In [1]:
```
#@title Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

Data augmentation

View on TensorFlow.org (https://www.tensorflow.org/tutorials/images/data_augmentation)

Run in Google Colab (https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/data_augmentation.ipynb)

View source on GitHub (https://github.com/tensorflow/docs/blob/master/site/en/tutorials/images/data_augmentation.ipynb)

Download notebook (https://storage.googleapis.com/tensorflow_docs/docs/site/en/tutorials/images/data_augmentation.ipynb)

Overview

This tutorial demonstrates manual image manipulations and augmentation using `tf.image` .

Data augmentation is a common technique to improve results and avoid overfitting, see Overfitting and Underfitting (../keras/overfit_and_underfit.ipynb) for others.

Setup

In [2]:
```
!pip install -q git+https://github.com/tensorflow/docs
```

In [3]:
```
import urllib

import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras import layers
AUTOTUNE = tf.data.experimental.AUTOTUNE

import tensorflow_docs as tfdocs
import tensorflow_docs.plots

import tensorflow_datasets as tfds

import PIL.Image

import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['figure.figsize'] = (12, 5)

import numpy as np
```

Let's check the data augmentation features on an image and then augment a whole dataset later to train a model.

Download [this image (https://commons.wikimedia.org/wiki/File:Felis_catus-cat_on_snow.jpg)](https://commons.wikimedia.org/wiki/File:Felis_catus-cat_on_snow.jpg), by Von.grzanka, for augmentation.

In [4]:
```
image_path = tf.keras.utils.get_file("cat.jpg", "https://storage.googleapis.
com/download.tensorflow.org/example_images/320px-Felis_catus-cat_on_snow.jp
g")
PIL.Image.open(image_path)
```

Out[4]:

Read and decode the image to tensor format.

In [5]:
```
image_string=tf.io.read_file(image_path)
image=tf.image.decode_jpeg(image_string,channels=3)
```

A function to visualize and compare the original and augmented image side by side.

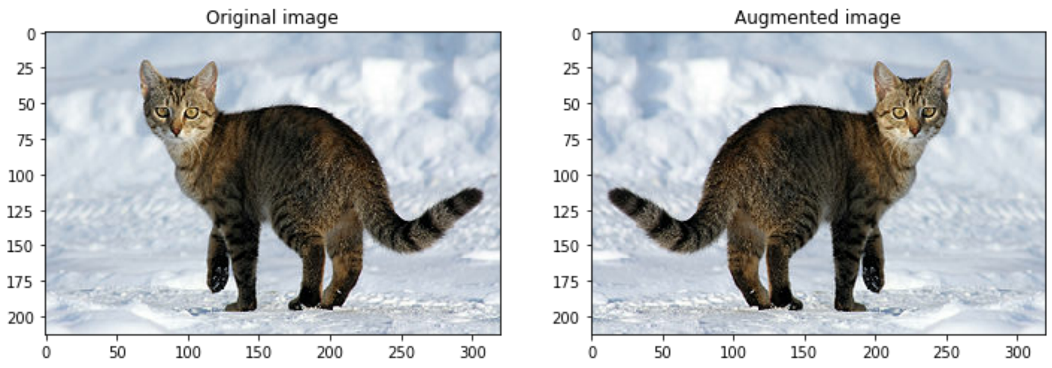
In [6]:
```
def visualize(original, augmented):
  fig = plt.figure()
  plt.subplot(1,2,1)
  plt.title('Original image')
  plt.imshow(original)

