Practical 1

MATRIX MULTIPLICATION, EIGEN VECTORS, EIGENVALUE COMPUTATION USING TENSORFLOW

[ ] ↳ 7 cells hidden

Practical 2

Deep Forward Network For XOR

[ ] ↳ 8 cells hidden

PRACTICAL 3A

CLASSIFICATION USING DNN

[ ] ↳ 11 cells hidden

PRACTICAL 3B

BINARY CLASSIFICATION USING MLP

[ ] ↳ 7 cells hidden

PRACTICAL 4

PREDICTING THE PROBABILITY OF THE CLASS

[ ] ↳ 5 cells hidden

PRACTICAL 5A

CNN FOR CIFAR10 IMAGES

[ ] ↳ 9 cells hidden

PRACTICAL 5B

IMAGE CLASSIFICATION

[ ] ↳ *27 cells hidden*

PRACTICAL 5C

DATA AUGMENTATION

Data augmentation is a strategy that enables practitioners to significantly increase the diversity of data available for training models, without actually collecting new data. Data augmentation techniques such as cropping, padding, and horizontal flipping are commonly used to train large neural networks.

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds
```

```
from tensorflow.keras import layers
(train_ds, val_ds, test_ds), metadata = tfds.load(
    'tf_flowers',
    split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
    with_info=True,
    as_supervised=True,
)
```

```
Downloading and preparing dataset 218.21 MiB (download: 218.21 MiB, generated: 221.83 MiB, tot
Dl Completed...: 100%                                    5/5 [00:03<00:00, 1.57 file/s]
Dataset tf_flowers downloaded and prepared to ~/tensorflow_datasets/tf_flowers/3.0.1. Subseque
```

```
num_classes = metadata.features['label'].num_classes
print(num_classes)
```

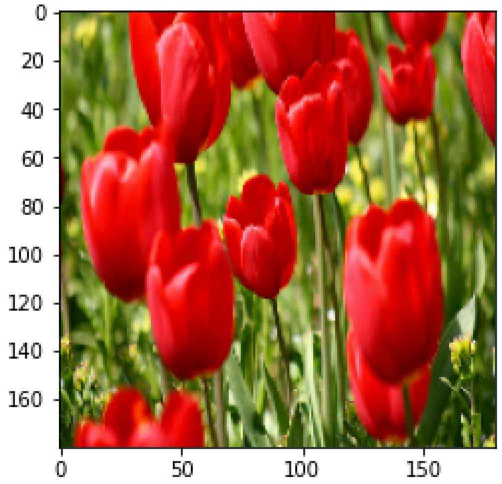
```
5
```

```
get_label_name = metadata.features['label'].int2str

image, label = next(iter(train_ds))
_ = plt.imshow(image)
_ = plt.title(get_label_name(label))
```


```
IMG_SIZE = 180

resize_and_rescale = tf.keras.Sequential([
  layers.Resizing(IMG_SIZE, IMG_SIZE),
  layers.Rescaling(1./255)
])
result = resize_and_rescale(image)
_ = plt.imshow(result)
```

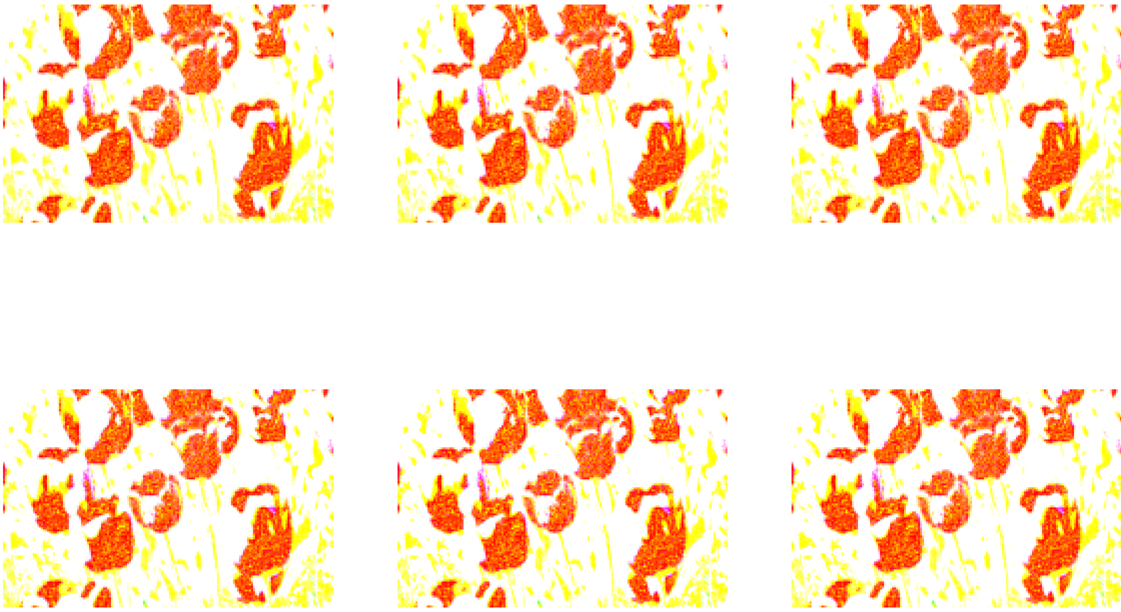



```
print("Min and max pixel values:", result.numpy().min(), result.numpy().max())
```

```
Min and max pixel values: 0.0 1.0
```

```
data_augmentation = tf.keras.Sequential([
  layers.RandomFlip("horizontal_and_vertical"),
  layers.RandomRotation(0.2),
])
# Add the image to a batch.
image = tf.expand_dims(image, 0)
plt.figure(figsize=(10, 10))
for i in range(9):
  augmented_image = data_augmentation(image)
  ax = plt.subplot(3, 3, i + 1)
  plt.imshow(augmented_image[0])
  plt.axis("off")
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..
```



```
model = tf.keras.Sequential([
  # Add the preprocessing layers you created earlier.
  resize_and_rescale,
  data_augmentation,
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  # Rest of your model.
])
```

```
aug_ds = train_ds.map(
  lambda x, y: (resize_and_rescale(x, training=True), y))
batch_size = 32
AUTOTUNE = tf.data.AUTOTUNE

def prepare(ds, shuffle=False, augment=False):
  # Resize and rescale all datasets.
  ds = ds.map(lambda x, y: (resize_and_rescale(x), y),
              num_parallel_calls=AUTOTUNE)

  if shuffle:
    ds = ds.shuffle(1000)

  # Batch all datasets.
  ds = ds.batch(batch_size)

  # Use data augmentation only on the training set.
  if augment:
    ds = ds.map(lambda x, y: (data_augmentation(x, training=True), y),
                num_parallel_calls=AUTOTUNE)
```

```
  # Use buffered prefetching on all datasets.
  return ds.prefetch(buffer_size=AUTOTUNE)

train_ds = prepare(train_ds, shuffle=True, augment=True)
val_ds = prepare(val_ds)
test_ds = prepare(test_ds)

model = tf.keras.Sequential([
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
epochs=5
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/5
92/92 [==============================] - 65s 680ms/step - loss: 1.4067 - accuracy: 0.3971 - va
Epoch 2/5
92/92 [==============================] - 61s 652ms/step - loss: 1.1268 - accuracy: 0.5364 - va
Epoch 3/5
92/92 [==============================] - 62s 663ms/step - loss: 1.0204 - accuracy: 0.5950 - va
Epoch 4/5
92/92 [==============================] - 61s 652ms/step - loss: 0.9607 - accuracy: 0.6213 - va
Epoch 5/5
92/92 [==============================] - 61s 659ms/step - loss: 0.8926 - accuracy: 0.6543 - va
```

```
loss, acc = model.evaluate(test_ds)
print("Accuracy", acc)
```

```
12/12 [==============================] - 3s 199ms/step - loss: 0.8842 - accuracy: 0.6512
Accuracy 0.6512261629104614
```

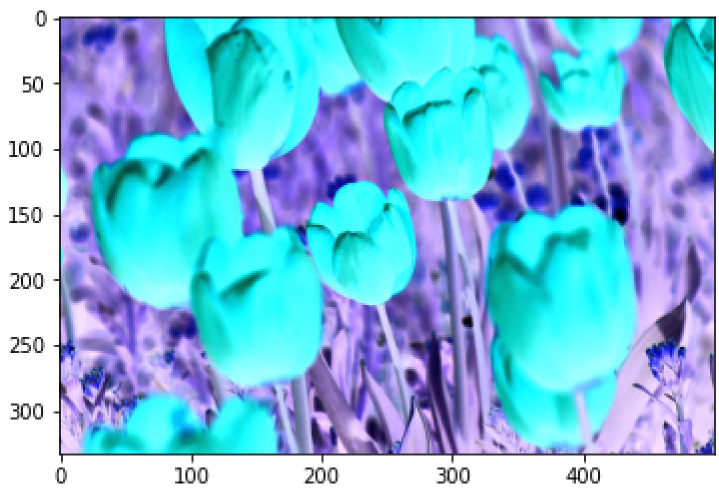
```
def random_invert_img(x, p=0.5):
  if  tf.random.uniform([]) < p:
    x = (255-x)
  else:
    x
  return x
def random_invert(factor=0.5):
  return layers.Lambda(lambda x: random_invert_img(x, factor))

random_invert = random_invert()
plt.figure(figsize=(10, 10))
```

```
for i in range(9):
  augmented_image = random_invert(image)
  ax = plt.subplot(3, 3, i + 1)
  plt.imshow(augmented_image[0].numpy().astype("uint8"))
  plt.axis("off")
```

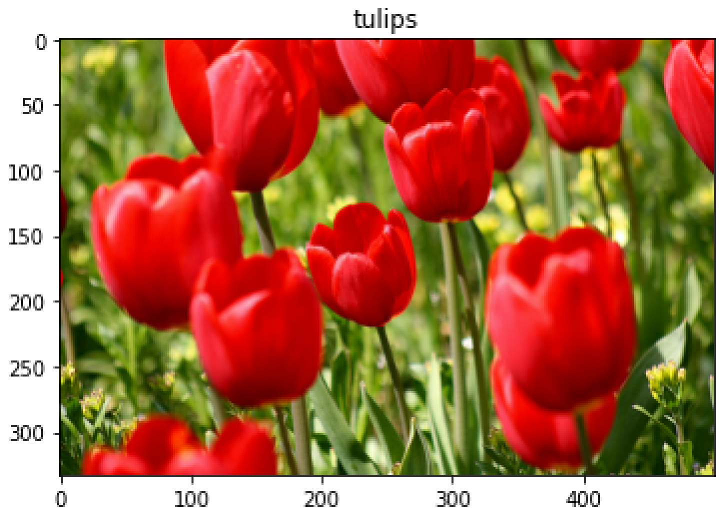
```
class RandomInvert(layers.Layer):
  def __init__(self, factor=0.5, **kwargs):
    super().__init__(**kwargs)
    self.factor = factor

  def call(self, x):
    return random_invert_img(x)
_ = plt.imshow(RandomInvert()(image)[0])
```



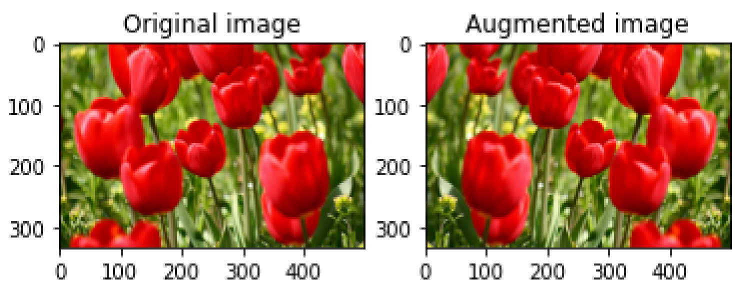

```
(train_ds, val_ds, test_ds), metadata = tfds.load(
```

```
    'tf_flowers',
    split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
    with_info=True,
    as_supervised=True,
)
image, label = next(iter(train_ds))
_ = plt.imshow(image)
_ = plt.title(get_label_name(label))
```



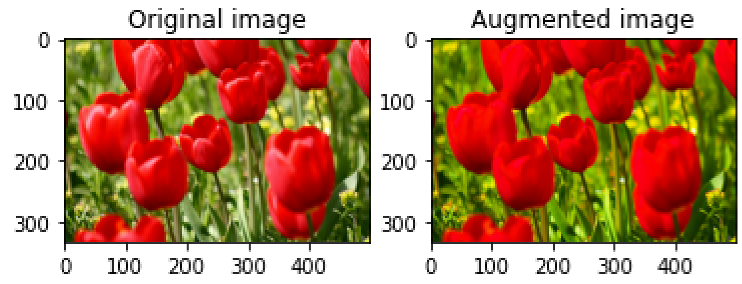

```
def visualize(original, augmented):
  fig = plt.figure()
  plt.subplot(1,2,1)
  plt.title('Original image')
  plt.imshow(original)

  plt.subplot(1,2,2)
  plt.title('Augmented image')
  plt.imshow(augmented)
flipped = tf.image.flip_left_right(image)
visualize(image, flipped)
```

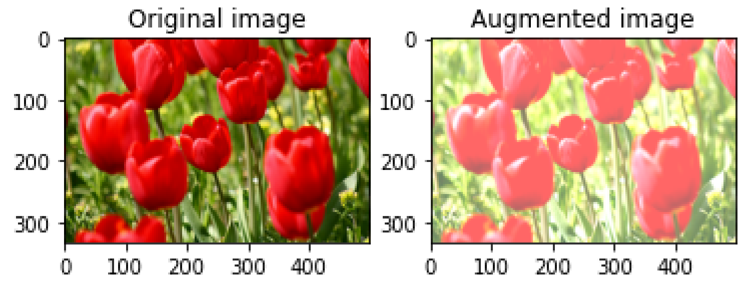



```
grayscaled = tf.image.rgb_to_grayscale(image)
visualize(image, tf.squeeze(grayscaled))
_ = plt.colorbar()
```

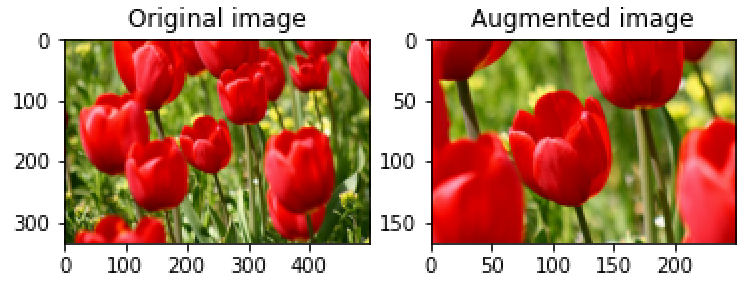

```
saturated = tf.image.adjust_saturation(image, 3)
visualize(image, saturated)
```




```
bright = tf.image.adjust_brightness(image, 0.4)
visualize(image, bright)
```




```
cropped = tf.image.central_crop(image, central_fraction=0.5)
visualize(image,cropped)
```




```
rotated = tf.image.rot90(image)
visualize(image, rotated)
```

Augmented image

PRACTICAL 6

BUILDING RNN USING SINGLE NEURON

[ ] ↳ *9 cells hidden*

PRACTICAL 7

NLP CORPUS

[ ] ↳ *92 cells hidden*

PRACTICAL 8

Lemmatization, Stemming, Tokenization, Stopwords

[ ] ↳ *46 cells hidden*

PRACTICAL 9

One-Hot Encoding, Bag of Words, N-grams, TF-IDF

[ ] ↳ *11 cells hidden*

PRACTICAL 10

Word Embedding

[ ] ↳ *11 cells hidden*

other ipynb files

cfg - https://colab.research.google.com/drive/1y9yIwn6X8ZyN52y8tA6M2v8mDeoDFwOz?usp=sharing
text to speech - https://colab.research.google.com/drive/1mR6gv2Yr5lYJpb6T_MdQfFlwJcjkPZ-l?usp=sharing