BlackBox_WM_CIFAR10_resnetv1

April 11, 2023

1 Embedding watermarks into Deep Neural Networks

Digital watermarking technology is used to protect intellectual property or detect intellectual property infringement of trained models.

1.0.1 Following code embeds watermarks into a resnet architecture using black box watermarking approach. The dataset used is CIFAR10.

1.1 Necessary Imports

[1]:
```python
import os
import tensorflow as tf
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
import numpy as np
import pandas as pd
import random
from tensorflow.keras.preprocessing.image import img_to_array, load_img,␣
 ↪array_to_img, ImageDataGenerator
import gzip
from skimage.util.noise import random_noise
from resnet20 import resnet_v1
```

[2]:
```python
from tensorflow import keras
from tensorflow.keras.layers import Dense, Conv2D, BatchNormalization,␣
 ↪Activation
from tensorflow.keras.layers import AveragePooling2D, Input, Flatten
from tensorflow.keras.regularizers import l2
from tensorflow.keras.models import Model



def resnet_layer(inputs, num_filters=16, kernel_size=3, strides=1,␣
 ↪activation='relu', batch_normalization=True, conv_first=True):
    conv = Conv2D(num_filters,
                  kernel_size=kernel_size,
                  strides=strides,
                  padding='same',
                  kernel_initializer='he_normal',
                  kernel_regularizer=l2(1e-4))
    x = inputs
    if conv_first:
        x = conv(x)
        if batch_normalization:
            x = BatchNormalization()(x)
        if activation is not None:
            x = Activation(activation)(x)
    else:
        if batch_normalization:
            x = BatchNormalization()(x)
        if activation is not None:
            x = Activation(activation)(x)
        x = conv(x)
    return x


def resnet_v1(input_shape, depth, num_classes=10):
    """ResNet Version 1 Model builder

    args:
        input_shape (tensor): shape of input image tensor
        depth (int): number of core convolutional layers
        num_classes (int): number of classes (CIFAR10 has 10)

    return:
        model (Model): Keras model instance
    """
    if (depth - 2) % 6 != 0:
        raise ValueError('depth should be 6n+2 (eg 20, 32, 44)')
    # Start model definition.
    num_filters = 16
    num_res_blocks = int((depth - 2) / 6)

    inputs = Input(shape=input_shape)
    x = resnet_layer(inputs=inputs)
    # Instantiate the stack of residual units
    for stack in range(3):
        for res_block in range(num_res_blocks):
            strides = 1
            if stack > 0 and res_block == 0:  # first layer but not first stack
                strides = 2  # downsample
            y = resnet_layer(inputs=x,
                             num_filters=num_filters,
                             strides=strides)
            y = resnet_layer(inputs=y,
                             num_filters=num_filters,
                               activation=None)
            if stack > 0 and res_block == 0:  # first layer but not first stack
                # linear projection residual shortcut connection to match
                # changed dims
                x = resnet_layer(inputs=x,
                                 num_filters=num_filters,
                                 kernel_size=1,
                                 strides=strides,
                                 activation=None,
                                 batch_normalization=False)
            x = keras.layers.add([x, y])
            x = Activation('relu')(x)
        num_filters *= 2

    # Add classifier on top.
    # v1 does not use BN after last shortcut connection-ReLU
    x = AveragePooling2D(pool_size=4)(x)
    y = Flatten()(x)
    y = Dense(num_classes,
                    kernel_initializer='he_normal')(y)
    outputs = Activation('softmax')(y)

    # Instantiate model.
    model = Model(inputs=inputs, outputs=outputs)
    return model
```

1.2 Defining Essential Functions

[3]:
```
print(tf.config.experimental.list_physical_devices('GPU'))
```

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

[4]:
```
os.environ["CUDA_VISIBLE_DEVICES"]='0'
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
    except RuntimeError as e:
        print(e)


def log(content):
    if log_dir is not None:
        log_file = log_dir + '/log.txt'
        with open(log_file, 'a') as f:
            print(content, file=f)
```

```
def lr_schedule(epoch):
    lr = 1e-3
    if epoch > 70:
        lr *= 1e-3
    if epoch > 60:
        lr *= 1e-2
    elif epoch > 40:
        lr *= 1e-1
    print('Learning rate: ', lr)
    return lr


def load_data(dataset: str):
    if dataset == 'MNIST':
        mnist = tf.keras.datasets.mnist
        (training_images, training_labels), (test_images, test_labels) = mnist.
↪load_data()
        training_images = training_images.reshape(60000, 28, 28, 1)
        test_images = test_images.reshape(10000, 28, 28, 1)
    elif dataset == 'CIFAR10':
        cifar10 = tf.keras.datasets.cifar10
        (training_images, training_labels), (test_images, test_labels) =␣
↪cifar10.load_data()

    return training_images, training_labels, test_images, test_labels


def get_unrelated_images(dataset: str, sample_rate):
    watermark_images = []
    if dataset == 'MNIST':
        # e-mnist for the mnist dataset
        train_images_path = './data/emnist/
↪emnist-letters-train-images-idx3-ubyte.gz'
        train_labels_path = './data/emnist/
↪emnist-letters-train-labels-idx1-ubyte.gz'
        with gzip.open(train_images_path, 'rb') as imgpath:
            images = np.frombuffer(imgpath.read(), np.uint8, offset=16).
↪reshape((-1, 28, 28, 1))
        with gzip.open(train_labels_path, 'rb') as lbpath:
            labels = np.frombuffer(lbpath.read(), np.uint8, offset=8)
        for i in range(images.shape[0]):
            if labels[i] == 23:
                watermark_images.append(images[i])
    elif dataset == 'CIFAR10':
        # mnist for the cifar10 dataset
        mnist = tf.keras.datasets.mnist
        (training_images, training_labels), (_, _) = mnist.load_data()
        for i in range(len(training_labels)):
            if training_labels[i] == 1:
                image = array_to_img(training_images[i].reshape(28, 28, 1))
                image = image.convert(mode='RGB')
                image = image.resize((32, 32))
                image = img_to_array(image)
                watermark_images.append(image)

    random.shuffle(watermark_images)
    watermark_images = np.array(watermark_images)
    train_sample_number = int(len(watermark_images) * sample_rate)
    train_sample = watermark_images[:train_sample_number]
    test_sample = watermark_images[train_sample_number:]

    return train_sample, test_sample


def watermark(train_images, train_labels, old_label, new_label, sample_rate,␣
 ↪dataset: str, wtype='content'):
    """prepare the dataset for training to embed the watermark

    args:
        train_images: clean training images
        train_labels: clean training labels
        old_label: label for watermarking
        new_label: label after watermarking
        sample_rate: sample rate for embedding the watermark
        wtype: watermarking type ('content', 'noise', 'unrelated')

    return:
        processed training and testing dataset for watermarking
    """
    if wtype == 'unrelated':
        train_sample, test_sample = get_unrelated_images(dataset, sample_rate)
    else:
        watermark_images = []
        for i in range(len(train_labels)):
            if train_labels[i] == old_label:
                watermark_images.append(train_images[i])

        if wtype == 'content':
            # add the trigger (size= 8*8) at the right bottom corner
            mark_image = load_img('./mark/apple_black.png',␣
 ↪color_mode='grayscale', target_size=(8, 8))
            for i in range(len(watermark_images)):
                image = array_to_img(watermark_images[i])
                image.paste(mark_image, box=(image.size[0] - 8, image.size[1] -␣
↪8))
                watermark_images[i] = img_to_array(image)
        elif wtype == 'noise':
            for i in range(len(watermark_images)):
                image = random_noise(watermark_images[i] / 255.0, seed=1)
                image = image * 255.0
                watermark_images[i] = image

        random.shuffle(watermark_images)
        watermark_images = np.array(watermark_images)
        train_sample_number = int(len(watermark_images) * sample_rate)
        train_sample = watermark_images[:train_sample_number]
        test_sample = watermark_images[train_sample_number:]

    if dataset == 'MNIST':
        return train_sample, np.ones(train_sample.shape[0]) * new_label,␣
↪test_sample, np.ones(
            test_sample.shape[0]) * new_label
    elif dataset == 'CIFAR10':
        return train_sample, np.ones((train_sample.shape[0], 1)) * new_label,␣
↪test_sample, np.ones((
            test_sample.shape[0], 1)) * new_label
```

1.3 Training ResNet V1 on CIFAR10 Dataset

[5]:
```
log_dir = './logs'
if not os.path.exists(log_dir):
    os.makedirs(log_dir)
print('log saved at ' + log_dir)

dataset = 'CIFAR10'
batch_size = 64
epochs = 30


training_images, training_labels, test_images, test_labels = load_data(dataset)
training_images = training_images / 255.0
test_images = test_images / 255.0
training_labels = tf.keras.utils.to_categorical(training_labels, 10)
test_labels = tf.keras.utils.to_categorical(test_labels, 10)

input_shape = training_images.shape[1:]
model = resnet_v1(input_shape=input_shape, depth=20)
model.summary(print_fn=log)
```

```
reduce_lr = tf.keras.callbacks.LearningRateScheduler(lr_schedule)
lr_reducer = tf.keras.callbacks.ReduceLROnPlateau(factor=np.sqrt(0.1),
                                                      cooldown=0,
                                                      patience=5,
                                                      min_lr=0.5e-6)
model.compile(optimizer=tf.keras.optimizers.Adam(),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

print('Using real-time data augmentation.')
data_gen = ImageDataGenerator(
            rotation_range=10,
            width_shift_range=0.1,
            height_shift_range=0.1
        )
history = model.fit(data_gen.flow(training_images, training_labels,␣
 ↪batch_size=batch_size),
                            epochs=epochs,
                            validation_data=(test_images, test_labels),
                            callbacks=[reduce_lr, lr_reducer],
                            steps_per_epoch=training_images.shape[0] //␣
 ↪batch_size)

pd.DataFrame(history.history).to_csv(log_dir + '/log.csv')

loss, acc = model.evaluate(test_images, test_labels, verbose=1)
print('Original Model Testing Accuracy: ', acc)
print('Original Model Testing Loss: ', loss)

if log_dir is not None:
    model.save(log_dir + '/original_model.h5')
```

```
log saved at ./logs
Using real-time data augmentation.
Learning rate:  0.001
Epoch 1/30
781/781 [==============================] - 41s 38ms/step - loss: 1.6448 -
accuracy: 0.4678 - val_loss: 1.3160 - val_accuracy: 0.5871 - lr: 0.0010
Learning rate:  0.001
Epoch 2/30
781/781 [==============================] - 29s 37ms/step - loss: 1.2390 -
accuracy: 0.6156 - val_loss: 1.4242 - val_accuracy: 0.5861 - lr: 0.0010
Learning rate:  0.001
Epoch 3/30
781/781 [==============================] - 29s 37ms/step - loss: 1.0689 -
accuracy: 0.6781 - val_loss: 1.1297 - val_accuracy: 0.6671 - lr: 0.0010
Learning rate:  0.001
```

```
Epoch 4/30
781/781 [==============================] - 29s 37ms/step - loss: 0.9708 -
accuracy: 0.7140 - val_loss: 1.0432 - val_accuracy: 0.7006 - lr: 0.0010
Learning rate:  0.001
Epoch 5/30
781/781 [==============================] - 28s 36ms/step - loss: 0.8818 -
accuracy: 0.7476 - val_loss: 1.1845 - val_accuracy: 0.6727 - lr: 0.0010
Learning rate:  0.001
Epoch 6/30
781/781 [==============================] - 32s 41ms/step - loss: 0.8275 -
accuracy: 0.7664 - val_loss: 0.8578 - val_accuracy: 0.7599 - lr: 0.0010
Learning rate:  0.001
Epoch 7/30
781/781 [==============================] - 29s 37ms/step - loss: 0.7849 -
accuracy: 0.7831 - val_loss: 1.0214 - val_accuracy: 0.7259 - lr: 0.0010
Learning rate:  0.001
Epoch 8/30
781/781 [==============================] - 28s 36ms/step - loss: 0.7414 -
accuracy: 0.7999 - val_loss: 0.8418 - val_accuracy: 0.7718 - lr: 0.0010
Learning rate:  0.001
Epoch 9/30
781/781 [==============================] - 28s 36ms/step - loss: 0.7167 -
accuracy: 0.8077 - val_loss: 0.9538 - val_accuracy: 0.7345 - lr: 0.0010
Learning rate:  0.001
Epoch 10/30
781/781 [==============================] - 29s 37ms/step - loss: 0.6862 -
accuracy: 0.8197 - val_loss: 0.8181 - val_accuracy: 0.7857 - lr: 0.0010
Learning rate:  0.001
Epoch 11/30
781/781 [==============================] - 28s 36ms/step - loss: 0.6676 -
accuracy: 0.8262 - val_loss: 0.8843 - val_accuracy: 0.7607 - lr: 0.0010
Learning rate:  0.001
Epoch 12/30
781/781 [==============================] - 30s 39ms/step - loss: 0.6404 -
accuracy: 0.8384 - val_loss: 1.0423 - val_accuracy: 0.7327 - lr: 0.0010
Learning rate:  0.001
Epoch 13/30
781/781 [==============================] - 29s 37ms/step - loss: 0.6298 -
accuracy: 0.8391 - val_loss: 0.8063 - val_accuracy: 0.7885 - lr: 0.0010
Learning rate:  0.001
Epoch 14/30
781/781 [==============================] - 28s 36ms/step - loss: 0.6084 -
accuracy: 0.8477 - val_loss: 0.8154 - val_accuracy: 0.7981 - lr: 0.0010
Learning rate:  0.001
Epoch 15/30
781/781 [==============================] - 28s 36ms/step - loss: 0.6026 -
accuracy: 0.8512 - val_loss: 0.7174 - val_accuracy: 0.8247 - lr: 0.0010
Learning rate:  0.001
```

```
Epoch 16/30
781/781 [==============================] - 28s 36ms/step - loss: 0.5862 -
accuracy: 0.8563 - val_loss: 0.7775 - val_accuracy: 0.8087 - lr: 0.0010
Learning rate:  0.001
Epoch 17/30
781/781 [==============================] - 28s 36ms/step - loss: 0.5782 -
accuracy: 0.8600 - val_loss: 0.9212 - val_accuracy: 0.7784 - lr: 0.0010
Learning rate:  0.001
Epoch 18/30
781/781 [==============================] - 28s 36ms/step - loss: 0.5606 -
accuracy: 0.8671 - val_loss: 0.7810 - val_accuracy: 0.8076 - lr: 0.0010
Learning rate:  0.001
Epoch 19/30
781/781 [==============================] - 28s 36ms/step - loss: 0.5514 -
accuracy: 0.8706 - val_loss: 0.7906 - val_accuracy: 0.8002 - lr: 0.0010
Learning rate:  0.001
Epoch 20/30
781/781 [==============================] - 28s 35ms/step - loss: 0.5461 -
accuracy: 0.8718 - val_loss: 0.6741 - val_accuracy: 0.8410 - lr: 0.0010
Learning rate:  0.001
Epoch 21/30
781/781 [==============================] - 28s 36ms/step - loss: 0.5413 -
accuracy: 0.8749 - val_loss: 0.6911 - val_accuracy: 0.8344 - lr: 0.0010
Learning rate:  0.001
Epoch 22/30
781/781 [==============================] - 28s 35ms/step - loss: 0.5279 -
accuracy: 0.8805 - val_loss: 0.6985 - val_accuracy: 0.8318 - lr: 0.0010
Learning rate:  0.001
Epoch 23/30
781/781 [==============================] - 28s 36ms/step - loss: 0.5200 -
accuracy: 0.8837 - val_loss: 0.8116 - val_accuracy: 0.8118 - lr: 0.0010
Learning rate:  0.001
Epoch 24/30
781/781 [==============================] - 31s 40ms/step - loss: 0.5227 -
accuracy: 0.8816 - val_loss: 0.9858 - val_accuracy: 0.7735 - lr: 0.0010
Learning rate:  0.001
Epoch 25/30
781/781 [==============================] - 28s 36ms/step - loss: 0.5174 -
accuracy: 0.8838 - val_loss: 0.8117 - val_accuracy: 0.8133 - lr: 0.0010
Learning rate:  0.001
Epoch 26/30
781/781 [==============================] - 28s 36ms/step - loss: 0.5036 -
accuracy: 0.8896 - val_loss: 0.8927 - val_accuracy: 0.7931 - lr: 0.0010
Learning rate:  0.001
Epoch 27/30
781/781 [==============================] - 28s 36ms/step - loss: 0.4993 -
accuracy: 0.8924 - val_loss: 1.0116 - val_accuracy: 0.7640 - lr: 0.0010
Learning rate:  0.001
```

```
Epoch 28/30
781/781 [==============================] - 28s 36ms/step - loss: 0.4967 -
accuracy: 0.8929 - val_loss: 0.7887 - val_accuracy: 0.8100 - lr: 0.0010
Learning rate:  0.001
Epoch 29/30
781/781 [==============================] - 28s 36ms/step - loss: 0.4953 -
accuracy: 0.8941 - val_loss: 0.6977 - val_accuracy: 0.8396 - lr: 0.0010
Learning rate:  0.001
Epoch 30/30
781/781 [==============================] - 28s 35ms/step - loss: 0.4899 -
accuracy: 0.8969 - val_loss: 0.8339 - val_accuracy: 0.8074 - lr: 0.0010
313/313 [==============================] - 3s 10ms/step - loss: 0.8339 -
accuracy: 0.8074
Original Model Testing Accuracy:  0.8073999881744385
Original Model Testing Loss:  0.8338788151741028
```

