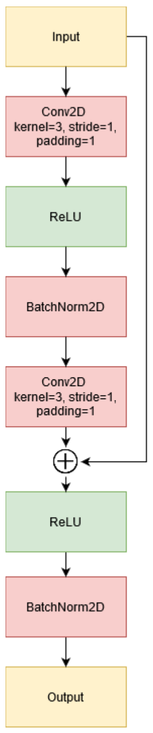
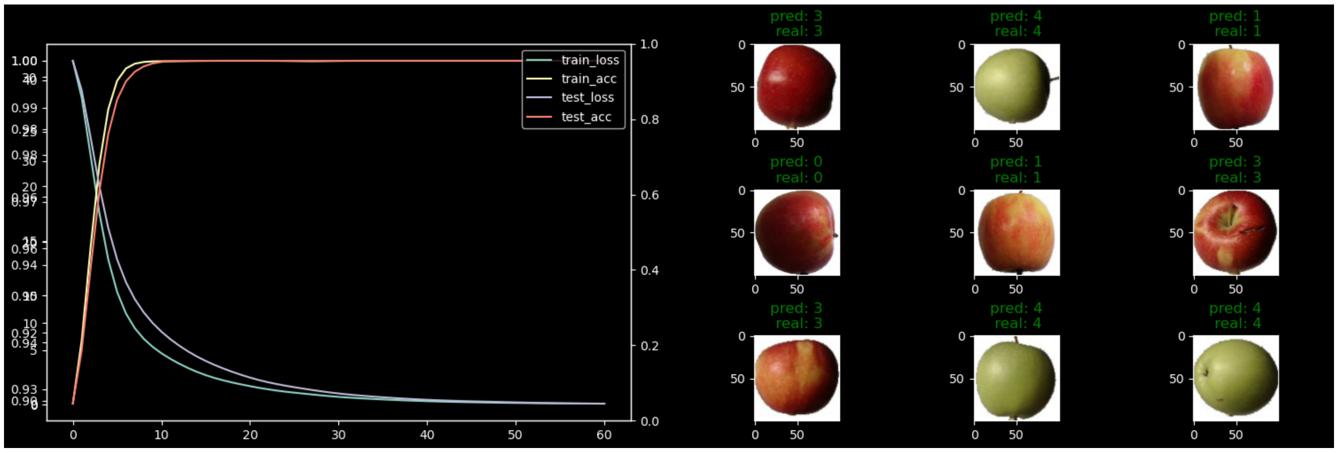
ResNet, DenseNet, InceptionNet

1) Implement ResNet according to scheme:


Input
Conv2D kernel=3, stride=1, padding=1
ReLU
BatchNorm2D
Conv2D kernel=3, stride=1, padding=1
ReLU
BatchNorm2D
Output

Performance:



2) Implement DenseNet according to scheme:

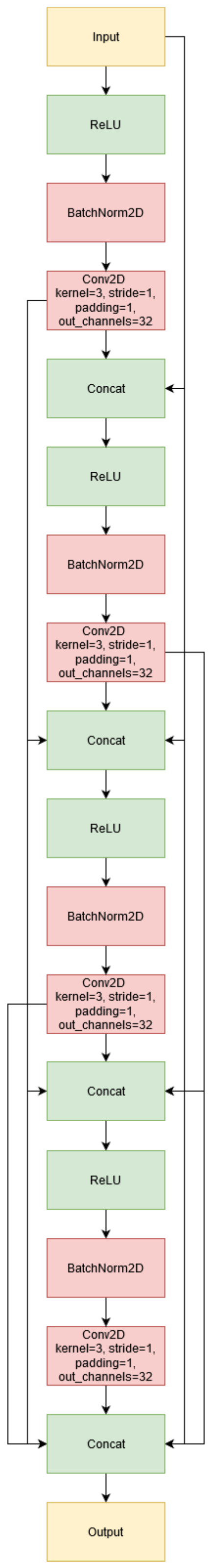

Input
ReLU
BatchNorm2D
Conv2D kernel=3, stride=1, padding=1, out_channels=32
Concat
ReLU
BatchNorm2D
Conv2D kernel=3, stride=1, padding=1, out_channels=32
Concat
ReLU
BatchNorm2D
Conv2D kernel=3, stride=1, padding=1, out_channels=32
Concat
ReLU
BatchNorm2D
Conv2D kernel=3, stride=1, padding=1, out_channels=32
Concat
Output

