HPS

yuvraj.singh@stud.uni-goettingen.de

Yuvraj Singh

What is new in Tensorflow and Keras

11 July 2024

Table of contents

Motivation

- **Embrace the Future**
  - Stay updated with evolving tools like TensorFlow and Keras.
- **Simplified Development**
  - New features and models make fine-tuning and deployment easier.
- **Less Complexity**
  - Simplified usage and accelerated experimentation with minimal code.
- **Increased Efficiency & Compatibility**
  - Improved frameworks save time and resources.
- **Create Impact**
  - Use these tools to drive change and deliver real-world solutions.

TensorFlow

- Free and open-source software library for ML and DL
- Developed by - Google Brain Team
- Major Milestones
  - February 2017 - Release of TensorFlow 1.0
  - March 2018 - TensorFlow Extended (TFX) for end-to-end deploying platform
  - September 2019 - TensorFlow 2.0 with major API changes
  - TensorFlow.js for machine learning in JavaScript

TensorFlow

- Some Features
  - Suitable for both research and production
  - Can be used in a variety of programming languages, like Python & C++
  - Cross-Platform development
- Current Version - TensorFlow 2.16.1 (9 March 2024)
  - Keras 3 will be the default Keras version for TensorFlow 2.16 onward

Keras

- **Keras**
  - High-level neural networks API, written in Python
  - Runs on top of TensorFlow, Torch, or JAX
  - Designed for fast experimentation, with a simple & user-friendly interface
  - Modular and extensible for easy customization
  - Supports a wide range of neural network architectures (CNNs, RNNs, Transformers, etc.)

JAX & JAX2TF

- **JAX**
  - High-performance numerical computing library developed by Google
  - Leverage power of GPU/TPU
  - **Challenge**: Lack of built-in deployment tools
- **JAX2TF**
  - A lightweight API that links JAX and TensorFlow
  - **Inference**: Deploy JAX models on servers/devices using TensorFlow
  - **Fine Tuning**: Continue training JAX-trained models in TensorFlow

Comparison: TensorFlow vs. PyTorch

- **Common Features**
  - GPU acceleration
  - Large, active communities
  - Flexible APIs for deep learning
- **TensorFlow**
  - Comprehensive ecosystem (i.e. Hub, Graphics, Keras)
  - TFX: Strong production and scalability support
  - Offers more tools for custom features
  - Native Keras integration
- **PyTorch**
  - Favored by researchers for ease & quick prototyping
  - Substantially less training time

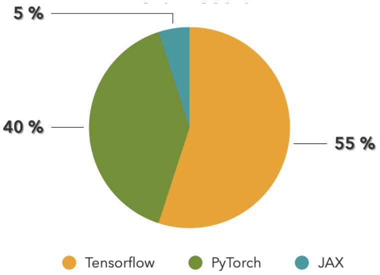


Figure: StackOverflow: Market share survey 2023

Outline

1 Introduction to TensorFlow & Keras

2 **Hardware Compatibility**

3 KerasCV

4 KerasNLP

5 Dtensor

Hardware Compatibility in TensorFlow

- **Apple Silicon**
  - Native support for Apple M chips
  - Optimized performance with Apple's ML Compute framework
- **NVIDIA GPUs**
  - NVIDIA CUDA libraries for Linux (Onwards tf 2.15)
  - Upgrade to Clang 17.0.1 and CUDA 12.2 (tf 2.15)
  - TensorFlow container images available with GPU support
- **Cross-platform Compatibility**
  - Runs on a variety of hardware: CPUs, GPUs, & TPUs
  - Supports deployment on cloud platforms like Google Cloud, AWS, & Azure
  - TensorFlow-Lite for mobile & embedded devices i.e. RaspberryPi & Jetson

Outline

1 Introduction to TensorFlow & Keras

2 Hardware Compatibility

3 KerasCV

4 KerasNLP

5 Dtensor

What is KerasCV

- **KerasCV**: Keras framework extension, which focuses on computer vision
  - Provides tools for image preprocessing, augmentation, and model training
  - Includes state-of-the-art models for image classification, object detection, segmentation, and more
  - Designed to be user-friendly and highly customizable
  - Well documented with examples

What is new in KerasCV - I

- **Advanced Data Augmentation**
  - API to apply complex augmentations with minimal code
  - Advanced augmentation layers, i.e., RandAugment
- **Object Detection & Image Classification**
  - Access to state of state-of-the-art models, i.e., YOLO
  - Possible to fine-tuning pre-trained models
- **Benefits**
  - Reduces effort for data augmentation
  - Reduces training time & faster convergence.
  - Improves accuracy and robustness of the model

KerasCV Practical Task-I

- **Finetuning pre-trained YOLOv8 model for DJI 300 RTK drone**
  - **Objective**: To detect and track DJI 300 RTK drone flying at a significant distance in complex settings including lighting conditions, rapid camera movements, etc.

Figure: Images of DJI 300 RTK Drone

KerasCV Practical Task-I

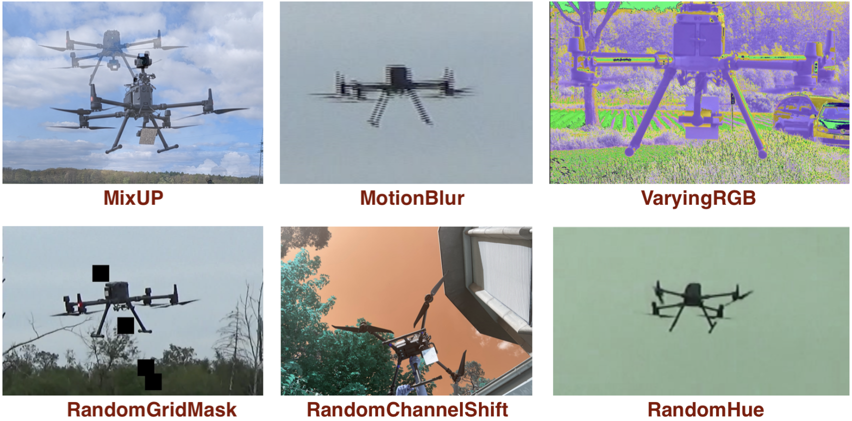
Task Outline

- Advanced data augmentation
- **Model-1**: Finetuning pre-trained YOLOv8 without data augmentation
- **Model-2**: Finetuning pre-trained YOLOv8 with data augmentation
- Performance comparison of models
- Challenges

Dataset Preparation

- HD images were taken using iPhone 12 & DSLR
- Images were taken in daylight - during flight, and indoors
- 671 labeled images and 50 background images, using makesense.ai

KerasCV Practical Task-I: Data augmentation



MixUP

MotionBlur

VaryingRGB

RandomGridMask

RandomChannelShift

RandomHue

Figure: Random samples of augmentation performed

KerasCV Practical Task-I: Model Training

	Model-1	**Model-2**
GPU Used	NVIDIA A100	NVIDIA A100
Model Used	YOLOv8 m	YOLOv8 m
Dataset size	671 images	3.177 images
Background images	50 images	185 images
Training image size	1080 px	1080 px
Training time	26 mins	125 mins

Table: Training settings for Model-1 & Model-2

KerasCV Practical Task-I: Model Evaluation

	Description
Test 1	Video recorded in a similar setting as training dataset
Test 2	Completely unseen, lower resolution, & distant flying
Test 3	Unseen, lower resolution, distant flying & complex background

Table: Description of tests performed

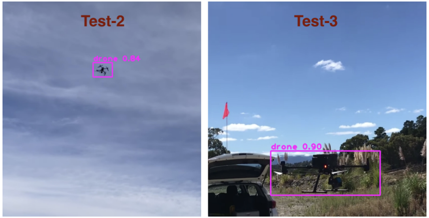


Figure: Shots from Test 2 & 3

KerasCV Practical Task-I: Model Evaluation

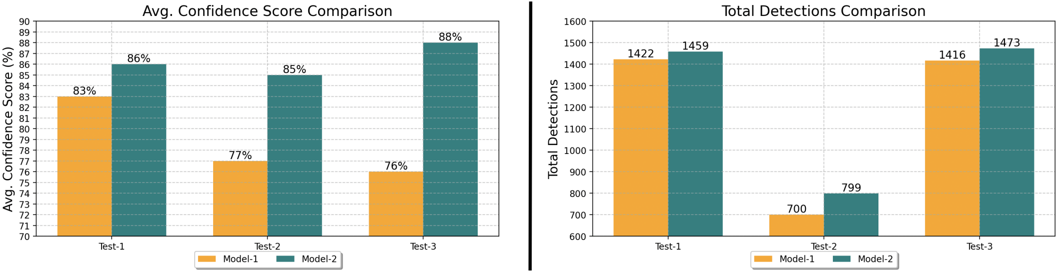


Figure: Model 1 & 2 performance comparison

KerasCV Practical Task-I: Challenges

- **Possible improvements**
  - Advanced trackers like CSRT can hold pixels if the model fails until recovery.
- **Model deployment**
  - High-performance models on devices like Pi compromise performance
  - Can be deployed on embedded devices like Jetson boards. i.e. Jetson Orin
  - Use a high-resolution camera with appropriate bandwidth

What is New in KerasCV-II

- **High-performance image generation using Stable Diffusion**
  - Implement Stable Diffusion using KerasCV
  - Generate high-quality, realistic images at no cost
  - Generate images based on text prompts
  - Suitable for applications in art, design, and entertainment

Listing: Text prompt to generate image using Keras-Stable Diffusion

```
import keras_cv
  model = keras_cv.models.StableDiffusion(
      img_width=512, img_height=512, jit_compile=False
  )
  images = model.text_to_image("steampunk airship, flying in the sky,
      ↪ intricate mechanical details, Victorian era style", batch_size=1)
```

KerasCV Practical Task-II

Steampunk airship

Majestic celestial dragon

Alien planet landscape

Enchanted forest

Ancient library

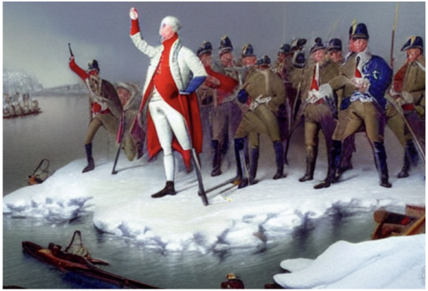
Futuristic cityscape at night

Figure: Image generation using Stable Diffusion on Mac M1

KerasCV Practical Task-II

Limitations of Stable Diffusion

- High specification hardware required (10-30GB VRAM GPU)
- Can only generate images of up to 1024x1024, standard size 512x512
- Biased towards English language and Western culture
- Generates unrealistic faces & limbs



George Washington and troops

Outline

1 Introduction to TensorFlow & Keras

2 Hardware Compatibility

3 KerasCV

4 KerasNLP

5 Dtensor

What is KerasNLP

- **KerasNLP** is an extension of the Keras to focus on NLP tasks.
  - Provides tools for text preprocessing, tokenization, and model training.
  - Offers state-of-the-art models for text generation, text classification, machine translation, and more.
  - Designed to be user-friendly and highly customizable.

KerasNLP Practical Task-I: Text Generation

- **Text Generation using pre-trained GPT-2 model**
  - Utilize pre-trained GPT-2 weights for generating text sequences
  - Auto-generate text from prompts, with just a few lines of code
  - Fully integrated with the TensorFlow ecosystem

Listing: Loading and initializing GPT-2 mode

```
#default length: 1024 words.
preprocessor =
keras_nlp.models.GPT2CausalLMPreprocessor.from_preset(
    "gpt2_base_en",
    sequence_length=128,
)
gpt2_lm = keras_nlp.models.GPT2CausalLM.from_preset(
    "gpt2_base_en", preprocessor=preprocessor
)
```

KerasNLP Practical Task-I: Text Generation

Listing: Example prompt to generate text

```python
output = gpt2_lm.generate("Autonomous vehicles (AV)", max_length=100)
print("\nGPT-2 output:")
print(output)
```

Output:

```
Autonomous vehicles (AV) and unmanned vehicles (UAVs) could become a major
    ↪ driver of transportation in the future, said an industry group.
The first autonomous vehicles were introduced in the United States in the
    ↪ 1970s. They were used in commercial vehicles to carry passengers,
    ↪ carry goods and carry out other tasks, and to carry out military
    ↪ missions.
```

KerasNLP Practical Task-I: Fine-tuning Text Generator

- **Fine-tuning pre-trained GPT-2 model**
  - Customize GPT-2 on your specific text dataset
  - Improve performance on domain-specific tasks
  - Steps involved:
    - Load and preprocess your dataset
    - Configure the training parameters
    - Train fine-tune the model

GPT Backbone	gpt2_base_en
Description of Text Used	Ethical issues with Autonomous Vehicles
Trained for Epochs	10
Training Time	7.5 Minutes

Table: Fine-tuning details

KerasNLP-Practical Task: Fine-tuning Text Generator

Listing: Example prompt for finetuned text generator

```
output = gpt2_lm.generate("Autonomous vehicles (AV)", max_length=100)
print("\nGPT-2 output:")
print(output)
```

Output after fine-tuning:

```
Autonomous vehicles (AV) are autonomous vehicles designed to operate safely.
    ↪ Autonomous vehicles (AV) represent a remarkable technological
    ↪ development in road transportation. Autonomous vehicles (AV) are
    ↪ highly dependent on data coming from the sensors, AI, and software
    ↪ making decisions.
```

Outline

1 Introduction to TensorFlow & Keras

2 Hardware Compatibility

3 KerasCV

4 KerasNLP

5 Dtensor

DTensor Overview

- DTensor: TensorFlow extension for distributed computing
- **Inspiration & Idea**
  - Scale up models and train them efficiently by combining and fine-tuning multiple parallelism techniques
- **Example Case**
  - Building transformer model, like the Open Pre-trained Transformer (OPT) through KerasNLP
- Global programming model for Tensors, manages distribution internally
- DTensor decoupling allows the App to run on multiple devices/clients

DTensor's Model of Distributed Tensors

- **Mesh**
  - Defines the group of devices for computation
  - Supports CPUs, GPUs, or TPUs
  - Represents available hardware resources
- **Layout**
  - Specifies how tensor is divided and spread across devices
  - Indicates which parts of the tensor go to which devices
- **Summary**
  - **Mesh**: Defines list of devices
  - **Layout**: Defines distribution of tensor across devices

Mesh Simplified

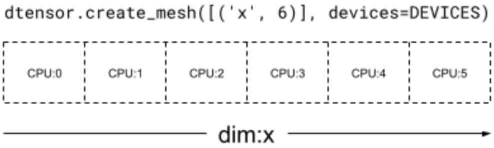
Example

- Hardware: 6 virtual CPUs

```
configure_virtual_cpus(6)
DEVICES = [f'CPU:{i}' for i in range(6)]
```

- **1D Mesh:** 6 CPU devices along a mesh dimension ’x’

```
mesh_1d = dtensor.create_mesh([('x', 6)], devices=DEVICES)
```

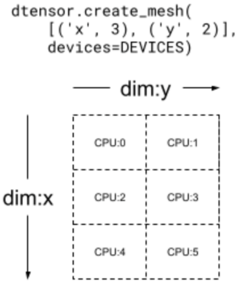


```
dtensor.create_mesh([('x', 6)], devices=DEVICES)
```

Mesh Simplified

- **Example**
  - **Multi-Dimensional Mesh:** A grid with more than one dimension

```
mesh_2d = dtensor.create_mesh([('x', 3),
    ↪ ('y', 2)], devices=DEVICES)
```



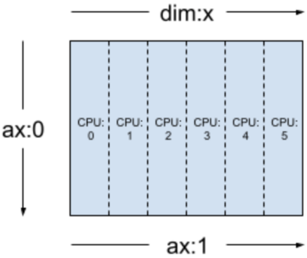
```
dtensor.create_mesh(
    [('x', 3), ('y', 2)],
    devices=DEVICES)
```

Layout Simplified

1D Mesh (6 devices)

- Shard the second axis of tensor across 6 devices

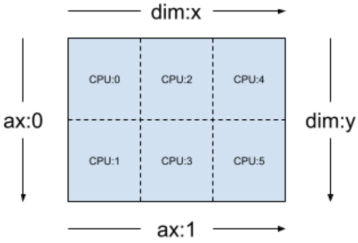
```
layout = dtensor.Layout([dtensor.UNSHARDED, ’x’], mesh_1d)
```



Layout Simplified

- **2D Mesh (3x2 devices)**
  - First axis sharded across ’y’, second axis across ’x’ of rank-2 Tensor

```
layout = dtensor.Layout([’y’, ’x’], mesh_2d)
```



Conclusion

- **Explored TensorFlow Ecosystem:**
  - Key features and advancements in TensorFlow and Keras
  - Enhanced hardware compatibility for Apple Silicon, NVIDIA GPUs
- **KerasCV & KerasNLP:**
  - **KerasCV Project 1:** Fine-tuning YOLOv8 for drone detection
  - **KerasCV Project 2:** Image generation using Stable Diffusion
  - **KerasNLP:** Trying GPT-2 model text generation and fine-tuning
- **Introduction to DTensor:**
  - Distributed training techniques with DTensor for efficient scaling

Sources

1 https://blog.tensorflow.org/2023/05/google-io-2023-whats-new-in-tensorflow-and-keras.html

2 https://www.tensorflow.org/guide/dtensor_overview

3 https://keras.io/guides/keras_cv/object_detection_keras_cv/

4 https://keras.io/guides/keras_cv/classification_with_keras_cv/

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

6 https://keras.io/guides/keras_cv/cut_mix_mix_up_and_rand_augment/

7 https://www.tensorflow.org/tutorials/images

8 https://keras.io/examples/generative/gpt2_text_generation_with_kerasnlp/

Mesh Simplified

- **Logical Grid**
  - Organizes devices into a grid with named dimensions
  - Each dimension is called a "Mesh dimension"
- **Unique Names**
  - Each dimension in the same Mesh must have a unique name
- **Reference by Layout**
  - Names of Mesh dimensions are used by Layout to describe tensor division
- **Multi-Dimensional Array**
  - Mesh as a multi-dimensional array, where each element is a device

Layout Simplified

- **Terms**
  - Dimension: Linked to the Mesh
  - Axis & Rank: Linked to the Tensor and Layout
- **Rank**
  - The rank (number of axes) of the Layout must match the rank of the Tensor
- **Sharding**
  - Each axis of the Tensor can be sharded across a Mesh dimension.
  - Tensor could also remain "UNSHARDED"
- **Matching Dimensions and Axes**
  - Number of Layout axes does not need to match number of Mesh dimensions