CS640 PROJECT

News Image Classification

Vedant Mahabaleshwarkar

U33181013

Kevin Rodrigues

U72564691

All the code, plots, models, and results can be found at the following link:
https://drive.google.com/open?id=1-fzGAtouQDRnDGlZ5At-4ghVvWKdIqoc

IDENTIFYING THE PROBLEM

In this project, we plan to create a system that can take an image as an input, and be able to classify the image in the following classes :

- Protest : 1 if the image contains people protesting, 0 otherwise
- Violence : A number between 0 and 1, higher the number more the violence
- Sign : 1 if the image contains people holding signs, 0 otherwise
- Photo : 1 if the image contains people holding photos, 0 otherwise
- Police : 1 if the image contains police, 0 otherwise
- Children : 1 if the image contains children, 0 otherwise
- Flag : 1 if the image contains flags, 0 otherwise
- Night : 1 if the image is taken during night-time, 0 otherwise
- Shouting : 1 if the image contains people shouting, 0 otherwise
- Group 20 : 1 if the image contains more than 20 people, 0 otherwise
- 1 if the image contains more than 100, 0 otherwise

USE-CASE

Such a system can be used by law enforcement agencies to react to protests faster. There are video cameras present on almost every street. After a set interval, a frame from the video feed could be run through this system to check for visual attributes. If protests, or violence is found, the system would trigger a notification to the agency so that law enforcement can quickly respond to the situation.

BACKGROUND SURVEY

This is a problem of image classification. Some of the possible solutions for implementing a solution for the same are as follows:

1. Feed Forward Network

a feed forward network can be implemented by flattening the image first and passing the flattened pixel values to the feed forward network. The goal of a feedforward network is to approximate some function f*. For example, for a classifier, y = f*(x) maps an input x to a category y. A feedforward network defines a mapping y = f(x; θ) and learns the value of the parameters θ that result in the best function approximation.(Reference)

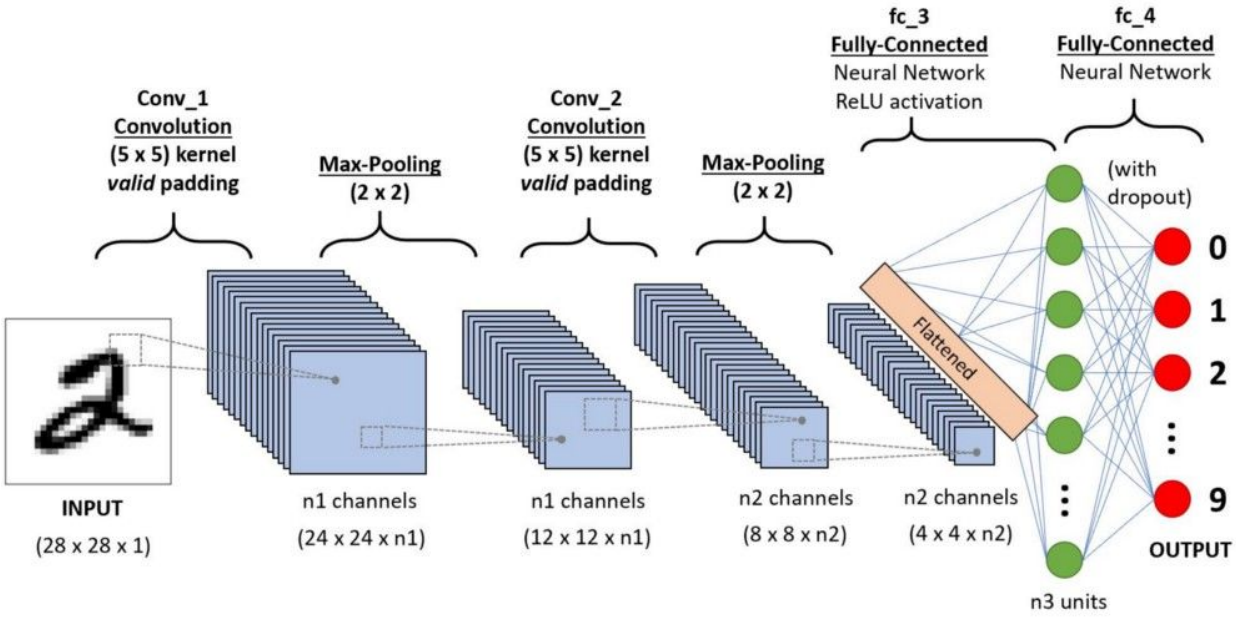
These models are called feedforward because information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y. There are no feedback connections in which outputs of the model are fed back into itself. When feedforward neural networks are extended to include feedback connections, they are called recurrent neural networks

