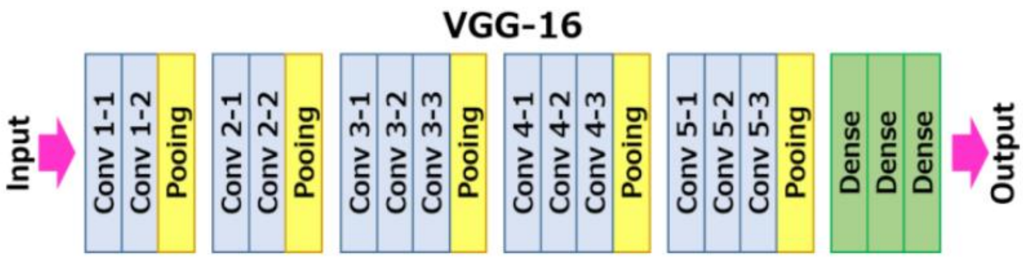
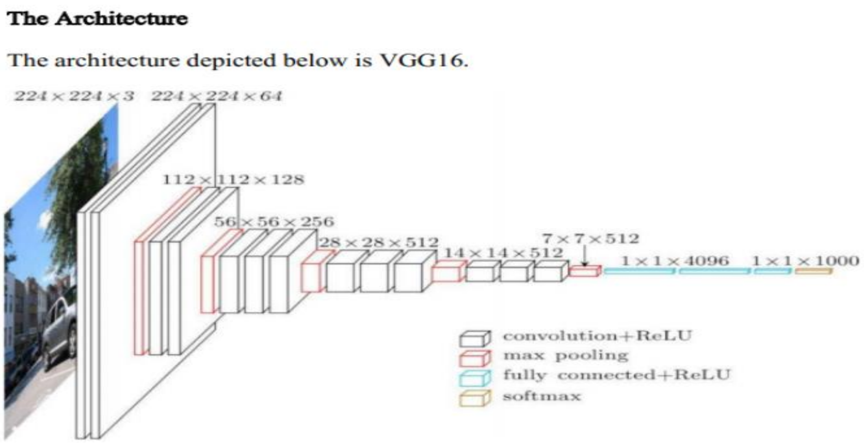
VGG16: A Deep Convolutional Neural Network for Image Recognition.

VGG16, introduced in 2014, is a convolutional neural network (CNN) architecture that played a significant role in advancing the field of computer vision.



The Architecture




```
import os
import random

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import cv2
from cv2 import resize
from glob import glob
from tqdm import tqdm

import tensorflow as tf
import keras
from keras.callbacks import EarlyStopping,ModelCheckpoint
from sklearn.metrics import confusion_matrix , accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import La#Create Files_Name
```

```
image_data='/kaggle/input/smart-grid-phasor-measurement-unit-faulty-data/IMG_PMU_DATA_NT_VF_001'
pd.DataFrame(os.listdir(image_data),columns=['Files_Name'])
```

	Files_Name
0	ISGT_2020_data_metadata.xlsx
1	DB_SMS
2	Read Me.rtf
3	DB_GNL
4	DB_FLT

```
img_height = 244
img_width = 244
train_ds = tf.keras.utils.image_dataset_from_directory(
  '/kaggle/input/smart-grid-phasor-measurement-unit-faulty-data/IMG_PMU_DATA_NT_VF_001',
  validation_split=0.2,
  subset='training',
  image_size=(img_height, img_width),
  batch_size=32,
  seed=42,
  shuffle=True)

val_ds = tf.keras.utils.image_dataset_from_directory(
  '/kaggle/input/smart-grid-phasor-measurement-unit-faulty-data/IMG_PMU_DATA_NT_VF_001',
  validation_split=0.2,
  subset='validation',
  image_size=(img_height, img_width),
  batch_size=32,
  seed=42,
  shuffle=True)
```

```
Found 404 files belonging to 3 classes.
Using 324 files for training.
Found 404 files belonging to 3 classes.
Using 80 files for validation.
```

```
class_names = train_ds.class_names
print(class_names)
```

```
['DB_FLT', 'DB_GNL', 'DB_SMS']
```

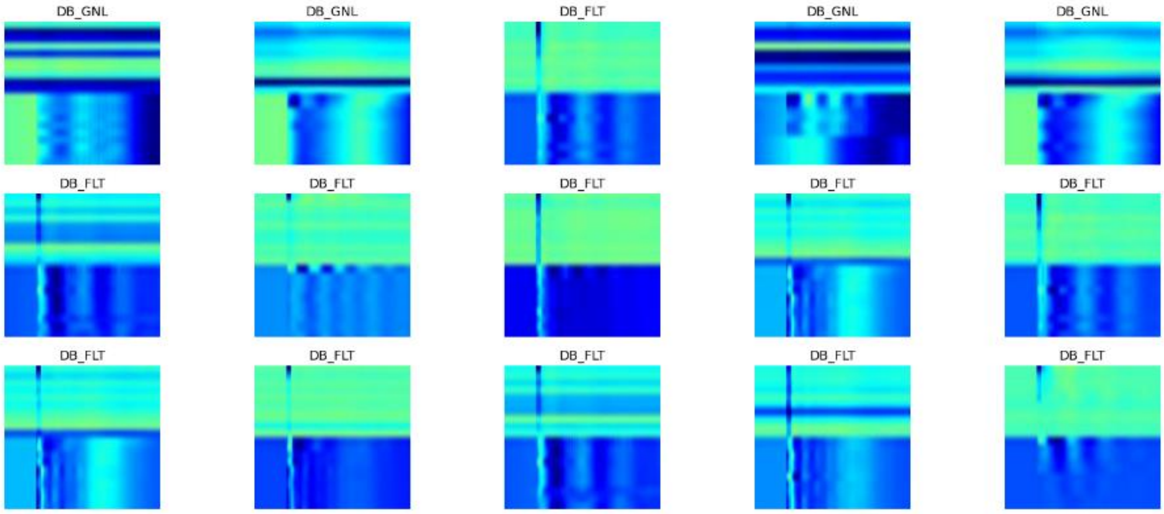
Phasor measurement units (PMUs) Description

DB_FLT: This folder contains 344 images representing faults in the power grid, like short circuits.

DB_GNL: This folder contains 140 images representing a loss of generation, where a power plant goes offline.

DB_SMS: This folder contains 21 images representing synchronous motor switching events, which are changes in how a large motor is connected to the grid.

```python
plt.figure(figsize=(20, 15))
for images, labels in train_ds.take(1):
    for i in range(25):
        ax = plt.subplot(5, 5, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
```



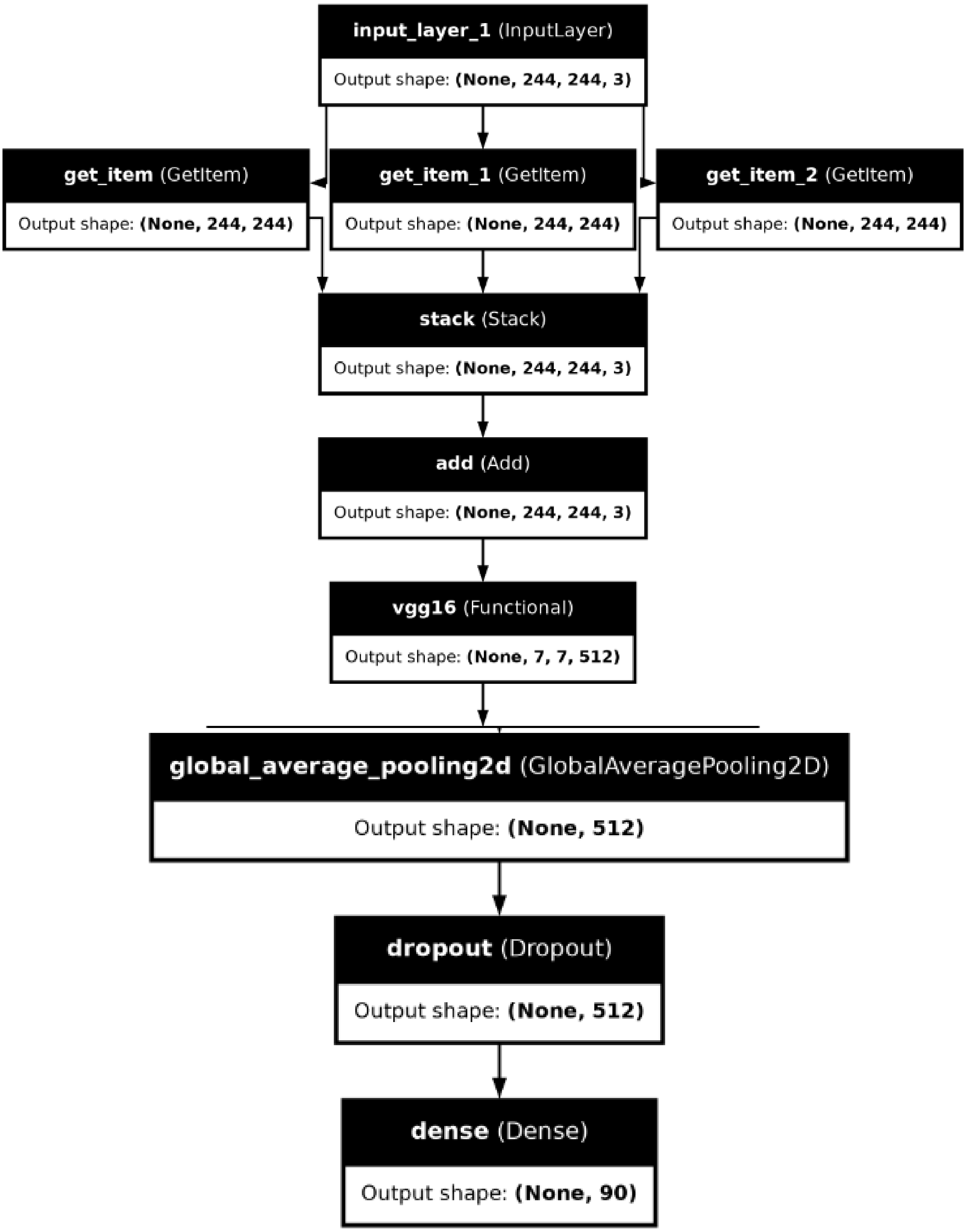

```python
base_model = tf.keras.applications.VGG16(
    include_top=False,
    weights='imagenet',
    input_shape=(img_height, img_width, 3)
)
base_model.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```python
inputs = tf.keras.Input(shape=(img_height, img_width, 3))
x = tf.keras.applications.vgg16.preprocess_input(inputs)
x = base_model(x, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dropout(0.3)(x)
outputs = tf.keras.layers.Dense(90)(x)
model = tf.keras.Model(inputs, outputs)

model.summary()
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 244, 244, 3)	0	-
get_item (GetItem)	(None, 244, 244)	0	input_layer_1[0]…
get_item_1 (GetItem)	(None, 244, 244)	0	input_layer_1[0]…
get_item_2 (GetItem)	(None, 244, 244)	0	input_layer_1[0]…
stack (Stack)	(None, 244, 244, 3)	0	get_item[0][0], get_item_1[0][0], get_item_2[0][0]
add (Add)	(None, 244, 244, 3)	0	stack[0][0]
vgg16 (Functional)	(None, 7, 7, 512)	14,714,688	add[0][0]
global_average_poo… (GlobalAveragePool…	(None, 512)	0	vgg16[0][0]
dropout (Dropout)	(None, 512)	0	global_average_p…
dense (Dense)	(None, 90)	46,170	dropout[0][0]

```python
import tensorflow as tf  # Assuming you're using TensorFlow

from tensorflow.keras.utils import plot_model

# Assuming you have your Keras model defined as 'model'

plot_model(model, to_file='cnn_plot.png', show_shapes=True, show_layer_names=True)
```



```
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=T
rue),
              metrics=['accuracy'])

epoch = 20
model.fit(train_ds, validation_data=val_ds, epochs=epoch,
    callbacks = [
        tf.keras.callbacks.EarlyStopping(
            monitor="val_loss",
            min_delta=1e-2,
            patience=3,
            verbose=1,
            restore_best_weights=True
        )
    ]
)
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 66s 3s/step - accuracy: 0.0113 - loss: 10.9388 - val_accuracy: 0.4625 - val_loss: 1.7720
Epoch 2/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 185ms/step - accuracy: 0.3959 - loss: 2.5831 - val_accuracy: 0.6125 - val_loss: 0.9492
Epoch 3/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 180ms/step - accuracy: 0.6701 - loss: 1.3291 - val_accuracy: 0.8000 - val_loss: 0.6836
Epoch 4/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 182ms/step - accuracy: 0.7834 - loss: 1.0465 - val_accuracy: 0.8625 - val_loss: 0.3837
Epoch 5/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 182ms/step - accuracy: 0.7787 - loss: 0.7489 - val_accuracy: 0.8500 - val_loss: 0.3495
Epoch 6/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 183ms/step - accuracy: 0.8444 - loss: 0.5833 - val_accuracy: 0.8625 - val_loss: 0.3013
Epoch 7/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 182ms/step - accuracy: 0.8447 - loss: 0.4677 - val_accuracy: 0.9250 - val_loss: 0.2537
Epoch 8/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 182ms/step - accuracy: 0.9116 - loss: 0.3531 - val_accuracy: 0.9250 - val_loss: 0.2251
Epoch 9/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 184ms/step - accuracy: 0.8742 - loss: 0.4357 - val_accuracy: 0.9125 - val_loss: 0.1961
Epoch 10/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 185ms/step - accuracy: 0.8750 - loss: 0.3648 - val_accuracy: 0.9250 - val_loss: 0.1615
Epoch 11/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 180ms/step - accuracy: 0.9113 - loss: 0.2477 - val_accuracy: 0.9375 - val_loss: 0.1531
Epoch 12/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 180ms/step - accuracy: 0.9020 - loss: 0.2983 - val_accuracy: 0.9375 - val_loss: 0.1604
Epoch 13/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 184ms/step - accuracy: 0.9246 - loss: 0.2345 - val_accuracy: 0.9375 - val_loss: 0.1258
Epoch 14/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 182ms/step - accuracy: 0.9347 - loss: 0.2649 - val_accuracy: 0.9500 - val_loss: 0.1888
Epoch 15/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 182ms/step - accuracy: 0.9299 - loss: 0.2288 - val_accuracy: 0.9500 - val_loss: 0.1301
Epoch 16/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 186ms/step - accuracy: 0.9495 - loss: 0.1595 - val_accuracy: 0.9625 - val_loss: 0.1123
Epoch 17/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 184ms/step - accuracy: 0.9084 - loss: 0.2248 - val_accuracy: 0.9750 - val_loss: 0.1048
Epoch 18/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 186ms/step - accuracy: 0.9287 - loss: 0.2830 - val_accuracy: 0.9625 - val_loss: 0.1176
Epoch 19/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 180ms/step - accuracy: 0.9412 - loss: 0.1884 - val_accuracy: 0.9875 - val_loss: 0.0913
Epoch 20/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 186ms/step - accuracy: 0.9284 - loss: 0.1375 - val_accuracy: 0.9750 - val_loss: 0.0856
Restoring model weights from the end of the best epoch: 19.
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=T
rue),
              metrics=['accuracy'])
```

```python
# fine tuning
base_model.trainable = True
for layer in base_model.layers[:14]:
    layer.trainable = False
model.summary()
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 244, 244, 3)	0	-
get_item (GetItem)	(None, 244, 244)	0	input_layer_1[0]…
get_item_1 (GetItem)	(None, 244, 244)	0	input_layer_1[0]…
get_item_2 (GetItem)	(None, 244, 244)	0	input_layer_1[0]…
stack (Stack)	(None, 244, 244, 3)	0	get_item[0][0], get_item_1[0][0], get_item_2[0][0]
add (Add)	(None, 244, 244, 3)	0	stack[0][0]
vgg16 (Functional)	(None, 7, 7, 512)	14,714,688	add[0][0]
global_average_poo… (GlobalAveragePool…	(None, 512)	0	vgg16[0][0]
dropout (Dropout)	(None, 512)	0	global_average_p…
dense (Dense)	(None, 90)	46,170	dropout[0][0]

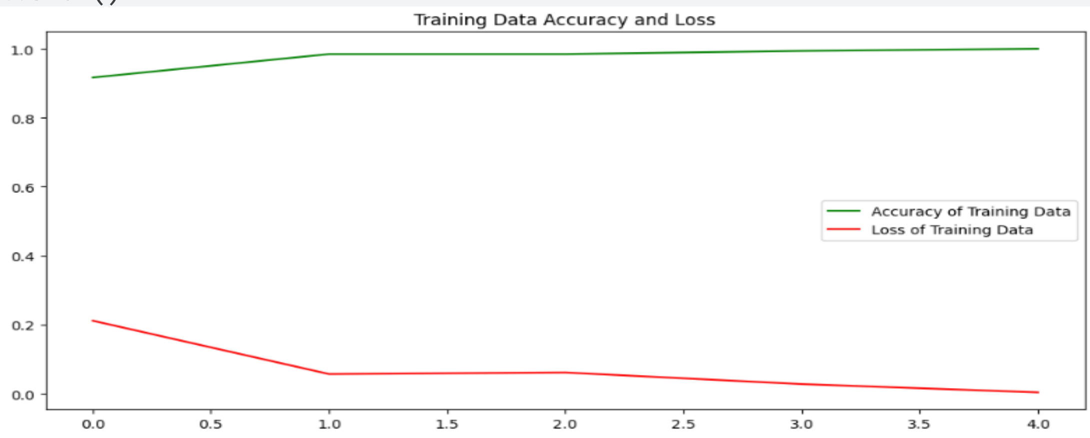
```python
model.compile(optimizer=tf.keras.optimizers.Adam(0.0001),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
epoch = 20
history = model.fit(train_ds, validation_data=val_ds, epochs=epoch,
    callbacks = [
        tf.keras.callbacks.EarlyStopping(
            monitor="val_loss",
            min_delta=1e-2,
            patience=3,
            verbose=1,
        )
    ]
)
```

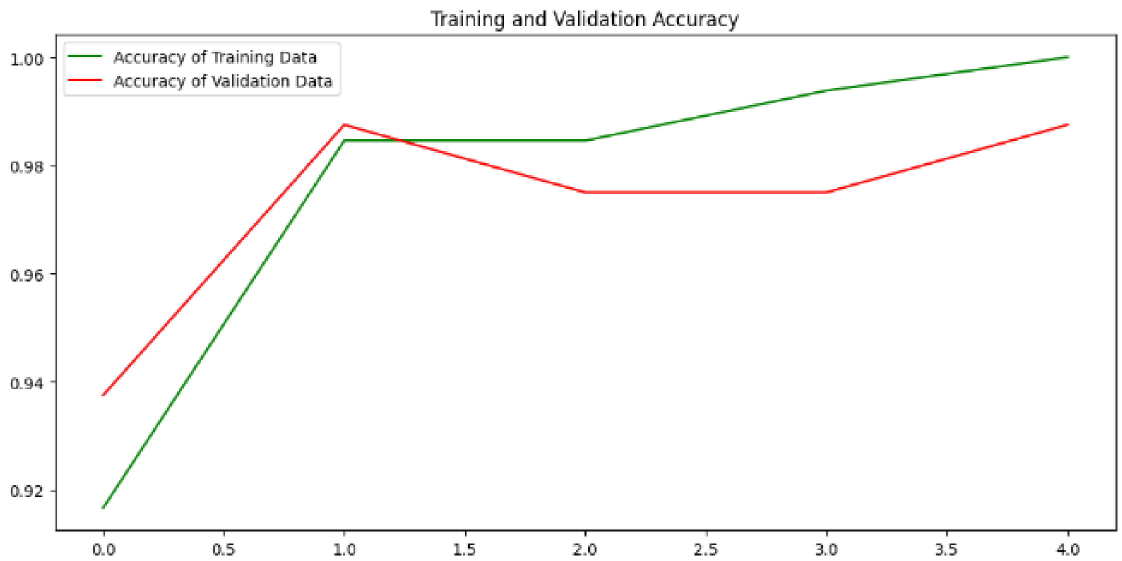
```
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 491ms/step - accuracy: 0.8948 - loss: 0.2968

W0000 00:00:1713167725.865202      77 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update

11/11 ━━━━━━━━━━━━━━━━━━━━ 14s 660ms/step - accuracy: 0.8967 - loss: 0.2896 - val_accuracy: 0.9375 - val_loss: 0.1423
Epoch 2/20

W0000 00:00:1713167726.698342      80 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update

11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 205ms/step - accuracy: 0.9657 - loss: 0.0902 - val_accuracy: 0.9875 - val_loss: 0.0468
Epoch 3/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 203ms/step - accuracy: 0.9851 - loss: 0.0533 - val_accuracy: 0.9750 - val_loss: 0.0639
Epoch 4/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 202ms/step - accuracy: 0.9924 - loss: 0.0323 - val_accuracy: 0.9750 - val_loss: 0.0576
Epoch 5/20
11/11 ━━━━━━━━━━━━━━━━━━━━ 2s 202ms/step - accuracy: 1.0000 - loss: 0.0031 - val_accuracy: 0.9875 - val_loss: 0.0864
Epoch 5: early stopping
```

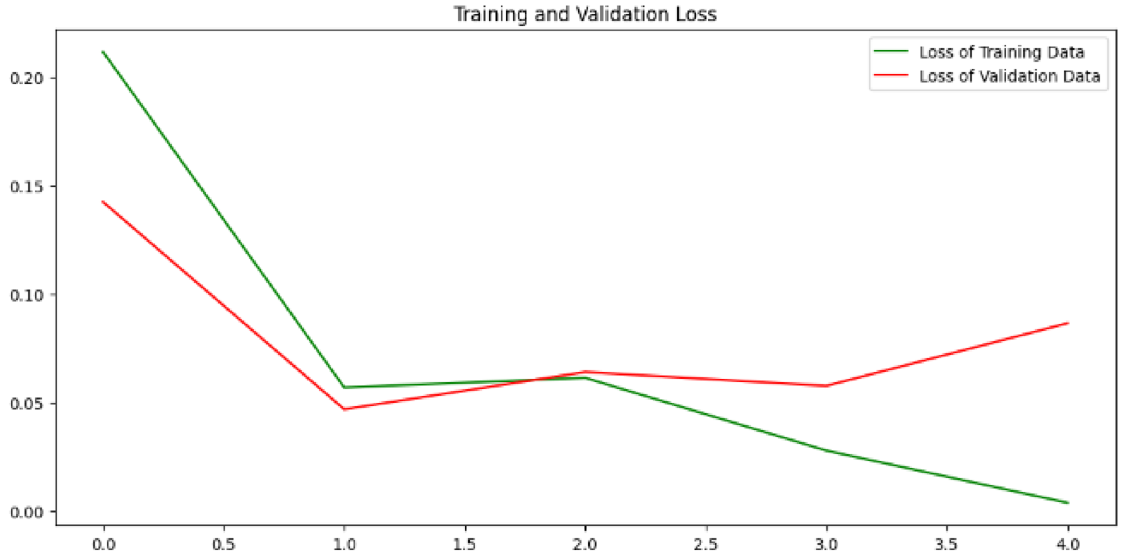
```
hist_=pd.DataFrame(history.history)
hist_
```

	accuracy	loss	val_accuracy	val_loss
0	0.916667	0.211469	0.9375	0.142317
1	0.984568	0.056811	0.9875	0.046768
2	0.984568	0.061251	0.9750	0.063935
3	0.993827	0.027710	0.9750	0.057560
4	1.000000	0.003674	0.9875	0.086432

```
get_ac = history.history['accuracy']
get_los = history.history['loss']
val_acc = history.history['val_accuracy']
val_loss = history.history['val_loss']

epochs = range(len(get_ac))

# Create a figure with 3 subplots arranged vertically
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(10, 15))  # Adjust figsize
as needed

# Plot training accuracy and loss
ax1.plot(epochs, get_ac, 'g', label='Accuracy of Training Data')
ax1.plot(epochs, get_los, 'r', label='Loss of Training Data')
ax1.set_title('Training Data Accuracy and Loss')
ax1.legend(loc=0)

# Plot training and validation accuracy
ax2.plot(epochs, get_ac, 'g', label='Accuracy of Training Data')
ax2.plot(epochs, val_acc, 'r', label='Accuracy of Validation Data')
ax2.set_title('Training and Validation Accuracy')
ax2.legend(loc=0)

# Plot training and validation loss
ax3.plot(epochs, get_los, 'g', label='Loss of Training Data')
ax3.plot(epochs, val_loss, 'r', label='Loss of Validation Data')
ax3.set_title('Training and Validation Loss')
ax3.legend(loc=0)

# Adjust spacing between subplots (optional)
plt.tight_layout()

plt.show()
```



