Getting Started with KerasCV

What is Keras 3?

- A full rewrite of of Keras
  - No technical debt
  - Much smaller codebase (43K LOC, ~3x smaller)
- Supports multi-backend (JAX, TensorFlow, PyTorch, and Numpy)
  - Numpy backend is inference only
- Drop in replacement for tf.keras when using TensorFlow backend
- Works seamlessly with KerasNLP and KerasCV

Slides courtesy: Aritra and Aakash

Why use Keras 3?

- One API for every major framework (Helps avoid the fragmentation in the ecosystem)
- Seamlessly switch from one framework to another
    - Develop in the most intuitive one
    - Deploy in the fastest one
- Use `keras.ops` package to develop backend agnostic operations
    - Mimics the Numpy API (Same functions same arguments)
    - Includes extra functionalities for Neural Network (softmax, conv, cross_entropy, etc.)
- Highly customizable

One API

```
import os
os.environ["KERAS_BACKEND"] = "tensorflow"
import keras
keras.ops.numpy.arange(5)
>>> <tf.Tensor: shape=(5,), dtype=int32, numpy=array([0, 1, 2, 3, 4], dtype=int32)>
```

```
import os
os.environ["KERAS_BACKEND"] = "jax"
import keras
keras.ops.numpy.arange(5)
>>> Array([0, 1, 2, 3, 4], dtype=int32)
```

```
import os
os.environ["KERAS_BACKEND"] = "torch"
import keras
keras.ops.numpy.arange(5)
>>> tensor([0., 1., 2., 3., 4.])
```

Advantages of Keras 3?

- Support for cross-framework data pipelines
- Pretrained models
- Progressive disclosure of complexity
- Introduces new stateless API for pure functional programming
- Distributed training as easy as non-distributed training

Advantages of Keras 3?

- Support for cross-framework data pipelines
  - tf.data.Dataset
  - torch.utils.data.DataLoader
  - Numpy arrays
  - pandas dataframes
  - PyDatasets
- Pretrained models
- Progressive disclosure of complexity
- Introduces new stateless API for pure functional programming
- Distributed training as easy as non-distributed training

Advantages of Keras 3?

- Support for cross-framework data pipelines
- Pretrained models
- Progressive disclosure of complexity
  - Start simple
  - Customize as per your needs
  - Go from Sequential/Functional to custom train_step to custom loops
- Introduces new stateless API for pure functional programming
- Distributed training as easy as non-distributed training

KerasCV

KerasCV is a library of modular computer vision components that work natively with TensorFlow, JAX, or PyTorch, built on Keras 3.

```
pip install —upgrade keras-cv tensorflow
pip install —upgrade keras
```

Source: https://keras.io/keras_cv/

Pre-trained models supported by KerasCV:

- **EfficientNetV2**: Various sizes of EfficientNet B-style architectures with different width and depth coefficients, boasting high accuracy on ImageNet.
- **MobileNetV3**: Large and small versions with hard-swish activation, optimized for mobile devices, pre-trained on ImageNet.
- **ResNet:** Both v1 and v2 versions with 50 layers, using batch normalization and ReLU activation, trained on ImageNet.
- **YOLOV8:** Different sizes of YOLOV8 backbones pre-trained on COCO for object detection tasks.

And more….

Seamless Switching

```
import os
os.environ["KERAS_BACKEND"] = "tensorflow"
import keras
>>> Using TensorFlow backend
```

```
import os
os.environ["KERAS_BACKEND"] = "jax"
import keras
>>> Using JAX backend
```

```
import os
os.environ["KERAS_BACKEND"] = "torch"
import keras
>>> Using PyTorch backend
```

Key Features:

- Offers models, layers, metrics, callbacks, and more that can be trained and serialized in any framework and re-used in another **without the need for costly migrations**.
- APIs **assist in common computer vision tasks** such as data augmentation, classification, object detection, segmentation, image generation, and more.
- Leverage KerasCV to quickly assemble **production-grade, state-of-the-art** training and inference pipelines for all of these common tasks.

Source: https://keras.io/keras_cv/

From tensorflow import keras?

```
import tensorflow as tf
import tensorflow_datasets as tfds

from keras_cv.backend import keras

import numpy as np

import keras_cv
from keras_cv import bounding_box
from keras_cv import visualization
from keras_cv.backend import ops
```

Do this instead

Key Components:

- **Classification**
- **Object Detection Pipelines**
- **Image Classification and Augmentation**
- **Semantic Segmentation**
- **Image Generation with Stable Diffusion**

Classification

Classification is the process of predicting a categorical label for a given input image. While classification is a relatively straightforward computer vision task. KerasCV provides APIs to construct commonly used components You can solve image classification problems at three levels of complexity:

- **Inference with a pre-trained classifier**

```
classifier = keras_cv.models.ImageClassifier.from_preset(
    "efficientnetv2_b0_imagenet_classifier"
)
```

Backbone model   Task

- **Training a image classifier from scratch (data augmentation, optimizer tuning, model, compile, fit)**

```
backbone = keras_cv.models.EfficientNetV2B0Backbone()
model = keras.Sequential(
    [
        backbone,
        keras.layers.GlobalMaxPooling2D(),
        keras.layers.Dropout(rate=0.5),
        keras.layers.Dense(101, activation="softmax"),
    ]
)
```

No pretrained weights involved

- ## **Fine tuning pre-trained classifier**

When labeled images specific to our task are available, fine-tuning a custom classifier can improve performance. If we want to train a Cats vs Dogs Classifier, using explicitly labeled Cat vs Dog data should perform better than the generic classifier!

```
data, dataset_info = tfds.load("cats_vs_dogs", with_info=True, as_supervised=True)
train_steps_per_epoch = dataset_info.splits["train"].num_examples // BATCH_SIZE
train_dataset = data["train"]

num_classes = dataset_info.features["label"].num_classes

resizing = keras_cv.layers.Resizing(
    IMAGE_SIZE[0], IMAGE_SIZE[1], crop_to_aspect_ratio=True
)


def preprocess_inputs(image, label):
    image = tf.cast(image, tf.float32)
    # Staticly resize images as we only iterate the dataset once.
    return resizing(image), tf.one_hot(label, num_classes)


# Shuffle the dataset to increase diversity of batches.
# 10*BATCH_SIZE follows the assumption that bigger machines can handle bigger
# shuffle buffers.
train_dataset = train_dataset.shuffle(
    10 * BATCH_SIZE, reshuffle_each_iteration=True
).map(preprocess_inputs, num_parallel_calls=AUTOTUNE)
train_dataset = train_dataset.batch(BATCH_SIZE)
```

Pretrained backbones extract more information from our labeled examples by leveraging patterns extracted from potentially much larger datasets.

Object Detection

KerasCV provides facilities to easily construct state-of-the-art object detection pipelines, offering pre-written data loaders and the ability to assemble production-grade data augmentation pipelines with KerasCV preprocessing layers.

```
pretrained_model = keras_cv.models.YOLOV8Detector.from_preset(
    "yolo_v8_m_pascalvoc", bounding_box_format="xywh"
)
```

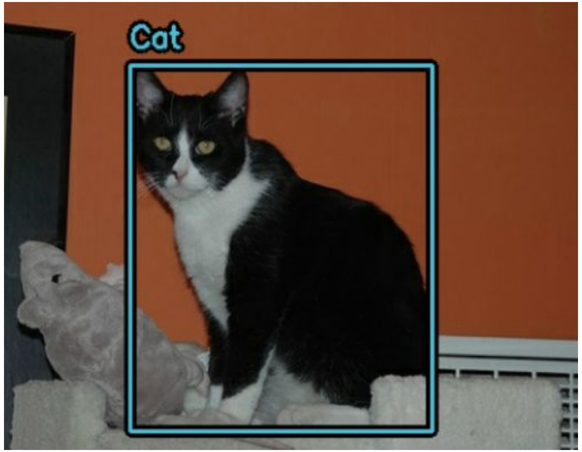
```
bounding_boxes = {
  "classes": [num_boxes],
  "boxes": [num_boxes, 4]
}
```

describes *exactly* what format the values in the `"boxes"` field of the label dictionary take in your pipeline.

Fine tuning pre-trained model

- Load Data

```
train_ds, ds_info = your_data_loader.load(
    split='train', bounding_box_format='xywh', batch_size=8
)
```



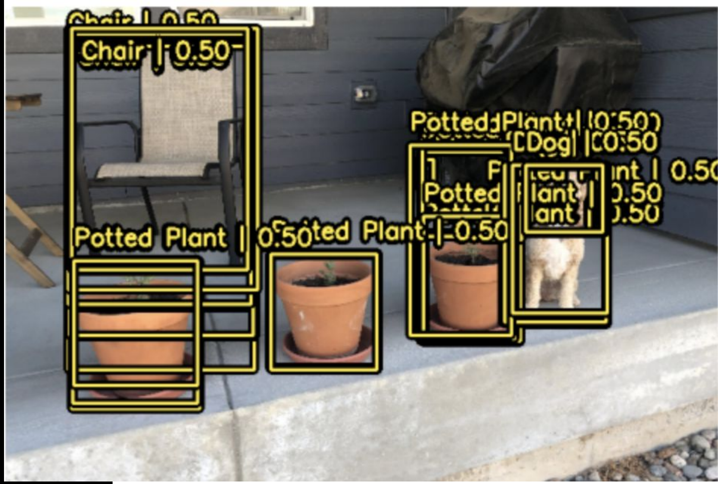
- Data Augmentation

```
augmenters = [
    keras_cv.layers.RandomFlip(mode="horizontal", bounding_box_format="xywh"),
    keras_cv.layers.JitteredResize(
        target_size=(640, 640), scale_factor=(0.75, 1.3), bounding_box_format="xywh"
    ),
]


def create_augmenter_fn(augmenters):
    def augmenter_fn(inputs):
        for augmenter in augmenters:
            inputs = augmenter(inputs)
        return inputs

    return augmenter_fn
```

- Model

```
prediction_decoder = keras_cv.layers.NonMaxSuppression(
    bounding_box_format="xywh",
    from_logits=True,
    # Decrease the required threshold to make predictions get pruned out
    iou_threshold=0.2,
    # Tune confidence threshold for predictions to pass NMS
    confidence_threshold=0.7,
)
pretrained_model = keras_cv.models.YOLOV8Detector.from_preset(
    "yolo_v8_m_pascalvoc",
    bounding_box_format="xywh",
    prediction_decoder=prediction_decoder,
)

y_pred = pretrained_model.predict(image_batch)
visualization.plot_bounding_box_gallery(
    image_batch,
    value_range=(0, 255),
    rows=1,
    cols=1,
    y_pred=y_pred,
    scale=5,
    font_scale=0.7,
    bounding_box_format="xywh",
    class_mapping=class_mapping,
)
```

Non-max suppression is a traditional computing algorithm that solves the problem of a model detecting multiple boxes for the same object.



Check out this notebook by **Ian Stenbit**

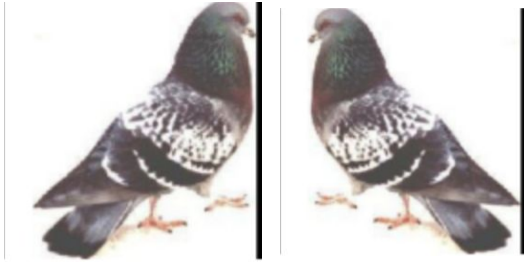
Data Augmentation

Data augmentation is a technique to make your model robust to changes in input data such as lighting, cropping, and orientation. KerasCV includes some of the most useful augmentations in the keras_cv.layers API.

KerasCV supports training powerful image classifiers and includes a suite of preprocessing layers implementing common data augmentation techniques such as CutMix, MixUp, and RandAugment.

Common Augmentations

- RandomFlip()



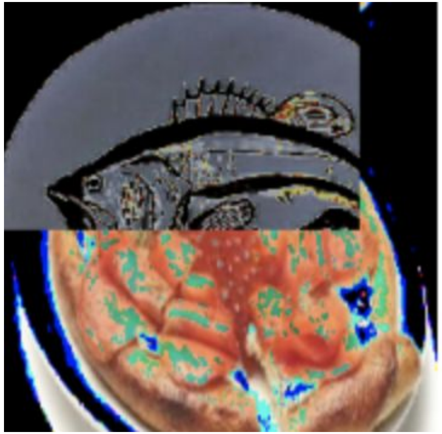
- RandomCropAndResize()

