Images of lego bricks classification

```
import matplotlib.pyplot as plt
import numpy as np
import PIL
import pathlib
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

The dataset

This notebook uses a dataset of about 6379 photos of lego parts from this link https://www.kaggle.com/datasets/joosthazelzet/lego-brick-images/data. The dataset contains five sub-directories, one per class:

```
LEGO_brick_images\
    11214_Bush_3M_friction_with_Cross_axle\
    18651_Cross_Axle_2M_with_Snap_friction\
    2357_Brick_corner_1x2x2\
    3003_Brick_2x2\
    3004_Brick_1x2\
    3005_Brick_1x1\
    3022_Plate_2x2\
    3023_Plate_1x2\
    3024_Plate_1x1\
    3040_Roof_Tile_1x2x45deg\
```

```
3069_Flat_Tile_1x2\
32123_half_Bush\
3673_Peg_2M\
3713_Bush_for_Cross_Axle\
6632_Technic_Lever_3M\
```

```
data_dir = 'Datasets\LEGO_brick_images'
data_dir = pathlib.Path(data_dir).with_suffix('')
```

```
image_count = len(list(data_dir.glob('*/*.png')))
print(image_count)
```

```
6379
```

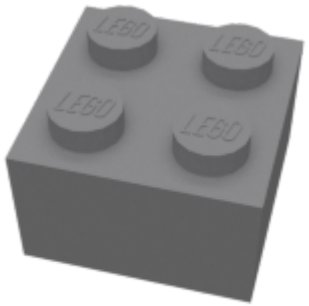
Here are some bricks corner 1x2x2:

```
brick_corner_1x2x2 = list(data_dir.glob('2357_Brick_corner_1x2x2/*'))
PIL.Image.open(str(brick_corner_1x2x2[0]))
```

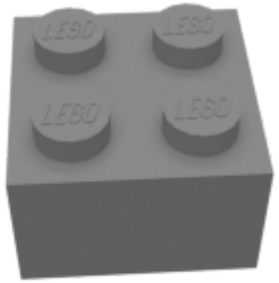
```
PIL.Image.open(str(brick_corner_1x2x2[1]))
```

And some bricks 2x2:

```
brick_2x2 = list(data_dir.glob('3003_Brick_2x2/*'))
PIL.Image.open(str(brick_2x2[0]))
```



```
PIL.Image.open(str(brick_2x2[1]))
```



Create a dataset

```
batch_size = 32
img_height = 360
img_width = 360
```

```
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=4321,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 6379 files belonging to 16 classes.
Using 5104 files for training.
```

```
val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=4321,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

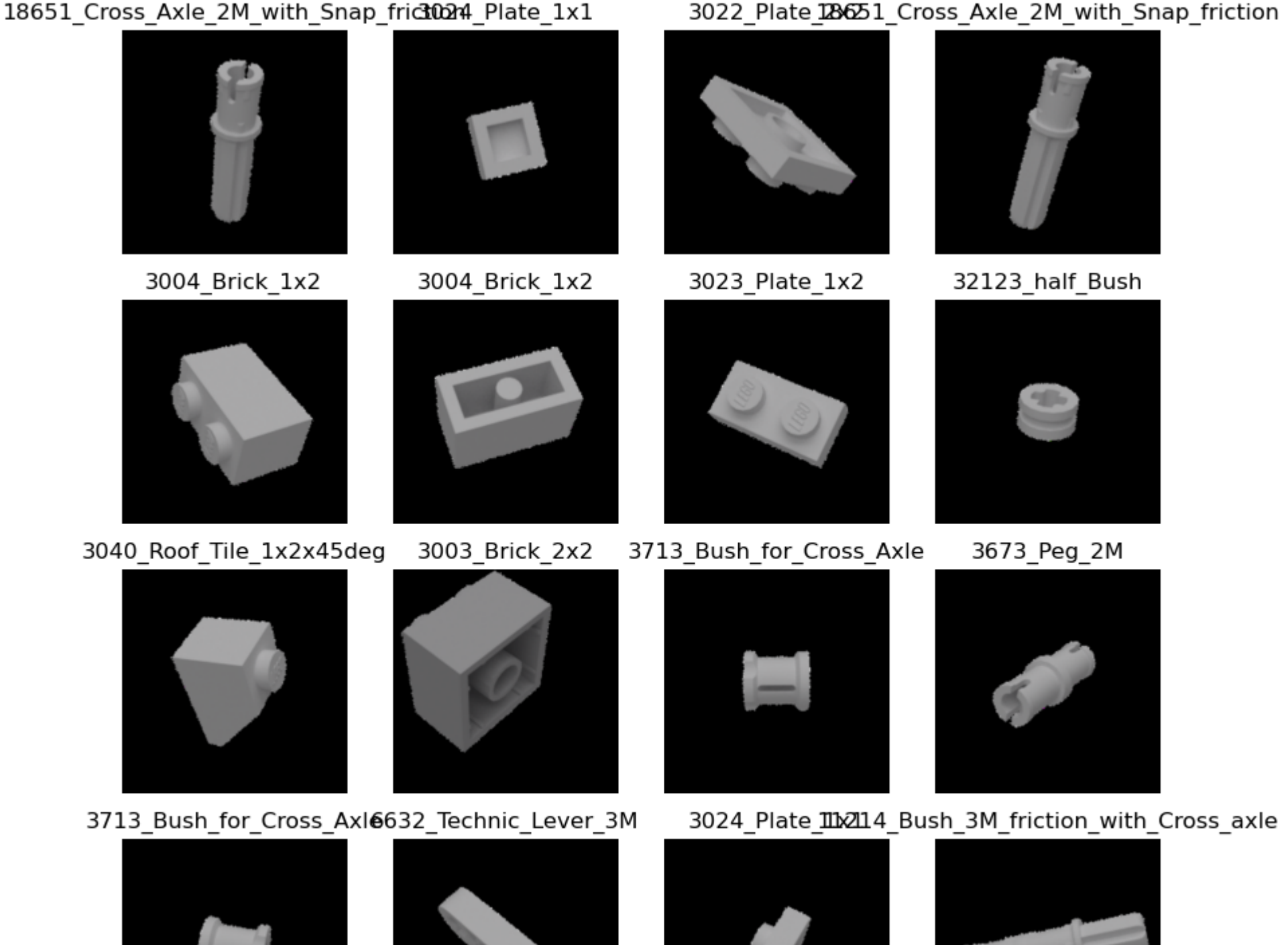
```
Found 6379 files belonging to 16 classes.
Using 1275 files for validation.
```

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

```
['11214_Bush_3M_friction_with_Cross_axle', '18651_Cross_Axle_2M_with_Snap_friction', '2357_Brick_corner_1x2x2', '300
```

Visualize the data

```
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(16):
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```


18651_Cross_Axle_2M_with_Snap_friction
3024_Plate_1x1
3022_Plate_2x2
18651_Cross_Axle_2M_with_Snap_friction
3004_Brick_1x2
3004_Brick_1x2
3023_Plate_1x2
32123_half_Bush
3040_Roof_Tile_1x2x45deg
3003_Brick_2x2
3713_Bush_for_Cross_Axle
3673_Peg_2M
3713_Bush_for_Cross_Axle
6632_Technic_Lever_3M
3024_Plate_1x1
11214_Bush_3M_friction_with_Cross_axle

```
for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

```
(32, 360, 360, 3)
(32,)
```

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

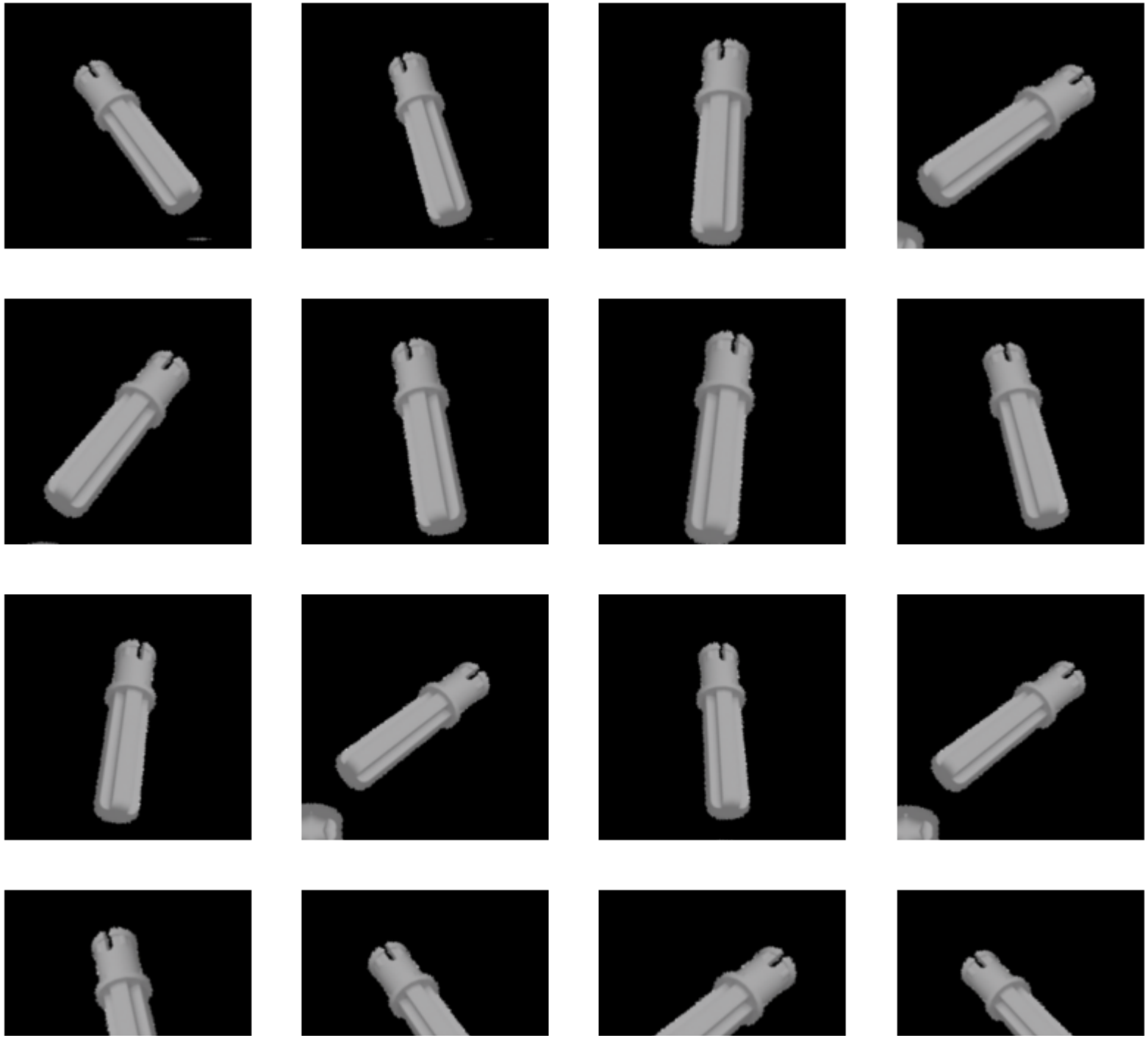
Standardize the data

```
normalization_layer = layers.Rescaling(1./255)
```

```
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
print(np.min(first_image), np.max(first_image))
```

```
0.0 0.9213744
```

```
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",
                      input_shape=(img_height,
                                   img_width,
                                   3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)
num_classes = len(class_names)


plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(16):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```






```
model = Sequential([
  data_augmentation,
  layers.Rescaling(1./255),
  layers.Conv2D(16, 4, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 4, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 4, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(128, 4, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.25),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes, name="outputs")
])
```

Compile and train the model

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

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 sequential (Sequential)     (None, 360, 360, 3)       0

 rescaling_1 (Rescaling)     (None, 360, 360, 3)       0

 conv2d (Conv2D)             (None, 360, 360, 16)      784

 max_pooling2d (MaxPooling2D  (None, 180, 180, 16)     0
 )

 conv2d_1 (Conv2D)           (None, 180, 180, 32)      8224

 max_pooling2d_1 (MaxPooling  (None, 90, 90, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 90, 90, 64)        32832

 max_pooling2d_2 (MaxPooling  (None, 45, 45, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 45, 45, 128)       131200

 max_pooling2d_3 (MaxPooling  (None, 22, 22, 128)      0
 2D)

 dropout (Dropout)           (None, 22, 22, 128)       0

 flatten (Flatten)           (None, 61952)             0

 dense (Dense)               (None, 128)               7929984

 outputs (Dense)             (None, 16)                2064

=================================================================
Total params: 8,105,088
Trainable params: 8,105,088
Non-trainable params: 0
_________________________________________________________________
```

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

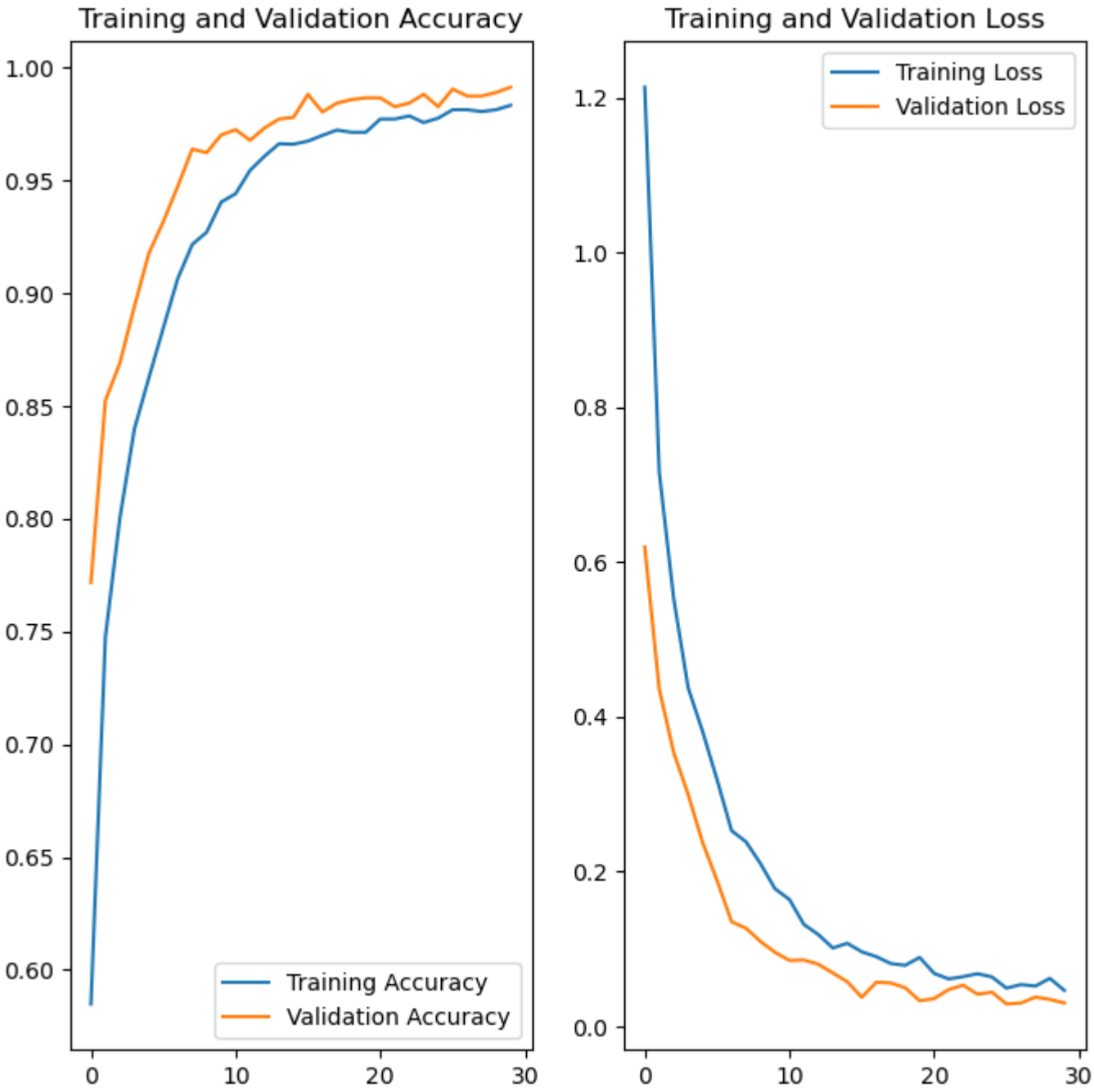
```
Epoch 1/30
160/160 [==============================] - 15s 88ms/step - loss: 1.2133 - accuracy: 0.5848 - val_loss: 0.6194 - val_
Epoch 2/30
160/160 [==============================] - 13s 81ms/step - loss: 0.7160 - accuracy: 0.7476 - val_loss: 0.4360 - val_
Epoch 3/30
160/160 [==============================] - 13s 80ms/step - loss: 0.5526 - accuracy: 0.8004 - val_loss: 0.3538 - val_
Epoch 4/30
160/160 [==============================] - 13s 80ms/step - loss: 0.4375 - accuracy: 0.8397 - val_loss: 0.2996 - val_
Epoch 5/30
160/160 [==============================] - 13s 80ms/step - loss: 0.3810 - accuracy: 0.8625 - val_loss: 0.2376 - val_
Epoch 6/30
160/160 [==============================] - 13s 80ms/step - loss: 0.3187 - accuracy: 0.8846 - val_loss: 0.1886 - val_
Epoch 7/30
160/160 [==============================] - 13s 81ms/step - loss: 0.2530 - accuracy: 0.9065 - val_loss: 0.1354 - val_
Epoch 8/30
160/160 [==============================] - 13s 80ms/step - loss: 0.2386 - accuracy: 0.9216 - val_loss: 0.1270 - val_
Epoch 9/30
160/160 [==============================] - 13s 80ms/step - loss: 0.2104 - accuracy: 0.9271 - val_loss: 0.1099 - val_
Epoch 10/30
160/160 [==============================] - 13s 80ms/step - loss: 0.1780 - accuracy: 0.9404 - val_loss: 0.0959 - val_
Epoch 11/30
160/160 [==============================] - 13s 80ms/step - loss: 0.1642 - accuracy: 0.9442 - val_loss: 0.0855 - val_
Epoch 12/30
160/160 [==============================] - 13s 80ms/step - loss: 0.1319 - accuracy: 0.9545 - val_loss: 0.0860 - val_
Epoch 13/30
160/160 [==============================] - 13s 80ms/step - loss: 0.1188 - accuracy: 0.9608 - val_loss: 0.0805 - val_
Epoch 14/30
160/160 [==============================] - 13s 80ms/step - loss: 0.1015 - accuracy: 0.9663 - val_loss: 0.0693 - val_
Epoch 15/30
160/160 [==============================] - 13s 80ms/step - loss: 0.1076 - accuracy: 0.9661 - val_loss: 0.0579 - val_
Epoch 16/30
160/160 [==============================] - 13s 80ms/step - loss: 0.0966 - accuracy: 0.9675 - val_loss: 0.0379 - val_
Epoch 17/30
```

```
160/160 [==============================] - 13s 80ms/step - loss: 0.0903 - accuracy: 0.9700 - val_loss: 0.0572 - val_
Epoch 18/30
160/160 [==============================] - 13s 80ms/step - loss: 0.0814 - accuracy: 0.9724 - val_loss: 0.0565 - val_
Epoch 19/30
160/160 [==============================] - 13s 81ms/step - loss: 0.0792 - accuracy: 0.9714 - val_loss: 0.0502 - val_
