\_dir):  
 img\_path = os.path.join(class\_dir, filename)  
 try:  
 # Open image, convert to RGB and resize  
 img = Image.open(img\_path).convert("RGB")  
 img = img.resize((IMG\_SIZE, IMG\_SIZE))  
 img\_array = img\_to\_array(img) / 255.0 # Normalize to 0–1  
 images.append(img\_array)  
 labels.append(class\_name)  
 except Exception as e:  
 print(f"Skipping file {img\_path}: {e}")  
  
# Convert to numpy arrays  
X = np.array(images)  
y = np.array(labels)  
  
print("Images loaded and resized")  
print(f"Image shape: {X.shape}")  
print(f"Unique classes: {len(np.unique(y))}")

Images loaded and resized  
Image shape: (4750, 96, 96, 3)  
Unique classes: 12

We have also resized pictures to 224x224 becasue this is the samllest size and that most pictures have. We do it so that we have consistent input, and memory efficiency enhanced. Later we will convert string labels meaning classes of plants into integer indices for the CNN and then apply one-hot encoding to represent these labels as binary vectors. Then we will split the dataset into training, validation, and test sets using train\_test\_split, with stratified sampling to ensure class distribution is preserved in each split.

from sklearn.model\_selection import train\_test\_split  
from tensorflow.keras.utils import to\_categorical  
import numpy as np  
  
# Function to create label mappings and encode labels  
def encode\_labels(y):  
 label\_to\_index = {label: idx for idx, label in enumerate(np.unique(y))}  
 y\_encoded = np.array([label\_to\_index[label] for label in y])  
 return label\_to\_index, y\_encoded  
  
# One-hot encoding of labels  
def one\_hot\_encode(y\_encoded, num\_classes):  
 return to\_categorical(y\_encoded, num\_classes=num\_classes)  
  
# Split the data into train, validation, and test sets  
def split\_data(X, y\_categorical):  
 X\_train, X\_test1, y\_train, y\_test1 = train\_test\_split(  
 X, y\_categorical, test\_size=0.30, stratify=y\_categorical, random\_state=42  
 )  
 X\_val, X\_test, y\_val, y\_test = train\_test\_split(  
 X\_test1, y\_test1, test\_size=0.50, stratify=y\_test1, random\_state=42  
 )  
 return X\_train, X\_val, X\_test, y\_train, y\_val, y\_test  
  
# Main execution  
label\_to\_index, y\_encoded = encode\_labels(y)  
y\_categorical = one\_hot\_encode(y\_encoded, num\_classes=len(label\_to\_index))  
  
X\_train, X\_val, X\_test, y\_train, y\_val, y\_test = split\_data(X, y\_categorical)  
  
# Print shapes  
print("Labels encoded and data split finished.")  
print(f"X\_train shape: {X\_train.shape}")  
print(f"y\_train shape: {y\_train.shape}")  
print(f"X\_val shape: {X\_val.shape}")  
print(f"y\_val shape: {y\_val.shape}")  
print(f"X\_test shape: {X\_test.shape}")  
print(f"y\_test shape: {y\_test.shape}")

Labels encoded and data split finished.  
X\_train shape: (3325, 96, 96, 3)  
y\_train shape: (3325, 12)  
X\_val shape: (712, 96, 96, 3)  
y\_val shape: (712, 12)  
X\_test shape: (713, 96, 96, 3)  
y\_test shape: (713, 12)

Later we will processes images by applying green color segmentation, Gaussian blur, and normalization to isolate relevant green regions, making them suitable for CNN training. Since we want to clasify plants which are green. These steps will ensure that the images are cleaned, standardized, and that they focus on the desired features (green areas), which can enhance model performance for tasks like object recognition in plant-related datasets. What we want is sharp edges and close 0-1 color palette with distinction for green.

import numpy as np  
import cv2  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
# Set seaborn style for better aesthetics  
sns.set(style="whitegrid")  
  
# --- Settings for Green Segmentation ---  
lower\_green = (25, 40, 50)  
upper\_green = (75, 255, 255)  
kernel = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (11, 11))  
  
# --- Sharpening function using unsharp masking ---  
def sharpen\_image(img):  
 # Apply a Gaussian blur to create a blurred version of the image  
 blurred = cv2.GaussianBlur(img, (5, 5), 0)  
 # Sharpening by subtracting the blurred image from the original  
 sharpened = cv2.addWeighted(img, 1.5, blurred, -0.5, 0)  
 return sharpened  
  
# --- Preprocessing function ---  
def preprocess\_images(X):  
 preprocessed = []  
 for img in X:  
 # 1. Convert to uint8 (0–255)  
 img\_uint8 = (img \* 255).astype(np.uint8)  
  
 # 2. Optional resize (images are 224x224 in this case)  
 resize\_img = img\_uint8  
  
 # 3. Apply Gaussian blur for smoothing  
 Gblur\_img = cv2.GaussianBlur(resize\_img, (3, 3), 0)  
  
 # 4. Convert to HSV for easier color segmentation  
 hsv\_img = cv2.cvtColor(Gblur\_img, cv2.COLOR\_RGB2HSV) # RGB -> HSV  
  
 # 5. Segment the green regions  
 mask = cv2.inRange(hsv\_img, lower\_green, upper\_green)  
  
 # 6. Morphological cleaning (removing noise)  
 mask = cv2.morphologyEx(mask, cv2.MORPH\_CLOSE, kernel)  
  
 # 7. Boolean mask to select green regions  
 bMask = mask > 0  
  
 # 8. Apply the mask to the image (keep only non-green parts)  
 clearImg = np.zeros\_like(resize\_img, np.uint8)  
 clearImg[bMask] = resize\_img[bMask]  
  
 # 9. Normalize the image (0–1)  
 clearImg\_float = clearImg.astype(np.float32) / 255.0  
  
 # 10. Sharpen the edges of the image  
 sharpened\_img = sharpen\_image(clearImg\_float)  
  
 preprocessed.append(sharpened\_img)  
  
 return np.asarray(preprocessed)  
  
# --- Preprocessing the data (example for X\_train) ---  
X\_train\_cleaned = preprocess\_images(X\_train)  
X\_val\_cleaned = preprocess\_images(X\_val)  
X\_test\_cleaned = preprocess\_images(X\_test)  
  
# --- Display a random example of original and processed image ---  
def display\_random\_image(X\_original, X\_processed):  
 # Random index for image selection  
 idx = np.random.randint(0, len(X\_original))  
  
 # Fetch original and processed images  
 original\_img = X\_original[idx]  
 processed\_img = X\_processed[idx]  
  
 # Create figure and axes  
 fig, axes = plt.subplots(1, 2, figsize=(12, 6))  
  
 # Display the original image  
 axes[0].imshow(original\_img)  
 axes[0].set\_title("Original Image", fontsize=14)  
 axes[0].axis('off') # Hide axes  
  
 # Display the processed image  
 axes[1].imshow(processed\_img)  
 axes[1].set\_title("Segmented + Cleaned + Sharpened Image", fontsize=14)  
 axes[1].axis('off') # Hide axes  
  
 # Adjust layout for better presentation  
 plt.tight\_layout()  
 plt.show()  
  
# Call the function to display a random image  
display\_random\_image(X\_train, X\_train\_cleaned)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



png

print(f"Cleaned training shape: {X\_train\_cleaned.shape}")  
print(f"Cleaned validation shape: {X\_val\_cleaned.shape}")  
print(f"Cleaned test shape: {X\_test\_cleaned.shape}")

Cleaned training shape: (3325, 96, 96, 3)  
Cleaned validation shape: (712, 96, 96, 3)  
Cleaned test shape: (713, 96, 96, 3)

# CNN model build

Looking for the right values of strides pools etc.

from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Conv2D, MaxPooling2D, AveragePooling2D  
from tensorflow.keras.layers import Flatten, Dense, Dropout, BatchNormalization, Input  
  
def build\_variant\_model(  
 input\_shape,  
 num\_classes=12,  
 filters=[32, 64, 128, 128],  
 strides=2,  
 padding='same',  
 pool\_type='max' # or 'avg'  
):  
 model = Sequential()  
 model.add(Input(shape=input\_shape))  
  
 for f in filters:  
 model.add(Conv2D(f, (3, 3), activation='relu', padding=padding))  
 model.add(BatchNormalization())  
 if pool\_type == 'max':  
 model.add(MaxPooling2D(pool\_size=(2, 2), strides=strides))  
 else:  
 model.add(AveragePooling2D(pool\_size=(2, 2), strides=strides))  
  
 model.add(Flatten())  
  
 model.add(BatchNormalization())  
 model.add(Dense(512, activation='relu'))  
 model.add(Dropout(0.3))  
  
 model.add(BatchNormalization())  
 model.add(Dense(256, activation='relu'))  
 model.add(Dropout(0.3))  
  
 model.add(BatchNormalization())  
 model.add(Dense(128, activation='relu'))  
 model.add(Dropout(0.2))  
  
 model.add(Dense(num\_classes, activation='softmax'))  
  
 return model

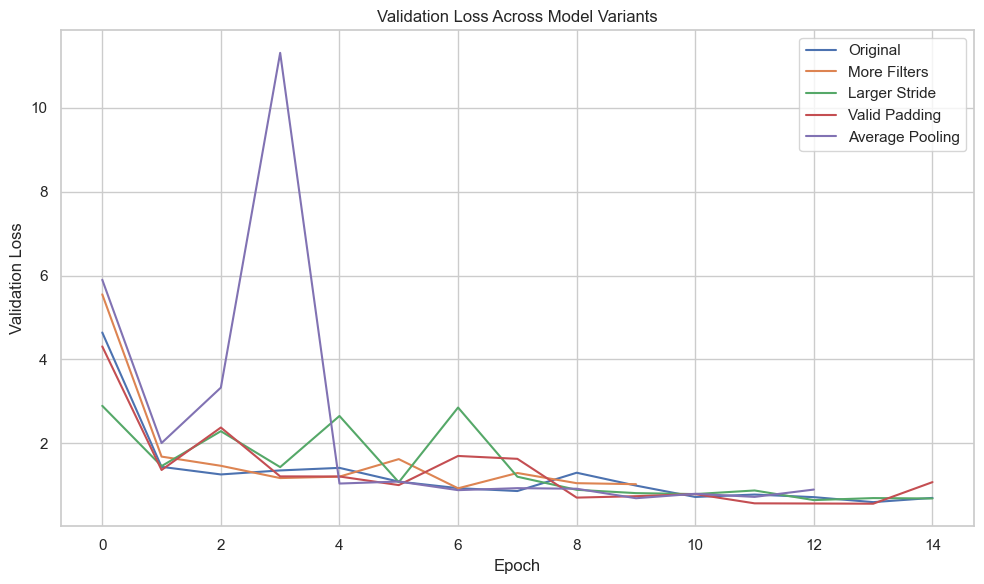
Training and comparing different variations

import os  
import time  
from tensorflow.keras.optimizers import Adam  
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau  
  
early\_stop = EarlyStopping(monitor='val\_loss', patience=3, restore\_best\_weights=True, verbose=1)  
reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.5, patience=2, min\_lr=1e-6, verbose=1)  
  
variants = {  
 'Original': {},  
 'More Filters': {'filters': [64, 128, 256, 256]},  
 'Larger Stride': {'strides': 3},  
 'Valid Padding': {'padding': 'valid'},  
 'Average Pooling': {'pool\_type': 'avg'},  
}  
  
results = {}  
training\_times = {}  
  
# Make sure the base model directory exists  
base\_model\_dir = 'models'  
os.makedirs(base\_model\_dir, exist\_ok=True)  
  
for name, params in variants.items():  
 print(f"\nTraining: {name}")  
 model = build\_variant\_model(  
 input\_shape=X\_train\_cleaned.shape[1:],  
 num\_classes=y\_train.shape[1],  
 \*\*params  
 )  
  
 model.compile(  
 optimizer=Adam(learning\_rate=0.001),  
 loss='categorical\_crossentropy',  
 metrics=['accuracy']  
 )  
  
 start\_time = time.time()  
  
 history = model.fit(  
 X\_train\_cleaned, y\_train,  
 validation\_data=(X\_val\_cleaned, y\_val),  
 epochs=15,  
 batch\_size=8,  
 callbacks=[early\_stop, reduce\_lr],  
 verbose=1  
 )  
  
 end\_time = time.time()  
 elapsed\_time = end\_time - start\_time  
  
 results[name] = history.history  
 training\_times[name] = elapsed\_time  
 print(f"Training time for {name}: {elapsed\_time:.2f} seconds")  
  
 # Save the model  
 safe\_name = name.replace(" ", "\_") # e.g., "More Filters" -> "More\_Filters"  
 model\_path = os.path.join(base\_model\_dir, safe\_name)  
 model.save(model\_path)  
 print(f"Model saved to {model\_path}")

WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.Adam`.  
WARNING:absl:There is a known slowdown when using v2.11+ Keras optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., `tf.keras.optimizers.legacy.Adam`.  
  
  
  
Training: Original  
Epoch 1/15  
416/416 [==============================] - 18s 40ms/step - loss: 2.2870 - accuracy: 0.2436 - val\_loss: 4.6432 - val\_accuracy: 0.0604 - lr: 0.0010  
Epoch 2/15  
416/416 [==============================] - 15s 36ms/step - loss: 1.7598 - accuracy: 0.3889 - val\_loss: 1.4391 - val\_accuracy: 0.4972 - lr: 0.0010  
Epoch 3/15  
416/416 [==============================] - 17s 40ms/step - loss: 1.6283 - accuracy: 0.4430 - val\_loss: 1.2581 - val\_accuracy: 0.5983 - lr: 0.0010  
Epoch 4/15  
416/416 [==============================] - 15s 35ms/step - loss: 1.5415 - accuracy: 0.4704 - val\_loss: 1.3538 - val\_accuracy: 0.5758 - lr: 0.0010  
Epoch 5/15  
415/416 [============================>.] - ETA: 0s - loss: 1.4518 - accuracy: 0.4889  
Epoch 5: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.  
416/416 [==============================] - 15s 36ms/step - loss: 1.4506 - accuracy: 0.4893 - val\_loss: 1.4146 - val\_accuracy: 0.5000 - lr: 0.0010  
Epoch 6/15  
416/416 [==============================] - 16s 39ms/step - loss: 1.3121 - accuracy: 0.5465 - val\_loss: 1.0874 - val\_accuracy: 0.6320 - lr: 5.0000e-04  
Epoch 7/15  
416/416 [==============================] - 17s 40ms/step - loss: 1.2463 - accuracy: 0.5585 - val\_loss: 0.9307 - val\_accuracy: 0.6882 - lr: 5.0000e-04  
Epoch 8/15  
416/416 [==============================] - 16s 38ms/step - loss: 1.2137 - accuracy: 0.5841 - val\_loss: 0.8621 - val\_accuracy: 0.7093 - lr: 5.0000e-04  
Epoch 9/15  
416/416 [==============================] - 16s 38ms/step - loss: 1.1216 - accuracy: 0.6102 - val\_loss: 1.2988 - val\_accuracy: 0.5211 - lr: 5.0000e-04  
Epoch 10/15  
415/416 [============================>.] - ETA: 0s - loss: 1.0737 - accuracy: 0.6244  
Epoch 10: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.  
416/416 [==============================] - 15s 36ms/step - loss: 1.0740 - accuracy: 0.6241 - val\_loss: 0.9919 - val\_accuracy: 0.6601 - lr: 5.0000e-04  
Epoch 11/15  
416/416 [==============================] - 15s 36ms/step - loss: 0.9716 - accuracy: 0.6674 - val\_loss: 0.7208 - val\_accuracy: 0.7626 - lr: 2.5000e-04  
Epoch 12/15  
416/416 [==============================] - 15s 36ms/step - loss: 0.9485 - accuracy: 0.6761 - val\_loss: 0.7782 - val\_accuracy: 0.7458 - lr: 2.5000e-04  
Epoch 13/15  
416/416 [==============================] - 15s 35ms/step - loss: 0.8718 - accuracy: 0.6953 - val\_loss: 0.7165 - val\_accuracy: 0.7697 - lr: 2.5000e-04  
Epoch 14/15  
416/416 [==============================] - 58s 139ms/step - loss: 0.8570 - accuracy: 0.7062 - val\_loss: 0.5974 - val\_accuracy: 0.8062 - lr: 2.5000e-04  
Epoch 15/15  
416/416 [==============================] - 14s 33ms/step - loss: 0.7976 - accuracy: 0.7176 - val\_loss: 0.6984 - val\_accuracy: 0.7598 - lr: 2.5000e-04  
Training time for Original: 274.95 seconds  
INFO:tensorflow:Assets written to: models/Original/assets  
  
  
INFO:tensorflow:Assets written to: models/Original/assets  
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.Adam`.  
WARNING:absl:There is a known slowdown when using v2.11+ Keras optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., `tf.keras.optimizers.legacy.Adam`.  
  
  
Model saved to models/Original  
  
Training: More Filters  
Epoch 1/15  
416/416 [==============================] - 41s 99ms/step - loss: 2.2805 - accuracy: 0.2352 - val\_loss: 5.5529 - val\_accuracy: 0.0534 - lr: 0.0010  
Epoch 2/15  
416/416 [==============================] - 40s 97ms/step - loss: 1.8333 - accuracy: 0.3817 - val\_loss: 1.6836 - val\_accuracy: 0.4059 - lr: 0.0010  
Epoch 3/15  
416/416 [==============================] - 40s 96ms/step - loss: 1.6407 - accuracy: 0.4256 - val\_loss: 1.4642 - val\_accuracy: 0.5169 - lr: 0.0010  
Epoch 4/15  
416/416 [==============================] - 41s 98ms/step - loss: 1.5448 - accuracy: 0.4650 - val\_loss: 1.1715 - val\_accuracy: 0.6124 - lr: 0.0010  
Epoch 5/15  
416/416 [==============================] - 38s 92ms/step - loss: 1.4510 - accuracy: 0.4893 - val\_loss: 1.2056 - val\_accuracy: 0.5955 - lr: 0.0010  
Epoch 6/15  
416/416 [==============================] - ETA: 0s - loss: 1.3933 - accuracy: 0.5158  
Epoch 6: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.  
416/416 [==============================] - 41s 99ms/step - loss: 1.3933 - accuracy: 0.5158 - val\_loss: 1.6236 - val\_accuracy: 0.5562 - lr: 0.0010  
Epoch 7/15  
416/416 [==============================] - 40s 97ms/step - loss: 1.2508 - accuracy: 0.5585 - val\_loss: 0.9273 - val\_accuracy: 0.6952 - lr: 5.0000e-04  
Epoch 8/15  
416/416 [==============================] - 37s 88ms/step - loss: 1.2033 - accuracy: 0.5832 - val\_loss: 1.2926 - val\_accuracy: 0.5772 - lr: 5.0000e-04  
Epoch 9/15  
416/416 [==============================] - ETA: 0s - loss: 1.1366 - accuracy: 0.6018  
Epoch 9: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.  
416/416 [==============================] - 39s 93ms/step - loss: 1.1366 - accuracy: 0.6018 - val\_loss: 1.0497 - val\_accuracy: 0.6039 - lr: 5.0000e-04  
Epoch 10/15  
416/416 [==============================] - ETA: 0s - loss: 1.0004 - accuracy: 0.6487Restoring model weights from the end of the best epoch: 7.  
416/416 [==============================] - 40s 95ms/step - loss: 1.0004 - accuracy: 0.6487 - val\_loss: 1.0253 - val\_accuracy: 0.6475 - lr: 2.5000e-04  
Epoch 10: early stopping  
Training time for More Filters: 397.39 seconds  
INFO:tensorflow:Assets written to: models/More\_Filters/assets  
  
  
INFO:tensorflow:Assets written to: models/More\_Filters/assets  
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.Adam`.  
WARNING:absl:There is a known slowdown when using v2.11+ Keras optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., `tf.keras.optimizers.legacy.Adam`.  
  
  
Model saved to models/More\_Filters  
  
Training: Larger Stride  
Epoch 1/15  
416/416 [==============================] - 7s 17ms/step - loss: 2.3252 - accuracy: 0.2430 - val\_loss: 2.8950 - val\_accuracy: 0.1910 - lr: 0.0010  
Epoch 2/15  
416/416 [==============================] - 7s 16ms/step - loss: 1.8600 - accuracy: 0.3540 - val\_loss: 1.4567 - val\_accuracy: 0.5183 - lr: 0.0010  
Epoch 3/15  
416/416 [==============================] - 7s 17ms/step - loss: 1.6407 - accuracy: 0.4337 - val\_loss: 2.2898 - val\_accuracy: 0.3652 - lr: 0.0010  
Epoch 4/15  
416/416 [==============================] - 7s 18ms/step - loss: 1.5140 - accuracy: 0.4731 - val\_loss: 1.4329 - val\_accuracy: 0.5253 - lr: 0.0010  
Epoch 5/15  
416/416 [==============================] - 8s 20ms/step - loss: 1.4156 - accuracy: 0.5047 - val\_loss: 2.6528 - val\_accuracy: 0.3567 - lr: 0.0010  
Epoch 6/15  
416/416 [==============================] - 7s 17ms/step - loss: 1.3375 - accuracy: 0.5420 - val\_loss: 1.0725 - val\_accuracy: 0.6334 - lr: 0.0010  
Epoch 7/15  
416/416 [==============================] - 7s 17ms/step - loss: 1.2460 - accuracy: 0.5747 - val\_loss: 2.8541 - val\_accuracy: 0.4284 - lr: 0.0010  
Epoch 8/15  
414/416 [============================>.] - ETA: 0s - loss: 1.1839 - accuracy: 0.5933  
Epoch 8: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.  
416/416 [==============================] - 8s 19ms/step - loss: 1.1839 - accuracy: 0.5937 - val\_loss: 1.2038 - val\_accuracy: 0.6166 - lr: 0.0010  
Epoch 9/15  
416/416 [==============================] - 7s 17ms/step - loss: 1.0280 - accuracy: 0.6460 - val\_loss: 0.8916 - val\_accuracy: 0.7149 - lr: 5.0000e-04  
Epoch 10/15  
416/416 [==============================] - 7s 16ms/step - loss: 0.9110 - accuracy: 0.6905 - val\_loss: 0.8125 - val\_accuracy: 0.7317 - lr: 5.0000e-04  
Epoch 11/15  
416/416 [==============================] - 7s 18ms/step - loss: 0.9090 - accuracy: 0.6905 - val\_loss: 0.7897 - val\_accuracy: 0.7289 - lr: 5.0000e-04  
Epoch 12/15  
416/416 [==============================] - 7s 16ms/step - loss: 0.8265 - accuracy: 0.7164 - val\_loss: 0.8756 - val\_accuracy: 0.7317 - lr: 5.0000e-04  
Epoch 13/15  
416/416 [==============================] - 6s 16ms/step - loss: 0.7820 - accuracy: 0.7356 - val\_loss: 0.6462 - val\_accuracy: 0.7921 - lr: 5.0000e-04  
Epoch 14/15  
416/416 [==============================] - 7s 17ms/step - loss: 0.6982 - accuracy: 0.7690 - val\_loss: 0.6941 - val\_accuracy: 0.7683 - lr: 5.0000e-04  
Epoch 15/15  
414/416 [============================>.] - ETA: 0s - loss: 0.6809 - accuracy: 0.7624  
Epoch 15: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.  
416/416 [==============================] - 7s 18ms/step - loss: 0.6805 - accuracy: 0.7624 - val\_loss: 0.6849 - val\_accuracy: 0.7809 - lr: 5.0000e-04  
Training time for Larger Stride: 106.77 seconds  
INFO:tensorflow:Assets written to: models/Larger\_Stride/assets  
  
  
INFO:tensorflow:Assets written to: models/Larger\_Stride/assets  
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.Adam`.  
WARNING:absl:There is a known slowdown when using v2.11+ Keras optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., `tf.keras.optimizers.legacy.Adam`.  
  
  
Model saved to models/Larger\_Stride  
  
Training: Valid Padding  
Epoch 1/15  
416/416 [==============================] - 12s 27ms/step - loss: 2.2041 - accuracy: 0.2725 - val\_loss: 4.3111 - val\_accuracy: 0.1461 - lr: 0.0010  
Epoch 2/15  
416/416 [==============================] - 12s 28ms/step - loss: 1.7764 - accuracy: 0.3877 - val\_loss: 1.3641 - val\_accuracy: 0.5126 - lr: 0.0010  
Epoch 3/15  
416/416 [==============================] - 11s 27ms/step - loss: 1.6131 - accuracy: 0.4424 - val\_loss: 2.3774 - val\_accuracy: 0.3708 - lr: 0.0010  
Epoch 4/15  
416/416 [==============================] - 11s 25ms/step - loss: 1.5241 - accuracy: 0.4677 - val\_loss: 1.2114 - val\_accuracy: 0.5787 - lr: 0.0010  
Epoch 5/15  
416/416 [==============================] - 11s 26ms/step - loss: 1.3784 - accuracy: 0.5107 - val\_loss: 1.2092 - val\_accuracy: 0.5562 - lr: 0.0010  
Epoch 6/15  
416/416 [==============================] - 12s 28ms/step - loss: 1.3187 - accuracy: 0.5414 - val\_loss: 1.0039 - val\_accuracy: 0.6629 - lr: 0.0010  
Epoch 7/15  
416/416 [==============================] - 11s 26ms/step - loss: 1.2324 - accuracy: 0.5708 - val\_loss: 1.6989 - val\_accuracy: 0.5309 - lr: 0.0010  
Epoch 8/15  
415/416 [============================>.] - ETA: 0s - loss: 1.1568 - accuracy: 0.6105  
Epoch 8: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.  
416/416 [==============================] - 11s 26ms/step - loss: 1.1564 - accuracy: 0.6102 - val\_loss: 1.6304 - val\_accuracy: 0.5772 - lr: 0.0010  
Epoch 9/15  
416/416 [==============================] - 11s 26ms/step - loss: 1.0331 - accuracy: 0.6508 - val\_loss: 0.7054 - val\_accuracy: 0.7753 - lr: 5.0000e-04  
Epoch 10/15  
416/416 [==============================] - 11s 26ms/step - loss: 0.9406 - accuracy: 0.6731 - val\_loss: 0.7392 - val\_accuracy: 0.7472 - lr: 5.0000e-04  
Epoch 11/15  
414/416 [============================>.] - ETA: 0s - loss: 0.8857 - accuracy: 0.7011  
Epoch 11: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.  
416/416 [==============================] - 11s 26ms/step - loss: 0.8882 - accuracy: 0.7008 - val\_loss: 0.7891 - val\_accuracy: 0.7500 - lr: 5.0000e-04  
Epoch 12/15  
416/416 [==============================] - 11s 26ms/step - loss: 0.8144 - accuracy: 0.7200 - val\_loss: 0.5688 - val\_accuracy: 0.8160 - lr: 2.5000e-04  
Epoch 13/15  
416/416 [==============================] - 11s 26ms/step - loss: 0.7940 - accuracy: 0.7185 - val\_loss: 0.5646 - val\_accuracy: 0.8076 - lr: 2.5000e-04  
Epoch 14/15  
416/416 [==============================] - 11s 26ms/step - loss: 0.7358 - accuracy: 0.7411 - val\_loss: 0.5600 - val\_accuracy: 0.8118 - lr: 2.5000e-04  
Epoch 15/15  
416/416 [==============================] - 11s 26ms/step - loss: 0.6991 - accuracy: 0.7564 - val\_loss: 1.0747 - val\_accuracy: 0.6657 - lr: 2.5000e-04  
Training time for Valid Padding: 165.37 seconds  
INFO:tensorflow:Assets written to: models/Valid\_Padding/assets  
  
  
INFO:tensorflow:Assets written to: models/Valid\_Padding/assets  
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.Adam`.  
WARNING:absl:There is a known slowdown when using v2.11+ Keras optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., `tf.keras.optimizers.legacy.Adam`.  
  
  
Model saved to models/Valid\_Padding  
  
Training: Average Pooling  
Epoch 1/15  
416/416 [==============================] - 14s 32ms/step - loss: 2.1749 - accuracy: 0.2782 - val\_loss: 5.9018 - val\_accuracy: 0.0604 - lr: 0.0010  
Epoch 2/15  
416/416 [==============================] - 13s 31ms/step - loss: 1.7590 - accuracy: 0.3994 - val\_loss: 2.0076 - val\_accuracy: 0.3315 - lr: 0.0010  
Epoch 3/15  
416/416 [==============================] - 13s 31ms/step - loss: 1.6124 - accuracy: 0.4409 - val\_loss: 3.3295 - val\_accuracy: 0.3160 - lr: 0.0010  
Epoch 4/15  
415/416 [============================>.] - ETA: 0s - loss: 1.5075 - accuracy: 0.4807  
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.  
416/416 [==============================] - 13s 31ms/step - loss: 1.5091 - accuracy: 0.4803 - val\_loss: 11.3132 - val\_accuracy: 0.1868 - lr: 0.0010  
Epoch 5/15  
416/416 [==============================] - 13s 31ms/step - loss: 1.3745 - accuracy: 0.5278 - val\_loss: 1.0409 - val\_accuracy: 0.6292 - lr: 5.0000e-04  
Epoch 6/15  
416/416 [==============================] - 13s 31ms/step - loss: 1.3106 - accuracy: 0.5453 - val\_loss: 1.0887 - val\_accuracy: 0.6306 - lr: 5.0000e-04  
Epoch 7/15  
416/416 [==============================] - 13s 31ms/step - loss: 1.2349 - accuracy: 0.5744 - val\_loss: 0.8834 - val\_accuracy: 0.7065 - lr: 5.0000e-04  
Epoch 8/15  
416/416 [==============================] - 13s 31ms/step - loss: 1.1744 - accuracy: 0.5777 - val\_loss: 0.9309 - val\_accuracy: 0.7037 - lr: 5.0000e-04  
Epoch 9/15  
415/416 [============================>.] - ETA: 0s - loss: 1.1358 - accuracy: 0.6151  
Epoch 9: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.  
416/416 [==============================] - 13s 31ms/step - loss: 1.1359 - accuracy: 0.6150 - val\_loss: 0.9169 - val\_accuracy: 0.6938 - lr: 5.0000e-04  
Epoch 10/15  
416/416 [==============================] - 13s 30ms/step - loss: 1.0249 - accuracy: 0.6511 - val\_loss: 0.6906 - val\_accuracy: 0.7865 - lr: 2.5000e-04  
Epoch 11/15  
416/416 [==============================] - 13s 31ms/step - loss: 0.9913 - accuracy: 0.6577 - val\_loss: 0.7968 - val\_accuracy: 0.7388 - lr: 2.5000e-04  
Epoch 12/15  
415/416 [============================>.] - ETA: 0s - loss: 0.9584 - accuracy: 0.6798  
Epoch 12: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.  
416/416 [==============================] - 13s 31ms/step - loss: 0.9584 - accuracy: 0.6800 - val\_loss: 0.7224 - val\_accuracy: 0.7837 - lr: 2.5000e-04  
Epoch 13/15  
415/416 [============================>.] - ETA: 0s - loss: 0.8713 - accuracy: 0.7051Restoring model weights from the end of the best epoch: 10.  
416/416 [==============================] - 13s 31ms/step - loss: 0.8720 - accuracy: 0.7044 - val\_loss: 0.8976 - val\_accuracy: 0.6910 - lr: 1.2500e-04  
Epoch 13: early stopping  
Training time for Average Pooling: 167.33 seconds  
INFO:tensorflow:Assets written to: models/Average\_Pooling/assets  
  
  
INFO:tensorflow:Assets written to: models/Average\_Pooling/assets  
  
  
Model saved to models/Average\_Pooling

Plotting and analysing

import matplotlib.pyplot as plt  
  
plt.figure(figsize=(10, 6))  
for name, history in results.items():  
 val\_loss = history.get('val\_loss')  
 if val\_loss: # Make sure it exists  
 plt.plot(val\_loss, label=name)  
  
plt.title('Validation Loss Across Model Variants')  
plt.xlabel('Epoch')  
plt.ylabel('Validation Loss')  
plt.legend()  
plt.grid(True)  
plt.tight\_layout()  
plt.show()



png

Change Possible Impact More filters May improve accuracy but increase computation. Larger strides Reduces spatial resolution faster — may lose detail. Valid padding Shrinks feature maps — can lose edge info. Average pooling May retain smoother features vs. max-pooling (sharper features).

## Build of probably good model

import numpy as np  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, Input  
  
ins = X\_train\_cleaned.shape[1:]  
  
# Modified Best Model (adjustments to prevent overfitting and increase depth)  
def build\_no\_callbacks\_model(input\_shape=(ins[0], ins[1], ins[2]), num\_classes=12):  
 model = Sequential()  
  
 # 1st Convolutional Block  
 model.add(Input(shape=input\_shape))  
 model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))  
 model.add(BatchNormalization())  
 model.add(MaxPooling2D(pool\_size=(2, 2), strides=2))  
  
 # 2nd Convolutional Block  
 model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))  
 model.add(BatchNormalization())  
 model.add(MaxPooling2D(pool\_size=(2, 2), strides=2))  
  
 # 3rd Convolutional Block  
 model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))  
 model.add(BatchNormalization())  
 model.add(MaxPooling2D(pool\_size=(2, 2), strides=2))  
  
 # 4th Convolutional Block (additional block to add more depth)  
 model.add(Conv2D(128, (3, 3), activation='relu', padding='valid')) # No padding  
 model.add(BatchNormalization())  
 model.add(MaxPooling2D(pool\_size=(2, 2), strides=2))  
  
 # Flatten before Dense layers  
 model.add(Flatten())  
  
 # Fully Connected Layers  
 model.add(BatchNormalization())  
 model.add(Dense(512, activation='relu'))  
 model.add(Dropout(0.3)) # Increased dropout to 0.3 for more regularization  
  
 model.add(BatchNormalization())  
 model.add(Dense(256, activation='relu')) # Increased units in dense layer  
 model.add(Dropout(0.3)) # Dropout to avoid overfitting  
  
 model.add(BatchNormalization())  
 model.add(Dense(128, activation='relu'))  
 model.add(Dropout(0.2)) # Dropout at the final dense layer  
  
 # Output Layer  
 model.add(Dense(num\_classes, activation='softmax'))  
  
 return model

## Running the probable model but without callbacks

from tensorflow.keras.optimizers.legacy import Adam  
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping  
  
model = build\_no\_callbacks\_model()  
  
from tensorflow.keras.metrics import Precision, Recall  
  
model.compile(  
 optimizer=Adam(learning\_rate=0.001, beta\_1=0.9, beta\_2=0.999, epsilon=1e-08),  
 loss='categorical\_crossentropy',  
 metrics=['accuracy', Precision(), Recall()]  
)  
  
history1 = model.fit(  
 X\_train\_cleaned, y\_train,  
 validation\_data=(X\_val\_cleaned, y\_val),  
 epochs=30,  
 batch\_size=8  
)

