Clock Classification

The dataset contains 144 classes of clocks. All 12 Hours with all multiple of 5 minutes are included (which are the 12 clock patterns).

Datset: https://www.kaggle.com/datasets/gpiosenka/time-image-datasetclassification/data

This notebook is used to train a model for clock classification using VGG19 base model and Nadam optimizer, with a little tweaking and hypertuning in order to extract a more accurate model.

Data: Clocks

Model: VGG19

Optimizer: Nadam

Epochs: 60

Batch Size: 32

Importing libraries

In [12]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Input,Dense, Dropout, BatchNormalization, GlobalAveragePooli
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Nadam
from tensorflow.keras.regularizers import l2
from tensorflow.keras import mixed_precision
from tensorflow.keras.applications import VGG19
```

Loading data

In [13]:

```
# make data generator for training, validation, and testing
Generator = ImageDataGenerator(rescale=1./255)
train_G = Generator.flow_from_directory('train', target_size=(128, 128), batch_size=128, class_mo
test_G = Generator.flow_from_directory('test', target_size=(128, 128), batch_size=128, class_mode
valid_G = Generator.flow_from_directory('valid', target_size=(128, 128), batch_size=128, class_mo
```

```
Found 11520 images belonging to 144 classes.
Found 1440 images belonging to 144 classes.
Found 1440 images belonging to 144 classes.
```

In [14]:

```
# Load the CSV file with label meanings
df = pd.read_csv('clocks.csv')
df
```

Out[14]:

	class index	filepaths	labels	data set
0	0	train/1-00/0.jpg	1_00	train
1	0	train/1-00/1.jpg	1_00	train
2	0	train/1-00/11.jpg	1_00	train
3	0	train/1-00/12.jpg	1_00	train
4	0	train/1-00/13.jpg	1_00	train
...	...	...	...	...
14395	143	valid/9-55/65.jpg	9_55	valid
14396	143	valid/9-55/74.jpg	9_55	valid
14397	143	valid/9-55/88.jpg	9_55	valid
14398	143	valid/9-55/90.jpg	9_55	valid
14399	143	valid/9-55/95.jpg	9_55	valid

14400 rows × 4 columns

In [15]:

```
# Plot the first 9 images in the training set and their labels
fig, ax = plt.subplots(3, 3, figsize=(10, 10))
for i in range(9):
    ax[i//3, i%3].imshow(train_G[0][0][i])
    ax[i//3, i%3].axis('off')

    # Use argmax to get the index of the maximum value in the label array
    label_index = np.argmax(train_G[0][1][i])

    # Look up the corresponding label meaning from the CSV file
    label_meaning = df.loc[df['class index'] == label_index, 'labels'].values[0]

    # Display the label meaning as the title
    ax[i//3, i%3].set_title(f'Label: {label_meaning}')
```

Creating the model

In [16]:
```
# setting policy for my gpu
policy = mixed_precision.Policy('mixed_float16')
mixed_precision.set_global_policy(policy)
```

In [17]:
```
# VGG19 base model
base = VGG19(weights='imagenet', include_top=False, input_shape=(128, 128, 3))
# print out the base model layers
base.summary()
```

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 128, 128, 3)]     0

 block1_conv1 (Conv2D)       (None, 128, 128, 64)      1792

 block1_conv2 (Conv2D)       (None, 128, 128, 64)      36928

 block1_pool (MaxPooling2D)  (None, 64, 64, 64)        0

 block2_conv1 (Conv2D)       (None, 64, 64, 128)       73856

 block2_conv2 (Conv2D)       (None, 64, 64, 128)       147584

 block2_pool (MaxPooling2D)  (None, 32, 32, 128)       0

 block3_conv1 (Conv2D)       (None, 32, 32, 256)       295168

 block3_conv2 (Conv2D)       (None, 32, 32, 256)       590080

 block3_conv3 (Conv2D)       (None, 32, 32, 256)       590080

 block3_conv4 (Conv2D)       (None, 32, 32, 256)       590080

 block3_pool (MaxPooling2D)  (None, 16, 16, 256)       0

 block4_conv1 (Conv2D)       (None, 16, 16, 512)       1180160

 block4_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_conv4 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 8, 8, 512)         0

 block5_conv1 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv2 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv3 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv4 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
_________________________________________________________________
```

