```
!pip install numpy
!pip install tensorflow
!pip install matplotlib
import tensorflow as tf
from tensorflow.keras import models,layers
import matplotlib.pyplot as plt
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.26.4)
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.17.1)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.6
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.12.1)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/py
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.68.0)
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.17.1)
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.5.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.26.4)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (0.13.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensor
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.18,
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->tensorflow
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->tensorfl
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.2.0
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.8.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.55.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.7)
Requirement already satisfied: numpy<2,>=1.21 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Generate | create a dataframe with 2 columns and 10 rows | Close

```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
from tensorflow.keras import models, layers
```

Generate | randomly select 5 items from a list | Close

```
from google.colab import drive
drive.mount('/content/drive')

# Path to your dataset folder in Google Drive
dataset_path = '/content/drive/My Drive/heartimages'

import tensorflow as tf
```

```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    dataset_path,
    shuffle=True,
    image_size=(256, 256),
    batch_size=32
)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Found 928 files belonging to 4 classes.
```

```
class_names=dataset.class_names
class_names
```

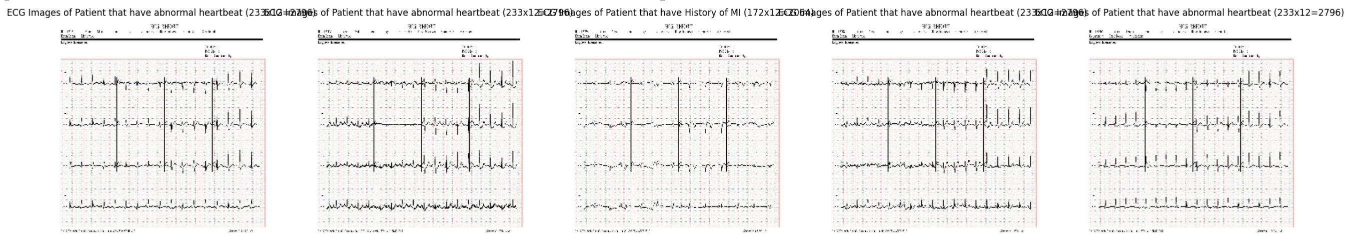
```
['ECG Images of Myocardial Infarction Patients (240x12=2880)',
 'ECG Images of Patient that have History of MI (172x12=2064)',
 'ECG Images of Patient that have abnormal heartbeat (233x12=2796)',
 'Normal Person ECG Images (284x12=3408)']
```

```
len(dataset)
```

```
29
```

```
plt.figure(figsize=(30,30))
for image_batch,label_batch in dataset.take(1):
  for i in range(5):
    ax=plt.subplot(1,5,i+1)
    plt.imshow(image_batch[i].numpy().astype("uint8"))
    plt.title(class_names[label_batch[i]])
    plt.axis("off")
    print(label_batch.numpy())
```

```
[2 2 1 2 2 0 3 2 3 3 3 3 0 3 3 3 3 0 3 2 0 3 0 0 2 0 2 3 1 3 1 0]
[2 2 1 2 2 0 3 2 3 3 3 3 0 3 3 3 3 0 3 2 0 3 0 0 2 0 2 3 1 3 1 0]
[2 2 1 2 2 0 3 2 3 3 3 3 0 3 3 3 3 0 3 2 0 3 0 0 2 0 2 3 1 3 1 0]
[2 2 1 2 2 0 3 2 3 3 3 3 0 3 3 3 3 0 3 2 0 3 0 0 2 0 2 3 1 3 1 0]
[2 2 1 2 2 0 3 2 3 3 3 3 0 3 3 3 3 0 3 2 0 3 0 0 2 0 2 3 1 3 1 0]
```


ECG Images of Patient that have abnormal heartbeat (233x12=2796)
ECG Images of Patient that have abnormal heartbeat (233x12=2796)
ECG Images of Patient that have History of MI (172x12=2064)
ECG Images of Patient that have abnormal heartbeat (233x12=2796)
ECG Images of Patient that have abnormal heartbeat (233x12=2796)

```
BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS = 32, 256, 256, 3
```

```
train_size=0.8
dataset.take(10)
```

```
<_TakeDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,),
dtype=tf.int32, name=None))>
```

```
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=10000):
    assert (train_split + test_split + val_split) == 1

    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds
```

```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)


train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

```
Start coding or generate with AI.
```

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Use tf.keras.layers directly for preprocessing layers in newer versions of TensorFlow.
resize_and_rescale = tf.keras.Sequential([
  layers.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.Rescaling(1./255)
])
```

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Use tf.keras.layers directly for preprocessing layers
data_augmentation = tf.keras.Sequential([
  layers.RandomFlip("horizontal_and_vertical"), # 'experimental' removed
  layers.RandomRotation(0.2), # 'experimental' removed
])
```

```
resize_and_rescale = tf.keras.layers.Rescaling(1.0 / 255)  # Normalizes pixel values
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip("horizontal_and_vertical"),  # Randomly flips images
    tf.keras.layers.RandomRotation(0.2),  # Randomly rotates images
])
```

```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 4

model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])

model.build(input_shape=input_shape)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model.summary()
```

```
Model: "sequential_20"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ rescaling_8 (Rescaling)              │ (32, 256, 256, 3)           │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_39 (Conv2D)                   │ (32, 254, 254, 32)          │             896 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_39 (MaxPooling2D)      │ (32, 127, 127, 32)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_40 (Conv2D)                   │ (32, 125, 125, 64)          │          18,496 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_40 (MaxPooling2D)      │ (32, 62, 62, 64)            │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_41 (Conv2D)                   │ (32, 60, 60, 64)            │          36,928 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_41 (MaxPooling2D)      │ (32, 30, 30, 64)            │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_42 (Conv2D)                   │ (32, 28, 28, 64)            │          36,928 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_42 (MaxPooling2D)      │ (32, 14, 14, 64)            │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_43 (Conv2D)                   │ (32, 12, 12, 64)            │          36,928 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_43 (MaxPooling2D)      │ (32, 6, 6, 64)              │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_44 (Conv2D)                   │ (32, 4, 4, 64)              │          36,928 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_44 (MaxPooling2D)      │ (32, 2, 2, 64)              │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten_10 (Flatten)                 │ (32, 256)                   │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_20 (Dense)                     │ (32, 64)                    │          16,448 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_21 (Dense)                     │ (32, 4)                     │             260 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 183,812 (718.02 KB)
 Trainable params: 183,812 (718.02 KB)
 Non-trainable params: 0 (0.00 B)
```

```
model.summary()
```

```
Model: "sequential_20"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ rescaling_8 (Rescaling)              │ (32, 256, 256, 3)           │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_39 (Conv2D)                   │ (32, 254, 254, 32)          │             896 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_39 (MaxPooling2D)      │ (32, 127, 127, 32)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_40 (Conv2D)                   │ (32, 125, 125, 64)          │          18,496 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_40 (MaxPooling2D)      │ (32, 62, 62, 64)            │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_41 (Conv2D)                   │ (32, 60, 60, 64)            │          36,928 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_41 (MaxPooling2D)      │ (32, 30, 30, 64)            │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_42 (Conv2D)                   │ (32, 28, 28, 64)            │          36,928 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_42 (MaxPooling2D)      │ (32, 14, 14, 64)            │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_43 (Conv2D)                   │ (32, 12, 12, 64)            │          36,928 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_43 (MaxPooling2D)      │ (32, 6, 6, 64)              │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_44 (Conv2D)                   │ (32, 4, 4, 64)              │          36,928 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_44 (MaxPooling2D)      │ (32, 2, 2, 64)              │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten_10 (Flatten)                 │ (32, 256)                   │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_20 (Dense)                     │ (32, 64)                    │          16,448 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_21 (Dense)                     │ (32, 4)                     │             260 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 183,812 (718.02 KB)
 Trainable params: 183,812 (718.02 KB)
 Non-trainable params: 0 (0.00 B)
```

```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

```
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=30,

    verbose=1
)
```

```
Epoch 1/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 107s 4s/step - accuracy: 0.2526 - loss: 1.3904 - val_accuracy: 0.3281 - val_loss: 1.3633
Epoch 2/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 139s 4s/step - accuracy: 0.3062 - loss: 1.3730 - val_accuracy: 0.3281 - val_loss: 1.3722
Epoch 3/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 146s 5s/step - accuracy: 0.2917 - loss: 1.3766 - val_accuracy: 0.3281 - val_loss: 1.3646
Epoch 4/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 101s 4s/step - accuracy: 0.3300 - loss: 1.3717 - val_accuracy: 0.3281 - val_loss: 1.3709
Epoch 5/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 145s 5s/step - accuracy: 0.3299 - loss: 1.3696 - val_accuracy: 0.3281 - val_loss: 1.3736
Epoch 6/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 139s 4s/step - accuracy: 0.2828 - loss: 1.3794 - val_accuracy: 0.5000 - val_loss: 1.3468
Epoch 7/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 142s 4s/step - accuracy: 0.3570 - loss: 1.3586 - val_accuracy: 0.4844 - val_loss: 1.1915
Epoch 8/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 140s 4s/step - accuracy: 0.4964 - loss: 1.1489 - val_accuracy: 0.7969 - val_loss: 0.6293
Epoch 9/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 100s 4s/step - accuracy: 0.7551 - loss: 0.6576 - val_accuracy: 0.8125 - val_loss: 0.4205
Epoch 10/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 102s 4s/step - accuracy: 0.7506 - loss: 0.6107 - val_accuracy: 0.8906 - val_loss: 0.2744
Epoch 11/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 101s 4s/step - accuracy: 0.9234 - loss: 0.2532 - val_accuracy: 0.9531 - val_loss: 0.1474
Epoch 12/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 141s 4s/step - accuracy: 0.9249 - loss: 0.2053 - val_accuracy: 0.9688 - val_loss: 0.0592
Epoch 13/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 142s 4s/step - accuracy: 0.9723 - loss: 0.0886 - val_accuracy: 0.9531 - val_loss: 0.0994
Epoch 14/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 143s 4s/step - accuracy: 0.9847 - loss: 0.0530 - val_accuracy: 1.0000 - val_loss: 0.0203
Epoch 15/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 103s 4s/step - accuracy: 0.9888 - loss: 0.0391 - val_accuracy: 1.0000 - val_loss: 0.0287
