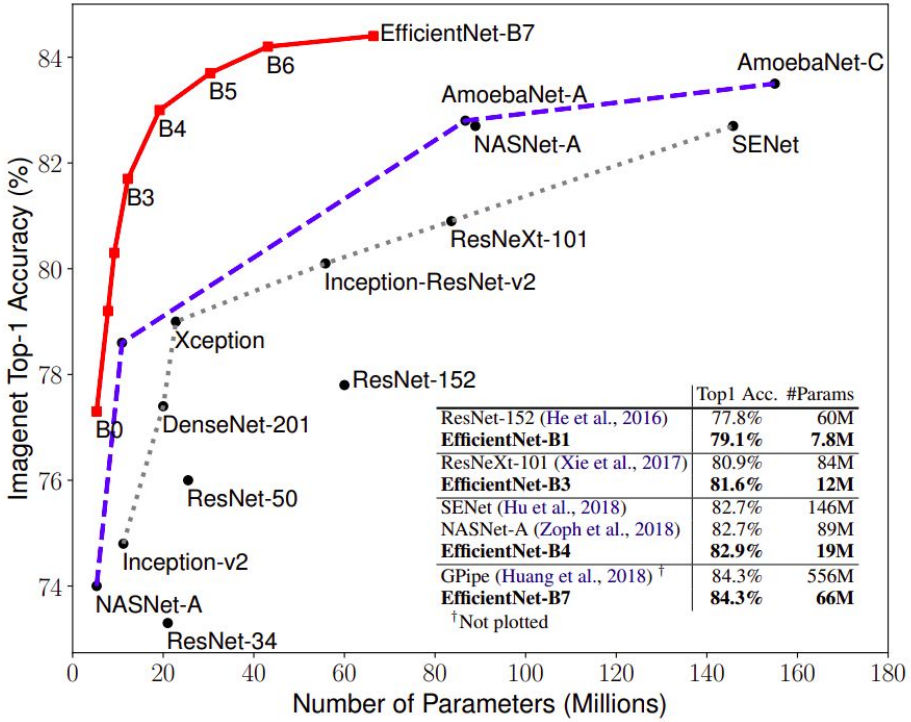
Transfer Learning and Tuning Model

Lesson of Content

1. EfficientNet
2. Transfer Learning
3. Tuning Model
4. Model Versioning

Link papers: https://arxiv.org/pdf/1905.11946.pdf
Year: 12.2020

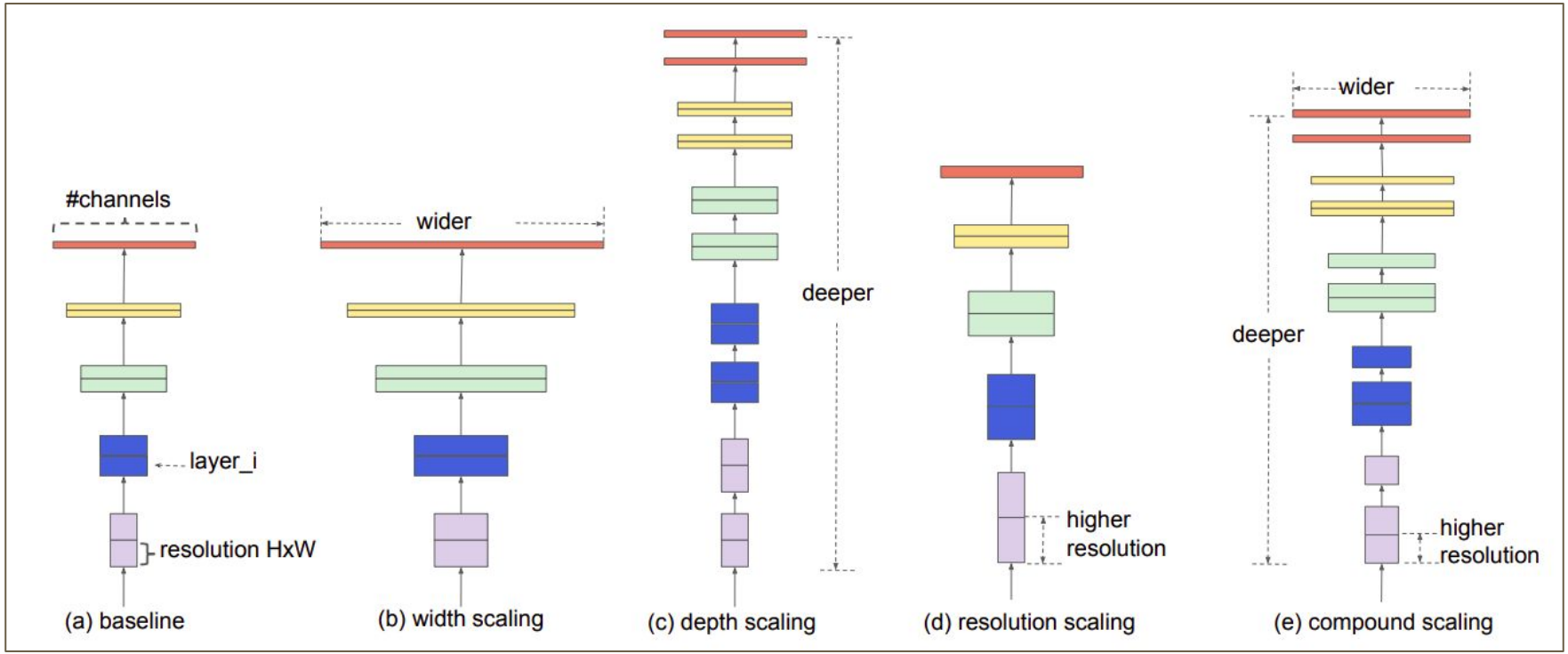


	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

To go even further, we use neural architecture search to design a new baseline network and scale it up to obtain a family of models, called *EfficientNets*, which achieve much better accuracy and efficiency than previous ConvNets. In particular, our EfficientNet-B7 achieves state-of-the-art 84.3% top-1 accuracy on ImageNet, while being **8.4x smaller** and **6.1x faster** on inference than the best existing ConvNet. Our EfficientNets also transfer well and achieve state-of-the-art accuracy on CIFAR-100 (91.7%), Flowers (98.8%), and 3 other transfer learning datasets, with an order of magnitude fewer parameters. Source code is at `https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet`.

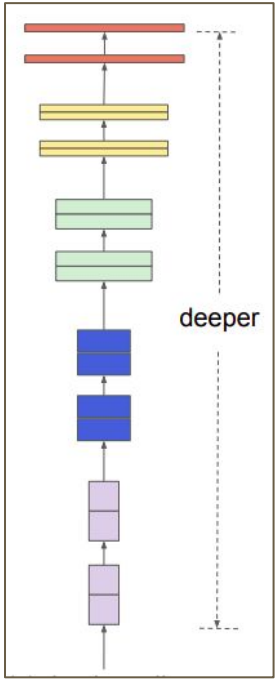
Main Idea



Width scaling: Mobilenet
Depth scaling: Resnet

Resolution scaling: Resize Image
Compound scaling: EfficientNet

Depth Scaling



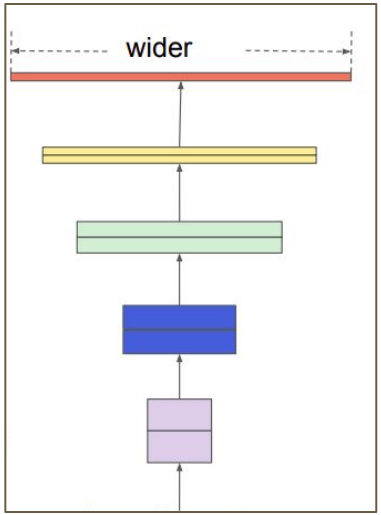
layer name	output size	18-layer	34-layer
conv1	112×112		
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$
	1×1	ave	
FLOPs		1.8×10^9	3.6×10^9

Resnet Architecture

Wide Scaling

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Table 1: *Bottleneck residual block* transforming from k to k' channels, with stride s, and expansion factor t.



Mobilenet v2

Main Idea

ConvNet Layer thứ i là một function sau: $Y_i = F_i(X_i)$

Với X_i là $< H_i, W_i, C_i >^1$

$$N = F_k \odot F_{k-1} \odot \ldots \odot F_1(X_1) = \bigoplus_{j=1\ldots s} F_j(X_{<H_j, W_j, C_j>})$$

Mỗi lớp ConvNet có thể lặp lại L lần

$$\Rightarrow N = \bigoplus_{j=1\ldots s} F_j^{L_j}(X_{<H_j, W_j, C_j>})$$

$$N(d, w, r) = \bigoplus_{j=1\ldots s} F_j^{d.\hat{L}_j}(X_{<r.\hat{H}_j, r.\hat{W}_j, w.\hat{C}_j>})$$

$$\max_{d,w,r} Accuracy(N(d, w, r))$$

Thay vì tìm hàm F để tối ưu hiệu suất mô hình.

Ý tưởng chính: Giữ nguyên hàm F, và phóng to, thu nhỏ mô hình để vừa đạt hiệu suất mô hình và chạy tốt trên các thiết bị hạn chế tài nguyên

Trong đó w, d, r lần lượt là hệ số tỷ lệ chiều rộng, chiều sâu và độ phân giải của ảnh.
$\hat{F}_i, \hat{L}_i, \hat{H}_i, \hat{W}_i, \hat{C}_i$ lần lượt là hàm, số lần lặp và kích thước ảnh được giữ cố định

Main Idea

$$\text{depth: } d = \alpha^\phi$$
$$\text{width: } w = \beta^\phi$$
$$\text{resolution: } r = \gamma^\phi$$
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$
$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

scaling a ConvNet with equation 3 will approximately increase total FLOPS by $(\alpha \cdot \beta^2 \cdot \gamma^2)^\phi$. In this paper, we constraint $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$ such that for any new ϕ, the total FLOPS will approximately[3] increase by 2^ϕ.

- STEP 1: we first fix $\phi = 1$, assuming twice more resources available, and do a small grid search of α, β, γ based on Equation 2 and 3. In particular, we find the best values for EfficientNet-B0 are $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$, under constraint of $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$.
- STEP 2: we then fix α, β, γ as constants and scale up baseline network with different ϕ using Equation 3, to obtain EfficientNet-B1 to B7 (Details in Table 2).

Table 1. **EfficientNet-B0 baseline network** – Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224 × 224	32	1
2	MBConv1, k3x3	112 × 112	16	1
3	MBConv6, k3x3	112 × 112	24	2
4	MBConv6, k5x5	56 × 56	40	2
5	MBConv6, k3x3	28 × 28	80	3
6	MBConv6, k5x5	14 × 14	112	3
7	MBConv6, k5x5	14 × 14	192	4
8	MBConv6, k3x3	7 × 7	320	1
9	Conv1x1 & Pooling & FC	7 × 7	1280	1

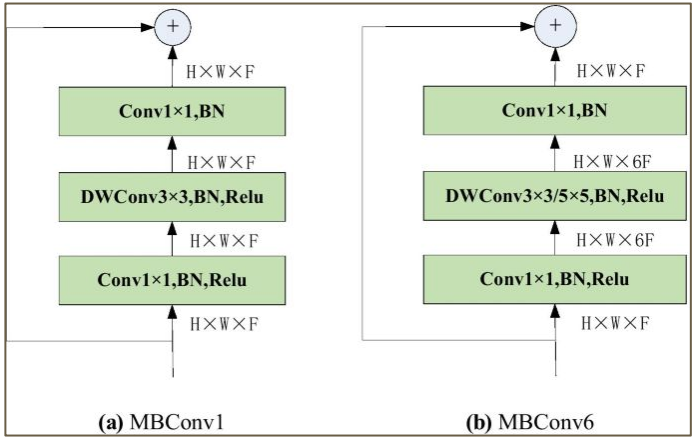
Main Idea

Table 1. **EfficientNet-B0 baseline network** – Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

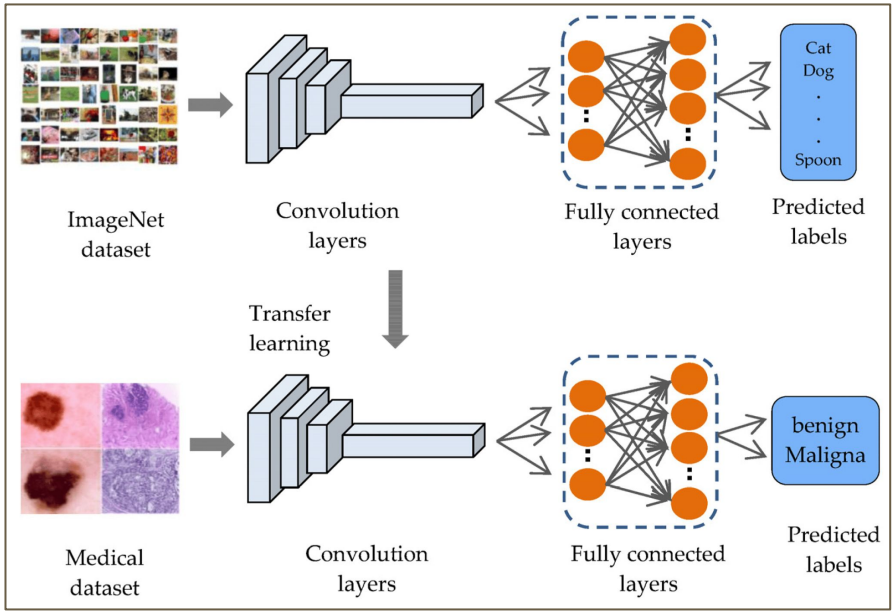
EfficientNet-B3

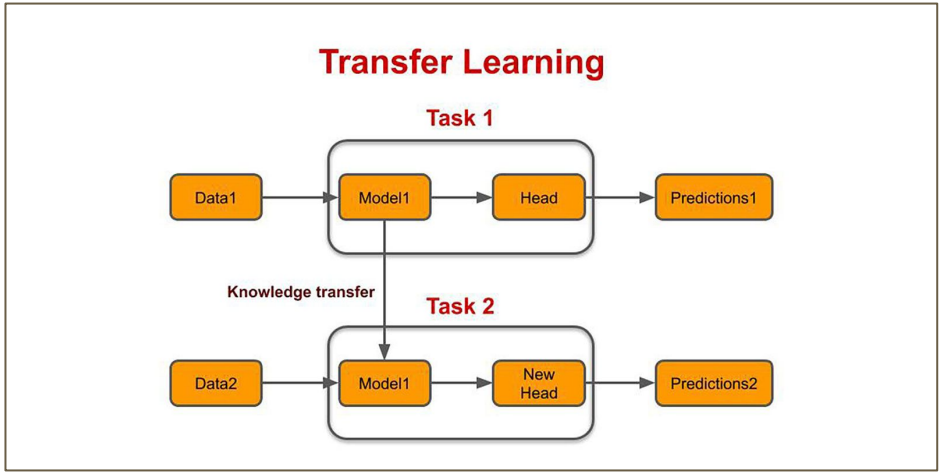
Block N0. (i)	Layer (F_i ( ))	Resolution (H_i x W_i)	No. of Layers (L_i)
1	Conv 3x3	300x300	1
2	MBConv1, 3x3	150x150	2
3	MBConv6, 3x3	150x150	3
4	MBConv6, 5x5	75x75	3
5	MBConv6, 3x3	38x38	5
6	MBConv6, 5x5	19x19	5
7	MBConv6, 5x5	10x10	6
8	MBConv6, 3x3	10x10	2
9	Conv 1x1	10x10	1
10	Global Pooling	10x10	1
11	Dense layer	10x10	1



r: resolution
w: wide
d: depth

Transfer Learning: Là việc tái sử dụng lại kiến trúc đã học và huấn luyện trên một tác vụ cụ thể nhằm cải thiện hiệu suất.





How to use ?

Torchvision (Computer Vision)

```
import torch
import torchvision.models as models

model = models.resnet18(pretrained=True)
num_classes = 10 # Define the number of classes for 10
in_features = model.fc.in_features
model.fc = torch.nn.Linear(in_features, num_classes)

for name, param in model.named_parameters():
	if 'layer4' in name or 'fc' in name:
		param.requires_grad = True
	else:
		param.requires_grad = False

# Dataloader
# OPtimizer
# Training Loop
```

Huggingface (NLP)

```
import torch
from transformers import XLMRobertaTokenizer,
XLMRobertaForSequenceClassification
from transformers import Trainer, TrainingArguments
from sklearn.model_selection import train_test_split
from datasets import load_dataset

# Load pre-trained XLM-RoBERTa tokenizer and model
model_name = "xlm-roberta-base"
tokenizer = XLMRobertaTokenizer.from_pretrained(model_name)
num_classes = 2
model = XLMRobertaForSequenceClassification.from_pretrained(model_name,
num_labels=num_classes)

# Dataloader
# Optimizer
# Training Loop
```

What is Optuna ?

Optuna: Dùng cho việc tối ưu siêu tham số của các mô hình. Giúp tìm kiếm nhanh và hiệu quả hơn với việc **Random Tuning.**

Optuna cung cấp một số thuật toán tối ưu hóa thông minh như:

- **Bayesian Optimizer (default)**
- Tree-Structured Parzen Estimators (TPE)

How to use Optuna?

```python
# Hàm huấn luyện mô hình
def train_model(dataloader, learning_rate, dropout_rate, beta1, beta2):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    model = ResNet18(num_classes=10, dropout_rate=dropout_rate).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate, betas=(beta1, beta2))

    for epoch in range(10):
        model.train()
        for inputs, targets in dataloader:
            inputs, targets = inputs.to(device), targets.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()
        f1 = calculate_f1_score(model, dataloader, device)
    return f1

```

```python
def objective(trial):
    batch_size = trial.suggest_categorical("batch_size", [2, 4, 6, 8, 16, 32])
    learning_rate = trial.suggest_float("learning_rate", 1e-5, 1e-1, log=True)
    dropout_rate = trial.suggest_float("dropout_rate", 0.0, 0.4)
    beta1 = trial.suggest_float("beta1", 0.9, 0.99, step=0.01)
    beta2 = trial.suggest_float("beta2", 0.99, 0.999, step=0.001)

    custom_dataset = CustomDataset(batch_size=batch_size)
    dataloader = custom_dataset.create_dataloader()

    f1 = train_model(dataloader, learning_rate, dropout_rate, beta1, beta2)
    return f1

```

```python
# suggest_float: Đề xuất một giá trị số thực trong khoảng [low, high]
learning_rate = trial.suggest_float("learning_rate", 1e-5, 1e-1, log=True)

# suggest_int: Đề xuất một giá trị số nguyên trong khoảng [low, high]
num_epochs = trial.suggest_int("num_epochs", 10, 100)

# suggest_categorical: Đề xuất một giá trị từ một danh sách rời rạc
optimizer = trial.suggest_categorical("optimizer", ["Adam", "SGD", "RMSprop"])

# suggest_uniform: Đề xuất một giá trị liên tục trong khoảng [low, high]
dropout_rate = trial.suggest_uniform("dropout_rate", 0.0, 0.5)
```

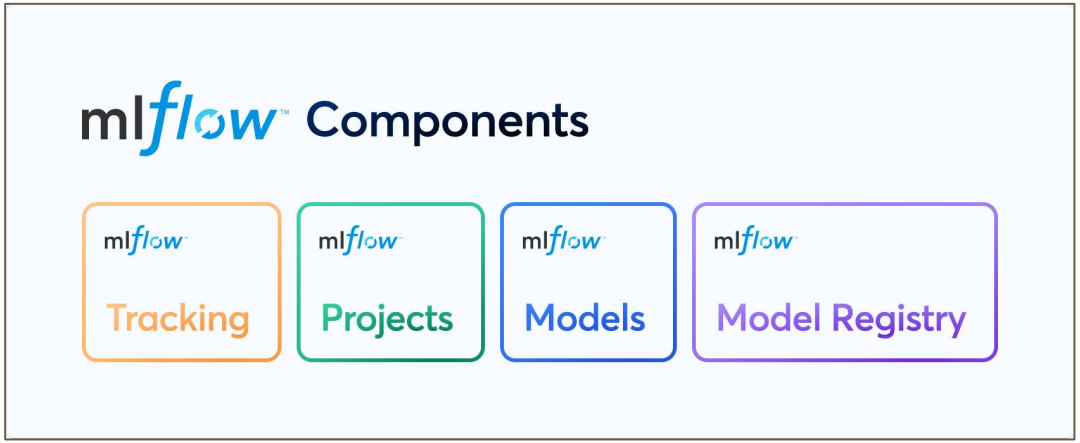
```python
# Khởi tạo optuna (Có thể là maximize hoặc minimize tùy vào hàm trả về là gì)
study = optuna.create_study(direction="maximize")

# Khởi chạy Tuning model với 50 lần thử nghiệm
study.optimize(objective, n_trials=50)

# Lấy được best parameter
trial = study.best_trial

# Print ra cặp keys-values
for key, value in trial.params.items():
    print(f"{key}: {value}")
```

What is Mlfow

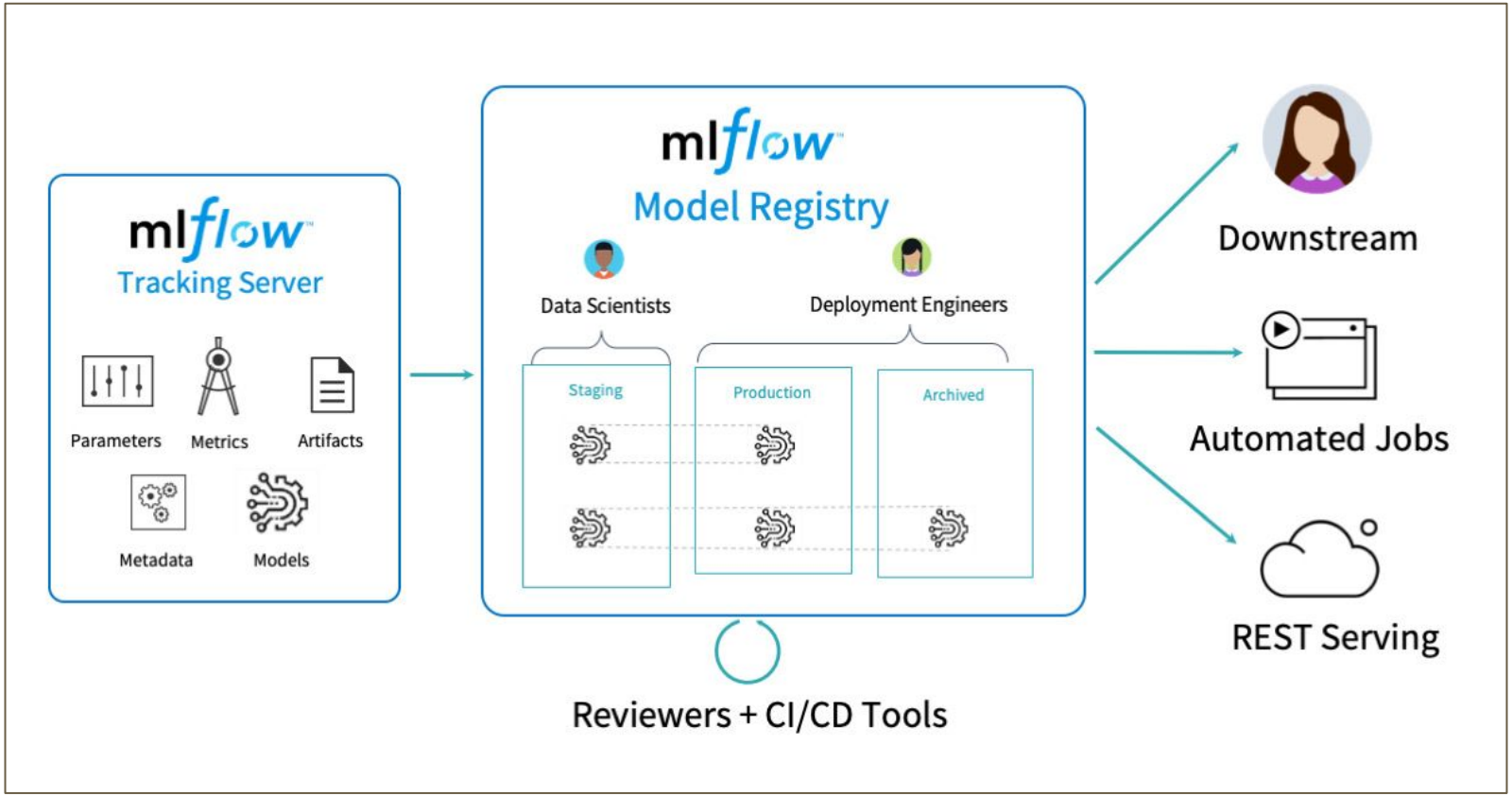


Là một mã nguồn mở, hỗ trợ việc quản lý, theo dõi và đánh giá mô hình từ đầu đến cuối.

Có 4 sản phẩm chính.

- Tracking (Theo dõi): Cho phép ghi lại các thông số về đào tạo mô hình như hyperparameters và kết quả sau khi đào tạo (metrics). Giúp so sánh hiệu suất của các mô hình với nhau.
- Projects (Dự án) : Cho phép tổ chức code và document của model. Glúp quản lý, tái sử dụng và chia sẻ mã nguồn dễ dàng hơn.
- Registry (Repos): Registry là một trung tâm lưu trữ các mô hình đã đào tạo. Nó giúp bạn theo dõi và quản lý các phiên bản của mô hình.
- Models (Mô hình): Cho phép triển khai mô hình đó lên các môi trường hoặc một bên khác sử dụng.

What is Mlfow



How to use mlflow ?

MLflow

Cài đặt thư viện:

! pip install mlflow
! pip install pydantic

Khởi chạy mlflow server trên local:

mlflow server --port 5001

Bạn phải luôn chạy câu lệnh này trước khi huấn luyện mô hình.

```python
# init mlflow
import mlflow
import mlflow.pytorch
mlflow.set_tracking_uri("http://127.0.0.1:5000")

with mlflow.start_run(run_name="experiments"):
        # log paramerter
        mlflow.log_param("batch_size", batch_size)
        mlflow.log_param("learning_rate", learning_rate)
        mlflow.log_param("dropout_rate", dropout_rate)
        mlflow.log_param("beta1", beta1)
        mlflow.log_param("beta2", beta2)
        mlflow.log_param("type_model", "resnet18")
        f1, best_model = train_model(...)
        mlflow.log_metric("f1_score", f1)
        mlflow.pytorch.log_model(best_model, artifact_path="pytorch-model")

# Load model from a specific run_id
run_id = "5a157f6cea1d4949806615b0ee7bdbd7"
loaded_model = mlflow.pytorch.load_model(f"runs:/{run_id}/pytorch-model")

# Load by stage
model_name = "production"
stage = "Production"
model = mlflow.pytorch.load_model(model_uri=f"models:/{model_name}/{stage}")
```