  plt.subplot(1,2,2)
  plt.title('Augmented image')
  plt.imshow(augmented)
```

Augment a single image

Flipping the image

Flip the image either vertically or horizontally.

In [7]:
```
flipped = tf.image.flip_left_right(image)
visualize(image, flipped)
```

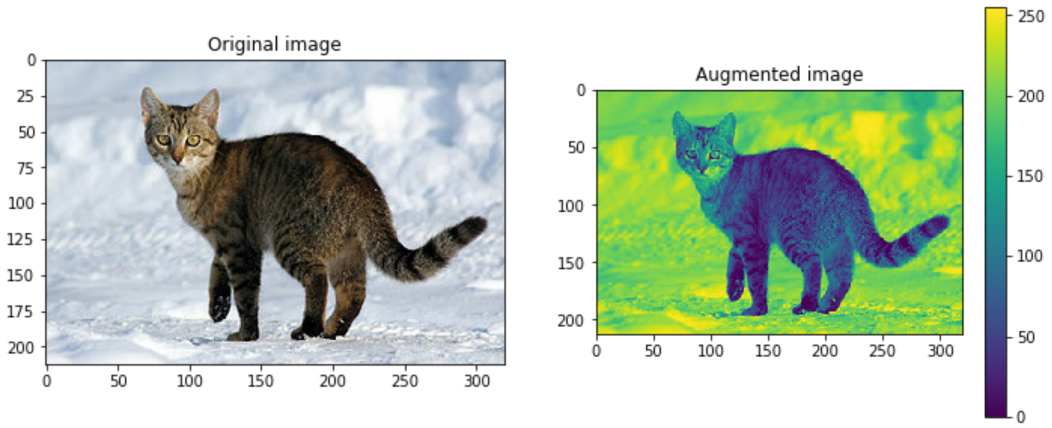


Grayscale the image

Grayscale an image.

In [8]:
```
grayscaled = tf.image.rgb_to_grayscale(image)
visualize(image, tf.squeeze(grayscaled))
plt.colorbar()
```

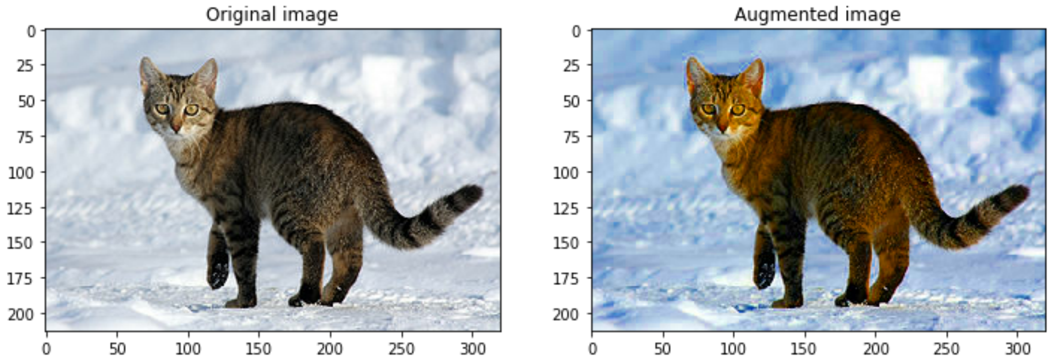
Out[8]: <matplotlib.colorbar.Colorbar at 0x1693a591dc8>



Saturate the image

Saturate an image by providing a saturation factor.

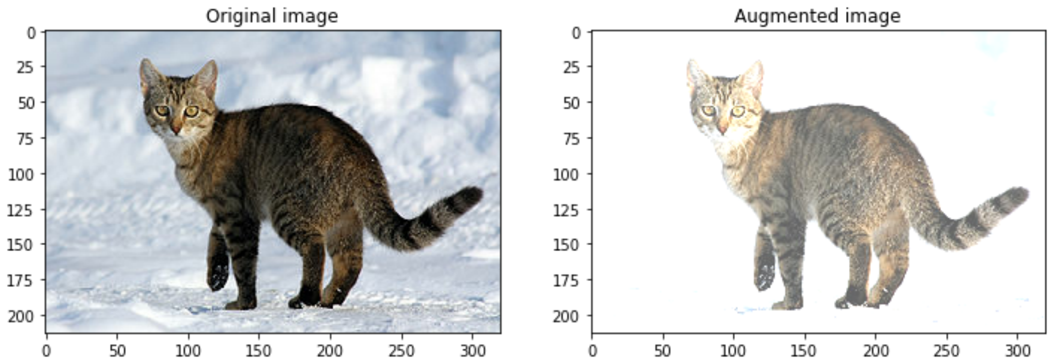
In [9]:
```
saturated = tf.image.adjust_saturation(image, 3)
visualize(image, saturated)
```



Change image brightness

Change the brightness of image by providing a brightness factor.

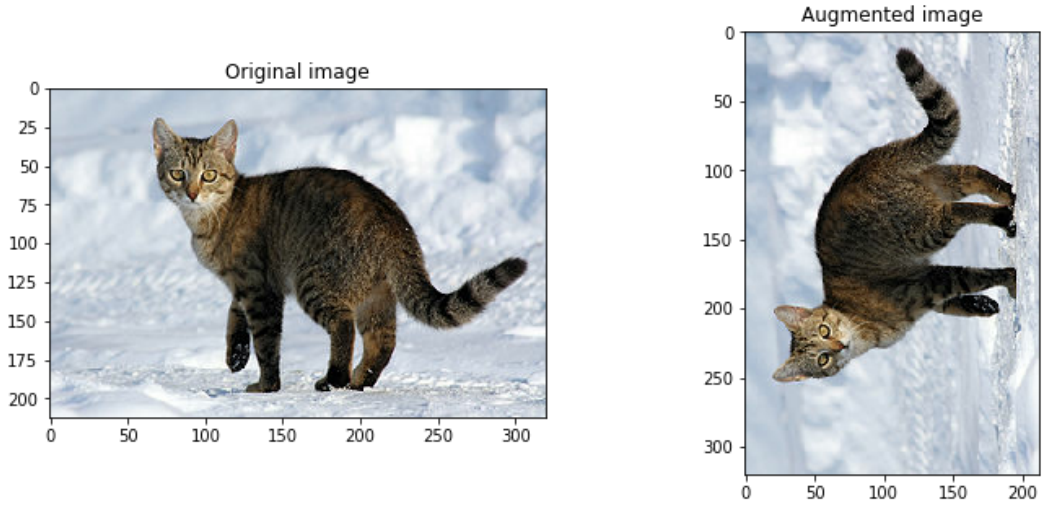
In [10]:
```