In [18]:
```
# with help from "Avantika" idea of freezing the last 4 layers: https://www.kaggle.com/code/avant
# Freezing the base model
for layer in base.layers[:-4]:
    layer.trainable = False
```

In [19]:
```
# Specify the input layer
input = Input(shape=(128, 128, 3))
```

```
# Adding the base model
x = base(input)
x = Dropout(0.4)(x)
# Adding Batch Normalization layer for VGG19
x = BatchNormalization()(x)
x = Dropout(0.4)(x)
# Adding Global Average Pooling layer for VGG19
x = GlobalAveragePooling2D()(x)
x = Dropout(0.4)(x)
# Adding a dense layer with 1024 neurons and ReLU activation along with Batch Normalization, Drop
x = Dense(1024, activation='relu', kernel_regularizer=l2(0.001))(x)
x = Dropout(0.4)(x)
x = BatchNormalization()(x)
x = Dropout(0.4)(x)
# Adding output layer
output = Dense(144, activation='softmax', kernel_regularizer=l2(0.01))(x)

# Dropout, Batch Normalization, and L2 are all regularization techniques used to prevent overfitt
```

In [20]:

```
# Creating our clock final model
clock = Model(input, output)
# Using Nadam optimizer
clock.compile(optimizer=Nadam(learning_rate=0.0003), loss='categorical_crossentropy', metrics=['a
# Showing the summary of the model
clock.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 128, 128, 3)]     0

 vgg19 (Functional)          (None, 4, 4, 512)         20024384

 dropout_5 (Dropout)         (None, 4, 4, 512)         0

 batch_normalization_2 (Batc  (None, 4, 4, 512)        2048
 hNormalization)

 dropout_6 (Dropout)         (None, 4, 4, 512)         0

 global_average_pooling2d_1   (None, 512)              0
 (GlobalAveragePooling2D)

 dropout_7 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 1024)              525312

 dropout_8 (Dropout)         (None, 1024)              0

 batch_normalization_3 (Batc  (None, 1024)             4096
 hNormalization)

 dropout_9 (Dropout)         (None, 1024)              0

 dense_3 (Dense)             (None, 144)               147600

=================================================================
Total params: 20,703,440
Trainable params: 7,755,408
Non-trainable params: 12,948,032
_________________________________________________________________
```