Epoch 20/30
160/160 [==============================] - 13s 81ms/step - loss: 0.0893 - accuracy: 0.9714 - val_loss: 0.0334 - val_
Epoch 21/30
160/160 [==============================] - 13s 81ms/step - loss: 0.0687 - accuracy: 0.9773 - val_loss: 0.0361 - val_
Epoch 22/30
160/160 [==============================] - 13s 81ms/step - loss: 0.0616 - accuracy: 0.9773 - val_loss: 0.0478 - val_
Epoch 23/30
160/160 [==============================] - 13s 81ms/step - loss: 0.0645 - accuracy: 0.9786 - val_loss: 0.0535 - val_
Epoch 24/30
160/160 [==============================] - 13s 81ms/step - loss: 0.0683 - accuracy: 0.9757 - val_loss: 0.0418 - val_
Epoch 25/30
160/160 [==============================] - 13s 80ms/step - loss: 0.0641 - accuracy: 0.9777 - val_loss: 0.0445 - val_
Epoch 26/30
160/160 [==============================] - 13s 81ms/step - loss: 0.0497 - accuracy: 0.9814 - val_loss: 0.0291 - val_
Epoch 27/30
160/160 [==============================] - 13s 81ms/step - loss: 0.0543 - accuracy: 0.9814 - val_loss: 0.0304 - val_
Epoch 28/30
160/160 [==============================] - 13s 80ms/step - loss: 0.0523 - accuracy: 0.9806 - val_loss: 0.0381 - val_
Epoch 29/30
160/160 [                              ] - 13s 80ms/step - loss: 0.0622 - accuracy: 0.9814 - val_loss: 0.0251 - val_
```

Visualize training results

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
```

```
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')
    plt.show()
```


Training and Validation Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
10
20
30
Training Accuracy
Validation Accuracy
Training and Validation Loss
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
Training Loss
Validation Loss

Predict on new data

```
brick_1x1_path = 'Datasets/LEGO_new/3005_Brick_1x1/0014.png'
img = tf.keras.utils.load_img(
    brick_1x1_path, target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)
predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

```
 This image most likely belongs to 3005_Brick_1x1 with a 99.99 percent confidence.
```

```
peg_2m_path = 'Datasets/LEGO_new/3673_Peg_2M/0002.png'
img = tf.keras.utils.load_img(
    peg_2m_path, target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)
predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

```
This image most likely belongs to 3673_Peg_2M with a 99.74 percent confidence.
```

Use TensorFlow Lite

TensorFlow Lite is a set of tools that enables on-device machine learning by helping developers run their models on mobile, embedded, and edge devices.