Andri Setiawan Susanto, 1002849
Zwe Wint Naing, 1002791

The Small Project PASCAL VOC

1. Dataset and Dataloader

This dataset consists of 20 categories. They are aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, diningtable, dog, horse, motorbike, person, pottedplant, sheep, sofa, train and tvmonitor.

For Dataloader, there are 3 functions used to get the multilabel for each images in either 'train', 'trainval' and 'val'. _annotation_file_from_img is used to get the xml path in the Annotation folder. Load_annotation is used to get the image label from the xml file specify by the path given in the _annotation_file_from_img. The label can be found in the "name" tag in xml file. Lastly, get_data_multilabel will take in type_of_data either 'train', 'trainval' and 'val' in order to get the relevant images xml file and return two dataframes that consist of the path of the images and their relevant labels. If the image belong to certain label, it will be given "1"; otherwise, the image does not belong to a certain label and thus will be given "0".

```
def _annotation_file_from_img(self, img_name):

    path = os.path.join(self.ann_dir, img_name) + '.xml'

    return path

def load_annotation(self, img_filename):

    xml = ""
    with open(self._annotation_file_from_img(img_filename)) as file:
        xml = file.readlines()
    xml = ''.join([line.strip("\t") for line in xml])
    return BeautifulSoup(xml, "xml")

def get_data_multilabel(self, type_of_data):

    filename = os.path.join(self.set_dir, type_of_data + ".txt")
    category_list = self.list_image_sets()
    df = pd.read_csv(filename, sep=' ', header=None, names=["filename"])

    for category_name in category_list:
        df[category_name] = 0
    for data in df.itertuples():
        ind, fname = data[0], data[1]
        annotation = self.load_annotation(fname)
        objects = annotation.findAll("object")
        for object in objects:
            object_names = object.findChildren("name")
            for name_tag in object_names:
                tag_name = name_tag.contents[0]
                tag_name = str(tag_name)
                if tag_name in category_list:
                    df.at[ind, tag_name] = 1
    df["filename"] = self.img_dir + df['filename'] + ".jpg"
    df_filename = df["filename"]
    df_label = df.drop(['filename'], axis=1)
    return df_filename, df_label.to_numpy()
```

The constructor used is shown below.

```
def __init__(self, root_dir, dataset, transform=None):
    """
    Summary:
        Init the class with root dir
    Args:
        root_dir (string): path to your voc dataset
    """
    self.root_dir = root_dir
    self.img_dir =  os.path.join(root_dir, 'JPEGImages/')
    self.ann_dir = os.path.join(root_dir, 'Annotations')
    self.set_dir = os.path.join(root_dir, 'ImageSets', 'Main')
    self.cache_dir = os.path.join(root_dir, 'csvs')
    self.transform = transform
    self.image_paths, self.labels = self.get_data_multilabel(dataset)
    if not os.path.exists(self.cache_dir):
        os.makedirs(self.cache_dir)
```

To improve the efficiency of the code, __len__ and __getitem__ are implemented so that only the required images are load when needed.

Andri Setiawan Susanto, 1002849
Zwe Wint Naing, 1002791

__getitem__ function will return the numpy array of image, the corresponding label and the path where this image resides.

```
def __len__(self):
    return len(self.image_paths)

def __getitem__(self, idx):
    # apply transforms
    image_path = self.image_paths[idx]
    image = self.get_image(image_path)
    if self.transform is not None:
        image = self.transform(image)
    label = (self.labels[idx])
    label =torch.from_numpy(label)
    return [image, label, image_path]
```

2. Model Used, Loss Learning Rate Schemes and Results Obtain

Model

In this project, ResNet34 and ResNet18 are used as the model because people widely used these models for multi-label classification problem.

Hyperparameters

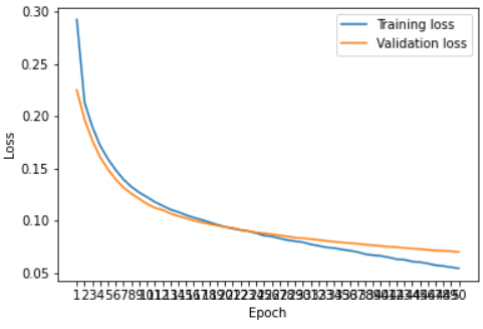
```
batch_size = 32
maxnumepochs = 50
learning_rate = 1e-3
num_categories = 5
num_imgs = 5
transform = transforms.Compose([transforms.CenterCrop(200), transforms.ToTensor()])
torch.manual_seed(0)
init_folder = 'results1'
```

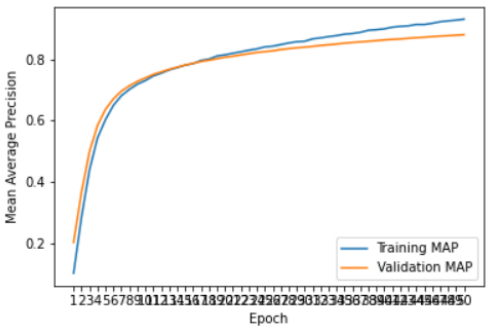
Torch.manual_seed(0) is used to ensure that the images used for every run is the same.

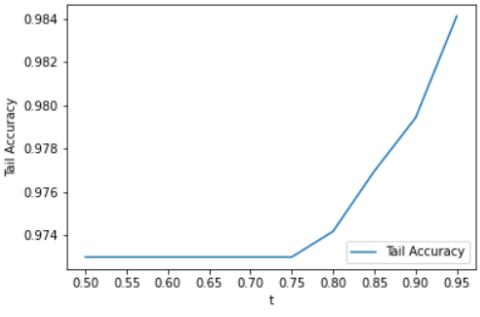
Loss Learning Rate

In this project, I have experimenting with 2 different loss learning rate. They are 1e-2 and 1e-3.

For ResNet34 with learning rate = 1e-3







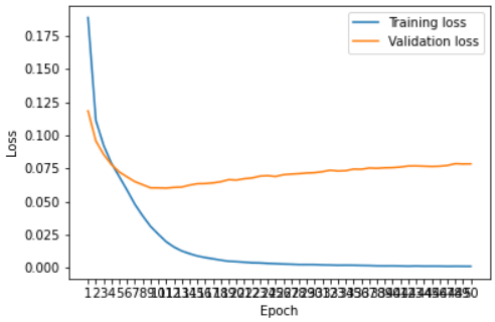
Average Precision:

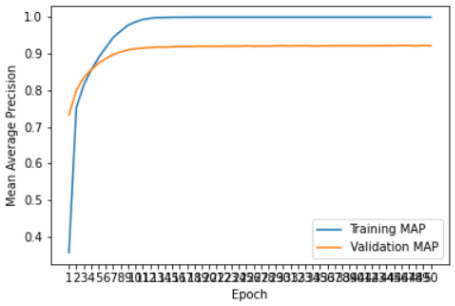
Andri Setiawan Susanto, 1002849
Zwe Wint Naing, 1002791

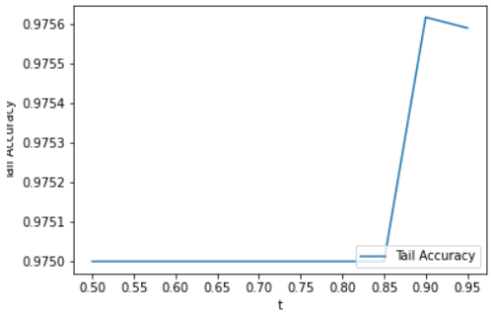
{0: 0.9792282047093291, 1: 0.8786391617654863, 2: 0.9669202673776496, 3: 0.9116640601862468, 4: 0.7414947556717909, 5: 0.9335916061635687, 6: 0.8367857283232921, 7: 0.9770307175132804, 8: 0.7843660611844124, 9: 0.8985020240085776, 10: 0.8453684820615659, 11: 0.9645802005373803, 12: 0.9534127095077383, 13: 0.9085720941857451, 14: 0.9593235202376442, 15: 0.6011183168614496, 16: 0.938316910507301, 17: 0.747056721373135, 18: 0.9646036002152231, 19: 0.8580873271441435}

Final mean average accuracy (MAP) = 79.98 %
There is an overfitting happen at around epoch 22.

For ResNet34 with learning rate = 1e-2



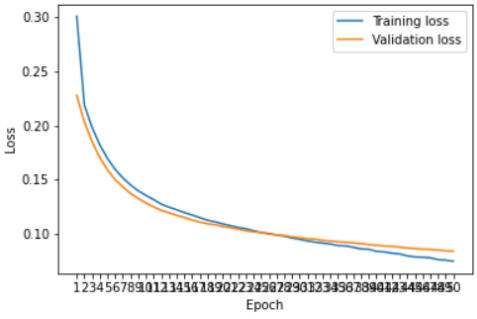


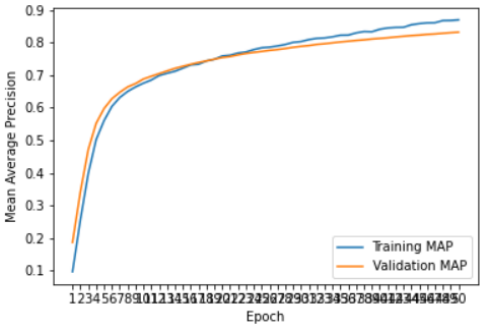


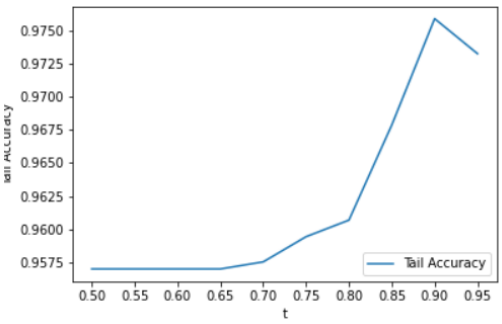
Average Precision:
{0: 0.9831002146393869, 1: 0.9142174938120852, 2: 0.9693620074914244, 3: 0.9297435821559111, 4: 0.8521906249470946, 5: 0.9547248875404556, 6: 0.8944524521952386, 7: 0.9806963876361284, 8: 0.8570195731987257, 9: 0.935210388642919, 10: 0.8769786821179784, 11: 0.9681911950707465, 12: 0.9649902371393393, 13: 0.9411662129086376, 14: 0.967029908853617, 15: 0.7761933390534287, 16: 0.9622817098303024, 17: 0.8458263220851484, 18: 0.966974785131696, 19: 0.8996111805316603}

Final mean average accuracy (MAP) = 79.43 %
There is an overfitting happen at around epoch 4.