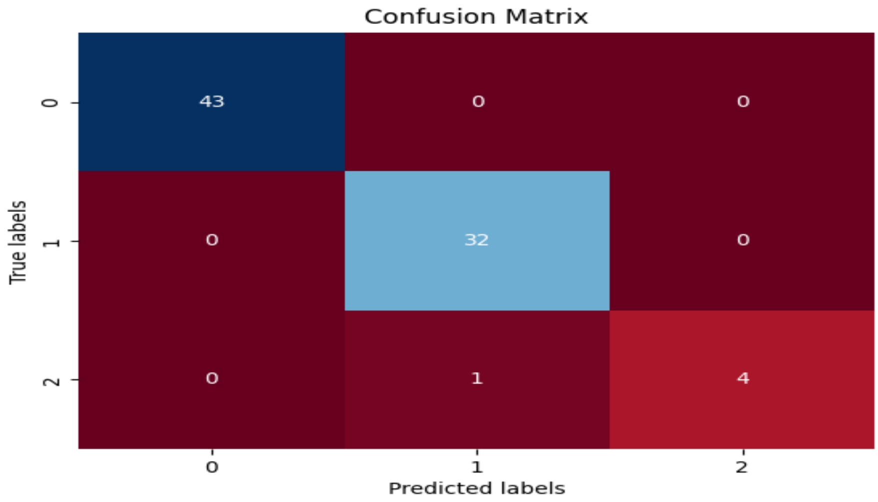





```
X_val,y_val,y_pred=[],[],[]
for images, labels in val_ds:
    y_val.extend(labels.numpy())
    X_val.extend(images.numpy())
predictions=model.predict(np.array(X_val))
for i in predictions:
    y_pred.append(np.argmax(i))
df=pd.DataFrame()
df['Actual'],df['Prediction']=y_val,y_pred
df
```

	Actual	Prediction
0	0	0
1	0	0
2	0	0
3	1	1
4	0	0
...	...	...
75	0	0
76	1	1
77	1	1
78	1	1
79	1	1

```
ax= plt.subplot()
CM = confusion_matrix(y_val,y_pred)
sns.heatmap(CM, annot=True, fmt='g', ax=ax,cbar=False,cmap='RdBu')
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
plt.show()
CM
```




```
Acc = accuracy_score(y_val,y_pred)
print("accuracy is: {0:.2f}%".format(Acc * 100))
```

```
accuracy is: 98.75%
```

```
loss, accuracy = model.evaluate(val_ds)  # Assuming you have this line

# Display accuracy percentage
print("Accuracy:", accuracy * 100, "%")  # Convert accuracy to percentage

# Get a batch of images and labels from the validation dataset
images, labels = next(iter(val_ds))  # This retrieves one batch

# Create a 4x4 grid of subplots
plt.figure(figsize=(20, 20))  # Adjust figure size as needed

for i in range(16):
    ax = plt.subplot(4, 4, i + 1)

    # Display the image
    plt.imshow(images[i].numpy().astype("uint8"))

    # Make predictions on the current image
    predictions = model.predict(tf.expand_dims(images[i], 0))
    score = tf.nn.softmax(predictions[0])

    # Determine the predicted class index and label
    predicted_class_idx = np.argmax(score)
    predicted_class_label = class_names[predicted_class_idx]
```

```
    # Determine the actual class label
    actual_class_label = class_names[labels[i]]

    # Set title and labels with accuracy percentage
    if actual_class_label == predicted_class_label:
        color = 'green'
        accuracy_text = f"Correct ({actual_class_label})"
    else:
        color = 'red'
        accuracy_text = f"Incorrect (Actual: {actual_class_label}, Predicted: {predicted_class_label})"

    # Convert score to NumPy array for rounding
    score_np = score.numpy()  # Convert TensorFlow tensor to NumPy array
    predicted_prob = round(score_np[predicted_class_idx] * 100, 2)  # Round prediction probability

    plt.title(f"{accuracy_text}\nProb: {predicted_prob}%", color=color, fontsize=15)
    plt.gca().axes.yaxis.set_ticklabels([])
    plt.gca().axes.xaxis.set_ticklabels([])

plt.tight_layout()  # Adjust spacing between subplots
plt.show()
```