1. Image Classification

Definition: Assigns one or more labels to an entire image.

```python
from torchvision import models, transforms
from PIL import Image
import torch

model = models.resnet18(pretrained=True).eval()
img = Image.open("image.jpg")
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor()
])
input_tensor = transform(img).unsqueeze(0)
output = model(input_tensor)
print("Predicted class index:", output.argmax().item())
```

2. Object Detection

Definition: Identifies and localizes objects with bounding boxes.

```python
import cv2

net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
img = cv2.imread("image.jpg")
blob = cv2.dnn.blobFromImage(img, 1/255, (416, 416), swapRB=True)
net.setInput(blob)
outputs = net.forward(net.getUnconnectedOutLayersNames())

for output in outputs:
    for detection in output:
        scores = detection[5:]
        class_id = scores.argmax()
        confidence = scores[class_id]
        if confidence > 0.5:
            print("Detected class ID:", class_id)
```

3. Semantic Segmentation

Definition: Classifies each pixel into a semantic category.

```python
import torchvision
from PIL import Image
import torch

model = torchvision.models.segmentation.deeplabv3_resnet50(pretrained=True).eval()
img = Image.open("image.jpg").convert("RGB")
transform = torchvision.transforms.Compose([
    torchvision.transforms.Resize((520, 520)),
    torchvision.transforms.ToTensor()
])
input_tensor = transform(img).unsqueeze(0)
output = model(input_tensor)["out"].argmax(1).squeeze().numpy()
print("Segmentation map shape:", output.shape)
```

4. Instance Segmentation

Definition: Segments and distinguishes individual object instances.

```python
import torchvision
from PIL import Image
import torch
from torchvision.transforms import functional as F

model = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=True).eval()
img = Image.open("image.jpg").convert("RGB")
input_tensor = F.to_tensor(img).unsqueeze(0)
output = model(input_tensor)[0]

for i, mask in enumerate(output['masks']):
    print(f"Instance {i} mask shape:", mask.shape)
```

5. Panoptic Segmentation

Definition: Combines semantic and instance segmentation.

```python
# Panoptic segmentation is available via Detectron2.
# Here's a basic example using Hugging Face Transformers:

from transformers import AutoImageProcessor, AutoModelForPanopticSegmentation
from PIL import Image
import torch

img = Image.open("image.jpg")
processor = AutoImageProcessor.from_pretrained("facebook/detectron2")
model = AutoModelForPanopticSegmentation.from_pretrained("facebook/detectron2")

inputs = processor(images=img, return_tensors="pt")
outputs = model(**inputs)
print("Panoptic segmentation output:", outputs.keys())
```

6. Keypoint Detection

Definition: Detects anatomical or structural points (e.g., joints).

```python
import mediapipe as mp
import cv2

pose = mp.solutions.pose.Pose()
img = cv2.imread("person.jpg")
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
results = pose.process(img_rgb)

if results.pose_landmarks:
    for lm in results.pose_landmarks.landmark:
        print("Keypoint:", lm.x, lm.y)
```

7. Pose Estimation

Definition: Predicts human or object poses via keypoints.

```python
import cv2
import mediapipe as mp

# Initialize MediaPipe pose class and drawing utilities
mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils

# Load video or webcam
cap = cv2.VideoCapture(0)  # Use 0 for webcam

with mp_pose.Pose(static_image_mode=False,
                model_complexity=1,
                enable_segmentation=False,
                min_detection_confidence=0.5,
                min_tracking_confidence=0.5) as pose:

    while cap.isOpened():
        success, frame = cap.read()
        if not success:
            break

        # Convert the image to RGB
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False

        # Process the image and detect pose
        results = pose.process(image)

        # Draw pose landmarks on the image
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        if results.pose_landmarks:
            mp_drawing.draw_landmarks(
                image, results.pose_landmarks, mp_pose.POSE_CONNECTIONS)

        # Display the result
        cv2.imshow('Pose Estimation', image)

        if cv2.waitKey(5) & 0xFF == 27:
            break

cap.release()
cv2.destroyAllWindows()
```

8. Facial Recognition

Definition: Identifies or verifies individual faces.

```
    import face_recognition

    img = face_recognition.load_image_file("face.jpg")
    encodings = face_recognition.face_encodings(img)
    print("Face encoding vector:", encodings[0])
```

9. Image Generation (Synthesis)

Definition: Generates realistic or stylized images from input.

```
    from diffusers import StableDiffusionPipeline

    pipe = StableDiffusionPipeline.from_pretrained("runwayml/stable-diffusion-v1-
5").to("cuda")
    image = pipe("a futuristic city at sunset").images[0]
    image.save("generated.png")
```

10. Image-to-Image Translation

Definition: Converts one image domain to another.

```
    # Example using Pix2Pix via Hugging Face Diffusers

    from diffusers import Pix2PixZeroPipeline
    pipe = Pix2PixZeroPipeline.from_pretrained("timbrooks/pix2pix-zero-portrait").to("cuda")
    image = pipe(prompt="make it look like a cartoon", image="portrait.jpg").images[0]
    image.save("translated.png")
```

11. Super Resolution

Definition: Enhances image resolution.

```
from realesrgan import RealESRGAN
from PIL import Image

model = RealESRGAN("cuda", scale=2)
model.load_weights("RealESRGAN_x2.pth")
img = Image.open("low_res.jpg")
sr_img = model.predict(img)
sr_img.save("high_res.jpg")
```

12. Image Denoising

Definition: Removes noise from images.

```
import cv2
img = cv2.imread("noisy.jpg")
denoised = cv2.fastNlMeansDenoisingColored(img, None, 10, 10, 7, 21)
cv2.imwrite("denoised.jpg", denoised)
```

13. Image Inpainting

Definition: Fills in missing regions.

```
import cv2
img = cv2.imread("damaged.jpg")
mask = cv2.imread("mask.jpg", 0)  # white where missing
inpainted = cv2.inpaint(img, mask, 3, cv2.INPAINT_TELEA)
cv2.imwrite("inpainted.jpg", inpainted)
```

14. Image Captioning

Definition: Generates natural language descriptions of images.

```
    from transformers import BlipProcessor, BlipForConditionalGeneration
    from PIL import Image

    processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
    model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-
base")

    img = Image.open("image.jpg")
    inputs = processor(images=img, return_tensors="pt")
    caption = model.generate(**inputs)
    print("Caption:", processor.decode(caption[0], skip_special_tokens=True))
```

15. Visual Question Answering (VQA)

Definition: Answers questions based on image content.

```
    from transformers import ViltProcessor, ViltForQuestionAnswering
    from PIL import Image

    processor = ViltProcessor.from_pretrained
```

16. Depth Estimation

Definition: Infers depth from single or stereo images.

```
    from transformers import DPTFeatureExtractor, DPTForDepthEstimation
    from PIL import Image
    import torch

    img = Image.open("scene.jpg")
    extractor = DPTFeatureExtractor.from_pretrained("Intel/dpt-large")
    model = DPTForDepthEstimation.from_pretrained("Intel/dpt-large")

    inputs = extractor(images=img, return_tensors="pt")
    with torch.no_grad():
        depth = model(**inputs).predicted_depth
    print("Depth map shape:", depth.shape)
```

17. 3D Reconstruction

Definition: Rebuilds 3D models from 2D images or video.

📦 **Note:** This task typically uses external tools like COLMAP or [OpenMVG]. Here's how to run COLMAP from Python using **`subprocess`**

```python
import subprocess

# Paths to your image folder and output workspace
image_path = "./images"
workspace_path = "./output"

# Run COLMAP automatic reconstruction
subprocess.run([
    "colmap", "automatic_reconstructor",
    "--image_path", image_path,
    "--workspace_path", workspace_path,
    "--data_type", "image",
    "--use_gpu", "1"
])
# This will generate a sparse and dense 3D model from an image set.
```

18. Optical Flow Estimation

Definition: Tracks motion between video frames.

```python
import cv2

prev = cv2.imread("frame1.jpg", cv2.IMREAD_GRAYSCALE)
next = cv2.imread("frame2.jpg", cv2.IMREAD_GRAYSCALE)
flow = cv2.calcOpticalFlowFarneback(prev, next, None, 0.5, 3, 15, 3, 5, 1.2, 0)
print("Optical flow shape:", flow.shape)
```

19. Video Classification

Definition: Assigns labels to video clips.

```
    # Example using Hugging Face TimeSformer:
    from transformers import TimesformerProcessor, TimesformerForVideoClassification
    import torch
    from decord import VideoReader, cpu

    video = VideoReader("video.mp4", ctx=cpu())
    frames = [video[i].asnumpy() for i in range(0, len(video), 4)]

    processor = TimesformerProcessor.from_pretrained("facebook/timesformer-base-finetuned-
k400")
    model = TimesformerForVideoClassification.from_pretrained("facebook/timesformer-base-
finetuned-k400")

    inputs = processor(videos=frames, return_tensors="pt")
    outputs = model(**inputs)
    label = model.config.label2id[outputs.logits.argmax().item()]
    print("Predicted label:", label)
```

20. Action Recognition

Definition: Identifies human actions from video.

📦 **Using PyTorchVideo's pretrained SlowFast model:**

```
    from pytorchvideo.models.hub import slowfast_r50
    import torch
    import torchvision.transforms as transforms
    import numpy as np
    import cv2

    # Load pretrained model
    model = slowfast_r50(pretrained=True).eval()

    # Load video and extract frames
    cap = cv2.VideoCapture("action.mp4")
    frames = []
    while len(frames) < 32:
        ret, frame = cap.read()
        if not ret: break
        frame = cv2.resize(frame, (256, 256))
        frames.append(frame)
    cap.release()

    # Preprocess frames
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize([0.45, 0.45, 0.45], [0.225, 0.225, 0.225])
    ])
    video_tensor = torch.stack([transform(f) for f in frames]).permute(1, 0, 2,
3).unsqueeze(0)

    # Run model
    with torch.no_grad():
        output = model(video_tensor)
        predicted_class = output.argmax().item()
    print("Predicted action class index:", predicted_class)
```

21. Object Tracking

Definition: Follows object location over time in video.

```
import cv2

tracker = cv2.TrackerCSRT_create()
video = cv2.VideoCapture("video.mp4")
ret, frame = video.read()
bbox = cv2.selectROI("Tracking", frame, False)
tracker.init(frame, bbox)

while True:
    ret, frame = video.read()
    if not ret: break
    success, box = tracker.update(frame)
    if success:
        x, y, w, h = [int(v) for v in box]
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0,255,0), 2)
    cv2.imshow("Tracking", frame)
    if cv2.waitKey(1) == 27: break
video.release()
cv2.destroyAllWindows()
```

22. Multimodal Learning

Definition: Combines vision with language or audio.

```
from transformers import CLIPProcessor, CLIPModel
from PIL import Image
import torch

model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

img = Image.open("image.jpg")
text = ["a dog", "a cat", "a car"]

inputs = processor(text=text, images=img, return_tensors="pt", padding=True)
outputs = model(**inputs)
probs = outputs.logits_per_image.softmax(dim=1)
print("Text match probabilities:", probs)
```

23. Scene Text Recognition (OCR)

Definition: Detects and reads text from images.

```
import pytesseract
import cv2

img = cv2.imread("text_image.jpg")
text = pytesseract.image_to_string(img)
print("Extracted text:", text)
```

24. Image Retrieval

Definition: Finds visually similar images.

```
from PIL import Image
from transformers import CLIPProcessor, CLIPModel
import torch

model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

def load_image(path): return Image.open(path).convert("RGB")
image_paths = ["query.jpg", "img1.jpg", "img2.jpg"]
images = [load_image(p) for p in image_paths]

inputs = processor(images=images, return_tensors="pt", padding=True)
with torch.no_grad():
    embeddings = model.get_image_features(**inputs)
embeddings = embeddings / embeddings.norm(dim=-1, keepdim=True)

query = embeddings[0]
similarities = torch.matmul(embeddings[1:], query.unsqueeze(0).T).squeeze()
for i in similarities.argsort(descending=True):
    print(f"Match: {image_paths[i+1]}, Score: {similarities[i].item():.4f}")
```

25. Domain Adaptation

Definition: Applies vision models to new but similar domains.

📦 **Using a simplified Domain-Adversarial Neural Network (DANN) setup:**

```
    import torch
    import torch.nn as nn
    from torch.autograd import Function

    # Gradient reversal layer
    class GradReverse(Function):
        @staticmethod
        def forward(ctx, x, lambd): ctx.lambd = lambd; return x.view_as(x)
        @staticmethod
        def backward(ctx, grad_output): return -ctx.lambd * grad_output, None

    # Feature extractor
    class FeatureExtractor(nn.Module):
        def __init__(self): super().__init__()
            self.conv = nn.Sequential(nn.Conv2d(3, 64, 3), nn.ReLU(), nn.Flatten())

        def forward(self, x): return self.conv(x)

    # Label predictor
    class LabelPredictor(nn.Module):
        def __init__(self): super().__init__()
            self.fc = nn.Linear(64 * 30 * 30, 10)

        def forward(self, x): return self.fc(x)

    # Domain classifier
    class DomainClassifier(nn.Module):
        def __init__(self): super().__init__()
            self.fc = nn.Linear(64 * 30 * 30, 2)

        def forward(self, x, lambd): x = GradReverse.apply(x, lambd); return self.fc(x)
    # You'd train this with source and target domain batches, optimizing both label and
domain losses.
```

26. Zero-shot Learning (ZSL)

Definition: Recognizes novel classes not seen during training.

```
from transformers import CLIPProcessor, CLIPModel
from PIL import Image

model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

img = Image.open("image.jpg")
labels = ["a zebra", "a giraffe", "a lion"]
inputs = processor(text=labels, images=img, return_tensors="pt", padding=True)
outputs = model(**inputs)
pred = labels[outputs.logits_per_image.argmax()]
print("Zero-shot prediction:", pred)
```

27. Few-shot Learning

Definition: Learns from very few labeled examples.

📦 **Using Prototypical Networks (simplified)**:

```
import torch
import torch.nn.functional as F

# Support set: 5 examples per class
support = torch.randn(10, 64)  # 2 classes × 5 samples
support_labels = torch.tensor([0]*5 + [1]*5)

# Query set: unlabeled examples
query = torch.randn(3, 64)

# Compute prototypes
prototypes = torch.stack([
    support[support_labels == c].mean(0) for c in torch.unique(support_labels)
])

# Compute distances and classify
dists = torch.cdist(query, prototypes)
preds = dists.argmin(dim=1)
print("Predicted classes:", preds.tolist())
```

28. Image Style Transfer

Definition: Applies artistic style from one image to another.

```python
import torch
import torchvision.transforms as transforms
from PIL import Image
from torchvision.models import vgg19

# Load content and style images
content = Image.open("content.jpg")
style = Image.open("style.jpg")
transform = transforms.Compose([transforms.Resize((256, 256)), transforms.ToTensor()])
content_tensor = transform(content).unsqueeze(0)
style_tensor = transform(style).unsqueeze(0)

# Use VGG features and optimize content image to match style
# Requires full neural style transfer pipeline (omitted for brevity)
```

29. Image Matting

Definition: Extracts foreground object with fine boundaries.

📦 **Using MODNet via ONNX Runtime**:

```python
import onnxruntime
import cv2
import numpy as np

# Load MODNet ONNX model
session = onnxruntime.InferenceSession("modnet.onnx")

# Preprocess image
img = cv2.imread("portrait.jpg")
img = cv2.resize(img, (512, 512))
input_blob = img.transpose(2, 0, 1).astype(np.float32) / 255.0
input_blob = np.expand_dims(input_blob, axis=0)

# Run inference
outputs = session.run(None, {"input": input_blob})
alpha = outputs[0][0][0]  # Alpha matte

# Apply matte
foreground = (img * alpha[..., None]).astype(np.uint8)
cv2.imwrite("matte.png", foreground)
```

30. Image Quality Assessment

Definition: Evaluates perceptual quality of images.

📦 **Using BRISQUE via `piq` (PyTorch Image Quality)**:

```python
import torch
import piq
from PIL import Image
from torchvision.transforms import ToTensor

img = Image.open("test.jpg").convert("RGB")
img_tensor = ToTensor()(img).unsqueeze(0)

score = piq.brisque(img_tensor)
print("BRISQUE score:", score.item())
```