Let's train our model

In [22]:
```
clock_history = clock.fit(train_G, epochs=60, validation_data=test_G, batch_size=32)
```

```
Epoch 1/60
90/90 [==============================] - 56s 594ms/step - loss: 9.0512 - accuracy: 0.0072 - val_
loss: 7.7631 - val_accuracy: 0.0090
Epoch 2/60
90/90 [==============================] - 51s 564ms/step - loss: 8.0499 - accuracy: 0.0150 - val_
loss: 6.8635 - val_accuracy: 0.0354
Epoch 3/60
90/90 [==============================] - 49s 543ms/step - loss: 6.8187 - accuracy: 0.0443 - val_
loss: 5.9329 - val_accuracy: 0.1160
Epoch 4/60
90/90 [==============================] - 49s 543ms/step - loss: 5.6603 - accuracy: 0.1156 - val_
loss: 4.9477 - val_accuracy: 0.2625
Epoch 5/60
90/90 [==============================] - 49s 544ms/step - loss: 4.7535 - accuracy: 0.2041 - val_
loss: 4.2158 - val_accuracy: 0.3833
Epoch 6/60
90/90 [==============================] - 51s 563ms/step - loss: 4.0651 - accuracy: 0.3091 - val_
loss: 3.8668 - val_accuracy: 0.4299
Epoch 7/60
90/90 [==============================] - 49s 543ms/step - loss: 3.5300 - accuracy: 0.4069 - val_
loss: 3.3999 - val_accuracy: 0.5500
Epoch 8/60
90/90 [==============================] - 49s 543ms/step - loss: 3.0829 - accuracy: 0.5013 - val_
loss: 2.9915 - val_accuracy: 0.6222
Epoch 9/60
90/90 [==============================] - 49s 543ms/step - loss: 2.7180 - accuracy: 0.5904 - val_
loss: 2.4798 - val_accuracy: 0.7333
Epoch 10/60
90/90 [==============================] - 49s 543ms/step - loss: 2.4154 - accuracy: 0.6629 - val_
loss: 2.2688 - val_accuracy: 0.7660
Epoch 11/60
90/90 [==============================] - 49s 543ms/step - loss: 2.1374 - accuracy: 0.7396 - val_
loss: 2.0690 - val_accuracy: 0.7847
Epoch 12/60
90/90 [==============================] - 49s 543ms/step - loss: 1.9076 - accuracy: 0.8060 - val_
loss: 1.8414 - val_accuracy: 0.8528
Epoch 13/60
90/90 [==============================] - 59s 659ms/step - loss: 1.7025 - accuracy: 0.8659 - val_
loss: 1.6270 - val_accuracy: 0.8687
Epoch 14/60
90/90 [==============================] - 49s 543ms/step - loss: 1.5353 - accuracy: 0.8977 - val_
loss: 1.4105 - val_accuracy: 0.9167
Epoch 15/60
90/90 [==============================] - 49s 543ms/step - loss: 1.3928 - accuracy: 0.9304 - val_
loss: 1.3364 - val_accuracy: 0.9090
Epoch 16/60
90/90 [==============================] - 49s 543ms/step - loss: 1.2615 - accuracy: 0.9530 - val_
loss: 1.2355 - val_accuracy: 0.9153
Epoch 17/60
90/90 [==============================] - 49s 543ms/step - loss: 1.1537 - accuracy: 0.9646 - val_
loss: 1.1742 - val_accuracy: 0.9257
Epoch 18/60
90/90 [==============================] - 49s 543ms/step - loss: 1.0684 - accuracy: 0.9743 - val_
loss: 1.0937 - val_accuracy: 0.9271
Epoch 19/60
90/90 [==============================] - 49s 543ms/step - loss: 0.9871 - accuracy: 0.9827 - val_
loss: 1.0092 - val_accuracy: 0.9375
Epoch 20/60
90/90 [==============================] - 49s 543ms/step - loss: 0.9252 - accuracy: 0.9870 - val_
loss: 1.0015 - val_accuracy: 0.9333
Epoch 21/60
90/90 [==============================] - 49s 543ms/step - loss: 0.8666 - accuracy: 0.9905 - val_
```

```
loss: 0.9081 - val_accuracy: 0.9535
Epoch 22/60
90/90 [==============================] - 49s 543ms/step - loss: 0.8128 - accuracy: 0.9924 - val_
loss: 0.8712 - val_accuracy: 0.9500
Epoch 23/60
90/90 [==============================] - 49s 543ms/step - loss: 0.7654 - accuracy: 0.9942 - val_
loss: 0.8771 - val_accuracy: 0.9403
Epoch 24/60
90/90 [==============================] - 49s 544ms/step - loss: 0.7272 - accuracy: 0.9952 - val_
loss: 0.8051 - val_accuracy: 0.9493
Epoch 25/60
90/90 [==============================] - 49s 543ms/step - loss: 0.6909 - accuracy: 0.9962 - val_
loss: 0.7890 - val_accuracy: 0.9493
Epoch 26/60
90/90 [==============================] - 49s 543ms/step - loss: 0.6600 - accuracy: 0.9963 - val_
loss: 0.7405 - val_accuracy: 0.9542
Epoch 27/60
90/90 [==============================] - 49s 543ms/step - loss: 0.6286 - accuracy: 0.9966 - val_
loss: 0.7161 - val_accuracy: 0.9486
Epoch 28/60
90/90 [==============================] - 49s 543ms/step - loss: 0.5990 - accuracy: 0.9970 - val_
loss: 0.6878 - val_accuracy: 0.9563
Epoch 29/60
90/90 [==============================] - 49s 542ms/step - loss: 0.5720 - accuracy: 0.9971 - val_
loss: 0.6674 - val_accuracy: 0.9569
Epoch 30/60
90/90 [==============================] - 49s 543ms/step - loss: 0.5457 - accuracy: 0.9978 - val_