bright = tf.image.adjust_brightness(image, 0.4)
visualize(image, bright)
```



Rotate the image

Rotate an image by 90 degrees.

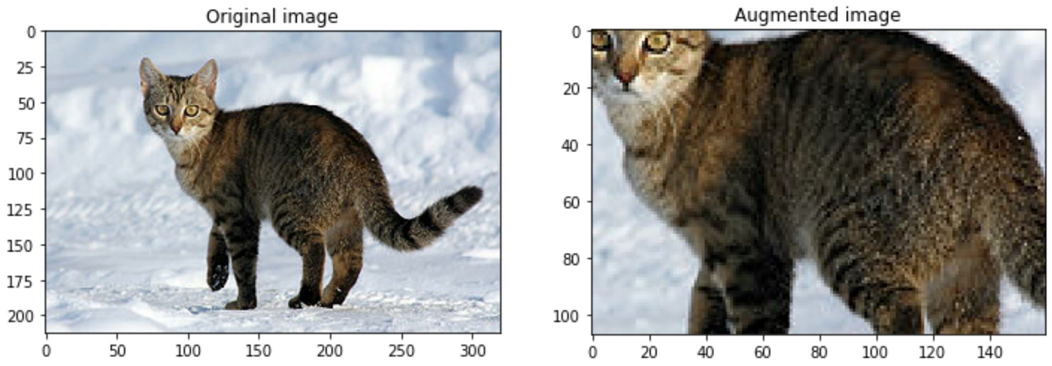
In [11]:
```
rotated = tf.image.rot90(image)
visualize(image, rotated)
```



Center crop the image

Crop the image from center upto the image part you desire.

In [12]:
```
cropped = tf.image.central_crop(image, central_fraction=0.5)
visualize(image,cropped)
```



See the `tf.image` reference for details about available augmentation options.

Augment a dataset and train a model with it

Train a model on an augmented dataset.

Note: The problem solved here is somewhat artificial. It trains a densely connected network to be shift invariant by jittering the input images. It's much more efficient to use convolutional layers instead.

In [13]:
```
dataset, info =  tfds.load('mnist', as_supervised=True, with_info=True)
train_dataset, test_dataset = dataset['train'], dataset['test']

num_train_examples= info.splits['train'].num_examples
```

```
Downloading and preparing dataset mnist/3.0.1 (download: 11.06 MiB, generate
d: 21.00 MiB, total: 32.06 MiB) to C:\Users\gcont\tensorflow_datasets\mnist\
3.0.1...
```

```
WARNING:absl:Dataset mnist is hosted on GCS. It will automatically be downloa
ded to your
local data directory. If you'd instead prefer to read directly from our publi
c
GCS bucket (recommended if you're running on GCP), you can instead pass
`try_gcs=True` to `tfds.load` or set `data_dir=gs://tfds-data/datasets`.
```

```
Dataset mnist downloaded and prepared to C:\Users\gcont\tensorflow_datasets\m
nist\3.0.1. Subsequent calls will reuse this data.
```

Write a function to augment the images. Map it over the the dataset. This returns a dataset that augments the data on the fly.

In [14]:
```
def convert(image, label):
  image = tf.image.convert_image_dtype(image, tf.float32) # Cast and normali
ze the image to [0,1]
  return image, label

def augment(image,label):
  image,label = convert(image, label)
  image = tf.image.convert_image_dtype(image, tf.float32) # Cast and normali
ze the image to [0,1]
  image = tf.image.resize_with_crop_or_pad(image, 34, 34) # Add 6 pixels of
padding
  image = tf.image.random_crop(image, size=[28, 28, 1]) # Random crop back t
o 28x28
  image = tf.image.random_brightness(image, max_delta=0.5) # Random brightne
ss

  return image,label
```

In [15]:
```
BATCH_SIZE = 64
# Only use a subset of the data so it's easier to overfit, for this tutorial
NUM_EXAMPLES = 2048
```

Create the augmented dataset.

In [16]:
```
augmented_train_batches = (
    train_dataset
    # Only train on a subset, so you can quickly see the effect.
    .take(NUM_EXAMPLES)
    .cache()
    .shuffle(num_train_examples//4)
    # The augmentation is added here.
    .map(augment, num_parallel_calls=AUTOTUNE)
    .batch(BATCH_SIZE)
    .prefetch(AUTOTUNE)
)
```

And a non-augmented one for comparison.

In [17]:
```
non_augmented_train_batches = (
    train_dataset
    # Only train on a subset, so you can quickly see the effect.
    .take(NUM_EXAMPLES)
    .cache()
    .shuffle(num_train_examples//4)
    # No augmentation.
    .map(convert, num_parallel_calls=AUTOTUNE)
    .batch(BATCH_SIZE)
    .prefetch(AUTOTUNE)
)
```

Setup the validation dataset. This doesn't change whether or not you're using the augmentation.

In [18]:
```
validation_batches = (
    test_dataset
    .map(convert, num_parallel_calls=AUTOTUNE)
    .batch(2*BATCH_SIZE)
)
```

Create and compile the model. The model is a two layered, fully-connected neural network without convolution.

In [19]:
```
def make_model():
  model = tf.keras.Sequential([
      layers.Flatten(input_shape=(28, 28, 1)),
      layers.Dense(4096, activation='relu'),
      layers.Dense(4096, activation='relu'),
      layers.Dense(10)
  ])
  model.compile(optimizer = 'adam',
                loss=tf.losses.SparseCategoricalCrossentropy(from_logits=Tru
e),
                metrics=['accuracy'])
  return model
```

Train the model, **without** augmentation:

In [20]:
```
model_without_aug = make_model()

no_aug_history = model_without_aug.fit(non_augmented_train_batches, epochs=5
0, validation_data=validation_batches)
```

```
Epoch 1/50
32/32 [==============================] - 10s 300ms/step - loss: 0.9558 - accu
racy: 0.7285 - val_loss: 0.4432 - val_accuracy: 0.8653
Epoch 2/50
32/32 [==============================] - 11s 335ms/step - loss: 0.2270 - accu
racy: 0.9282 - val_loss: 0.3182 - val_accuracy: 0.9058
Epoch 3/50
32/32 [==============================] - 6s 202ms/step - loss: 0.0797 - accur
acy: 0.9756 - val_loss: 0.2658 - val_accuracy: 0.9237
Epoch 4/50
32/32 [==============================] - 6s 200ms/step - loss: 0.0364 - accur
acy: 0.9893 - val_loss: 0.2954 - val_accuracy: 0.9280
Epoch 5/50
32/32 [==============================] - 7s 211ms/step - loss: 0.0228 - accur
acy: 0.9932 - val_loss: 0.3575 - val_accuracy: 0.9222
Epoch 6/50
32/32 [==============================] - 7s 216ms/step - loss: 0.0249 - accur
acy: 0.9893 - val_loss: 0.4029 - val_accuracy: 0.9183
Epoch 7/50
32/32 [==============================] - 7s 216ms/step - loss: 0.0265 - accur
acy: 0.9922 - val_loss: 0.3789 - val_accuracy: 0.9238
Epoch 8/50
32/32 [==============================] - 7s 216ms/step - loss: 0.0367 - accur
acy: 0.9868 - val_loss: 0.4761 - val_accuracy: 0.9010
Epoch 9/50
32/32 [==============================] - 7s 218ms/step - loss: 0.0650 - accur
acy: 0.9810 - val_loss: 0.3904 - val_accuracy: 0.9238
Epoch 10/50
32/32 [==============================] - 7s 213ms/step - loss: 0.0362 - accur
acy: 0.9893 - val_loss: 0.3626 - val_accuracy: 0.9272
Epoch 11/50
32/32 [==============================] - 7s 220ms/step - loss: 0.0268 - accur
acy: 0.9907 - val_loss: 0.4220 - val_accuracy: 0.9200
Epoch 12/50
32/32 [==============================] - 7s 223ms/step - loss: 0.0484 - accur
acy: 0.9829 - val_loss: 0.4235 - val_accuracy: 0.9141
Epoch 13/50
32/32 [==============================] - 7s 231ms/step - loss: 0.0402 - accur
acy: 0.9902 - val_loss: 0.4773 - val_accuracy: 0.9118
Epoch 14/50
32/32 [==============================] - 8s 237ms/step - loss: 0.0196 - accur
acy: 0.9951 - val_loss: 0.4108 - val_accuracy: 0.9215
Epoch 15/50
32/32 [==============================] - 8s 250ms/step - loss: 0.0080 - accur
acy: 0.9980 - val_loss: 0.3638 - val_accuracy: 0.9299
Epoch 16/50
32/32 [==============================] - 9s 285ms/step - loss: 0.0077 - accur
acy: 0.9971 - val_loss: 0.3838 - val_accuracy: 0.9290
Epoch 17/50
32/32 [==============================] - 11s 338ms/step - loss: 0.0117 - accu
racy: 0.9966 - val_loss: 0.3424 - val_accuracy: 0.9380
Epoch 18/50
32/32 [==============================] - 10s 310ms/step - loss: 0.0058 - accu
racy: 0.9980 - val_loss: 0.5034 - val_accuracy: 0.9218
Epoch 19/50
32/32 [==============================] - 9s 269ms/step - loss: 0.0332 - accur
acy: 0.9922 - val_loss: 0.4584 - val_accuracy: 0.9254
Epoch 20/50
32/32 [==============================] - 9s 284ms/step - loss: 0.0271 - accur
acy: 0.9917 - val_loss: 0.5029 - val_accuracy: 0.9201
Epoch 21/50
32/32 [==============================] - 9s 290ms/step - loss: 0.0163 - accur
acy: 0.9937 - val_loss: 0.4485 - val_accuracy: 0.9294
Epoch 22/50
32/32 [==============================] - 11s 335ms/step - loss: 0.0205 - accu
racy: 0.9946 - val_loss: 0.5501 - val_accuracy: 0.9161
Epoch 23/50
32/32 [==============================] - 10s 315ms/step - loss: 0.0354 - accu
```

```
racy: 0.9912 - val_loss: 0.4424 - val_accuracy: 0.9284
Epoch 24/50
32/32 [==============================] - 14s 424ms/step - loss: 0.0380 - accu
racy: 0.9893 - val_loss: 0.5404 - val_accuracy: 0.9182
Epoch 25/50
32/32 [==============================] - 12s 388ms/step - loss: 0.0409 - accu
racy: 0.9873 - val_loss: 0.4819 - val_accuracy: 0.9208
Epoch 26/50
32/32 [==============================] - 11s 340ms/step - loss: 0.0162 - accu
racy: 0.9946 - val_loss: 0.4960 - val_accuracy: 0.9247
Epoch 27/50
32/32 [==============================] - 11s 336ms/step - loss: 0.0303 - accu
racy: 0.9897 - val_loss: 0.6399 - val_accuracy: 0.9092
Epoch 28/50
32/32 [==============================] - 10s 313ms/step - loss: 0.0293 - accu
racy: 0.9941 - val_loss: 0.5493 - val_accuracy: 0.9231
Epoch 29/50
32/32 [==============================] - 11s 342ms/step - loss: 0.0518 - accu
racy: 0.9893 - val_loss: 0.5894 - val_accuracy: 0.9119
Epoch 30/50
32/32 [==============================] - 9s 272ms/step - loss: 0.0171 - accur
acy: 0.9951 - val_loss: 0.4361 - val_accuracy: 0.9327
Epoch 31/50
32/32 [==============================] - 9s 290ms/step - loss: 0.0071 - accur
acy: 0.9980 - val_loss: 0.4741 - val_accuracy: 0.9277
Epoch 32/50
32/32 [==============================] - 9s 277ms/step - loss: 0.0049 - accur
acy: 0.9990 - val_loss: 0.4174 - val_accuracy: 0.9367
Epoch 33/50
32/32 [==============================] - 9s 280ms/step - loss: 6.7054e-04 - a
ccuracy: 0.9995 - val_loss: 0.4276 - val_accuracy: 0.9356
Epoch 34/50
32/32 [==============================] - 9s 278ms/step - loss: 3.4375e-04 - a
ccuracy: 1.0000 - val_loss: 0.4298 - val_accuracy: 0.9352
Epoch 35/50
32/32 [==============================] - 9s 276ms/step - loss: 9.0659e-05 - a
ccuracy: 1.0000 - val_loss: 0.4292 - val_accuracy: 0.9370
Epoch 36/50
32/32 [==============================] - 9s 275ms/step - loss: 5.5513e-05 - a
ccuracy: 1.0000 - val_loss: 0.4296 - val_accuracy: 0.9373
Epoch 37/50
32/32 [==============================] - 10s 303ms/step - loss: 4.7191e-05 -
accuracy: 1.0000 - val_loss: 0.4304 - val_accuracy: 0.9373
Epoch 38/50
32/32 [==============================] - 12s 371ms/step - loss: 4.1376e-05 -
accuracy: 1.0000 - val_loss: 0.4313 - val_accuracy: 0.9372
Epoch 39/50
32/32 [==============================] - 11s 329ms/step - loss: 3.7425e-05 -
accuracy: 1.0000 - val_loss: 0.4322 - val_accuracy: 0.9371
Epoch 40/50
32/32 [==============================] - 9s 296ms/step - loss: 3.4229e-05 - a
ccuracy: 1.0000 - val_loss: 0.4330 - val_accuracy: 0.9365
Epoch 41/50
32/32 [==============================] - 13s 394ms/step - loss: 3.1477e-05 -
accuracy: 1.0000 - val_loss: 0.4340 - val_accuracy: 0.9365
Epoch 42/50
32/32 [==============================] - 11s 348ms/step - loss: 2.9094e-05 -
accuracy: 1.0000 - val_loss: 0.4349 - val_accuracy: 0.9365
Epoch 43/50
32/32 [==============================] - 10s 307ms/step - loss: 2.7075e-05 -
accuracy: 1.0000 - val_loss: 0.4359 - val_accuracy: 0.9366
Epoch 44/50
32/32 [==============================] - 10s 309ms/step - loss: 2.5183e-05 -
accuracy: 1.0000 - val_loss: 0.4369 - val_accuracy: 0.9367
Epoch 45/50
32/32 [==============================] - 10s 303ms/step - loss: 2.3403e-05 -
accuracy: 1.0000 - val_loss: 0.4379 - val_accuracy: 0.9366
Epoch 46/50
```

```
32/32 [==============================] - 9s 289ms/step - loss: 2.1840e-05 - a
ccuracy: 1.0000 - val_loss: 0.4393 - val_accuracy: 0.9366
Epoch 47/50
32/32 [==============================] - 10s 319ms/step - loss: 2.0270e-05 -
accuracy: 1.0000 - val_loss: 0.4404 - val_accuracy: 0.9365
Epoch 48/50
32/32 [==============================] - 9s 288ms/step - loss: 1.8933e-05 - a
ccuracy: 1.0000 - val_loss: 0.4418 - val_accuracy: 0.9365
Epoch 49/50
32/32 [==============================] - 9s 286ms/step - loss: 1.7515e-05 - a
ccuracy: 1.0000 - val_loss: 0.4433 - val_accuracy: 0.9364
Epoch 50/50
32/32 [==============================] - 12s 382ms/step - loss: 1.6231e-05 -
accuracy: 1.0000 - val_loss: 0.4449 - val_accuracy: 0.9364
```

Train it again with augmentation:

In [21]:
```
model_with_aug = make_model()

aug_history = model_with_aug.fit(augmented_train_batches, epochs=50, validat
ion_data=validation_batches)
```

```
Epoch 1/50
32/32 [==============================] - 12s 379ms/step - loss: 2.3040 - accu
racy: 0.3076 - val_loss: 1.1106 - val_accuracy: 0.7139
Epoch 2/50
32/32 [==============================] - 10s 311ms/step - loss: 1.3123 - accu
racy: 0.5674 - val_loss: 0.7567 - val_accuracy: 0.7586
Epoch 3/50
32/32 [==============================] - 10s 326ms/step - loss: 0.9369 - accu
racy: 0.6807 - val_loss: 0.5183 - val_accuracy: 0.8437
Epoch 4/50
32/32 [==============================] - 10s 322ms/step - loss: 0.7522 - accu
racy: 0.7412 - val_loss: 0.3666 - val_accuracy: 0.8905
Epoch 5/50
32/32 [==============================] - 10s 324ms/step - loss: 0.6499 - accu
racy: 0.7778 - val_loss: 0.3120 - val_accuracy: 0.9111
Epoch 6/50
32/32 [==============================] - 9s 296ms/step - loss: 0.5878 - accur
acy: 0.7979 - val_loss: 0.3062 - val_accuracy: 0.9061
Epoch 7/50
32/32 [==============================] - 13s 402ms/step - loss: 0.5146 - accu
racy: 0.8413 - val_loss: 0.2507 - val_accuracy: 0.9240
Epoch 8/50
32/32 [==============================] - 11s 351ms/step - loss: 0.5211 - accu
racy: 0.8306 - val_loss: 0.3510 - val_accuracy: 0.8806
Epoch 9/50
32/32 [==============================] - 12s 379ms/step - loss: 0.5186 - accu
racy: 0.8218 - val_loss: 0.2766 - val_accuracy: 0.9113
Epoch 10/50
32/32 [==============================] - 10s 323ms/step - loss: 0.4263 - accu
racy: 0.8652 - val_loss: 0.2462 - val_accuracy: 0.9235
Epoch 11/50
32/32 [==============================] - 14s 425ms/step - loss: 0.4073 - accu
racy: 0.8672 - val_loss: 0.2096 - val_accuracy: 0.9346
Epoch 12/50
32/32 [==============================] - 11s 350ms/step - loss: 0.3593 - accu
racy: 0.8857 - val_loss: 0.2102 - val_accuracy: 0.9331
Epoch 13/50
32/32 [==============================] - 9s 288ms/step - loss: 0.3796 - accur
acy: 0.8667 - val_loss: 0.2231 - val_accuracy: 0.9314
Epoch 14/50
32/32 [==============================] - 9s 280ms/step - loss: 0.3449 - accur
acy: 0.8921 - val_loss: 0.2314 - val_accuracy: 0.9263
Epoch 15/50
32/32 [==============================] - 9s 283ms/step - loss: 0.3128 - accur
acy: 0.8931 - val_loss: 0.2220 - val_accuracy: 0.9257
Epoch 16/50
32/32 [==============================] - 9s 280ms/step - loss: 0.3405 - accur
acy: 0.8838 - val_loss: 0.2003 - val_accuracy: 0.9357
Epoch 17/50
32/32 [==============================] - 9s 291ms/step - loss: 0.2647 - accur
acy: 0.9136 - val_loss: 0.1985 - val_accuracy: 0.9406
Epoch 18/50
32/32 [==============================] - 13s 411ms/step - loss: 0.3147 - accu
racy: 0.8896 - val_loss: 0.2109 - val_accuracy: 0.9332
Epoch 19/50
32/32 [==============================] - 9s 281ms/step - loss: 0.2909 - accur
acy: 0.9038 - val_loss: 0.1985 - val_accuracy: 0.9380
Epoch 20/50
32/32 [==============================] - 9s 287ms/step - loss: 0.2791 - accur
acy: 0.9048 - val_loss: 0.1897 - val_accuracy: 0.9414
Epoch 21/50
32/32 [==============================] - 9s 285ms/step - loss: 0.2887 - accur
acy: 0.8994 - val_loss: 0.1900 - val_accuracy: 0.9405
Epoch 22/50
32/32 [==============================] - 11s 339ms/step - loss: 0.2550 - accu
racy: 0.9209 - val_loss: 0.1785 - val_accuracy: 0.9459
Epoch 23/50
32/32 [==============================] - 12s 373ms/step - loss: 0.2853 - accu
```

