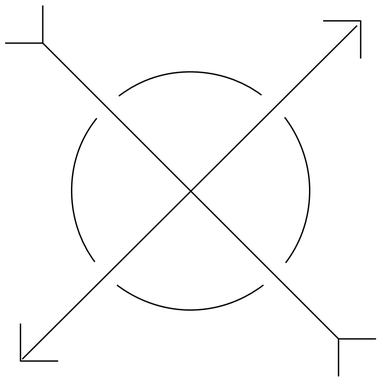


FaceHack

Group 24

https://github.com/peppermenta/faceHack

```
for X,y in testLoader:
  X = X.to(device)
  out = model(X)
  _,pred = torch.max(out,dim=1)
  if pred == 0:
    labels.append(0)
  else:
    out = model1(X)
    _,pred = torch.max(out,dim=1)
    labels.append(pred.item() + 1)
for i in range(testDataset.data.shape[0]):
  true_label = testDataset.data[i][1]
  if true_label == 'NOT smile':
    true_labels.append(0)
  elif true_label == 'negative smile':
    true_labels.append(1)
  else:
    true_labels.append(2)
count = 0
for i in range(len(labels)):
  if labels[i] == true_labels[i]:
    count += 1
print(count / len(true_labels))
```

Smile Detection

Binary Classification

The problem statement involved two binary classification of images into:

1. Positive Smile vs. No Smile
2. Positive Smile vs. Negative Smile

Our Models

For the classification, we have used two separate classification models. This avoids the need for making complex models with many classes; a simple binary classification can be implemented instead.

01

Smile vs. No Smile

First, a model which identifies the presence of a smile is run on the input image, and is labelled accordingly

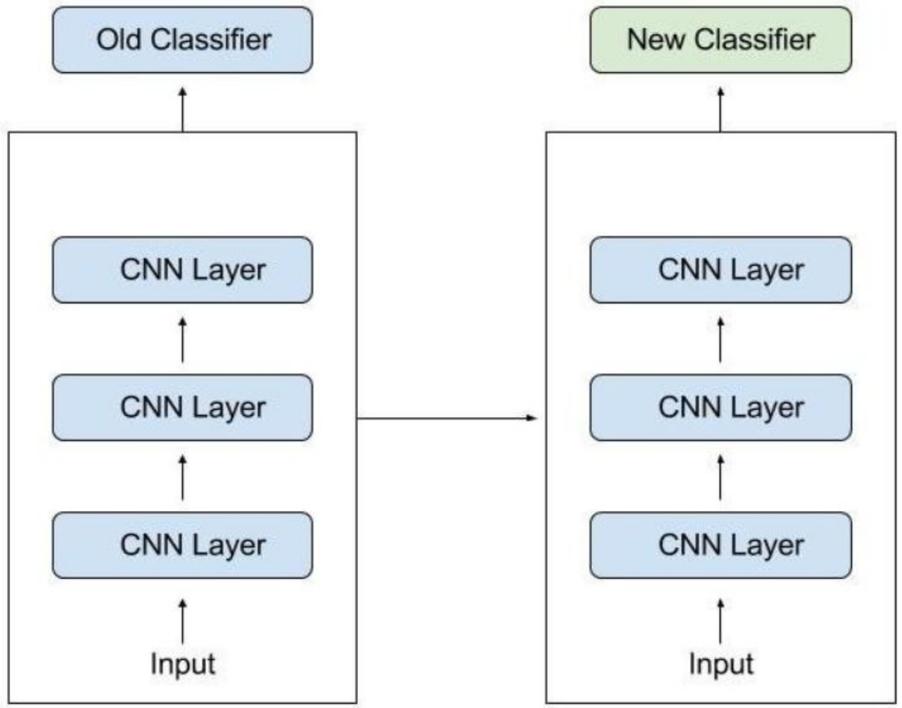
02

Positive Smile vs. Negative Smile

If the image is classified as a smile, then the next classifier is run on the input, thus classifying it as a positive or negative smile

Transfer Learning

In transfer learning, the knowledge of an already trained machine learning model is applied to a different but related problem. We transfer the weights that a network has learned at "task A" to a new "task B."



Transfer Learning has the following benefits:

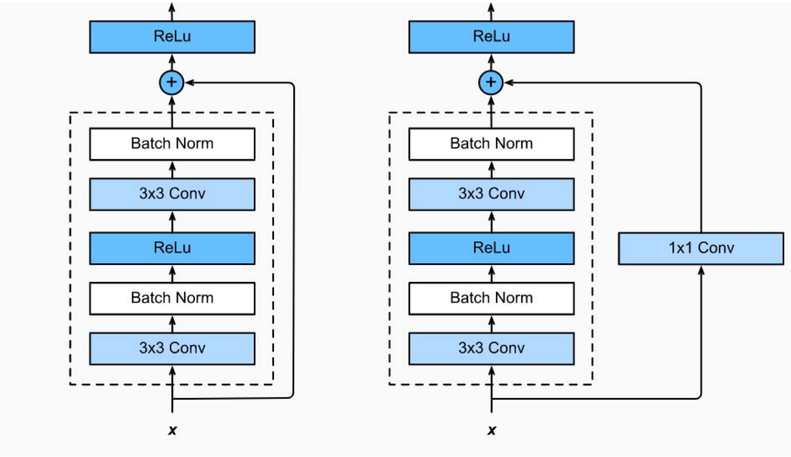
1. The Dataset for the Problem was small for CNNs, comprising of only 6000 images, where normal models require tens of thousands or maybe millions. Transfer Learning allows us to use a small Data Set and get accurate results.

2. Manipulating the Layers of the Model is easier, where the required functionality of the Model could be extracted and applied as needed. Using the model on its own resulted in an overfit in the final model, which was solved adding two extra layers and manipulating the layers as required.

3. Finally, using this method saves time, as training a model of this scale and accuracy on its own would take much longer.

ResNet + Transfer Learning

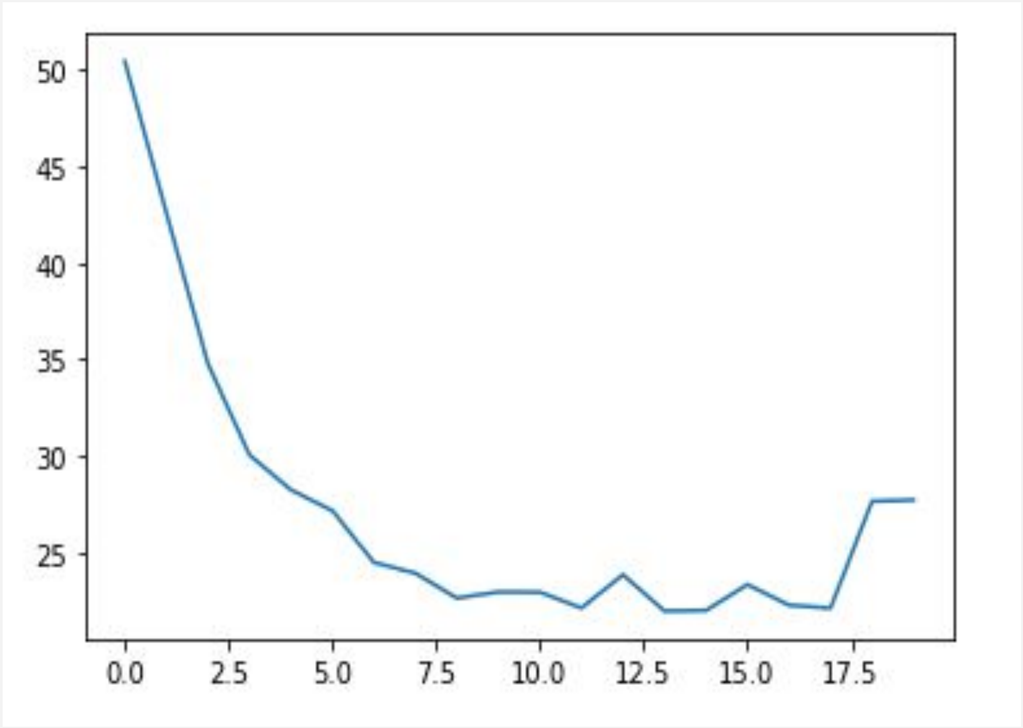
Residual Deep Neural Network(ResNets), which is a deep convolutional network with residual connections.