For ResNet18 with learning rate = 1e-3







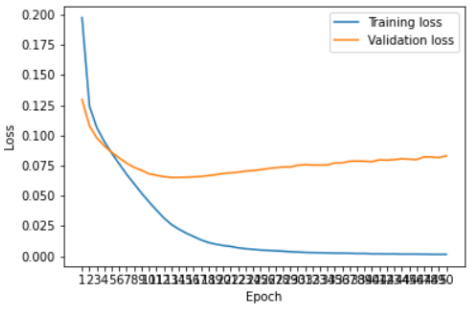
Average Precision:

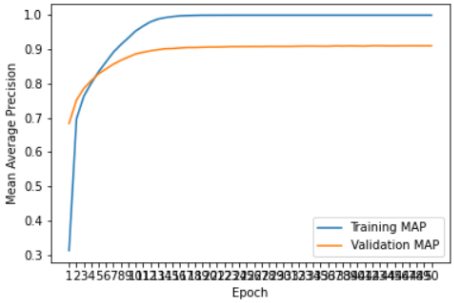
Andri Setiawan Susanto, 1002849
Zwe Wint Naing, 1002791

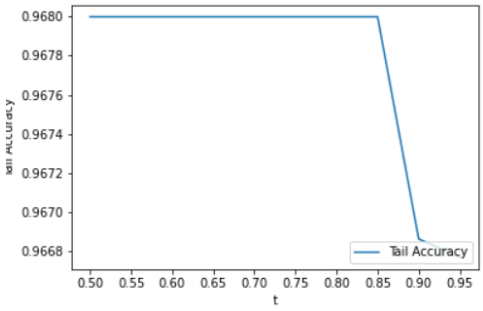
{0: 0.9730852005563075, 1: 0.8165779009374502, 2: 0.952128037211699, 3: 0.8925186093276948, 4: 0.6464312774781417, 5: 0.9151221017214807, 6: 0.8043068608126162, 7: 0.9623853604167171, 8: 0.7221027916504202, 9: 0.7435660137972866, 10: 0.7550513354476702, 11: 0.9406336878164174, 12: 0.8552678290803835, 13: 0.8658597141056189, 14: 0.9488081712923602, 15: 0.5443309210077434, 16: 0.8849259718562396, 17: 0.6767263644183411, 18: 0.9352284949663364, 19: 0.8110069872387793}

Final mean average accuracy (MAP) = 76.42 %
There is an overfitting happen at around epoch 26.

For ResNet18 with learning rate = 1e-2







Average Precision:
{0: 0.9759057495412112, 1: 0.9010349771501676, 2: 0.9604692888445684, 3: 0.9338578779233827, 4: 0.8277564710406713, 5: 0.9571213501717681, 6: 0.8844242188995047, 7: 0.9749899183301665, 8: 0.8678407646257802, 9: 0.8948542561907392, 10: 0.8486328707311338, 11: 0.9542921920800616, 12: 0.9395856661531564, 13: 0.9272123605235043, 14: 0.9652391624303329, 15: 0.7758927579112789, 16: 0.9504747813832073, 17: 0.8347622039188443, 18: 0.9526175148473875, 19: 0.8832725402274157}

Final mean average accuracy (MAP) = 76.48 %
There is an overfitting happen at around epoch 5.

Result

From all the different adjustment made to the hyperparameters, when the learning rate of 1e-2 is used instead of 1e-3, overfitting problem happen very early and there is a huge divergence between training loss and validation loss regardless of whether ResNet34 or ResNet18. This causes the classification to be inaccurate. In conclusion, ResNet34 with learning rate of 1e-3 give the best result based on the experiments above.

Loss Function (BCEWithLogitLoss)

As labels for each image is a multi-hot encoded tensor. The labels would have the same shape as the model output. BCEWithLogitLoss applies sigmoid internally.

Top 5 and Last 5 images for each 5 random classes

Andri Setiawan Susanto, 1002849
Zwe Wint Naing, 1002791

Top 5

Chair


1

2

3

4

5

Horse


1

2

3

4

5

Person


1

2

3

4

5

Sheep


1

2

3

4

5

Tvmonitor


1

2

3

4

5

Last 5

Chair


1

2

3

4

5

Horse


1

2

3

4

5

Person


1

2

3

4

5

Sheep

1

2

3

4

5

Tvmonitor

1

2

3

4

5