```
racy: 0.9077 - val_loss: 0.1728 - val_accuracy: 0.9477
Epoch 24/50
32/32 [==============================] - 11s 346ms/step - loss: 0.2869 - accu
racy: 0.9072 - val_loss: 0.1935 - val_accuracy: 0.9333
Epoch 25/50
32/32 [==============================] - 12s 373ms/step - loss: 0.2534 - accu
racy: 0.9141 - val_loss: 0.1925 - val_accuracy: 0.9367
Epoch 26/50
32/32 [==============================] - 11s 340ms/step - loss: 0.2565 - accu
racy: 0.9219 - val_loss: 0.1866 - val_accuracy: 0.9422
Epoch 27/50
32/32 [==============================] - 9s 293ms/step - loss: 0.2349 - accur
acy: 0.9194 - val_loss: 0.1566 - val_accuracy: 0.9509
Epoch 28/50
32/32 [==============================] - 11s 348ms/step - loss: 0.2156 - accu
racy: 0.9268 - val_loss: 0.1638 - val_accuracy: 0.9495
Epoch 29/50
32/32 [==============================] - 10s 302ms/step - loss: 0.2387 - accu
racy: 0.9219 - val_loss: 0.1692 - val_accuracy: 0.9482
Epoch 30/50
32/32 [==============================] - 11s 334ms/step - loss: 0.2155 - accu
racy: 0.9336 - val_loss: 0.1677 - val_accuracy: 0.9520
Epoch 31/50
32/32 [==============================] - 9s 296ms/step - loss: 0.1932 - accur
acy: 0.9321 - val_loss: 0.1833 - val_accuracy: 0.9463
Epoch 32/50
32/32 [==============================] - 10s 301ms/step - loss: 0.1790 - accu
racy: 0.9409 - val_loss: 0.1658 - val_accuracy: 0.9507
Epoch 33/50
32/32 [==============================] - 11s 347ms/step - loss: 0.2296 - accu
racy: 0.9282 - val_loss: 0.1741 - val_accuracy: 0.9459
Epoch 34/50
32/32 [==============================] - 11s 352ms/step - loss: 0.1880 - accu
racy: 0.9370 - val_loss: 0.1764 - val_accuracy: 0.9473
Epoch 35/50
32/32 [==============================] - 9s 278ms/step - loss: 0.2127 - accur
acy: 0.9341 - val_loss: 0.1917 - val_accuracy: 0.9435
Epoch 36/50
32/32 [==============================] - 11s 342ms/step - loss: 0.1955 - accu
racy: 0.9370 - val_loss: 0.1766 - val_accuracy: 0.9482
Epoch 37/50
32/32 [==============================] - 10s 318ms/step - loss: 0.1772 - accu
racy: 0.9429 - val_loss: 0.1575 - val_accuracy: 0.9526
Epoch 38/50
32/32 [==============================] - 9s 276ms/step - loss: 0.1931 - accur
acy: 0.9390 - val_loss: 0.1689 - val_accuracy: 0.9510
Epoch 39/50
32/32 [==============================] - 9s 272ms/step - loss: 0.2634 - accur
acy: 0.9219 - val_loss: 0.1764 - val_accuracy: 0.9464
Epoch 40/50
32/32 [==============================] - 9s 278ms/step - loss: 0.2062 - accur
acy: 0.9321 - val_loss: 0.1577 - val_accuracy: 0.9514
Epoch 41/50
32/32 [==============================] - 9s 270ms/step - loss: 0.1666 - accur
acy: 0.9419 - val_loss: 0.1740 - val_accuracy: 0.9489
Epoch 42/50
32/32 [==============================] - 9s 287ms/step - loss: 0.1860 - accur
acy: 0.9399 - val_loss: 0.1524 - val_accuracy: 0.9536
Epoch 43/50
32/32 [==============================] - 9s 269ms/step - loss: 0.1613 - accur
acy: 0.9497 - val_loss: 0.1756 - val_accuracy: 0.9496
Epoch 44/50
32/32 [==============================] - 9s 278ms/step - loss: 0.1695 - accur
acy: 0.9434 - val_loss: 0.1503 - val_accuracy: 0.9540
Epoch 45/50
32/32 [==============================] - 9s 270ms/step - loss: 0.1501 - accur
acy: 0.9517 - val_loss: 0.1672 - val_accuracy: 0.9491
Epoch 46/50
```

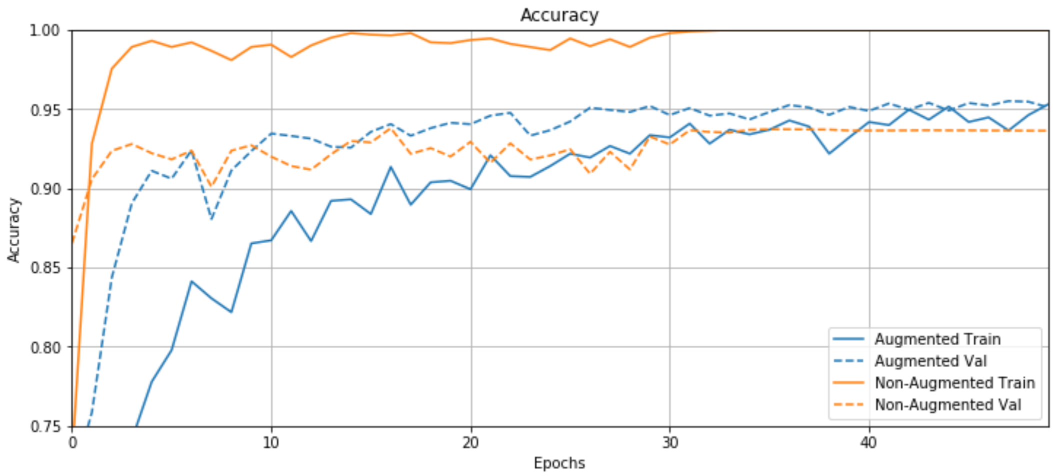
```
32/32 [==============================] - 9s 277ms/step - loss: 0.1519 - accur
acy: 0.9419 - val_loss: 0.1701 - val_accuracy: 0.9539
Epoch 47/50
32/32 [==============================] - 9s 270ms/step - loss: 0.1752 - accur
acy: 0.9448 - val_loss: 0.1597 - val_accuracy: 0.9523
Epoch 48/50
32/32 [==============================] - 9s 276ms/step - loss: 0.1835 - accur
acy: 0.9365 - val_loss: 0.1495 - val_accuracy: 0.9551
Epoch 49/50
32/32 [==============================] - 9s 280ms/step - loss: 0.1559 - accur
acy: 0.9463 - val_loss: 0.1629 - val_accuracy: 0.9547
Epoch 50/50
32/32 [==============================] - 9s 273ms/step - loss: 0.1431 - accur
acy: 0.9531 - val_loss: 0.1663 - val_accuracy: 0.9511
```

Conclusion:

In this example the augmented model converges to an accuracy ~95% on validation set. This is slightly higher (+1%) than the model trained without data augmentation.

In [22]:
```
plotter = tfdocs.plots.HistoryPlotter()
plotter.plot({"Augmented": aug_history, "Non-Augmented": no_aug_history}, me
tric = "accuracy")
plt.title("Accuracy")
plt.ylim([0.75,1])
```

Out[22]:
```
(0.75, 1)
```



In terms of loss, the non-augmented model is obviously in the overfitting regime. The augmented model, while a few epoch slower, is still training correctly and clearly not overfitting.

In [23]:
```
plotter = tfdocs.plots.HistoryPlotter()
plotter.plot({"Augmented": aug_history, "Non-Augmented": no_aug_history}, me
tric = "loss")
plt.title("Loss")
plt.ylim([0,1])
```

Out[23]: (0, 1)