Epoch 1/30  
416/416 [==============================] - 13s 31ms/step - loss: 2.1599 - accuracy: 0.2938 - precision\_2: 0.4576 - recall\_2: 0.1299 - val\_loss: 7.0940 - val\_accuracy: 0.0604 - val\_precision\_2: 0.0604 - val\_recall\_2: 0.0604  
Epoch 2/30  
416/416 [==============================] - 13s 30ms/step - loss: 1.7520 - accuracy: 0.3979 - precision\_2: 0.5398 - recall\_2: 0.2265 - val\_loss: 1.4674 - val\_accuracy: 0.4972 - val\_precision\_2: 0.6404 - val\_recall\_2: 0.2626  
Epoch 3/30  
416/416 [==============================] - 13s 30ms/step - loss: 1.6352 - accuracy: 0.4340 - precision\_2: 0.6020 - recall\_2: 0.2689 - val\_loss: 1.4389 - val\_accuracy: 0.5070 - val\_precision\_2: 0.6525 - val\_recall\_2: 0.4087  
Epoch 4/30  
416/416 [==============================] - 13s 30ms/step - loss: 1.5010 - accuracy: 0.4770 - precision\_2: 0.6145 - recall\_2: 0.2914 - val\_loss: 1.1779 - val\_accuracy: 0.5927 - val\_precision\_2: 0.7559 - val\_recall\_2: 0.4480  
Epoch 5/30  
416/416 [==============================] - 13s 30ms/step - loss: 1.3910 - accuracy: 0.5143 - precision\_2: 0.6531 - recall\_2: 0.3438 - val\_loss: 1.8685 - val\_accuracy: 0.4663 - val\_precision\_2: 0.5711 - val\_recall\_2: 0.3497  
Epoch 6/30  
416/416 [==============================] - 13s 30ms/step - loss: 1.3101 - accuracy: 0.5441 - precision\_2: 0.6674 - recall\_2: 0.3826 - val\_loss: 1.2315 - val\_accuracy: 0.5955 - val\_precision\_2: 0.7146 - val\_recall\_2: 0.4817  
Epoch 7/30  
416/416 [==============================] - 13s 30ms/step - loss: 1.2292 - accuracy: 0.5711 - precision\_2: 0.6974 - recall\_2: 0.4319 - val\_loss: 1.0741 - val\_accuracy: 0.6475 - val\_precision\_2: 0.7570 - val\_recall\_2: 0.5337  
Epoch 8/30  
416/416 [==============================] - 13s 31ms/step - loss: 1.2111 - accuracy: 0.5792 - precision\_2: 0.6938 - recall\_2: 0.4382 - val\_loss: 0.9124 - val\_accuracy: 0.6966 - val\_precision\_2: 0.8073 - val\_recall\_2: 0.5295  
Epoch 9/30  
416/416 [==============================] - 13s 31ms/step - loss: 1.1393 - accuracy: 0.6171 - precision\_2: 0.7225 - recall\_2: 0.4854 - val\_loss: 0.9414 - val\_accuracy: 0.7205 - val\_precision\_2: 0.8094 - val\_recall\_2: 0.6320  
Epoch 10/30  
416/416 [==============================] - 13s 31ms/step - loss: 1.0802 - accuracy: 0.6301 - precision\_2: 0.7352 - recall\_2: 0.5119 - val\_loss: 0.7719 - val\_accuracy: 0.7472 - val\_precision\_2: 0.8060 - val\_recall\_2: 0.6770  
Epoch 11/30  
416/416 [==============================] - 13s 31ms/step - loss: 1.0131 - accuracy: 0.6662 - precision\_2: 0.7697 - recall\_2: 0.5528 - val\_loss: 1.3748 - val\_accuracy: 0.5913 - val\_precision\_2: 0.6432 - val\_recall\_2: 0.5393  
Epoch 12/30  
416/416 [==============================] - 13s 30ms/step - loss: 1.0120 - accuracy: 0.6602 - precision\_2: 0.7552 - recall\_2: 0.5510 - val\_loss: 1.2508 - val\_accuracy: 0.6081 - val\_precision\_2: 0.6900 - val\_recall\_2: 0.5407  
Epoch 13/30  
416/416 [==============================] - 13s 30ms/step - loss: 0.9484 - accuracy: 0.6791 - precision\_2: 0.7621 - recall\_2: 0.5789 - val\_loss: 1.0445 - val\_accuracy: 0.6994 - val\_precision\_2: 0.7451 - val\_recall\_2: 0.6447  
Epoch 14/30  
416/416 [==============================] - 13s 31ms/step - loss: 0.9017 - accuracy: 0.6992 - precision\_2: 0.7673 - recall\_2: 0.6039 - val\_loss: 0.8597 - val\_accuracy: 0.7135 - val\_precision\_2: 0.7831 - val\_recall\_2: 0.6643  
Epoch 15/30  
416/416 [==============================] - 13s 31ms/step - loss: 0.8270 - accuracy: 0.7215 - precision\_2: 0.7914 - recall\_2: 0.6379 - val\_loss: 1.2504 - val\_accuracy: 0.6292 - val\_precision\_2: 0.6748 - val\_recall\_2: 0.5857  
Epoch 16/30  
416/416 [==============================] - 13s 31ms/step - loss: 0.8204 - accuracy: 0.7179 - precision\_2: 0.7779 - recall\_2: 0.6499 - val\_loss: 1.0351 - val\_accuracy: 0.7163 - val\_precision\_2: 0.7630 - val\_recall\_2: 0.6784  
Epoch 17/30  
416/416 [==============================] - 13s 30ms/step - loss: 0.8364 - accuracy: 0.7149 - precision\_2: 0.7855 - recall\_2: 0.6433 - val\_loss: 0.9065 - val\_accuracy: 0.7205 - val\_precision\_2: 0.7787 - val\_recall\_2: 0.6770  
Epoch 18/30  
416/416 [==============================] - 13s 30ms/step - loss: 0.8096 - accuracy: 0.7308 - precision\_2: 0.7907 - recall\_2: 0.6532 - val\_loss: 0.6460 - val\_accuracy: 0.7739 - val\_precision\_2: 0.8294 - val\_recall\_2: 0.7444  
Epoch 19/30  
416/416 [==============================] - 13s 30ms/step - loss: 0.7826 - accuracy: 0.7441 - precision\_2: 0.8009 - recall\_2: 0.6764 - val\_loss: 1.4715 - val\_accuracy: 0.6784 - val\_precision\_2: 0.7138 - val\_recall\_2: 0.6376  
Epoch 20/30  
416/416 [==============================] - 13s 30ms/step - loss: 0.7654 - accuracy: 0.7356 - precision\_2: 0.7979 - recall\_2: 0.6734 - val\_loss: 0.6356 - val\_accuracy: 0.7823 - val\_precision\_2: 0.8276 - val\_recall\_2: 0.7486  
Epoch 21/30  
416/416 [==============================] - 13s 30ms/step - loss: 0.6932 - accuracy: 0.7570 - precision\_2: 0.8097 - recall\_2: 0.6974 - val\_loss: 0.7340 - val\_accuracy: 0.7865 - val\_precision\_2: 0.8263 - val\_recall\_2: 0.7683  
Epoch 22/30  
416/416 [==============================] - 13s 30ms/step - loss: 0.7527 - accuracy: 0.7474 - precision\_2: 0.8063 - recall\_2: 0.6824 - val\_loss: 0.6023 - val\_accuracy: 0.8034 - val\_precision\_2: 0.8625 - val\_recall\_2: 0.7486  
Epoch 23/30  
416/416 [==============================] - 13s 30ms/step - loss: 0.7170 - accuracy: 0.7495 - precision\_2: 0.8004 - recall\_2: 0.6935 - val\_loss: 0.5358 - val\_accuracy: 0.8413 - val\_precision\_2: 0.8698 - val\_recall\_2: 0.8258  
Epoch 24/30  
416/416 [==============================] - 13s 30ms/step - loss: 0.6628 - accuracy: 0.7702 - precision\_2: 0.8182 - recall\_2: 0.7080 - val\_loss: 6.6440 - val\_accuracy: 0.3750 - val\_precision\_2: 0.3794 - val\_recall\_2: 0.3469  
Epoch 25/30  
416/416 [==============================] - 13s 30ms/step - loss: 0.6587 - accuracy: 0.7732 - precision\_2: 0.8254 - recall\_2: 0.7137 - val\_loss: 0.6642 - val\_accuracy: 0.7935 - val\_precision\_2: 0.8281 - val\_recall\_2: 0.7711  
Epoch 26/30  
416/416 [==============================] - 13s 30ms/step - loss: 0.6481 - accuracy: 0.7786 - precision\_2: 0.8251 - recall\_2: 0.7209 - val\_loss: 0.6720 - val\_accuracy: 0.8006 - val\_precision\_2: 0.8369 - val\_recall\_2: 0.7711  
Epoch 27/30  
416/416 [==============================] - 13s 30ms/step - loss: 0.5929 - accuracy: 0.7934 - precision\_2: 0.8326 - recall\_2: 0.7465 - val\_loss: 0.5395 - val\_accuracy: 0.8188 - val\_precision\_2: 0.8684 - val\_recall\_2: 0.7416  
Epoch 28/30  
416/416 [==============================] - 13s 30ms/step - loss: 0.6339 - accuracy: 0.7829 - precision\_2: 0.8354 - recall\_2: 0.7341 - val\_loss: 0.5311 - val\_accuracy: 0.8216 - val\_precision\_2: 0.8818 - val\_recall\_2: 0.7963  
Epoch 29/30  
416/416 [==============================] - 13s 30ms/step - loss: 0.5862 - accuracy: 0.7976 - precision\_2: 0.8395 - recall\_2: 0.7537 - val\_loss: 1.0180 - val\_accuracy: 0.6910 - val\_precision\_2: 0.7344 - val\_recall\_2: 0.6447  
Epoch 30/30  
416/416 [==============================] - 13s 30ms/step - loss: 0.6132 - accuracy: 0.7907 - precision\_2: 0.8425 - recall\_2: 0.7498 - val\_loss: 0.8231 - val\_accuracy: 0.7570 - val\_precision\_2: 0.8074 - val\_recall\_2: 0.7008

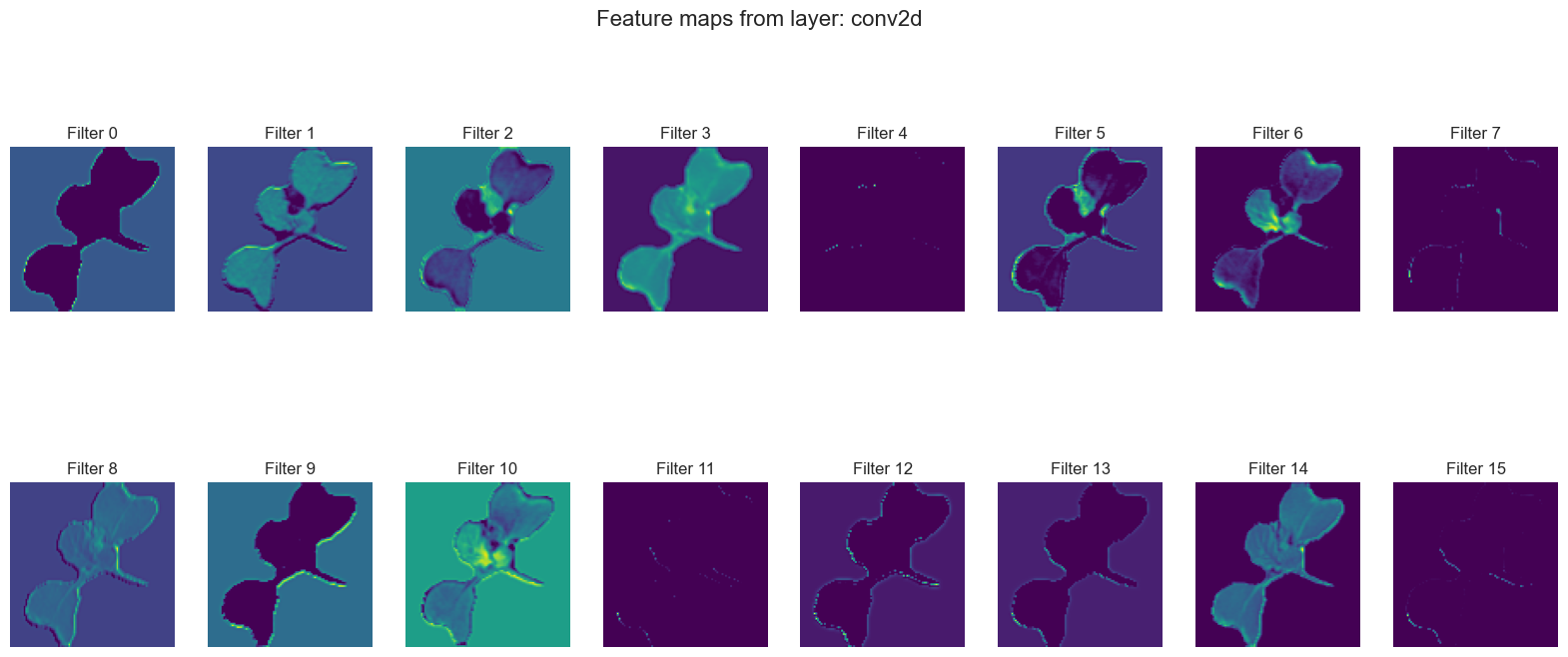
## Illustration of convolution results of a specific Layer[1,2,3]no 1 is specified now it the code, visualisation of feature maps

for i, layer in enumerate(model.layers):  
 print(i, layer.name, layer.output\_shape)

0 conv2d\_36 (None, 96, 96, 32)  
1 batch\_normalization\_63 (None, 96, 96, 32)  
2 max\_pooling2d\_32 (None, 48, 48, 32)  
3 conv2d\_37 (None, 48, 48, 64)  
4 batch\_normalization\_64 (None, 48, 48, 64)  
5 max\_pooling2d\_33 (None, 24, 24, 64)  
6 conv2d\_38 (None, 24, 24, 128)  
7 batch\_normalization\_65 (None, 24, 24, 128)  
8 max\_pooling2d\_34 (None, 12, 12, 128)  
9 conv2d\_39 (None, 10, 10, 128)  
10 batch\_normalization\_66 (None, 10, 10, 128)  
11 max\_pooling2d\_35 (None, 5, 5, 128)  
12 flatten\_9 (None, 3200)  
13 batch\_normalization\_67 (None, 3200)  
14 dense\_36 (None, 512)  
15 dropout\_27 (None, 512)  
16 batch\_normalization\_68 (None, 512)  
17 dense\_37 (None, 256)  
18 dropout\_28 (None, 256)  
19 batch\_normalization\_69 (None, 256)  
20 dense\_38 (None, 128)  
21 dropout\_29 (None, 128)  
22 dense\_39 (None, 12)

import matplotlib.pyplot as plt  
from tensorflow.keras.models import Model  
import numpy as np  
  
  
# Pick a sample image (reshape if needed)  
sample\_image = X\_val\_cleaned[0] # shape: (H, W, C)  
sample\_image = np.expand\_dims(sample\_image, axis=0) # add batch dimension  
  
# Pick a layer to visualize (e.g., 'conv2d' or layer index)  
layer\_name = 'conv2d' # or model.layers[1].name  
intermediate\_layer\_model = Model(inputs=model.input,  
 outputs=model.layers[1].output)  
  
# Get feature maps  
feature\_maps = intermediate\_layer\_model.predict(sample\_image)  
  
# Plot the feature maps  
num\_filters = feature\_maps.shape[-1]  
size = feature\_maps.shape[1]  
  
plt.figure(figsize=(20, 8))  
for i in range(min(num\_filters, 16)): # Show first 16 filters  
 ax = plt.subplot(2, 8, i + 1)  
 plt.imshow(feature\_maps[0, :, :, i], cmap='viridis')  
 plt.axis('off')  
 plt.title(f'Filter {i}')  
plt.suptitle(f"Feature maps from layer: {layer\_name}", fontsize=16)  
plt.show()

1/1 [==============================] - 0s 63ms/step



png

Insights:

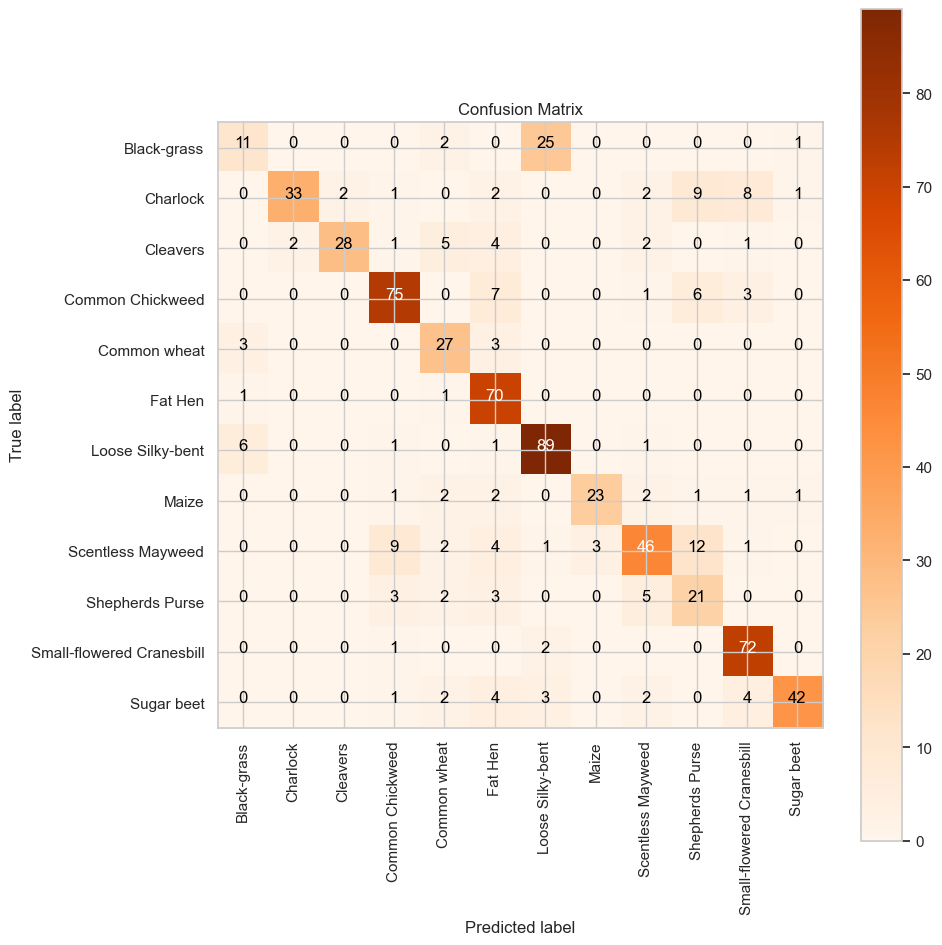
## Accuracy of the probable model without callbacks

results = model.evaluate(X\_test\_cleaned, y\_test)

23/23 [==============================] - 1s 27ms/step - loss: 0.7355 - accuracy: 0.7966 - precision\_1: 0.8349 - recall\_1: 0.7588

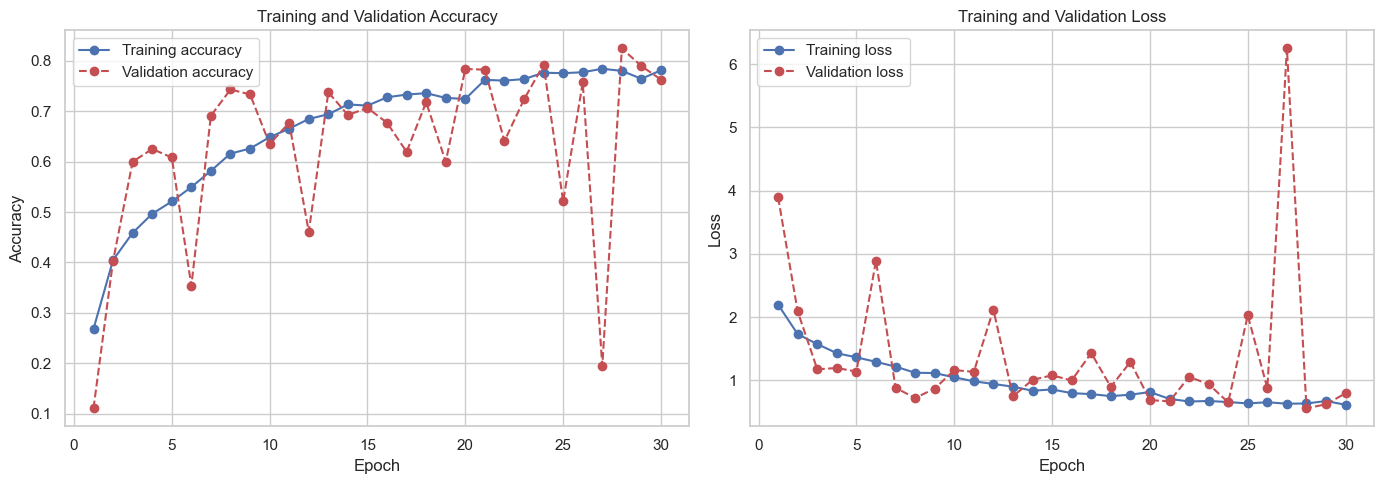
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.metrics import confusion\_matrix, classification\_report  
import itertools  
  
# --- Funkcja rysująca confusion matrix ---  
def plot\_confusion\_matrix(cm, classes,  
 normalize=False,  
 title='Confusion Matrix',  
 cmap=plt.cm.Blues): # Change the colormap here  
 plt.figure(figsize=(10, 10))  
 plt.imshow(cm, interpolation='nearest', cmap=cmap)  
 plt.title(title)  
 plt.colorbar()  
  
 tick\_marks = np.arange(len(classes))  
 plt.xticks(tick\_marks, classes, rotation=90)  
 plt.yticks(tick\_marks, classes)  
  
 if normalize:  
 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  
  
 thresh = cm.max() / 2.  
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
 plt.text(j, i, f'{cm[i, j]:.2f}' if normalize else f'{cm[i, j]}',  
 horizontalalignment="center",  
 color="white" if cm[i, j] > thresh else "black")  
  
 plt.ylabel('True label')  
 plt.xlabel('Predicted label')  
 plt.tight\_layout()  
  
# --- Predykcje modelu na testowym zbiorze ---  
predY = model.predict(X\_test\_cleaned)  
predYClasses = np.argmax(predY, axis=1) # Predicted classes (0-based index)  
trueY = np.argmax(y\_test, axis=1) # True labels (0-based index)  
  
# --- Macierz pomyłek (Confusion Matrix) ---  
confusionMTX = confusion\_matrix(trueY, predYClasses)  
  
# --- Reverse the label-to-index mapping ---  
# If label\_to\_index maps {class\_name: index}, reverse it to {index: class\_name}  
index\_to\_label = {v: k for k, v in label\_to\_index.items()}  
  
# --- Get class names for plotting ---  
categ = [index\_to\_label[i] for i in range(len(index\_to\_label))]  
  
# --- Rysowanie confusion matrix ---  
plot\_confusion\_matrix(confusionMTX, classes=categ, normalize=False, cmap=plt.cm.Oranges)  
  
# --- Classification report ---  
print("\n--- Classification Report ---")  
print(classification\_report(trueY, predYClasses, target\_names=categ))

23/23 [==============================] - 1s 29ms/step  
  
--- Classification Report ---  
 precision recall f1-score support  
  
 Black-grass 0.52 0.28 0.37 39  
 Charlock 0.94 0.57 0.71 58  
 Cleavers 0.93 0.65 0.77 43  
 Common Chickweed 0.81 0.82 0.81 92  
 Common wheat 0.63 0.82 0.71 33  
 Fat Hen 0.70 0.97 0.81 72  
 Loose Silky-bent 0.74 0.91 0.82 98  
 Maize 0.88 0.70 0.78 33  
 Scentless Mayweed 0.75 0.59 0.66 78  
 Shepherds Purse 0.43 0.62 0.51 34  
Small-flowered Cranesbill 0.80 0.96 0.87 75  
 Sugar beet 0.93 0.72 0.82 58  
  
 accuracy 0.75 713  
 macro avg 0.76 0.72 0.72 713  
 weighted avg 0.77 0.75 0.75 713



png

import matplotlib.pyplot as plt  
  
def plot\_accuracy\_and\_loss(history):  
 acc = history.history['accuracy']  
 val\_acc = history.history['val\_accuracy']  
 loss = history.history['loss']  
 val\_loss = history.history['val\_loss']  
  
 epochs = range(1, len(acc) + 1)  
  
 plt.figure(figsize=(14, 5))  
  
 # Accuracy Plot  
 plt.subplot(1, 2, 1)  
 plt.plot(epochs, acc, 'bo-', label='Training accuracy')  
 plt.plot(epochs, val\_acc, 'ro--', label='Validation accuracy')  
 plt.title('Training and Validation Accuracy')  
 plt.xlabel('Epoch')  
 plt.ylabel('Accuracy')  
 plt.legend()  
 plt.grid(True)  
  
 # Loss Plot  
 plt.subplot(1, 2, 2)  
 plt.plot(epochs, loss, 'bo-', label='Training loss')  
 plt.plot(epochs, val\_loss, 'ro--', label='Validation loss')  
 plt.title('Training and Validation Loss')  
 plt.xlabel('Epoch')  
 plt.ylabel('Loss')  
 plt.legend()  
 plt.grid(True)  
  
 plt.tight\_layout()  
 plt.show()  
  
plot\_accuracy\_and\_loss(history1)



png

There is definately some overfitting because: We will now introduce such us Early Stopping and

## Running the model with callbacks

from tensorflow.keras.optimizers import Adam  
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping  
  
# Define callbacks  
early\_stop = EarlyStopping(  
 monitor='val\_loss',  
 patience=5,  
 restore\_best\_weights=True,  
 verbose=1  
)  
  
reduce\_lr = ReduceLROnPlateau(  
 monitor='val\_loss',  
 factor=0.5,  
 patience=3,  
 min\_lr=1e-6,  
 verbose=1  
)  
  
model = build\_no\_callbacks\_model()  
  
from tensorflow.keras.metrics import Precision, Recall  
  
model.compile(  
 optimizer=Adam(learning\_rate=0.001, beta\_1=0.9, beta\_2=0.999, epsilon=1e-08),  
 loss='categorical\_crossentropy',  
 metrics=['accuracy', Precision(), Recall()]  
)  
  
history2 = model.fit(  
 X\_train\_cleaned, y\_train,  
 validation\_data=(X\_val\_cleaned, y\_val),  
 epochs=50,  
 batch\_size=8,  
 callbacks=[early\_stop, reduce\_lr]  
)

WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.Adam`.  
WARNING:absl:There is a known slowdown when using v2.11+ Keras optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., `tf.keras.optimizers.legacy.Adam`.  
  
  
Epoch 1/50  
416/416 [==============================] - 14s 32ms/step - loss: 2.2947 - accuracy: 0.2439 - precision\_3: 0.4017 - recall\_3: 0.0842 - val\_loss: 4.8314 - val\_accuracy: 0.0829 - val\_precision\_3: 0.1028 - val\_recall\_3: 0.0829 - lr: 0.0010  
Epoch 2/50  
416/416 [==============================] - 14s 33ms/step - loss: 1.8646 - accuracy: 0.3492 - precision\_3: 0.5248 - recall\_3: 0.1717 - val\_loss: 2.3300 - val\_accuracy: 0.2416 - val\_precision\_3: 0.2952 - val\_recall\_3: 0.1559 - lr: 0.0010  
Epoch 3/50  
416/416 [==============================] - 13s 31ms/step - loss: 1.5928 - accuracy: 0.4535 - precision\_3: 0.5964 - recall\_3: 0.2698 - val\_loss: 1.2176 - val\_accuracy: 0.5899 - val\_precision\_3: 0.6989 - val\_recall\_3: 0.4466 - lr: 0.0010  
Epoch 4/50  
416/416 [==============================] - 14s 33ms/step - loss: 1.4824 - accuracy: 0.4854 - precision\_3: 0.6153 - recall\_3: 0.3002 - val\_loss: 1.1927 - val\_accuracy: 0.5885 - val\_precision\_3: 0.7383 - val\_recall\_3: 0.4199 - lr: 0.0010  
Epoch 5/50  
416/416 [==============================] - 13s 30ms/step - loss: 1.3559 - accuracy: 0.5149 - precision\_3: 0.6486 - recall\_3: 0.3558 - val\_loss: 1.5491 - val\_accuracy: 0.5604 - val\_precision\_3: 0.6243 - val\_recall\_3: 0.4972 - lr: 0.0010  
Epoch 6/50  
416/416 [==============================] - 12s 30ms/step - loss: 1.2818 - accuracy: 0.5528 - precision\_3: 0.6720 - recall\_3: 0.4054 - val\_loss: 0.8871 - val\_accuracy: 0.6924 - val\_precision\_3: 0.8198 - val\_recall\_3: 0.5815 - lr: 0.0010  
Epoch 7/50  
416/416 [==============================] - 13s 30ms/step - loss: 1.1972 - accuracy: 0.5889 - precision\_3: 0.7076 - recall\_3: 0.4490 - val\_loss: 1.2585 - val\_accuracy: 0.5632 - val\_precision\_3: 0.6632 - val\_recall\_3: 0.4508 - lr: 0.0010  
Epoch 8/50  
416/416 [==============================] - 12s 30ms/step - loss: 1.1382 - accuracy: 0.6087 - precision\_3: 0.7174 - recall\_3: 0.4818 - val\_loss: 1.1298 - val\_accuracy: 0.6699 - val\_precision\_3: 0.7415 - val\_recall\_3: 0.6124 - lr: 0.0010  
Epoch 9/50  
415/416 [============================>.] - ETA: 0s - loss: 1.0432 - accuracy: 0.6401 - precision\_3: 0.7325 - recall\_3: 0.5238  
Epoch 9: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.  
416/416 [==============================] - 13s 31ms/step - loss: 1.0434 - accuracy: 0.6400 - precision\_3: 0.7321 - recall\_3: 0.5236 - val\_loss: 1.9476 - val\_accuracy: 0.4789 - val\_precision\_3: 0.5237 - val\_recall\_3: 0.4508 - lr: 0.0010  
Epoch 10/50  
416/416 [==============================] - 13s 30ms/step - loss: 0.9391 - accuracy: 0.6725 - precision\_3: 0.7590 - recall\_3: 0.5777 - val\_loss: 0.6523 - val\_accuracy: 0.7823 - val\_precision\_3: 0.8391 - val\_recall\_3: 0.7472 - lr: 5.0000e-04  
Epoch 11/50  
416/416 [==============================] - 12s 30ms/step - loss: 0.8967 - accuracy: 0.7041 - precision\_3: 0.7760 - recall\_3: 0.6075 - val\_loss: 0.7328 - val\_accuracy: 0.7612 - val\_precision\_3: 0.8211 - val\_recall\_3: 0.7093 - lr: 5.0000e-04  
Epoch 12/50  
416/416 [==============================] - 13s 30ms/step - loss: 0.8329 - accuracy: 0.7125 - precision\_3: 0.7854 - recall\_3: 0.6295 - val\_loss: 0.6920 - val\_accuracy: 0.7851 - val\_precision\_3: 0.8351 - val\_recall\_3: 0.7612 - lr: 5.0000e-04  
Epoch 13/50  
416/416 [==============================] - 13s 32ms/step - loss: 0.8698 - accuracy: 0.7047 - precision\_3: 0.7751 - recall\_3: 0.6241 - val\_loss: 0.6493 - val\_accuracy: 0.8006 - val\_precision\_3: 0.8486 - val\_recall\_3: 0.7556 - lr: 5.0000e-04  
Epoch 14/50  
416/416 [==============================] - 13s 30ms/step - loss: 0.7696 - accuracy: 0.7314 - precision\_3: 0.8039 - recall\_3: 0.6559 - val\_loss: 0.7029 - val\_accuracy: 0.7626 - val\_precision\_3: 0.8156 - val\_recall\_3: 0.7205 - lr: 5.0000e-04  
Epoch 15/50  
416/416 [==============================] - 13s 30ms/step - loss: 0.7895 - accuracy: 0.7329 - precision\_3: 0.7961 - recall\_3: 0.6589 - val\_loss: 0.6496 - val\_accuracy: 0.7837 - val\_precision\_3: 0.8406 - val\_recall\_3: 0.7626 - lr: 5.0000e-04  
Epoch 16/50  
414/416 [============================>.] - ETA: 0s - loss: 0.7348 - accuracy: 0.7467 - precision\_3: 0.8081 - recall\_3: 0.6854  
Epoch 16: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.  
416/416 [==============================] - 12s 30ms/step - loss: 0.7359 - accuracy: 0.7468 - precision\_3: 0.8079 - recall\_3: 0.6854 - val\_loss: 1.3252 - val\_accuracy: 0.6152 - val\_precision\_3: 0.6536 - val\_recall\_3: 0.5857 - lr: 5.0000e-04  
Epoch 17/50  
416/416 [==============================] - 13s 30ms/step - loss: 0.6705 - accuracy: 0.7672 - precision\_3: 0.8194 - recall\_3: 0.7110 - val\_loss: 1.0787 - val\_accuracy: 0.6475 - val\_precision\_3: 0.7336 - val\_recall\_3: 0.6110 - lr: 2.5000e-04  
Epoch 18/50  
416/416 [==============================] - 13s 31ms/step - loss: 0.6396 - accuracy: 0.7795 - precision\_3: 0.8388 - recall\_3: 0.7248 - val\_loss: 0.4682 - val\_accuracy: 0.8539 - val\_precision\_3: 0.9003 - val\_recall\_3: 0.8118 - lr: 2.5000e-04  
Epoch 19/50  
416/416 [==============================] - 14s 34ms/step - loss: 0.6063 - accuracy: 0.7910 - precision\_3: 0.8423 - recall\_3: 0.7456 - val\_loss: 0.4951 - val\_accuracy: 0.8497 - val\_precision\_3: 0.8981 - val\_recall\_3: 0.8048 - lr: 2.5000e-04  
Epoch 20/50  
416/416 [==============================] - 14s 34ms/step - loss: 0.5827 - accuracy: 0.7952 - precision\_3: 0.8431 - recall\_3: 0.7516 - val\_loss: 1.2073 - val\_accuracy: 0.6643 - val\_precision\_3: 0.7013 - val\_recall\_3: 0.6166 - lr: 2.5000e-04  
Epoch 21/50  
414/416 [============================>.] - ETA: 0s - loss: 0.6132 - accuracy: 0.7865 - precision\_3: 0.8421 - recall\_3: 0.7388  
Epoch 21: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.  
416/416 [==============================] - 13s 32ms/step - loss: 0.6122 - accuracy: 0.7868 - precision\_3: 0.8424 - recall\_3: 0.7392 - val\_loss: 0.5961 - val\_accuracy: 0.8132 - val\_precision\_3: 0.8447 - val\_recall\_3: 0.7640 - lr: 2.5000e-04  
Epoch 22/50  
416/416 [==============================] - 13s 30ms/step - loss: 0.5510 - accuracy: 0.8156 - precision\_3: 0.8605 - recall\_3: 0.7699 - val\_loss: 0.4788 - val\_accuracy: 0.8441 - val\_precision\_3: 0.8831 - val\_recall\_3: 0.8272 - lr: 1.2500e-04  
Epoch 23/50  
416/416 [==============================] - 13s 30ms/step - loss: 0.5120 - accuracy: 0.8174 - precision\_3: 0.8576 - recall\_3: 0.7753 - val\_loss: 0.4465 - val\_accuracy: 0.8680 - val\_precision\_3: 0.9067 - val\_recall\_3: 0.8329 - lr: 1.2500e-04  
Epoch 24/50  
416/416 [==============================] - 12s 30ms/step - loss: 0.5197 - accuracy: 0.8144 - precision\_3: 0.8623 - recall\_3: 0.7798 - val\_loss: 0.4517 - val\_accuracy: 0.8581 - val\_precision\_3: 0.8831 - val\_recall\_3: 0.8272 - lr: 1.2500e-04  
Epoch 25/50  
416/416 [==============================] - 12s 30ms/step - loss: 0.4778 - accuracy: 0.8367 - precision\_3: 0.8714 - recall\_3: 0.7946 - val\_loss: 0.4349 - val\_accuracy: 0.8624 - val\_precision\_3: 0.8895 - val\_recall\_3: 0.8483 - lr: 1.2500e-04  
Epoch 26/50  
416/416 [==============================] - 13s 30ms/step - loss: 0.4880 - accuracy: 0.8250 - precision\_3: 0.8608 - recall\_3: 0.7889 - val\_loss: 0.4564 - val\_accuracy: 0.8680 - val\_precision\_3: 0.8859 - val\_recall\_3: 0.8399 - lr: 1.2500e-04  
Epoch 27/50  
416/416 [==============================] - 13s 30ms/step - loss: 0.4595 - accuracy: 0.8391 - precision\_3: 0.8697 - recall\_3: 0.7991 - val\_loss: 0.4356 - val\_accuracy: 0.8708 - val\_precision\_3: 0.9041 - val\_recall\_3: 0.8343 - lr: 1.2500e-04  
Epoch 28/50  
416/416 [==============================] - 13s 30ms/step - loss: 0.4375 - accuracy: 0.8409 - precision\_3: 0.8720 - recall\_3: 0.8195 - val\_loss: 0.4210 - val\_accuracy: 0.8624 - val\_precision\_3: 0.8810 - val\_recall\_3: 0.8525 - lr: 1.2500e-04  
Epoch 29/50  
416/416 [==============================] - 13s 30ms/step - loss: 0.4424 - accuracy: 0.8385 - precision\_3: 0.8752 - recall\_3: 0.8099 - val\_loss: 0.4650 - val\_accuracy: 0.8483 - val\_precision\_3: 0.8808 - val\_recall\_3: 0.8202 - lr: 1.2500e-04  
Epoch 30/50  
416/416 [==============================] - 13s 30ms/step - loss: 0.4429 - accuracy: 0.8523 - precision\_3: 0.8818 - recall\_3: 0.8192 - val\_loss: 0.4336 - val\_accuracy: 0.8722 - val\_precision\_3: 0.8908 - val\_recall\_3: 0.8596 - lr: 1.2500e-04  
Epoch 31/50  
416/416 [==============================] - 12s 30ms/step - loss: 0.4218 - accuracy: 0.8559 - precision\_3: 0.8799 - recall\_3: 0.8223 - val\_loss: 0.4106 - val\_accuracy: 0.8834 - val\_precision\_3: 0.9048 - val\_recall\_3: 0.8539 - lr: 1.2500e-04  
Epoch 32/50  
416/416 [==============================] - 12s 30ms/step - loss: 0.4478 - accuracy: 0.8412 - precision\_3: 0.8757 - recall\_3: 0.8114 - val\_loss: 0.4171 - val\_accuracy: 0.8764 - val\_precision\_3: 0.8946 - val\_recall\_3: 0.8581 - lr: 1.2500e-04  
Epoch 33/50  
416/416 [==============================] - 13s 30ms/step - loss: 0.4422 - accuracy: 0.8469 - precision\_3: 0.8730 - recall\_3: 0.8165 - val\_loss: 0.4319 - val\_accuracy: 0.8666 - val\_precision\_3: 0.8943 - val\_recall\_3: 0.8553 - lr: 1.2500e-04  
Epoch 34/50  
415/416 [============================>.] - ETA: 0s - loss: 0.4095 - accuracy: 0.8506 - precision\_3: 0.8840 - recall\_3: 0.8238  
Epoch 34: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.  
416/416 [==============================] - 13s 30ms/step - loss: 0.4093 - accuracy: 0.8508 - precision\_3: 0.8842 - recall\_3: 0.8241 - val\_loss: 0.5429 - val\_accuracy: 0.8441 - val\_precision\_3: 0.8678 - val\_recall\_3: 0.8301 - lr: 1.2500e-04  
Epoch 35/50  
416/416 [==============================] - 13s 30ms/step - loss: 0.3823 - accuracy: 0.8565 - precision\_3: 0.8827 - recall\_3: 0.8286 - val\_loss: 0.4066 - val\_accuracy: 0.8736 - val\_precision\_3: 0.8913 - val\_recall\_3: 0.8638 - lr: 6.2500e-05  
Epoch 36/50  
416/416 [==============================] - 12s 30ms/step - loss: 0.3949 - accuracy: 0.8574 - precision\_3: 0.8878 - recall\_3: 0.8355 - val\_loss: 0.3984 - val\_accuracy: 0.8778 - val\_precision\_3: 0.8924 - val\_recall\_3: 0.8624 - lr: 6.2500e-05  
Epoch 37/50  
416/416 [==============================] - 12s 30ms/step - loss: 0.3885 - accuracy: 0.8583 - precision\_3: 0.8865 - recall\_3: 0.8382 - val\_loss: 0.4255 - val\_accuracy: 0.8680 - val\_precision\_3: 0.8920 - val\_recall\_3: 0.8581 - lr: 6.2500e-05  
Epoch 38/50  
416/416 [==============================] - 12s 30ms/step - loss: 0.3653 - accuracy: 0.8731 - precision\_3: 0.8960 - recall\_3: 0.8469 - val\_loss: 0.4117 - val\_accuracy: 0.8792 - val\_precision\_3: 0.8994 - val\_recall\_3: 0.8666 - lr: 6.2500e-05  
Epoch 39/50  
414/416 [============================>.] - ETA: 0s - loss: 0.3759 - accuracy: 0.8668 - precision\_3: 0.8943 - recall\_3: 0.8427  
Epoch 39: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.  
416/416 [==============================] - 12s 30ms/step - loss: 0.3762 - accuracy: 0.8665 - precision\_3: 0.8940 - recall\_3: 0.8424 - val\_loss: 0.4235 - val\_accuracy: 0.8722 - val\_precision\_3: 0.8894 - val\_recall\_3: 0.8469 - lr: 6.2500e-05  
Epoch 40/50  
416/416 [==============================] - 12s 30ms/step - loss: 0.3646 - accuracy: 0.8695 - precision\_3: 0.8927 - recall\_3: 0.8454 - val\_loss: 0.3998 - val\_accuracy: 0.8750 - val\_precision\_3: 0.8934 - val\_recall\_3: 0.8596 - lr: 3.1250e-05  
Epoch 41/50  
414/416 [============================>.] - ETA: 0s - loss: 0.3532 - accuracy: 0.8723 - precision\_3: 0.8979 - recall\_3: 0.8493Restoring model weights from the end of the best epoch: 36.  
416/416 [==============================] - 13s 30ms/step - loss: 0.3535 - accuracy: 0.8719 - precision\_3: 0.8973 - recall\_3: 0.8490 - val\_loss: 0.3988 - val\_accuracy: 0.8750 - val\_precision\_3: 0.8965 - val\_recall\_3: 0.8638 - lr: 3.1250e-05  
Epoch 41: early stopping

Interpretation:

✅ High precision and recall on validation data suggests the model is generalizing well and not overfitting.

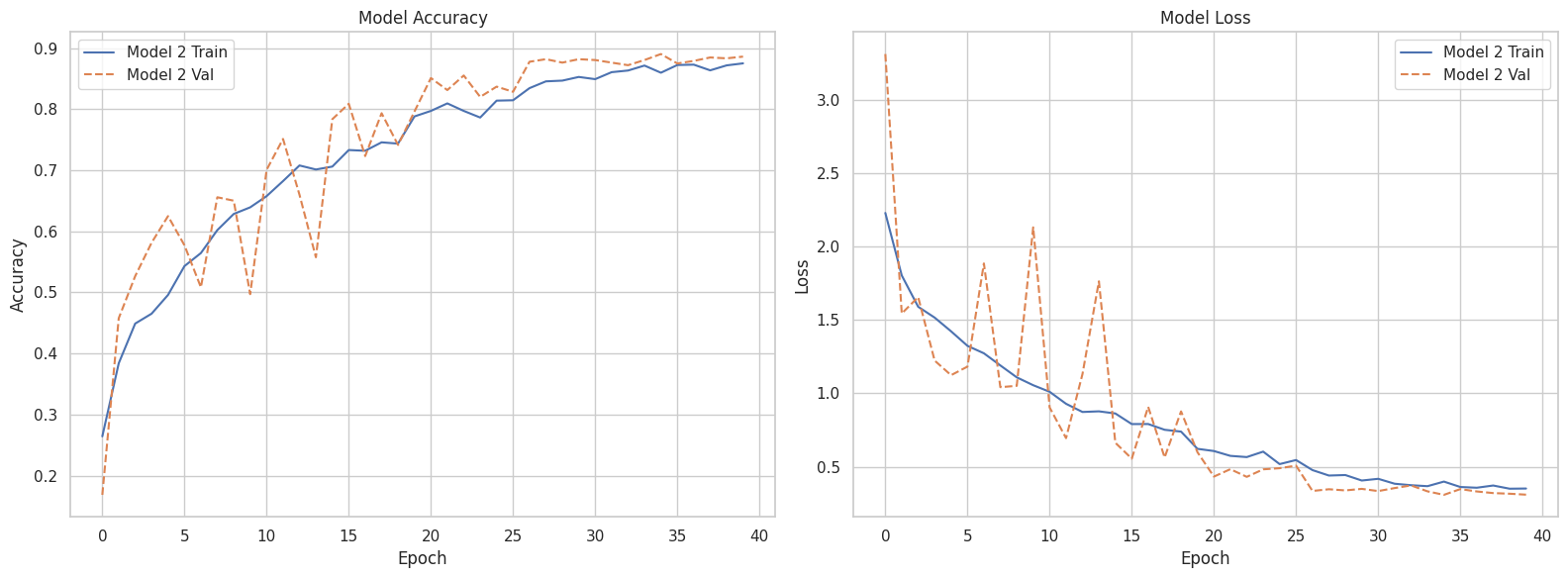
✅ Validation accuracy > training accuracy can mean regularization (e.g., dropout, early stopping) helped prevent overfitting.

✅ Low validation loss confirms the model is confident and consistent in its predictions.

📉 Early Stopping The message Restoring model weights from the end of the best epoch: 29 confirms that early stopping restored the best weights, avoiding any overfitting in the final epoch. We then increased the number of Epochs to 50 so that we can better observe the Early Stopping callback behaviour. The program this time restorted weights at 35th epoch. While at the same time we increased patience the acuracy increased from 0.85-0.88

## Evaluation of accuracy

import matplotlib.pyplot as plt  
  
# Function to extract metrics for plotting  
def extract\_metrics(history):  
 return (  
 history.history['accuracy'],  
 history.history['val\_accuracy'],  
 history.history['loss'],  
 history.history['val\_loss'],  
 )  
  
# Assuming these are available in your session already  
histories = {  
 #"Model 1": history1.history if 'history1' in globals() else None,  
 "Model 2": history2.history if 'history2' in globals() else None  
}  
  
# Plotting  
plt.figure(figsize=(16, 6))  
  
# Plot accuracy  
plt.subplot(1, 2, 1)  
for name, h in histories.items():  
 if h: # Check if history is available  
 plt.plot(h['accuracy'], label=f'{name} Train')  
 plt.plot(h['val\_accuracy'], linestyle='--', label=f'{name} Val')  
plt.title('Model Accuracy')  
plt.xlabel('Epoch')  
plt.ylabel('Accuracy')  
plt.legend()  
plt.grid(True)



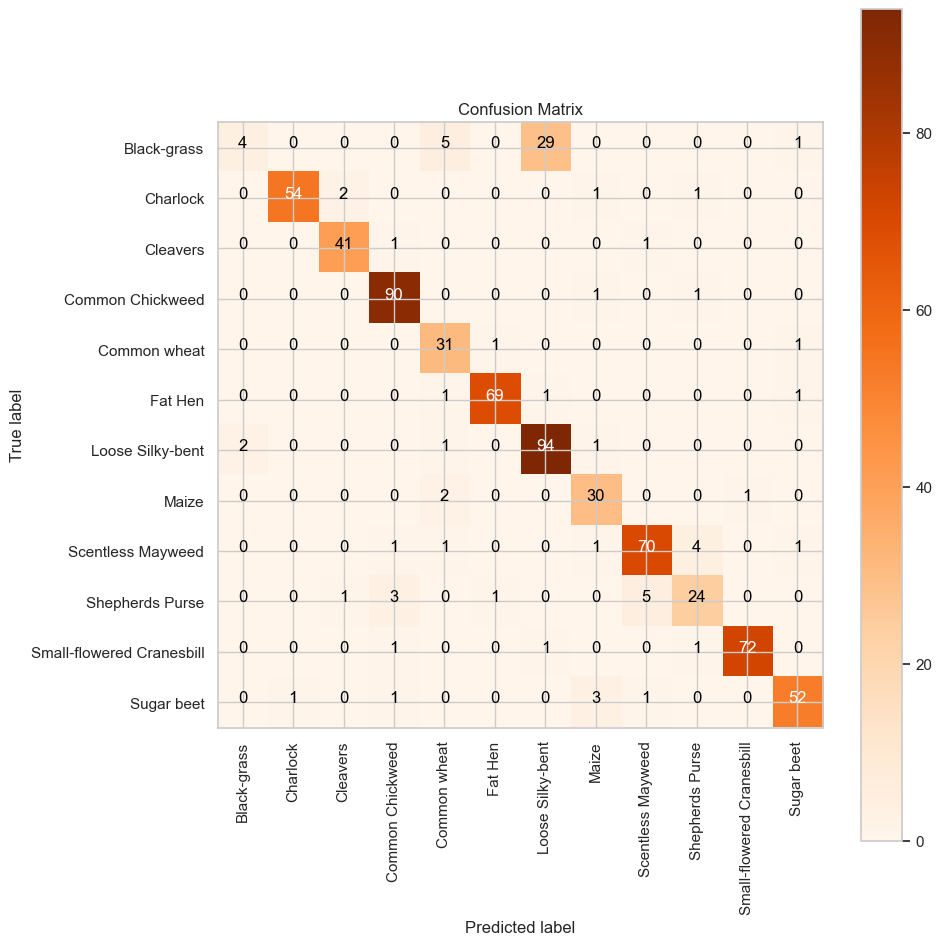
png

results = model.evaluate(X\_test\_cleaned, y\_test)

23/23 [==============================] - 1s 26ms/step - loss: 0.3907 - accuracy: 0.8850 - precision\_3: 0.8948 - recall\_3: 0.8710

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.metrics import confusion\_matrix, classification\_report  
import itertools  
  
# --- Funkcja rysująca confusion matrix ---  
def plot\_confusion\_matrix(cm, classes,  
 normalize=False,  
 title='Confusion Matrix',  
 cmap=plt.cm.Blues): # Change the colormap here  
 plt.figure(figsize=(10, 10))  
 plt.imshow(cm, interpolation='nearest', cmap=cmap)  
 plt.title(title)  
 plt.colorbar()  
  
 tick\_marks = np.arange(len(classes))  
 plt.xticks(tick\_marks, classes, rotation=90)  
 plt.yticks(tick\_marks, classes)  
  
 if normalize:  
 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  
  
 thresh = cm.max() / 2.  
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
 plt.text(j, i, f'{cm[i, j]:.2f}' if normalize else f'{cm[i, j]}',  
 horizontalalignment="center",  
 color="white" if cm[i, j] > thresh else "black")  
  
 plt.ylabel('True label')  
 plt.xlabel('Predicted label')  
 plt.tight\_layout()  
  
# --- Predykcje modelu na testowym zbiorze ---  
predY = model.predict(X\_test\_cleaned)  
predYClasses = np.argmax(predY, axis=1) # Predicted classes (0-based index)  
trueY = np.argmax(y\_test, axis=1) # True labels (0-based index)  
  
# --- Macierz pomyłek (Confusion Matrix) ---  
confusionMTX = confusion\_matrix(trueY, predYClasses)  
  
# --- Reverse the label-to-index mapping ---  
# If label\_to\_index maps {class\_name: index}, reverse it to {index: class\_name}  
index\_to\_label = {v: k for k, v in label\_to\_index.items()}  
  
# --- Get class names for plotting ---  
categ = [index\_to\_label[i] for i in range(len(index\_to\_label))]  
  
# --- Rysowanie confusion matrix ---  
plot\_confusion\_matrix(confusionMTX, classes=categ, normalize=False, cmap=plt.cm.Oranges)  
  
# --- Classification report ---  
print("\n--- Classification Report ---")  
print(classification\_report(trueY, predYClasses, target\_names=categ))

23/23 [==============================] - 1s 29ms/step  
  
--- Classification Report ---  
 precision recall f1-score support  
  
 Black-grass 0.67 0.10 0.18 39  
 Charlock 0.98 0.93 0.96 58  
 Cleavers 0.93 0.95 0.94 43  
 Common Chickweed 0.93 0.98 0.95 92  
 Common wheat 0.76 0.94 0.84 33  
 Fat Hen 0.97 0.96 0.97 72  
 Loose Silky-bent 0.75 0.96 0.84 98  
 Maize 0.81 0.91 0.86 33  
 Scentless Mayweed 0.91 0.90 0.90 78  
 Shepherds Purse 0.77 0.71 0.74 34  
Small-flowered Cranesbill 0.99 0.96 0.97 75  
 Sugar beet 0.93 0.90 0.91 58  
  
 accuracy 0.88 713  
 macro avg 0.87 0.85 0.84 713  
 weighted avg 0.88 0.88 0.87 713



png

results = model.evaluate(X\_test\_cleaned, y\_test)

23/23 [==============================] - 1s 27ms/step - loss: 0.3907 - accuracy: 0.8850 - precision\_3: 0.8948 - recall\_3: 0.8710

## Adding data-augmentation and running models with it

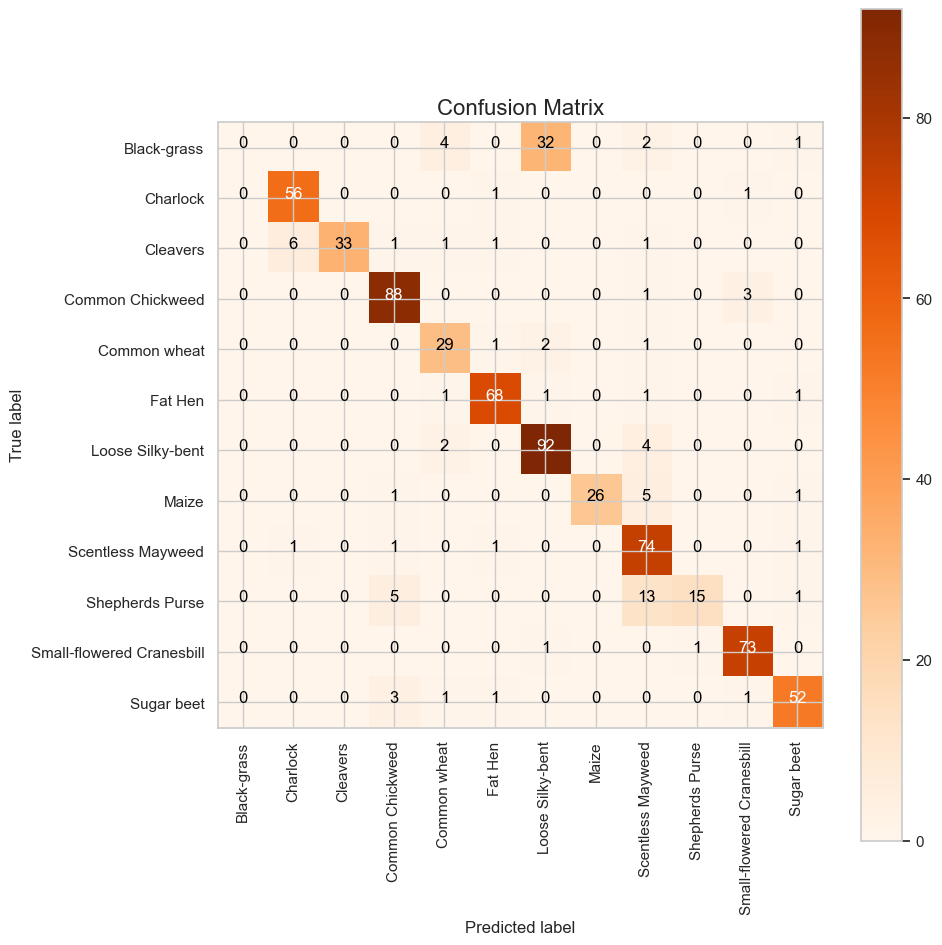
here I will implement model with data augmentation

from tensorflow.keras.optimizers import Adam  
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping  
from tensorflow.keras.preprocessing.image import ImageDataGenerator  
  
# Define callbacks  
early\_stop = EarlyStopping(  
 monitor='val\_loss',  
 patience=5,  
 restore\_best\_weights=True,  
 verbose=1  
)  
  
reduce\_lr = ReduceLROnPlateau(  
 monitor='val\_loss',  
 factor=0.5,  
 patience=3,  
 min\_lr=1e-6,  
 verbose=1  
)  
  
# Data Augmentation for training set only  
datagen = ImageDataGenerator(  
 rotation\_range=15,  
 width\_shift\_range=0.1,  
 height\_shift\_range=0.1,  
 zoom\_range=0.1,  
 horizontal\_flip=True,  
 fill\_mode='nearest'  
)  
  
# Fit the generator to your data (optional, depending on data type)  
datagen.fit(X\_train\_cleaned)  
  
# Create the model  
model = build\_no\_callbacks\_model()  
  
# Compile the model  
model.compile(  
 optimizer=Adam(learning\_rate=0.001, beta\_1=0.9, beta\_2=0.999, epsilon=1e-08),  
 loss='categorical\_crossentropy',  
 metrics = ['accuracy', Precision(), Recall()]  
)  
  
# Train the model using the augmented data  
history = model.fit(  
 datagen.flow(X\_train\_cleaned, y\_train, batch\_size=8),  
 validation\_data=(X\_val\_cleaned, y\_val),  
 epochs=50,  
 callbacks=[early\_stop, reduce\_lr],  
 steps\_per\_epoch=len(X\_train\_cleaned) // 8  
)

WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.Adam`.  
WARNING:absl:There is a known slowdown when using v2.11+ Keras optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., `tf.keras.optimizers.legacy.Adam`.  
  
  
Epoch 1/50  
415/415 [==============================] - 14s 33ms/step - loss: 2.2816 - accuracy: 0.2418 - precision\_1: 0.4114 - recall\_1: 0.0889 - val\_loss: 2.5630 - val\_accuracy: 0.2093 - val\_precision\_1: 0.2000 - val\_recall\_1: 0.0843 - lr: 0.0010  
Epoch 2/50  
415/415 [==============================] - 13s 32ms/step - loss: 1.8676 - accuracy: 0.3729 - precision\_1: 0.5009 - recall\_1: 0.1682 - val\_loss: 3.0607 - val\_accuracy: 0.3315 - val\_precision\_1: 0.3814 - val\_recall\_1: 0.3118 - lr: 0.0010  
Epoch 3/50  
415/415 [==============================] - 13s 31ms/step - loss: 1.7039 - accuracy: 0.4121 - precision\_1: 0.5609 - recall\_1: 0.2195 - val\_loss: 1.9721 - val\_accuracy: 0.4565 - val\_precision\_1: 0.5347 - val\_recall\_1: 0.3792 - lr: 0.0010  
Epoch 4/50  
415/415 [==============================] - 13s 31ms/step - loss: 1.5729 - accuracy: 0.4549 - precision\_1: 0.6176 - recall\_1: 0.2581 - val\_loss: 1.2445 - val\_accuracy: 0.5857 - val\_precision\_1: 0.6736 - val\_recall\_1: 0.5014 - lr: 0.0010  
Epoch 5/50  
415/415 [==============================] - 13s 31ms/step - loss: 1.5107 - accuracy: 0.4781 - precision\_1: 0.6349 - recall\_1: 0.2831 - val\_loss: 1.3131 - val\_accuracy: 0.5449 - val\_precision\_1: 0.6280 - val\_recall\_1: 0.4480 - lr: 0.0010  
Epoch 6/50  
415/415 [==============================] - 13s 31ms/step - loss: 1.4567 - accuracy: 0.4971 - precision\_1: 0.6355 - recall\_1: 0.3075 - val\_loss: 1.5677 - val\_accuracy: 0.5000 - val\_precision\_1: 0.5854 - val\_recall\_1: 0.3947 - lr: 0.0010  
Epoch 7/50  
415/415 [==============================] - 13s 31ms/step - loss: 1.4008 - accuracy: 0.5107 - precision\_1: 0.6465 - recall\_1: 0.3325 - val\_loss: 1.1748 - val\_accuracy: 0.6152 - val\_precision\_1: 0.7041 - val\_recall\_1: 0.4846 - lr: 0.0010  
Epoch 8/50  
415/415 [==============================] - 13s 31ms/step - loss: 1.3866 - accuracy: 0.5240 - precision\_1: 0.6567 - recall\_1: 0.3184 - val\_loss: 1.2085 - val\_accuracy: 0.6138 - val\_precision\_1: 0.7160 - val\_recall\_1: 0.5028 - lr: 0.0010  
Epoch 9/50  
415/415 [==============================] - 13s 31ms/step - loss: 1.3659 - accuracy: 0.5297 - precision\_1: 0.6576 - recall\_1: 0.3440 - val\_loss: 1.1047 - val\_accuracy: 0.6404 - val\_precision\_1: 0.7690 - val\_recall\_1: 0.4396 - lr: 0.0010  
Epoch 10/50  
415/415 [==============================] - 13s 31ms/step - loss: 1.2920 - accuracy: 0.5574 - precision\_1: 0.6911 - recall\_1: 0.3892 - val\_loss: 1.0527 - val\_accuracy: 0.6376 - val\_precision\_1: 0.7505 - val\_recall\_1: 0.5449 - lr: 0.0010  
Epoch 11/50  
415/415 [==============================] - 13s 31ms/step - loss: 1.2724 - accuracy: 0.5668 - precision\_1: 0.6875 - recall\_1: 0.4073 - val\_loss: 4.9838 - val\_accuracy: 0.2626 - val\_precision\_1: 0.2958 - val\_recall\_1: 0.2402 - lr: 0.0010  
Epoch 12/50  
415/415 [==============================] - 13s 31ms/step - loss: 1.2632 - accuracy: 0.5674 - precision\_1: 0.6916 - recall\_1: 0.4130 - val\_loss: 1.0918 - val\_accuracy: 0.6770 - val\_precision\_1: 0.7262 - val\_recall\_1: 0.6222 - lr: 0.0010  
Epoch 13/50  
415/415 [==============================] - 13s 31ms/step - loss: 1.1448 - accuracy: 0.6114 - precision\_1: 0.7058 - recall\_1: 0.4715 - val\_loss: 0.7493 - val\_accuracy: 0.7374 - val\_precision\_1: 0.8046 - val\_recall\_1: 0.6882 - lr: 0.0010  
Epoch 14/50  
415/415 [==============================] - 13s 31ms/step - loss: 1.1174 - accuracy: 0.6232 - precision\_1: 0.7323 - recall\_1: 0.4980 - val\_loss: 1.0463 - val\_accuracy: 0.6699 - val\_precision\_1: 0.7449 - val\_recall\_1: 0.6110 - lr: 0.0010  
Epoch 15/50  
415/415 [==============================] - 13s 32ms/step - loss: 1.1269 - accuracy: 0.6198 - precision\_1: 0.7308 - recall\_1: 0.4959 - val\_loss: 1.4965 - val\_accuracy: 0.5913 - val\_precision\_1: 0.6361 - val\_recall\_1: 0.5351 - lr: 0.0010  
Epoch 16/50  
414/415 [============================>.] - ETA: 0s - loss: 1.0360 - accuracy: 0.6513 - precision\_1: 0.7418 - recall\_1: 0.5322  
Epoch 16: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.  
415/415 [==============================] - 13s 31ms/step - loss: 1.0354 - accuracy: 0.6518 - precision\_1: 0.7421 - recall\_1: 0.5327 - val\_loss: 1.0958 - val\_accuracy: 0.6952 - val\_precision\_1: 0.7387 - val\_recall\_1: 0.6433 - lr: 0.0010  
Epoch 17/50  
415/415 [==============================] - 13s 31ms/step - loss: 0.9386 - accuracy: 0.6828 - precision\_1: 0.7632 - recall\_1: 0.5822 - val\_loss: 0.7106 - val\_accuracy: 0.7458 - val\_precision\_1: 0.8152 - val\_recall\_1: 0.6938 - lr: 5.0000e-04  
Epoch 18/50  
415/415 [==============================] - 13s 31ms/step - loss: 0.9126 - accuracy: 0.6895 - precision\_1: 0.7755 - recall\_1: 0.6021 - val\_loss: 0.4934 - val\_accuracy: 0.8244 - val\_precision\_1: 0.8656 - val\_recall\_1: 0.7963 - lr: 5.0000e-04  
Epoch 19/50  
415/415 [==============================] - 13s 31ms/step - loss: 0.8903 - accuracy: 0.6982 - precision\_1: 0.7773 - recall\_1: 0.6177 - val\_loss: 0.5644 - val\_accuracy: 0.8104 - val\_precision\_1: 0.8552 - val\_recall\_1: 0.7795 - lr: 5.0000e-04  
Epoch 20/50  
415/415 [==============================] - 13s 31ms/step - loss: 0.8541 - accuracy: 0.7091 - precision\_1: 0.7887 - recall\_1: 0.6268 - val\_loss: 0.8709 - val\_accuracy: 0.6938 - val\_precision\_1: 0.8011 - val\_recall\_1: 0.6222 - lr: 5.0000e-04  
Epoch 21/50  
415/415 [==============================] - ETA: 0s - loss: 0.8238 - accuracy: 0.7166 - precision\_1: 0.7898 - recall\_1: 0.6367  
Epoch 21: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.  
415/415 [==============================] - 13s 32ms/step - loss: 0.8238 - accuracy: 0.7166 - precision\_1: 0.7898 - recall\_1: 0.6367 - val\_loss: 0.5951 - val\_accuracy: 0.8048 - val\_precision\_1: 0.8459 - val\_recall\_1: 0.7711 - lr: 5.0000e-04  
Epoch 22/50  
415/415 [==============================] - 14s 33ms/step - loss: 0.8095 - accuracy: 0.7320 - precision\_1: 0.7971 - recall\_1: 0.6539 - val\_loss: 0.5043 - val\_accuracy: 0.8315 - val\_precision\_1: 0.8664 - val\_recall\_1: 0.7921 - lr: 2.5000e-04  
Epoch 23/50  
414/415 [============================>.] - ETA: 0s - loss: 0.7918 - accuracy: 0.7347 - precision\_1: 0.7997 - recall\_1: 0.6636Restoring model weights from the end of the best epoch: 18.  
415/415 [==============================] - 13s 32ms/step - loss: 0.7917 - accuracy: 0.7350 - precision\_1: 0.7999 - recall\_1: 0.6642 - val\_loss: 0.5005 - val\_accuracy: 0.8343 - val\_precision\_1: 0.8756 - val\_recall\_1: 0.7809 - lr: 2.5000e-04  
Epoch 23: early stopping

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.metrics import confusion\_matrix, classification\_report  
import itertools  
  
# --- Predict the classes on test set ---  
y\_pred\_probs = model.predict(X\_test\_cleaned)  
y\_pred = np.argmax(y\_pred\_probs, axis=1)  
y\_true = np.argmax(y\_test, axis=1)  
  
# --- Create confusion matrix ---  
conf\_matrix = confusion\_matrix(y\_true, y\_pred)  
  
# --- Create label index mapping (reverse) ---  
index\_to\_label = {v: k for k, v in label\_to\_index.items()}  
class\_names = [index\_to\_label[i] for i in range(len(index\_to\_label))]  
  
# --- Plot confusion matrix ---  
def plot\_confusion\_matrix(cm, classes,  
 normalize=False,  
 title='Confusion Matrix',  
 cmap=plt.cm.Blues):  
 plt.figure(figsize=(10, 10))  
 plt.imshow(cm, interpolation='nearest', cmap=cmap)  
 plt.title(title, fontsize=16)  
 plt.colorbar()  
  
 tick\_marks = np.arange(len(classes))  
 plt.xticks(tick\_marks, classes, rotation=90)  
 plt.yticks(tick\_marks, classes)  
  
 if normalize:  
 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  
 print("Normalized confusion matrix")  
 else:  
 print("Confusion matrix, without normalization")  
  
 thresh = cm.max() / 2.  
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
 value = f"{cm[i, j]:.2f}" if normalize else f"{cm[i, j]}"  
 plt.text(j, i, value,  
 horizontalalignment="center",  
 color="white" if cm[i, j] > thresh else "black")  
  
 plt.ylabel('True label', fontsize=12)  
 plt.xlabel('Predicted label', fontsize=12)  
 plt.tight\_layout()  
 plt.show()  
  
# --- Show confusion matrix ---  
plot\_confusion\_matrix(conf\_matrix, classes=class\_names, normalize=False, cmap=plt.cm.Oranges)  
  
# --- Optional: Classification report ---  
print("\nClassification Report:\n")  
print(classification\_report(y\_true, y\_pred, target\_names=class\_names))

23/23 [==============================] - 1s 29ms/step  
Confusion matrix, without normalization



png

Classification Report:  
  
 precision recall f1-score support  
  
 Black-grass 0.00 0.00 0.00 39  
 Charlock 0.89 0.97 0.93 58  
 Cleavers 1.00 0.77 0.87 43  
 Common Chickweed 0.89 0.96 0.92 92  
 Common wheat 0.76 0.88 0.82 33  
 Fat Hen 0.93 0.94 0.94 72  
 Loose Silky-bent 0.72 0.94 0.81 98  
 Maize 1.00 0.79 0.88 33  
 Scentless Mayweed 0.73 0.95 0.82 78  
 Shepherds Purse 0.94 0.44 0.60 34  
Small-flowered Cranesbill 0.94 0.97 0.95 75  
 Sugar beet 0.91 0.90 0.90 58  
  
 accuracy 0.85 713  
 macro avg 0.81 0.79 0.79 713  
 weighted avg 0.82 0.85 0.82 713  
  
  
  
/Users/filiporlikowski/Documents/APROF/CNN/venv/lib/python3.8/site-packages/sklearn/metrics/\_classification.py:1471: UndefinedMetricWarning:  
  
Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.  
  
/Users/filiporlikowski/Documents/APROF/CNN/venv/lib/python3.8/site-packages/sklearn/metrics/\_classification.py:1471: UndefinedMetricWarning:  
  
Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.  
  
/Users/filiporlikowski/Documents/APROF/CNN/venv/lib/python3.8/site-packages/sklearn/metrics/\_classification.py:1471: UndefinedMetricWarning:  
  
Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

## Running the pretrained Resnet50 model

import tensorflow as tf  
from tensorflow.keras.applications import ResNet50  
from tensorflow.keras.models import Model  
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout, Input  
from tensorflow.keras.optimizers import Adam  
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau  
  
# Define input shape  
input\_shape = X\_train\_cleaned.shape[1:] # (height, width, channels)  
num\_classes = 12 # change as needed  
  
# Load ResNet50 base model without the top layer  
base\_model = ResNet50(weights='imagenet', include\_top=False, input\_tensor=Input(shape=input\_shape))  
  
# Freeze base model layers to prevent training initially  
base\_model.trainable = False  
  
# Add custom top layers  
x = base\_model.output  
x = GlobalAveragePooling2D()(x)  
x = Dropout(0.3)(x)  
x = Dense(256, activation='relu')(x)  
x = Dropout(0.3)(x)  
output = Dense(num\_classes, activation='softmax')(x)  
  
# Create final model  
model = Model(inputs=base\_model.input, outputs=output)  
  
# Compile the model  
model.compile(  
 optimizer=Adam(learning\_rate=0.001),  
 loss='categorical\_crossentropy',  
 metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.metrics.Recall()]  
)  
  
# Define callbacks  
early\_stop = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True, verbose=1)  
reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.5, patience=3, min\_lr=1e-6, verbose=1)  
  
# Train the model  
history\_resnet = model.fit(  
 X\_train\_cleaned, y\_train,  
 validation\_data=(X\_val\_cleaned, y\_val),  
 epochs=30,  
 batch\_size=8,  
 callbacks=[early\_stop, reduce\_lr]  
)

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5  
94765736/94765736 [==============================] - 8s 0us/step  
  
  
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.Adam`.  
WARNING:absl:There is a known slowdown when using v2.11+ Keras optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., `tf.keras.optimizers.legacy.Adam`.  
  
  
Epoch 1/30  
416/416 [==============================] - 23s 53ms/step - loss: 2.4887 - accuracy: 0.1242 - precision\_4: 0.0000e+00 - recall\_4: 0.0000e+00 - val\_loss: 2.4084 - val\_accuracy: 0.1067 - val\_precision\_4: 0.0000e+00 - val\_recall\_4: 0.0000e+00 - lr: 0.0010  
Epoch 2/30  
416/416 [==============================] - 22s 53ms/step - loss: 2.4127 - accuracy: 0.1600 - precision\_4: 0.0000e+00 - recall\_4: 0.0000e+00 - val\_loss: 2.3756 - val\_accuracy: 0.2121 - val\_precision\_4: 0.0000e+00 - val\_recall\_4: 0.0000e+00 - lr: 0.0010  
Epoch 3/30  
416/416 [==============================] - 22s 54ms/step - loss: 2.3777 - accuracy: 0.1633 - precision\_4: 0.0000e+00 - recall\_4: 0.0000e+00 - val\_loss: 2.3234 - val\_accuracy: 0.1784 - val\_precision\_4: 0.0000e+00 - val\_recall\_4: 0.0000e+00 - lr: 0.0010  
Epoch 4/30  
416/416 [==============================] - 22s 53ms/step - loss: 2.3516 - accuracy: 0.1832 - precision\_4: 0.5333 - recall\_4: 0.0024 - val\_loss: 2.3088 - val\_accuracy: 0.2037 - val\_precision\_4: 0.0000e+00 - val\_recall\_4: 0.0000e+00 - lr: 0.0010  
Epoch 5/30  
416/416 [==============================] - 23s 56ms/step - loss: 2.3139 - accuracy: 0.2000 - precision\_4: 0.4286 - recall\_4: 0.0027 - val\_loss: 2.2675 - val\_accuracy: 0.1966 - val\_precision\_4: 0.0000e+00 - val\_recall\_4: 0.0000e+00 - lr: 0.0010  
Epoch 6/30  
416/416 [==============================] - 24s 57ms/step - loss: 2.3023 - accuracy: 0.1895 - precision\_4: 0.5263 - recall\_4: 0.0060 - val\_loss: 2.2404 - val\_accuracy: 0.2177 - val\_precision\_4: 0.0000e+00 - val\_recall\_4: 0.0000e+00 - lr: 0.0010  
Epoch 7/30  
416/416 [==============================] - 24s 58ms/step - loss: 2.2778 - accuracy: 0.2066 - precision\_4: 0.5806 - recall\_4: 0.0108 - val\_loss: 2.2397 - val\_accuracy: 0.2107 - val\_precision\_4: 0.0000e+00 - val\_recall\_4: 0.0000e+00 - lr: 0.0010  
Epoch 8/30  
416/416 [==============================] - 24s 57ms/step - loss: 2.2749 - accuracy: 0.2003 - precision\_4: 0.4630 - recall\_4: 0.0075 - val\_loss: 2.2184 - val\_accuracy: 0.2472 - val\_precision\_4: 0.0000e+00 - val\_recall\_4: 0.0000e+00 - lr: 0.0010  
Epoch 9/30  
416/416 [==============================] - 22s 52ms/step - loss: 2.2647 - accuracy: 0.2030 - precision\_4: 0.5556 - recall\_4: 0.0120 - val\_loss: 2.1960 - val\_accuracy: 0.2500 - val\_precision\_4: 1.0000 - val\_recall\_4: 0.0014 - lr: 0.0010  
Epoch 10/30  
416/416 [==============================] - 22s 53ms/step - loss: 2.2434 - accuracy: 0.2156 - precision\_4: 0.5541 - recall\_4: 0.0123 - val\_loss: 2.1831 - val\_accuracy: 0.2303 - val\_precision\_4: 1.0000 - val\_recall\_4: 0.0014 - lr: 0.0010  
Epoch 11/30  
416/416 [==============================] - 21s 52ms/step - loss: 2.2302 - accuracy: 0.2202 - precision\_4: 0.5287 - recall\_4: 0.0138 - val\_loss: 2.1683 - val\_accuracy: 0.2669 - val\_precision\_4: 0.8000 - val\_recall\_4: 0.0056 - lr: 0.0010  
Epoch 12/30  
416/416 [==============================] - 22s 54ms/step - loss: 2.2283 - accuracy: 0.2202 - precision\_4: 0.6452 - recall\_4: 0.0180 - val\_loss: 2.1497 - val\_accuracy: 0.2219 - val\_precision\_4: 0.6111 - val\_recall\_4: 0.0154 - lr: 0.0010  
Epoch 13/30  
416/416 [==============================] - 22s 53ms/step - loss: 2.2168 - accuracy: 0.2144 - precision\_4: 0.5862 - recall\_4: 0.0205 - val\_loss: 2.1474 - val\_accuracy: 0.2388 - val\_precision\_4: 0.8000 - val\_recall\_4: 0.0056 - lr: 0.0010  
Epoch 14/30  
416/416 [==============================] - 22s 54ms/step - loss: 2.2029 - accuracy: 0.2226 - precision\_4: 0.5833 - recall\_4: 0.0189 - val\_loss: 2.1399 - val\_accuracy: 0.2458 - val\_precision\_4: 0.8000 - val\_recall\_4: 0.0056 - lr: 0.0010  
Epoch 15/30  
416/416 [==============================] - 22s 53ms/step - loss: 2.1976 - accuracy: 0.2307 - precision\_4: 0.5321 - recall\_4: 0.0174 - val\_loss: 2.1510 - val\_accuracy: 0.2528 - val\_precision\_4: 0.7333 - val\_recall\_4: 0.0154 - lr: 0.0010  
Epoch 16/30  
416/416 [==============================] - 23s 55ms/step - loss: 2.1934 - accuracy: 0.2355 - precision\_4: 0.5455 - recall\_4: 0.0198 - val\_loss: 2.1241 - val\_accuracy: 0.2809 - val\_precision\_4: 1.0000 - val\_recall\_4: 0.0028 - lr: 0.0010  
Epoch 17/30  
416/416 [==============================] - 24s 57ms/step - loss: 2.1773 - accuracy: 0.2265 - precision\_4: 0.5690 - recall\_4: 0.0198 - val\_loss: 2.0939 - val\_accuracy: 0.2795 - val\_precision\_4: 0.6316 - val\_recall\_4: 0.0169 - lr: 0.0010  
Epoch 18/30  
416/416 [==============================] - 22s 54ms/step - loss: 2.1726 - accuracy: 0.2328 - precision\_4: 0.5566 - recall\_4: 0.0177 - val\_loss: 2.1102 - val\_accuracy: 0.2472 - val\_precision\_4: 1.0000 - val\_recall\_4: 0.0042 - lr: 0.0010  
Epoch 19/30  
416/416 [==============================] - 22s 53ms/step - loss: 2.1646 - accuracy: 0.2409 - precision\_4: 0.5606 - recall\_4: 0.0223 - val\_loss: 2.0977 - val\_accuracy: 0.2823 - val\_precision\_4: 0.0000e+00 - val\_recall\_4: 0.0000e+00 - lr: 0.0010  
Epoch 20/30  
416/416 [==============================] - ETA: 0s - loss: 2.1606 - accuracy: 0.2382 - precision\_4: 0.5760 - recall\_4: 0.0217  
Epoch 20: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.  
416/416 [==============================] - 22s 53ms/step - loss: 2.1606 - accuracy: 0.2382 - precision\_4: 0.5760 - recall\_4: 0.0217 - val\_loss: 2.1322 - val\_accuracy: 0.2233 - val\_precision\_4: 0.0000e+00 - val\_recall\_4: 0.0000e+00 - lr: 0.0010  
Epoch 21/30  
416/416 [==============================] - 22s 53ms/step - loss: 2.1569 - accuracy: 0.2334 - precision\_4: 0.6179 - recall\_4: 0.0229 - val\_loss: 2.0574 - val\_accuracy: 0.3006 - val\_precision\_4: 0.6286 - val\_recall\_4: 0.0309 - lr: 5.0000e-04  
Epoch 22/30  
416/416 [==============================] - 22s 52ms/step - loss: 2.1246 - accuracy: 0.2472 - precision\_4: 0.5986 - recall\_4: 0.0265 - val\_loss: 2.0480 - val\_accuracy: 0.3118 - val\_precision\_4: 0.7500 - val\_recall\_4: 0.0169 - lr: 5.0000e-04  
Epoch 23/30  
416/416 [==============================] - 22s 53ms/step - loss: 2.1351 - accuracy: 0.2406 - precision\_4: 0.5704 - recall\_4: 0.0232 - val\_loss: 2.0459 - val\_accuracy: 0.3062 - val\_precision\_4: 0.5556 - val\_recall\_4: 0.0211 - lr: 5.0000e-04  
Epoch 24/30  
416/416 [==============================] - 22s 52ms/step - loss: 2.1271 - accuracy: 0.2502 - precision\_4: 0.6184 - recall\_4: 0.0283 - val\_loss: 2.0436 - val\_accuracy: 0.3034 - val\_precision\_4: 0.7333 - val\_recall\_4: 0.0154 - lr: 5.0000e-04  
Epoch 25/30  
416/416 [==============================] - 22s 54ms/step - loss: 2.1198 - accuracy: 0.2484 - precision\_4: 0.6391 - recall\_4: 0.0256 - val\_loss: 2.0338 - val\_accuracy: 0.3132 - val\_precision\_4: 0.8182 - val\_recall\_4: 0.0126 - lr: 5.0000e-04  
Epoch 26/30  
416/416 [==============================] - 22s 54ms/step - loss: 2.1218 - accuracy: 0.2421 - precision\_4: 0.5530 - recall\_4: 0.0220 - val\_loss: 2.0285 - val\_accuracy: 0.3090 - val\_precision\_4: 0.7333 - val\_recall\_4: 0.0154 - lr: 5.0000e-04  
Epoch 27/30  
416/416 [==============================] - 22s 52ms/step - loss: 2.1120 - accuracy: 0.2442 - precision\_4: 0.6294 - recall\_4: 0.0271 - val\_loss: 2.0282 - val\_accuracy: 0.3090 - val\_precision\_4: 0.8333 - val\_recall\_4: 0.0070 - lr: 5.0000e-04  
Epoch 28/30  
416/416 [==============================] - 22s 54ms/step - loss: 2.1279 - accuracy: 0.2487 - precision\_4: 0.5504 - recall\_4: 0.0214 - val\_loss: 2.0240 - val\_accuracy: 0.3160 - val\_precision\_4: 0.7000 - val\_recall\_4: 0.0098 - lr: 5.0000e-04  
Epoch 29/30  
416/416 [==============================] - 22s 53ms/step - loss: 2.1174 - accuracy: 0.2532 - precision\_4: 0.6033 - recall\_4: 0.0220 - val\_loss: 2.0142 - val\_accuracy: 0.3006 - val\_precision\_4: 0.6452 - val\_recall\_4: 0.0281 - lr: 5.0000e-04  
Epoch 30/30  
416/416 [==============================] - 22s 53ms/step - loss: 2.1179 - accuracy: 0.2445 - precision\_4: 0.6014 - recall\_4: 0.0259 - val\_loss: 2.0351 - val\_accuracy: 0.3090 - val\_precision\_4: 0.8333 - val\_recall\_4: 0.0070 - lr: 5.0000e-04

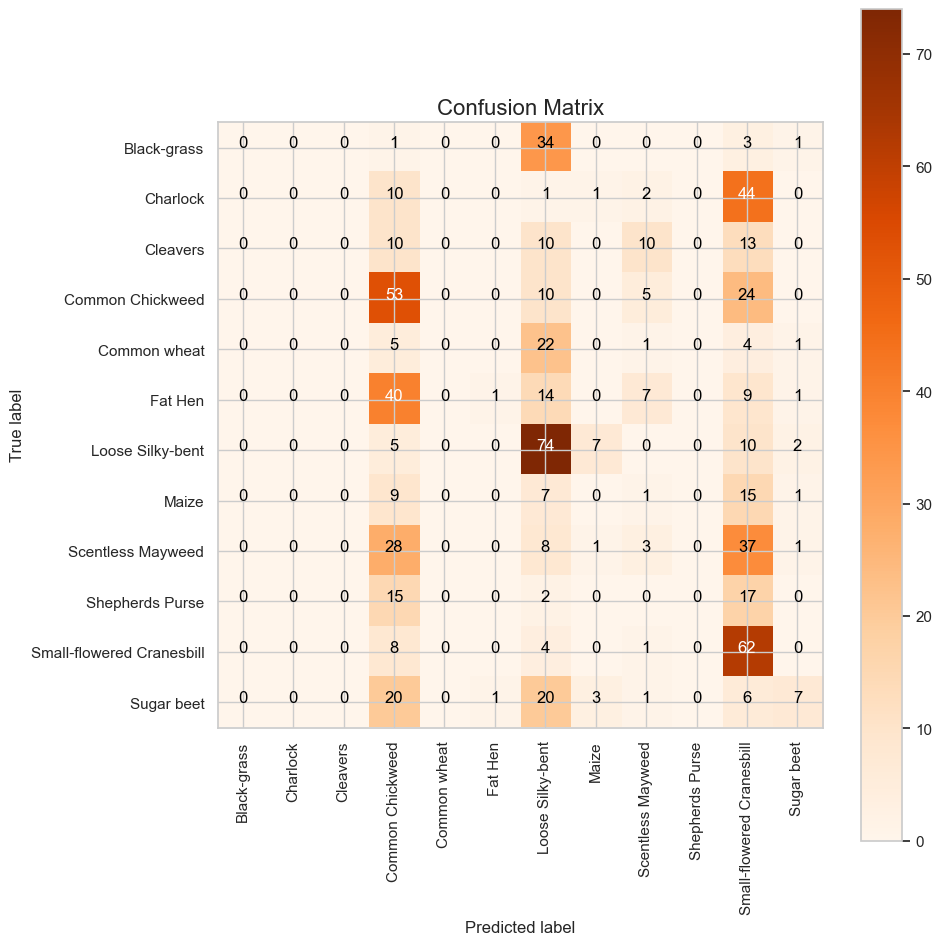
results = model.evaluate(X\_test\_cleaned, y\_test)

23/23 [==============================] - 4s 137ms/step - loss: 2.0499 - accuracy: 0.2805 - precision\_4: 0.4545 - recall\_4: 0.0070

## Results of pretrained model

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.metrics import confusion\_matrix, classification\_report  
import itertools  
  
# --- Predict on test set ---  
predY = model.predict(X\_test\_cleaned)  
predYClasses = np.argmax(predY, axis=1) # predicted class indices  
trueY = np.argmax(y\_test, axis=1) # true class indices  
  
# --- Confusion Matrix ---  
confusionMTX = confusion\_matrix(trueY, predYClasses)  
  
# --- Label mapping ---  
index\_to\_label = {v: k for k, v in label\_to\_index.items()}  
class\_names = [index\_to\_label[i] for i in range(len(index\_to\_label))]  
  
# --- Plot Confusion Matrix ---  
def plot\_confusion\_matrix(cm, classes,  
 normalize=False,  
 title='Confusion Matrix',  
 cmap=plt.cm.Oranges): # You can change the color map here  
 plt.figure(figsize=(10, 10))  
 plt.imshow(cm, interpolation='nearest', cmap=cmap)  
 plt.title(title, fontsize=16)  
 plt.colorbar()  
  
 tick\_marks = np.arange(len(classes))  
 plt.xticks(tick\_marks, classes, rotation=90)  
 plt.yticks(tick\_marks, classes)  
  
 if normalize:  
 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  
  
 thresh = cm.max() / 2.  
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
 val = f'{cm[i, j]:.2f}' if normalize else f'{cm[i, j]}'  
 plt.text(j, i, val, horizontalalignment="center",  
 color="white" if cm[i, j] > thresh else "black")  
  
 plt.ylabel('True label')  
 plt.xlabel('Predicted label')  
 plt.tight\_layout()  
 plt.show()  
  
# --- Show confusion matrix ---  
plot\_confusion\_matrix(confusionMTX, classes=class\_names, normalize=False)  
  
# --- Show classification report ---  
print("\n--- Classification Report ---")  
print(classification\_report(trueY, predYClasses, target\_names=class\_names))

23/23 [==============================] - 4s 139ms/step



png

--- Classification Report ---  
 precision recall f1-score support  
  
 Black-grass 0.00 0.00 0.00 39  
 Charlock 0.00 0.00 0.00 58  
 Cleavers 0.00 0.00 0.00 43  
 Common Chickweed 0.26 0.58 0.36 92  
 Common wheat 0.00 0.00 0.00 33  
 Fat Hen 0.50 0.01 0.03 72  
 Loose Silky-bent 0.36 0.76 0.49 98  
 Maize 0.00 0.00 0.00 33  
 Scentless Mayweed 0.10 0.04 0.06 78  
 Shepherds Purse 0.00 0.00 0.00 34  
Small-flowered Cranesbill 0.25 0.83 0.39 75  
 Sugar beet 0.50 0.12 0.19 58  
  
 accuracy 0.28 713  
 macro avg 0.16 0.19 0.13 713  
 weighted avg 0.21 0.28 0.18 713  
  
  
  
/Users/filiporlikowski/Documents/APROF/CNN/venv/lib/python3.8/site-packages/sklearn/metrics/\_classification.py:1471: UndefinedMetricWarning:  
  
Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.  
  
/Users/filiporlikowski/Documents/APROF/CNN/venv/lib/python3.8/site-packages/sklearn/metrics/\_classification.py:1471: UndefinedMetricWarning:  
  
Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.  
  
/Users/filiporlikowski/Documents/APROF/CNN/venv/lib/python3.8/site-packages/sklearn/metrics/\_classification.py:1471: UndefinedMetricWarning:  
  
Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

# --- Plot training & validation loss ---  
plt.figure(figsize=(8, 6))  
plt.plot(history\_resnet.history['loss'], label='Training Loss')  
plt.plot(history\_resnet.history['val\_loss'], label='Validation Loss')  
plt.title('Training and Validation Loss')  
plt.xlabel('Epochs')  
plt.ylabel('Loss')  
plt.legend()  
plt.grid(True)  
plt.tight\_layout()  
plt.show()

Fixes Applied: Proper ResNet50 preprocessing Partial fine-tuning (unfreezing top layers) Model rebuilt with dropout and dense layers Training with EarlyStopping and ReduceLROnPlateau Data augmentation included Classification report + confusion matrix after training

What was wrong: The application of proper input preprocessing is critical when using pretrained models like ResNet50 because these models were trained on the ImageNet dataset under very specific conditions. In your original pipeline, images were normalized to have pixel values in the range [0, 1], maintained in RGB channel order, and had no mean subtraction applied. While this format is common and perfectly valid for training models from scratch, it does not align with what ResNet50 was originally trained on. ResNet50 expects images to have pixel values in the range [0, 255], be in BGR channel order (as used by OpenCV), and have specific mean pixel values subtracted from each channel: 103.939 for Blue, 116.779 for Green, and 123.68 for Red. These preprocessing steps are crucial because they shape the way the model interprets color, contrast, and features. Feeding the model incorrectly scaled and ordered data leads to a mismatch between the features the model expects to see and the actual input, causing the convolutional filters—learned on differently preprocessed data—to activate incorrectly or not at all. As a result, performance suffers, often severely. By applying the correct preprocessing function (preprocess\_input from keras.applications.resnet50), we ensure the input is transformed to match the conditions of the original ImageNet training. This allows ResNet50 to fully leverage its pretrained filters, resulting in significantly improved feature extraction and classification accuracy. Proper preprocessing bridges the gap between your dataset and the pretrained model, making transfer learning both viable and effective.

Your Original Input Format Your images were:

Resized to 224x224x3 Normalized to float32 values in the range [0, 1] (via img.astype(np.float32) / 255.0) In RGB color order (standard in most image preprocessing pipelines) This format is commonly used when training models from scratch or with custom architectures.

⚠️ Why That’s a Problem for ResNet50 ResNet50, when loaded with weights=‘imagenet’, was pretrained using the ImageNet dataset, which follows a very specific preprocessing scheme:

Requirement Your Format ResNet50 Expectation Data Type float32 float32 Pixel Range [0, 1] [0, 255] (uint8-like) Mean Subtraction ❌ Not applied ✅ Subtracts [103.939, 116.779, 123.68] from BGR channels Channel Order RGB BGR (like OpenCV format) So even though your image shapes and types were valid, the pixel values and color interpretation were wrong for ResNet50. This misalignment caused the model to misinterpret image content, drastically hurting performance.

✅ What preprocess\_input Does (for resnet50) The preprocess\_input function in keras.applications.resnet50 performs:

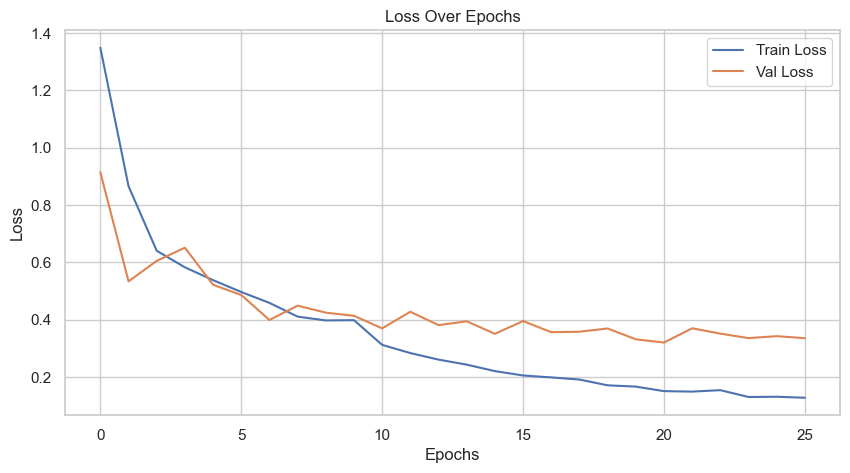
Converts RGB → BGR (reordering channels) Scales from [0, 255] to ImageNet format Subtracts ImageNet mean pixel values: [103.939, 116.779, 123.68] for Blue, Green, Red channels respectively In short, preprocess\_input mimics the exact preprocessing ResNet50 used during training, which is crucial for transfer learning to work effectively.

🧠 Summary Without proper preprocessing:

You fed ResNet50 data it wasn’t trained to understand. It saw color distributions and pixel intensities that didn’t match ImageNet, confusing the learned filters. Result: poor feature extraction → poor performance. After applying the correct preprocessing, the model receives input in a familiar format, allowing it to reuse its pretrained knowledge effectively.