Figure 1: DenseNet block

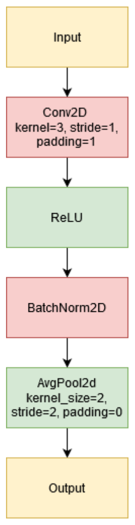
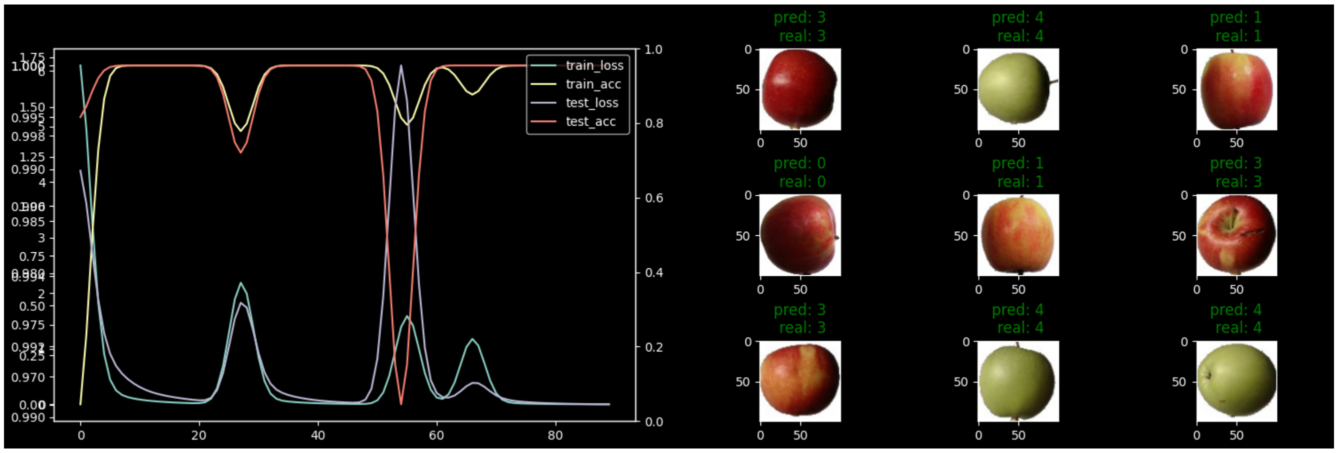

Input
Conv2D kernel=3, stride=1, padding=1
ReLU
BatchNorm2D
AvgPool2d kernel_size=2, stride=2, padding=0
Output

Figure 2: Transition layer

Performance:



3) Implement InceptionNet according to scheme:

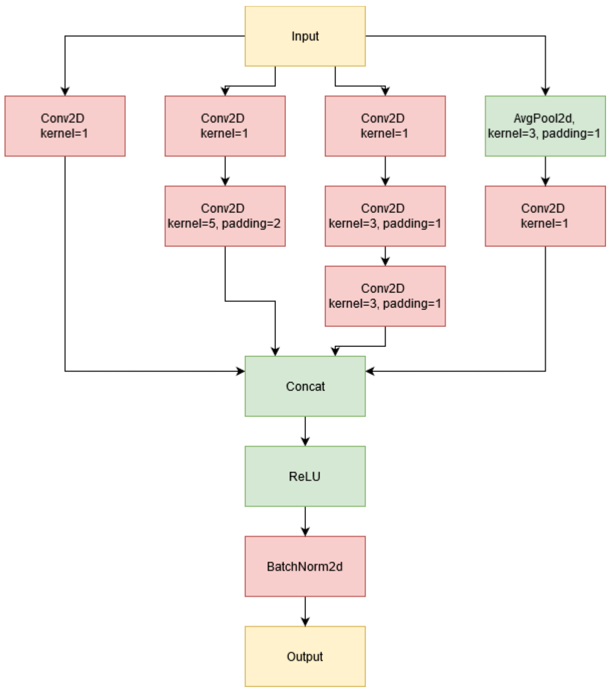

Input
Conv2D kernel=1
Conv2D kernel=1
Conv2D kernel=1
AvgPool2d, kernel=3, padding=1
Conv2D kernel=5, padding=2
Conv2D kernel=3, padding=1
Conv2D kernel=1
Conv2D kernel=3, padding=1
Concat
ReLU
BatchNorm2d
Output

Figure 3: InceptionNet block

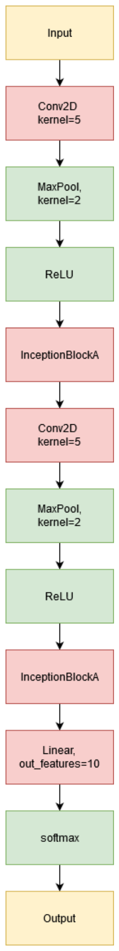
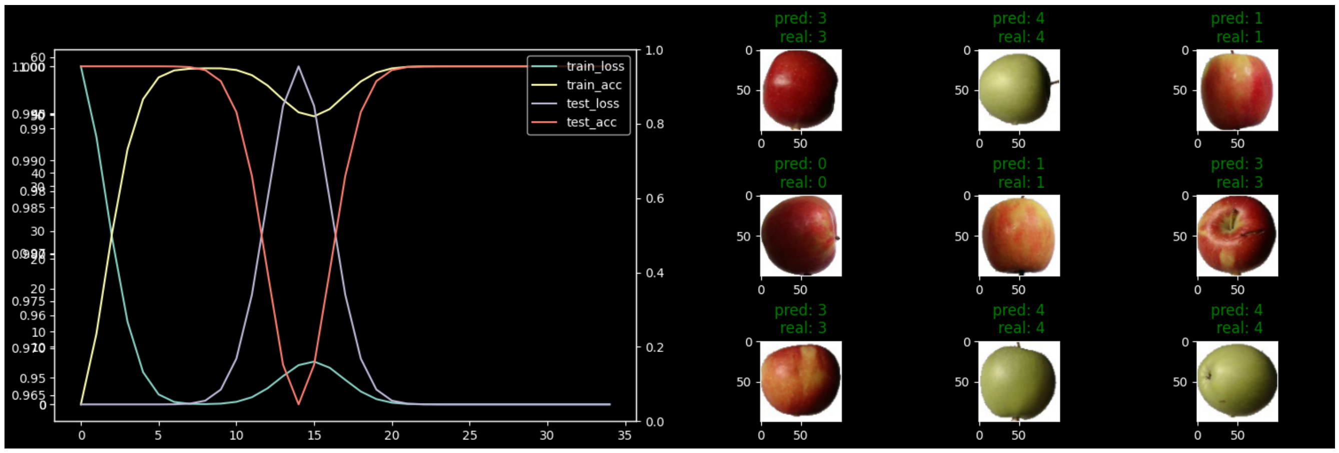


Figure 4: InceptionNet

Performance:

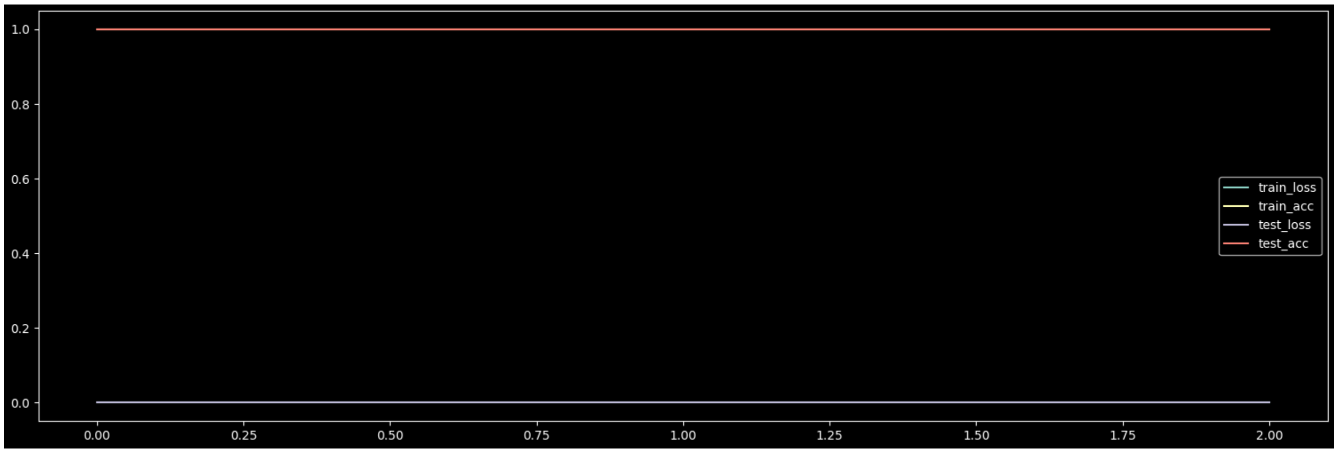


4) Implement own dataset based on NumPy memmap:

```
class DatasetFlickrImageNumpyMmap(torch.utils.data.Dataset):
    def __init__(self, root: str = 'data', force: bool = False):
        super().__init__()
        kaggle.api.authenticate()
        image_dir_name = 'flickr30k_images'
        result_file_name = 'results.csv'
        dataset_file_name = 'data.npy'
        dataset_path = Path(Path(__file__).parent, root, image_dir_name)

        dataset_file_path = Path(dataset_path, dataset_file_name)
        image_dir_path = Path(dataset_path, image_dir_name)
        result_file_path = Path(dataset_path, result_file_name)
        metadata_file_path = Path(dataset_path, 'metadata_mmap.json')

        self.y = []
        self.image_height = 256
        self.image_width = self.image_height
        self.bytes_per_value = 64 / 8

        if force or not dataset_path.exists():
            kaggle.api.dataset_download_files('hsankesara/flickr-image-dataset', path=root, quiet=False, unzip=True,
                                             force=force)

            # Removing duplicated data
            rmtree(Path(image_dir_path, image_dir_name), ignore_errors=True)
            Path(image_dir_path, result_file_name).unlink(missing_ok=True)

        if force or not dataset_file_path.exists() or not metadata_file_path.exists():
            with open(result_file_path, mode='r', encoding='utf-8') as f:
                reader = csv.reader(f, delimiter='|')
                # Skipping header
                next(reader)

                results = tuple(reader)
                get_filename = itemgetter(0)
                filenames = set(map(get_filename, results))

            self.data_length = len(filenames)
            self.dataset_shape = (self.data_length, 3, self.image_height, self.image_width)

            self.x = open_memmap(
                str(dataset_file_path), mode='w+', dtype='float64', shape=self.dataset_shape
            )

            for idx, filename in enumerate(filenames):
                image_file_path = Path(image_dir_path, filename)
                image = Image.open(image_file_path)
                width, height = image.size  # Get dimensions
                new_size = min(width, height)

                left = int((width - new_size) / 2)
                top = int((height - new_size) / 2)
                right = int((width + new_size) / 2)
                bottom = int((height + new_size) / 2)
                # Crop the center of the image
                image = image.crop((left, top, right, bottom))

                image = image.resize((self.image_height, self.image_width), resample=Resampling.LANCZOS)
                image = np.asarray(image) / 255.0
                # Converting HxWxC to CxHxW
                image = np.transpose(image, (2, 0, 1))
                self.x[idx, :, :] = image[:, :]

                # TODO: what to use???
                self.y.append(0)

            self.x.flush()

            with open(metadata_file_path, 'w') as f:
                metadata = {
                    'shape': self.dataset_shape,
                    'labels': self.y
                }
                json.dump(metadata, f)
        else:
            with open(metadata_file_path) as f:
                metadata = json.load(f)

            self.dataset_shape = metadata['shape']
            self.data_length = self.dataset_shape[0]
            self.y = metadata['labels']

        self.x = open_memmap(
            str(dataset_file_path), mode='r', dtype='float64', shape=self.dataset_shape
        )
        self.y = F.one_hot(torch.LongTensor(self.y))

    def __len__(self):
        if MAX_LEN:
            return MAX_LEN

        return self.data_length

    def __getitem__(self, idx):
        x = torch.from_numpy(np.copy(self.x[idx]))

        return x, self.y[idx]
```

Jautājums: Varbūt būtu labāk jau no paša sākuma pārveidot datus tensorā un saglabāt?

Performance:



Accuracy is 1 and loss is 0 because I had no labels, so they always match “0”.
Takeaway: my Network had so too many channels for batch size of 64 (for 16GB RAM), I thought it was memmap which loaded complete dataset everytime, but debugging showed that it was model.forward(x). So I decreased batch size to 32 and it stopped swapping.

5) Implement own dataset using filesystem

```python
class DatasetFlickrImage(torch.utils.data.Dataset):
    def __init__(self, root: str = 'data', force: bool = False, transform=None, target_transform=None):
        super().__init__()
        kaggle.api.authenticate()
        image_dir_name = 'flickr30k_images'
        result_file_name = 'results.csv'
        dataset_path = Path(Path(__file__).parent, root, image_dir_name)

        self.image_dir_path = Path(dataset_path, image_dir_name)
        result_file_path = Path(dataset_path, result_file_name)
        metadata_file_path = Path(dataset_path, 'metadata_file.json')

        self.transform = transform
        self.y = []
        self.image_height = 256
        self.image_width = self.image_height
        self.bytes_per_value = 64 / 8

        if force or not dataset_path.exists():
            kaggle.api.dataset_download_files('hsankesara/flickr-image-dataset', path=root, quiet=False, unzip=True,
                                              force=force)

            # Removing duplicated data
            rmtree(Path(self.image_dir_path, image_dir_name), ignore_errors=True)
            Path(self.image_dir_path, result_file_name).unlink(missing_ok=True)

        if force or not metadata_file_path.exists():
            with open(result_file_path, mode='r', encoding='utf-8') as f:
                reader = csv.reader(f, delimiter='|')
                # Skipping header
                next(reader)

                results = tuple(reader)
                get_filename = itemgetter(0)
                filenames = set(map(get_filename, results))
                self.data_length = len(filenames)

                with open(metadata_file_path, 'w') as fm:
                    # TODO: what to use???
                    metadata = {filename: 0 for filename in filenames}
                    json.dump(metadata, fm)

        with open(metadata_file_path) as f:
            metadata = json.load(f)

        self.x = tuple(metadata.keys())
        self.y = tuple(metadata.values())
        self.data_length = len(self.y)

        # How do I know class count when passing transformation? idk...
        if target_transform:
            self.y = target_transform(self.y)

        # So I transform it manually inside
        self.y = F.one_hot(torch.LongTensor(self.y))
```

```
def __len__(self):
    if MAX_LEN:
        return MAX_LEN

    return self.data_length

def __getitem__(self, idx):
    image_path = Path(self.image_dir_path, self.x[idx])
    x = read_image(str(image_path))
    y = self.y[idx]

    if self.transform:
        x = self.transform(x)

    x = x / 255.0

    return x, y
```

6) Implement own dataset using Zarr:

```
class DatasetFlickrImageZarr(torch.utils.data.Dataset):
    def __init__(self, root: str = 'data', force: bool = False, chunks=None):
        super().__init__()
        kaggle.api.authenticate()
        image_dir_name = 'flickr30k_images'
        result_file_name = 'results.csv'
        dataset_file_name = 'data.zarr'
        dataset_path = Path(Path(__file__).parent, root, image_dir_name)

        dataset_file_path = Path(dataset_path, dataset_file_name)
        image_dir_path = Path(dataset_path, image_dir_name)
        result_file_path = Path(dataset_path, result_file_name)

        self.y = []
        self.image_height = 256
        self.image_width = self.image_height
        self.bytes_per_value = 64 / 8

        if force or not dataset_path.exists():
            kaggle.api.dataset_download_files('hsankesara/flickr-image-dataset', path=root, quiet=False, unzip=True,
                                              force=force)

            # Removing duplicated data
            rmtree(Path(image_dir_path, image_dir_name), ignore_errors=True)
            Path(image_dir_path, result_file_name).unlink(missing_ok=True)

        if force or not dataset_file_path.exists():
            with open(result_file_path, mode='r', encoding='utf-8') as f:
                reader = csv.reader(f, delimiter='|')
                # Skipping header
                next(reader)

                results = tuple(reader)
                get_filename = itemgetter(0)
                filenames = set(map(get_filename, results))

            self.data_length = len(filenames)
            self.dataset_shape = (self.data_length, 3, self.image_height, self.image_width)

            self.dataset = zarr.open(str(dataset_file_path), mode='w')
            self.x = self.dataset.zeros('samples', shape=self.dataset_shape, chunks=chunks, dtype='float64')
            self.y = self.dataset.zeros('labels', dtype='int64', shape=self.data_length if USE_CUDA else MAX_LEN)

            y = []

            for idx, filename in enumerate(filenames):
                image_file_path = Path(image_dir_path, filename)
                image = Image.open(image_file_path)
                width, height = image.size  # Get dimensions
                new_size = min(width, height)

                left = int((width - new_size) / 2)
                top = int((height - new_size) / 2)
                right = int((width + new_size) / 2)
                bottom = int((height + new_size) / 2)
            # Crop the center of the image
            image = image.crop((left, top, right, bottom))

            image = image.resize((self.image_height, self.image_width), resample=Resampling.LANCZOS)
            image = np.asarray(image) / 255.0
            # Converting HxWxC to CxHxW
            image = np.transpose(image, (2, 0, 1))
            self.x[idx, :, :] = image

            # TODO: what to use???
            y.append(0)

            # WORKAROUND: save time
            if idx >= MAX_LEN - 1:
                break

        self.y[:] = torch.tensor(y, dtype=torch.long)

    self.root = zarr.open(str(dataset_file_path), mode='r')
    self.x = self.root['samples']
    self.y = F.one_hot(self.root['labels'])
    self.data_length = len(self.y)

def __len__(self):
    if MAX_LEN:
        return MAX_LEN

    return self.data_length

def __getitem__(self, idx):
    x = torch.from_numpy(np.copy(self.x[idx]))

    return x, self.y[idx]
```

7) Implement own dataset using HDF5

TODO

8) Implement own Dataset using CuPy
TODO