loss: 0.6386 - val_accuracy: 0.9597
Epoch 31/60
90/90 [==============================] - 49s 543ms/step - loss: 0.5227 - accuracy: 0.9971 - val_
loss: 0.6172 - val_accuracy: 0.9549
Epoch 32/60
90/90 [==============================] - 49s 542ms/step - loss: 0.5016 - accuracy: 0.9976 - val_
loss: 0.6063 - val_accuracy: 0.9576
Epoch 33/60
90/90 [==============================] - 49s 543ms/step - loss: 0.4791 - accuracy: 0.9981 - val_
loss: 0.5856 - val_accuracy: 0.9590
Epoch 34/60
90/90 [==============================] - 49s 543ms/step - loss: 0.4587 - accuracy: 0.9980 - val_
loss: 0.5706 - val_accuracy: 0.9590
Epoch 35/60
90/90 [==============================] - 49s 543ms/step - loss: 0.4433 - accuracy: 0.9975 - val_
loss: 0.5725 - val_accuracy: 0.9576
Epoch 36/60
90/90 [==============================] - 49s 543ms/step - loss: 0.4259 - accuracy: 0.9978 - val_
loss: 0.5365 - val_accuracy: 0.9563
Epoch 37/60
90/90 [==============================] - 49s 543ms/step - loss: 0.4073 - accuracy: 0.9983 - val_
loss: 0.5134 - val_accuracy: 0.9597
Epoch 38/60
90/90 [==============================] - 49s 543ms/step - loss: 0.3952 - accuracy: 0.9977 - val_
loss: 0.5068 - val_accuracy: 0.9611
Epoch 39/60
90/90 [==============================] - 49s 543ms/step - loss: 0.3797 - accuracy: 0.9980 - val_
loss: 0.5080 - val_accuracy: 0.9590
Epoch 40/60
90/90 [==============================] - 49s 543ms/step - loss: 0.3626 - accuracy: 0.9981 - val_
loss: 0.4846 - val_accuracy: 0.9590
Epoch 41/60
90/90 [==============================] - 49s 543ms/step - loss: 0.3494 - accuracy: 0.9984 - val_
loss: 0.4664 - val_accuracy: 0.9618
Epoch 42/60
```

```
90/90 [==============================] - 49s 543ms/step - loss: 0.3386 - accuracy: 0.9981 - val_
loss: 0.4772 - val_accuracy: 0.9556
Epoch 43/60
90/90 [==============================] - 49s 543ms/step - loss: 0.3341 - accuracy: 0.9974 - val_
loss: 0.4588 - val_accuracy: 0.9597
Epoch 44/60
90/90 [==============================] - 49s 543ms/step - loss: 0.3250 - accuracy: 0.9976 - val_
loss: 0.4565 - val_accuracy: 0.9556
Epoch 45/60
90/90 [==============================] - 49s 543ms/step - loss: 0.3069 - accuracy: 0.9984 - val_
loss: 0.4464 - val_accuracy: 0.9563
Epoch 46/60
90/90 [==============================] - 49s 543ms/step - loss: 0.3065 - accuracy: 0.9975 - val_
loss: 0.4279 - val_accuracy: 0.9597
Epoch 47/60
90/90 [==============================] - 49s 543ms/step - loss: 0.2913 - accuracy: 0.9979 - val_
loss: 0.4194 - val_accuracy: 0.9639
Epoch 48/60
90/90 [==============================] - 49s 543ms/step - loss: 0.2854 - accuracy: 0.9974 - val_
loss: 0.4712 - val_accuracy: 0.9500
Epoch 49/60
90/90 [==============================] - 49s 543ms/step - loss: 0.3335 - accuracy: 0.9898 - val_
loss: 0.6621 - val_accuracy: 0.8917
Epoch 50/60
90/90 [==============================] - 49s 543ms/step - loss: 0.3523 - accuracy: 0.9878 - val_
loss: 0.4798 - val_accuracy: 0.9403
Epoch 51/60
90/90 [==============================] - 49s 543ms/step - loss: 0.3261 - accuracy: 0.9927 - val_
loss: 0.4280 - val_accuracy: 0.9576
Epoch 52/60
90/90 [==============================] - 49s 543ms/step - loss: 0.2913 - accuracy: 0.9963 - val_
loss: 0.4092 - val_accuracy: 0.9563
Epoch 53/60
90/90 [==============================] - 49s 543ms/step - loss: 0.2693 - accuracy: 0.9981 - val_
loss: 0.3674 - val_accuracy: 0.9660
Epoch 54/60
90/90 [==============================] - 49s 543ms/step - loss: 0.2563 - accuracy: 0.9983 - val_
loss: 0.3804 - val_accuracy: 0.9556
Epoch 55/60
90/90 [==============================] - 49s 543ms/step - loss: 0.2494 - accuracy: 0.9980 - val_
loss: 0.3796 - val_accuracy: 0.9576
Epoch 56/60
90/90 [==============================] - 49s 543ms/step - loss: 0.2404 - accuracy: 0.9981 - val_
loss: 0.3593 - val_accuracy: 0.9625
Epoch 57/60
90/90 [==============================] - 49s 543ms/step - loss: 0.2350 - accuracy: 0.9984 - val_
loss: 0.3516 - val_accuracy: 0.9646
Epoch 58/60
90/90 [==============================] - 49s 544ms/step - loss: 0.2289 - accuracy: 0.9984 - val_
loss: 0.3502 - val_accuracy: 0.9639
Epoch 59/60
90/90 [==============================] - 49s 543ms/step - loss: 0.2241 - accuracy: 0.9984 - val_
loss: 0.3564 - val_accuracy: 0.9597
Epoch 60/60
90/90 [==============================] - 49s 543ms/step - loss: 0.2217 - accuracy: 0.9983 - val_
loss: 0.3407 - val_accuracy: 0.9674
```

Model Evaluation

In [23]:
```
clock.evaluate(valid_G)
```

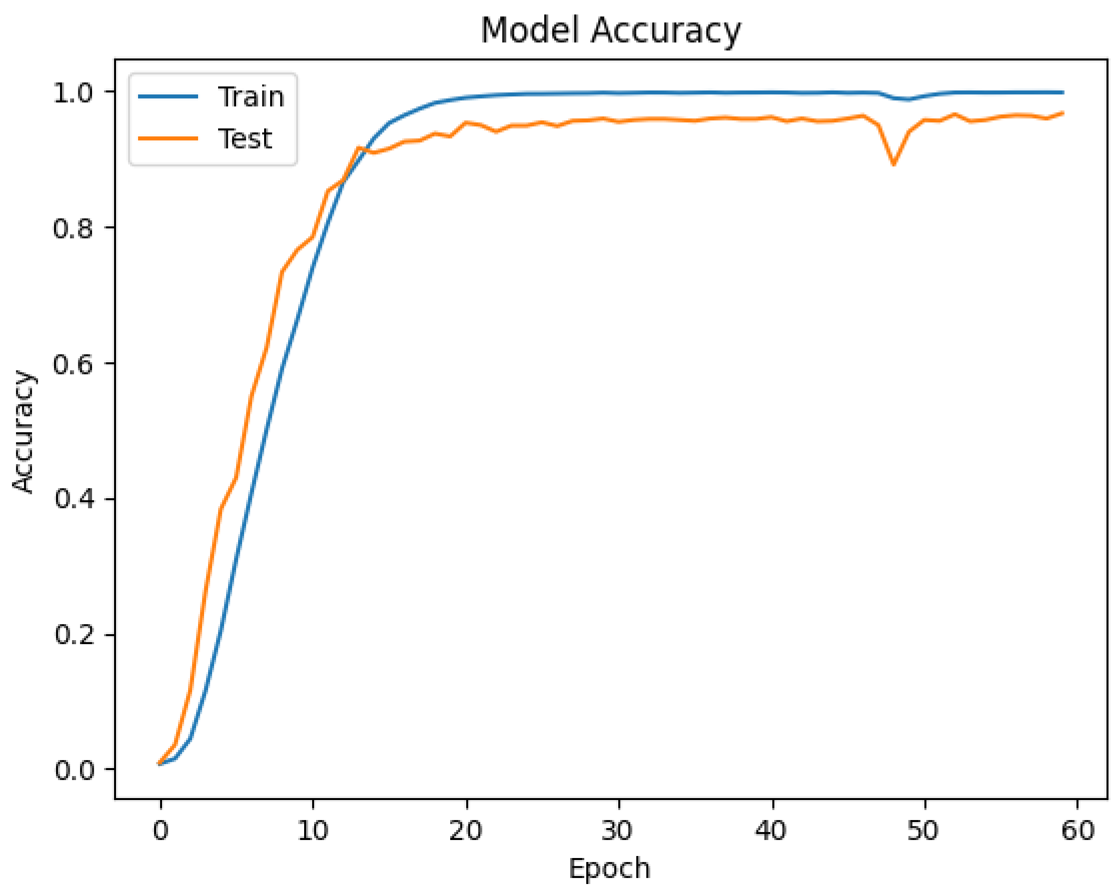
```
12/12 [==============================] - 7s 597ms/step - loss: 0.3203 - accuracy: 0.9715
```

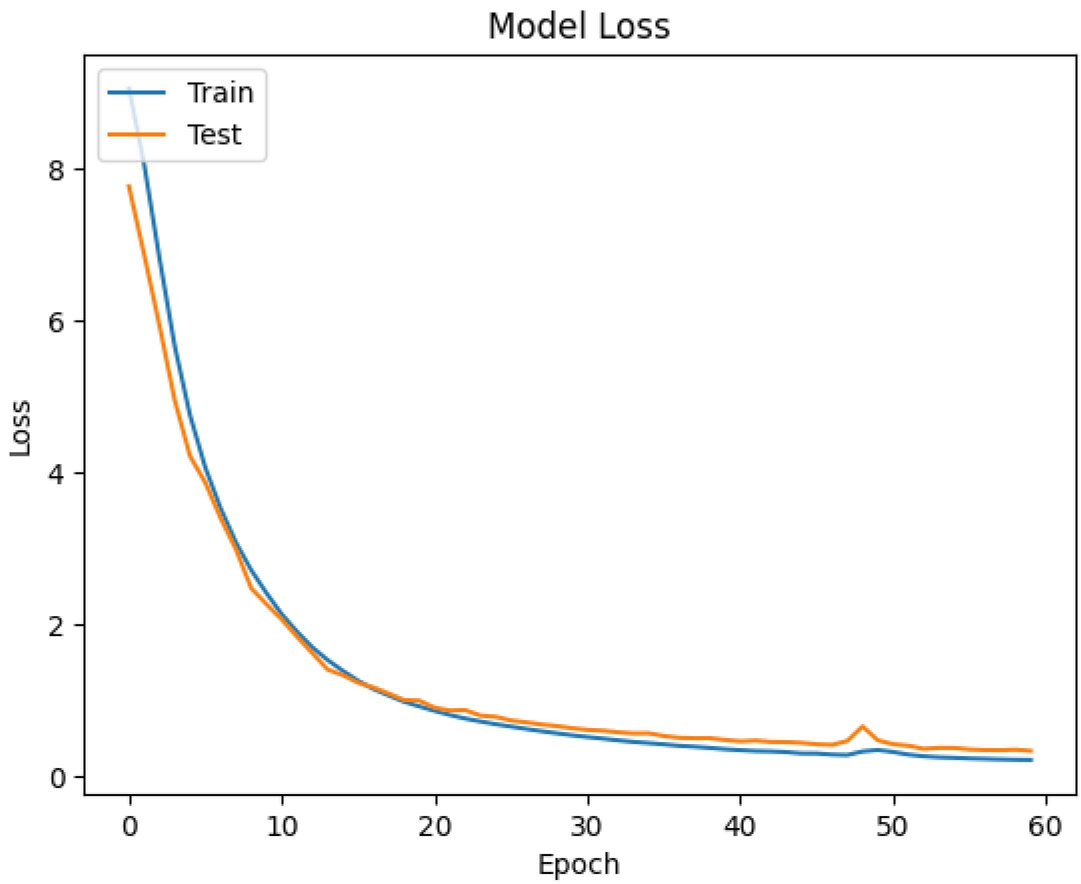
Out[23]: [0.32030895352363586, 0.9715277552604675]

Our Accuracy is 97.15%

In [24]:
```
# Plot the curve of accuracy and Loss
plt.plot(clock_history.history['accuracy'])
plt.plot(clock_history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

plt.plot(clock_history.history['loss'])
plt.plot(clock_history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```





Saving Our Model

In [27]:
```
clock.save('VGG19_clock_Nadam_97.15.h5')
```

In [ ]: