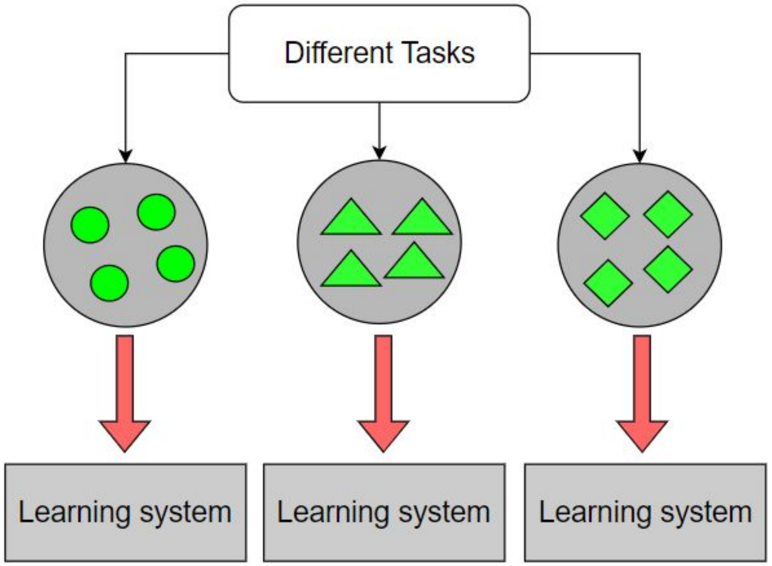
AI VIETNAM
All-in-One Course

Tricks to Improve Performance

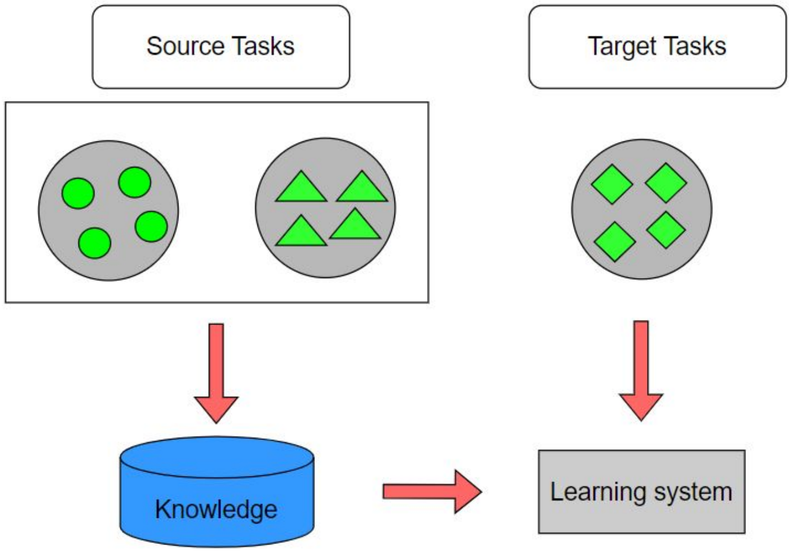
TA Hùng An

1 - Transfer Learning

Traditional vs Transfer Learning



Traditional Machine Learning



Transfer Learning

1 - Transfer Learning

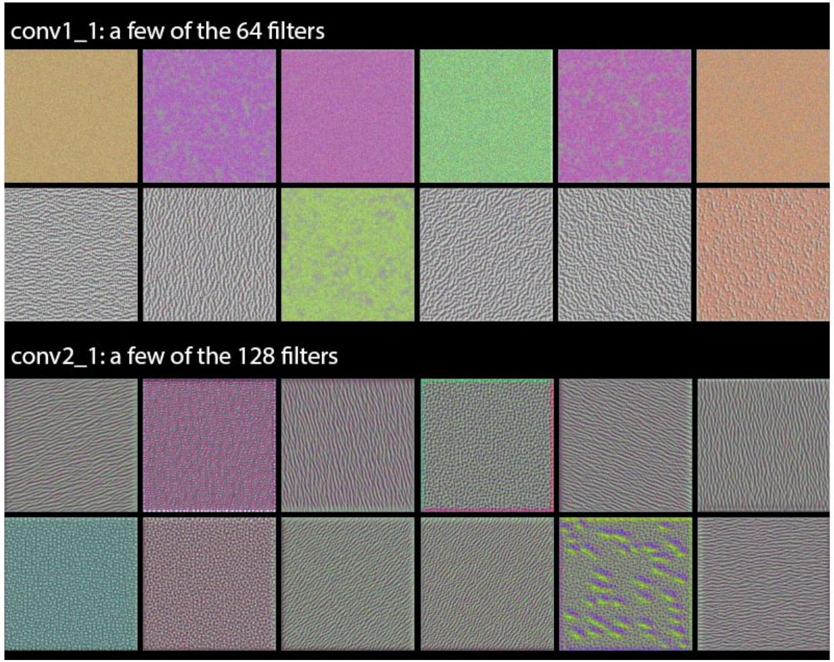
Transfer Learning Types

Types	Description	Examples
Inductive	Adapt existing **supervised** training model on new **labeled** dataset	Classification, Regression
Transductive	Adapt existing **supervised** training model on new **unlabeled** dataset	Classification, Regression
Unsupervised	Adapt existing **unsupervised** training model on new **unlabeled** dataset	Clustering

1 - Transfer Learning

Neural Network Layers: General to specific

1. Bottom/first/earlier layers: general learners
   - Low-level: edges, visual shapes
2. Top/last/later layers: specific learners
   - High-level features: eyes, feathers

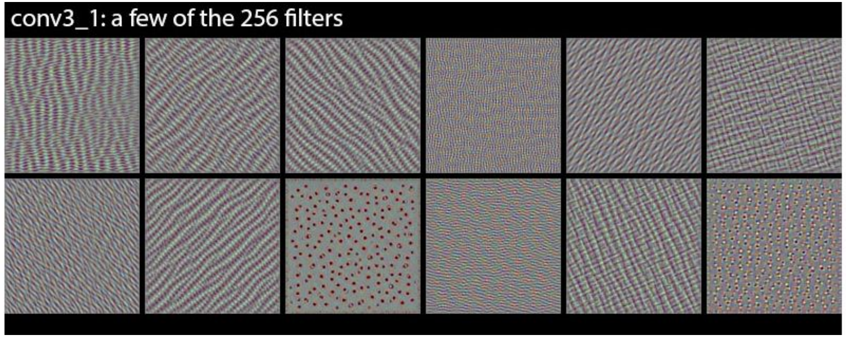


Earlier layers

1 - Transfer Learning

Neural Network Layers: General to specific

1. Bottom/first/earlier layers: general learners
   - Low-level: edges, visual shapes
2. Top/last/later layers: specific learners
   - High-level features: eyes, feathers



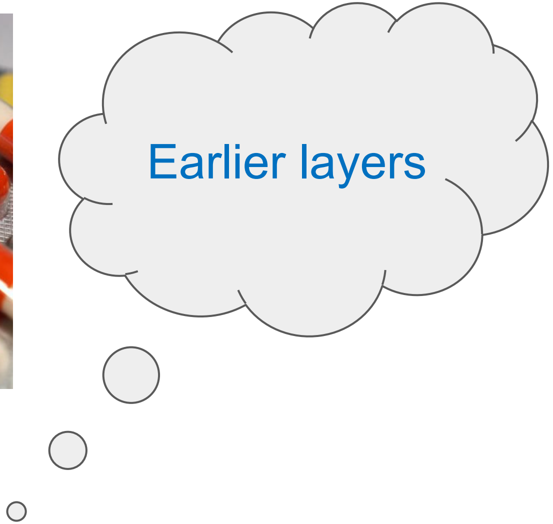
Mid layers

1 - Transfer Learning

Neural Network Layers: General to specific

ImageNet

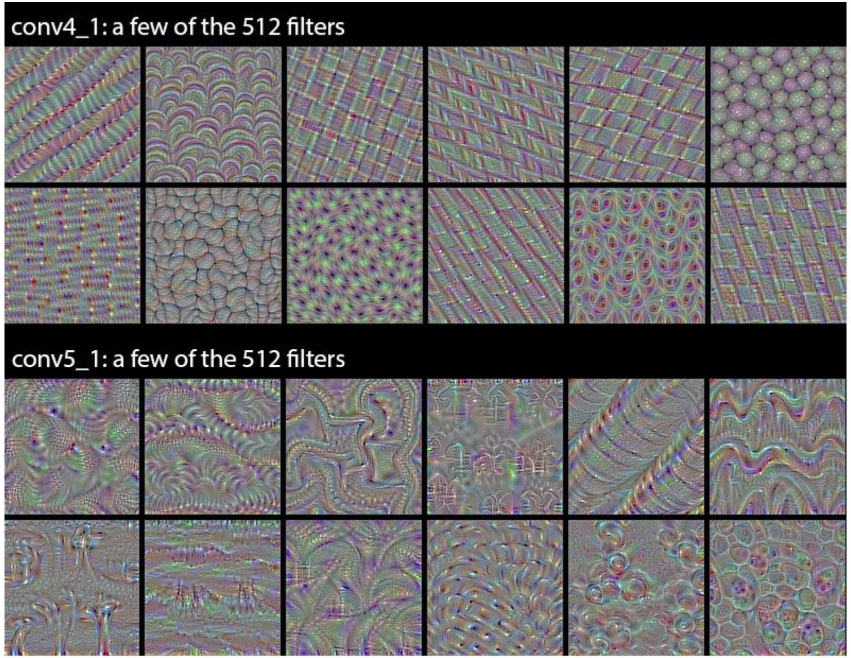
Pill data



1 - Transfer Learning

Neural Network Layers: General to specific

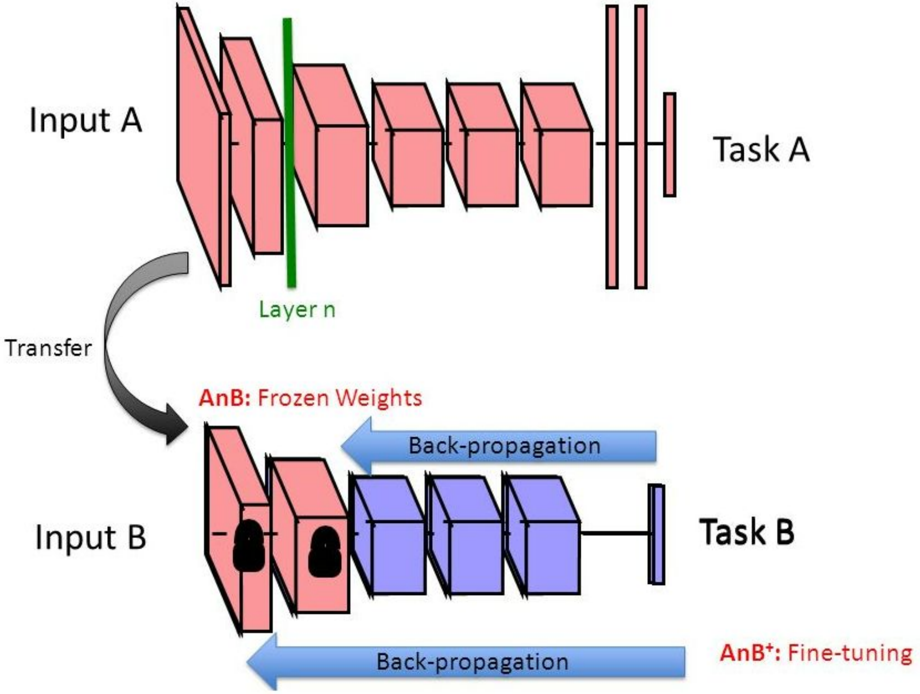
1. Bottom/first/earlier layers: general learners
   - Low-level: edges, visual shapes
2. Top/last/later layers: specific learners
   - High-level features: eyes, feathers



Later layers

1 - Transfer Learning

Transfer Learning: Overview

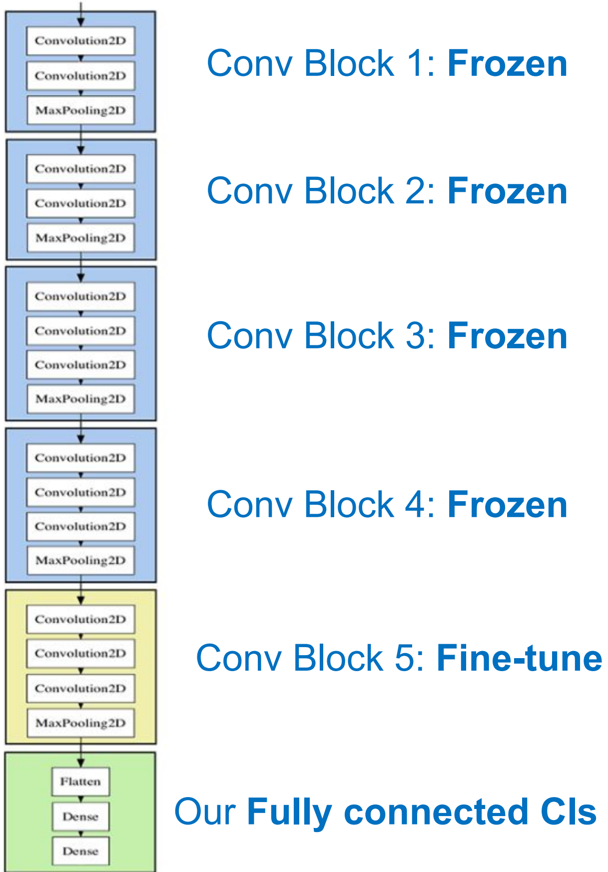


1 - Transfer Learning

Transfer Learning: Process

1. Start with pre-trained network
2. Partition network into
   - Featurizers: Identify which layer to keep
   - Classifiers: Identify which layer to replace
3. Re-train classifier layers with new data
4. Unfreeze weights and fine-tun whole network with smaller learning rate

Which layers to re-train?
- Depends on the domain
- Start by re-training the last layers
- Work backwards if performance is not satisfactory

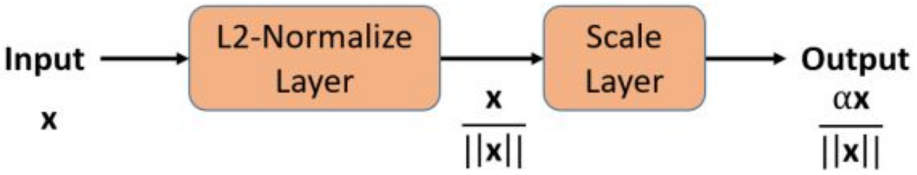


1 - Transfer Learning

When and how to fine-tune?

Dataset size	Dataset similarity	Recommendation
Large	Very different	Train model B from scratch Initalize weights from model A
Large	Similar	OK to fine-tune (less likely to overfit)
Small	Very different	Train classifier using the earlier layers
Small	Similar	Don't fine-tune (overfitting). Train a linear classifier

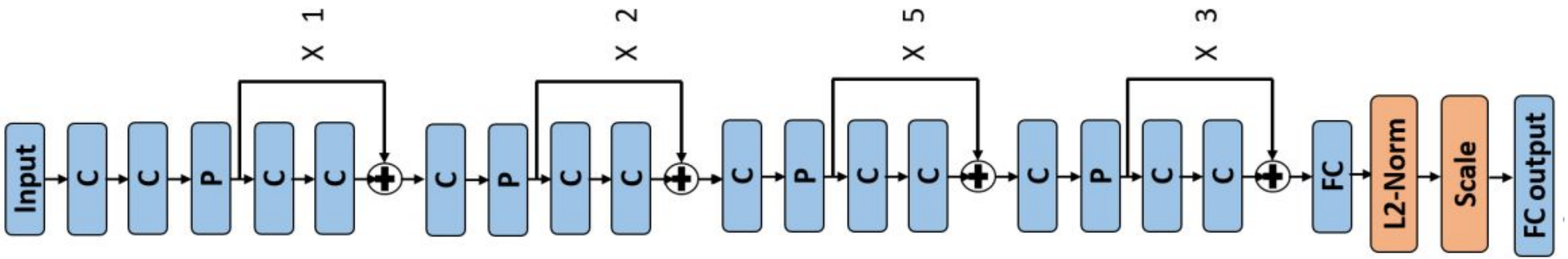
2 - Normalize



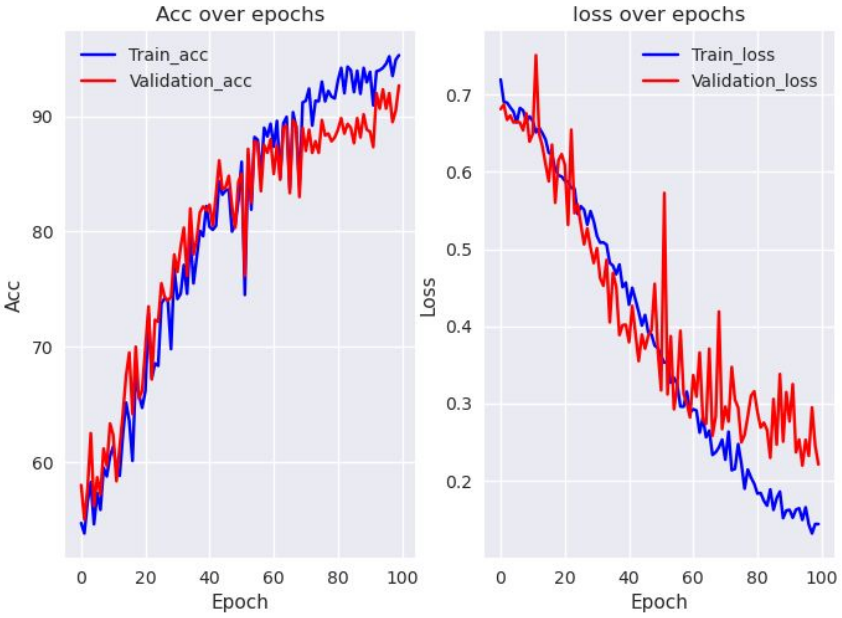
$$\alpha_{low} = \log \frac{p(C-2)}{1-p}$$

Thêm normalize layer để chuẩn hóa feature

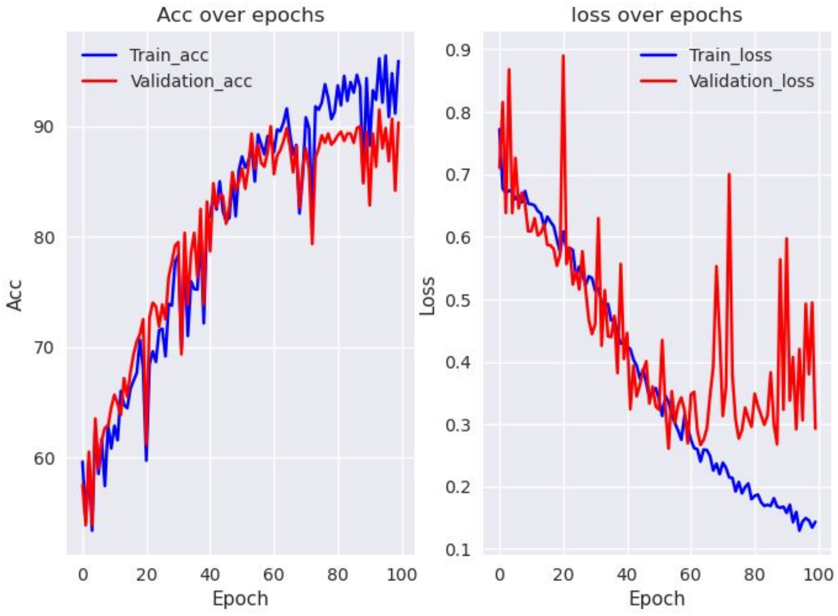
p: xác suất của lần train trước đó
C: số lượng Class



2 - Normalize



With Normalize



Without Normalize

3 - Prevents CUDA Error: Out of Memory

- Koila: a light-weight wrapper over native PyTorch.
- **Automatically computes the amount of remaining GPU memory and uses the right batch size**, saving everyone from having to manually fine-tune the batch size whenever a model is used.

4 - Iterate over rows in dataframe

```
def iterrows(df):
    list_id = []
    for index, row in df.iterrows():
        list_id.append(row["id"])

    return list_id
```

```
time: 1.04 ms (started: 2023-07-19 07:52:15 +00:00)
```

```
list_id = iterrows(df)
```

```
time: 17.1 s (started: 2023-07-19 07:53:47 +00:00)
```

```
def np_vectorization(df):
    np_arr = df.to_numpy()

    return np_arr[:,0]
```

```
time: 738 µs (started: 2023-07-19 07:54:53 +00:00)
```

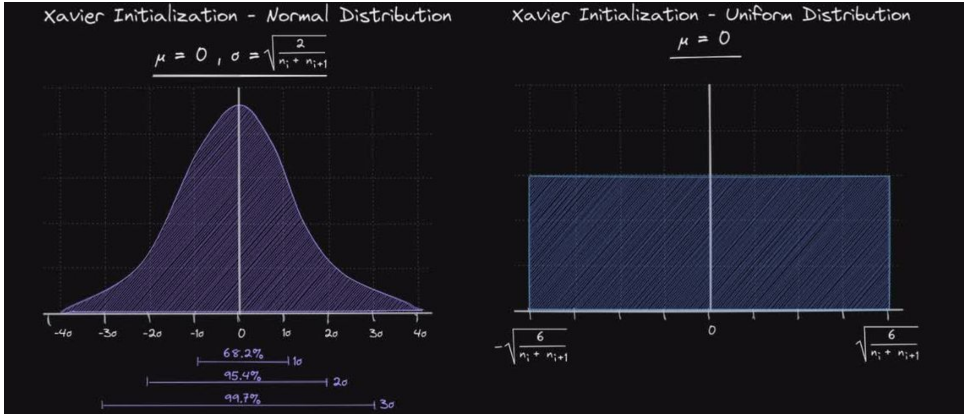
```
list_id = np_vectorization(df)
```

```
time: 939 µs (started: 2023-07-19 07:54:58 +00:00)
```

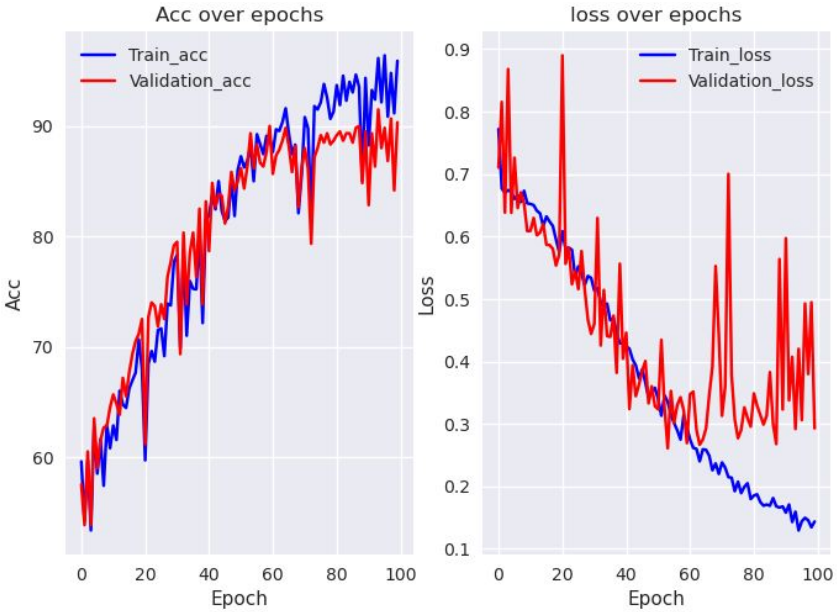
Pandas vectorization far outperforms Pandas iterrows for computing stuff with dataframes.

5 - Xavier Init

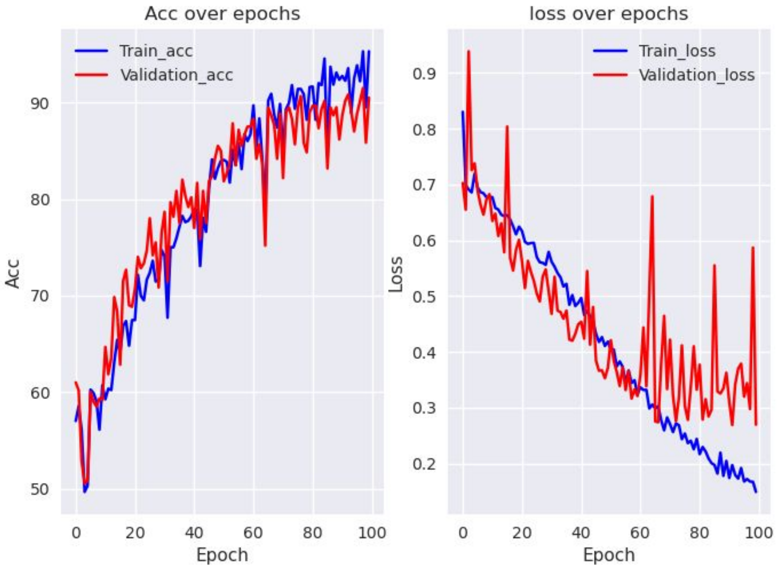
- **Weight initialization** is an **important** consideration in the **design** of a neural network model.
- The **nodes** in neural networks are **composed** of **parameters referred** to as **weights** used to **calculate** a **weighted** sum of the inputs.
- The **xavier initialization** method is **calculated** as a **random number** with a **uniform probability distribution** (U) between the range **[-(1/sqrt(n)), 1/sqrt(n)]**, where n is the number of inputs to the node.



5 - Xavier Init



Baseline

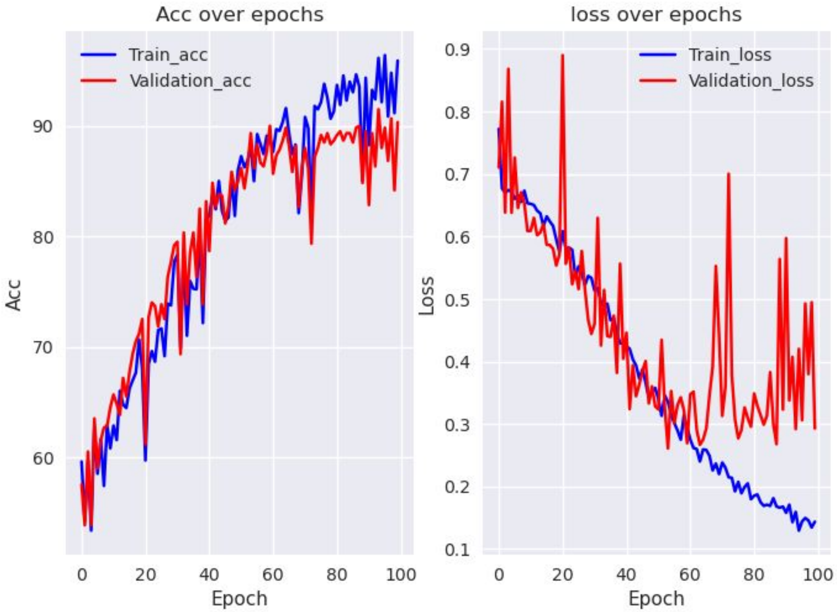


Baseline + Xavier

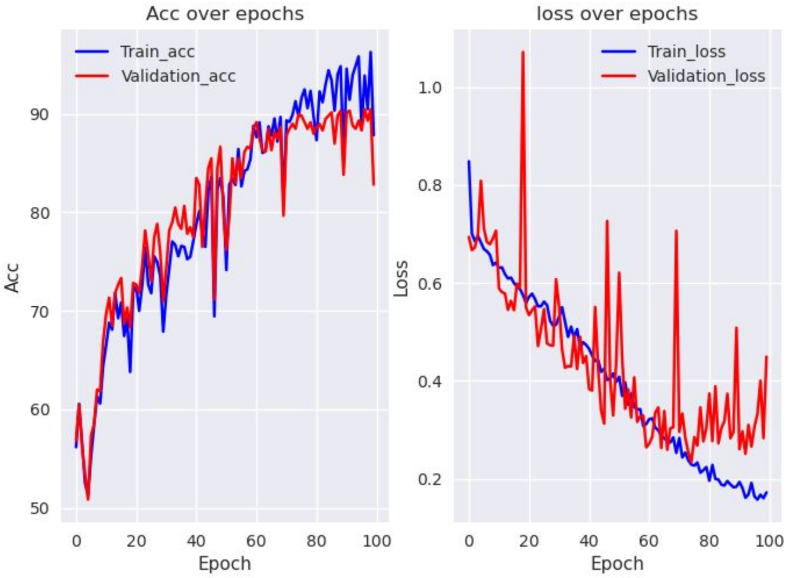
6 - No bias decay

```python
def split_weights(net):
    """split network weights into to categlories, one are weights in conv layer and linear layer,
    others are other learnable paramters(conv bias, bn weights, bn bias, linear bias)

    Args:
        net: network architecture

    Returns:
        a dictionary of params splite into to categlories
    """

    decay = []
    no_decay = []

    for m in net.modules():
        if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
            decay.append(m.weight)

            if m.bias is not None:
                no_decay.append(m.bias)

        else:
            if hasattr(m, 'weight'):
                no_decay.append(m.weight)
            if hasattr(m, 'bias'):
                no_decay.append(m.bias)

    assert len(list(net.parameters())) == len(decay) + len(no_decay)

    return [dict(params=decay), dict(params=no_decay, weight_decay=0)]
```

6 - No bias decay



Baseline



Baseline + Xavier
+ No bias decay