1.4 Training resnet v1 on CIFAR10 Dataset with black-box watermarking approach

[6]:
```python
if __name__ == '__main__':
    wtype = 'content'
    dataset = 'CIFAR10'
    training_nums = 25000
    batch_size = 64
    epochs = 30 # 80 for cifar10 and 10 for mnist
    no_augmentation = False
    old_label = 1
    new_label = 3
    log_dir = './logs'
    if not os.path.exists(log_dir):
        os.makedirs(log_dir)
    print('log saved at ' + log_dir)


    training_images, training_labels, test_images, test_labels =␣
 ↪load_data(dataset)
    train_sample_images, train_sample_labels, test_sample_images,␣
 ↪test_sample_labels = watermark(training_images,
                                                                                ␣
 ↪                  training_labels, old_label, new_label,
                                                                                ␣
 ↪                  0.1, dataset,
                                                                                ␣
 ↪                  wtype=wtype)

    training_labels = tf.keras.utils.to_categorical(training_labels, 10)
    test_labels = tf.keras.utils.to_categorical(test_labels, 10)
    train_sample_labels = tf.keras.utils.to_categorical(train_sample_labels, 10)
    test_sample_labels = tf.keras.utils.to_categorical(test_sample_labels, 10)

    training_images = training_images / 255.0
    test_images = test_images / 255.0
    train_sample_images = train_sample_images / 255.0
    test_sample_images = test_sample_images / 255.0
    training_all_images = np.concatenate((training_images[:training_nums],␣
↪train_sample_images), axis=0)
    training_all_labels = np.concatenate((training_labels[:training_nums],␣
↪train_sample_labels), axis=0)

    input_shape = training_images.shape[1:]
    model = resnet_v1(input_shape=input_shape, depth=20)
    model.summary(print_fn=log)

    reduce_lr = tf.keras.callbacks.LearningRateScheduler(lr_schedule)
    lr_reducer = tf.keras.callbacks.ReduceLROnPlateau(factor=np.sqrt(0.1),
                                                      cooldown=0,
                                                      patience=5,
                                                      min_lr=0.5e-6)
    model.compile(optimizer=tf.keras.optimizers.Adam(),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    if no_augmentation:
        print('Not using data augmentation.')
        history0 = model.fit(training_all_images, training_all_labels,
                            batch_size=batch_size,
                            epochs=epochs,
                            validation_data=(test_images, test_labels),
                            callbacks=[reduce_lr, lr_reducer])
    else:
        print('Using real-time data augmentation.')
        data_gen = ImageDataGenerator(
            rotation_range=10,
            width_shift_range=0.1,
            height_shift_range=0.1
        )
        history0 = model.fit(data_gen.flow(training_all_images,␣
↪training_all_labels, batch_size=batch_size),
                            epochs=epochs,
                            validation_data=(test_images, test_labels),
