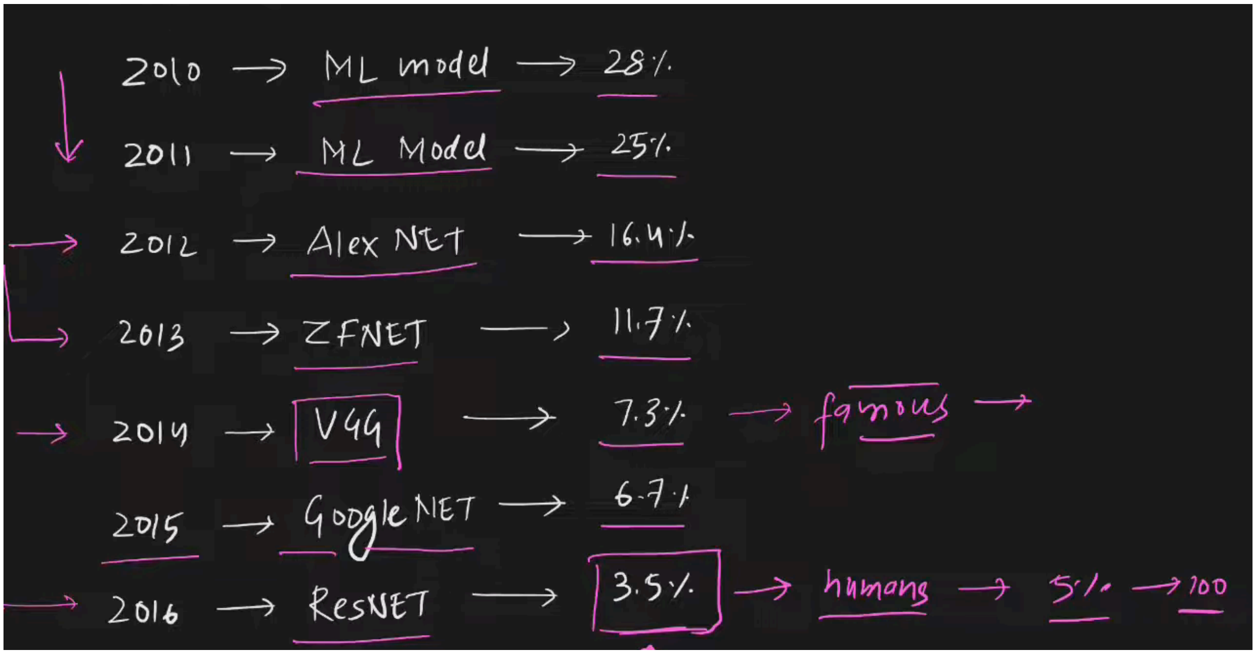
Pretrained models in CNN

- Models someone else has made on other dataset.

Pretrained CNN models are networks that were already trained on large datasets (usually on massive datasets like ***ImageNet***) and can be reused for new tasks.

- They **save time**, **require less data**, and give **very good accuracy** even for beginners.
- Popular ones: **VGG16, ResNet, Inception, MobileNet, EfficientNet**.
- You can either **use them as-is** (for predictions) or **fine-tune** them (retrain partially for your task).



2010 → ML model → 28%

2011 → ML Model → 25%

2012 → Alex NET → 16.4%

2013 → ZFNET → 11.7%

2014 → VGG → 7.3% → famous →

2015 → Google NET → 6.7%

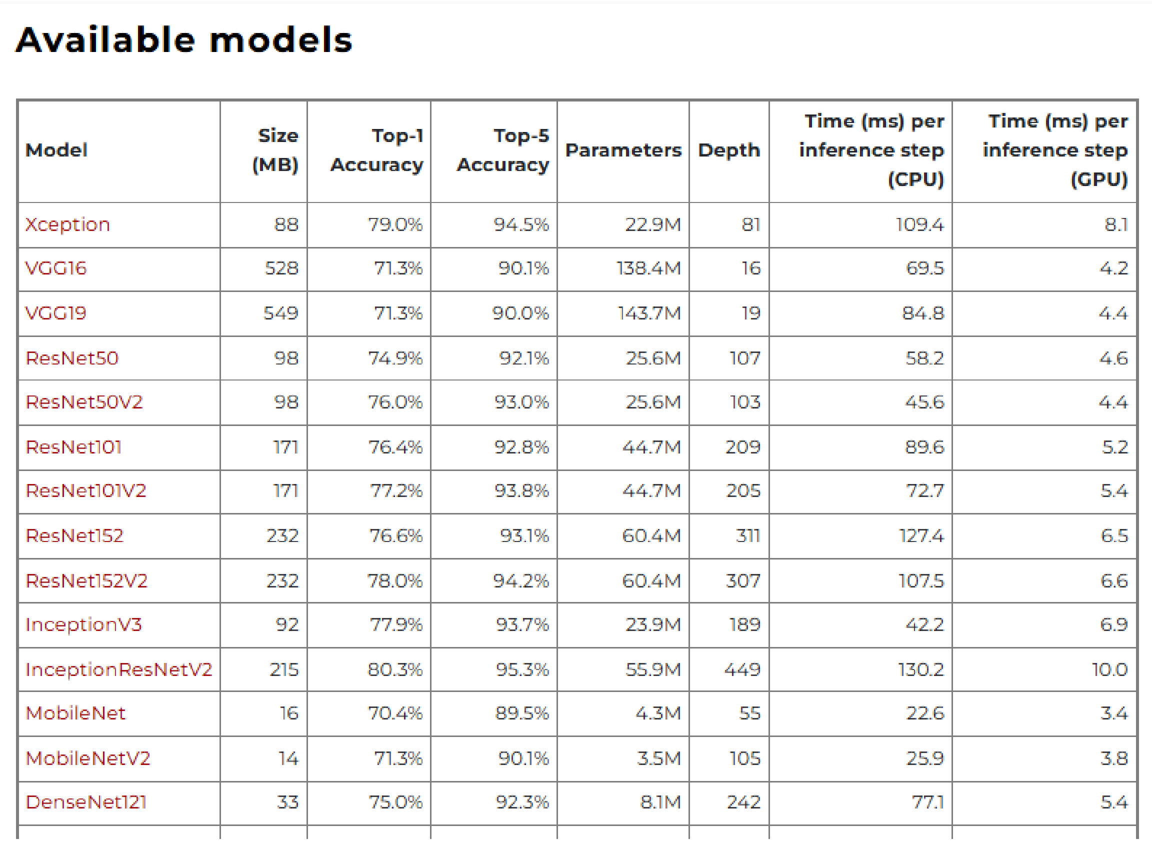
2016 → ResNET → 3.5% → humans → 5% → 100

Why Use Pretrained Models?

1. **Transfer Learning**: Leverage features learned from large datasets.
2. **Faster Training**: No need to train from scratch.
3. **Better Performance**: Especially useful for small datasets.

Keras Pretrained Models:

https://keras.io/api/applications/



Available models

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4

Popular Pretrained Models (Keras/TensorFlow)

Model	Size	Top-1 Accuracy	Use Case
VGG16	528MB	71.3%	Good for feature extraction
ResNet50	98MB	74.9%	General-purpose tasks
EfficientNet	~20MB	77-84%	Best accuracy/size trade-off
MobileNet	~16MB	70-75%	Mobile/edge devices

📐 Typical Image Sizes per Model

Model	Required Image Size
VGG16	224×224
ResNet50	224×224
InceptionV3	299×299
MobileNet	160×160, 192×192
EfficientNet	Varies by version

💡 Tip

If your dataset is:

- **Small** → Use frozen pretrained model + train only dense layers
- **Medium/Large** → Fine-tune the top few convolutional layers as well

How to Use Pretrained Models?

Step 1: Import Required Libraries

We will use TensorFlow and Keras for loading and using pretrained models.

```
# Import Required Libraries
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np
```

Step 2: Load a Pretrained Model

We will load the VGG16 model, which is pretrained on the ImageNet dataset.

```
# Load the VGG16 Model
base_model = VGG16(include_top=False, input_shape=(224, 224, 3), weights='imagenet')
```

`include_top`=False Removes the final classification layer (used in ImageNet)

Result:

You get only the **convolutional layers** (the “feature extractor” part).

Default settings:

```
keras.applications.VGG16(
    include_top=True,
    weights="imagenet",
    input_tensor=None,
    input_shape=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
    name="vgg16",
)
```

🔍Step 3: Freezing the Layers

```
base_model.trainable = False
```

- Prevents any updates to the weights during training.
- You're saying: "Keep the pre-learned filters as they are. Just reuse them."

We need to preprocess the input image to match the input requirements of the model.

```
# Load and Preprocess the Image
image_path = r'c:\Users\Jeevan\Downloads\Set33_0e3cebcf-51d6-451a-a76f-1bd69813cab3.webp'  # Replace with the path to your image
image = load_img(image_path, target_size=(224, 224))
image_array = img_to_array(image)
image_array = np.expand_dims(image_array, axis=0)
image_array = preprocess_input(image_array)
```

Step 4: 📊 Extracting Features (General Use)

Now you can pass your own image into this base_model and get its **feature map**.

```
# Load and Preprocess the Image
image_path = r'c:\Users\Jeevan\Downloads\download.jpg'  # Replace with the path to your image
img= load_img(image_path, target_size=(224, 224))
img
```

- `target_size=(224, 224)` resizes your image to the shape VGG16 expects.
- `img` is now a PIL Image object (like a pillow image in Python).

Step 5: Convert to NumPy Array:

```
x = img_to_array(img)
```

- This converts the image into a NumPy array of shape **(224, 224, 3)**.
- Each pixel has 3 values: **R, G, B**

Step 6: Add a Batch Dimension:

```
x = np.expand_dims(x, axis=0)
```

- Models always expect **batch input** — even for 1 image.
- This changes the shape from **(224, 224, 3)** to **(1, 224, 224, 3)**

Think of it like: "***I'm giving you 1 image in a batch***."

Step 7: Preprocess the Image

```
x = preprocess_input(x)
```

- This normalizes the pixel values **in the way VGG16 was trained**.
- It:
    - Converts from RGB to BGR
    - Subtracts the mean pixel value for each channel
    - Does not scale to 0–1 (unlike other models like MobileNet)

Without this step, predictions will be completely wrong.

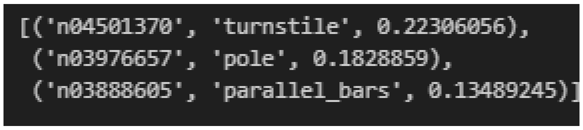
Step 8: Get the Predictions

```
preds = model.predict(x)
```

- This returns a NumPy array of shape **(1, 1000)**.
- Each number is a probability for one of 1000 classes (from ImageNet).

Step 9: Decode Predictions (Get Human-Readable Labels)

```
decode_predictions(preds, top=3)[0]
```



```
[('n04501370', 'turnstile', 0.22306056),
 ('n03976657', 'pole', 0.1828859),
 ('n03888605', 'parallel_bars', 0.13489245)]
```

👆 Wrong Prediction in this case

🔍 SHAPE EXPLANATION

Step	Shape	Meaning
After `img_to_array()`	(224, 224, 3)	Single image
After `expand_dims()`	(1, 224, 224, 3)	1 image in a batch
After `predict()`	(1, 1000)	Probabilities for 1000 classes

Code In short:

```
model = ResNet50(weights='imagenet')

# Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels.h5

img_path = '/content/tomato.jpg'

img = image.load_img(img_path, target_size=(224, 224))

x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model.predict(x)
print('Predicted:', decode_predictions(preds, top=3)[0])
```

If You Want to Keep `include_top=False`

You need to add classification layers:

```
base_model = ResNet50(include_top=False, weights='imagenet')
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(1000, activation='softmax')  # 1000 for ImageNet
])
# Then use model.predict() as before
```