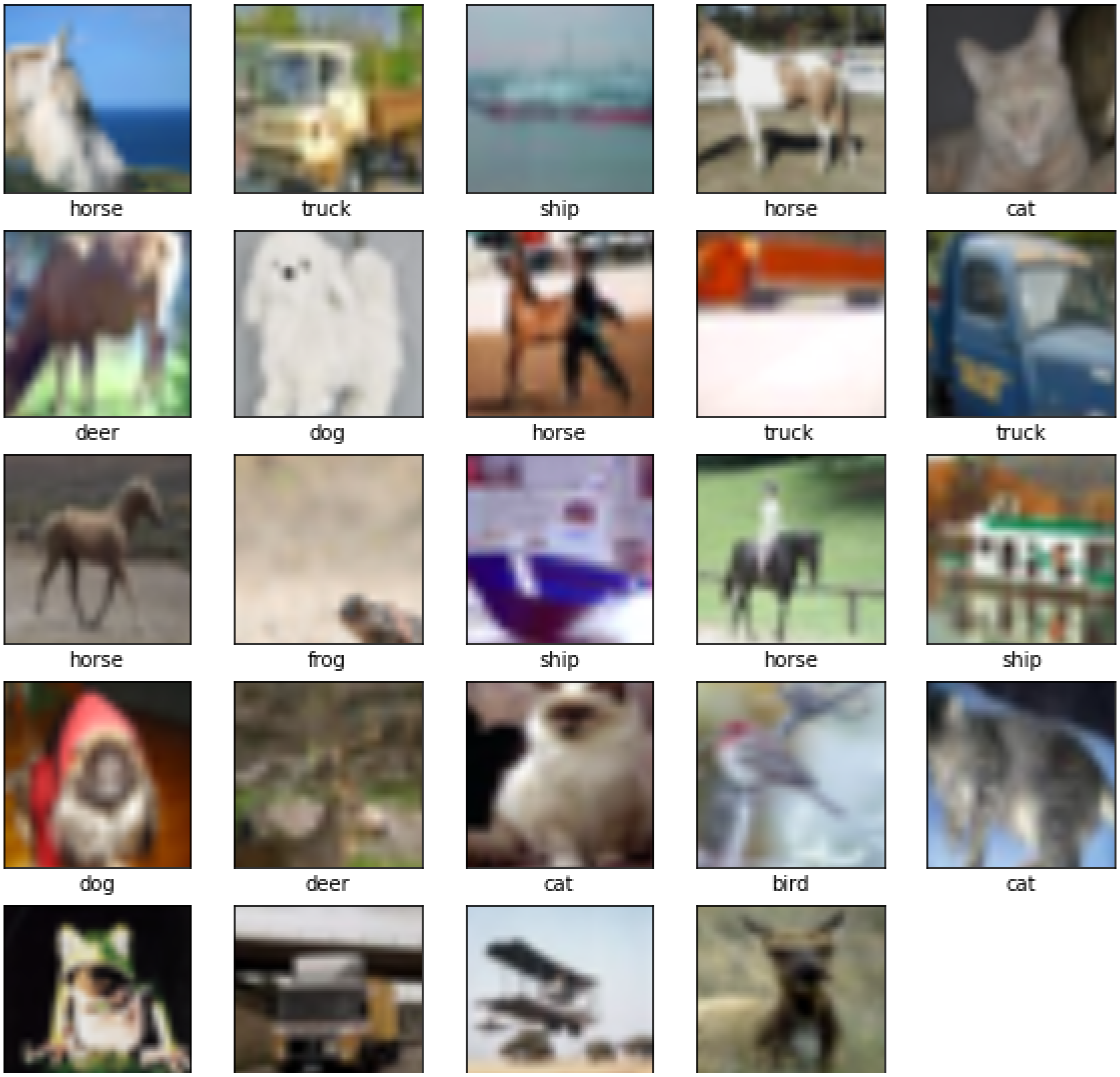
```
import datetime
import math
import os

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.convert_image_dtype(image, tf.float32)
    if augment:
        # Randomly applied horizontal flip:
        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()

print(val_data)
print(train_data)

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())
```

```
    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')])
    history = LeNet.fit(train_data,epochs=50,steps_per_epoch=train_steps_per_epoch,validation_dat
         Downloading and preparing dataset cifar10/3.0.2 (download: 162.17 MiB, generated: 132.4
         Dl Completed...: 100%                                   1/1 [00:04<00:00, 4.55s/ url]

         Dl Size...: 100%                                        162/162 [00:04<00:00, 35.82 MiB/s]

         Extraction completed...: 100%                           1/1 [00:04<00:00, 4.47s/ file]

         Shuffling and writing examples to /root/tensorflow_datasets/cifar10/3.0.2.incompleteTZQ
         87%                                                     43270/50000 [00:00<00:00, 63820.75 examples/s]
         Shuffling and writing examples to /root/tensorflow_datasets/cifar10/3.0.2.incompleteTZQ
         0%                                                      0/10000 [00:00<?, ? 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
```



```
<PrefetchDataset shapes: ((None, 299, 299, 3), (None,)), types: (tf.float32, tf.int64)>
<PrefetchDataset shapes: ((None, 299, 299, 3), (None,)), types: (tf.float32, tf.int64)>
Model: "LeNet"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     (None, 2048)              21802784
_________________________________________________________________
logits_pred (Dense)          (None, 10)                20490
=================================================================
Total params: 21,823,274
Trainable params: 20,490
Non-trainable params: 21,802,784
_________________________________________________________________
None
```

```
LeNet.save_weights("lenet_weights",overwrite=True)
LeNet.save('LeNet_with_augmentation')
```

```
INFO:tensorflow:Assets written to: LeNet_with_augmentation/assets
INFO:tensorflow:Assets written to: LeNet_with_augmentation/assets
Epoch 4/50
```

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

```
1250/1250 [==============================] - 170s 136ms/step - loss: 0.3770 - acc: 0.86
```

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

```
Epoch 7/50
```

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

```
10000

1250/1250 [==============================] - 170s 136ms/step - loss: 0.3124 - acc: 0.89
```

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

```
Epoch 14/50
```

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

```
313/313 [==============================] - 40s 125ms/step - loss: 0.4954 - acc: 0.8323
[0.4953974485397339, 0.8323000073432922, 0.9937000274658203]


1250/1250 [==============================] - 170s 136ms/step - loss: 0.2782 - acc: 0.90
```

```
%load_ext tensorboard
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
    Mounted at /content/drive
```

```
ls
```

```
    checkpoint                          lenet_weights.index       models/
    lenet_weights.data-00000-of-00001  LeNet_with_augmentation/  sample_data/
```

```
%tensorboard --logdir models/LeNet/
```