Google Developers

Preprocessing Layers in Keras

ML Bootcamp India (2022)

Sayak Paul
ML Engineer at Carted
@RisingSayak

$whoami

- ML Engineer at Carted
- Open-source 🥑 (Keras, KerasCV, 🤗 Transformers etc.)
- Netflix nerd
- Coordinates at sayak.dev

Agenda

- A bit about data preprocessing in ML
- Preprocessing layers in Keras
    - Usage
    - Benefits
- QnA

Materials are here:

bit.ly/code-mlb-2022

Data preprocessing in ML

- Scale the pixel values to [0, 1] or any other forms of normalization
- Tokenization of text sequences
- Normalization of numerical features and projection of categorical features (entity embeddings for example)

Data preprocessing in ML

Requirements

- To use the same preprocessing pipeline applied during model training ***and*** inference wherever possible
- Not every training preprocessing step can be applied during inference (MixUp for example)
- Some preprocessing methods have different train and test behaviours – RandomResizedCrop

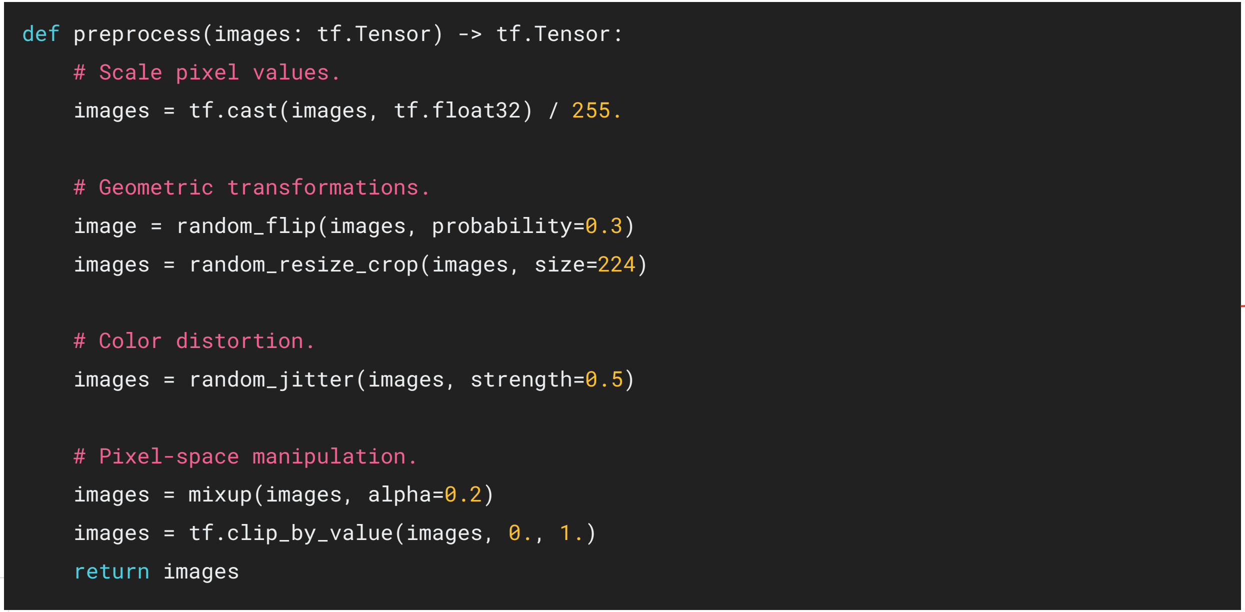
Data preprocessing in ML

- Data preprocessing is included in the data pipeline decoupling it from the model definition.
- What if the end consumer of the model misses the preprocessing steps?
- We'd want to ship a model that is as self-sufficient as possible.

Data preprocessing in ML

```python
def preprocess(images: tf.Tensor) -> tf.Tensor:
    # Scale pixel values.
    images = tf.cast(images, tf.float32) / 255.

    # Geometric transformations.
    image = random_flip(images, probability=0.3)
    images = random_resize_crop(images, size=224)

    # Color distortion.
    images = random_jitter(images, strength=0.5)

    # Pixel-space manipulation.
    images = mixup(images, alpha=0.2)
    images = tf.clip_by_value(images, 0., 1.)
    return images
```

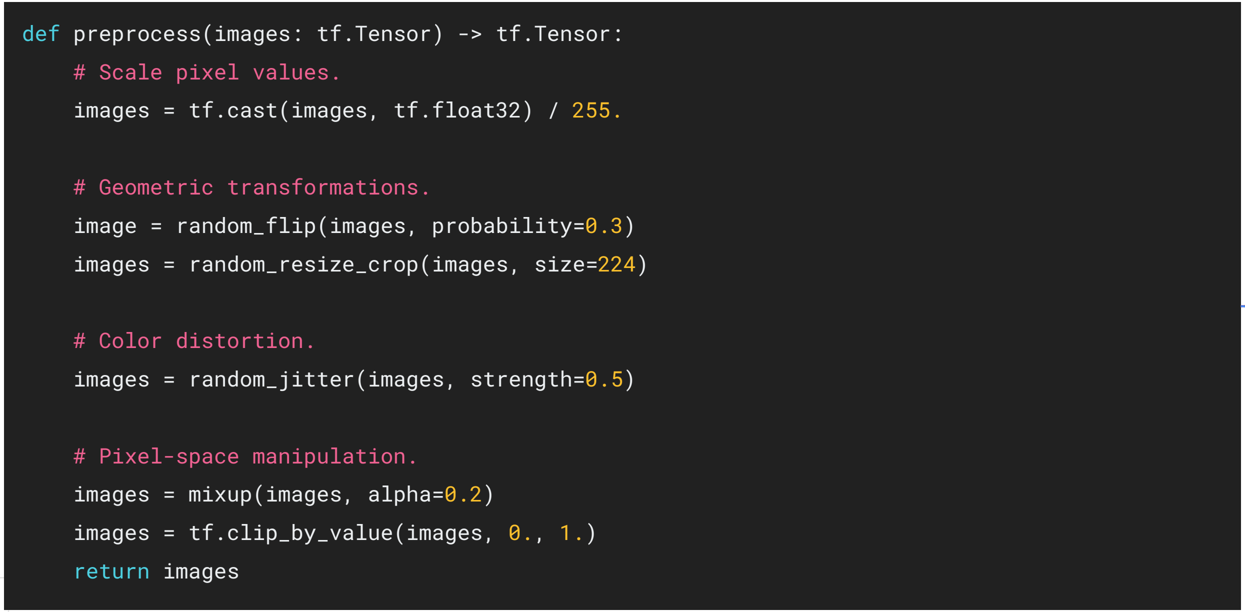
Data preprocessing in ML



```python
def preprocess(images: tf.Tensor) -> tf.Tensor:
    # Scale pixel values.
    images = tf.cast(images, tf.float32) / 255.

    # Geometric transformations.
    image = random_flip(images, probability=0.3)
    images = random_resize_crop(images, size=224)

    # Color distortion.
    images = random_jitter(images, strength=0.5)

    # Pixel-space manipulation.
    images = mixup(images, alpha=0.2)
    images = tf.clip_by_value(images, 0., 1.)
    return images
```

Part of the input data pipeline.

Data preprocessing in ML



```python
def preprocess(images: tf.Tensor) -> tf.Tensor:
    # Scale pixel values.
    images = tf.cast(images, tf.float32) / 255.

    # Geometric transformations.
    image = random_flip(images, probability=0.3)
    images = random_resize_crop(images, size=224)

    # Color distortion.
    images = random_jitter(images, strength=0.5)

    # Pixel-space manipulation.
    images = mixup(images, alpha=0.2)
    images = tf.clip_by_value(images, 0., 1.)
    return images
```

Part of the input data pipeline.

How do we define the inference behaviour?

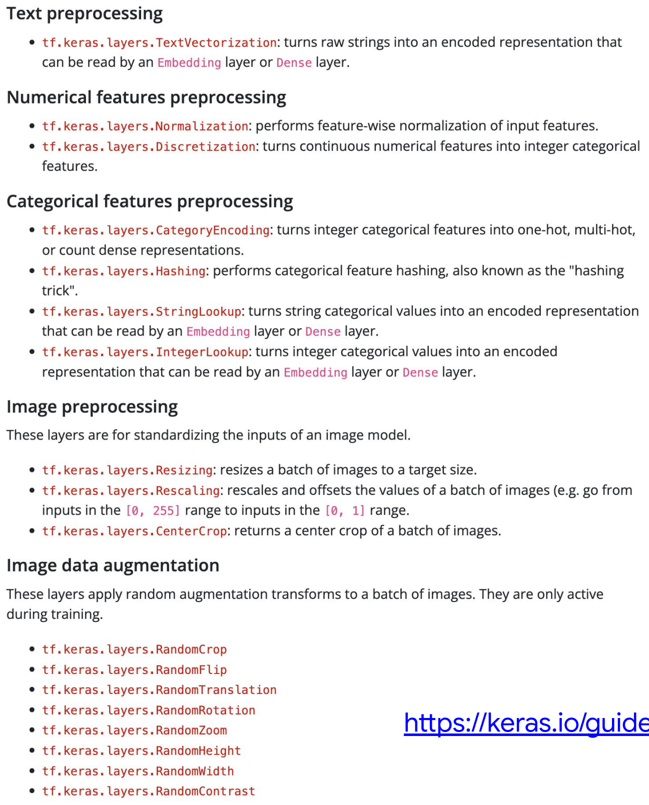
Experts

Google Developers

Data preprocessing in ML

Can we ship a model that has the preprocessing steps included?

Preprocessing layers in Keras



Text preprocessing

- `tf.keras.layers.TextVectorization`: turns raw strings into an encoded representation that can be read by an `Embedding` layer or `Dense` layer.

Numerical features preprocessing

- `tf.keras.layers.Normalization`: performs feature-wise normalization of input features.
- `tf.keras.layers.Discretization`: turns continuous numerical features into integer categorical features.

Categorical features preprocessing

- `tf.keras.layers.CategoryEncoding`: turns integer categorical features into one-hot, multi-hot, or count dense representations.
- `tf.keras.layers.Hashing`: performs categorical feature hashing, also known as the "hashing trick".
- `tf.keras.layers.StringLookup`: turns string categorical values into an encoded representation that can be read by an `Embedding` layer or `Dense` layer.
- `tf.keras.layers.IntegerLookup`: turns integer categorical values into an encoded representation that can be read by an `Embedding` layer or `Dense` layer.

Image preprocessing

These layers are for standardizing the inputs of an image model.

- `tf.keras.layers.Resizing`: resizes a batch of images to a target size.
- `tf.keras.layers.Rescaling`: rescales and offsets the values of a batch of images (e.g. go from inputs in the `[0, 255]` range to inputs in the `[0, 1]` range.
- `tf.keras.layers.CenterCrop`: returns a center crop of a batch of images.

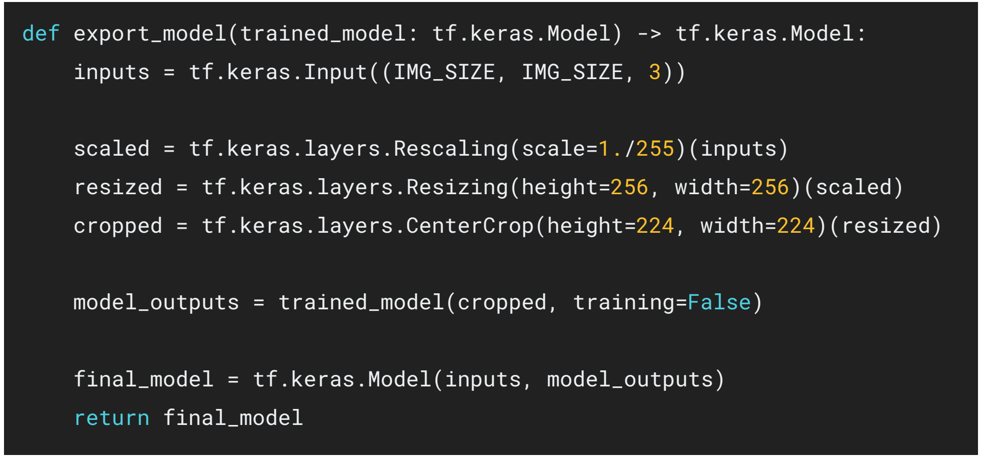
Image data augmentation

These layers apply random augmentation transforms to a batch of images. They are only active during training.

- `tf.keras.layers.RandomCrop`
- `tf.keras.layers.RandomFlip`
- `tf.keras.layers.RandomTranslation`
- `tf.keras.layers.RandomRotation`
- `tf.keras.layers.RandomZoom`
- `tf.keras.layers.RandomHeight`
- `tf.keras.layers.RandomWidth`
- `tf.keras.layers.RandomContrast`

https://keras.io/guides/preprocessing_layers/

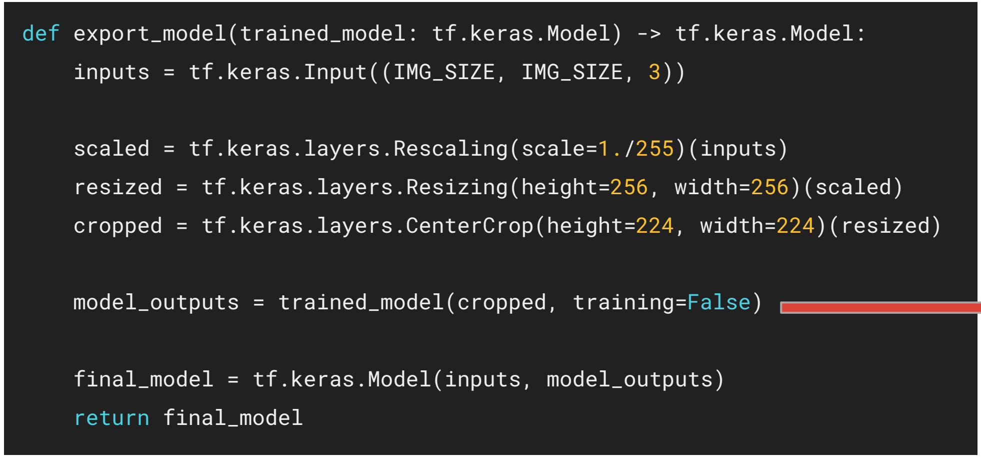
Shipping self-sufficient models (1)



```python
def export_model(trained_model: tf.keras.Model) -> tf.keras.Model:
    inputs = tf.keras.Input((IMG_SIZE, IMG_SIZE, 3))

    scaled = tf.keras.layers.Rescaling(scale=1./255)(inputs)
    resized = tf.keras.layers.Resizing(height=256, width=256)(scaled)
    cropped = tf.keras.layers.CenterCrop(height=224, width=224)(resized)

    model_outputs = trained_model(cropped, training=False)

    final_model = tf.keras.Model(inputs, model_outputs)
    return final_model
```

- Scale inputs.
- Resizing and center crop.

Shipping self-sufficient models (1)



```python
def export_model(trained_model: tf.keras.Model) -> tf.keras.Model:
    inputs = tf.keras.Input((IMG_SIZE, IMG_SIZE, 3))

    scaled = tf.keras.layers.Rescaling(scale=1./255)(inputs)
    resized = tf.keras.layers.Resizing(height=256, width=256)(scaled)
    cropped = tf.keras.layers.CenterCrop(height=224, width=224)(resized)

    model_outputs = trained_model(cropped, training=False)

    final_model = tf.keras.Model(inputs, model_outputs)
    return final_model
```

Infer with the trained model.

Shipping self-sufficient models (2)

```
text_vectorizer = tf.keras.layers.TextVectorization(
    max_tokens=vocabulary_size, ngrams=2, output_mode="tf_idf"
)
with tf.device("/CPU:0"):
    text_vectorizer.adapt(train_dataset.map(lambda text, label: text))

# Train model.
model.compile(...)
model.fit(...)

# Model export.
model_for_inference = tf.keras.Sequential([text_vectorizer, model])
```

https://keras.io/examples/nlp/multi_label_classification

Advantages of these preprocessing layers

- Supports execution on GPUs.
- TPU support available for some layers.
- Supports batched inputs.
- Have their train/test behaviours defined.

Why not include these layers during training?

During training it’s a recommended practice to do data preprocessing async on CPUs so that any hardware accelerator is only utilized for model training.

Learn more

- [Working with preprocessing layers]
- [Classify structured data using Keras preprocessing layers]
- [Classify structured data with feature columns]

Questions?

Experts

Google Developers

Thank you!

Sayak Paul
ML Engineer at Carted
@RisingSayak