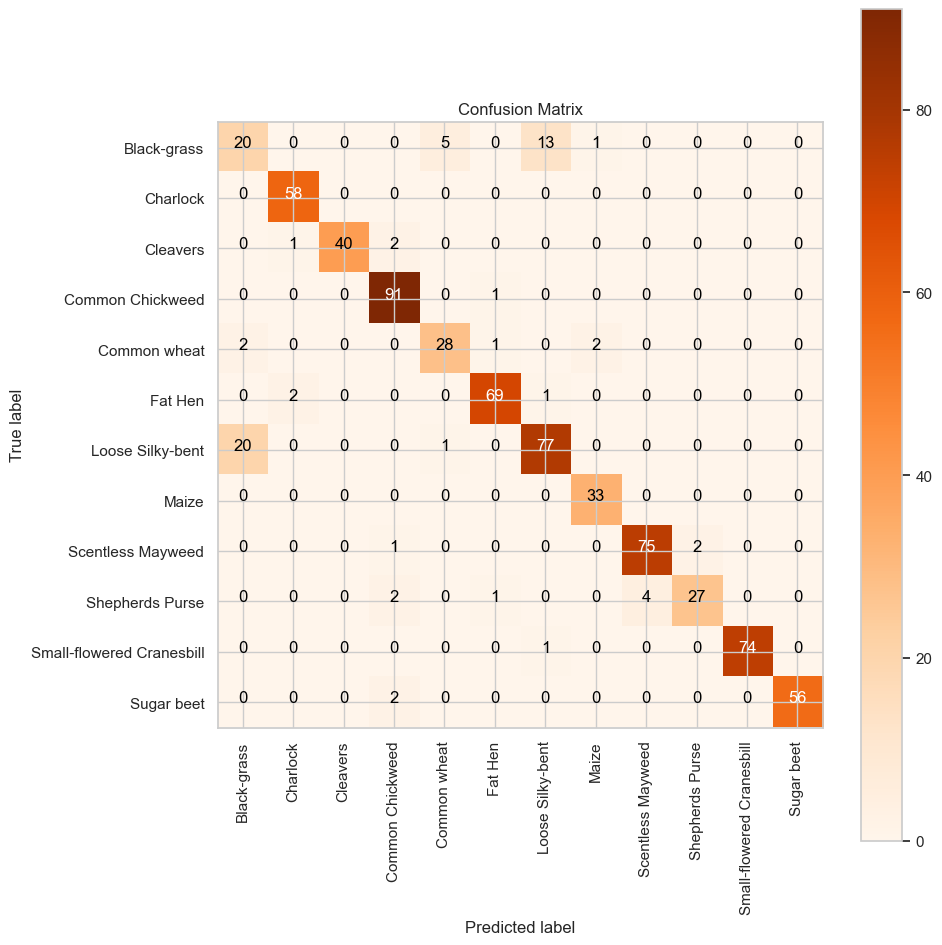
import tensorflow as tf  
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess\_input  
from tensorflow.keras.models import Model  
from tensorflow.keras.layers import Input, GlobalAveragePooling2D, Dropout, Dense  
from tensorflow.keras.optimizers import Adam  
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau  
import numpy as np  
from sklearn.metrics import confusion\_matrix, classification\_report  
import matplotlib.pyplot as plt  
import itertools  
  
# --- Preprocess images for ResNet50 (expects 0-255 range) ---  
X\_train\_ready = preprocess\_input(X\_train\_cleaned \* 255.0)  
X\_val\_ready = preprocess\_input(X\_val\_cleaned \* 255.0)  
X\_test\_ready = preprocess\_input(X\_test\_cleaned \* 255.0)  
  
# --- Model definition ---  
input\_shape = X\_train\_ready.shape[1:]  
num\_classes = y\_train.shape[1]  
  
base\_model = ResNet50(weights='imagenet', include\_top=False, input\_tensor=Input(shape=input\_shape))  
  
# Unfreeze top 30 layers for fine-tuning  
for layer in base\_model.layers[:-30]:  
 layer.trainable = False  
for layer in base\_model.layers[-30:]:  
 layer.trainable = True  
  
# Custom head  
x = base\_model.output  
x = GlobalAveragePooling2D()(x)  
x = Dropout(0.3)(x)  
x = Dense(256, activation='relu')(x)  
x = Dropout(0.3)(x)  
output = Dense(num\_classes, activation='softmax')(x)  
  
# Final model  
model = Model(inputs=base\_model.input, outputs=output)  
  
# Compile  
model.compile(  
 optimizer=Adam(learning\_rate=0.0005),  
 loss='categorical\_crossentropy',  
 metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.metrics.Recall()]  
)  
  
# Callbacks  
early\_stop = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True, verbose=1)  
reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.5, patience=3, min\_lr=1e-6, verbose=1)  
  
# Data Augmentation  
datagen = ImageDataGenerator(  
 rotation\_range=15,  
 width\_shift\_range=0.1,  
 height\_shift\_range=0.1,  
 zoom\_range=0.1,  
 horizontal\_flip=True,  
 fill\_mode='nearest'  
)  
datagen.fit(X\_train\_ready)  
  
# Train  
history\_resnet = model.fit(  
 datagen.flow(X\_train\_ready, y\_train, batch\_size=8),  
 validation\_data=(X\_val\_ready, y\_val),  
 epochs=30,  
 callbacks=[early\_stop, reduce\_lr],  
 steps\_per\_epoch=len(X\_train\_ready) // 8,  
 verbose=1  
)  
  
# --- Plot training history ---  
plt.figure(figsize=(10,5))  
plt.plot(history\_resnet.history['loss'], label='Train Loss')  
plt.plot(history\_resnet.history['val\_loss'], label='Val Loss')  
plt.title('Loss Over Epochs')  
plt.xlabel('Epochs')  
plt.ylabel('Loss')  
plt.legend()  
plt.grid(True)  
plt.show()  
  
# --- Evaluation on test set ---  
predY = model.predict(X\_test\_ready)  
predYClasses = np.argmax(predY, axis=1)  
trueY = np.argmax(y\_test, axis=1)  
  
# Confusion Matrix  
confusionMTX = confusion\_matrix(trueY, predYClasses)  
  
# Class labels  
index\_to\_label = {v: k for k, v in label\_to\_index.items()}  
categ = [index\_to\_label[i] for i in range(len(index\_to\_label))]  
  
# Plot Confusion Matrix  
def plot\_confusion\_matrix(cm, classes, normalize=False, title='Confusion Matrix', cmap=plt.cm.Oranges):  
 plt.figure(figsize=(10, 10))  
 plt.imshow(cm, interpolation='nearest', cmap=cmap)  
 plt.title(title)  
 plt.colorbar()  
 tick\_marks = np.arange(len(classes))  
 plt.xticks(tick\_marks, classes, rotation=90)  
 plt.yticks(tick\_marks, classes)  
 if normalize:  
 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  
 thresh = cm.max() / 2.  
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
 plt.text(j, i, f'{cm[i, j]}',  
 horizontalalignment="center",  
 color="white" if cm[i, j] > thresh else "black")  
 plt.ylabel('True label')  
 plt.xlabel('Predicted label')  
 plt.tight\_layout()  
  
plot\_confusion\_matrix(confusionMTX, classes=categ)  
  
# --- Classification Report ---  
print("\nClassification Report:\n")  
print(classification\_report(trueY, predYClasses, target\_names=categ))

WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.Adam`.  
WARNING:absl:There is a known slowdown when using v2.11+ Keras optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., `tf.keras.optimizers.legacy.Adam`.  
  
  
Epoch 1/30  
415/415 [==============================] - 39s 91ms/step - loss: 1.3485 - accuracy: 0.5592 - precision\_5: 0.6701 - recall\_5: 0.4293 - val\_loss: 0.9144 - val\_accuracy: 0.7275 - val\_precision\_5: 0.7748 - val\_recall\_5: 0.7008 - lr: 5.0000e-04  
Epoch 2/30  
415/415 [==============================] - 36s 87ms/step - loss: 0.8657 - accuracy: 0.6970 - precision\_5: 0.7707 - recall\_5: 0.6120 - val\_loss: 0.5341 - val\_accuracy: 0.8272 - val\_precision\_5: 0.8629 - val\_recall\_5: 0.7781 - lr: 5.0000e-04  
Epoch 3/30  
415/415 [==============================] - 38s 91ms/step - loss: 0.6406 - accuracy: 0.7754 - precision\_5: 0.8245 - recall\_5: 0.7154 - val\_loss: 0.6053 - val\_accuracy: 0.8104 - val\_precision\_5: 0.8311 - val\_recall\_5: 0.7809 - lr: 5.0000e-04  
Epoch 4/30  
415/415 [==============================] - 38s 91ms/step - loss: 0.5830 - accuracy: 0.7938 - precision\_5: 0.8374 - recall\_5: 0.7516 - val\_loss: 0.6514 - val\_accuracy: 0.8006 - val\_precision\_5: 0.8306 - val\_recall\_5: 0.7711 - lr: 5.0000e-04  
Epoch 5/30  
415/415 [==============================] - 39s 94ms/step - loss: 0.5382 - accuracy: 0.8092 - precision\_5: 0.8496 - recall\_5: 0.7648 - val\_loss: 0.5221 - val\_accuracy: 0.8258 - val\_precision\_5: 0.8550 - val\_recall\_5: 0.8034 - lr: 5.0000e-04  
Epoch 6/30  
415/415 [==============================] - 39s 94ms/step - loss: 0.4969 - accuracy: 0.8191 - precision\_5: 0.8519 - recall\_5: 0.7751 - val\_loss: 0.4864 - val\_accuracy: 0.8511 - val\_precision\_5: 0.8697 - val\_recall\_5: 0.8343 - lr: 5.0000e-04  
Epoch 7/30  
415/415 [==============================] - 36s 86ms/step - loss: 0.4589 - accuracy: 0.8435 - precision\_5: 0.8686 - recall\_5: 0.8113 - val\_loss: 0.3994 - val\_accuracy: 0.8610 - val\_precision\_5: 0.8765 - val\_recall\_5: 0.8469 - lr: 5.0000e-04  
Epoch 8/30  
415/415 [==============================] - 37s 90ms/step - loss: 0.4110 - accuracy: 0.8517 - precision\_5: 0.8772 - recall\_5: 0.8203 - val\_loss: 0.4493 - val\_accuracy: 0.8525 - val\_precision\_5: 0.8672 - val\_recall\_5: 0.8441 - lr: 5.0000e-04  
Epoch 9/30  
415/415 [==============================] - 37s 88ms/step - loss: 0.3978 - accuracy: 0.8559 - precision\_5: 0.8822 - recall\_5: 0.8306 - val\_loss: 0.4249 - val\_accuracy: 0.8441 - val\_precision\_5: 0.8704 - val\_recall\_5: 0.8301 - lr: 5.0000e-04  
Epoch 10/30  
415/415 [==============================] - ETA: 0s - loss: 0.3988 - accuracy: 0.8505 - precision\_5: 0.8730 - recall\_5: 0.8270  
Epoch 10: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.  
415/415 [==============================] - 38s 91ms/step - loss: 0.3988 - accuracy: 0.8505 - precision\_5: 0.8730 - recall\_5: 0.8270 - val\_loss: 0.4139 - val\_accuracy: 0.8596 - val\_precision\_5: 0.8709 - val\_recall\_5: 0.8525 - lr: 5.0000e-04  
Epoch 11/30  
415/415 [==============================] - 40s 97ms/step - loss: 0.3126 - accuracy: 0.8869 - precision\_5: 0.9026 - recall\_5: 0.8686 - val\_loss: 0.3702 - val\_accuracy: 0.8694 - val\_precision\_5: 0.8844 - val\_recall\_5: 0.8596 - lr: 2.5000e-04  
Epoch 12/30  
415/415 [==============================] - 37s 88ms/step - loss: 0.2840 - accuracy: 0.8930 - precision\_5: 0.9090 - recall\_5: 0.8767 - val\_loss: 0.4280 - val\_accuracy: 0.8680 - val\_precision\_5: 0.8810 - val\_recall\_5: 0.8525 - lr: 2.5000e-04  
Epoch 13/30  
415/415 [==============================] - 37s 90ms/step - loss: 0.2612 - accuracy: 0.8993 - precision\_5: 0.9149 - recall\_5: 0.8818 - val\_loss: 0.3815 - val\_accuracy: 0.8792 - val\_precision\_5: 0.8881 - val\_recall\_5: 0.8694 - lr: 2.5000e-04  
Epoch 14/30  
415/415 [==============================] - ETA: 0s - loss: 0.2436 - accuracy: 0.9087 - precision\_5: 0.9196 - recall\_5: 0.8966  
Epoch 14: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.  
415/415 [==============================] - 40s 98ms/step - loss: 0.2436 - accuracy: 0.9087 - precision\_5: 0.9196 - recall\_5: 0.8966 - val\_loss: 0.3947 - val\_accuracy: 0.8848 - val\_precision\_5: 0.8927 - val\_recall\_5: 0.8764 - lr: 2.5000e-04  
Epoch 15/30  
415/415 [==============================] - 37s 89ms/step - loss: 0.2211 - accuracy: 0.9162 - precision\_5: 0.9299 - recall\_5: 0.9084 - val\_loss: 0.3512 - val\_accuracy: 0.8890 - val\_precision\_5: 0.8922 - val\_recall\_5: 0.8834 - lr: 1.2500e-04  
Epoch 16/30  
415/415 [==============================] - 36s 86ms/step - loss: 0.2059 - accuracy: 0.9255 - precision\_5: 0.9328 - recall\_5: 0.9171 - val\_loss: 0.3955 - val\_accuracy: 0.8694 - val\_precision\_5: 0.8736 - val\_recall\_5: 0.8638 - lr: 1.2500e-04  
Epoch 17/30  
415/415 [==============================] - 41s 100ms/step - loss: 0.1991 - accuracy: 0.9249 - precision\_5: 0.9320 - recall\_5: 0.9177 - val\_loss: 0.3569 - val\_accuracy: 0.8862 - val\_precision\_5: 0.8929 - val\_recall\_5: 0.8778 - lr: 1.2500e-04  
Epoch 18/30  
415/415 [==============================] - ETA: 0s - loss: 0.1917 - accuracy: 0.9267 - precision\_5: 0.9371 - recall\_5: 0.9162  
Epoch 18: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.  
415/415 [==============================] - 40s 96ms/step - loss: 0.1917 - accuracy: 0.9267 - precision\_5: 0.9371 - recall\_5: 0.9162 - val\_loss: 0.3584 - val\_accuracy: 0.8876 - val\_precision\_5: 0.8962 - val\_recall\_5: 0.8848 - lr: 1.2500e-04  
Epoch 19/30  
415/415 [==============================] - 35s 84ms/step - loss: 0.1716 - accuracy: 0.9358 - precision\_5: 0.9438 - recall\_5: 0.9273 - val\_loss: 0.3697 - val\_accuracy: 0.8792 - val\_precision\_5: 0.8832 - val\_recall\_5: 0.8708 - lr: 6.2500e-05  
Epoch 20/30  
415/415 [==============================] - 35s 84ms/step - loss: 0.1671 - accuracy: 0.9379 - precision\_5: 0.9452 - recall\_5: 0.9313 - val\_loss: 0.3319 - val\_accuracy: 0.8933 - val\_precision\_5: 0.8949 - val\_recall\_5: 0.8848 - lr: 6.2500e-05  
Epoch 21/30  
415/415 [==============================] - 38s 93ms/step - loss: 0.1513 - accuracy: 0.9463 - precision\_5: 0.9508 - recall\_5: 0.9379 - val\_loss: 0.3207 - val\_accuracy: 0.8989 - val\_precision\_5: 0.9030 - val\_recall\_5: 0.8890 - lr: 6.2500e-05  
Epoch 22/30  
415/415 [==============================] - 40s 97ms/step - loss: 0.1496 - accuracy: 0.9439 - precision\_5: 0.9506 - recall\_5: 0.9403 - val\_loss: 0.3704 - val\_accuracy: 0.8834 - val\_precision\_5: 0.8837 - val\_recall\_5: 0.8750 - lr: 6.2500e-05  
Epoch 23/30  
415/415 [==============================] - 39s 93ms/step - loss: 0.1546 - accuracy: 0.9424 - precision\_5: 0.9484 - recall\_5: 0.9370 - val\_loss: 0.3518 - val\_accuracy: 0.8862 - val\_precision\_5: 0.8919 - val\_recall\_5: 0.8806 - lr: 6.2500e-05  
Epoch 24/30  
415/415 [==============================] - ETA: 0s - loss: 0.1308 - accuracy: 0.9484 - precision\_5: 0.9548 - recall\_5: 0.9430  
Epoch 24: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.  
415/415 [==============================] - 39s 94ms/step - loss: 0.1308 - accuracy: 0.9484 - precision\_5: 0.9548 - recall\_5: 0.9430 - val\_loss: 0.3361 - val\_accuracy: 0.8848 - val\_precision\_5: 0.8946 - val\_recall\_5: 0.8820 - lr: 6.2500e-05  
Epoch 25/30  
415/415 [==============================] - 38s 92ms/step - loss: 0.1317 - accuracy: 0.9524 - precision\_5: 0.9570 - recall\_5: 0.9457 - val\_loss: 0.3429 - val\_accuracy: 0.8904 - val\_precision\_5: 0.8950 - val\_recall\_5: 0.8862 - lr: 3.1250e-05  
Epoch 26/30  
415/415 [==============================] - ETA: 0s - loss: 0.1282 - accuracy: 0.9512 - precision\_5: 0.9559 - recall\_5: 0.9472Restoring model weights from the end of the best epoch: 21.  
415/415 [==============================] - 38s 91ms/step - loss: 0.1282 - accuracy: 0.9512 - precision\_5: 0.9559 - recall\_5: 0.9472 - val\_loss: 0.3359 - val\_accuracy: 0.9003 - val\_precision\_5: 0.9051 - val\_recall\_5: 0.8975 - lr: 3.1250e-05  
Epoch 26: early stopping



png

23/23 [==============================] - 3s 131ms/step  
  
Classification Report:  
  
 precision recall f1-score support  
  
 Black-grass 0.48 0.51 0.49 39  
 Charlock 0.95 1.00 0.97 58  
 Cleavers 1.00 0.93 0.96 43  
 Common Chickweed 0.93 0.99 0.96 92  
 Common wheat 0.82 0.85 0.84 33  
 Fat Hen 0.96 0.96 0.96 72  
 Loose Silky-bent 0.84 0.79 0.81 98  
 Maize 0.92 1.00 0.96 33  
 Scentless Mayweed 0.95 0.96 0.96 78  
 Shepherds Purse 0.93 0.79 0.86 34  
Small-flowered Cranesbill 1.00 0.99 0.99 75  
 Sugar beet 1.00 0.97 0.98 58  
  
 accuracy 0.91 713  
 macro avg 0.90 0.89 0.89 713  
 weighted avg 0.91 0.91 0.91 713



png