Epoch 16/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 101s 4s/step - accuracy: 0.9947 - loss: 0.0323 - val_accuracy: 1.0000 - val_loss: 0.0044
Epoch 17/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 101s 4s/step - accuracy: 1.0000 - loss: 0.0148 - val_accuracy: 1.0000 - val_loss: 0.0104
Epoch 18/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 141s 4s/step - accuracy: 0.9939 - loss: 0.0234 - val_accuracy: 1.0000 - val_loss: 0.0231
Epoch 19/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 99s 4s/step - accuracy: 0.9995 - loss: 0.0118 - val_accuracy: 1.0000 - val_loss: 0.0024
Epoch 20/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 142s 4s/step - accuracy: 1.0000 - loss: 0.0029 - val_accuracy: 1.0000 - val_loss: 0.0012
Epoch 21/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 102s 4s/step - accuracy: 1.0000 - loss: 0.0011 - val_accuracy: 1.0000 - val_loss: 4.3459e-04
Epoch 22/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 99s 4s/step - accuracy: 1.0000 - loss: 7.7155e-04 - val_accuracy: 1.0000 - val_loss: 3.2375e-04
Epoch 23/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 103s 5s/step - accuracy: 1.0000 - loss: 4.5360e-04 - val_accuracy: 1.0000 - val_loss: 2.8043e-04
Epoch 24/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 139s 4s/step - accuracy: 1.0000 - loss: 3.8701e-04 - val_accuracy: 1.0000 - val_loss: 2.4190e-04
Epoch 25/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 142s 4s/step - accuracy: 1.0000 - loss: 3.2726e-04 - val_accuracy: 1.0000 - val_loss: 2.0749e-04
Epoch 26/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 143s 4s/step - accuracy: 1.0000 - loss: 3.0973e-04 - val_accuracy: 1.0000 - val_loss: 1.9326e-04
Epoch 27/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 140s 4s/step - accuracy: 1.0000 - loss: 2.3954e-04 - val_accuracy: 1.0000 - val_loss: 1.6648e-04
Epoch 28/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 141s 4s/step - accuracy: 1.0000 - loss: 2.1615e-04 - val_accuracy: 1.0000 - val_loss: 1.5156e-04
Epoch 29/30
23/23 ━━━━━━━━━━━━━━━━━━━━ 102s 4s/step - accuracy: 1.0000 - loss: 1.8514e-04 - val_accuracy: 1.0000 - val_loss: 1.3828e-04
```

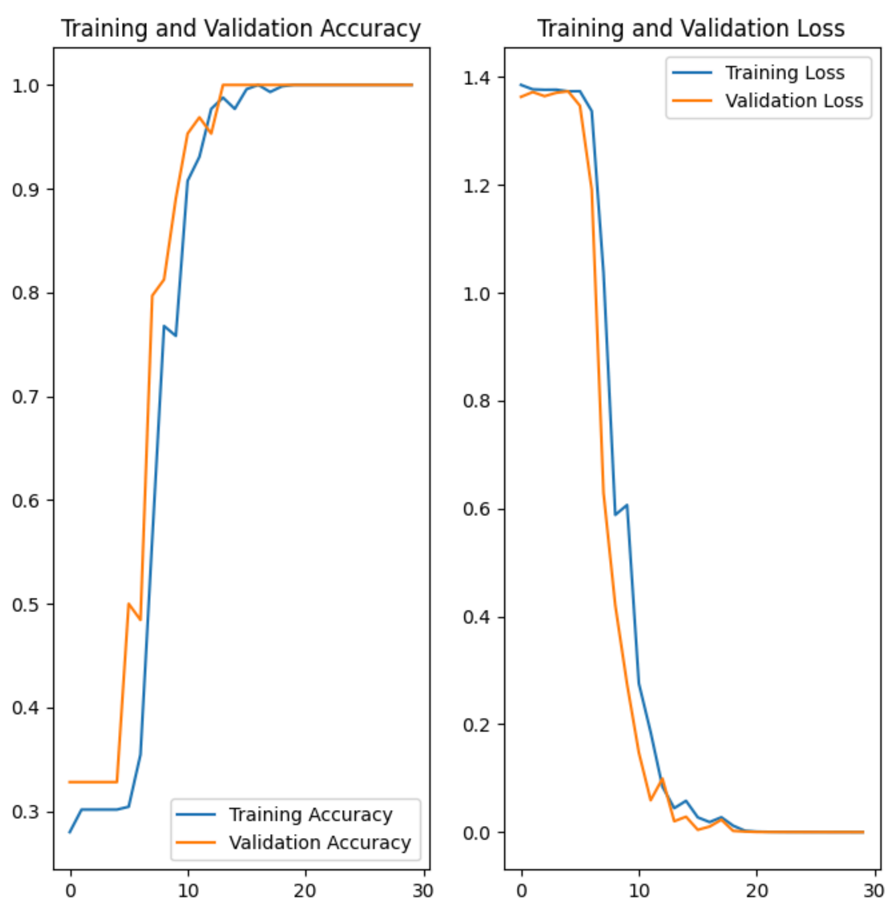
```
scores = model.evaluate(test_ds)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 1s/step - accuracy: 0.9969 - loss: 0.0133
```

