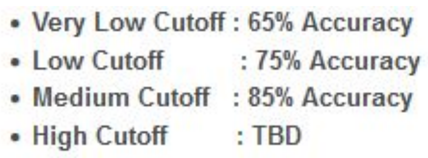
Hackathon 2

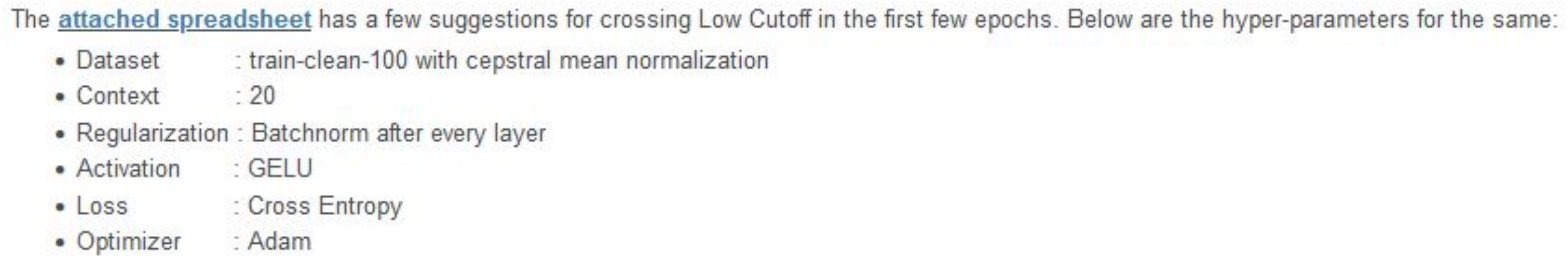
Swathi Jadav, Sarthak Bisht

Resources for HW1P2

- Recitation 1c (Piazza @72): Phoneme Classification Using MLP (toy dataset)
- HW0P2 (Piazza @11): Variations of dataset class for MFCCs (Ex Dataset 3, 4, 5, 6)
- HW1P2 Writeup, Starter Notebook, Suggestions (Recitation 2 & Piazza @118)
- Neural Architecture Search on Toy Dataset (Piazza @221)
- HW1P2 Low Cutoff Architectures (Piazza @261)



Very Low Cutoff	: 65% Accuracy
Low Cutoff	: 75% Accuracy
Medium Cutoff	: 85% Accuracy
High Cutoff	: TBD



The attached spreadsheet has a few suggestions for crossing Low Cutoff in the first few epochs. Below are the hyper-parameters for the same:

Dataset	: train-clean-100 with cepstral mean normalization
Context	: 20
Regularization	: Batchnorm after every layer
Activation	: GELU
Loss	: Cross Entropy
Optimizer	: Adam

Errata - Starter Notebook

```
class AudioDataset(torch.utils.data.Dataset):
      def __init__(self, root, phonemes = PHONEMES, context=0, partition= "train-clean-100"):
            # ⁝
            # ⁝
            # NOTE:
            # Each mfcc is of shape T1 x 27, T2 x 27, ...
            # Each transcript is of shape ~~(T1+2) x 27, (T2+2) x 27~~ (T1+2, ), (T2+2, ) before removing [SOS] and
            [EOS]

            # TODO: Concatenate all mfccs in self.mfccs such that
            # the final shape is T x 27 (Where T = T1 + T2 + ...)
            self.mfccs          = NotImplemented

            # TODO: Concatenate all transcripts in self.transcripts such that
            # the final shape is (T,) meaning, each time step has one phoneme output
            self.transcripts    = NotImplemented
            # Hint: Use numpy to concatenate
            # ⁝
```

Errata - Starter Notebook

```
# Model Initialization
INPUT_SIZE  = (2*CONTEXT + 1) * train_data.mfcc_feat_dim
print("Input size: ", INPUT_SIZE)
model       = Network(INPUT_SIZE, ~~len(train_data.phonemes)~~ 40).to(DEVICE)
summary(model, (INPUT_SIZE, ))
```

Errata - Writeup

Section 5.3: Cepstral Normalization

⋮

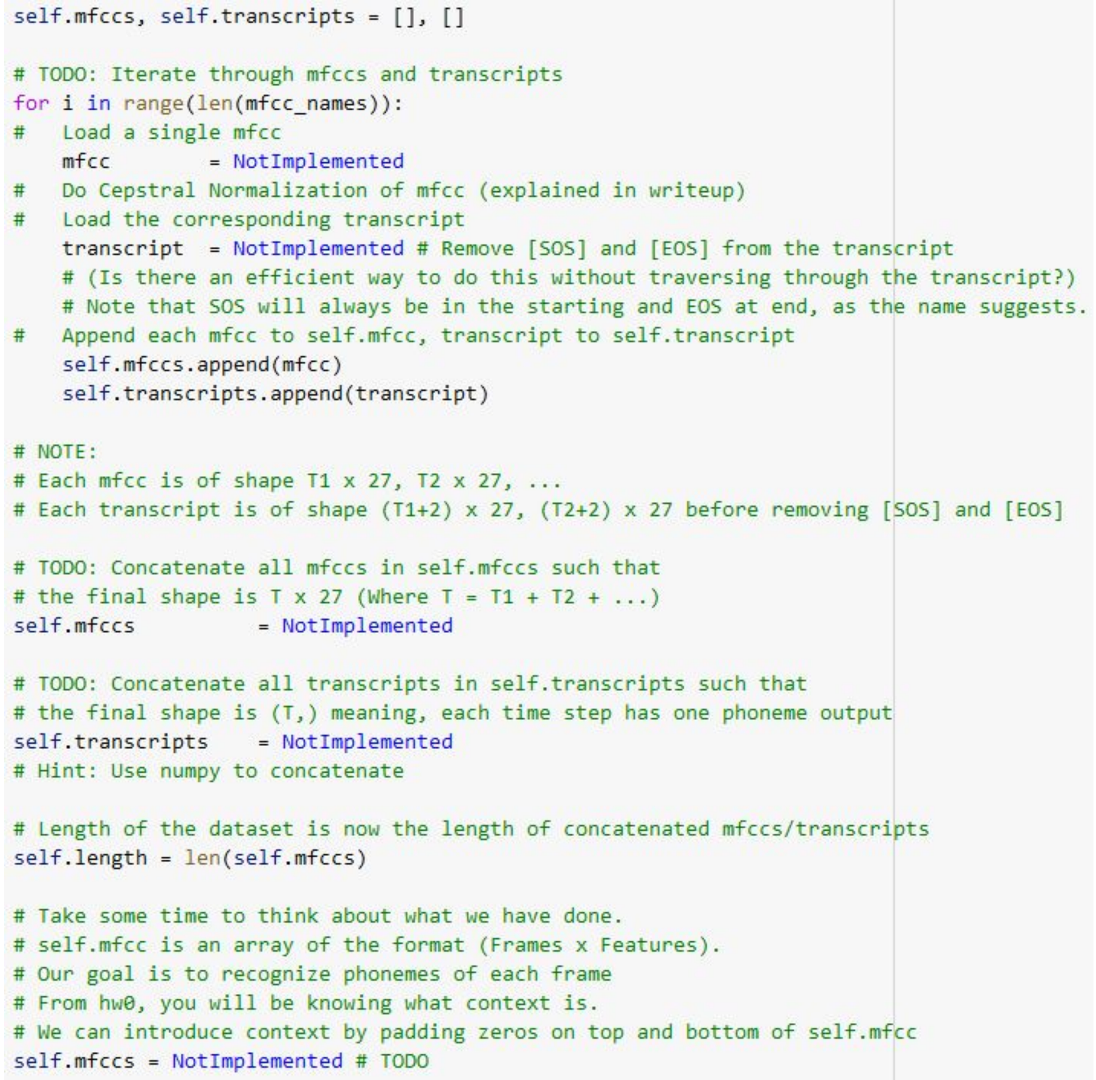
⋮

We now removed channel effects from our signal. In simple words, to our cepstrum, we have subtracted the average from each coefficient and divided by ~~variance~~ standard deviation to perform Cepstral Normalization.

⋮

Dataset Class

1. Create Empty List
2. Append each mfcc, transcript
3. Concatenate
4. Pad for Context
5. Map phoneme (str) to index (int)



```
self.mfccs, self.transcripts = [], []

# TODO: Iterate through mfccs and transcripts
for i in range(len(mfcc_names)):
#   Load a single mfcc
    mfcc        = NotImplemented
#   Do Cepstral Normalization of mfcc (explained in writeup)
#   Load the corresponding transcript
    transcript  = NotImplemented # Remove [SOS] and [EOS] from the transcript
    # (Is there an efficient way to do this without traversing through the transcript?)
    # Note that SOS will always be in the starting and EOS at end, as the name suggests.
#   Append each mfcc to self.mfcc, transcript to self.transcript
    self.mfccs.append(mfcc)
    self.transcripts.append(transcript)

# NOTE:
# Each mfcc is of shape T1 x 27, T2 x 27, ...
# Each transcript is of shape (T1+2) x 27, (T2+2) x 27 before removing [SOS] and [EOS]

# TODO: Concatenate all mfccs in self.mfccs such that
# the final shape is T x 27 (Where T = T1 + T2 + ...)
self.mfccs          = NotImplemented

# TODO: Concatenate all transcripts in self.transcripts such that
# the final shape is (T,) meaning, each time step has one phoneme output
self.transcripts    = NotImplemented
# Hint: Use numpy to concatenate

# Length of the dataset is now the length of concatenated mfccs/transcripts
self.length = len(self.mfccs)

# Take some time to think about what we have done.
# self.mfcc is an array of the format (Frames x Features).
# Our goal is to recognize phonemes of each frame
# From hw0, you will be knowing what context is.
# We can introduce context by padding zeros on top and bottom of self.mfcc
self.mfccs = NotImplemented # TODO
```

Another Approach for Loading Data

```
self.mfccs, self.transcripts = [], []
for i in range(len(mfcc_names)):
    # Load a single mfcc
    mfcc        = NotImplemented
    transcript  = NotImplemented
    self.mfccs.append(mfcc)
    self.transcripts.append(transcript)
# Concatenate into (T, 27) & (T, ) shape
self.mfccs          = NotImplemented
self.transcripts    = NotImplemented
self.length = len(self.mfccs) # length (T)
self.mfccs = NotImplemented # Padding Zeros
# Map phoneme (str) to index (int) values
self.transcripts = NotImplemented
```

```
# How to find T?
self.mfccs = np.zeros((T + pad, 27),
dtype=np.float32)
self.transcripts = np.zeros((T, ), dtype=int)

cx, cy = context, 0
for i in range(length):
    mfcc        = NotImplemented
    transcript  = NotImplemented
    # Map phoneme (str) to index (int) values
    # How to find T_i?
    self.mfccs[cx:cx+T_i] = mfcc
    self.transcripts[cy:cy+T_i] = transcript
    cx, cy = cx + T_i, cy + T_i
```

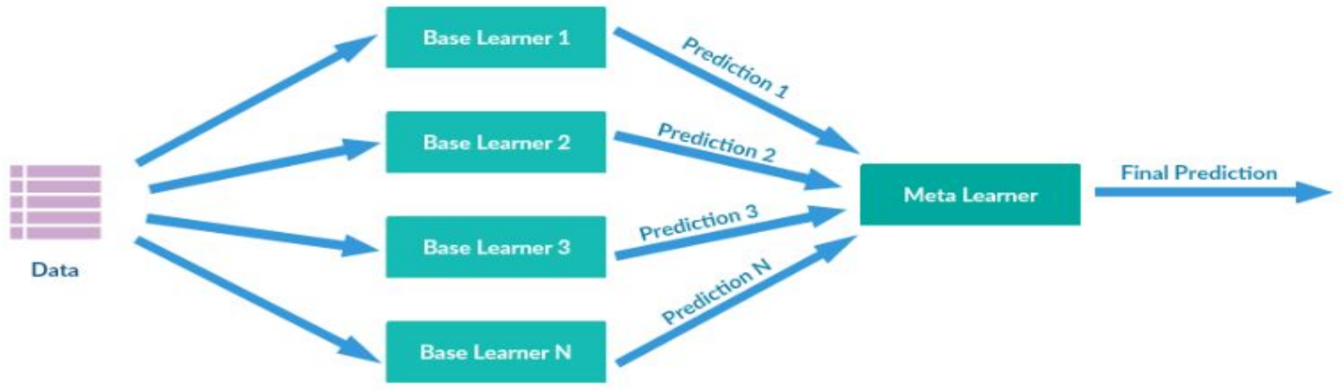
Ensembles

There's advantage of diversity in making mistakes.

So, make interesting mistakes!..

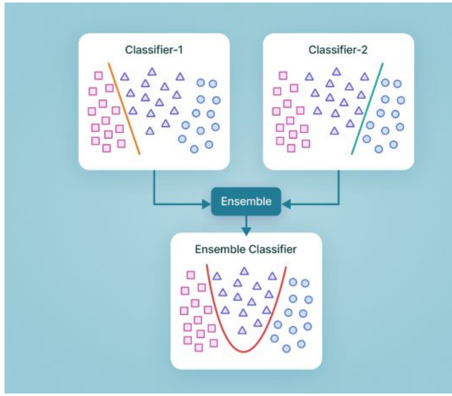
Ensemble Methods

- **Ensemble methods** is a machine learning technique that combines several base models in order to produce one optimal predictive model.
- It is the process of combining multiple deep learning models to obtain a collective performance i.e., to improve the performance of existing models by combining several models thus resulting in one reliable model.



Advantages of Ensembling

- Ensemble Learning is a method of reaching a consensus in predictions by fusing the salient properties of two or more models.
- The final ensemble learning framework is more robust than the individual models that constitute the ensemble because ensembling reduces the variance in the prediction errors.
- Ensemble Learning tries to capture complementary information from its different contributing models.
- there may arise situations where different models perform better on some distributions within the dataset- ensembling overcomes this situation.
- Sometimes, a problem can have a complex decision boundary, and it might become impossible for a single classifier to generate the appropriate boundary.
- Ensemble learning model is information fusion for enhancing classification performance.



Kinds of Ensembling

- **Majority Voting:** The class with Majority predictions is output.
- **Max Rule :** The class prediction of the classifier that predicts with the highest confidence score is deemed the prediction of the ensemble framework.
- **Probability Averaging:** The probability scores for multiple models are first computed. Then, the scores are averaged over all the models for all the classes in the dataset. The class that has the highest probability after the averaging operation is assigned as the predicted class.
- **Weighted Probability Averaging:**Similar to the previous method, the probability or confidence scores are extracted from the different contributing models.But here, unlike the other case, we calculate a weighted average of the probability.

For more info, check the PyTorch Documentation - https://pytorch.org/functorch/1.13/notebooks/ensembling.html

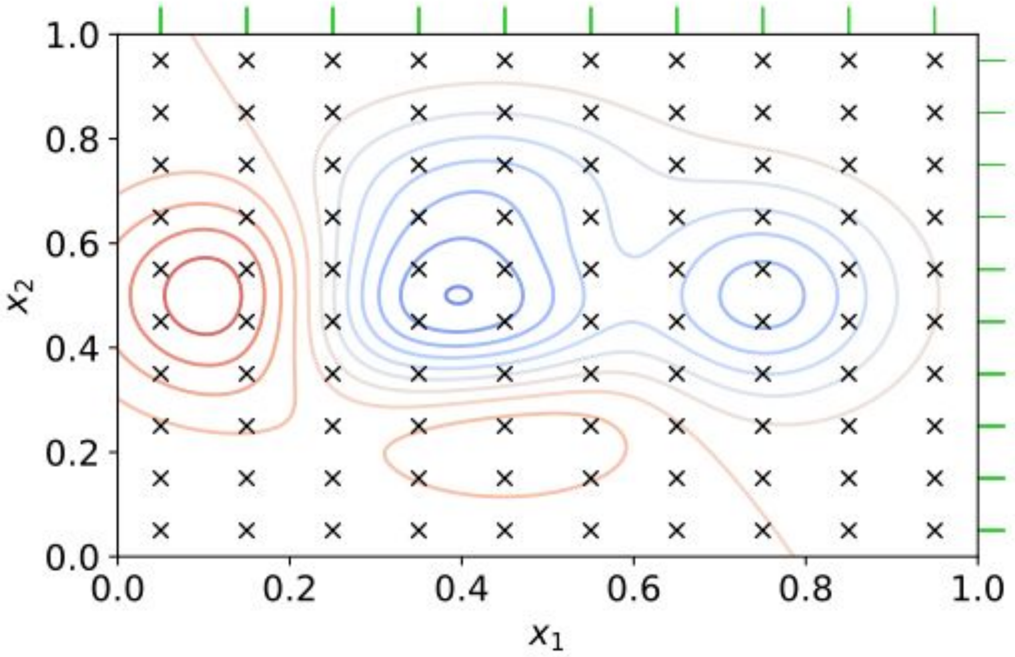
Neural Architecture Search

Finding the best model and hyperparameters... is more of an art than a science.

PS: I prefer Engineering.

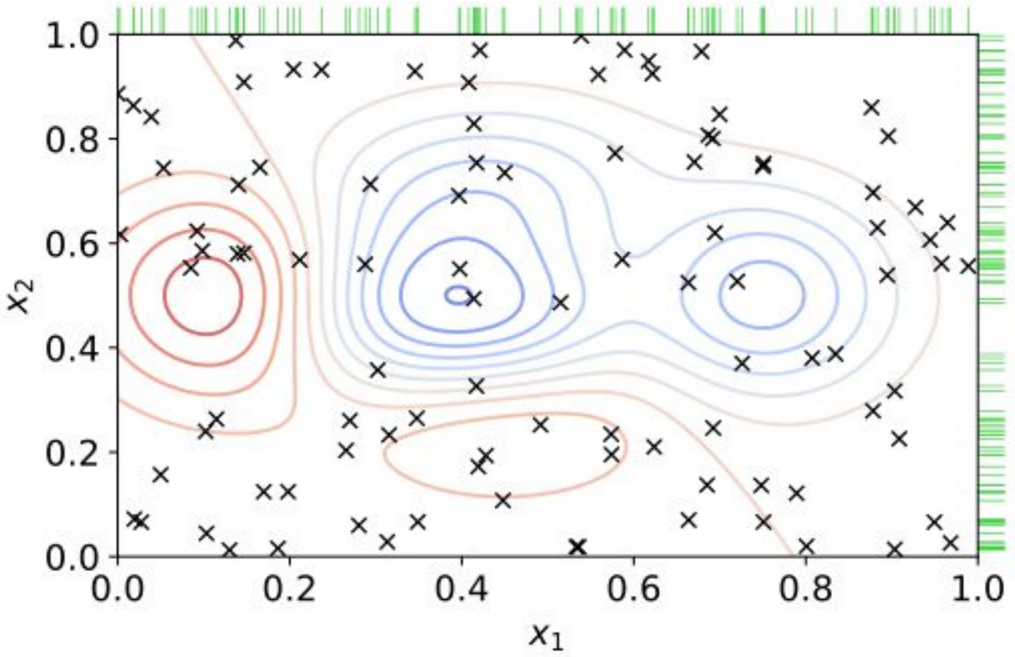
Grid Search

```
MAX_HIDDEN_SIZE = 8000
MAX_CONTEXT = 64
hidden_size = (int)(x1 * MAX_HIDDEN_SIZE)
context = (int)(x2 * MAX_CONTEXT)
in_size = 2*context + 1
model = torch.nn.Sequential(
  torch.nn.Linear(in_size, hidden_size),
  torch.nn.ReLU(),
  torch.nn.Linear(hidden_size, out_size),
)
```

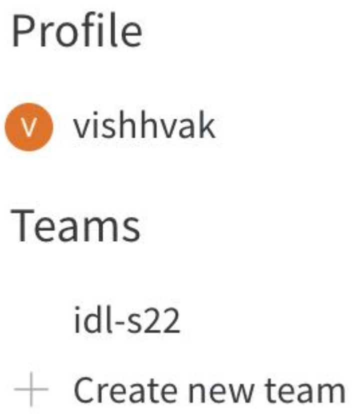


Random Grid Search

```
MAX_HIDDEN_SIZE = 8000
MAX_CONTEXT = 64
hidden_size = (int)(x1 * MAX_HIDDEN_SIZE)
context = (int)(x2 * MAX_CONTEXT)
in_size = 2*context + 1
model = torch.nn.Sequential(
  torch.nn.Linear(in_size, hidden_size),
  torch.nn.ReLU(),
  torch.nn.Linear(hidden_size, out_size),
)
```



Random Grid Search & WandB Team Ablations

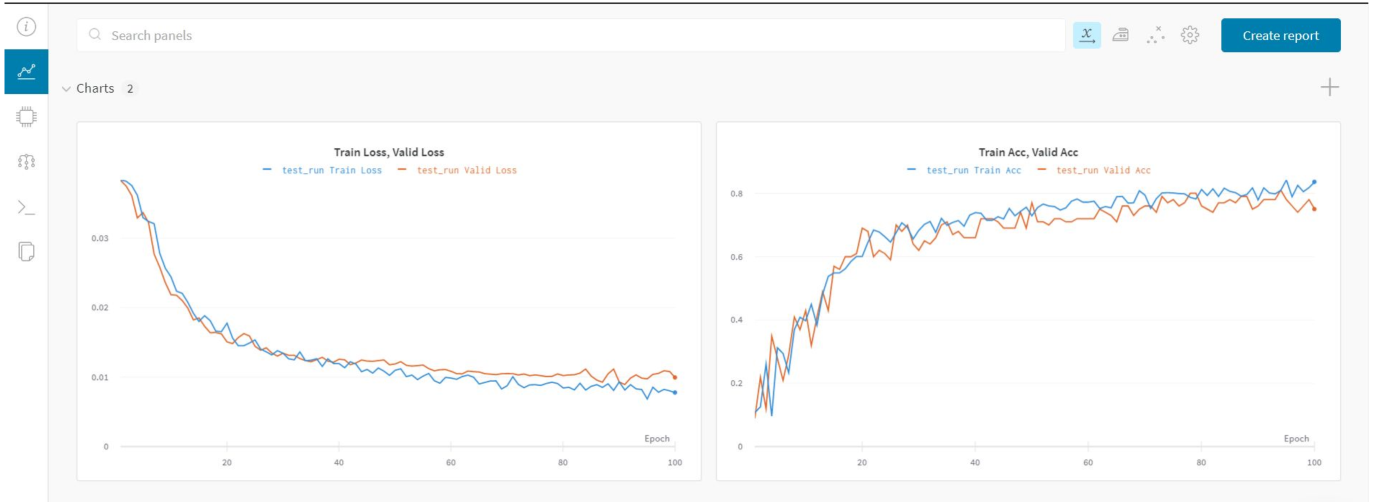

Profile

V vishhvak

Teams

idl-s22

Create new team



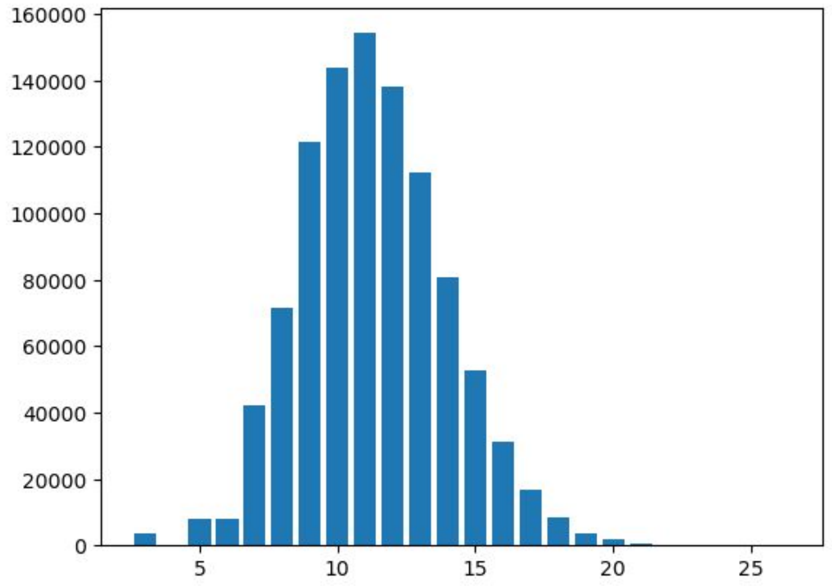
Random Grid Search - Code

```
input_size = 250
target_size = 40
max_parameters = 10_000_000
unused_parameters  =
max_parameters
used_parameters   =  0
min_width = target_size
layers = []
```

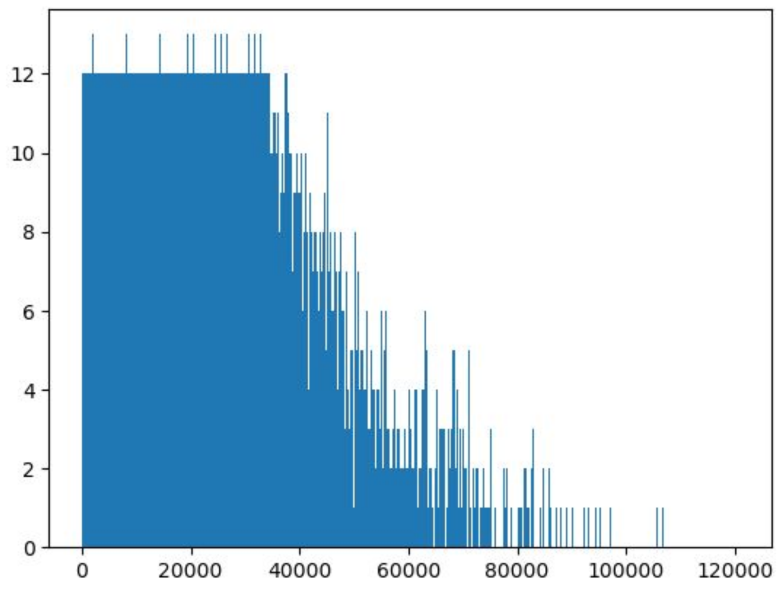
```
while used_parameters < max_parameters:
    max_width  = unused_parameters // (input_size +
target_size)
    if max_width <= min_width:
        break
    output_size = np.random.randint(min_width,
max_width)
    used_parameters   += input_size * output_size
    unused_parameters -= input_size * output_size
    layers.append(output_size)
    input_size = output_size
used_parameters += output_size * target_size
```

Visualizing 1 Million Samples from Simple NAS

Depth vs No. of Samples Drawn



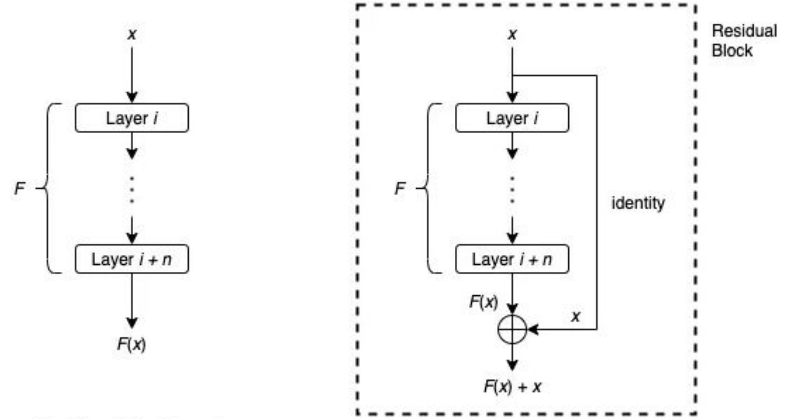
Layer Width vs No. of Samples Drawn



Residual Connections

Residual Connections

- In traditional feedforward neural networks, data flows through each layer **sequentially**: The output of a layer is the input for the next layer.
- **Residual connection** provides another path for data to reach latter parts of the neural network by **skipping** some layers.



Traditional Feedforward without Residual Connection

With Residual Connection

Residual Connections

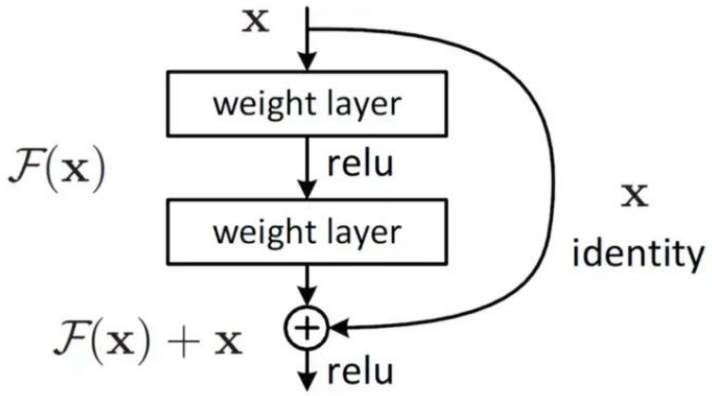
- The residual connection first applies identity mapping to x
- Then it performs element-wise addition $F(x) + x$.
- The whole architecture that takes an input x and produces output $F(x) + x$ is usually called a residual block or a building block.
- Quite often, a residual block will also include an activation function such as ReLU applied to $F(x) + x$.

How do they help??

For feedforward neural networks, training a deep network is usually very difficult, due to problems such as **exploding gradients and vanishing gradients**.

On the other hand, the training process of a neural network with residual connections is empirically shown to converge much more easily, even if the network has several hundreds layers.

It is easier to learn Zero weights than an Identity mapping, if the residual connections aren't present.



Basic and Bottleneck Block

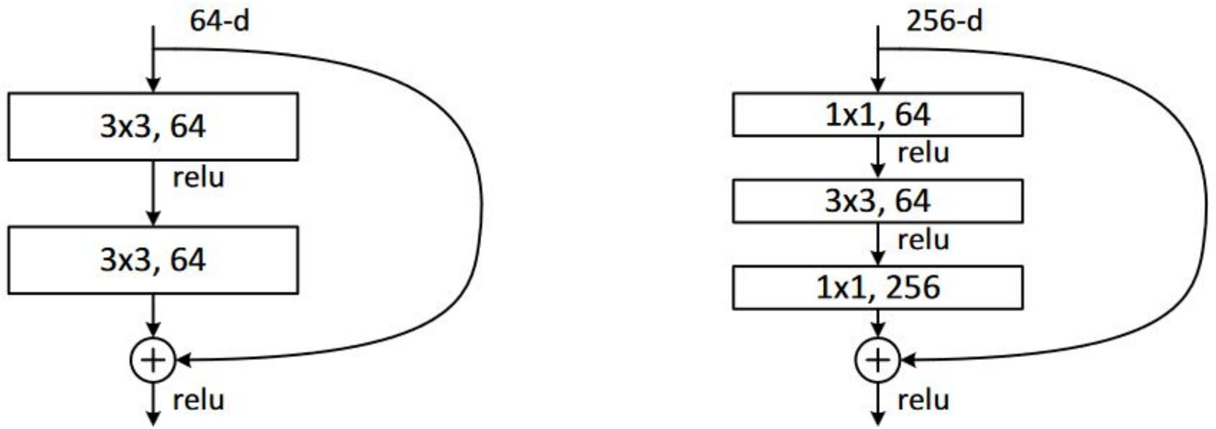


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

Residual Connection - Basic Block

```
class BasicBlock(torch.nn.Module):

  def __init__(self, n_h):

    self.linear0 = torch.nn.Linear(n_h, n_h)
    self.linear1 = torch.nn.Linear(n_h, n_h)

    self.bn0 = torch.nn.BatchNorm1d(n_h)
    self.bn1 = torch.nn.BatchNorm1d(n_h)

    self.relu = torch.nn.ReLU(inplace=True)
```

```
def forward(self, A0):

  R0  = A0

  Z0  = self.linear0(A0)
  BZ0 = self.bn0(Z0)
  A1  = self.relu(BZ0)

  Z1  = self.linear1(A1)
  BZ1 = self.bn1(Z1)
  A2  = self.relu(BZ1 + R0)

  return A2
```

Residual Connection - Bottleneck Block

```python
class Bottleneck(torch.nn.Module):

  def __init__(self, n_h):

    self.residual = torch.nn.Linear(n_h, n_h*4)

    self.linear0 = torch.nn.Linear(n_h, n_h  )
    self.linear1 = torch.nn.Linear(n_h, n_h  )
    self.linear2 = torch.nn.Linear(n_h, n_h*4)

    self.bn0 = torch.nn.BatchNorm1d(n_h  )
    self.bn1 = torch.nn.BatchNorm1d(n_h  )
    self.bn2 = torch.nn.BatchNorm1d(n_h*4)

    self.relu = torch.nn.ReLU(inplace=True)
```

```python
def forward(self, A0):
  R0  = self.residual(A0)

  Z0  = self.linear0(A0)
  BZ0 = self.bn0(Z0)
  A1  = self.relu(BZ0)

  Z1  = self.linear1(A1)
  BZ1 = self.bn1(Z1)
  A2  = self.relu(BZ1)

  Z2  = self.linear2(A2)
  BZ2 = self.bn2(Z2)
  A3  = self.relu(BZ2 + R0)

  return A3
```