Pre-Trained Models in Deep Learning

What is Pre-Trained Model

A **pre-trained model** is a **machine learning or deep learning model that has already been trained** on a **large dataset** for a **general task** — and then shared publicly so that others can reuse it **without training from scratch**.

Why Do We Use Pre-trained Models?

Training big models from scratch needs:

- **Huge datasets** (millions of examples)
- **High computational power** (GPUs/TPUs)
- **A lot of time** (hours or even weeks)

So instead, we take a **model that's already trained** and then:

- Use it **as-is** (for prediction tasks)
- Or **fine-tune** it (adjust it slightly for our specific problem)

➡ This saves **time, cost, and resources** — and often gives **better accuracy**.

How Pre-trained Models Work

Let's imagine we are using a pre-trained model for image classification, like **ResNet**, trained on **ImageNet** (1.2 million images, 1000 classes).

Step 1: Pre-training

- Researchers train ResNet on ImageNet to recognize general image features like edges, textures, shapes, etc.
- The model learns to extract meaningful visual representations.

Step 2: Reuse / Fine-tuning

Now, if you want to build your own model (say to classify **X-ray images**):

- You take the **pre-trained ResNet**
- Keep its learned weights (knowledge of edges, shapes)

- Change only the **final layers** to suit your new task (X-rays → healthy/sick)
- Train again for a short time with your smaller dataset.

This is called **transfer learning** — because knowledge is **transferred** from one task to another.

Where Pre-trained Models Are Used

Domain	Common Pre-trained Models	Typical Tasks
Computer Vision	ResNet, VGG, Inception, EfficientNet, YOLO	Object detection, Image classification, Segmentation
Natural Language Processing (NLP)	BERT, GPT, T5, RoBERTa	Text classification, Question answering, Chatbots
Speech	Wav2Vec, Whisper, DeepSpeech	Speech-to-text, Speaker recognition
Audio / Music	YAMNet, OpenL3	Sound event detection, Music genre classification
Healthcare / Biology	BioBERT, AlphaFold	Medical text understanding, Protein folding

---

Advantages of Pre-trained Models

Faster Development — No need to train from zero
Better Performance — Uses knowledge from large datasets
Less Data Required — Works even with small custom datasets
Lower Cost — Saves GPU/TPU compute
Easier to Start — Great for beginners and professionals alike

---

Limitations

Bias in Training Data — The pre-trained model may carry biases from the original data
Domain Mismatch — If your data is very different (e.g., satellite vs. cats/dogs), performance may drop
Large File Sizes — Pre-trained models can be huge (GBs)
Overfitting when Fine-tuned Poorly — Requires careful adjustment

---

Pre-Trained models by task:

Pre-Trained Model for Object detection

What it is: find bounding boxes + labels for multiple objects in an image.
Common pre-trained detectors: YOLO (v3/v5/v8), Faster R-CNN, SSD, RetinaNet, EfficientDet.
Datasets: COCO, Pascal VOC.
Typical pipeline:

- Start with a detector pre-trained on COCO (general objects).
- Fine-tune on your object dataset (annotation format: Pascal VOC or COCO JSON).
  Metrics: mAP (mean Average Precision) at IoU thresholds (e.g. mAP@0.5).
  Tradeoffs: YOLO family = fast, good for real-time; Faster R-CNN = more accurate but slower. Use lightweight variants (Tiny YOLO, MobileNet-SSD) for edge devices.

Pre-Trained Model for Plant disease detection

What it is: classify whether a plant leaf is healthy or has a specific disease (or segment disease area).
Common pre-trained backbones: ResNet, EfficientNet, MobileNet, DenseNet; sometimes segmentation backbones (U-Net, DeepLab) if you need pixel masks.
Popular dataset(s): PlantVillage (lots of leaf images, multiple crops/diseases).
Typical pipeline:

- Transfer learning: replace final FC layer of e.g. ResNet trained on ImageNet → fine-tune on your labeled plant images.
- Data augmentation is *critical*: rotations, flips, color jitter, random crops (disease appearance varies).
- Optionally use segmentation (U-Net) to localize lesion regions before classification.
  Metrics: accuracy, precision/recall, F1; for segmentation: IoU / Dice score.
  Challenges: domain shift (lab photos vs field photos), small datasets, varying lighting/background.
  Tips: use EfficientNet or MobileNet for mobile deployment; consider class-balancing and augmentation; use domain adaptation if field images differ.

Pre-Trained Model for Image classification (general)

What it is: assign one (or multiple) labels to an entire image.
Common pre-trained models: ResNet, VGG, EfficientNet, Inception, MobileNet, DenseNet. Available via Keras Applications, PyTorch Hub.
Typical pipeline: freeze backbone → replace classifier head → train head → optionally

unfreeze last blocks and fine-tune with a small LR.
Metrics: accuracy, top-k accuracy, precision/recall, F1.
When to use pretraining: almost always for small/medium datasets — ImageNet pretraining gives robust low-level features.

Pre-Trained Model for Sentiment analysis

What it is: classify text polarity (positive/negative/neutral) or intensity.
Common pre-trained models: BERT, RoBERTa, DistilBERT, ALBERT, XLNet, (for multilingual: mBERT, XLM-Roberta). Available on Hugging Face.
Datasets: SST, IMDB, Amazon reviews, Twitter sentiment datasets.
Typical pipeline: `from_pretrained("bert-base-uncased")` → add classifier head → fine-tune on labeled sentiment data.
Metrics: accuracy, precision/recall, F1, AUC (for imbalanced).
Notes: domain adaptation (product reviews vs tweets) can be important; smaller distilled models (DistilBERT) speed up inference.

Pre-Trained Model for Emotion detection

What it is: detect emotions (anger, joy, sadness, etc.) from text, audio, images, or video (multimodal).
Pre-trained models:

- Text: BERT/RoBERTa fine-tuned on emotion datasets (GoEmotions, EmotionLines).
- Audio: Wav2Vec2 / Whisper features + classifier.
- Vision: CNNs trained on facial emotion datasets (FER2013, AffectNet) or specialized networks.
- Multimodal: models combining text + audio + vision (CLIP + audio encoders, or multimodal transformers).
  Metrics: accuracy, F1 (per class), macro F1 if classes skewed.
  Challenges: cultural differences in expression, subtle emotions, noisy audio/video.

Pre-Trained Model for Face recognition

What it is: verify or identify a person from their face. (Face *detection* is separate.)
Pre-trained models: FaceNet, ArcFace (ResNet backbone with special loss), VGGFace, OpenFace.
Datasets: LFW (verification), MS-Celeb, VGGFace2.
Typical pipeline: use pre-trained embedding model → compute embeddings for probe and gallery → compare with cosine distance / thresholding.

Metrics: verification accuracy, TAR @ FAR, identification recall.
Privacy/ethics: face recognition has major privacy & bias issues — be cautious about use and dataset bias.

Pre-Trained Model for Image segmentation

What it is: per-pixel labeling (semantic segmentation) or instance segmentation (separate object instances).
Pre-trained models: U-Net, DeepLabv3(+), FCN, PSPNet (semantic). Mask R-CNN (instance). Pretrained backbones: ResNet, MobileNet.
Datasets: COCO (instance), Cityscapes (urban scenes), Pascal VOC.
Metrics: mIoU (mean intersection over union), pixel accuracy, Dice.
Typical pipeline: use pre-trained encoder (ImageNet) and train decoder on your mask annotations. For small datasets, freeze encoder first.

Pre-Trained Model for License plate detection (and recognition / OCR)

What it is: detect license plates → read the characters (ALPR).
Typical pipeline:

1. Text/plate detection: use object detector (YOLO, Faster R-CNN) or text detector (EAST).
2. Plate cropping + textRecognition: CRNN (CNN + RNN + CTC) or transformer OCR models, or Tesseract OCR for simple cases.
   Pre-trained components: detection models pre-trained on COCO; OCR models pre-trained on synthetic text datasets.
   Metrics: detection: mAP; recognition: character error rate (CER) / word accuracy.
   Challenges: varied fonts, motion blur, viewpoint, lighting, different country formats.

Pre-Trained Model for Hand gesture recognition

What it is: classify gestures (static) or recognise sequences (dynamic) from images or video.
Common approaches / models:

- Hand/keypoint detection: MediaPipe Hands (Google) — very popular and fast.
- Pose-based: OpenPose / BlazePose for skeleton/keypoints.
- Classifier: CNN on cropped hand images, or temporal models (LSTM / Transformers) on sequences of keypoints for dynamic gestures.
  Datasets: EgoHands, HandNet, custom datasets.
  Use cases: sign language, HCI, AR/VR control.

Transfer learning approaches (two main styles)

1. **Feature extraction:** freeze most of the pre-trained model; only replace and train the final classification head. Fast, lower risk of overfitting.
2. **Fine-tuning:** unfreeze some (or all) of the pre-trained layers and train with a low learning rate. Yields better accuracy if you have enough data and compute.

Evaluation metrics

- Classification: accuracy, top-k accuracy, precision/recall, F1.
- Detection: mAP (mean Average Precision).
- Segmentation: IoU / mIoU, Dice.
- Retrieval/recognition: recall@k, ROC / TAR @ FAR.
- OCR: CER (character error rate), word accuracy.

Popular sources for pretrained models

- **Hugging Face Model Hub** (NLP, vision, multimodal) — huge collection, `from_pretrained`.
- **TensorFlow Hub** — TF/Keras models.
- **PyTorch Hub / Torchvision models** — ResNet, Faster-RCNN, etc.
- **Ultralytics / YOLO repos** — state-of-the-art detectors.
- **OpenCV / MediaPipe** — efficient detectors/keypoint systems.
- **Model Zoos** (Detectron2, MMDetection, segmentation_models.pytorch).

Deployment & performance considerations

- **Edge vs cloud:** for mobile/edge, prefer MobileNet, EfficientNet-Lite, Tiny YOLO; for server, heavier models OK.
- **Quantization & pruning:** reduce model size/latency (INT8 quantization).
- **Latency vs accuracy trade-off:** pick model by constraints (real-time vs high-accuracy offline).
- **Ethics & bias:** pretrained models inherit dataset biases — evaluate fairness and privacy.

Other popular pretrained tasks & model families

- **Optical Character Recognition (OCR):** Tesseract, CRNN, TrOCR (transformer OCR).
- **Speech recognition / ASR:** Wav2Vec2, DeepSpeech, Whisper.
- **Machine translation:** mBART, MarianMT, T5.
- **Image captioning / VQA:** OSCAR, VinVL, BLIP (vision+language).
- **Multimodal embeddings:** CLIP (image + text), ALIGN.
- **Anomaly detection / defect detection:** pretrained encoders + One-Class methods.
- **Pose estimation:** OpenPose, MediaPipe, HRNet.
- **Image generation / inpainting / style transfer:** GANs (StyleGAN), diffusion models (Stable Diffusion).
- **Protein structure / bioinformatics:** AlphaFold (specialized).
- **Recommendation systems:** pretrained embedding models and retrieval models.
- **Document understanding:** LayoutLMv3, Donut (OCR + structure understanding).

Practical example — Fine-tuning a pre-trained ResNet (PyTorch)

A compact example to **fine-tune ResNet** for plant disease classification (replace classes & dataset as needed). Copy/paste and adapt paths.

```
# pip install torch torchvision
import torch
from torch import nn, optim
from torchvision import models, transforms, datasets
from torch.utils.data import DataLoader

# 1) Data transforms
train_tf = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])
val_tf = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])

# 2) Datasets (replace 'data/train' and 'data/val' with your folders)
train_ds = datasets.ImageFolder('data/train', transform=train_tf)
val_ds = datasets.ImageFolder('data/val', transform=val_tf)
```

```
train_loader = DataLoader(train_ds, batch_size=32, shuffle=True,
num_workers=4)
val_loader = DataLoader(val_ds, batch_size=32, shuffle=False, num_workers=4)

# 3) Load pretrained ResNet
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = models.resnet50(pretrained=True)

# 4) Replace the final layer
num_classes = len(train_ds.classes)
model.fc = nn.Linear(model.fc.in_features, num_classes)
model = model.to(device)

# 5) Loss, optimizer (feature extraction: freeze base first)
for param in model.parameters():
    param.requires_grad = True  # set False to freeze

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4)

# 6) Training loop (very simple)
def train_epoch():
    model.train()
    total, correct = 0, 0
    for imgs, labels in train_loader:
        imgs, labels = imgs.to(device), labels.to(device)
        optimizer.zero_grad()
        out = model(imgs)
        loss = criterion(out, labels)
        loss.backward()
        optimizer.step()
        preds = out.argmax(dim=1)
        correct += (preds == labels).sum().item()
        total += labels.size(0)
    print("Train acc:", correct/total)

def eval_epoch():
    model.eval()
    total, correct = 0, 0
    with torch.no_grad():
        for imgs, labels in val_loader:
            imgs, labels = imgs.to(device), labels.to(device)
            out = model(imgs)
            preds = out.argmax(dim=1)
            correct += (preds == labels).sum().item()
            total += labels.size(0)
    print("Val acc:", correct/total)

for epoch in range(1, 6):
    print("Epoch", epoch)
    train_epoch()
    eval_epoch()
```

Notes on the example: if you have a small dataset, set `param.requires_grad = False` for most of the `model` layers, train only `model.fc` first; then unfreeze top blocks and fine-tune with a lower learning rate.