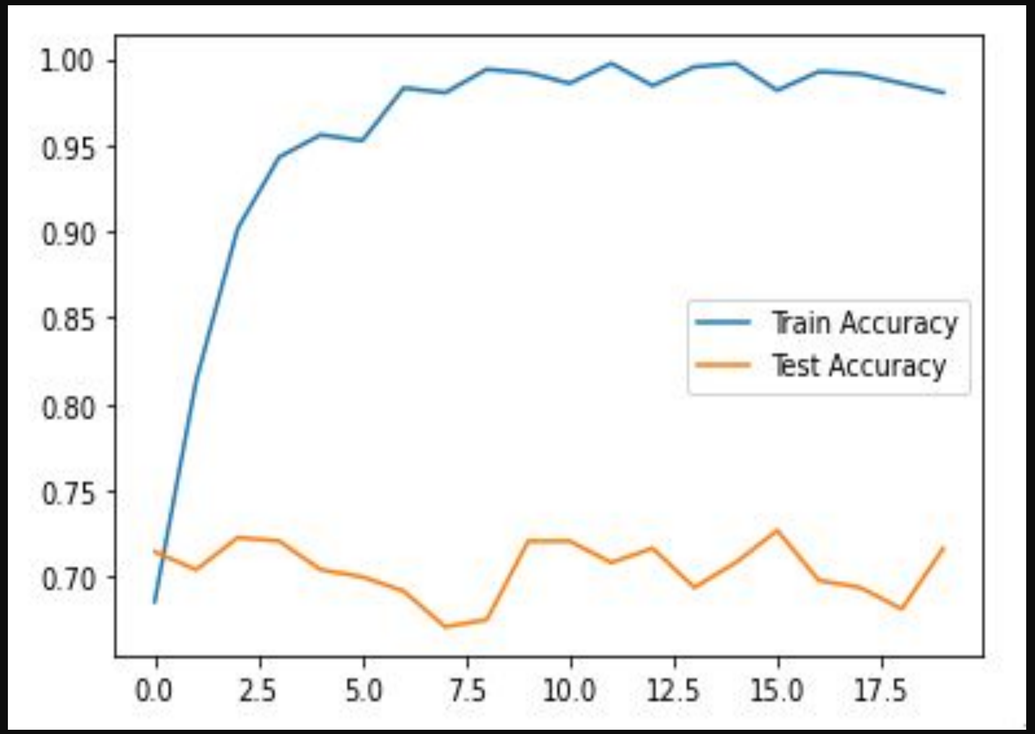

```
model = torchvision.models.resnet18(pretrained=True)

for param in model.parameters():
  param.requires_grad = False
for param in model.layer4.parameters():
  param.requires_grad = True

in_features = model.fc.in_features
model.fc = torch.nn.Sequential(
    torch.nn.Linear(in_features=in_features,out_features=128),
    torch.nn.Linear(in_features=128,out_features=2)
)

model = model.to(device)
```

- Here, we'll be using a variant of this ResNet, called ResNet18, pretrained on the ImageNet Dataset. It is consists of 4 residual blocks, and can classify images onto 1000 classes, with the input image being of size 224x224.
- We used a Resnet18, pre-trained on ImageNet, as the feature extractor for the fully connected classification head.
- Resnet-18 contains 4 residual blocks, followed by a fully connected layer. We replaced the fully connected layer to a binary output, and froze the pre-trained weights, except the last residual block, and the last fully connected head.
- The small size of the dataset doesn't allow the network to learn useful convolutional filter weights in the lower layers. This was the motivation for using the pre-trained ResNet and transfer learning for a better classification accuracy.

Results:



Training Loss
vs. Epochs



Training and Test
Accuracy vs. Epochs

Thank You.

Group 24