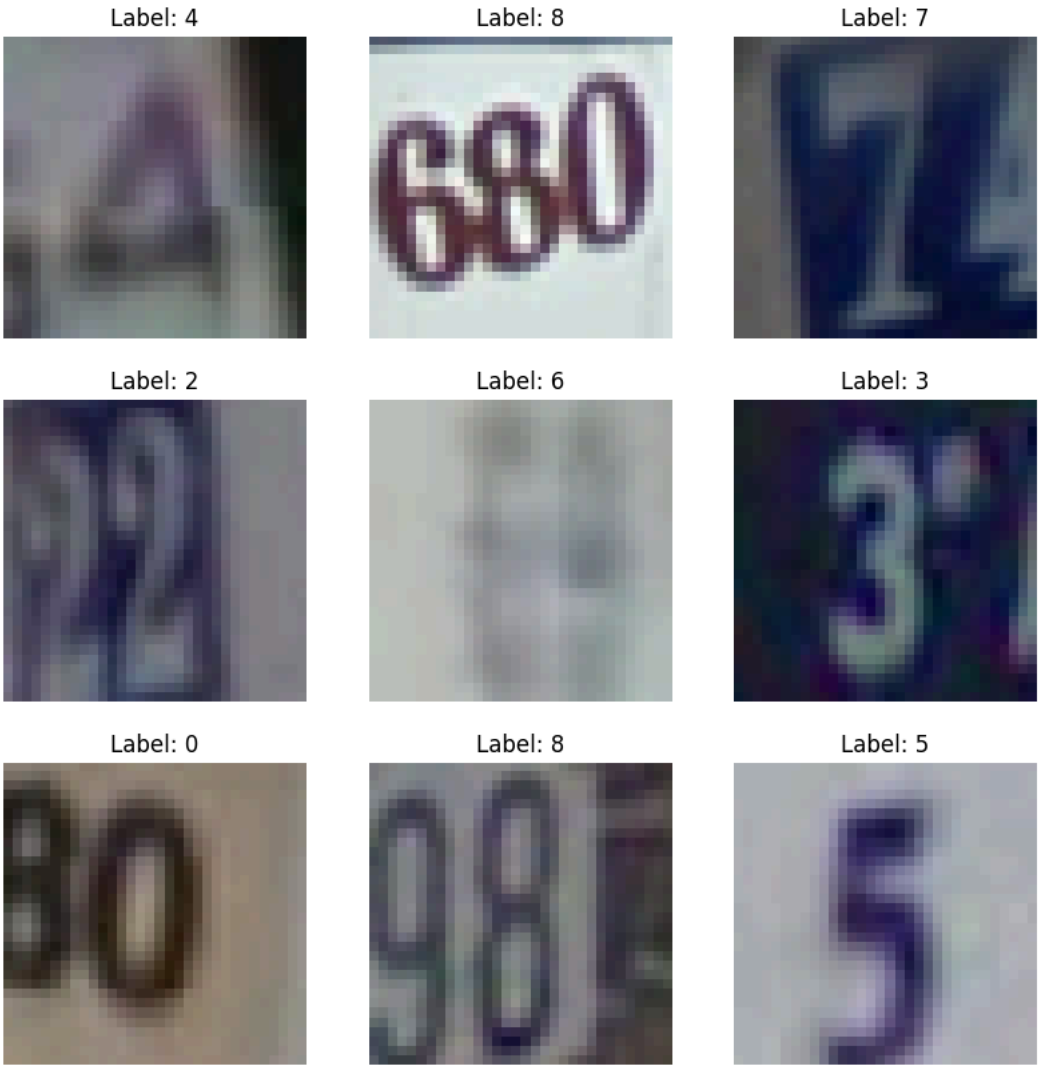
```
import tensorflow as tf
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
import numpy as np

print("Loading 'svhn_cropped' dataset from TFDS...")

# Load the dataset
(raw_train, raw_test), metadata = tfds.load(
    'svhn_cropped',
    split=['train', 'test'],  # We'll split the train set manually later
    with_info=True,
    as_supervised=True,  # Loads as (image, label) tuples
)

print("Dataset loaded successfully.")
```

```
Loading 'svhn_cropped' dataset from TFDS...
WARNING:absl:Variant folder /root/tensorflow_datasets/svhn_cropped/3.1.0 has no dataset_info.json
Downloading and preparing dataset Unknown size (download: Unknown size, generated: Unknown size, total: Unknown size) to /root/t
Dl Completed...: 100%  3/3 [00:59<00:00, 24.47s/ url]
Dl Size...: 100%  1501/1501 [00:59<00:00, 20.52 MiB/s]

Dataset svhn_cropped downloaded and prepared to /root/tensorflow_datasets/svhn_cropped/3.1.0. Subsequent calls will reuse this d
Dataset loaded successfully.
```

```
# Get the class names from metadata
class_names = metadata.features['label'].names
print("Class names:", class_names)

# Get the number of examples
num_train = metadata.splits['train'].num_examples
num_test = metadata.splits['test'].num_examples

print(f"Number of training examples: {num_train}")
print(f"Number of testing examples: {num_test}")

# Check an example image shape
for image, label in raw_train.take(1):
    print(f"\nImage shape: {image.shape}")
```

```
Class names: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
Number of training examples: 73257
Number of testing examples: 26032

Image shape: (32, 32, 3)
```

```
plt.figure(figsize=(10, 10))
# Take 9 examples from the training set
for i, (image, label) in enumerate(raw_train.take(9)):
    ax = plt.subplot(3, 3, i + 1)
    # These are color, so we don't need cmap='gray'
    plt.imshow(image)
    plt.title(f"Label: {class_names[label]}")
    plt.axis('off')
plt.show()
```




```
def preprocess(image, label):
    # Cast the image to float32
    image = tf.cast(image, tf.float32)
    # Normalize the pixel values to [0, 1]
    image = image / 255.0
    return image, label
```

```
# This dictionary will hold our final datasets
datasets = {}

# Get the total number of training examples
num_train = metadata.splits['train'].num_examples

# Calculate 10% of the training data for validation
num_validation = int(0.1 * num_train)

# Create a new validation set (first 10% of train data)
val_set = raw_train.take(num_validation)

# Create a new training set (the remaining 90%)
train_set = raw_train.skip(num_validation)

# Now, apply preprocessing to all three splits
datasets['train'] = train_set.map(preprocess)
datasets['val'] = val_set.map(preprocess)
datasets['test'] = raw_test.map(preprocess)

# Let's check the new counts
print(f"Original training examples: {num_train}")
print(f"New validation examples: {num_validation}")
print(f"New training examples: {num_train - num_validation}")
```

```
Original training examples: 73257
New validation examples: 7325
New training examples: 65932
```

```
BATCH_SIZE = 32

for split in ['train', 'val', 'test']:
    # Shuffle the training data
    if split == 'train':
        datasets[split] = datasets[split].shuffle(1000)

    datasets[split] = datasets[split].batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)

print("All datasets are preprocessed, batched, and ready.")
print("Example of a batch (shape):")
for images, labels in datasets['train'].take(1):
    print(f" - Images batch shape: {images.shape}")
    print(f" - Labels batch shape: {labels.shape}")
```

```
All datasets are preprocessed, batched, and ready.
Example of a batch (shape):
 - Images batch shape: (32, 32, 32, 3)
 - Labels batch shape: (32,)
```

```
model = tf.keras.models.Sequential([
    # Input layer specifies the shape of our color images
    tf.keras.layers.Input(shape=(32, 32, 3)),

    # First convolution block
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.3),

    # Second convolution block
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.3),

    # Third convolution block (added one more for a deeper model)
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.3),

    # Flatten the 3D feature maps into a 1D vector
    tf.keras.layers.Flatten(),

    # Dense (fully connected) layers
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),

    # Output layer
    # 10 units for 10 classes (digits 0-9)
    # 'softmax' activation to get probabilities for each class
    tf.keras.layers.Dense(10, activation='softmax')
])

model.summary()
```

**Model: "sequential"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 32, 32, 32) | 896 |
| max_pooling2d (MaxPooling2D) | (None, 16, 16, 32) | 0 |
| dropout (Dropout) | (None, 16, 16, 32) | 0 |
| conv2d_1 (Conv2D) | (None, 16, 16, 64) | 18,496 |
| max_pooling2d_1 (MaxPooling2D) | (None, 8, 8, 64) | 0 |
| dropout_1 (Dropout) | (None, 8, 8, 64) | 0 |
| conv2d_2 (Conv2D) | (None, 8, 8, 128) | 73,856 |
| max_pooling2d_2 (MaxPooling2D) | (None, 4, 4, 128) | 0 |
| dropout_2 (Dropout) | (None, 4, 4, 128) | 0 |
| flatten (Flatten) | (None, 2048) | 0 |
| dense (Dense) | (None, 128) | 262,272 |
| dropout_3 (Dropout) | (None, 128) | 0 |
| dense_1 (Dense) | (None, 10) | 1,290 |

**Total params:** 356,810 (1.36 MB)

```
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

print("Model compiled.")
```

```
Model compiled.
```

```
EPOCHS = 15

print("Starting training...")

history = model.fit(
    datasets['train'],
    epochs=EPOCHS,
    validation_data=datasets['val']
)

print("Training finished.")
```

```
Starting training...
Epoch 1/15
2061/2061 ━━━━━━━━━━━━━━━━━━━━ 197s 94ms/step - accuracy: 0.2802 - loss: 2.0198 - val_accuracy: 0.7779 - val_loss: 0.7265
Epoch 2/15
2061/2061 ━━━━━━━━━━━━━━━━━━━━ 195s 94ms/step - accuracy: 0.7064 - loss: 0.9220 - val_accuracy: 0.8340 - val_loss: 0.5634
Epoch 3/15
2061/2061 ━━━━━━━━━━━━━━━━━━━━ 191s 92ms/step - accuracy: 0.7655 - loss: 0.7470 - val_accuracy: 0.8523 - val_loss: 0.4943
Epoch 4/15
2061/2061 ━━━━━━━━━━━━━━━━━━━━ 195s 94ms/step - accuracy: 0.7938 - loss: 0.6694 - val_accuracy: 0.8685 - val_loss: 0.4488
Epoch 5/15
2061/2061 ━━━━━━━━━━━━━━━━━━━━ 204s 99ms/step - accuracy: 0.8051 - loss: 0.6278 - val_accuracy: 0.8789 - val_loss: 0.4016
Epoch 6/15
2061/2061 ━━━━━━━━━━━━━━━━━━━━ 193s 93ms/step - accuracy: 0.8149 - loss: 0.5998 - val_accuracy: 0.8874 - val_loss: 0.3855
Epoch 7/15
2061/2061 ━━━━━━━━━━━━━━━━━━━━ 193s 93ms/step - accuracy: 0.8233 - loss: 0.5695 - val_accuracy: 0.8859 - val_loss: 0.3834
Epoch 8/15
2061/2061 ━━━━━━━━━━━━━━━━━━━━ 196s 94ms/step - accuracy: 0.8282 - loss: 0.5584 - val_accuracy: 0.8859 - val_loss: 0.3911
Epoch 9/15
2061/2061 ━━━━━━━━━━━━━━━━━━━━ 193s 93ms/step - accuracy: 0.8326 - loss: 0.5473 - val_accuracy: 0.8965 - val_loss: 0.3598
Epoch 10/15
2061/2061 ━━━━━━━━━━━━━━━━━━━━ 202s 93ms/step - accuracy: 0.8327 - loss: 0.5422 - val_accuracy: 0.8997 - val_loss: 0.3440
Epoch 11/15
2061/2061 ━━━━━━━━━━━━━━━━━━━━ 190s 92ms/step - accuracy: 0.8383 - loss: 0.5317 - val_accuracy: 0.8983 - val_loss: 0.3425
Epoch 12/15
2061/2061 ━━━━━━━━━━━━━━━━━━━━ 190s 91ms/step - accuracy: 0.8386 - loss: 0.5190 - val_accuracy: 0.8995 - val_loss: 0.3460
Epoch 13/15
2061/2061 ━━━━━━━━━━━━━━━━━━━━ 190s 91ms/step - accuracy: 0.8454 - loss: 0.5034 - val_accuracy: 0.9002 - val_loss: 0.3351
```

```
Epoch 14/15
2061/2061 ━━━━━━━━━━━━━━━━━━━━ 192s 93ms/step - accuracy: 0.8460 - loss: 0.5035 - val_accuracy: 0.9059 - val_loss: 0.3247
Epoch 15/15
2061/2061 ━━━━━━━━━━━━━━━━━━━━ 191s 92ms/step - accuracy: 0.8520 - loss: 0.4921 - val_accuracy: 0.9032 - val_loss: 0.3367
Training finished.
```

```
print("Evaluating on test data...")
loss, accuracy = model.evaluate(datasets['test'])

print(f"\nTest Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
Evaluating on test data...
814/814 ━━━━━━━━━━━━━━━━━━━━ 34s 41ms/step - accuracy: 0.9067 - loss: 0.3236

Test Loss: 0.3267
Test Accuracy: 90.58%
```