2. Convolution Neural Network.

It is the most accepted solution for performing Image Classification is by implementing. A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

An example of architecture for a CNN is shown below :



WHY WE CHOSE CNN:

In cases of extremely basic binary images, the method of using just a flattened image (i.e image matrix) and feeding it to a Multi-Level Perceptron for classification purposes shows an average precision score while performing prediction of classes but would have little to no accuracy when it comes to complex images having pixel dependencies throughout.

A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

IMPLEMENTATION OPTIONS FOR BUILDING A CNN:

1. Implement a CNN from scratch.

    Using PyTorch, code can be written to add the layers of a CNN one by one to a self-defined model as follows :

```
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.drop_out = nn.Dropout()
        self.fc1 = nn.Linear(7 * 7 * 64, 1000)
        self.fc2 = nn.Linear(1000, 10)
```

    Given the short duration of this project, this approach seemed impractical, so we decided not to go ahead with this.

2. Transfer learning

    2.1 Using the architecture of implemented models

    There are many pre-trained models available that have implemented CNNs on public datasets. These models can be used to form the building block for a custom model by adding task specific layers before or after the model. These models already have weights assigned to them that were acquired by the model when it was trained. These weights can be disabled using a parameter (pretrained = False) in the pytorch code, effectively making it an empty architecture with no weights assigned. In such a scenario, the model would be assigned weights during the training phase of our project.

2.2 Fine tuning the ConvNet

This approach is similar to the previous approach. However, in this approach we used a pretrained model along with its weights. As most of the models have been trained on ImageNet which is a huge dataset of images containing up to 24K labels. Thus, using these models along with their weights helps our model learn faster.

FINDING EXISTING SOLUTIONS AND REPRODUCING BASELINE:

We found a publicly available solution for our problem that uses the pretrained resnet-50 model to train a neural network for our task. We reproduced the code, but for our baseline we did not apply transfer learning. We just used the architecture of resnet-50, discarding the pretrained weights.

1.Loading The Data:
The Data is loaded in three frames. One frame for protest, one frame for violence, and one frame for all the remaining visual attributes.

Loading data for Training:

```
class ProtestDataset(Dataset):
    """
    dataset for training and evaluation
    """

    def __init__(self, txt_file, img_dir, transform=None):
        """
        Args:
            txt_file: Path to txt file with annotation
            img_dir: Directory with images
            transform: Optional transform to be applied on a sample.
        """
        self.label_frame = pd.read_csv(txt_file, delimiter="\t").replace('-', 0)
        self.img_dir = img_dir
        self.transform = transform

    def __len__(self):
        return len(self.label_frame)

    def __getitem__(self, idx):
        imgpath = os.path.join(self.img_dir,
                               self.label_frame.iloc[idx, 0])
        image = pil_loader(imgpath)

        protest = self.label_frame.iloc[idx, 1:2].values.astype('float')
        violence = self.label_frame.iloc[idx, 2:3].values.astype('float')
        visattr = self.label_frame.iloc[idx, 3:].values.astype('float')
        label = {'protest': protest, 'violence': violence, 'visattr': visattr}

        sample = {"image": image, "label": label}
        if self.transform:
            sample["image"] = self.transform(sample["image"])
        return sample
```

Loading data for Testing:

```
class ProtestDatasetEval(Dataset):
    """
    dataset for just calculating the output (does not need an annotation file)
    """

    def __init__(self, img_dir):
        """
        Args:
            img_dir: Directory with images
        """
        self.img_dir = img_dir
        self.transform = transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225]),
        ])
        self.img_list = sorted(os.listdir(img_dir))

    def __len__(self):
        return len(self.img_list)

    def __getitem__(self, idx):
        imgpath = os.path.join(self.img_dir,
                               self.img_list[idx])
        image = pil_loader(imgpath)
        # we need this variable to check if the image is protest or not)
        sample = {"imgpath": imgpath, "image": image}
        sample["image"] = self.transform(sample["image"])
        return sample
```

2. Creating the Model:

Defining the creation of the model to satisfy the needs of the project:

```
def modified_resnet50():
    # load pretrained resnet50 with a modified last fully connected layer
    model = models.resnet50(pretrained=True)
    model.fc = FinalLayer()

    # uncomment following lines if you wnat to freeze early layersz
    # i = 0
    # for child in model.children():
    #     i += 1
    #     if i < 4:
    #         for param in child.parameters():
    #             param.requires_grad = False

    return model
```

Defining the addition of final layer that is to be added over the pretrained model for transfer learning:

```
class FinalLayer(nn.Module):
    """modified last layer for resnet50 for our dataset"""

    def __init__(self):
        super(FinalLayer, self).__init__()
        self.fc = nn.Linear(2048, 12)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        out = self.fc(x)
        out = self.sigmoid(out)
        return out
```

3.Training the model:

```
end = time.time()
loss_history = []
for i, sample in enumerate(train_loader):
    # measure data loading batch_time
    input, target = sample['image'], sample['label']
    data_time.update(time.time() - end)
    if args.cuda:
        input = input.cuda()
        for k, v in target.items():
            target[k] = v.cuda()
    target_var = {}
    for k,v in target.items():
        target_var[k] = Variable(v)

    input_var = Variable(input)
    output = model(input_var)

    losses, scores, N_protest = calculate_loss(output, target_var, criterions)

    optimizer.zero_grad()
    loss = 0
    for l in losses:
        loss += l
    # back prop
    loss.backward()
    optimizer.step()
    if N_protest:
        loss_protest.update(losses[0].item(), input.size(0))
        loss_v.update(loss.item() - losses[0].item(), N_protest)
    else:
        # when there is no protest image in the batch
        loss_protest.update(losses[0].item(), input.size(0))
    loss_history.append(loss.item())
    protest_acc.update(scores['protest_acc'], input.size(0))
    violence_mse.update(scores['violence_mse'], N_protest)
    visattr_acc.update(scores['visattr_acc'], N_protest)

    batch_time.update(time.time() - end)
    end = time.time()
```

4.Testing the model:

```
def eval_one_dir(img_dir, model):
        """
        return model output of all the images in a directory
        """
        model.eval()
        # make dataloader
        dataset = ProtestDatasetEval(img_dir = img_dir)
        data_loader = DataLoader(dataset,
                                 num_workers = args.workers,
                                 batch_size = args.batch_size)
        # load model

        outputs = []
        imgpaths = []

        n_imgs = len(os.listdir(img_dir))
        with tqdm(total=n_imgs) as pbar:
            for i, sample in enumerate(data_loader):
                imgpath, input = sample['imgpath'], sample['image']
                if args.cuda:
                    input = input.cuda()

                input_var = Variable(input)
                output = model(input_var)
                outputs.append(output.cpu().data.numpy())
                imgpaths += imgpath
                if i < n_imgs / args.batch_size:
                    pbar.update(args.batch_size)
                else:
                    pbar.update(n_imgs%args.batch_size)


        df = pd.DataFrame(np.zeros((len(os.listdir(img_dir)), 13)))
        df.columns = ["imgpath", "protest", "violence", "sign", "photo",
                      "fire", "police", "children", "group_20", "group_100",
                      "flag", "night", "shouting"]
        df['imgpath'] = imgpaths
        df.iloc[:,1:] = np.concatenate(outputs)
        df.sort_values(by = 'imgpath', inplace=True)
        return df
```

3. Visualization

The results of testing are stored in a csv file. Each image has predicted values ranging between 0 and 1.

