CNN Rock Paper Scissors

Notebook adapted from the Rock Paper Scissors (using Convolutional Neural Network) notebook.
Modified by: Gábor Major
Last Modified date: 2025-03-18

Import libraries.

```
import tensorflow as tf
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
import numpy as np
import platform
import datetime
import os
import math
import random

print('Python version:', platform.python_version())
print('Tensorflow version:', tf.__version__)
print('Keras version:', tf.keras.__version__)
```

```
2025-03-21 10:01:10.601958: I
external/local_xla/xla/tsl/cuda/cudart_stub.cc:32] Could not find cuda
drivers on your machine, GPU will not be used.
2025-03-21 10:01:10.685873: I
external/local_xla/xla/tsl/cuda/cudart_stub.cc:32] Could not find cuda
drivers on your machine, GPU will not be used.
2025-03-21 10:01:10.719811: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:467] Unable to
register cuFFT factory: Attempting to register factory for plugin
cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
E0000 00:00:1742551270.787600     216 cuda_dnn.cc:8579] Unable to
register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
E0000 00:00:1742551270.801565     216 cuda_blas.cc:1407] Unable to
register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
W0000 00:00:1742551270.904685     216 computation_placer.cc:177]
computation placer already registered. Please check linkage and avoid
linking the same target more than once.
W0000 00:00:1742551270.904705     216 computation_placer.cc:177]
computation placer already registered. Please check linkage and avoid
linking the same target more than once.
```

```
W0000 00:00:1742551270.904706     216 computation_placer.cc:177]
computation placer already registered. Please check linkage and avoid
linking the same target more than once.
W0000 00:00:1742551270.904707     216 computation_placer.cc:177]
computation placer already registered. Please check linkage and avoid
linking the same target more than once.
2025-03-21 10:01:10.911306: I
tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow
binary is optimized to use available CPU instructions in performance-
critical operations.
To enable the following instructions: AVX2 FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.

Python version: 3.12.5
Tensorflow version: 2.19.0
Keras version: 3.9.0
```

Load Data

The data used is a collection of images of hands in the three different states of rock, paper or scissors.
The data is downloaded from the TensorFlow datasets.
Each image is 300 by 300 pixel RGB images. The training set contains 2,520 images and the testing set contains 372 images.

```
DATASET_NAME = 'rock_paper_scissors'

(dataset_train_raw, dataset_test_raw), dataset_info = tfds.load(
    name=DATASET_NAME,
    data_dir='tmp',
    with_info=True,
    as_supervised=True,
    split=[tfds.Split.TRAIN, tfds.Split.TEST],
)

2025-03-21 10:01:14.822108: E
external/local_xla/xla/stream_executor/cuda/cuda_platform.cc:51]
failed call to cuInit: INTERNAL: CUDA error: Failed call to cuInit:
UNKNOWN ERROR (303)

print('Raw train dataset:', dataset_train_raw)
print('Raw train dataset size:', len(list(dataset_train_raw)), '\n')
print('Raw test dataset:', dataset_test_raw)
print('Raw test dataset size:', len(list(dataset_test_raw)), '\n')
print(dataset_info)

2025-03-21 10:01:14.951174: I
tensorflow/core/kernels/data/tf_record_dataset_op.cc:387] The default
```

```
buffer size is 262144, which is overridden by the user specified
`buffer_size` of 8388608

Raw train dataset: <_PrefetchDataset
element_spec=(TensorSpec(shape=(300, 300, 3), dtype=tf.uint8,
name=None), TensorSpec(shape=(), dtype=tf.int64, name=None))>

2025-03-21 10:01:16.089786: I
tensorflow/core/framework/local_rendezvous.cc:407] Local rendezvous is
aborting with status: OUT_OF_RANGE: End of sequence

Raw train dataset size: 2520

Raw test dataset: <_PrefetchDataset
element_spec=(TensorSpec(shape=(300, 300, 3), dtype=tf.uint8,
name=None), TensorSpec(shape=(), dtype=tf.int64, name=None))>
Raw test dataset size: 372

tfds.core.DatasetInfo(
    name='rock_paper_scissors',
    full_name='rock_paper_scissors/3.0.0',
    description="""
    Images of hands playing rock, paper, scissor game.
    """,
    homepage='http://laurencemoroney.com/rock-paper-scissors-dataset',
    data_dir='tmp/rock_paper_scissors/3.0.0',
    file_format=tfrecord,
    download_size=219.53 MiB,
    dataset_size=219.23 MiB,
    features=FeaturesDict({
        'image': Image(shape=(300, 300, 3), dtype=uint8),
        'label': ClassLabel(shape=(), dtype=int64, num_classes=3),
    }),
    supervised_keys=('image', 'label'),
    disable_shuffling=False,
    nondeterministic_order=False,
    splits={
        'test': <SplitInfo num_examples=372, num_shards=1>,
        'train': <SplitInfo num_examples=2520, num_shards=2>,
    },
    citation="""@ONLINE {rps,
    author = "Laurence Moroney",
    title = "Rock, Paper, Scissors Dataset",
    month = "feb",
    year = "2019",
    url = "http://laurencemoroney.com/rock-paper-scissors-dataset"
    }""",
)
```

```
2025-03-21 10:01:16.308387: I
tensorflow/core/framework/local_rendezvous.cc:407] Local rendezvous is
aborting with status: OUT_OF_RANGE: End of sequence
```

```
NUM_TRAIN_EXAMPLES = dataset_info.splits['train'].num_examples
NUM_TEST_EXAMPLES = dataset_info.splits['test'].num_examples
NUM_CLASSES = dataset_info.features['label'].num_classes

print('Number of TRAIN examples:', NUM_TRAIN_EXAMPLES)
print('Number of TEST examples:', NUM_TEST_EXAMPLES)
print('Number of label classes:', NUM_CLASSES)
```

```
Number of TRAIN examples: 2520
Number of TEST examples: 372
Number of label classes: 3
```

```
INPUT_IMG_SIZE_ORIGINAL = dataset_info.features['image'].shape[0]
INPUT_IMG_SHAPE_ORIGINAL = dataset_info.features['image'].shape

INPUT_IMG_SIZE_REDUCED = INPUT_IMG_SIZE_ORIGINAL // 2
INPUT_IMG_SHAPE_REDUCED = (
    INPUT_IMG_SIZE_REDUCED,
    INPUT_IMG_SIZE_REDUCED,
    INPUT_IMG_SHAPE_ORIGINAL[2]
)

# Here we may switch between bigger or smaller image sized that we
will train our model on.
INPUT_IMG_SIZE = INPUT_IMG_SIZE_REDUCED
INPUT_IMG_SHAPE = INPUT_IMG_SHAPE_REDUCED

print('Input image size (original):', INPUT_IMG_SIZE_ORIGINAL)
print('Input image shape (original):', INPUT_IMG_SHAPE_ORIGINAL)
print('\n')
print('Input image size (reduced):', INPUT_IMG_SIZE_REDUCED)
print('Input image shape (reduced):', INPUT_IMG_SHAPE_REDUCED)
print('\n')
print('Input image size:', INPUT_IMG_SIZE)
print('Input image shape:', INPUT_IMG_SHAPE)
```

```
Input image size (original): 300
Input image shape (original): (300, 300, 3)


Input image size (reduced): 150
Input image shape (reduced): (150, 150, 3)


Input image size: 150
Input image shape: (150, 150, 3)
```

```
# Function to convert label ID to labels string.
get_label_name = dataset_info.features['label'].int2str
print(get_label_name(0))
print(get_label_name(1))
print(get_label_name(2))
```

```
rock
paper
scissors
```

Explore Data

Look at the data that is available in different formats.
First look at the raw data, which shows the colour of the pixels from 0 to 255. Because these images have three channels one for each colour, red, green and blue, the numbers represent the intensity of that specific colour at each pixel for each image.

```
# Explore what values are used to represent the image.
(first_image, first_lable) = list(dataset_train_raw.take(1))[0]
print('Label:', first_lable.numpy(), '\n')
print('Image shape:', first_image.numpy().shape, '\n')
print(first_image.numpy())
```

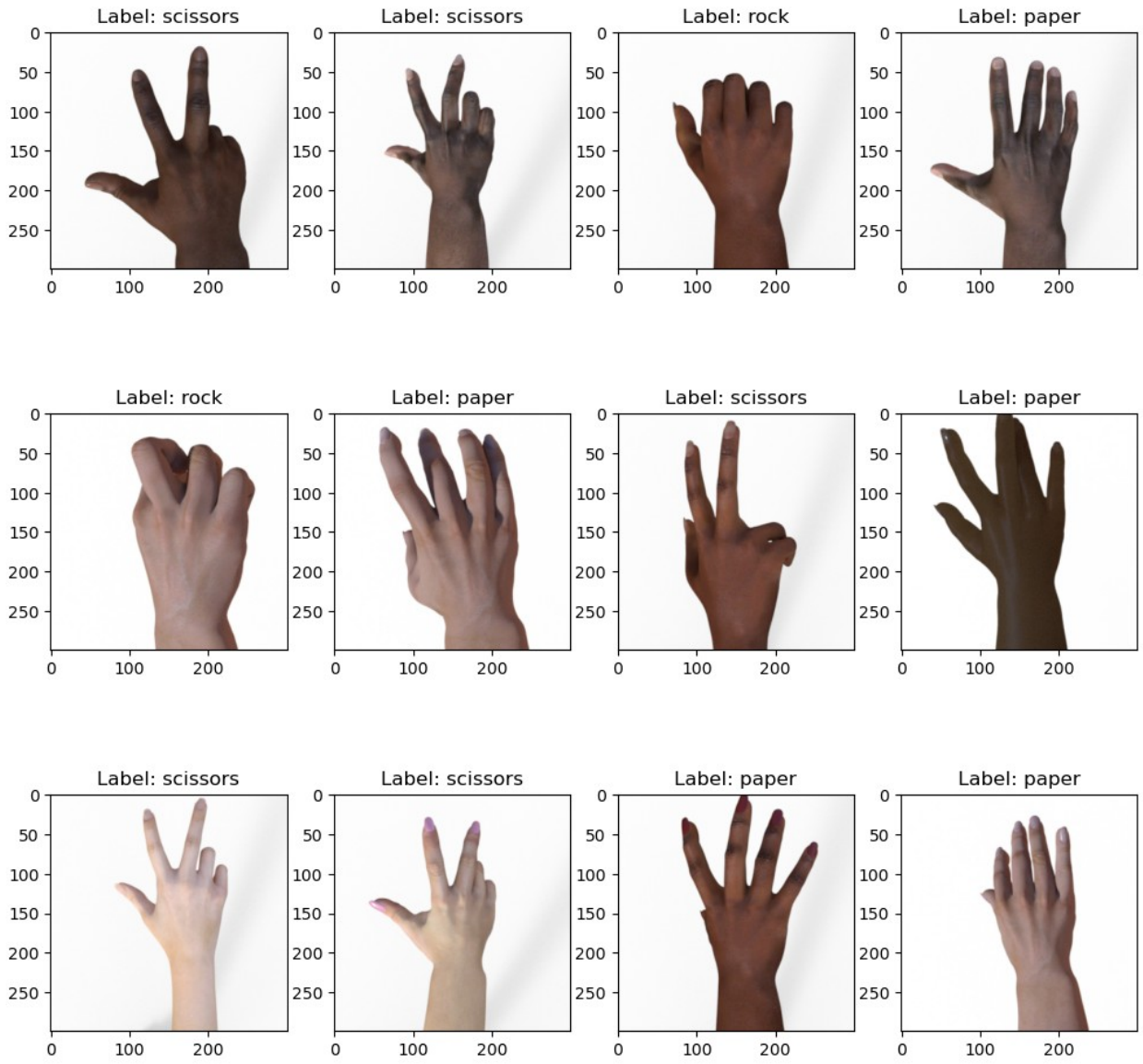
```
Label: 2

Image shape: (300, 300, 3)

[[[254 254 254]
  [253 253 253]
  [254 254 254]
  ...
  [251 251 251]
  [250 250 250]
  [250 250 250]]

 [[254 254 254]
  [254 254 254]
  [253 253 253]
  ...
  [250 250 250]
  [251 251 251]
  [249 249 249]]

 [[254 254 254]
  [254 254 254]
  [254 254 254]
  ...
  [251 251 251]
  [250 250 250]
```

```
  [252 252 252]]

 ...

 [[252 252 252]
  [251 251 251]
  [252 252 252]
  ...
  [247 247 247]
  [249 249 249]
  [248 248 248]]

 [[253 253 253]
  [253 253 253]
  [251 251 251]
  ...
  [248 248 248]
  [248 248 248]
  [248 248 248]]

 [[252 252 252]
  [253 253 253]
  [252 252 252]
  ...
  [248 248 248]
  [247 247 247]
  [250 250 250]]]
```

Next look at the actual images.

```
def preview_dataset(dataset):
    plt.figure(figsize=(12, 12))
    plot_index = 0
    for features in dataset.take(12):
        (image, label) = features
        plot_index += 1
        plt.subplot(3, 4, plot_index)
        # plt.axis('Off')
        label = get_label_name(label.numpy())
        plt.title('Label: %s' % label)
        plt.imshow(image.numpy())
# Explore raw training dataset images.
preview_dataset(dataset_train_raw)
```

```
2025-03-21 10:01:16.601594: I
tensorflow/core/framework/local_rendezvous.cc:407] Local rendezvous is
aborting with status: OUT_OF_RANGE: End of sequence
```



Pre-process Data

The images are preprocessed which in this case is scaling them down by two and applying a normalisation to the values to remap them from 0 to 255 to 0 to 1.

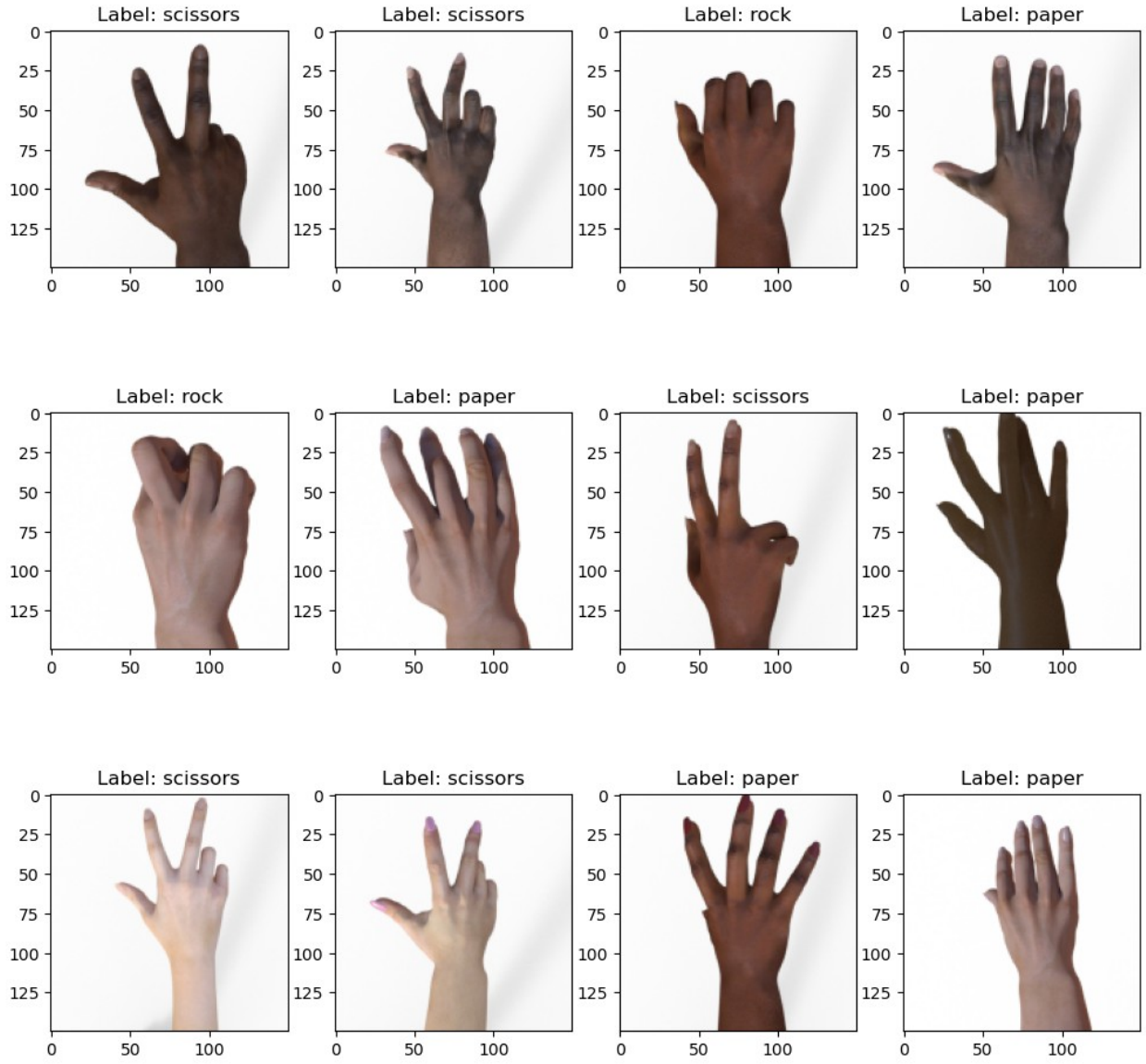
```
def format_example(image, label):
    # Make image color values to be float.
    image = tf.cast(image, tf.float32)
    # Make image color values to be in [0..1] range.
    image = image / 255.
    # Make sure that image has a right size
    image = tf.image.resize(image, [INPUT_IMG_SIZE, INPUT_IMG_SIZE])
    return image, label
```

```
dataset_train = dataset_train_raw.map(format_example)
dataset_test = dataset_test_raw.map(format_example)

# Explore what values are used to represent the image.
(first_image, first_lable) = list(dataset_train.take(1))[0]
print('Label:', first_lable.numpy(), '\n')
print('Image shape:', first_image.numpy().shape, '\n')
print(first_image.numpy())
```

```
Label: 2

Image shape: (150, 150, 3)

[[[0.995098   0.995098   0.995098  ]
  [0.995098   0.995098   0.995098  ]
  [0.995098   0.995098   0.995098  ]
  ...
  [0.9852941  0.9852941  0.9852941 ]
  [0.9843137  0.9843137  0.9843137 ]
  [0.98039216 0.98039216 0.98039216]]

 [[0.99607843 0.99607843 0.99607843]
  [0.995098   0.995098   0.995098  ]
  [0.995098   0.995098   0.995098  ]
  ...
  [0.98333335 0.98333335 0.98333335]
  [0.9813726  0.9813726  0.9813726 ]
  [0.98333335 0.98333335 0.98333335]]

 [[0.99607843 0.99607843 0.99607843]
  [0.9941176  0.9941176  0.9941176 ]
  [0.9941176  0.9941176  0.9941176 ]
  ...
  [0.9852941  0.9852941  0.9852941 ]
  [0.9852941  0.9852941  0.9852941 ]
  [0.9813726  0.9813726  0.9813726 ]]

 ...

 [[0.9862745  0.9862745  0.9862745 ]
  [0.98725486 0.98725486 0.98725486]
  [0.9882353  0.9882353  0.9882353 ]
  ...
  [0.9705882  0.9705882  0.9705882 ]
  [0.97352946 0.97352946 0.97352946]
  [0.9754902  0.9754902  0.9754902 ]]

 [[0.9882353  0.9882353  0.9882353 ]
  [0.98725486 0.98725486 0.98725486]
  [0.9862745  0.9862745  0.9862745 ]
```

```
  ...
  [0.9676471  0.9676471  0.9676471 ]
  [0.97156864 0.97156864 0.97156864]
  [0.972549   0.972549   0.972549  ]]

 [[0.9911765  0.9911765  0.9911765 ]
  [0.9862745  0.9862745  0.9862745 ]
  [0.9882353  0.9882353  0.9882353 ]
  ...
  [0.97352946 0.97352946 0.97352946]
  [0.9705882  0.9705882  0.9705882 ]
  [0.97352946 0.97352946 0.97352946]]]
```

```
# Explore preprocessed training dataset images.
preview_dataset(dataset_train)
```

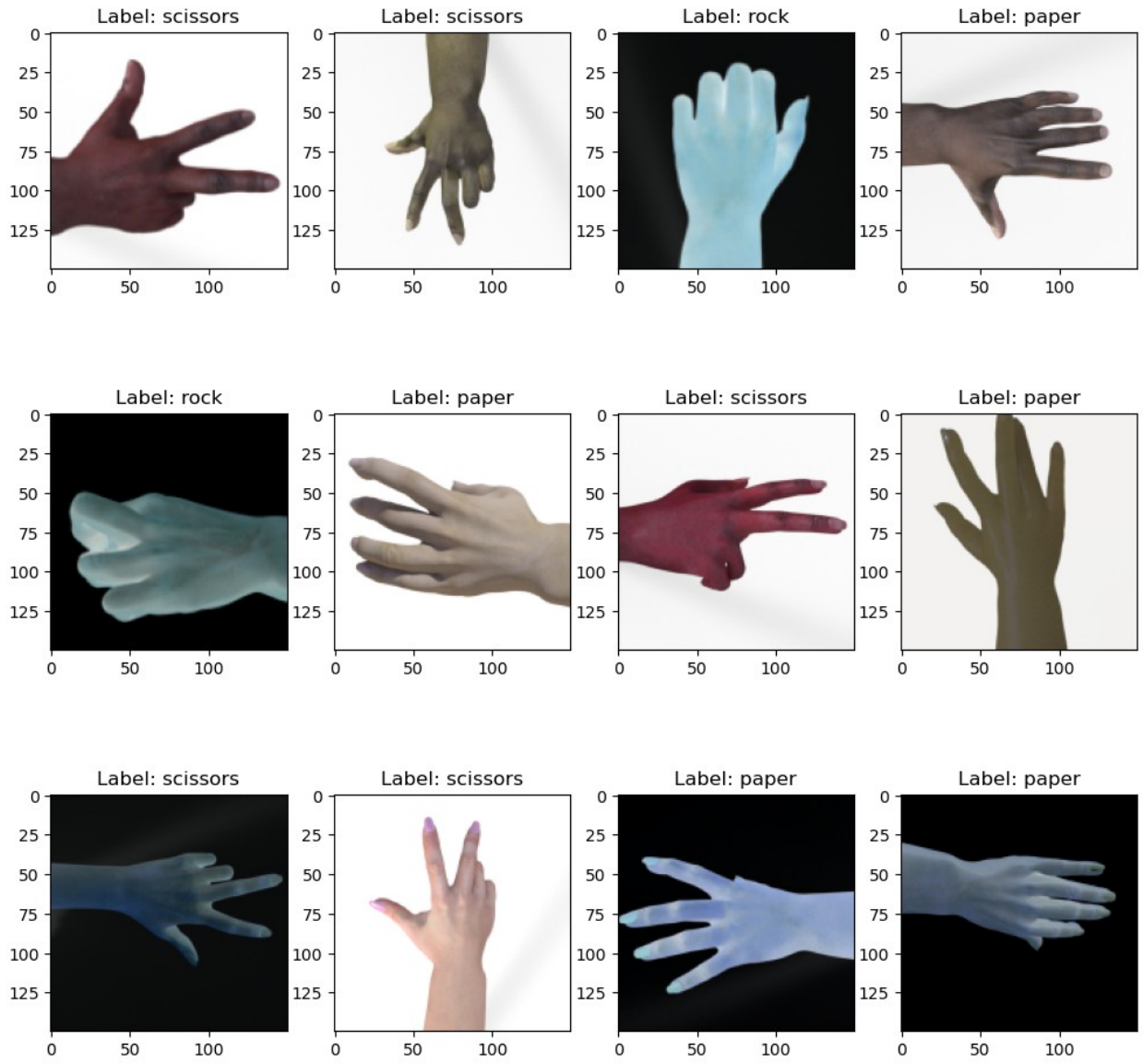


Data Augmentation

To help with overfitting data is augmentated from the existing dataset.
The images are flipped, changed colours, rotated, changed zoom level and inverted.

```
def augment_flip(image: tf.Tensor) -> tf.Tensor:
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_flip_up_down(image)
    return image

def augment_colour(image: tf.Tensor) -> tf.Tensor:
    image = tf.image.random_hue(image, max_delta=0.08)
    image = tf.image.random_saturation(image, lower=0.7, upper=1.3)
    image = tf.image.random_brightness(image, 0.05)
    image = tf.image.random_contrast(image, lower=0.8, upper=1)
    image = tf.clip_by_value(image, clip_value_min=0,
clip_value_max=1)
    return image

def augment_rotation(image: tf.Tensor) -> tf.Tensor:
    # Rotate 0, 90, 180, 270 degrees
    return tf.image.rot90(
        image,
        tf.random.uniform(shape=[], minval=0, maxval=4,
dtype=tf.int32)
    )

def augment_inversion(image: tf.Tensor) -> tf.Tensor:
    random = tf.random.uniform(shape=[], minval=0, maxval=1)
    if random > 0.5:
        image = tf.math.multiply(image, -1)
        image = tf.math.add(image, 1)
    return image

def augment_zoom(image: tf.Tensor, min_zoom=0.8, max_zoom=1.0) ->
tf.Tensor:
    image_width, image_height, image_colors = image.shape
    crop_size = (image_width, image_height)

    # Generate crop settings, ranging from a 1% to 20% crop.
    scales = list(np.arange(min_zoom, max_zoom, 0.01))
    boxes = np.zeros((len(scales), 4))

    for i, scale in enumerate(scales):
        x1 = y1 = 0.5 - (0.5 * scale)
        x2 = y2 = 0.5 + (0.5 * scale)
        boxes[i] = [x1, y1, x2, y2]

    def random_crop(img):
        # Create different crops for an image
```

```
        crops = tf.image.crop_and_resize(
            [img],
            boxes=boxes,
            box_indices=np.zeros(len(scales)),
            crop_size=crop_size
        )
        # Return a random crop
        return crops[tf.random.uniform(shape=[], minval=0,
maxval=len(scales), dtype=tf.int32)]

    choice = tf.random.uniform(shape=[], minval=0., maxval=1.,
dtype=tf.float32)

    # Only apply cropping 50% of the time
    return tf.cond(choice < 0.5, lambda: image, lambda:
random_crop(image))

def augment_data(image, label):
    image = augment_flip(image)
    image = augment_colour(image)
    image = augment_rotation(image)
    image = augment_zoom(image)
    image = augment_inversion(image)
    return image, label

dataset_train_augmented = dataset_train.map(augment_data)
# Explore augmented training dataset.
preview_dataset(dataset_train_augmented)
```



Data Shuffling and Batching

Next the data is shuffled to avoid and bias from grouping images. The data is also split up into multiple batches to speed up with training.

```
BATCH_SIZE = 32

dataset_train_augmented_shuffled = dataset_train_augmented.shuffle(
    buffer_size=NUM_TRAIN_EXAMPLES
)

dataset_train_augmented_shuffled = dataset_train_augmented.batch(
    batch_size=BATCH_SIZE
)
```

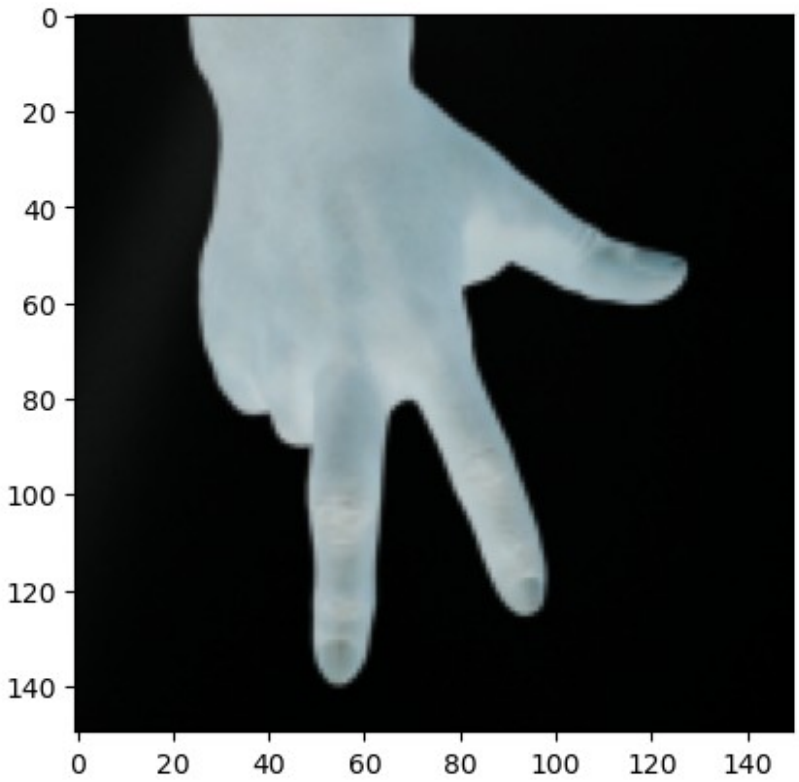
```
# Prefetch will enable the input pipeline to asynchronously fetch
batches while your model is training.
dataset_train_augmented_shuffled =
dataset_train_augmented_shuffled.prefetch(
    buffer_size=tf.data.experimental.AUTOTUNE
)

dataset_test_shuffled = dataset_test.batch(BATCH_SIZE)
print(dataset_train_augmented_shuffled)
print(dataset_test_shuffled)
```

```
<_PrefetchDataset element_spec=(TensorSpec(shape=(None, 150, 150, 3),
dtype=tf.float32, name=None), TensorSpec(shape=(None,),
dtype=tf.int64, name=None))>
<_BatchDataset element_spec=(TensorSpec(shape=(None, 150, 150, 3),
dtype=tf.float32, name=None), TensorSpec(shape=(None,),
dtype=tf.int64, name=None))>
```

```
# Debugging the batches using conversion to Numpy arrays.
batches = tfds.as_numpy(dataset_train_augmented_shuffled)
for batch in batches:
    image_batch, label_batch = batch
    print('Label batch shape:', label_batch.shape, '\n')
    print('Image batch shape:', image_batch.shape, '\n')
    print('Label batch:', label_batch, '\n')

    for batch_item_index in range(len(image_batch)):
        print('First batch image:', image_batch[batch_item_index], '\
n')
        plt.imshow(image_batch[batch_item_index])
        plt.show()
        # Break to shorten the output.
        break
    # Break to shorten the output.
    break
```

```
Label batch shape: (32,)

Image batch shape: (32, 150, 150, 3)

Label batch: [2 2 0 1 0 1 2 1 2 2 1 1 2 1 1 1 1 1 1 1 1 0 0 0 0 1 1 2
2 2 0 0]

First batch image: [[[0.01972544 0.02220392 0.02279264]
  [0.02234781 0.02482629 0.025415  ]
  [0.01972544 0.02220392 0.02279264]
  ...
  [0.00661361 0.00909215 0.00968087]
  [0.00836194 0.01084048 0.01142919]
  [0.00399131 0.00646985 0.0070585 ]]
```

```
 [[0.02059954 0.02307808 0.02366674]
  [0.02147365 0.02395219 0.02454084]
  [0.02497011 0.02744865 0.02803731]
  ...
  [0.00836194 0.01084048 0.01142919]
  [0.00748783 0.00996637 0.01055503]
  [0.00661361 0.00909215 0.00968087]]

 [[0.01797724 0.02045572 0.02104443]
  [0.01972544 0.02220392 0.02279264]
  [0.02234781 0.02482629 0.025415  ]
  ...
  [0.00661361 0.00909215 0.00968087]
  [0.00748783 0.00996637 0.01055503]
  [0.00836194 0.01084048 0.01142919]]

 ...

 [[0.01273245 0.01521099 0.0157997 ]
  [0.00923604 0.01171458 0.01230323]
  [0.00923604 0.01171458 0.01230323]
  ...
  [0.001369   0.00384748 0.00443619]
  [0.001369   0.00384748 0.00443619]
  [0.         0.00209916 0.00268787]]

 [[0.01098424 0.01346278 0.0140515 ]
  [0.01273245 0.01521099 0.0157997 ]
  [0.01098424 0.01346278 0.0140515 ]
  ...
  [0.00049484 0.00297344 0.00356209]
  [0.00049484 0.00297344 0.00356209]
  [0.         0.00209916 0.00268787]]

 [[0.01360655 0.01608509 0.0166738 ]
  [0.01011014 0.01258868 0.01317739]
  [0.00923604 0.01171458 0.01230323]
  ...
  [0.00049484 0.00297344 0.00356209]
  [0.00049484 0.00297344 0.00356209]
  [0.00049484 0.00297344 0.00356209]]]
```



Build the Model

A sequential Keras model is created with 12 layers.
First there are four layers of convultions.
These convolutions consits of a Convolution2D layer and a MaxPooling2D layer.
Next a Flatten layer is used to translate the 2D matrix into a 1D vactor.
A Dropout layer is used to randomly change the values to 0 to prevent overfitting.
The data is then fed into a Dense layer of 512 neurons. Finally the Output layer is used with 3 neurons for each of the results.

```
model = tf.keras.models.Sequential()

# First convolution.
model.add(tf.keras.layers.Convolution2D(
    input_shape=INPUT_IMG_SHAPE,
    filters=64,
    kernel_size=3,
    activation=tf.keras.activations.relu
))
model.add(tf.keras.layers.MaxPooling2D(
    pool_size=(2, 2),
    strides=(2, 2)
))

# Second convolution.
```

```
model.add(tf.keras.layers.Convolution2D(
    filters=64,
    kernel_size=3,
    activation=tf.keras.activations.relu
))
model.add(tf.keras.layers.MaxPooling2D(
    pool_size=(2, 2),
    strides=(2, 2)
))

# Third convolution.
model.add(tf.keras.layers.Convolution2D(
    filters=128,
    kernel_size=3,
    activation=tf.keras.activations.relu
))
model.add(tf.keras.layers.MaxPooling2D(
    pool_size=(2, 2),
    strides=(2, 2)
))

# Fourth convolution.
model.add(tf.keras.layers.Convolution2D(
    filters=128,
    kernel_size=3,
    activation=tf.keras.activations.relu
))
model.add(tf.keras.layers.MaxPooling2D(
    pool_size=(2, 2),
    strides=(2, 2)
))

# Flatten the results to feed into dense layers.
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dropout(0.5))

# 512 neuron dense layer.
model.add(tf.keras.layers.Dense(
    units=512,
    activation=tf.keras.activations.relu
))

# Output layer.
model.add(tf.keras.layers.Dense(
    units=NUM_CLASSES,
    activation=tf.keras.activations.softmax
))
```

```
/home/gabor/.var/app/org.jupyter.JupyterLab/config/jupyterlab-
desktop/jlab_server/lib/python3.12/site-packages/keras/src/layers/
```

```
convolutional/base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

model.summary()

Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━
┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━
┩
│ conv2d (Conv2D)                      │ (None, 148, 148, 64)        │
1,792 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ max_pooling2d (MaxPooling2D)         │ (None, 74, 74, 64)          │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ conv2d_1 (Conv2D)                    │ (None, 72, 72, 64)          │
36,928 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ max_pooling2d_1 (MaxPooling2D)       │ (None, 36, 36, 64)          │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ conv2d_2 (Conv2D)                    │ (None, 34, 34, 128)         │
73,856 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ max_pooling2d_2 (MaxPooling2D)       │ (None, 17, 17, 128)         │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ conv2d_3 (Conv2D)                    │ (None, 15, 15, 128)         │
147,584 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ max_pooling2d_3 (MaxPooling2D)       │ (None, 7, 7, 128)           │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ flatten (Flatten)                    │ (None, 6272)                │
```

```
0 │
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ dropout (Dropout)                   │ (None, 6272)                │
0 │
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ dense (Dense)                       │ (None, 512)                 │
3,211,776 │
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ dense_1 (Dense)                     │ (None, 3)                   │
1,539 │
└─────────────────────────────────────┴─────────────────────────────┴──────────
──────┘

 Total params: 3,473,475 (13.25 MB)

 Trainable params: 3,473,475 (13.25 MB)

 Non-trainable params: 0 (0.00 B)
```

Compile the Model

```
rmsprop_optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.001)

model.compile(
    optimizer=rmsprop_optimizer,
    loss=tf.keras.losses.sparse_categorical_crossentropy,
    metrics=['accuracy']
)
```

Train the Model

```
steps_per_epoch = NUM_TRAIN_EXAMPLES // BATCH_SIZE
validation_steps = NUM_TEST_EXAMPLES // BATCH_SIZE

print('steps_per_epoch:', steps_per_epoch)
print('validation_steps:', validation_steps)

# Preparing callbacks.
os.makedirs('logs/fit', exist_ok=True)
tensorboard_log_dir = 'logs/fit/' +
datetime.datetime.now().strftime('%Y%m%d-%H%M%S')
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=tensorboard_log_dir,
    histogram_freq=1
)

os.makedirs('tmp/checkpoints', exist_ok=True)
```

```
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath='tmp/checkpoints/weights.{epoch:02d}-
{val_loss:.2f}.keras'
)

early_stopping_callback = tf.keras.callbacks.EarlyStopping(
    patience=5,
    monitor='val_accuracy'
    # monitor='val_loss'
)

training_history = model.fit(
    x=dataset_train_augmented_shuffled.repeat(),
    validation_data=dataset_test_shuffled.repeat(),
    epochs=15,
    steps_per_epoch=steps_per_epoch,
    validation_steps=validation_steps,
    callbacks=[
        # model_checkpoint_callback,
        # early_stopping_callback,
        tensorboard_callback
    ],
    verbose=1
)
```

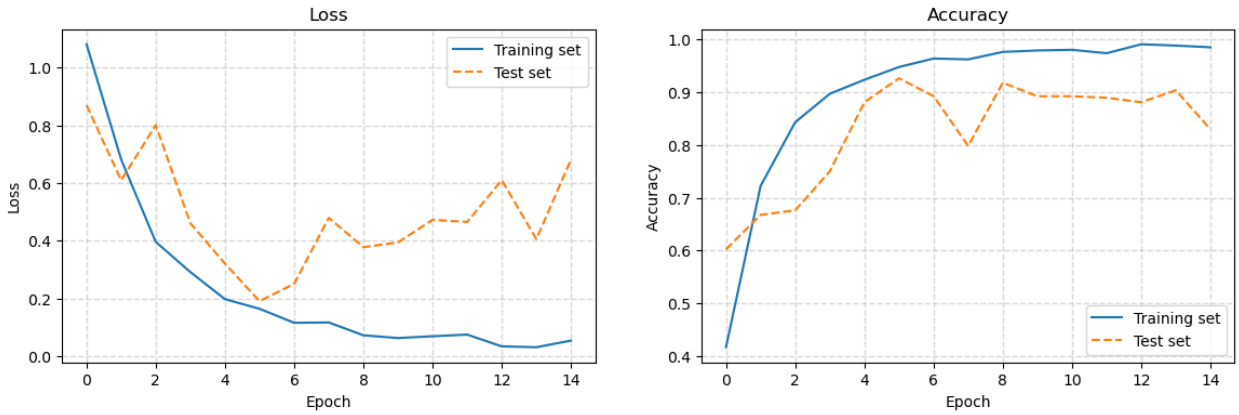
```
steps_per_epoch: 78
validation_steps: 11
Epoch 1/15

2025-03-21 10:01:22.285306: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83]
Allocation of 179437568 exceeds 10% of free system memory.
2025-03-21 10:01:22.348031: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83]
Allocation of 44859392 exceeds 10% of free system memory.
2025-03-21 10:01:22.366940: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83]
Allocation of 42467328 exceeds 10% of free system memory.
2025-03-21 10:01:22.649505: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83]
Allocation of 42467328 exceeds 10% of free system memory.
2025-03-21 10:01:22.680655: W
external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83]
Allocation of 44859392 exceeds 10% of free system memory.

78/78 ━━━━━━━━━━━━━━━━━━━━ 69s 874ms/step - accuracy: 0.3550 - loss:
1.1659 - val_accuracy: 0.6023 - val_loss: 0.8696
Epoch 2/15
78/78 ━━━━━━━━━━━━━━━━━━━━ 66s 846ms/step - accuracy: 0.6648 - loss:
0.7887 - val_accuracy: 0.6676 - val_loss: 0.6098
```

```
Epoch 3/15
78/78 ━━━━━━━━━━━━━━━━━━━━ 74s 958ms/step - accuracy: 0.8232 - loss:
0.4476 - val_accuracy: 0.6761 - val_loss: 0.8009
Epoch 4/15
78/78 ━━━━━━━━━━━━━━━━━━━━ 63s 809ms/step - accuracy: 0.8841 - loss:
0.3170 - val_accuracy: 0.7500 - val_loss: 0.4608
Epoch 5/15
78/78 ━━━━━━━━━━━━━━━━━━━━ 66s 842ms/step - accuracy: 0.9207 - loss:
0.2097 - val_accuracy: 0.8807 - val_loss: 0.3210
Epoch 6/15
78/78 ━━━━━━━━━━━━━━━━━━━━ 68s 874ms/step - accuracy: 0.9451 - loss:
0.1879 - val_accuracy: 0.9261 - val_loss: 0.1907
Epoch 7/15
78/78 ━━━━━━━━━━━━━━━━━━━━ 68s 871ms/step - accuracy: 0.9513 - loss:
0.1448 - val_accuracy: 0.8920 - val_loss: 0.2507
Epoch 8/15
78/78 ━━━━━━━━━━━━━━━━━━━━ 62s 791ms/step - accuracy: 0.9605 - loss:
0.1255 - val_accuracy: 0.7983 - val_loss: 0.4795
Epoch 9/15
78/78 ━━━━━━━━━━━━━━━━━━━━ 61s 780ms/step - accuracy: 0.9750 - loss:
0.0742 - val_accuracy: 0.9176 - val_loss: 0.3779
Epoch 10/15
78/78 ━━━━━━━━━━━━━━━━━━━━ 61s 782ms/step - accuracy: 0.9810 - loss:
0.0656 - val_accuracy: 0.8920 - val_loss: 0.3942
Epoch 11/15
78/78 ━━━━━━━━━━━━━━━━━━━━ 61s 781ms/step - accuracy: 0.9837 - loss:
0.0618 - val_accuracy: 0.8920 - val_loss: 0.4726
Epoch 12/15
78/78 ━━━━━━━━━━━━━━━━━━━━ 61s 779ms/step - accuracy: 0.9729 - loss:
0.0704 - val_accuracy: 0.8892 - val_loss: 0.4652
Epoch 13/15
78/78 ━━━━━━━━━━━━━━━━━━━━ 61s 779ms/step - accuracy: 0.9886 - loss:
0.0377 - val_accuracy: 0.8807 - val_loss: 0.6101
Epoch 14/15
13/78 ━━━━━━━━━━━━━━━━━━━━ 47s 727ms/step - accuracy: 0.9938 - loss:
0.0126

2025-03-21 10:15:30.588183: W
tensorflow/core/kernels/data/prefetch_autotuner.cc:52] Prefetch
autotuner tried to allocate 8640256 bytes after encountering the first
element of size 8640256 bytes.This already causes the autotune ram
budget to be exceeded. To stay within the ram budget, either increase
the ram budget or reduce element size

78/78 ━━━━━━━━━━━━━━━━━━━━ 61s 784ms/step - accuracy: 0.9871 - loss:
0.0337 - val_accuracy: 0.9034 - val_loss: 0.4058
Epoch 15/15
13/78 ━━━━━━━━━━━━━━━━━━━━ 50s 779ms/step - accuracy: 0.9858 - loss:
0.0483
```

```
2025-03-21 10:16:32.577736: W
tensorflow/core/kernels/data/prefetch_autotuner.cc:52] Prefetch
autotuner tried to allocate 270008 bytes after encountering the first
element of size 270008 bytes.This already causes the autotune ram
budget to be exceeded. To stay within the ram budget, either increase
the ram budget or reduce element size
2025-03-21 10:16:32.894340: W
tensorflow/core/kernels/data/prefetch_autotuner.cc:52] Prefetch
autotuner tried to allocate 8640256 bytes after encountering the first
element of size 8640256 bytes.This already causes the autotune ram
budget to be exceeded. To stay within the ram budget, either increase
the ram budget or reduce element size

78/78 ━━━━━━━━━━━━━━━━━━━━ 62s 787ms/step - accuracy: 0.9818 - loss:
0.0790 - val_accuracy: 0.8295 - val_loss: 0.6812
```

Plot the loss function and accuracy of the model against the training time.

```python
def render_training_history(training_history):
    loss = training_history.history['loss']
    val_loss = training_history.history['val_loss']

    accuracy = training_history.history['accuracy']
    val_accuracy = training_history.history['val_accuracy']

    plt.figure(figsize=(14, 4))

    plt.subplot(1, 2, 1)
    plt.title('Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.plot(loss, label='Training set')
    plt.plot(val_loss, label='Test set', linestyle='--')
    plt.legend()
    plt.grid(linestyle='--', linewidth=1, alpha=0.5)

    plt.subplot(1, 2, 2)
    plt.title('Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.plot(accuracy, label='Training set')
    plt.plot(val_accuracy, label='Test set', linestyle='--')
    plt.legend()
    plt.grid(linestyle='--', linewidth=1, alpha=0.5)

    plt.show()

render_training_history(training_history)
```



Evaluate Model

The model is compared on its accuracy of the training and testing sets.

```
train_loss, train_accuracy = model.evaluate(
    x=dataset_train.batch(BATCH_SIZE).take(NUM_TRAIN_EXAMPLES)
)

test_loss, test_accuracy = model.evaluate(
    x=dataset_test.batch(BATCH_SIZE).take(NUM_TEST_EXAMPLES)
)

print('Training loss: ', train_loss)
print('Training accuracy: ', train_accuracy)
print('\n')
print('Test loss: ', test_loss)
print('Test accuracy: ', test_accuracy)
```

```
79/79 ━━━━━━━━━━━━━━━━━━━━ 16s 202ms/step - accuracy: 0.9875 - loss:
0.0452
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 206ms/step - accuracy: 0.8300 - loss:
0.7049
Training loss:  0.045085880905389786
Training accuracy:  0.9876984357833862


Test loss:  0.6882150173187256
Test accuracy:  0.8279569745063782
```

Save the Model

```
model_name = 'models/rock_paper_scissors_cnn.keras'
model.save(model_name)
```