```
print("[INFO] Calculating model accuracy")
scores = model.evaluate(test_ds)
print(f"Test Accuracy: {round(scores[1],4)*100}%")
```

```
[INFO] Calculating model accuracy
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 3s/step - accuracy: 0.9917 - loss: 0.0339
Test Accuracy: 99.22%
```

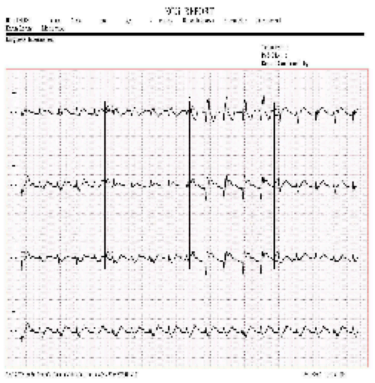
```
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(30), acc, label='Training Accuracy')
plt.plot(range(30), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(range(30), loss, label='Training Loss')
plt.plot(range(30), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



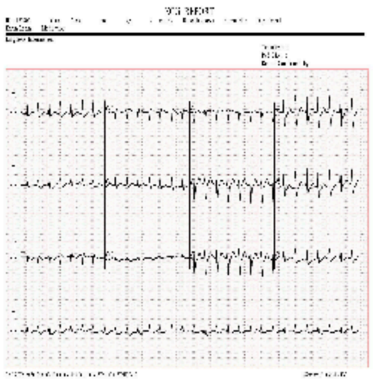
```
import numpy as np
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0) # Create a batch

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence


plt.figure(figsize=(25,25))
for images, labels in test_ds.take(1):
    for i in range(5):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        predicted_class, confidence = predict(model, images[i].numpy())

        actual_class = class_names[labels[i]]
        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")
        plt.axis("off")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 240ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step
```

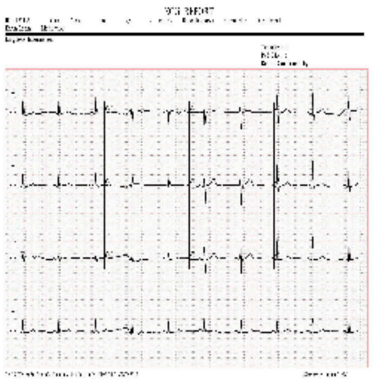
Actual: ECG Images of Myocardial Infarction Patients (240x12=2880),
Predicted: ECG Images of Myocardial Infarction Patients (240x12=2880).
Confidence: 99.98%



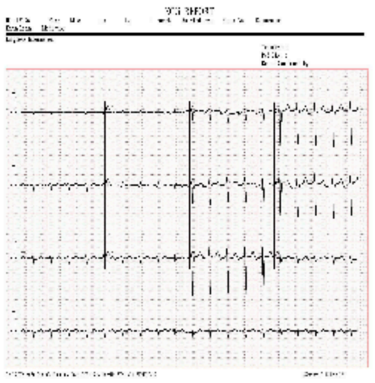
Actual: ECG Images of Patient that have abnormal heartbeat (233x12=2796),
Predicted: ECG Images of Patient that have abnormal heartbeat (233x12=2796).
Confidence: 100.0%



Actual: Normal Person ECG Images (284x12=3408),
Predicted: Normal Person ECG Images (284x12=3408).
Confidence: 100.0%



Actual: ECG Images of Patient that have abnormal heartbeat (233x12=2796),
Predicted: ECG Images of Patient that have abnormal heartbeat (233x12=2796).
Confidence: 100.0%



Actual: Normal Person ECG Images (284x12=3408),
Predicted: Normal Person ECG Images (284x12=3408).
Confidence: 100.0%