We visualize the plots in a different python code where the csv result is stored in pandas dataframes and the floating values are converted to 0 or 1 based on a threshold value(0.5). This conversion is done for all the attributes except violence, as violence needs to be a floating value.

The results for the baseline were as follows.

As the baseline, we used resnet-50 (pretrained = False)
Plots are available at the following link:
https://drive.google.com/open?id=1ywXrDF-tdxzkMdTH5zZEOdyfhwIWs8xx

Attribute	*Accuracy*	*Area under the curve*
Children	96.8%	57.7%
Fire	93.3%	71.2%
Flag	91.7%	56.9%
Group_20	72.1%	67.3%
Group_100	75.2%	69.4%
Night	92.9%	89.7%
Photo	97.1%	58.6%
Police	93.6%	65.0%
Protest	74.7%	80.0%
Shouting	95.3%	60.1%
Sign	81.8%	63.5%
Violence	Correlation = 0.392	

TWEAKING THE BASELINE:

Resnet-50

1. Resnet-50 (pretrained = True)

We used the pretrained resnet-50 model (i.e along with weights) to see how it performs. The results were as follows. Plots and results are available at the link : https://drive.google.com/open?id=1ItyeVJnOZjX4t2vYPsKUYZz7Bnuqpw7b

Attribute	*Accuracy*	*Area under the curve*
Children	96.8%	71%
Fire	96.7%	97%
Flag	91.8%	81%
Group_20	76.2%	80.1%
Group_100	78.3%	81.7%
Night	94.1%	92.9%
Photo	97.1%	69.8%
Police	94.0%	89.5%
Protest	86.9%	93.6%
Shouting	95.3%	78.3%
Sign	87.8%	90.8%
Violence	Correlation = 0.902	

2. Resnet-50 (pretrained = True) as a fixed feature extractor (all the resnet layers frozen)

This is called “feature extraction” and it is a variation of transfer learning. In theory, when the layers of a pretrained model are frozen, the error is not propagated all the way back to the initial layers. The error is only propagated to the layer we added on top of resnet-50. This would drop the accuracy but the model trains faster. Plots and results are available at the link : https://drive.google.com/open?id=1pgKrkdqm9wITKVU370G2tY1Lh32gdKGd

Attribute	*Accuracy*	*Area under the curve*
Children	96.8%	74.7%
Fire	96.5%	97.6%
Flag	91.7%	79.5%
Group_20	75.5%	79.2%
Group_100	79.3%	80.9%
Night	94.2%	92.0%
Photo	97.1%	71.4%
Police	90.4%	90.6%
Protest	84.2%	91.9%
Shouting	95.3%	81.9%
Sign	85.0%	88.1%
Violence	Correlation = 0.104	

3. Resnet-50 (pretrained = True) first 4 layers frozen

Since the results from feature extraction were not as good as 1, we decided to freeze only 4 layers to see the performance of the model. Plots and results are available at the link : https://drive.google.com/open?id=1WkfLcJhhaDv-5SDHs4dPO_YkppWL9y9P

Attribute	*Accuracy*	*Area under the curve*
Children	96.8%	79.6%
Fire	97.8%	98.6%
Flag	91.9%	83.3%
Group_20	78.9%	81.5%
Group_100	81.1%	84.4%
Night	93.7%	94.0%
Photo	97.1%	81.4%
Police	90.4%	90.6%
Protest	85.3%	93.2%
Shouting	95.2%	84.1%
Sign	88.6%	91.8%
Violence	Correlation = 0.911	

4. Changing the optimizer

We tried using different types of optimizers such as Adam, Adagrad, Adamax, RMSprop and Adadelta to check if we could get better results. We could not better the results of SGD. But Adadelta provided a very close accuracy to the best model. Plots and results are available at the link : https://drive.google.com/open?id=1Tfo8ZIhVck3W09RgdfKXDPPA71kr5xo9

Attribute	*Accuracy*	*Area under the curve*
Children	96.8%	78.1%
Fire	97.0%	98.6%
Flag	91.7%	85.0%
Group_20	76.8%	81.5%
Group_100	80.0%	84.4%
Night	93.8%	94.0%
Photo	97.1%	81.4%
Police	93.6%	90.6%
Protest	90.0%	93.2%
Shouting	95.3%	84.1%
Sign	88.3%	91.8%
Violence	Correlation = 0.878	

Resnet-152

1. Resnet-152 (pretrained = True)

Compared to the resnet-50 network, resnet-150 is a deeper implementation with more layers. This model won the ILSVRC award in 2015. Plots and results are available at the link : https://drive.google.com/open?id=1uCegWZaOfm2DpQ22ey8lkj9yFEdFdKFR

Attribute	*Accuracy*	*Area under the curve*
Children	96.8%	78.1%
Fire	97.8%	98.6%
Flag	92.1%	85.0%
Group_20	79.0%	81.5%
Group_100	80.9%	84.4%
Night	93.6%	94.0%
Photo	97.1%	81.4%
Police	96.8%	90.6%
Protest	91.7%	93.2%
Shouting	95.2%	84.1%
Sign	88.6%	91.8%
Violence	Correlation = 0.905	

2. Resnet-152 (pretrained = True, learning rate = 0.001)

When the learning rate is reduced, the model learns slower but captures more features from the data. We changed the learning rate from 0.01 to 0.001 and trained the resnet-152 model again with the following results. Plots and results are available at the link : https://drive.google.com/open?id=1rF22PlFmuAJaIsJ4rnQHr7lN4dSVuO3a

Attribute	*Accuracy*	*Area under the curve*
Children	97.0%	82.3%
Fire	97.8%	98.8%
Flag	92.0%	85.5%
Group_20	78.3%	81.5%
Group_100	80.7%	84.7%
Night	93.9%	94.6%
Photo	97.2%	82.1%
Police	95.5%	93.4%
Protest	92.0%	97.2%
Shouting	95.1%	85.6%
Sign	89.4%	92.2%
Violence	Correlation = 0.909	

RESULT COMPARISON

model	children	fire	flag	group 20	group 100	night	photo	police	protest	shouting	sign	violence
Baseline	96.80 %	93.30 %	91.70 %	72.10 %	75.20 %	92.90 %	97.10 %	93.60 %	74.70 %	95.30 %	81.80 %	0.392
Resnet-50 (Pretrained = True)	96.80 %	96.70 %	91.80 %	76.20 %	78.30 %	94.10 %	97.10 %	94%	86.90 %	95.30 %	87.80 %	0.865
Resnet-50 (Pretrained = True) All layers frozen	96.80 %	96.50 %	91.70 %	75.50 %	79.30 %	94.20 %	97.10 %	90.40 %	84.20 %	95.30 %	85%	0.104
Resnet-50 (Pretrained = True) 4 layers frozen	96.80 %	97.80 %	91.90 %	78.90 %	81.10 %	93.70 %	97.10 %	95.30 %	91.90 %	95.20 %	88.60 %	0.911
Adadelta	96.80 %	97.00 %	91.70 %	76.80 %	80%	93.80 %	97.10 %	93.60 %	90%	95.30 %	88.30 %	0.878
Resnet152 (Pretrained = True)	96.80 %	97.80 %	92.10 %	79%	80.90 %	93.60 %	97.10 %	96.80 %	91.70 %	95.20 %	88.60 %	0.905
Resnet-152 (pretrained = True, learning rate = 0.001)	97%	97.80 %	92%	78.30 %	80.70 %	93.90 %	97.20 %	95.50 %	92%	95.10 %	89.40 %	0.909

CONCLUSION

Resnet-50 (Pretrained = True) All layers frozen and Resnet-152 (pretrained = True, learning rate = 0.001) are the two models that provide the best solutions as seen from the above result comparison table.