channel_pruning_distance

November 11, 2022

[ ]:
```
import torch  # Provides basic tensor operation and nn operation
import load_dataset as dl  # create dataloader for selected dataset
import load_model as lm  # facilitate loading and manipulating models
import train_model as tm  # Facilitate training of the model
import initialize_pruning as ip  # Initialize and provide basic parameter␣
 ↪require for pruning
import facilitate_pruning as fp  # Compute Pruning Value and many things
import torch.nn.utils.prune as prune
import os  # use to access the files
from datetime import date
```

[ ]:
```
dataset_dir = '/home/pragnesh/project/Dataset/'; selected_dataset_dir =␣
 ↪'IntelIC'
train_folder = 'train'; test_folder = 'test'

# String Parameter for Model
loadModel = False; is_transfer_learning = False

program_name = 'vgg_net_kernel_pruning_3Aug'; model_dir = '/home/pragnesh/
 ↪project/Model/'
selectedModel = 'vgg16_IntelIc_Prune'
load_path = f'{model_dir}{program_name}/{selected_dataset_dir}/{selectedModel}'
```

[ ]:
```
# String parameter to Log Output
logDir = '/home/pragnesh/project/Logs/'
folder_path = f'{logDir}{program_name}/{selected_dataset_dir}/'
logResultFile = f'{folder_path}result.log'
outFile = f'{folder_path}lastResult.log'
outLogFile = f'{folder_path}outLogFile.log'
```

[ ]:
```
if torch.cuda.is_available():
    device1 = torch.device('cuda')
else:
    device1 = torch.device('cpu')
```

[ ]:
```
def ensure_dir(dir_path):
    directory = os.path.dirname(dir_path)
```

```
    if not os.path.exists(directory):
        os.makedirs(directory)
```

[ ]:
```
ensure_dir(f'{model_dir}{program_name}/')  # dir /home/pragnesh/project/Model/
 ↪vgg_net_kernel_pruning_3Aug/
ensure_dir(f'{model_dir}{program_name}/{selected_dataset_dir}/')  # dir ~/
 ↪vgg_net_kernel_pruning_3Aug/IntelIc/
ensure_dir(f'{logDir}{program_name}')  # dir /home/pragnesh/project/Logs/
 ↪program_name/
ensure_dir(f'{logDir}{program_name}/{selected_dataset_dir}/')  # dir /home/
 ↪pragnesh/project/Logs/program_name/IntelIC
```

[ ]:
```
dl.set_image_size(224)
dl.set_batch_size = 16
dataLoaders = dl.data_loader(set_datasets_arg=dataset_dir,␣
 ↪selected_dataset_arg=selected_dataset_dir,
                              train_arg=train_folder, test_arg=test_folder)
```

[ ]:
```
if loadModel:  # Load the saved trained model
    new_model = torch.load(load_path, map_location=torch.device(device1))
else:  # Load the standard model from library
    # if we don't have any saved trained model download pretrained model for␣
 ↪transfer learning
    new_model = lm.load_model(model_name='vgg16', number_of_class=6,␣
 ↪pretrainval=is_transfer_learning,
                               freeze_feature_arg=False, device_l=device1)
```

[ ]:
```
today = date.today()
d1 = today.strftime("%d-%m")
print(f"\n..........OutLog For the {d1}.............")
with open(outLogFile, 'a') as f:
    f.write(f"\n\n.......................OutLog For the {d1}.............
 ↪.....\n\n")
f.close()
```

[ ]:
```
block_list = []; feature_list = []; conv_layer_index = []; module = []
prune_count = []; new_list = []; candidate_conv_layer = []
layer_number = 0; st = 0; en = 0
```

[ ]:
```
def initialize_lists_for_pruning():
    global block_list, feature_list, conv_layer_index, module, prune_count,␣
 ↪new_list, candidate_conv_layer
    global layer_number, st, en
    block_list = ip.create_block_list(new_model)  # ip.getBlockList('vgg16')
    feature_list = ip.create_feature_list(new_model)
    conv_layer_index = ip.find_conv_index(new_model)
```

```
    module = ip.make_list_conv_param(new_model)
    prune_count = ip.get_prune_count(module=module, blocks=block_list, max_pr=.
 ↪1)
    new_list = []
    layer_number = 0
    st = 0
    en = 0
    candidate_conv_layer = []

initialize_lists_for_pruning()
```

[ ]:
```
def compute_conv_layer_dist_channel_pruning(module_cand_conv, block_list_l,␣
 ↪block_id):
    global layer_number
    candidate_convolution_layer = []
    end_index = 0
    for bl in range(len(block_list_l)):
        start_index = end_index
        end_index = end_index + block_list_l[bl]
        if bl != block_id:
            continue

        with open(outLogFile, "a") as out_file:
            out_file.write(f'\nblock ={bl} blockSize={block_list_l[bl]},␣
 ↪start={start_index}, End={end_index}')
        out_file.close()
        # newList = []
        # candidList = []
        for lno in range(start_index, end_index):
            # layer_number =st+i
            with open(outLogFile, 'a') as out_file:
                out_file.write(f"\nlno in compute candidate {lno}")
            out_file.close()
            candidate_convolution_layer.append(fp.
 ↪compute_distance_score_channel(
                module_cand_conv[lno]._parameters['weight'],
                n=1,
                dim_to_keep=[0],
                prune_amount=prune_count[lno]))
        break
    return candidate_convolution_layer
```

[ ]:
```
class ChannelPruningMethodSimilarities(prune.BasePruningMethod):
    PRUNING_TYPE = 'unstructured'

    def compute_mask(self, t, default_mask):
        with open(outLogFile, "a") as log_file:
```

```
            log_file.write("\n Executing Compute Mask")
        log_file.close()
        mask = default_mask.clone()
        # mask.view(-1)[::2] = 0
        size = t.shape
        print(f"\n{size}")
        with open(outLogFile, "a") as log_file:
            log_file.write(f'\nLayer Number:{layer_number} \nstart={st}␣
↪\nlength of new list={len(new_list)}')
        log_file.close()
        for k1 in range(len(new_list)):
            for k2 in range(len(new_list[layer_number - st][k1])):
                i = new_list[layer_number - st][k1][k2][1]
                j = new_list[layer_number - st][k1][k2][0]
                if k1 == j:
                    print(":", end='')
                # print(f"i= {i} , j= {j}")

                mask[i][j] = 0
        return mask
```

[ ]:
```
def channel_unstructured_similarities(kernel_module, name):
    ChannelPruningMethodSimilarities.apply(kernel_module, name)
    return kernel_module
```

[ ]:
```

```