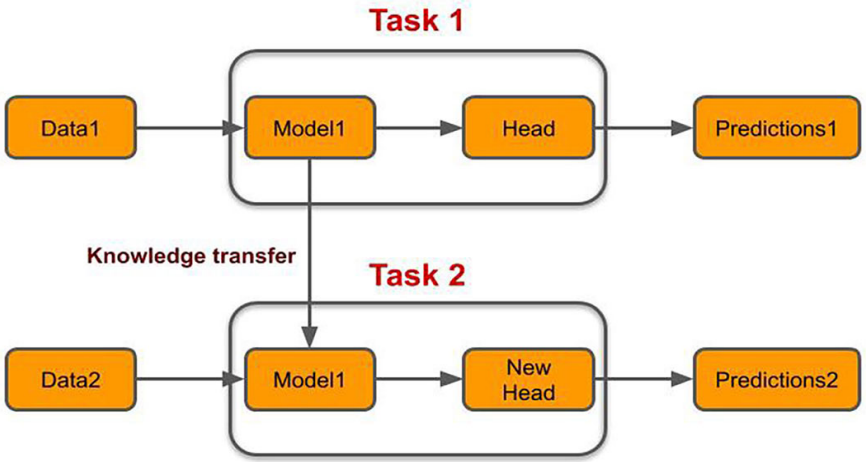
Deep Transfer Learning

UMSL 2022 DL Workshop

Building a network

- Suppose we built to identify image as cat or dog with very high accuracy
- We are now asked to create another network to identify pictures of bears
  - Could build new network from scratch
    - Increased training time
    - May not have enough data
- In practice, most people use pre-existing network architectures and then adjust them for new task

Deep Transfer Learning



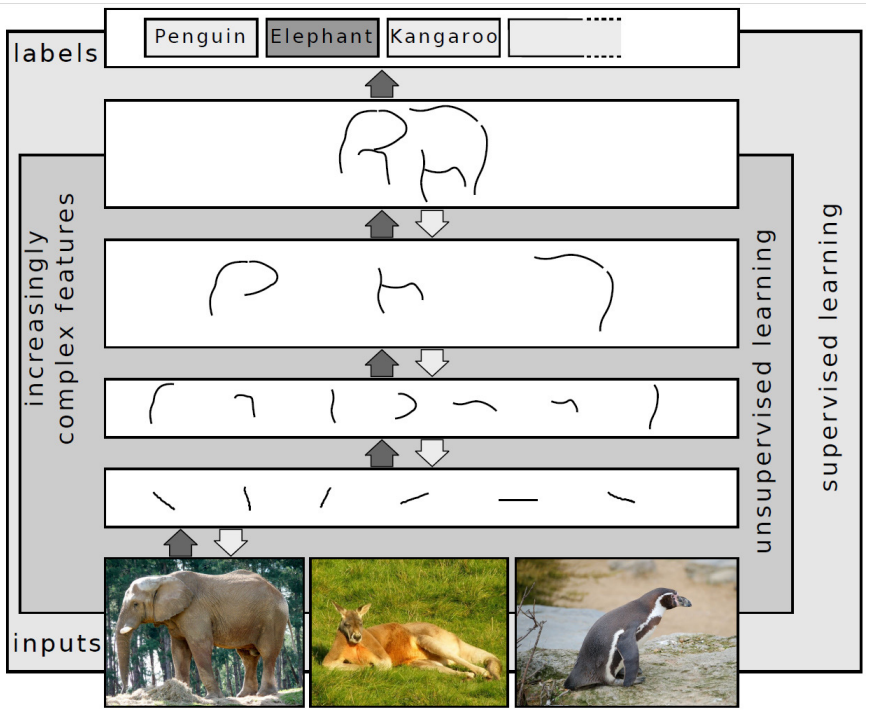
- Repurposing of trained deep learning networks for other tasks
  - Useful when data for desired task is limited or the two tasks (old and new) are related in some way

- Assumption is that networks learn certain core features that transfer across different domains

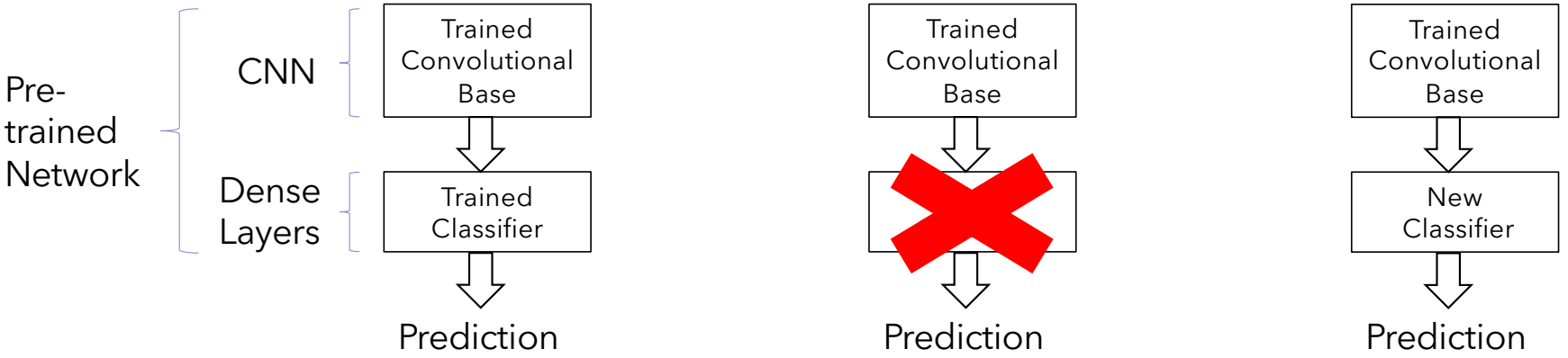
Feature Learning

- Neural networks will typically learn edges in the first layers, forms in middle layers, and task-specific features in the latter layers
  - Modifying the latter layers is what we are interested in

- Two ways we can use pre-trained networks
  - Feature Extraction
    - Best for when data from target task is very small
  - Fine-tuning
    - Best for when data from target task is very large and shares some domain with pre-trained network

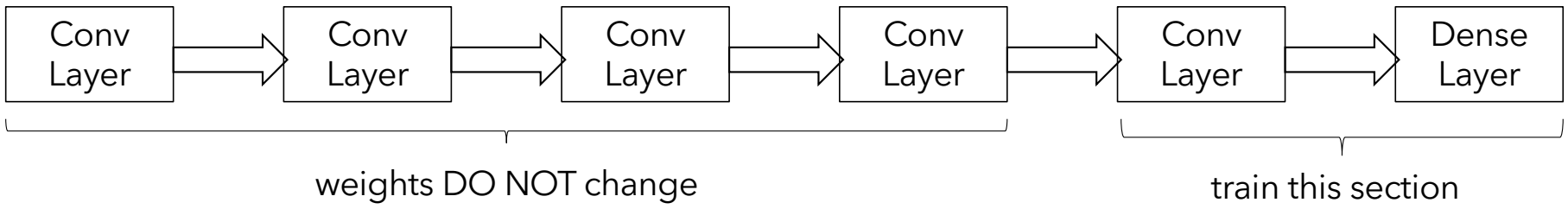


Feature Extraction



- Remove the dense layers from a pre-trained network, leaving only a "convolutional base"
- Run data through "convolutional base" (the weights DO NOT change)
  - Representations learned by base are more generic and therefore transferrable
- Train a newly manufactured dense layer with the output from the convolutions

Fine-Tuning



- Remove dense layers from pre-trained network (feature extraction)
- Make some layers of CNN trainable
  - Slightly adjust the abstract representations of the pre-trained network to make them relevant for problem at hand
- Train both the adjustable layers of CNN and the dense layers

- Advantages
  - Faster training time
  - Existing CNNs are trained on massive amounts of data
  - Higher accuracy after training

- Available networks (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
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0