```
import datetime
import math
import os

import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow_datasets as tfds
import tensorflow_hub as hub

BATCH_SIZE = 32

train_data, train_info = tfds.load('cifar10', split='train[10%:90%]', with_info=True)
val_data = tfds.load('cifar10', split='train[0%:10%]')
test_data = tfds.load('cifar10', split='test')
print(train_data)

num_train_data = 0
for _ in train_data:
    num_train_data += 1
print(num_train_data)

num_val_data = 0
for _ in val_data:
    num_val_data += 1
print(num_val_data)

train_steps_per_epoch = math.ceil(num_train_data / BATCH_SIZE)
val_steps_per_epoch = math.ceil(num_val_data / BATCH_SIZE)


def normalizer(features, input_shape=[299, 299, 3], augment=True, seed=42):
    input_shape = tf.convert_to_tensor(input_shape)
    image = features['image']
```



```
        image = tf.image.random_flip_left_right(image, seed=seed)

        # Random B/S changes:
        image = tf.image.random_brightness(image, max_delta=0.1, seed=seed)
        image = tf.image.random_saturation(image, lower=0.5, upper=1.5, seed=seed)
        image = tf.clip_by_value(image, 0.0, 1.0)  # keeping pixel values in check

        # Random resize and random crop back to expected size:

        random_scale_factor = tf.random.uniform([1], minval=1., maxval=1.4, dtype=tf.float32,
        scaled_height = tf.cast(tf.cast(input_shape[0], tf.float32) * random_scale_factor,
                                tf.int32)
        scaled_width = tf.cast(tf.cast(input_shape[1], tf.float32) * random_scale_factor,
                               tf.int32)
```

```
            scaled_shape = tf.squeeze(tf.stack([scaled_height, scaled_width]))
            image = tf.image.resize(image, scaled_shape)
            image = tf.image.random_crop(image, input_shape, seed=seed)
        else:
            image = tf.image.resize(image, input_shape[:2])
        label = features['label']
        features = (image, label)
        return features


    train_data = train_data.map(normalizer)
    val_data = val_data.map(normalizer)

    print(train_data)
    print(val_data)
    class_names = train_info.features["label"].names
    print(class_names)

    plt.figure(figsize=(10, 10))
    for i, (image, label) in enumerate(train_data.take(24)):
        # image = image.numpy().reshape([28,28,3])
        plt.subplot(5, 5, i + 1)
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        plt.imshow(image, cmap=plt.cm.binary)
        plt.xlabel(class_names[label])
    plt.show()

    train_data = train_data.batch(BATCH_SIZE)
    val_data = val_data.batch(BATCH_SIZE)
    train_data = train_data.prefetch(1)
    val_data = val_data.prefetch(1)
    # train_data = train_data.cache()
```

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu

```
    Inception_url = "https://tfhub.dev/google/tf2-preview/inception_v3/feature_vector/2"
    inception_v3 = hub.KerasLayer(
        Inception_url, trainable=False,
        input_shape=[299, 299, 3],
        output_shape=[2048],
        dtype=tf.float32
    )

    LeNet = tf.keras.Sequential([
        inception_v3,
        tf.keras.layers.Dense(10, activation='softmax', name='logits_pred')
    ], name="LeNet")

    print(LeNet.summary())
```

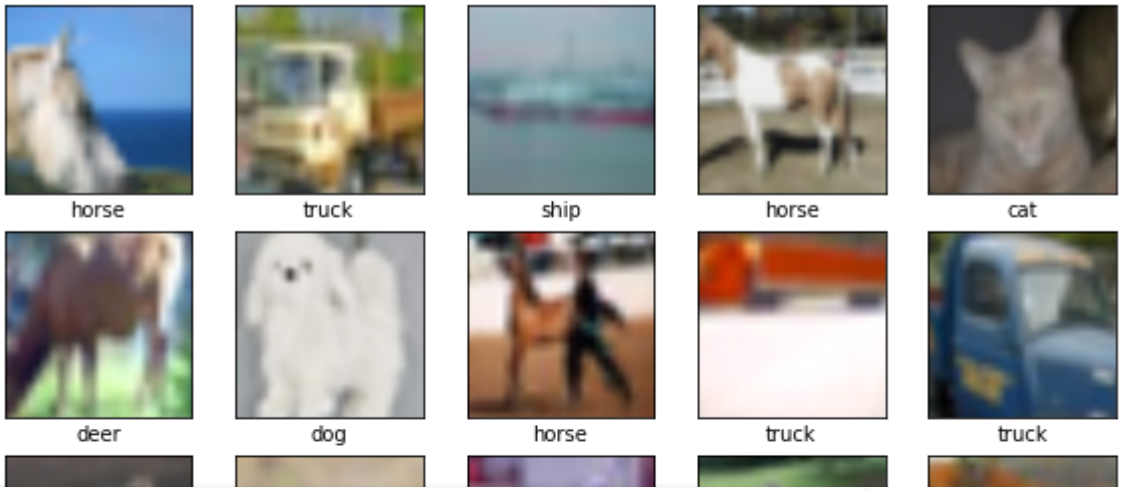
```
model_dir = './models/LeNet'
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
callbacks = [
    # Callback to interrupt the training if the validation loss/metrics stops improving for s
    tf.keras.callbacks.EarlyStopping(patience=8, monitor='val_acc',
                                     restore_best_weights=True),
    # Callback to log the graph, losses and metrics into TensorBoard:

    tf.keras.callbacks.TensorBoard(model_dir, histogram_freq=1, write_graph=True)

    # Callback to simply log metrics at the end of each epoch (saving space compared to verbo
]
LeNet.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=[tf.keras.metrics.SparseCategoricalAccuracy(name='acc'),
                       tf.keras.metrics.SparseTopKCategoricalAccuracy(k=5, name='top5_acc')])
```

```
Dl Completed...: 100%                    1/1 [00:05<00:00, 5.00s/ url]
Dl Size...: 100%                         162/162 [00:04<00:00, 32.61 MiB/s]
Extraction completed...: 100%            1/1 [00:04<00:00, 4.91s/ file]
```

```
Shuffling and writing examples to /root/tensorflow_datasets/cifar10/3.0.2.incompleteYTP
93%                                      46744/50000 [00:00<00:00, 79542.34 examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/cifar10/3.0.2.incompleteYTP
95%                                      9494/10000 [00:00<00:00, 94939.41 examples/s]
Dataset cifar10 downloaded and prepared to /root/tensorflow_datasets/cifar10/3.0.2. Sub
<PrefetchDataset shapes: {id: (), image: (32, 32, 3), label: ()}, types: {id: tf.string
40000
5000
<MapDataset shapes: ((299, 299, 3), ()), types: (tf.float32, tf.int64)>
<MapDataset shapes: ((299, 299, 3), ()), types: (tf.float32, tf.int64)>
['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truc
```



To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu

```
INFO:tensorflow:Assets written to: LeNet_with_augmentation/assets
INFO:tensorflow:Assets written to: LeNet_with_augmentation/assets
```

```
test_data = tfds.load('cifar10', split='test')
```

```
LeNet = tf.keras.models.load_model('LeNet_with_augmentation')
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
<ipython-input-4-38a4aeb845b9> in <module>()
----> 1 LeNet = tf.keras.models.load_model('LeNet_with_augmentation')

                                  1 frames
/usr/local/lib/python3.7/dist-packages/tensorflow/python/saved_model/loader_impl.py in parse_saved_model(export_dir)
    112                   (export_dir,
    113                    constants.SAVED_MODEL_FILENAME_PBTXT,
--> 114                    constants.SAVED_MODEL_FILENAME_PB))
    115
```

```
num_tests=0
for _ in test_data:
    num_tests+=1
print(num_tests)
```

```
10000
```

```
test_data = test_data.map(normalizer)
test_data = test_data.batch(BATCH_SIZE)
test_data = test_data.prefetch(1)
```

```
LeNet.evaluate(test_data,batch_size=BATCH_SIZE,verbose=1)
```

```
313/313 [==============================] - 40s 111ms/step - loss: 0.5829 - acc: 0.8319
[0.582894504070282, 0.8319000005722046, 0.9940000176429749]
```

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu