# file2

May 1, 2024

[ ]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
import seaborn as sns
import matplotlib.pyplot as plt


import warnings
warnings.filterwarnings('ignore')
```

[ ]:
```
from sklearn.neural_network import MLPClassifier

import os
from os import listdir
from PIL import Image
import keras
from keras.models import Sequential
from tensorflow.keras import layers
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

[ ]:
```
# import tensorflow_hub as hub
from tensorflow.keras import utils
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
```

[ ]:
```
input_shape_2D = (224,224)
input_shape_3D = (224,224,3)
seed = 1
batch_size = 32
epochs = 30
```

[ ]:
```
data = tf.keras.utils.
 ↪image_dataset_from_directory(directory="rice_leaf_diseases",
                                                  labels='inferred',
                                                  label_mode='int',
```

```
                                                    class_names=None ,
                                                    color_mode='rgb',
                                                    image_size=input_shape_2D,
                                                    seed=seed)
```

```
Found 283 files belonging to 7 classes.
```

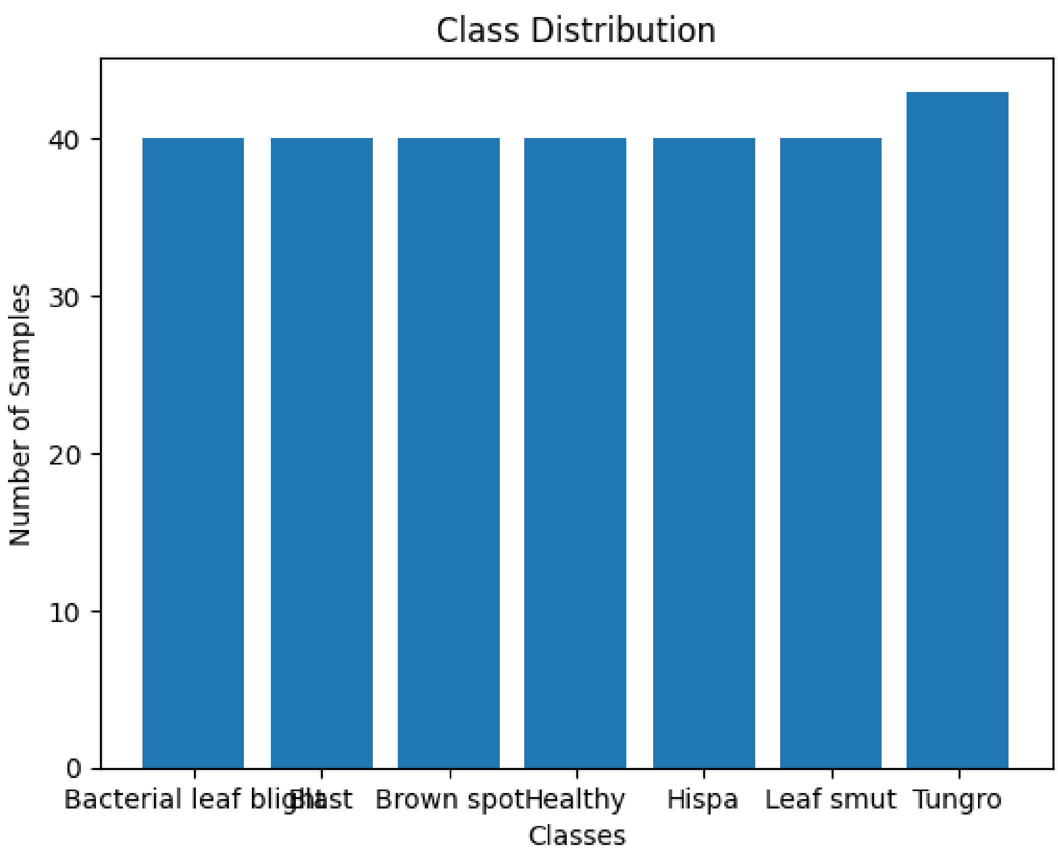
[ ]:
```
class_names = data.class_names
class_names
```

[ ]:
```
['Bacterial leaf blight',
 'Blast',
 'Brown spot',
 'Healthy',
 'Hispa',
 'Leaf smut',
 'Tungro']
```

[ ]:
```
import os
from PIL import Image
import numpy as np

def load_images_from_folder(folder_path, target_size=(224, 224)):
  images = []
  for image_file in os.listdir(folder_path):
    image_path = os.path.join(folder_path, image_file)
    img = Image.open(image_path).resize(target_size)
    images.append(np.array(img))  # Assuming you only need the image data
  return images
images_var1 = load_images_from_folder('D:/Semester 7/Rice_disease_prediction/
 ↪rice_leaf_diseases/Bacterial leaf blight')
images_var2 = load_images_from_folder('D:/Semester 7/Rice_disease_prediction/
 ↪rice_leaf_diseases/Blast')
images_var3 = load_images_from_folder('D:/Semester 7/Rice_disease_prediction/
 ↪rice_leaf_diseases/Brown spot')
images_var4 = load_images_from_folder('D:/Semester 7/Rice_disease_prediction/
 ↪rice_leaf_diseases/Healthy')
images_var5 = load_images_from_folder('D:/Semester 7/Rice_disease_prediction/
 ↪rice_leaf_diseases/Hispa')
images_var6 = load_images_from_folder('D:/Semester 7/Rice_disease_prediction/
 ↪rice_leaf_diseases/Leaf smut')
images_var7 = load_images_from_folder('D:/Semester 7/Rice_disease_prediction/
 ↪rice_leaf_diseases/Tungro')
```

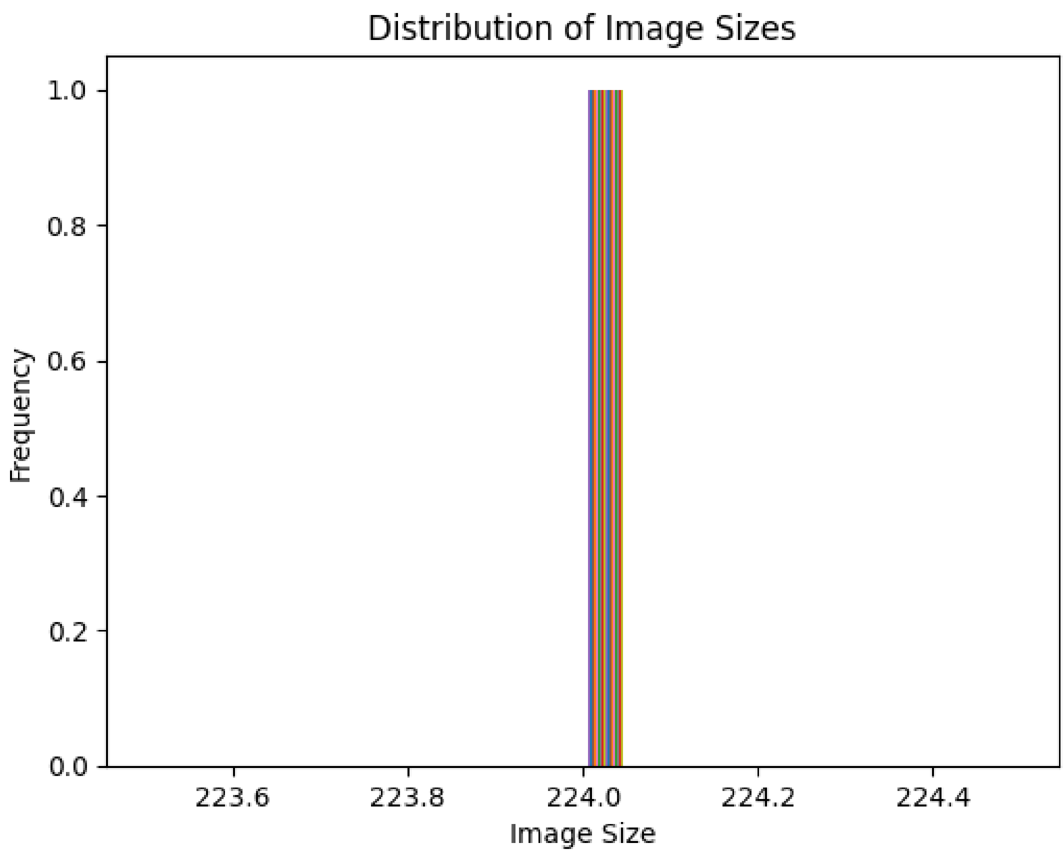
[ ]:
```
class_counts = {class_name: len(images) for class_name, images in {
    'Bacterial leaf blight': images_var1,
    'Blast': images_var2,
```

```
    'Brown spot': images_var3,
    'Healthy': images_var4,
    'Hispa': images_var5,
    'Leaf smut': images_var6,
    'Tungro': images_var7,

}.items()}

plt.bar(class_counts.keys(), class_counts.values())
plt.xlabel('Classes')
plt.ylabel('Number of Samples')
plt.title('Class Distribution')
plt.show()
```




[ ]:
```
image_sizes = [img.shape[:1] for img in images_var1 + images_var2 + images_var3␣
↪+ images_var4 + images_var5 + images_var6 + images_var7]

plt.hist(image_sizes, bins=20)
plt.xlabel('Image Size')
```

```
plt.ylabel('Frequency')
plt.title('Distribution of Image Sizes')
plt.show()
```




[ ]:
```
no_of_samples = 15

dir1_set = images_var1[:no_of_samples]
dir2_set = images_var2[:no_of_samples]
dir3_set = images_var3[:no_of_samples]
dir4_set = images_var4[:no_of_samples]
dir5_set = images_var5[:no_of_samples]
dir6_set = images_var6[:no_of_samples]
dir7_set = images_var7[:no_of_samples]
```

[ ]:
```
fig, axes = plt.subplots(nrows = no_of_samples, ncols = 7, figsize=(20,20))

for i in range(no_of_samples):
    axes[i,0].imshow(dir1_set[i][0])
    axes[i,0].set_title('Bacterial leaf blight')
```

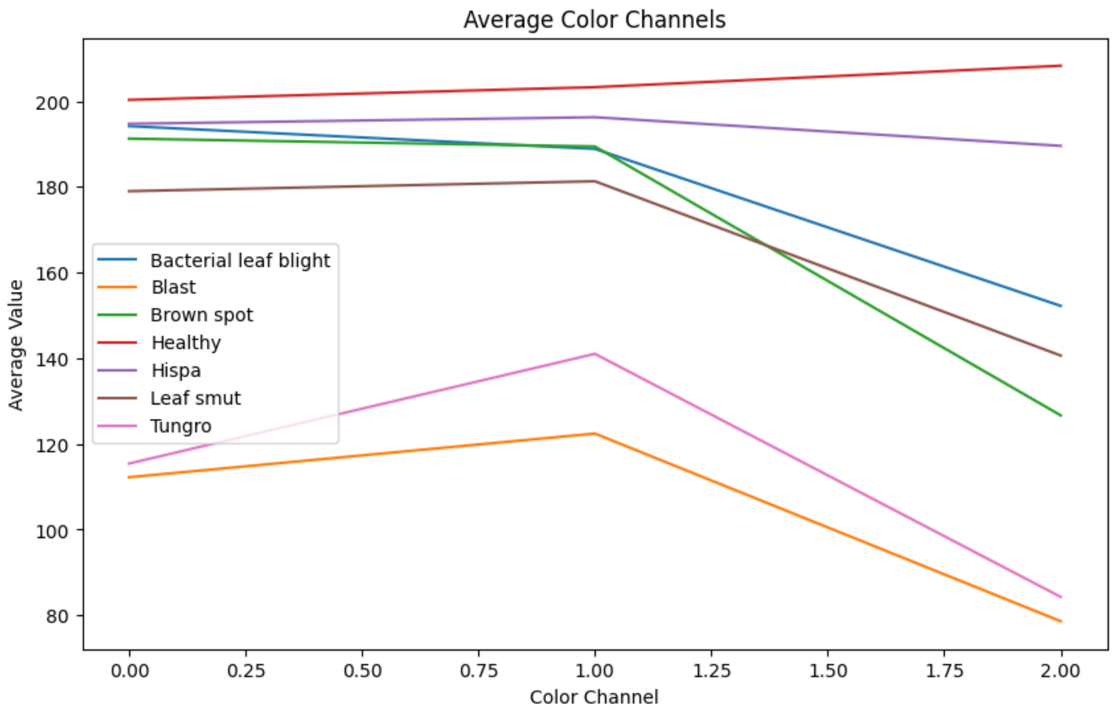
```
    axes[i,0].axis('off')

    axes[i,1].imshow(dir2_set[i][0])
    axes[i,1].set_title('Blast')
    axes[i,1].axis('off')

    axes[i,2].imshow(dir3_set[i][0])
    axes[i,2].set_title('Brown spot')
    axes[i,2].axis('off')

    axes[i,3].imshow(dir4_set[i][0])
    axes[i,3].set_title('Healthy')
    axes[i,3].axis('off')

    axes[i,4].imshow(dir5_set[i][0])
    axes[i,4].set_title('Hispa')
    axes[i,4].axis('off')

    axes[i,5].imshow(dir6_set[i][0])
    axes[i,5].set_title('Leaf smut')
    axes[i,5].axis('off')

    axes[i,6].imshow(dir7_set[i][0])
    axes[i,6].set_title('Tungro')
    axes[i,6].axis('off')

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


[ ]:
```
plt.figure(figsize=(10,10))
for images , labels in data.take(1):
    for i in range(25):
        plt.subplot(5,5,i+1)
        plt.imshow(images[i].numpy().astype('uint8'))
        plt.title(class_names[labels[i]])
        plt.axis('off')
```

```
    plt.tight_layout()
```

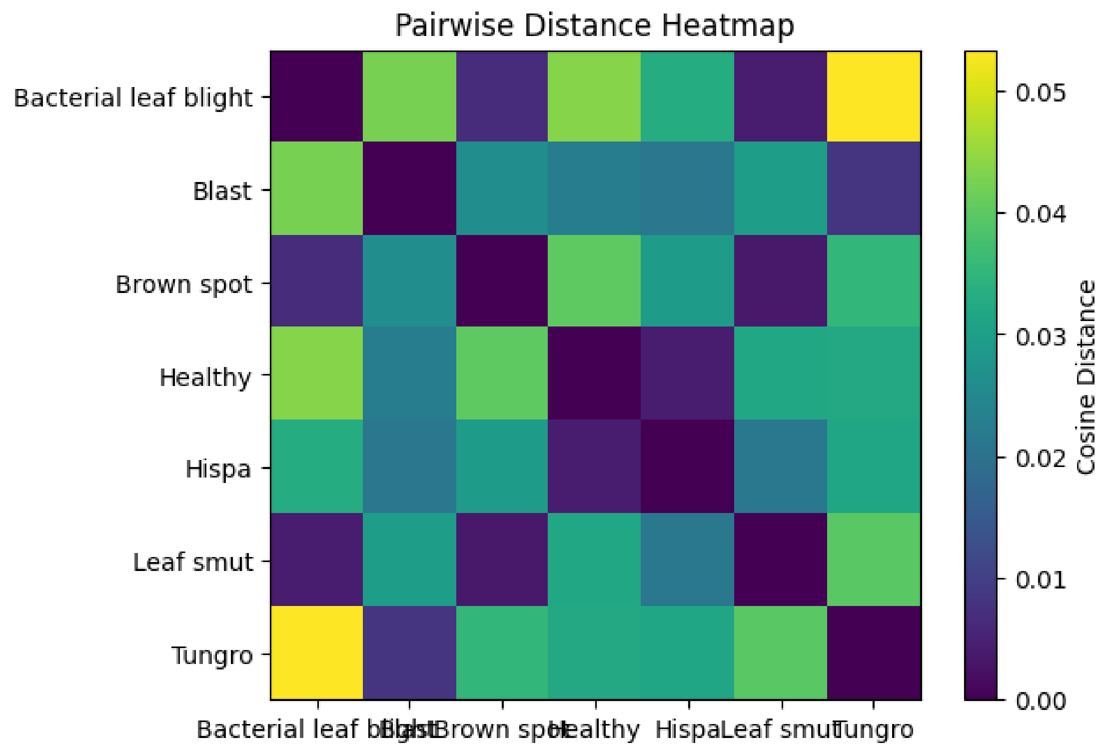

[ ]:
```
# Calculate average color channels for each class
avg_colors = {}
for class_name, images in {
    'Bacterial leaf blight': images_var1,
    'Blast': images_var2,
    'Brown spot': images_var3,
    'Healthy': images_var4,
    'Hispa': images_var5,
    'Leaf smut': images_var6,
    'Tungro':images_var7
```

```
}.items():
    avg_color = np.mean([np.mean(img, axis=(0, 1)) for img in images], axis=0)
    avg_colors[class_name] = avg_color

plt.figure(figsize=(10, 6))
for class_name, avg_color in avg_colors.items():
    plt.plot(avg_color, label=class_name)
plt.xlabel('Color Channel')
plt.ylabel('Average Value')
plt.title('Average Color Channels')
plt.legend()
plt.show()
```



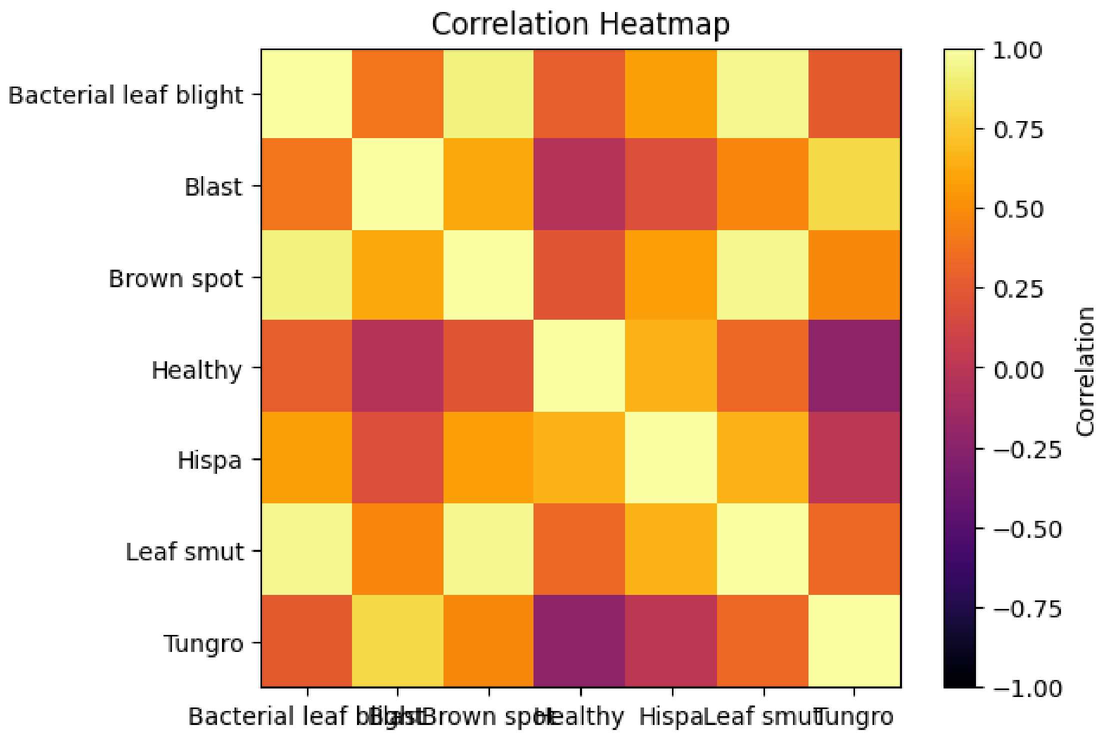

[ ]:
```
from sklearn.metrics.pairwise import pairwise_distances

class_avg_images = [np.mean([img for img in images], axis=0) for images in␣
 ↪[images_var1, images_var2,␣
 ↪images_var3,images_var4,images_var5,images_var6,images_var7]]

class_avg_images_reshaped = [avg_img.reshape(-1) for avg_img in␣
 ↪class_avg_images]
```

```python
distances = pairwise_distances(class_avg_images_reshaped, metric='cosine')

plt.imshow(distances, cmap='viridis', interpolation='nearest')
plt.colorbar(label='Cosine Distance')
plt.xticks(range(7), ['Bacterial leaf blight', 'Blast', 'Brown␣
 ↪spot','Healthy','Hispa','Leaf smut','Tungro'])
plt.yticks(range(7), ['Bacterial leaf blight', 'Blast', 'Brown␣
 ↪spot','Healthy','Hispa','Leaf smut','Tungro'])
plt.title('Pairwise Distance Heatmap')
plt.show()
```




[ ]:
```python
class_avg_images = [np.mean([img for img in images], axis=0) for images in␣
 ↪[images_var1, images_var2,␣
 ↪images_var3,images_var4,images_var5,images_var6,images_var7]]


class_avg_images_flattened = [avg_img.flatten() for avg_img in class_avg_images]

correlations = np.corrcoef(class_avg_images_flattened)

plt.imshow(correlations, cmap='inferno', vmin=-1, vmax=1)
plt.colorbar(label='Correlation')
```

```
plt.xticks(range(7), ['Bacterial leaf blight', 'Blast', 'Brown␣
 ↪spot','Healthy','Hispa','Leaf smut','Tungro'])
plt.yticks(range(7), ['Bacterial leaf blight', 'Blast', 'Brown␣
 ↪spot','Healthy','Hispa','Leaf smut','Tungro'])
plt.title('Correlation Heatmap')
plt.show()
```




[ ]:
```
from tensorflow.keras.preprocessing.image import img_to_array
```

[ ]:
```
x_image = []
y_label = []

for img in images_var1:
    x_image.append(img)
    y_label.append('Bacterial leaf blight')

for img in images_var2:
    x_image.append(img)
    y_label.append('Blast')

for img in images_var3:
    x_image.append(img)
    y_label.append('Brown spot')
```

```
for img in images_var4:
    x_image.append(img)
    y_label.append('Healthy')

for img in images_var5:
    x_image.append(img)
    y_label.append('Hispa')

for img in images_var6:
    x_image.append(img)
    y_label.append('Leaf smut')
for img in images_var7:
    x_image.append(img)
    y_label.append('Tungro')
```

[ ]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
```

[ ]:
```
#converting image and labels to arrays
x_image =np.array(x_image)
y_label = np.array(y_label)

y_encoded = label_encoder.fit_transform(y_label)
```

[ ]:
```
x_image.shape
```

[ ]: (283, 224, 224, 3)

[ ]:
```
x_image = x_image / 255.0
```

[ ]:
```
x_train, x_test, y_train, y_test = train_test_split(x_image, y_encoded,␣
↪test_size=0.2, random_state=42)
```

[ ]:
```
print(x_train.shape)
print(y_train.shape)
```

```
(226, 224, 224, 3)
(226,)
```

[ ]:
```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
```

```
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(label_encoder.classes_), activation='softmax')
])
```

[ ]:
```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[ ]:
```
# model training
# epoch(n) - the training process will iterate through the entire training␣
 ↪dataset n number of times,
            #updating the model's parameters in each iteration.
history = model.fit(x_train, y_train, epochs=30, batch_size=32,␣
 ↪validation_split=0.2)

# testing the model
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_accuracy:.4f}")
```

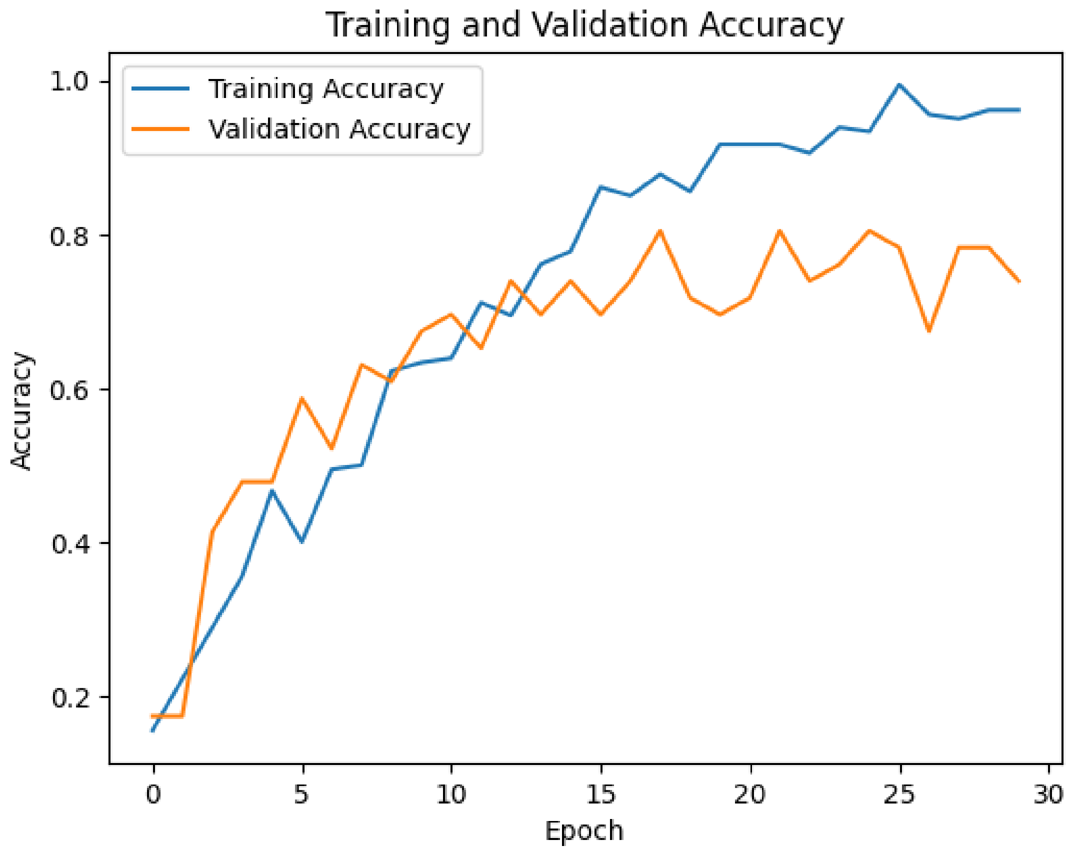
```
Epoch 1/30
6/6               21s 3s/step -
accuracy: 0.1376 - loss: 2.8393 - val_accuracy: 0.1739 - val_loss: 1.9223
Epoch 2/30
6/6               15s 2s/step -
accuracy: 0.2123 - loss: 1.9081 - val_accuracy: 0.1739 - val_loss: 1.8286
Epoch 3/30
6/6               12s 2s/step -
accuracy: 0.2764 - loss: 1.8349 - val_accuracy: 0.4130 - val_loss: 1.6346
Epoch 4/30
6/6               11s 2s/step -
accuracy: 0.3532 - loss: 1.5699 - val_accuracy: 0.4783 - val_loss: 1.2044
Epoch 5/30
6/6               10s 2s/step -
accuracy: 0.4664 - loss: 1.3285 - val_accuracy: 0.4783 - val_loss: 1.0119
Epoch 6/30
6/6               15s 2s/step -
accuracy: 0.3970 - loss: 1.3140 - val_accuracy: 0.5870 - val_loss: 0.9455
Epoch 7/30
6/6               11s 2s/step -
accuracy: 0.4445 - loss: 1.2707 - val_accuracy: 0.5217 - val_loss: 1.0585
Epoch 8/30
6/6               12s 2s/step -
accuracy: 0.4870 - loss: 1.2052 - val_accuracy: 0.6304 - val_loss: 0.9286
Epoch 9/30
6/6               12s 2s/step -
accuracy: 0.6313 - loss: 0.9750 - val_accuracy: 0.6087 - val_loss: 0.8269
Epoch 10/30
```

```
6/6             12s 2s/step -
accuracy: 0.6374 - loss: 0.8675 - val_accuracy: 0.6739 - val_loss: 0.7557
Epoch 11/30
6/6             15s 2s/step -
accuracy: 0.6385 - loss: 0.8194 - val_accuracy: 0.6957 - val_loss: 0.7590
Epoch 12/30
6/6             17s 2s/step -
accuracy: 0.6816 - loss: 0.8347 - val_accuracy: 0.6522 - val_loss: 0.8535
Epoch 13/30
6/6             12s 2s/step -
accuracy: 0.6839 - loss: 0.7634 - val_accuracy: 0.7391 - val_loss: 0.9242
Epoch 14/30
6/6             11s 2s/step -
accuracy: 0.7576 - loss: 0.7178 - val_accuracy: 0.6957 - val_loss: 0.8470
Epoch 15/30
6/6             11s 2s/step -
accuracy: 0.7787 - loss: 0.6150 - val_accuracy: 0.7391 - val_loss: 0.7946
Epoch 16/30
6/6             12s 2s/step -
accuracy: 0.8268 - loss: 0.4543 - val_accuracy: 0.6957 - val_loss: 1.0086
Epoch 17/30
6/6             12s 2s/step -
accuracy: 0.8597 - loss: 0.4011 - val_accuracy: 0.7391 - val_loss: 0.8304
Epoch 18/30
6/6             12s 2s/step -
accuracy: 0.8556 - loss: 0.3903 - val_accuracy: 0.8043 - val_loss: 0.7951
Epoch 19/30
6/6             11s 2s/step -
accuracy: 0.8411 - loss: 0.3288 - val_accuracy: 0.7174 - val_loss: 0.9389
Epoch 20/30
6/6             10s 2s/step -
accuracy: 0.9097 - loss: 0.2378 - val_accuracy: 0.6957 - val_loss: 0.9617
Epoch 21/30
6/6             11s 2s/step -
accuracy: 0.9211 - loss: 0.2384 - val_accuracy: 0.7174 - val_loss: 1.0041
Epoch 22/30
6/6             12s 2s/step -
accuracy: 0.9042 - loss: 0.2570 - val_accuracy: 0.8043 - val_loss: 1.1522
Epoch 23/30
6/6             11s 2s/step -
accuracy: 0.8787 - loss: 0.3114 - val_accuracy: 0.7391 - val_loss: 1.1293
Epoch 24/30
6/6             12s 2s/step -
accuracy: 0.9406 - loss: 0.1855 - val_accuracy: 0.7609 - val_loss: 1.1703
Epoch 25/30
6/6             13s 2s/step -
accuracy: 0.9103 - loss: 0.2283 - val_accuracy: 0.8043 - val_loss: 0.7821
Epoch 26/30
```

```
6/6              13s 2s/step -
accuracy: 0.9927 - loss: 0.1298 - val_accuracy: 0.7826 - val_loss: 0.8941
Epoch 27/30
6/6              12s 2s/step -
accuracy: 0.9612 - loss: 0.1587 - val_accuracy: 0.6739 - val_loss: 1.4524
Epoch 28/30
6/6              11s 2s/step -
accuracy: 0.9334 - loss: 0.2641 - val_accuracy: 0.7826 - val_loss: 0.9848
Epoch 29/30
6/6              14s 3s/step -
accuracy: 0.9618 - loss: 0.1141 - val_accuracy: 0.7826 - val_loss: 1.1706
Epoch 30/30
6/6              16s 3s/step -
accuracy: 0.9647 - loss: 0.0868 - val_accuracy: 0.7391 - val_loss: 1.3424
2/2              1s 383ms/step -
accuracy: 0.7750 - loss: 0.8201
Test accuracy: 0.7719
```

[ ]:

```
# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()

predictions = model.predict(x_test)
predicted_labels = np.argmax(predictions, axis=1)

predicted_class_labels = label_encoder.inverse_transform(predicted_labels)

plt.figure(figsize=(12, 12))
for i in range(15):
    plt.subplot(3, 5, i + 1)
    plt.imshow(x_test[i])
    plt.title(predicted_class_labels[i])
    plt.axis('off')
plt.tight_layout()
plt.show()
```




```
2/2                    1s 402ms/step
```


[ ]:
```
#hyperparameter tuning -  LEARNING RATE
learning_rate = 0.0001
```

[ ]:
```
model_hp = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(label_encoder.classes_), activation='softmax')
])
```

[ ]:
```
optimizer = Adam(learning_rate=learning_rate)
```

[ ]:
```
model_hp.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',␣
↪metrics=['accuracy'])
```

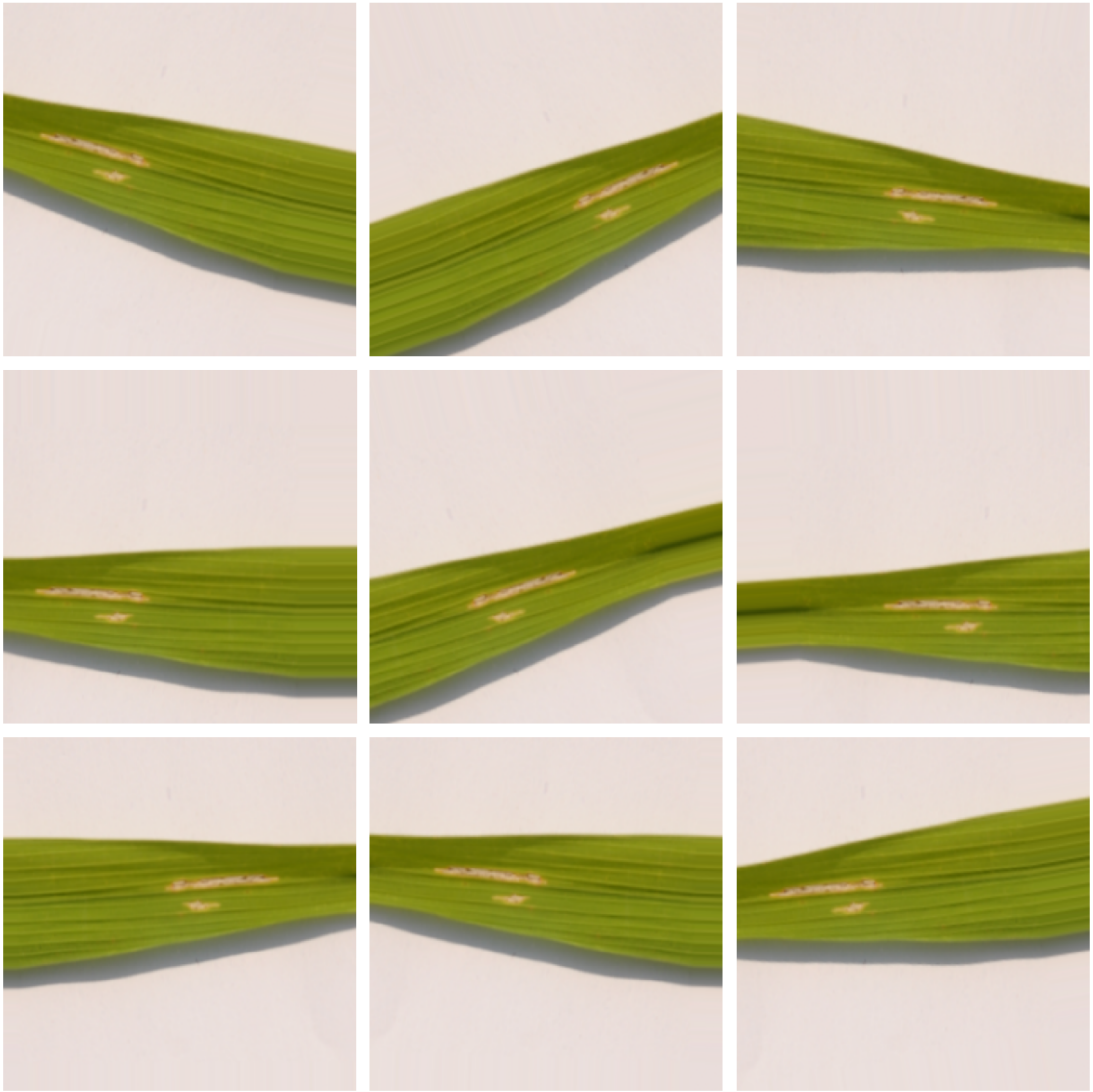
[ ]:
```
history = model_hp.fit(x_train, y_train, epochs=30, batch_size=32,␣
↪validation_split=0.2)

test_loss, test_accuracy = model_hp.evaluate(x_test, y_test)
print(f"Test accuracy: {test_accuracy:.4f}")
```

```
Epoch 1/30
6/6               26s 3s/step -
accuracy: 0.2003 - loss: 1.9097 - val_accuracy: 0.2391 - val_loss: 1.9056
Epoch 2/30
6/6               22s 4s/step -
accuracy: 0.1848 - loss: 1.8884 - val_accuracy: 0.2391 - val_loss: 1.9040
Epoch 3/30
6/6               17s 3s/step -
accuracy: 0.2142 - loss: 1.8917 - val_accuracy: 0.2391 - val_loss: 1.9030
Epoch 4/30
6/6               16s 3s/step -
accuracy: 0.2188 - loss: 1.8870 - val_accuracy: 0.2609 - val_loss: 1.9026
Epoch 5/30
6/6               38s 5s/step -
accuracy: 0.2305 - loss: 1.8885 - val_accuracy: 0.2391 - val_loss: 1.9016
Epoch 6/30
6/6               22s 4s/step -
accuracy: 0.2367 - loss: 1.8916 - val_accuracy: 0.2609 - val_loss: 1.9002
Epoch 7/30
6/6               30s 5s/step -
accuracy: 0.1738 - loss: 1.8985 - val_accuracy: 0.2391 - val_loss: 1.8989
Epoch 8/30
6/6               33s 5s/step -
accuracy: 0.2465 - loss: 1.9003 - val_accuracy: 0.2174 - val_loss: 1.8986
Epoch 9/30
6/6               21s 3s/step -
accuracy: 0.2468 - loss: 1.8792 - val_accuracy: 0.2174 - val_loss: 1.8968
Epoch 10/30
6/6               29s 5s/step -
accuracy: 0.1959 - loss: 1.8989 - val_accuracy: 0.2174 - val_loss: 1.8951
Epoch 11/30
6/6               34s 3s/step -
accuracy: 0.1994 - loss: 1.8844 - val_accuracy: 0.2391 - val_loss: 1.8933
Epoch 12/30
6/6               47s 4s/step -
accuracy: 0.2151 - loss: 1.8971 - val_accuracy: 0.2391 - val_loss: 1.8915
Epoch 13/30
6/6               21s 3s/step -
accuracy: 0.2139 - loss: 1.8780 - val_accuracy: 0.2391 - val_loss: 1.8897
```

```
Epoch 14/30
6/6            35s 6s/step -
accuracy: 0.2129 - loss: 1.8808 - val_accuracy: 0.2391 - val_loss: 1.8881
Epoch 15/30
6/6            26s 4s/step -
accuracy: 0.2072 - loss: 1.8820 - val_accuracy: 0.2391 - val_loss: 1.8872
Epoch 16/30
6/6            23s 4s/step -
accuracy: 0.2157 - loss: 1.8781 - val_accuracy: 0.2391 - val_loss: 1.8849
Epoch 17/30
6/6            37s 6s/step -
accuracy: 0.2374 - loss: 1.8869 - val_accuracy: 0.2391 - val_loss: 1.8831
Epoch 18/30
6/6            20s 3s/step -
accuracy: 0.3083 - loss: 1.8496 - val_accuracy: 0.2391 - val_loss: 1.8821
Epoch 19/30
6/6            19s 3s/step -
accuracy: 0.2068 - loss: 1.8797 - val_accuracy: 0.2391 - val_loss: 1.8807
Epoch 20/30
6/6            26s 4s/step -
accuracy: 0.3037 - loss: 1.8755 - val_accuracy: 0.2391 - val_loss: 1.8795
Epoch 21/30
6/6            50s 6s/step -
accuracy: 0.2629 - loss: 1.8660 - val_accuracy: 0.2391 - val_loss: 1.8784
Epoch 22/30
6/6            26s 4s/step -
accuracy: 0.2435 - loss: 1.8720 - val_accuracy: 0.2391 - val_loss: 1.8751
Epoch 23/30
6/6            34s 6s/step -
accuracy: 0.2695 - loss: 1.8681 - val_accuracy: 0.2391 - val_loss: 1.8734
Epoch 24/30
6/6            27s 4s/step -
accuracy: 0.2189 - loss: 1.8629 - val_accuracy: 0.3043 - val_loss: 1.8708
Epoch 25/30
6/6            17s 3s/step -
accuracy: 0.2431 - loss: 1.8818 - val_accuracy: 0.3043 - val_loss: 1.8704
Epoch 26/30
6/6            28s 5s/step -
accuracy: 0.2857 - loss: 1.8370 - val_accuracy: 0.2826 - val_loss: 1.8682
Epoch 27/30
6/6            40s 4s/step -
accuracy: 0.2607 - loss: 1.8726 - val_accuracy: 0.2391 - val_loss: 1.8676
Epoch 28/30
6/6            16s 3s/step -
accuracy: 0.2272 - loss: 1.8662 - val_accuracy: 0.2826 - val_loss: 1.8659
Epoch 29/30
6/6            19s 3s/step -
accuracy: 0.2736 - loss: 1.8458 - val_accuracy: 0.3043 - val_loss: 1.8646
```

```
Epoch 30/30
6/6              26s 4s/step -
accuracy: 0.3196 - loss: 1.8558 - val_accuracy: 0.3043 - val_loss: 1.8628
2/2              1s 404ms/step -
accuracy: 0.2379 - loss: 1.9021
Test accuracy: 0.2632
```

[ ]:
```
#Visualizing Augmented Images:
# we visualize augmented images to ensure they still represent the original␣
↪classes. this has to be done in
    #data visualisation section

from tensorflow.keras.preprocessing.image import ImageDataGenerator

data_generator = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True
)

sample_image = x_image[2]
sample_image = np.expand_dims(sample_image, axis=0)

plt.figure(figsize=(12, 12))
for i, augmented_image in enumerate(data_generator.flow(sample_image,␣
↪batch_size=1)):
    plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_image[0])
    plt.axis('off')
    if i == 8:
        break
plt.tight_layout()
plt.show()
```



```python
[ ]: def create_model(input_shape, num_classes):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')  # Output layer
    ])
```

```
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
↪metrics=['accuracy'])
    return model
```

[ ]:
```
datagen = ImageDataGenerator(
    rotation_range=45,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
#roation_angle - an angle where the image is rotated and the algorithm is run.␣
↪it goes from -45 to +45(exampe)
#width& height_shift_range - randomly shifting the image horizontally and␣
↪vertically. This helps the model tolerate small translations in the input␣
↪data.
#shear_range - maximum range of shear on the axis
#fill_mode - "nearest," "constant," "reflect," and "wrap."
```

[ ]:
```
input_shape = (224, 224, 3)
num_classes = len(label_encoder.classes_)
model_da = create_model(input_shape, num_classes)
```

[ ]:
```
augmented_data = datagen.flow(x_train, y_train, batch_size=32)
```

[ ]:
```
history = model_da.fit(augmented_data, epochs=30, steps_per_epoch=len(x_train) /
↪/ 32, validation_data=(x_test, y_test))
```

```
Epoch 1/30
7/7               27s 2s/step -
accuracy: 0.1247 - loss: 3.0801 - val_accuracy: 0.1228 - val_loss: 1.9385
Epoch 2/30
7/7               4s 249ms/step -
accuracy: 0.1250 - loss: 1.1169 - val_accuracy: 0.1404 - val_loss: 1.9279
Epoch 3/30
7/7               26s 3s/step -
accuracy: 0.2006 - loss: 1.9441 - val_accuracy: 0.2632 - val_loss: 1.8785
Epoch 4/30
7/7               4s 191ms/step -
accuracy: 0.0938 - loss: 1.0753 - val_accuracy: 0.2982 - val_loss: 1.8492
Epoch 5/30
7/7               20s 2s/step -
accuracy: 0.2896 - loss: 1.8173 - val_accuracy: 0.4561 - val_loss: 1.5520
Epoch 6/30
7/7               4s 181ms/step -
accuracy: 0.3750 - loss: 0.9106 - val_accuracy: 0.4561 - val_loss: 1.5118
```

```
Epoch 7/30
7/7            22s 2s/step -
accuracy: 0.3630 - loss: 1.5580 - val_accuracy: 0.2456 - val_loss: 1.4993
Epoch 8/30
7/7            4s 198ms/step -
accuracy: 0.2500 - loss: 0.8849 - val_accuracy: 0.4035 - val_loss: 1.3183
Epoch 9/30
7/7            24s 2s/step -
accuracy: 0.3731 - loss: 1.5275 - val_accuracy: 0.4737 - val_loss: 1.2333
Epoch 10/30
7/7            4s 219ms/step -
accuracy: 0.3750 - loss: 0.8069 - val_accuracy: 0.4386 - val_loss: 1.2256
Epoch 11/30
7/7            22s 2s/step -
accuracy: 0.4385 - loss: 1.3549 - val_accuracy: 0.5263 - val_loss: 1.0547
Epoch 12/30
7/7            4s 199ms/step -
accuracy: 0.4688 - loss: 0.7837 - val_accuracy: 0.4386 - val_loss: 1.1389
Epoch 13/30
7/7            22s 3s/step -
accuracy: 0.2563 - loss: 1.6131 - val_accuracy: 0.6140 - val_loss: 0.9837
Epoch 14/30
7/7            5s 218ms/step -
accuracy: 0.5312 - loss: 0.5775 - val_accuracy: 0.5439 - val_loss: 0.9375
Epoch 15/30
7/7            25s 2s/step -
accuracy: 0.5107 - loss: 1.7270 - val_accuracy: 0.2632 - val_loss: 1.4698
Epoch 16/30
7/7            4s 220ms/step -
accuracy: 0.3125 - loss: 0.8240 - val_accuracy: 0.2982 - val_loss: 1.4548
Epoch 17/30
7/7            21s 2s/step -
accuracy: 0.3413 - loss: 1.4627 - val_accuracy: 0.1754 - val_loss: 1.4600
Epoch 18/30
7/7            5s 312ms/step -
accuracy: 0.3125 - loss: 0.8353 - val_accuracy: 0.2105 - val_loss: 1.4258
Epoch 19/30
7/7            26s 2s/step -
accuracy: 0.3192 - loss: 1.4778 - val_accuracy: 0.3860 - val_loss: 1.3449
Epoch 20/30
7/7            3s 162ms/step -
accuracy: 0.3438 - loss: 0.8702 - val_accuracy: 0.4035 - val_loss: 1.2972
Epoch 21/30
7/7            24s 3s/step -
accuracy: 0.3675 - loss: 1.4725 - val_accuracy: 0.4561 - val_loss: 1.3380
Epoch 22/30
7/7            5s 186ms/step -
accuracy: 0.4062 - loss: 0.8841 - val_accuracy: 0.4912 - val_loss: 1.2935
```

```
Epoch 23/30
7/7            19s 2s/step -
accuracy: 0.3702 - loss: 1.2566 - val_accuracy: 0.4561 - val_loss: 1.1606
Epoch 24/30
7/7            3s 218ms/step -
accuracy: 0.4375 - loss: 0.7845 - val_accuracy: 0.4561 - val_loss: 1.1915
Epoch 25/30
7/7            27s 3s/step -
accuracy: 0.4039 - loss: 1.3258 - val_accuracy: 0.5263 - val_loss: 1.1115
Epoch 26/30
7/7            4s 306ms/step -
accuracy: 0.4375 - loss: 0.7952 - val_accuracy: 0.5439 - val_loss: 1.0299
Epoch 27/30
7/7            23s 2s/step -
accuracy: 0.4953 - loss: 1.2346 - val_accuracy: 0.5263 - val_loss: 0.9682
Epoch 28/30
7/7            3s 147ms/step -
accuracy: 0.3125 - loss: 0.7997 - val_accuracy: 0.5439 - val_loss: 0.9760
Epoch 29/30
7/7            20s 2s/step -
accuracy: 0.4685 - loss: 1.1645 - val_accuracy: 0.6316 - val_loss: 0.9080
Epoch 30/30
7/7            4s 221ms/step -
accuracy: 0.5625 - loss: 0.6110 - val_accuracy: 0.5789 - val_loss: 0.9274
```