                            callbacks=[reduce_lr, lr_reducer],
                            steps_per_epoch=training_all_images.shape[0] //␣
↪batch_size)
```

```
    pd.DataFrame(history.history).to_csv(log_dir + '/log.csv')

    if log_dir is not None:
        model.save(log_dir + '/watermarked_model.h5')
        np.savez(log_dir + "/content_trigger.npz",␣
↪test_sample_images=test_sample_images, test_sample_labels=test_sample_labels)

    loss, TSA = model.evaluate(test_sample_images, test_sample_labels)
    print('Watermarked Model Testing Accuracy: ', TSA)
    print('Watermarked Model Testing Loss: ', loss)
```

```
log saved at ./logs
Using real-time data augmentation.
Learning rate:  0.001
Epoch 1/30
398/398 [==============================] - 19s 41ms/step - loss: 1.8579 -
accuracy: 0.3905 - val_loss: 1.7988 - val_accuracy: 0.4235 - lr: 0.0010
Learning rate:  0.001
Epoch 2/30
398/398 [==============================] - 16s 39ms/step - loss: 1.4815 -
accuracy: 0.5188 - val_loss: 2.6490 - val_accuracy: 0.3754 - lr: 0.0010
Learning rate:  0.001
Epoch 3/30
398/398 [==============================] - 16s 39ms/step - loss: 1.2932 -
accuracy: 0.5947 - val_loss: 2.4316 - val_accuracy: 0.3989 - lr: 0.0010
Learning rate:  0.001
Epoch 4/30
398/398 [==============================] - 16s 39ms/step - loss: 1.1620 -
accuracy: 0.6391 - val_loss: 1.5222 - val_accuracy: 0.5429 - lr: 0.0010
Learning rate:  0.001
Epoch 5/30
398/398 [==============================] - 16s 39ms/step - loss: 1.0767 -
accuracy: 0.6721 - val_loss: 1.5486 - val_accuracy: 0.5724 - lr: 0.0010
Learning rate:  0.001
Epoch 6/30
398/398 [==============================] - 16s 39ms/step - loss: 1.0134 -
accuracy: 0.7010 - val_loss: 1.0793 - val_accuracy: 0.6794 - lr: 0.0010
Learning rate:  0.001
Epoch 7/30
398/398 [==============================] - 16s 39ms/step - loss: 0.9481 -
accuracy: 0.7205 - val_loss: 1.1695 - val_accuracy: 0.6594 - lr: 0.0010
Learning rate:  0.001
Epoch 8/30
398/398 [==============================] - 16s 39ms/step - loss: 0.9044 -
accuracy: 0.7362 - val_loss: 1.3734 - val_accuracy: 0.6052 - lr: 0.0010
Learning rate:  0.001
Epoch 9/30
398/398 [==============================] - 16s 39ms/step - loss: 0.8520 -
```

```
accuracy: 0.7565 - val_loss: 1.2076 - val_accuracy: 0.6686 - lr: 0.0010
Learning rate:  0.001
Epoch 10/30
398/398 [==============================] - 16s 40ms/step - loss: 0.8209 -
accuracy: 0.7662 - val_loss: 1.4792 - val_accuracy: 0.6158 - lr: 0.0010
Learning rate:  0.001
Epoch 11/30
398/398 [==============================] - 16s 39ms/step - loss: 0.7952 -
accuracy: 0.7773 - val_loss: 0.9743 - val_accuracy: 0.7311 - lr: 0.0010
Learning rate:  0.001
Epoch 12/30
398/398 [==============================] - 16s 39ms/step - loss: 0.7661 -
accuracy: 0.7877 - val_loss: 0.9387 - val_accuracy: 0.7336 - lr: 0.0010
Learning rate:  0.001
Epoch 13/30
398/398 [==============================] - 16s 39ms/step - loss: 0.7378 -
accuracy: 0.7990 - val_loss: 1.1381 - val_accuracy: 0.6907 - lr: 0.0010
Learning rate:  0.001
Epoch 14/30
398/398 [==============================] - 16s 39ms/step - loss: 0.7186 -
accuracy: 0.8047 - val_loss: 0.9705 - val_accuracy: 0.7380 - lr: 0.0010
Learning rate:  0.001
Epoch 15/30
398/398 [==============================] - 16s 39ms/step - loss: 0.6944 -
accuracy: 0.8126 - val_loss: 1.0087 - val_accuracy: 0.7300 - lr: 0.0010
Learning rate:  0.001
Epoch 16/30
398/398 [==============================] - 16s 39ms/step - loss: 0.6682 -
accuracy: 0.8256 - val_loss: 1.1714 - val_accuracy: 0.6972 - lr: 0.0010
Learning rate:  0.001
Epoch 17/30
398/398 [==============================] - 16s 39ms/step - loss: 0.6558 -
accuracy: 0.8277 - val_loss: 1.2302 - val_accuracy: 0.6772 - lr: 0.0010
Learning rate:  0.001
Epoch 18/30
398/398 [==============================] - 16s 39ms/step - loss: 0.6439 -
accuracy: 0.8333 - val_loss: 1.0973 - val_accuracy: 0.7210 - lr: 0.0010
Learning rate:  0.001
Epoch 19/30
398/398 [==============================] - 16s 39ms/step - loss: 0.6227 -
accuracy: 0.8422 - val_loss: 1.2000 - val_accuracy: 0.7072 - lr: 0.0010
Learning rate:  0.001
Epoch 20/30
398/398 [==============================] - 16s 40ms/step - loss: 0.6058 -
accuracy: 0.8474 - val_loss: 1.0406 - val_accuracy: 0.7354 - lr: 0.0010
Learning rate:  0.001
Epoch 21/30
398/398 [==============================] - 16s 39ms/step - loss: 0.5967 -
```

```
accuracy: 0.8539 - val_loss: 1.0226 - val_accuracy: 0.7463 - lr: 0.0010
Learning rate:  0.001
Epoch 22/30
398/398 [==============================] - 16s 39ms/step - loss: 0.5845 -
accuracy: 0.8570 - val_loss: 0.9245 - val_accuracy: 0.7625 - lr: 0.0010
Learning rate:  0.001
Epoch 23/30
398/398 [==============================] - 16s 40ms/step - loss: 0.5728 -
accuracy: 0.8615 - val_loss: 1.0424 - val_accuracy: 0.7380 - lr: 0.0010
Learning rate:  0.001
Epoch 24/30
398/398 [==============================] - 16s 39ms/step - loss: 0.5623 -
accuracy: 0.8650 - val_loss: 0.8738 - val_accuracy: 0.7832 - lr: 0.0010
Learning rate:  0.001
Epoch 25/30
398/398 [==============================] - 16s 40ms/step - loss: 0.5505 -
accuracy: 0.8712 - val_loss: 0.9544 - val_accuracy: 0.7688 - lr: 0.0010
Learning rate:  0.001
Epoch 26/30
398/398 [==============================] - 16s 39ms/step - loss: 0.5349 -
accuracy: 0.8740 - val_loss: 1.0515 - val_accuracy: 0.7594 - lr: 0.0010
Learning rate:  0.001
Epoch 27/30
398/398 [==============================] - 16s 39ms/step - loss: 0.5292 -
accuracy: 0.8782 - val_loss: 0.9345 - val_accuracy: 0.7797 - lr: 0.0010
Learning rate:  0.001
Epoch 28/30
398/398 [==============================] - 16s 39ms/step - loss: 0.5282 -
accuracy: 0.8782 - val_loss: 1.0616 - val_accuracy: 0.7589 - lr: 0.0010
Learning rate:  0.001
Epoch 29/30
398/398 [==============================] - 16s 39ms/step - loss: 0.5132 -
accuracy: 0.8839 - val_loss: 0.9437 - val_accuracy: 0.7746 - lr: 0.0010
Learning rate:  0.001
Epoch 30/30
398/398 [==============================] - 16s 39ms/step - loss: 0.5018 -
accuracy: 0.8906 - val_loss: 1.0501 - val_accuracy: 0.7494 - lr: 0.0010
141/141 [==============================] - 2s 11ms/step - loss: 0.1894 -
accuracy: 1.0000
Watermarked Model Testing Accuracy:  1.0
Watermarked Model Testing Loss:  0.18940557539463043
```

[ ]: