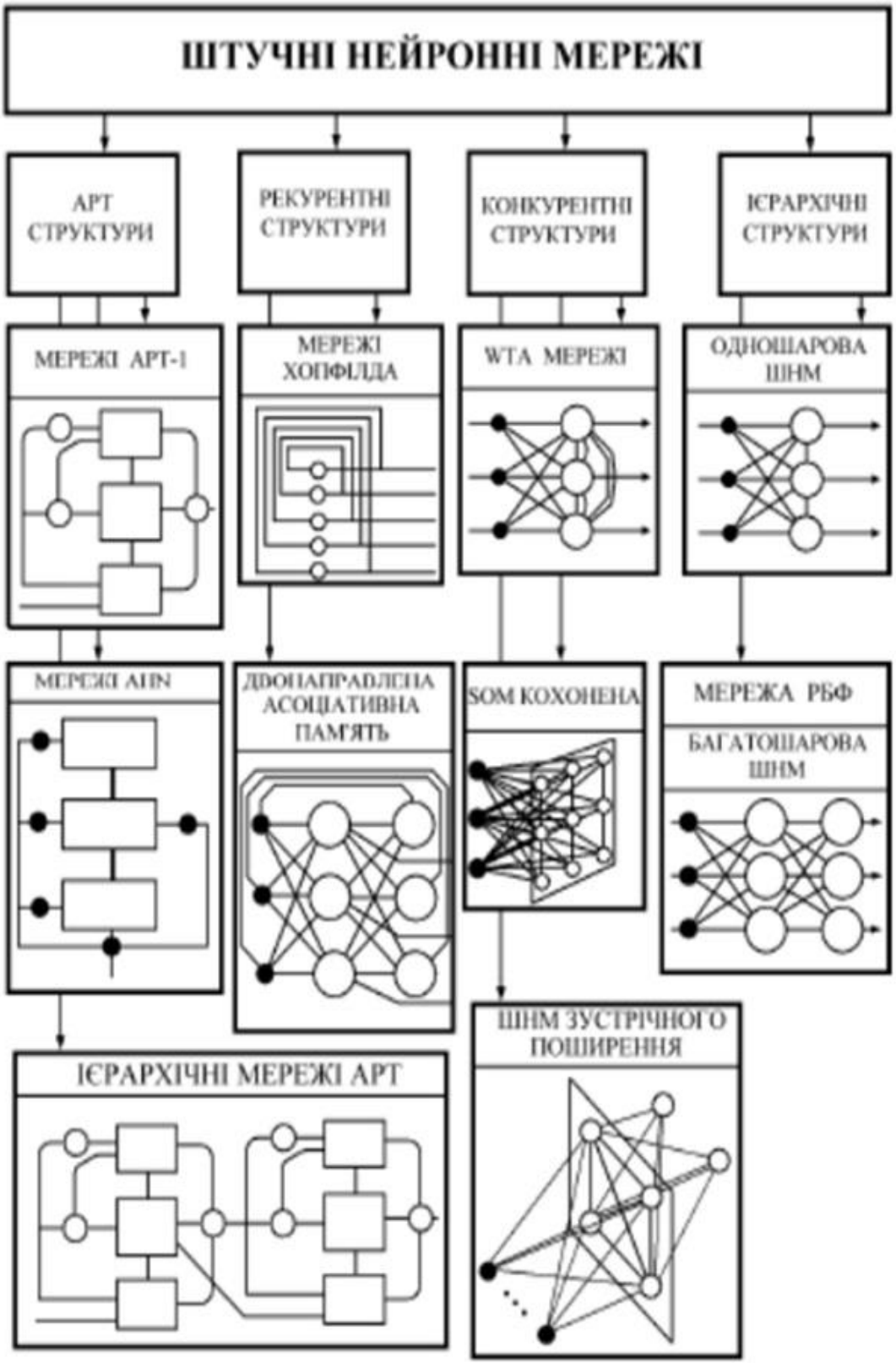
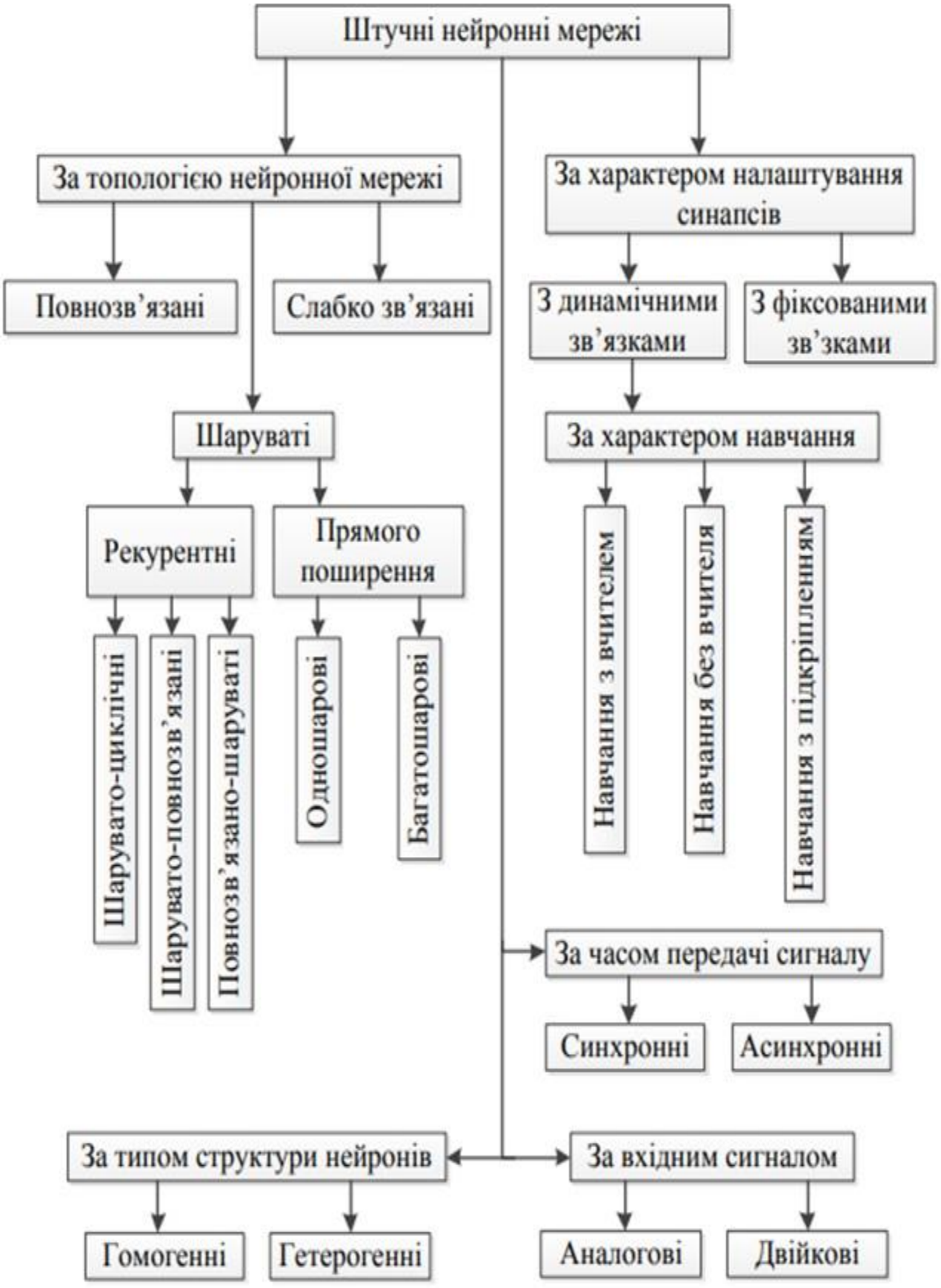
ДОДАТОК А

Графічний матеріал роботи

А.1 Різні типи нейронних структур



А.2 Основні підходи до класифікації штучної нейронної мережі



А.3 Приклади зображень створених за допомогою штучної нейронної мережі NERF

А.4 Слайд-презентація побудови реалістичних 3D зображень

Побудова реалістичних 3D зображень за допомогою штучної нейронної мережі NeRF

Виконав: ст. гр.
Студент 2 курсу, групи КІТм-21-1
факультету «комп'ютерної інженерії та управління»
Плеханов Д.В.

Актуальність дослідження.

Системи розпізнавання графічної інформації відіграють важливу роль та є основними компонентами обчислювальних пристроїв, застосовуваних у різних сферах людської діяльності. Такі системи дозволяють значно спростити вирішення повсякденних задач, пов'язаних з обробкою графічної інформації, що й виділяє їм особливе місце у галузях. Наразі існує значна кількість нейромережевих систем здатних до використання у вирішенні задач розпізнавання об'єктів. Не зважаючи на значну кількість наукових досліджень що виконуються в цій області, наразі не існує найкращої в усіх аспектах моделі нейронної мережі. Однією із найперспективніших штучних нейронних мереж для створеннях 3D зображення є NeRF.

Мета роботи.

Проаналізувати особливості побудови реалістичних 3D зображень за допомогою штучної нейронної мережі NeRF.

Об'єктомдослідженняє побудова реалістичних 3D зображень.

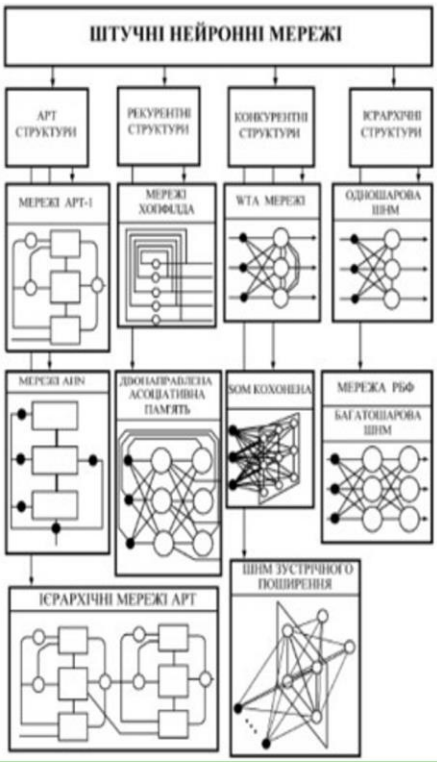
Предмет дослідження:

сукупність необхідних умов, що забезпечують найкращий підхід до особливості побудови реалістичних 3D зображень за допомогою штучної нейронної мережі NeRF.

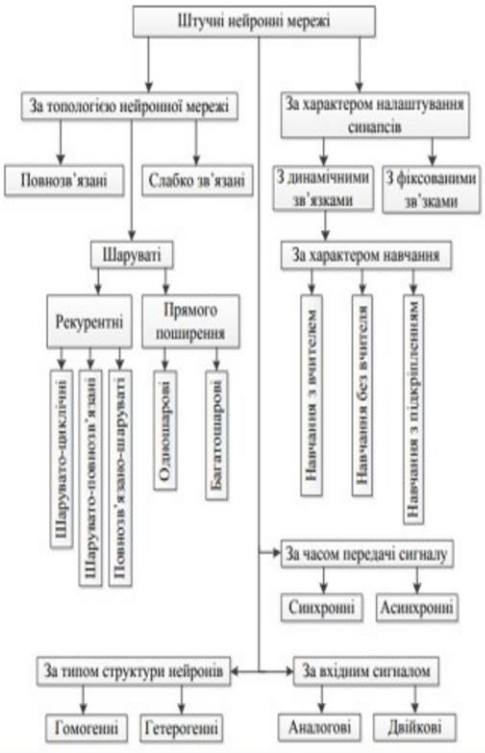
Завдання дослідження можна сформулювати так:

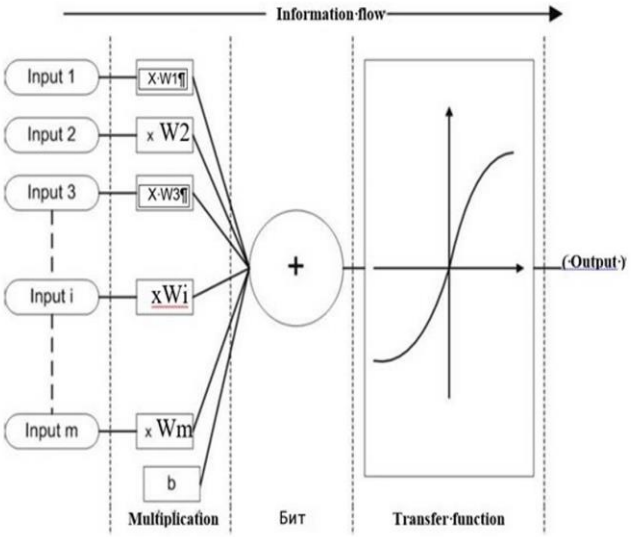
- розкрити сутність поняття «штучні нейронні мережі» та головні складові мережі;
  розглянути особливості створення штучного нейрону, способи навчання та оптимізації ;
- провести огляд нейронних мереж для створення зображень;
- описати архітектуру штучних нейронних мереж та математична сутність нейронної мережі NERF;
- перелічити існуючі переваги та недоліки нейронної мережі NERF;
- навести приклад практичного використання мережі NERF для створення зображень.

Різні типи нейронних структур

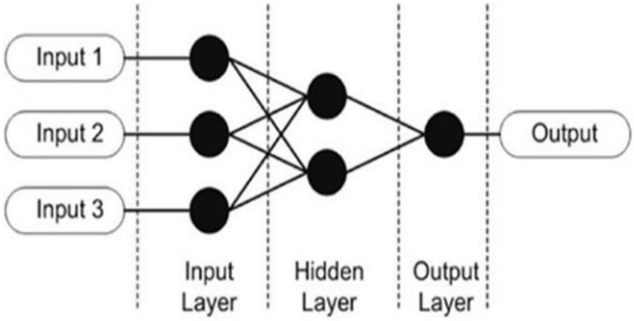


Основні підходи до класифікації штучної нейронної мережі



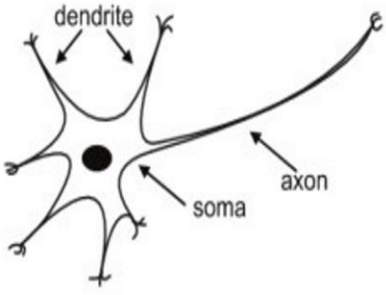


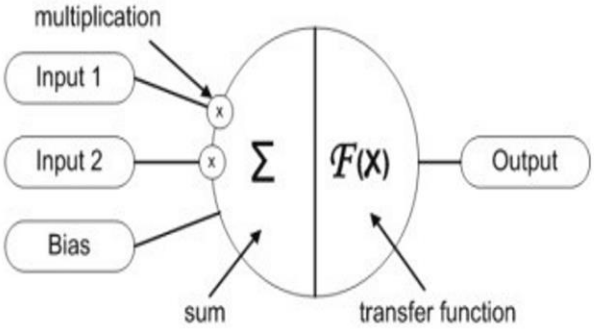
Схематичне зображення штучного нейрону



Штучна нейронна мережа

Порівняння біологічного та штучного нейрону





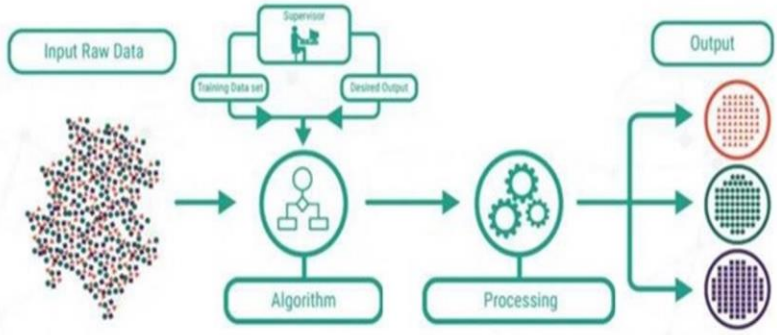


Схема навчання з учителем

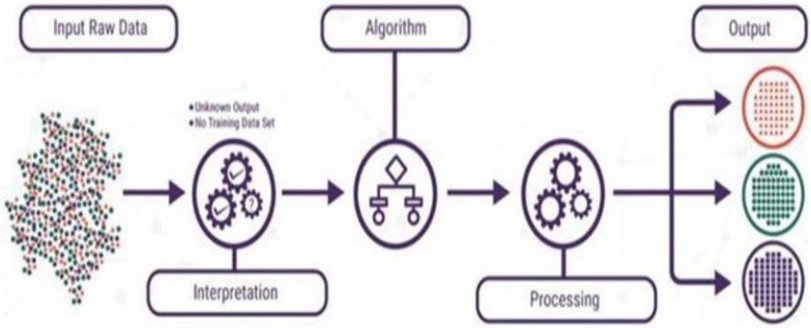
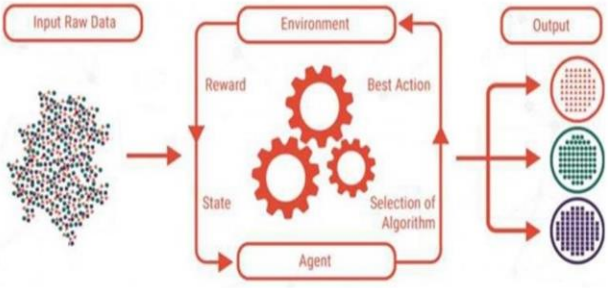
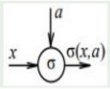
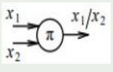
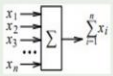


Схема навчання без учителя

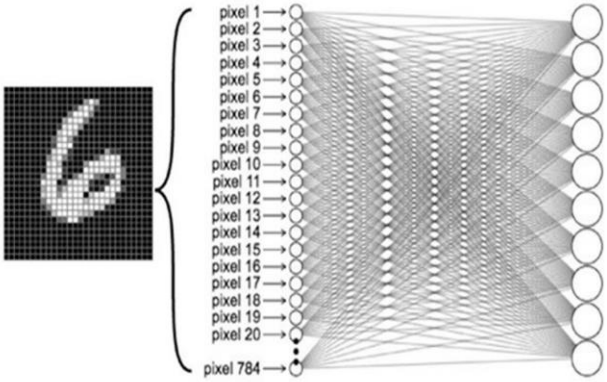
Схема навчання з підкріпленням



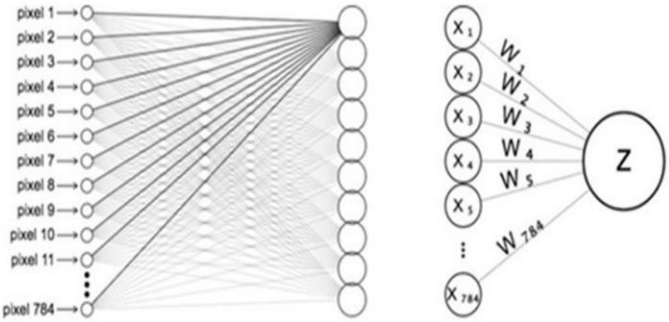
Основні елементи нейронних мереж

	Синапс, має два входи: вхід сигналу x та вхід синаптичної ваги a, вихідний сигнал буде рівний добутку цих значень
	Точка розгалуження, має один вхід x та декілька виходів x_1, x_2, x_n. Причому Xi=X
	Нелінійна сигмоїдальна функція активації, має один вхідний сигнал д: та один параметр a.
	Пороговий перетворювач, який реалізує функцію визначення знаку.
	Нелінійний Паде перетворювач. Якщо вхідні сигнали x_1, x_2, то вихідний сигнал буде рівний x_1 / x_2
	Помножувач. Має два входи та реалізує функцію множення
	Суматор. Якщо x_1, x_2,.., x_n входи суматора, то на вихід рівний $\sum_{i=1}^{n} x_i$

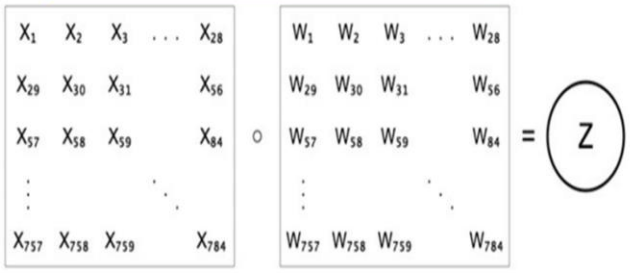
Одношарова штучна нейронна мережа для розпізнавання рукописних цифр



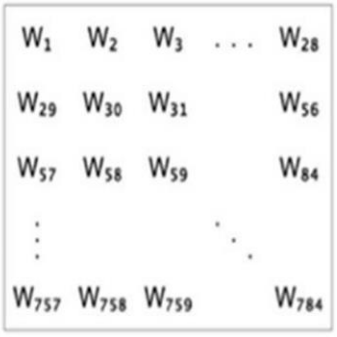
З'єднання до вихідного нейрона

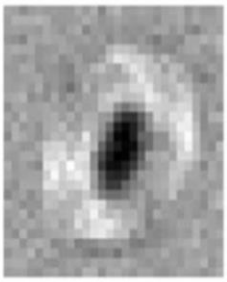


Візуалізація множення пікселів на ваги

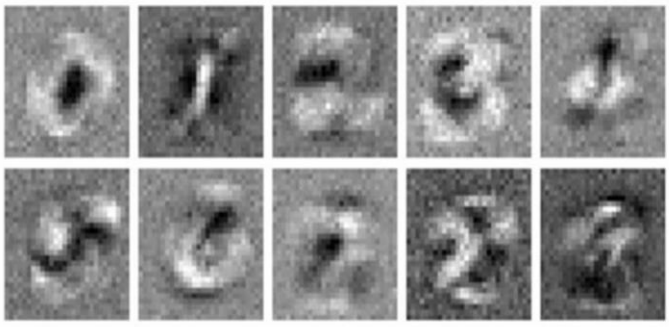


Візуалізація ваг вихідного нейрона, який відповідає за класифікацію цифри 0

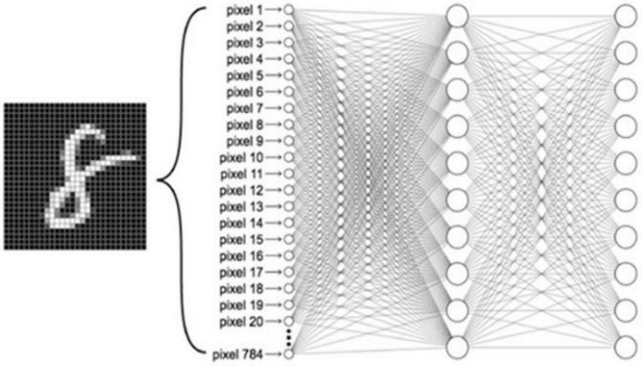




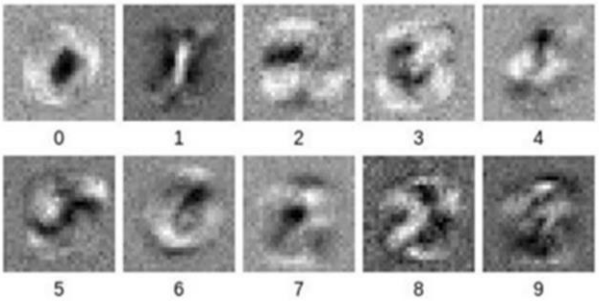
Візуалізація ваг вихідних нейронів



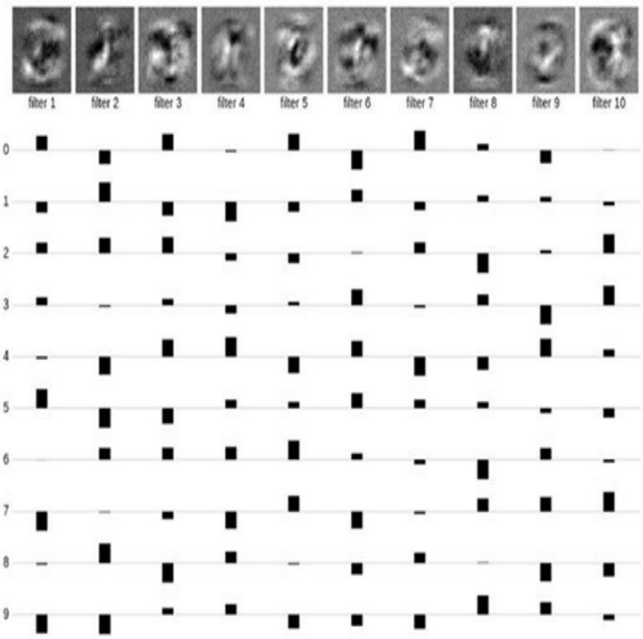
Двошарова штучна нейронна мережа для розпізнавання рукописних цифр



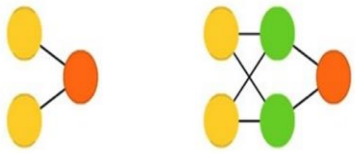
Візуалізація ваг першого (прихованого) шару нейронів



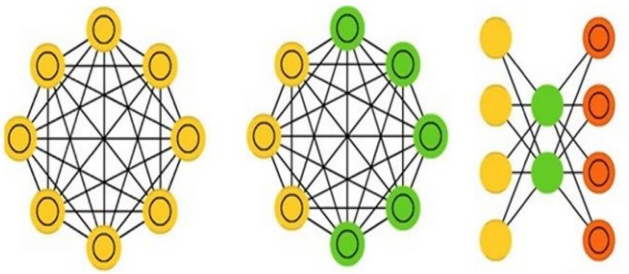
Візуалізація ваг другого (вихідного) шару нейронів та активації нейронів для кожного класу



Огляд нейронних мереж для створення зображень



Структура нейронної мережі прямого поширення

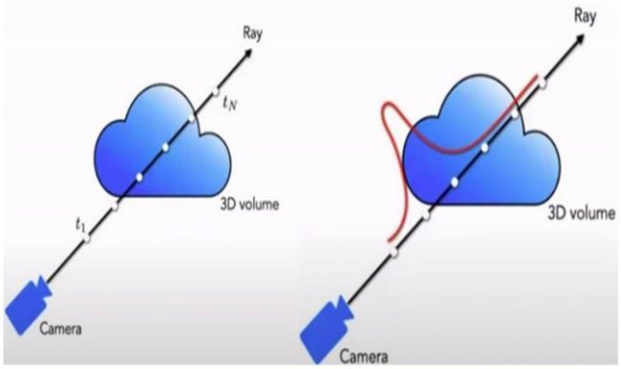


Структура мережі Хопфілда

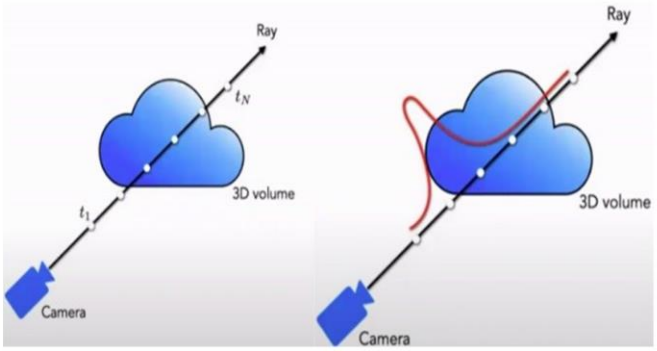
Структура машини Больцмана

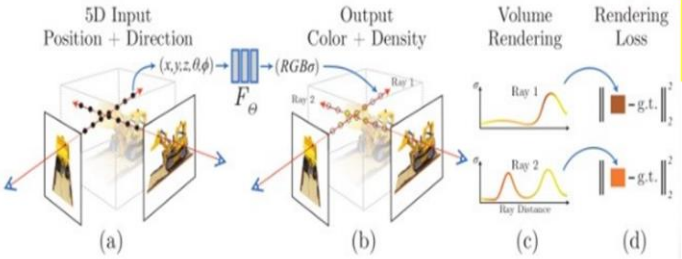
Структура автоенкодера

Neural Radiance Fields (NeRF)



На промені семплексується N точок, щоб змоделювати розподіл щільності на конкретному напрямку



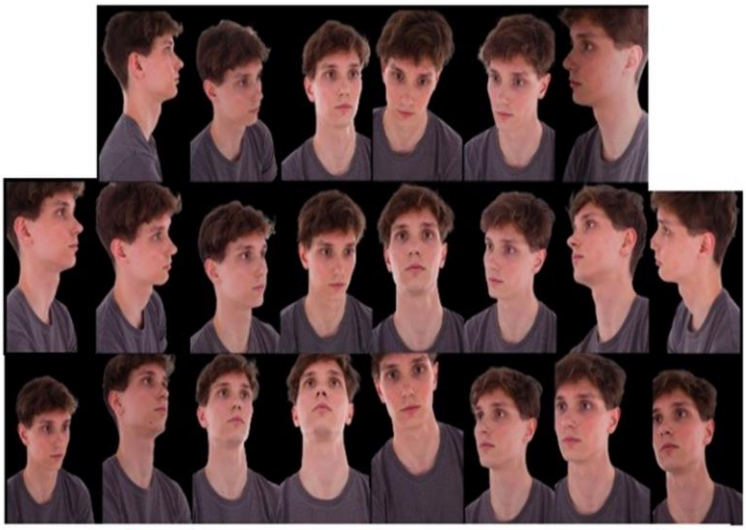


Розподіл на промені уточнюється після додаткового семплювання для Fine сітки

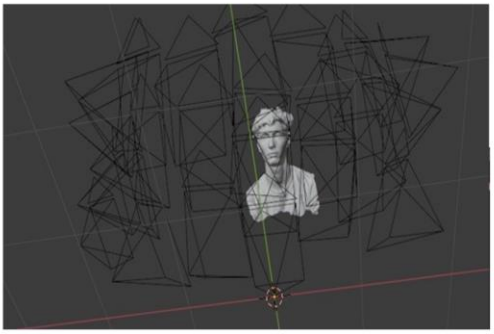
Схематичне зображення процесу навчання мережі NERF

Практичне використання мережі NERF для створення зображень

Вихідні фотографія До та після видалення фону



Набір фотографій особи, отриманих за допомогою камер для фотограмметрії



Сетап сцени, що вивантажений в Blender

Приклади зображень створених за допомогою штучної нейронної мережі NERF

Обов'язково дивитись зображення в режимі «показ слайда»

Модуль, який відповідає за тренування штучного інтелекту, необхідного для машинного навчання

```
from absl import *
import gin
from functools import *
import tensorflow as tf
from nerfies import *
from typing import *
import numpy as np
from flax import *
from flax. metrics import tensorboard as TB
from jax import *
def TB_in_log (printN, postition, states, poll, necessary_lib) :
step = int (postition. optimizer. postition. step)
iterate = ['good', 'rough']
for hbc in iterate:
if hbc in poll:
for pks, ern in poll [hbc]. items():
printN. write (str (pks) + "/" + str (hbc) + " " + str (ern) + str (step))
for k, v in necessary_lib. items():
printN. scalar («execution " + str (k) + " " + str (v) + " " + str (step))
if 'MinusScene' not in poll: return
printN. scalar('dropping' + str (poll['MinusScene']) + " " + str (step))
if __name__ == '__main__':
constFS = flags. constFS
tf. config. experimental . set_visible_tools([], 'GPU')
gin_configs = constFS. gin_configs
gin. parse_config_files_and_bindings (gin_configs, constFS. gin_bindings, True)
save = GPath (constFS. base_folder) / subname / 'mainTs' / 'learn' / 'saves'
mainT = GPath (constFS. base_folder) / subname / 'mainTs' / 'learn'
```

```
        tutorCG = learnConfig()
        baseCG = replicaConfig()
        frame = ExperimentConfig()
        clue = PclueKey (random_seed)
        np. random. seed (random_seed + process_index())
        detailOc = baseAsets. from_config (rawNec, image_scale, baseCG.
use_appearance_metabaseA,
        baseCG. use_camera_metabaseA, baseCG. use_warp,
        baseCG. warp_metabaseA_encoder_infoTP == 'time', random_seed)
        tools = local_tools()
        rawNec = {'infoTP': detailOc_infoTP, 'basePS': constFS. basePS}
        dots = None
        if tutorCG. use_MinusScene :
        dotsMain = detailOc. load_points (shuffle=True)
        dotsQ = min (np. size (dotsMain), np. size (tools) · tutorCG. basement_dotsQ) –
min (np. size (dotsMain), np. size (tools) · tutorCG. basement_dotsQ)% np. size (tools)
        dotsST = tf. baseA. baseAset. from_tensor_slices (dotsMain)
        dots = iterator_from_baseAset (dotsST,lenB=dotsQ, cache= 3, tools=tools)
        dropEWS = from_config (tutorCG. dropEWSule)
clockMAINS = from_config (tutorCG. clockMAINSule)
chargeLS = from_config (tutorCG. lr_schedule)
twistMAINS = from_config (tutorCG. twistMAINSule)
clue, key = random. split (clue)
params = {}
replica, params['replica'] = construct_nerf (key, baseCG, tutorCG. lenB, detailOc.
overlPS, detailOc. shootPS, detailOc. bend, detailOc. close, detailOc. long, tutorCG.
use_elastic_loss, tutorCG. use_elastic_loss)
enhance = optim. Adam (chargeLS (0))
optimizer = enhance. create (params)
```

ВИСНОВКИ

1. На початку дослідження наголошено на тому, що системи розпізнавання графічної інформації відіграють важливу роль та є основними компонентами обчислювальних пристроїв, застосовуваних у різних сферах людської діяльності.

2. Розкрито сутність поняття «штучні нейронні мережі» та названо головні складові нейронної мережі. Описано особливості створення штучного нейрону та проаналізовано три способи навчання, перелічено їх недоліки та переваги. Зроблено огляд нейронних мереж для створення зображень.

3. Описана архітектура штучних нейронних мереж та зроблено математичний опис нейронної мережі NERF та сказано, що сама нейромережа навчається на обмеженій кількості фотографій і може бути використана для отримання досить якісних карт глибин зображень.

4. Наведено приклад створення якісних зображень з допомогою NeRF при наведено програмний код застосунку.

ДОДАТОК Б
КОДОВИЙ МАТЕРІАЛ РОБОТИ

Б.1 Модуль, який відповідає за тренування штучного інтелекту, необхідного для машинного навчання

```
from absl import *
import gin
from functools import *
import tensorflow as tf
from nerfies import *
from typing import *
import numpy as np
from flax import *
from flax. metrics import tensorboard as TB
from jax import *
def TB_in_log (printN, postition, states, poll, necessary_lib):
step = int (postition. optimizer. postition. step)
iterate = ['good', 'rough']
for hbc in iterate:
if hbc in poll:
for pks, ern in poll [hbc]. items():
printN. write (str (pks) + "/" + str (hbc) + " " + str (ern) + str (step))
for k, v in necessary_lib. items():
printN. scalar («execution " + str (k) + " " + str (v) + " " + str (step))
if 'MinusScene' not in poll: return
printN. scalar('dropping ' + str (poll['MinusScene']) + " " + str (step))
if __name__ == '__main__':
constFS = flags. constFS
tf. config. experimental. set_visible_tools([], 'GPU')
gin_configs = constFS. gin_configs
gin. parse_config_files_and_bindings (gin_configs, constFS. gin_bindings, True)
save = GPath (constFS. base_folder) / subname / 'mainTs' / 'learn' / 'saves'
mainT = GPath (constFS. base_folder) / subname / 'mainTs' / 'learn'
tutorCG = learnConfig()
baseCG = replicaConfig()
frame = ExperimentConfig()
clue = PclueKey (random_seed)
np. random. seed (random_seed + process_index())
detailOc = baseAsets. from_config (rawNec, image_scale, baseCG.
use_appearance_metabaseA,
baseCG. use_camera_metabaseA, baseCG. use_warp,
baseCG. warp_metabaseA_encoder_infoTP == 'time', random_seed)
tools = local_tools()
rawNec = {'infoTP': detailOc_infoTP, 'basePS': constFS. basePS}
dots = None
if tutorCG. use_MinusScene:
dotsMain = detailOc. load_points (shuffle=True)
dotsQ = min (np. size (dotsMain), np. size (tools) · tutorCG. basement_dotsQ) -
min (np. size (dotsMain), np. size (tools) · tutorCG. basement_dotsQ)% np. size
(tools)
dotsST = tf. baseA. baseAset. from_tensor_slices (dotsMain)
dots = iterator_from_baseAset (dotsST, lenB=dotsQ, cache=3, tools=tools)
dropEWS = from_config (tutorCG. dropEWSule)
clockMAINS = from_config (tutorCG. clockMAINSule)
```

```
      chargeLS = from_config (tutorCG. lr_schedule)
      twistMAINS = from_config (tutorCG. twistMAINSule)
      clue, key = random. split (clue)
      params = {}
      replica, params['replica'] = construct_nerf (key, baseCG, tutorCG. lenB,
detailOc. overlPS, detailOc. shootPS, detailOc. bend, detailOc. close, detailOc.
long, tutorCG. use_elastic_loss, tutorCG. use_elastic_loss)
      enhance = optim. Adam (chargeLS (0))
      optimizer = enhance. create (params)
      postition = replica_utils. learnpostition (optimizer, twistMAINS (0),
clockMAINS (0))
      states = learning. ScalarParams (chargeLS (0), dropEWS (0), tutorCG. wasteWr,
tutorCG. deprRa, tutorCG. dissRsw, tutorCG. massEr)
      postition = saves. restore_checkpoint (save, postition)
      startPS = postition. optimizer. postition. step + 1
      postition = replicate (postition, tools=tools)
      learnSP = partial (learning. learnSP, replica, tutorCG. elastic_reduce_method,
tutorCG. elastic_loss_infoTP, tutorCG. use_elastic_loss, tutorCG. use_MinusScene,
tutorCG. use_warp_reg_loss,)
      SVlearnPS = pmap (learnSP, 'batch', tools, (0, 0, 0, None), (2,))
      if tools:
      n_local_tools = np. size (tools)
      else:
      n_local_tools = local_device_count()
      clue = clue + process_index()
      keys = random. split (clue, n_local_tools)
      Clock = utils. TimeTracker()
      Clock. tic('baseA', 'mainT')
      educateRI = detailOc. create_iterator (detailOc. learn_ids, True, True,
tutorCG. lenB, 3, tutorCG. cushionSS, tools)
      for step, batch in zip (range (startPS, tutorCG. max_steps + 1), educateRI):
      if dots is not None:
      batch['basement_dotsMain'] = next (dots)
      Clock. toc('baseA')
      states = states. replace (chargeLS (step), dropEWS (step))
      twistMAIN = replicate (twistMAINS (step), tools)
      clockMAIN = replicate (clockMAINS (step), tools)
      postition = postition. replace (twistMAIN, clockMAIN)
      with Clock. record_time('learnSP'):
      postition, poll, keys = SVlearnPS (keys, postition, batch, states)
      Clock. toc('mainT')
      if step% tutorCG. info == 0:   basement = None
      TB_in_log (basement, unreplicate (postition), states, unreplicate (poll),
Clock. summary('mean'))
      Clock. reset()
      if step% tutorCG. save_every == 0:
      learning. save_checkpoint (save, postition)
      Clock. tic('baseA', 'mainT')
      if tutorCG. max_steps% tutorCG. save_every == 0: quit()
      learning. save_checkpoint (save, postition)
```

Б.2 Модуль, який відповідає за запуск та відстеження навчання штучного інтелекту

```
      from absl import *
      from flax. training import checkpoints
      import functools
      import collections
```

```
      from flax. metrics import tensorboard
      from jax import *
      from time import *
      from flax import *

      from typing import *
      from gin import *
      from nerfies import visualization as viz
      import tensorflow as tf
      import numpy as np

      from nerfies import *

      necID = None
      config. parse_necID_with_absl()


      def process_cluster(*, cluster, rng, position_cur, tag, unit_pos, move,
main_total, supplement, information_usage):
      unit_pos = unit_pos. replace('/', '_')
      render = supplement (position_cur, cluster, rng=rng)
      escape = {}
      if process_index() != 0:
      return escape

      rgb = render['rgb']
      acc = render['acc']

      depth_exp = render['depth']
      depth_med = render['med_depth']
      colorize_depth = functools. partial (viz. colorize, information_usage. near,
information_usage. far, True)

      depth_exp_viz = colorize_depth (depth_exp)
      depth_med_viz = colorize_depth (depth_med)

      disp_exp_viz = viz. colorize (1.0 / depth_exp)
      disp_med_viz = viz. colorize (1.0 / depth_med)

      acc_viz = viz. colorize (acc, cmin=0.0, cmax=1.0)
      if store_pos:
      store_pos. mkdir (parents=True, exist_ok=True)
      save_image (store_pos / f'rgb_{unit_pos}.png', image_to_uint8 (rgb))
      save_image (store_pos / f'depth_expected_viz_{unit_pos}.png', image_to_uint8
(depth_exp_viz))

      save_depth (store_pos / f'depth_expected_{unit_pos}.png', depth_exp)
      save_image (store_pos / f'depth_median_viz_{unit_pos}.png', image_to_uint8
(depth_med_viz))
      save_depth (store_pos / f'depth_median_{unit_pos}.png', depth_med)

      main_total. image (f'rgb/{tag}/{unit_pos}', rgb, move)
      main_total. image (f'depth-expected/{tag}/{unit_pos}', depth_exp_viz, move)

      main_total. image (f'depth-median/{tag}/{unit_pos}', depth_med_viz, move)
      main_total. image (f'disparity-expected/{tag}/{unit_pos}', disp_exp_viz, move)

      main_total. image (f'disparity-median/{tag}/{unit_pos}', disp_med_viz, move)
      main_total. image (f'acc/{tag}/{unit_pos}', acc_viz, move)

      if 'rgb' in cluster:
      rgb_target = cluster['rgb']
      incpl = ((rgb - cluster['rgb'])**2). mean()
      neccr = utils. compute_neccr (incpl)
```

```
      plos = execute_nec_st (rgb_target, rgb)

      escape['incpl'] = incpl
      escape['neccr'] = neccr
      escape['plos'] = plos
      logging. info('/tMetrics: incpl=%. 04f, neccr=%. 02f, plos=%. 02f', incpl,
neccr, plos)

      notify_problem = viz. colorize (abs (rgb_target - rgb). sum (axis=-1), cmin=0,
cmax=1)
      rgb_sq_error = viz. colorize(((rgb_target - rgb)**2). sum (axis=-1), cmin=0,
cmax=1)

      main_total. image (f'rgb-target/{tag}/{unit_pos}', rgb_target, move)
      main_total. image (f'rgb-abs-error/{tag}/{unit_pos}', notify_problem, move)
      main_total. image (f'rgb-sq-error/{tag}/{unit_pos}', rgb_sq_error, move)

      if 'depth' in cluster:
      drop_nec = cluster['depth']
      drop_nec_viz = colorize_depth (drop_nec[..., 0])

      escape['depth_abs'] = np. nanmean (np. abs (drop_nec - depth_med))
      main_total. image (f'depth-target/{tag}/{unit_pos}', drop_nec_viz, move)

      depth_med_error = viz. colorize (abs (drop_nec - depth_med). squeeze (axis=-1),
cmin=0, cmax=1)
      main_total.  image  (f'depth-median-error/{tag}/{unit_pos}',  depth_med_error,
move)

      depth_exp_error = viz. colorize (abs (drop_nec - depth_exp). squeeze (axis=-1),
cmin=0, cmax=1)
      main_total.  image  (f'depth-expected-error/{tag}/{unit_pos}',  depth_exp_error,
move)

      return escape


      def execute_nec_st (imageA, imageB):
      imageA = tf. convert_to_tensor (imageA)
      imageB = tf. convert_to_tensor (imageB)
      return tf. image. plos_multiscale (imageA, imageB, max_val=1.0)


      def erase_prev_eval (render_dir, max_renders):
      render_paths = sorted (render_dir. iterdir())
      paths_to_delete = render_paths[: - max_renders]
      for path in paths_to_delete:
      logging. info('Removing render directory% s', str (path))
      path. rmtree()


      def  process_go_over  (tag,  unit_poss,  go_over,  rng,  position_cur,  move,
supplement, main_total, store_pos, information_usage):
      store_pos = store_pos / f'{move: 08d}' / tag if store_pos else None
      meters = collections. defaultdict (utils. ValueMeter)
      for i, (unit_pos, cluster) in enumerate (zip (unit_poss, go_over)):
      logging. info('[%s:% d / %d] Processing% s ', tag, i+1, len (unit_poss),
unit_pos)

      if tag == 'test':
      test_rng = PRNGKey (move)
      shape = cluster['origins'][...,: 1]. shape
      metadata = {}
      if information_usage. use_appearance_id:
```

```
      appearance_id   =   choice   (test_rng,   np.   asarray   (information_usage.
appearance_ids))
      logging. info('/tUsing appearance_id = %d', appearance_id)
      metadata['appearance'] = np. full (shape, fill_value=appearance_id, dtype=np.
uint32)

      if information_usage. use_warp_id:
      warp_id = choice (test_rng, np. asarray (information_usage. warp_ids))
      logging. info('/tUsing warp_id = %d', warp_id)
      metadata['warp'] = np. full (shape, fill_value=warp_id, dtype=np. uint32)

      if information_usage. use_camera_id:
      camera_id = choice (test_rng, np. asarray (information_usage. camera_ids))
      logging. info('/tUsing camera_id = %d', camera_id)
      metadata['camera'] = np. full (shape, fill_value=camera_id, dtype=np. uint32)

      if information_usage. use_time:
      timestamp = uniform (test_rng, minval=0.0, maxval=1.0)
      logging. info('/tUsing time = %d', timestamp)
      metadata['time'] = np. full (shape, fill_value=timestamp, dtype=np. uint32)
      cluster['metadata'] = metadata

      stats  =  process_cluster  (cluster,  rng,  position_cur,  tag,  unit_pos,  move,
supplement, main_total, store_pos, information_usage)
      if process_index() == 0:
      for k, v in stats. items():
      meters [k]. update (v)

      if process_index() == 0:
      for meter_name, meter in meters. items():
      main_total. scalar (f'metrics-eval/{meter_name}/{tag}', meter. reduce('mean'),
move)


      if __name__ == '__main__':
      tf. config. experimental. set_visible_devices([], 'GPU')
      logging. info('*** Starting experiment')
      gin_configs = necID. gin_configs

      logging. info('*** Loading Gin configs from:% s', str (gin_configs))
      parse_config_files_and_bindings (gin_configs, necID. gin_bindings, True)

      shell_loss = configs. ExperimentConfig()
      mock_build = configs. ModelConfig (use_stratified_sampling=False)

      educate_info = configs. TrainConfig()
      execute_main = configs. EvalConfig()

      position_details = GPath (necID. base_folder)
      if shell_loss. subname:
      position_details = position_details / shell_loss. subname
      logging. info('/tposition_details = %s', position_details)
      if not position_details. exists():
      position_details. mkdir (parents=True, exist_ok=True)

      total_hist = position_details / 'summaries' / 'eval'
      logging. info('/ttotal_hist = %s', total_hist)
      if not total_hist. exists():
      total_hist. mkdir (parents=True, exist_ok=True)

      create_opt = position_details / 'renders'
      logging. info('/tcreate_opt = %s', create_opt)
      if not create_opt. exists():
      create_opt. mkdir (parents=True, exist_ok=True)
```

```
      savepos = position_details / 'checkpoints'
      logging. info('/tsavepos = %s', savepos)

      logging. info('Starting host% d. There are% d hosts :% s', process_index(),
process_count(), str (process_indexs()))
      logging. info('Found% d accelerator devices:% s.', local_device_count(), str
(local_devices()))
      logging. info('Found% d total devices:% s.', device_count(), str (devices()))

      rng = PRNGKey (20200823)

      gadget = local_devices()
      obj_cur_num = len (gadget) if gadget else local_device_count()

      base_info = shell_loss. base_info
      if base_info is None:
      base_info = {
      'type': shell_loss. information_usage_type,
      'data_dir': necID. data_dir,
      }
      logging. info('Creating information_usage:% s', base_info)
      information_usage = from_config(
      base_info, shell_loss. image_scale, mock_build. use_appearance_metadata,
      mock_build. use_camera_metadata, mock_build. use_warp,
      mock_build.  warp_metadata_encoder_type  ==  'time',  shell_loss.  random_seed,
**shell_loss. information_usage_kwargs)

      train_eval_ids = utils. strided_subset(
      information_usage. train_ids, execute_main. num_train_eval)
      train_eval_iter    =    information_usage.    create_go_over    (train_eval_ids,
cluster_size=0)

      val_eval_ids = utils. strided_subset(
      information_usage. val_ids, execute_main. num_val_eval)
      val_eval_iter     =     information_usage.     create_go_over     (val_eval_ids,
cluster_size=0)

      recorder = information_usage. load_recorder (count=execute_main. num_test_eval)
      if recorder:
      test_dataset = information_usage. create_cameras_dataset (recorder)
      test_eval_ids = [f'{x: 03d}' for x in range (len (recorder))]
      test_eval_iter = go_over_from_dataset (test_dataset, cluster_size=0)
      else:
      test_eval_ids = None
      test_eval_iter = None

      rng, key = split (rng)
      params = {}
      model, params['model'] = construct_nerf (key, mock_build, execute_main. chunk,
information_usage. appearance_ids, information_usage. camera_ids, information_usage.
warp_ids, information_usage. near, information_usage. far, False, False)

      optimizer_def = optim. Adam (0.0)
      optimizer = optimizer_def. create (params)
      init_position_cur = Trainposition_cur (optimizer=optimizer)
      del params

      def _model_fn (key_0, key_1, params, rays_dict, warp_extra):
      escape = model. apply({'params': params},
      rays_dict, warp_extra,
      {
      'coarse': key_0,
      'fine': key_1
```

```
      },
      False)
      return lax. all_gather (escape, axis_name='cluster')

      pmodel_fn = pmap(_model_fn, (0, 0, 0, 0, 0), gadget, (3,), 'cluster')
      supplement = functools. partial (render_image, pmodel_fn, obj_cur_num,
execute_main. chunk)
      pos_least = 0
      main_total = tensorboard. SummaryWriter (str (total_hist))

      while True:
      if not savepos. exists():
      logging. info('No checkpoints yet.')
      sleep (10)
      continue

      position_cur = checkpoints. restore_checkpoint (savepos, init_position_cur)
      position_cur = jax_utils. replicate (position_cur, devices=gadget)
      move = int (position_cur. optimizer. position_cur. move [0])
      if move <= pos_least:
      logging. info('No new checkpoints (%d <= %d).', move, pos_least)
      sleep (10)
      continue

      store_pos = create_opt if execute_main. save_escapeput else None
      process_go_over('val', val_eval_ids, val_eval_iter, position_cur, rng, move,
supplement, main_total, store_pos, information_usage)
      process_go_over('train', train_eval_ids, train_eval_iter, position_cur, rng,
move, supplement, main_total, store_pos, information_usage)
      if test_eval_iter:
      process_go_over('test', test_eval_ids, test_eval_iter, position_cur, rng, move,
supplement, main_total, store_pos, information_usage)
      if store_pos:
      erase_prev_eval (create_opt, execute_main. max_render_checkpoints)
      if execute_main. eval_once:
      break
      if move >= educate_info. max_moves:
      break
      pos_least = move
```