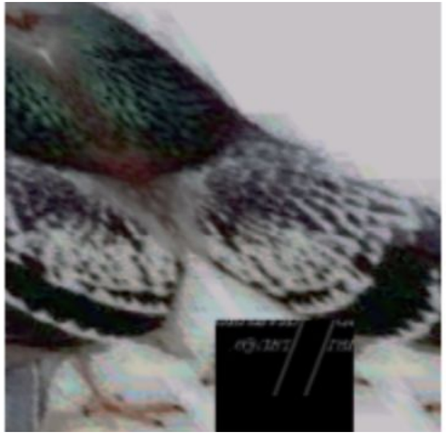
- RandomCutout()

CutMix

Instead of replacing the cut-out areas with black pixels, `CutMix` replaces these regions with regions of other images sampled from within your training set! Following this replacement, the image's classification label is updated to be a blend of the original and mixed image's class label.

```
cut_mix = keras_cv.layers.CutMix()
# CutMix needs to modify both images and labels
inputs = {"images": image_batch, "labels": label_batch}

keras_cv.visualization.plot_image_gallery(
    cut_mix(inputs)["images"],
    rows=3,
    cols=3,
    value_range=(0, 255),
    show=True,
)
```

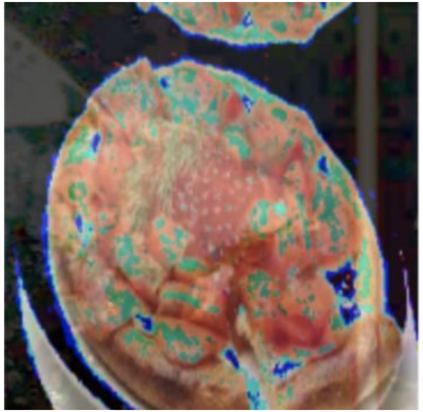


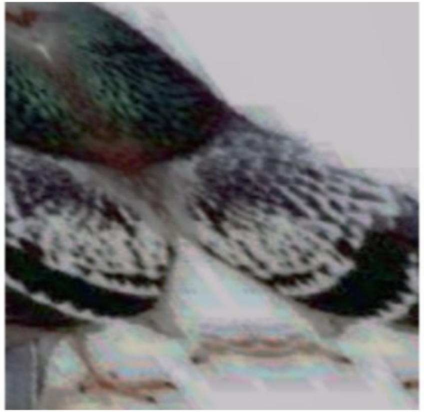


MixUp

`MixUp()` works by sampling two images from a batch, then proceeding to literally blend together their pixel intensities as well as their classification labels.

```
mix_up = keras_cv.layers.MixUp()
# MixUp needs to modify both images and labels
inputs = {"images": image_batch, "labels": label_batch}

keras_cv.visualization.plot_image_gallery(
    mix_up(inputs)["images"],
    rows=3,
    cols=3,
    value_range=(0, 255),
    show=True,
)
```





Segmentation

Semantic segmentation is a type of computer vision task that involves assigning a class label such as person, bike, or background to each individual pixel of an image, effectively dividing the image into regions that correspond to different object classes or categories.

DeepLabv3+ developed by Google for semantic segmentation, combining advanced techniques for accurate and detailed segmentation results.

Semantic segmentation with a pretrained DeepLabv3+ model

```
model = keras_cv.models.DeepLabV3Plus.from_preset(
    "deeplab_v3_plus_resnet50_pascalvoc",
    num_classes=21,
    input_shape=[512, 512, 3],
)
```

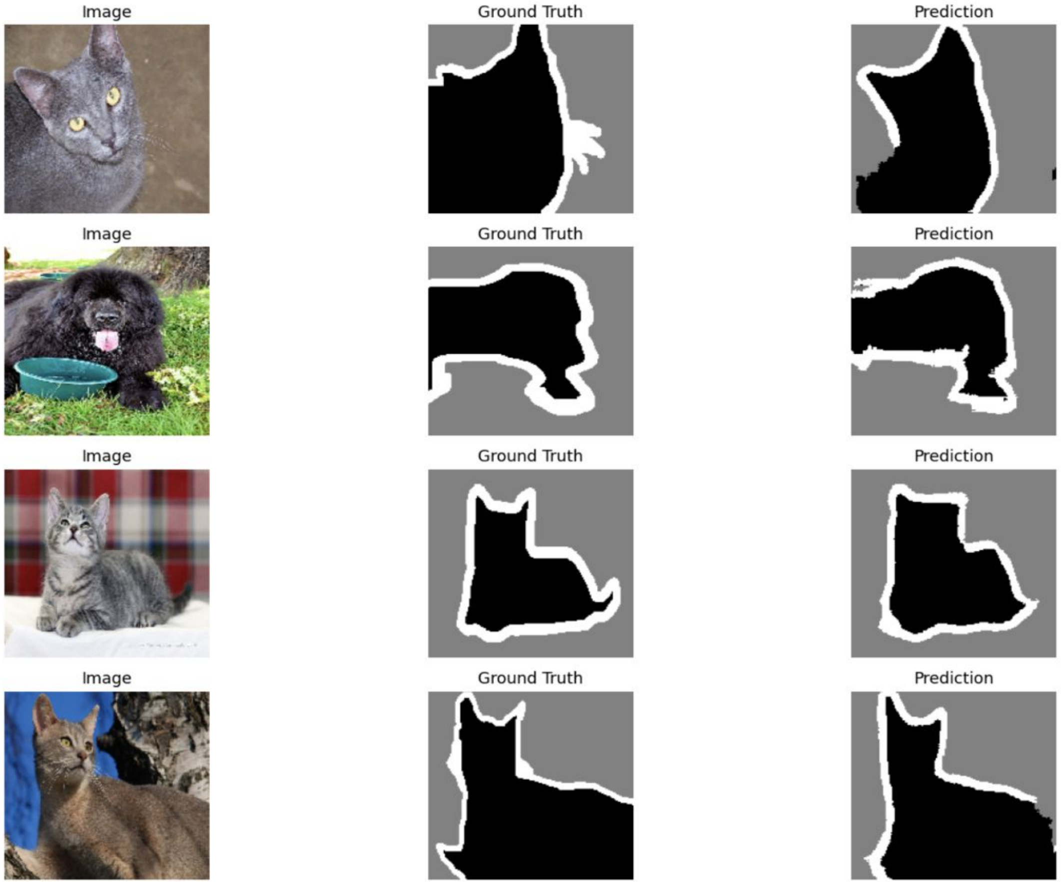


Image
Ground Truth
Prediction
Image
Ground Truth
Prediction
Image
Ground Truth
Prediction
Image
Ground Truth
Prediction

Image Generation with Stable Diffusion

Stable Diffusion is a powerful image generation model that can be used, among other things, to generate pictures according to a short text description (called a "prompt"). To generate novel images based on a text prompt using the KerasCV implementation of stability.ai's text-to-image model, Stable Diffusion.

```
model = keras_cv.models.StableDiffusion(
    img_width=512, img_height=512, jit_compile=False
)
```

```
images = model.text_to_image("photograph of an grinch on christmas", batch_size=3)
import matplotlib.pyplot as plt

def plot_images(images):
    plt.figure(figsize=(20, 20))
    for i in range(len(images)):
        ax = plt.subplot(1, len(images), i + 1)
        plt.imshow(images[i])
        plt.axis("off")
```

Comparison with 🤗 Diffusers

Both implementations were tasked to generate 3 images with a step count of 50 for each image with a Tesla T4 GPU.

GPU	Model	Runtime
Tesla T4	KerasCV (Warm Start)	**28.97s**
Tesla T4	diffusers (Warm Start)	41.33s
Tesla V100	KerasCV (Warm Start)	**12.45**
Tesla V100	diffusers (Warm Start)	12.72

However, the results are still not comparable for cold start. Cold-start execution time includes the one-time cost of model creation and compilation, and is therefore negligible in a production environment (where you would reuse the same model instance many times).

Note that the StableDiffusion API, as well as the APIs of the sub-components of StableDiffusion (e.g. ImageEncoder, DiffusionModel) should be considered unstable at this point. We do not guarantee backwards compatibility for future changes to these APIs.

Resources

- Keras_CV Github: https://github.com/keras-team/keras-cv
- Classification: https://keras.io/guides/keras_cv/classification_with_keras_cv/
- Object Detection: https://keras.io/guides/keras_cv/object_detection_keras_cv/
- Image Augmentation: https://keras.io/guides/keras_cv/custom_image_augmentations/
- Segmentation: https://keras.io/guides/keras_cv/semantic_segmentation_deeplab_v3_plus/
- Image Generation with Stable diffusion:

  https://keras.io/guides/keras_cv/generate_images_with_stable_diffusion/

THANK YOU