Course Project: MNASNet
ECE 590

Gilberto Barrientos
George Gomez
Robert Wallace

Overview

The project investigated the MNASnet architecture which is a result of MNAS, a platform-aware neural architecture search for mobile. We used open-source code to implement the MNASnet, then applied transfer learning by reconfiguring the MNASnet architecture to be compatible with the CIFAR10 dataset and the FashionMNIST dataset. Thereafter, we measured accuracy and latency. The aim of this project was to implement the foundation for NAS and reinforcement learning which take both accuracy and latency into account when calculating the reward.

Several challenges were encountered during the implementation of the project. The first challenge encountered was finding open-source code for MNAS. Significant effort and time were spent attempting to find the open-source code for the paper's neural architecture search. The code was not officially released and there are no open-source implementations of it available since Google did not want it publicized. The only open-source code available were implementations for the MNASnet architecture. The research paper supplies a link to their official MNASnet tutorial on Google's Cloud TPU, however, the website states that their MNASnet tutorial is deprecated and that platform requires a billing account to use their services. Furthermore, many of the other MNASnet tutorials found on GitHub repositories had little to no documentation. We overcame this issue by continuing to search and ultimately finding open-source code and documentation for MNASnet on Pytorch and the aim of the project was shifted to demonstrate our understanding of the MNASnet architecture, NAS, and reinforcement learning rather than implement the MNAS from scratch. When applying transfer learning, we learned that different datasets are not compatible with the MNASnet model immediately. To resolve this issue, we had to adjust the input convolutional layer and the output layers to satisfy the needs of the dataset used. MNASnet was originally trained using ImageNet, which has RGB colored images, which led to three input channels. Only the FashionMNIST data set required the input convolutional to be changed from three input channels to one input channel because the FashionMNIST database consists of only gray-scale images. Similarly, the output layers had to be changed since ImageNet has 1000 output classifications, while FashionMNIST and CIFAR10 have 10 output classifications each.

Method Description

The implementation of MNAS is made up of several key points: Factorized Hierarchical Search Space, Cost Function, and Search Algorithm. Figure 1 shows the factorized hierarchical search space used in MNAS. It is shown that the CNN is broken up into several blocks. Each block is unique and the NAS searches for the operation and connections between each block separately. This is done as the input and output shapes can be used to help determine the accuracy and latency tradeoff.

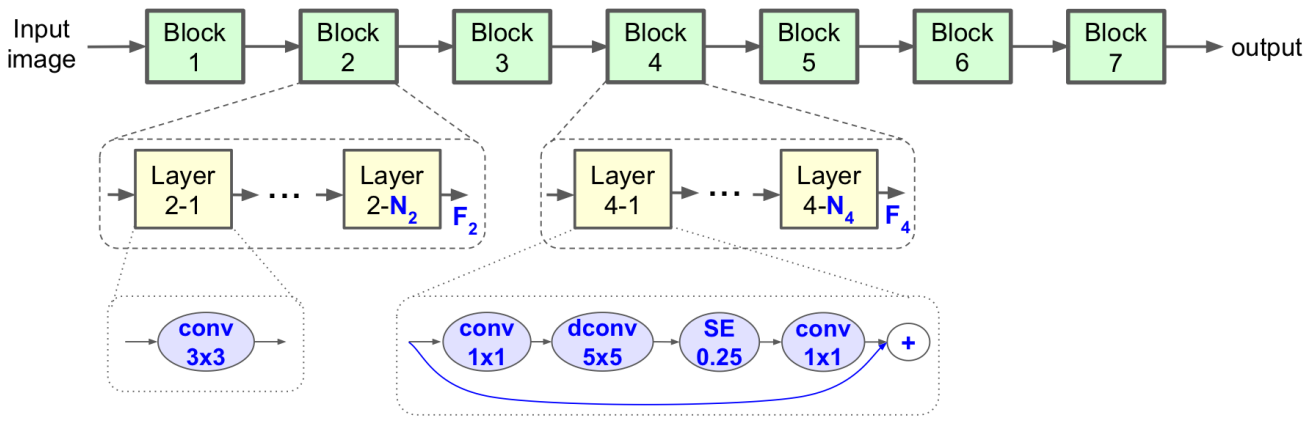


Figure 1: CNN Macroarchitecture

The search space for each block is shown in Figure 2. Each block consists of the same list shown in Figure 2, but there architecture search also implements the per-block subspace, which consists of Convolutional operations, convolutional kernel size, squeeze and excitement, skip operations, output filter size, and number of layers per block. N determines the number of times a layer is repeated for a block, and the remaining determine the layer's architecture. This approach to the search space leads to increased layer diversity rather than simply stacking the same cell repeatedly.

Search Space Per Block i:

- ConvOp: dconv, conv, ...
- KernelSize: 3x3, 5x5
- SERatio: 0, 0.25, ...
- SkipOp: identity, pool, ...
- FilterSize: F_i
- #Layers: N_i

Contents in blue are searched

Figure 2: Search Space

The cost function is an evaluation technique used to evaluate model performance. This is especially prevalent in reinforcement learning where after training, accuracy and latency are observed and altered where seen fit to improve these factors. Such as in Figure 3, where MnasNet's cost function is used to update model parameters. To where m is the sampled model and R(m) denotes the objective value from Figure 4. Through every iteration of reinforcement learning, a new cost function is computed until a number of user-defined steps are done or parameters converge.

The search algorithm was inspired by reinforced learning and chosen due to its ability to adjust rewards with relative ease. Its methodology approach is to find multiple Pareto optimal solutions for multi-objective search problems. Pareto solution is when it has the highest accuracy without increasing latency or lowest latency without decreasing accuracy. This approach overtakes the shortcomings of previous approaches, such as in figure 2, where instead of multiple solutions only looks at a singular metric and measures to maximize accuracy under a target latency constraint. An approach also taken is the mapping of each convolutional neural network model in the search space to a list of tokens. Tokens are determined via a_1:T as seen in Figure 3 based on parameters θ. It looks to maximize reward via the cost function. The search framework consists of 3 main components: the recurrent neural network controller; a trainer to evaluate accuracy, and a mobile device to obtain latency. The first step of which, batches of models are sampled via θ by a prediction of a sequence of tokens. The models are then evaluated for accuracy, given a target task, and evaluated on mobile devices for latency evolution. Reward is then promptly calculated using figure 4. Parameters are then adjusted in the controller using the equation in figure 3 by utilizing proximal policy optimization. These sequence actions are looped until it reaches a user-defined step count or until parameters θ converge.

$$J = E_{P(a_{1:T};\theta)}[R(m)]$$

Figure 3: Cost Function

$$\underset{m}{\text{maximize}} \quad ACC(m) \times \left[\frac{LAT(m)}{T}\right]^{w}$$

Figure 4: Pareto Optimal Solution

$$w = \begin{cases} \alpha, & \text{if } LAT(m) \leq T \\ \beta, & \text{otherwise} \end{cases}$$

Figure 5: Weight Factor

Important Code to Display

Colab Project Notebook Link:
https://colab.research.google.com/drive/1kUxFnoIRW6NkLjrSU7dRR9azNDG3zs83?usp=sharing

To adapt the model to a different dataset requires the modification of the input and output layers of the model. Specifically for CIFAR10, the classification layer needs to be reduced from 1000 outputs to 10 outputs. The following code is used to manage those layers from the preexisting mode:

```
class CifarMnasNet(nn.Module):
  def __init__(self, in_channels=1, pretrained=False):
    super(CifarMnasNet, self).__init__()

    # Load a pretrained mnas model from torchvision.models in Pytorch
    self.model = models.mnasnet1_0(pretrained=pretrained)

    # Change the output layer to output 10 classes instead of 1000 classes
    self.model.classifier[1] = nn.Linear(1280, 10)

  def forward(self, x):
    return self.model(x)
```

Figure 6: CIFAR10 Model Modifications

For FashionMNIST, the number of input channels needs to be reduced from 3 to 1 and the classification layer output needs to also be reduced from 1000 to 10. Below is the code that is used to change the MNASnet model to work with the FashionMNIST model.

```
class MnistMnasNet(nn.Module):
  def __init__(self, in_channels=1, pretrained=False):
    super(MnistMnasNet, self).__init__()

    # Load a pretrained mnas model from torchvision.models in Pytorch
    self.model = models.mnasnet1_0(pretrained=pretrained)

    # Change the input layer to take Grayscale image, instead of RGB images.
    # Hence in_channels is set as 1 or 3 respectively
    # original definition of the first layer on the ResNet class
    # self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)
    self.model.layers[0] = nn.Conv2d(in_channels, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

    # Change the output layer to output 10 classes instead of 1000 classes
    self.model.classifier[1] = nn.Linear(1280, 10)

  def forward(self, x):
    return self.model(x)
```

Figure 7: FashionMNIST Model Modifications

Once the models have been adjusting, training can be performed. A training sequence can be seen below. This process is relatively simple. Using the specified training set, the model is trained using SGD and the results are printed to the console

```
def train(data_loader, model, criterion, optimizer, epoch, device='cpu'):
    all_losses = []
    correct = 0
    print_freq = int(len(data_loader)/20)+1
    for batch_idx, (inputs, targets) in enumerate(data_loader):
        inputs, targets = inputs.to(device), targets.to(device)

        optimizer.zero_grad()                # set optimizer to zero gradient
        outputs = model(inputs)              # perform inference
        loss = criterion(outputs, targets)   # Check the criterion
        loss.backward()                      # compute the gradient
        optimizer.step()                     # update parameters

        # Trace loss and accuracy & Print log information
        all_losses.append(loss)
        pred = outputs.data.max(1, keepdim=True)[1]
        correct += pred.eq(targets.data.view_as(pred)).cpu().sum()
        if batch_idx % print_freq == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(inputs), len(data_loader.dataset),
                100. * batch_idx / len(data_loader), loss.data.item()))

    train_acc = 100. * correct / len(data_loader.dataset)
    train_loss = float(sum(all_losses))/len(data_loader.dataset)
    print("Train set: Average loss: {:.4f}, Accuracy {}/{} ({:.2f}%)".format(
        train_loss, correct, len(data_loader.dataset), train_acc))
    return train_loss, train_acc
```

Figure 8: Training Function

```
lr = 0.01
batch_size=32
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(net.parameters(), lr=lr)
```

Figure 9: Training Specifications

After training, the model can perform inference. Accuracy of the inference along with latency is measured. Currently the latency is measured for the inference on the Colab platform. This technique for latency measurement can be expanded to other platforms for future work in regards to implementation of the NAS.

```
for batch_idx, (inputs, targets) in enumerate(data_loader):
    inputs, targets = inputs.to(device), targets.to(device)
    outputs = run_inference(model, inputs)
    if i==0: # We only want to take a benchmark once -- takes a long time
        timer = benchmark.Timer(stmt="run_inference(model, inputs)",
                                setup="from __main__ import run_inference",
                                globals={
                                    "model": model,
                                    "inputs": inputs
                                })
        #outputs = model(inputs)
        result = timer.timeit(num_repeats)
        test_latency = result.mean
        i = 1
```

Figure 10: Obtaining latency of the model inference

Screenshot of Running Experimental Code and Experimental Results

There are four different experiments that are performed on the model. The first is the model modified for CIFAR10 using a non-pretrained version of the model. The results of training and inference are as follows:

```
net = CifarMnasNet(pretrained=False)
execution(net, 5, 10000, "CF") # CF: Cifar10 -- FM: FashionMNIST
```

```
-------------------- model properties --------------------
Number of parameters: 3115122
================================================================================
-------------------- dataset properites --------------------
Files already downloaded and verified
Files already downloaded and verified
Number of images in train dataset: 10000
Number of images in test dataset: 10000
================================================================================
-------------------- network training begin --------------------
Train Epoch: 0 [0/10000 (0%)]   Loss: 2.395326
Train Epoch: 0 [512/10000 (5%)] Loss: 2.323548
Train Epoch: 0 [1024/10000 (10%)]       Loss: 2.385136
Train Epoch: 0 [1536/10000 (15%)]       Loss: 2.282568
Train Epoch: 0 [2048/10000 (20%)]       Loss: 2.363574
Train Epoch: 0 [2560/10000 (26%)]       Loss: 2.616946
Train Epoch: 0 [3072/10000 (31%)]       Loss: 2.276786
Train Epoch: 0 [3584/10000 (36%)]       Loss: 2.379588
Train Epoch: 0 [4096/10000 (41%)]       Loss: 2.235781
Train Epoch: 0 [4608/10000 (46%)]       Loss: 2.369758
Train Epoch: 0 [5120/10000 (51%)]       Loss: 2.067430
Train Epoch: 0 [5632/10000 (56%)]       Loss: 2.363451
Train Epoch: 0 [6144/10000 (61%)]       Loss: 2.126935
Train Epoch: 0 [6656/10000 (66%)]       Loss: 2.272606
Train Epoch: 0 [7168/10000 (72%)]       Loss: 2.041586
Train Epoch: 0 [7680/10000 (77%)]       Loss: 2.119275
Train Epoch: 0 [8192/10000 (82%)]       Loss: 1.904212
Train Epoch: 0 [8704/10000 (87%)]       Loss: 2.098122
Train Epoch: 0 [9216/10000 (92%)]       Loss: 2.184490
Train Epoch: 0 [9728/10000 (97%)]       Loss: 2.403967
Train set: Average loss: 0.0697, Accuracy 1752/10000 (17.52%)
Test set: Average loss: 0.0721, Accuracy: 1000/10000 (10.00%), Latency: 113.04 ms
```

Figure 11: Non-pretrained CIFAR10 MNASnet

```
Train Epoch: 4 [4096/10000 (41%)]       Loss: 2.302059
Train Epoch: 4 [4608/10000 (46%)]       Loss: 2.312592
Train Epoch: 4 [5120/10000 (51%)]       Loss: 2.309999
Train Epoch: 4 [5632/10000 (56%)]       Loss: 2.307795
Train Epoch: 4 [6144/10000 (61%)]       Loss: 2.306988
Train Epoch: 4 [6656/10000 (66%)]       Loss: 2.294028
Train Epoch: 4 [7168/10000 (72%)]       Loss: 2.310076
Train Epoch: 4 [7680/10000 (77%)]       Loss: 2.305343
Train Epoch: 4 [8192/10000 (82%)]       Loss: 2.303447
Train Epoch: 4 [8704/10000 (87%)]       Loss: 2.300390
Train Epoch: 4 [9216/10000 (92%)]       Loss: 2.300987
Train Epoch: 4 [9728/10000 (97%)]       Loss: 2.303740
Train set: Average loss: 0.0721, Accuracy 1044/10000 (10.44%)
Test set: Average loss: 0.0721, Accuracy: 1000/10000 (10.00%), Latency: 111.83 ms
================================================================================
------------------- plotting loss and accuracy --------------------
================================================================================
```

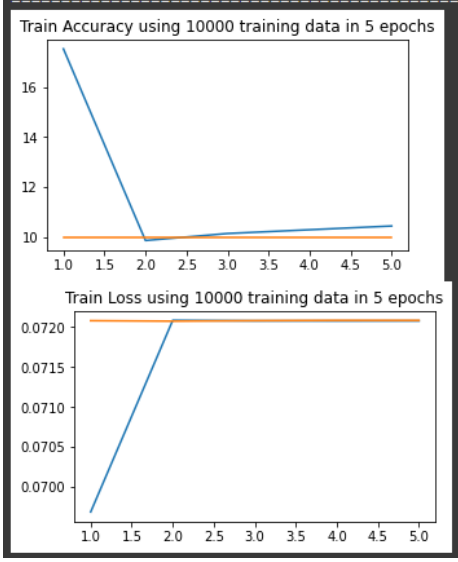


Figure 12: Results of Non-pretrained CIFAR10 MNASnet

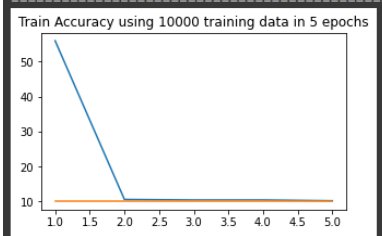
The next experiment is on the FashionMNIST using a non-pretrained model.This results in a similar outcome to the CIFAR10 results. We believe this is due to the light training regime that the model undergoes and that it has much more potential with additional training.

```
net = MnistMnasNet(pretrained=False)
execution(net, 5, 10000, "FM")
```

```
-------------------- model properties --------------------
Number of parameters: 3114546
================================================================================
-------------------- dataset properites --------------------
Number of images in train dataset: 10000
Number of images in test dataset: 10000
================================================================================
-------------------- network training begin --------------------
Train Epoch: 0 [0/10000 (0%)]   Loss: 2.340383
Train Epoch: 0 [512/10000 (5%)] Loss: 2.237601
Train Epoch: 0 [1024/10000 (10%)]       Loss: 1.860770
Train Epoch: 0 [1536/10000 (15%)]       Loss: 1.829366
Train Epoch: 0 [2048/10000 (20%)]       Loss: 1.740903
Train Epoch: 0 [2560/10000 (26%)]       Loss: 0.985441
Train Epoch: 0 [3072/10000 (31%)]       Loss: 1.409422
Train Epoch: 0 [3584/10000 (36%)]       Loss: 1.045402
Train Epoch: 0 [4096/10000 (41%)]       Loss: 0.906548
Train Epoch: 0 [4608/10000 (46%)]       Loss: 1.434517
Train Epoch: 0 [5120/10000 (51%)]       Loss: 0.909087
Train Epoch: 0 [5632/10000 (56%)]       Loss: 0.758513
Train Epoch: 0 [6144/10000 (61%)]       Loss: 0.769700
Train Epoch: 0 [6656/10000 (66%)]       Loss: 1.060483
Train Epoch: 0 [7168/10000 (72%)]       Loss: 1.091627
Train Epoch: 0 [7680/10000 (77%)]       Loss: 0.598268
Train Epoch: 0 [8192/10000 (82%)]       Loss: 0.653755
Train Epoch: 0 [8704/10000 (87%)]       Loss: 0.903959
Train Epoch: 0 [9216/10000 (92%)]       Loss: 0.754430
Train Epoch: 0 [9728/10000 (97%)]       Loss: 0.881197
Train set: Average loss: 0.0375, Accuracy 5599/10000 (55.99%)
Test set: Average loss: 0.0721, Accuracy: 1000/10000 (10.00%), Latency: 111.96 ms
```

Figure 13: Non-pretrained FashionMNIST MNASnet

```
Train Epoch: 4 [3072/10000 (31%)]       Loss: 2.302570
Train Epoch: 4 [3584/10000 (36%)]       Loss: 2.297776
Train Epoch: 4 [4096/10000 (41%)]       Loss: 2.298766
Train Epoch: 4 [4608/10000 (46%)]       Loss: 2.302546
Train Epoch: 4 [5120/10000 (51%)]       Loss: 2.300660
Train Epoch: 4 [5632/10000 (56%)]       Loss: 2.296638
Train Epoch: 4 [6144/10000 (61%)]       Loss: 2.306887
Train Epoch: 4 [6656/10000 (66%)]       Loss: 2.299558
Train Epoch: 4 [7168/10000 (72%)]       Loss: 2.308881
Train Epoch: 4 [7680/10000 (77%)]       Loss: 2.308446
Train Epoch: 4 [8192/10000 (82%)]       Loss: 2.301673
Train Epoch: 4 [8704/10000 (87%)]       Loss: 2.301069
Train Epoch: 4 [9216/10000 (92%)]       Loss: 2.301039
Train Epoch: 4 [9728/10000 (97%)]       Loss: 2.305466
Train set: Average loss: 0.0721, Accuracy 1023/10000 (10.23%)
Test set: Average loss: 0.0721, Accuracy: 1000/10000 (10.00%), Latency: 112.40 ms
================================================================================
-------------------- plotting loss and accuracy --------------------
================================================================================
```



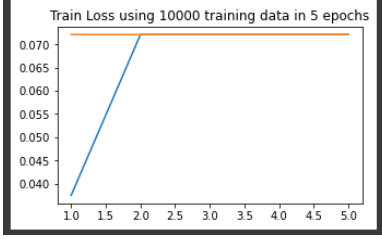


Figure 14: Results of Non-pretrained FashionMNIST MNASnet

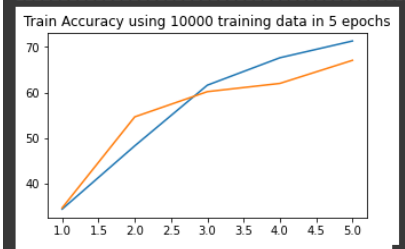
The next two experiments are using a pretrained model on both CIFAR10 and FashionMNIST. The results for the CIFAR10 run are below,

```
net = CifarMnasNet(pretrained=True)
execution(net, 5, 10000, "CF")
```

```
-------------------- model properties --------------------
Number of parameters: 3115122
================================================================================
-------------------- dataset properites --------------------
Files already downloaded and verified
Files already downloaded and verified
Number of images in train dataset: 10000
Number of images in test dataset: 10000
================================================================================
-------------------- network training begin --------------------
Train Epoch: 0 [0/10000 (0%)]   Loss: 2.396159
Train Epoch: 0 [512/10000 (5%)] Loss: 2.376410
Train Epoch: 0 [1024/10000 (10%)]       Loss: 2.136255
Train Epoch: 0 [1536/10000 (15%)]       Loss: 2.330246
Train Epoch: 0 [2048/10000 (20%)]       Loss: 1.982321
Train Epoch: 0 [2560/10000 (26%)]       Loss: 1.831935
Train Epoch: 0 [3072/10000 (31%)]       Loss: 1.875579
Train Epoch: 0 [3584/10000 (36%)]       Loss: 1.835690
Train Epoch: 0 [4096/10000 (41%)]       Loss: 1.792144
Train Epoch: 0 [4608/10000 (46%)]       Loss: 1.819068
Train Epoch: 0 [5120/10000 (51%)]       Loss: 1.940229
Train Epoch: 0 [5632/10000 (56%)]       Loss: 1.688919
Train Epoch: 0 [6144/10000 (61%)]       Loss: 1.926326
Train Epoch: 0 [6656/10000 (66%)]       Loss: 1.483679
Train Epoch: 0 [7168/10000 (72%)]       Loss: 1.312970
Train Epoch: 0 [7680/10000 (77%)]       Loss: 1.676060
Train Epoch: 0 [8192/10000 (82%)]       Loss: 1.543164
Train Epoch: 0 [8704/10000 (87%)]       Loss: 1.674388
Train Epoch: 0 [9216/10000 (92%)]       Loss: 1.600244
Train Epoch: 0 [9728/10000 (97%)]       Loss: 1.693534
Train set: Average loss: 0.0590, Accuracy 3440/10000 (34.40%)
Test set: Average loss: 0.0625, Accuracy: 3467/10000 (34.67%), Latency: 114.59 ms
```

Figure 15: Pretrained CIFAR10 MNASnet

```
Train Epoch: 4 [5120/10000 (51%)]       Loss: 0.918707
Train Epoch: 4 [5632/10000 (56%)]       Loss: 0.733920
Train Epoch: 4 [6144/10000 (61%)]       Loss: 0.580362
Train Epoch: 4 [6656/10000 (66%)]       Loss: 0.623570
Train Epoch: 4 [7168/10000 (72%)]       Loss: 0.737466
Train Epoch: 4 [7680/10000 (77%)]       Loss: 0.918626
Train Epoch: 4 [8192/10000 (82%)]       Loss: 1.273532
Train Epoch: 4 [8704/10000 (87%)]       Loss: 0.785433
Train Epoch: 4 [9216/10000 (92%)]       Loss: 0.782194
Train Epoch: 4 [9728/10000 (97%)]       Loss: 0.996801
Train set: Average loss: 0.0250, Accuracy 7129/10000 (71.29%)
Test set: Average loss: 0.0299, Accuracy: 6705/10000 (67.05%), Latency: 113.39 ms
================================================================================
-------------------- plotting loss and accuracy --------------------
================================================================================
```



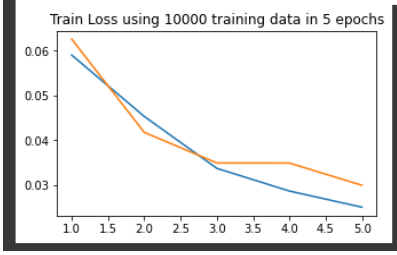


Figure 16: Results of pretrained CIFAR10 MNASnet

The results for FashionMNIST appear to have a higher accuracy and lower latency. Unfortunately the pretraining that the models undergo before download is not described, however it appears that FashionMNIST has a better result than CIFAR10 undergoing the same training run.

```
net = MnistMnasNet(pretrained=True)
execution(net, 5, 10000, "FM")
```

```
-------------------- model properties --------------------
Number of parameters: 3114546
================================================================================
-------------------- dataset properites --------------------
Number of images in train dataset: 10000
Number of images in test dataset: 10000
================================================================================
-------------------- network training begin --------------------
Train Epoch: 0 [0/10000 (0%)]   Loss: 2.389838
Train Epoch: 0 [512/10000 (5%)] Loss: 2.100960
Train Epoch: 0 [1024/10000 (10%)]       Loss: 2.019928
Train Epoch: 0 [1536/10000 (15%)]       Loss: 2.033895
Train Epoch: 0 [2048/10000 (20%)]       Loss: 1.787088
Train Epoch: 0 [2560/10000 (26%)]       Loss: 1.679891
Train Epoch: 0 [3072/10000 (31%)]       Loss: 1.435857
Train Epoch: 0 [3584/10000 (36%)]       Loss: 1.646465
Train Epoch: 0 [4096/10000 (41%)]       Loss: 1.533411
Train Epoch: 0 [4608/10000 (46%)]       Loss: 1.335779
Train Epoch: 0 [5120/10000 (51%)]       Loss: 1.318745
Train Epoch: 0 [5632/10000 (56%)]       Loss: 1.280284
Train Epoch: 0 [6144/10000 (61%)]       Loss: 1.434171
Train Epoch: 0 [6656/10000 (66%)]       Loss: 1.523582
Train Epoch: 0 [7168/10000 (72%)]       Loss: 1.040102
Train Epoch: 0 [7680/10000 (77%)]       Loss: 1.053563
Train Epoch: 0 [8192/10000 (82%)]       Loss: 1.046189
Train Epoch: 0 [8704/10000 (87%)]       Loss: 1.084570
Train Epoch: 0 [9216/10000 (92%)]       Loss: 0.934825
Train Epoch: 0 [9728/10000 (97%)]       Loss: 0.906499
Train set: Average loss: 0.0456, Accuracy 5133/10000 (51.33%)
Test set: Average loss: 0.0726, Accuracy: 1000/10000 (10.00%), Latency: 114.18 ms
```

Figure 17: Pretrained FashionMNIST MNASnet

```
Train Epoch: 4 [7168/10000 (72%)]       Loss: 0.415817
Train Epoch: 4 [7680/10000 (77%)]       Loss: 0.435732
Train Epoch: 4 [8192/10000 (82%)]       Loss: 0.434942
Train Epoch: 4 [8704/10000 (87%)]       Loss: 0.573434
Train Epoch: 4 [9216/10000 (92%)]       Loss: 0.389902
Train Epoch: 4 [9728/10000 (97%)]       Loss: 0.592647
Train set: Average loss: 0.0171, Accuracy 7924/10000 (79.24%)
Test set: Average loss: 0.0168, Accuracy: 7932/10000 (79.32%), Latency: 110.42 ms
================================================================================
-------------------- plotting loss and accuracy --------------------
================================================================================
```

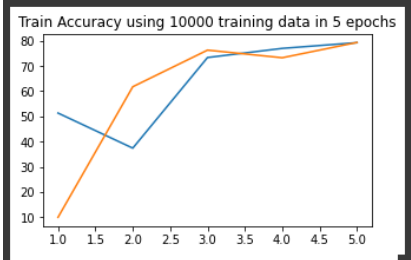


Figure 18: Results of pretrained FashionMNIST MNASnet

The final results accumulated together are as follows. All models underwent 5 epochs of training with a training set of 10,000 and a test set of 10,000. Top-1 accuracy is the only thing that we measure because a top 5 accuracy would appear very high due to only having 10 classifications in each of the data sets.

Dataset	Parameters	Accuracy	Latency
CIFAR10	3,115,122	10.00%	111.83 ms
FashionMNIST	3,114,546	10.00%	112.40 ms
CIFAR10 Pretrained	3,115,122	67.05%	113.39 ms
FashionMNIST Pretrained	3,114,546	79.32%	110.42 ms

Table 1: Experimental Results

Conclusion

The project aimed to investigate MNASnet and to gain a deeper understanding of MNAS and reinforcement learning. The experiment was completed successfully. Through the experiment, we were able to apply transfer learning on MNASnet to the CIFAR10 and FashionMNIST datasets. Each model was run for 5 epochs of training with a training set and testing set of 10,000. Furthermore, we were successfully able to benchmark the latency during the experiments. Comparing the Top-1 accuracies of the experiment, we see that the FashionMNIST dataset running on the pretrained MNASnet had the highest accuracy and the lowest latency. Through this experiment, we are able to learn more about neural architecture searches, gain hands-on practice researching open-source machine learning models, and experience modifying open-source code to our experimental needs. Our project demonstrated the foundation for reinforcement learning. Thus, a possible improvement to this project is to fully implement a NAS from scratch using the building blocks shown throughout our process.

Reference

Link to original paper: https://arxiv.org/abs/1807.11626
Link to source code: https://pytorch.org/vision/stable/_modules/torchvision/models/mnasnet.html
Link to any tutorials used:

Transfer Learning:
https://colab.research.google.com/github/kjamithash/Pytorch_DeepLearning_Experiments/blob/master/FashionMNIST_ResNet_TransferLearning.ipynb