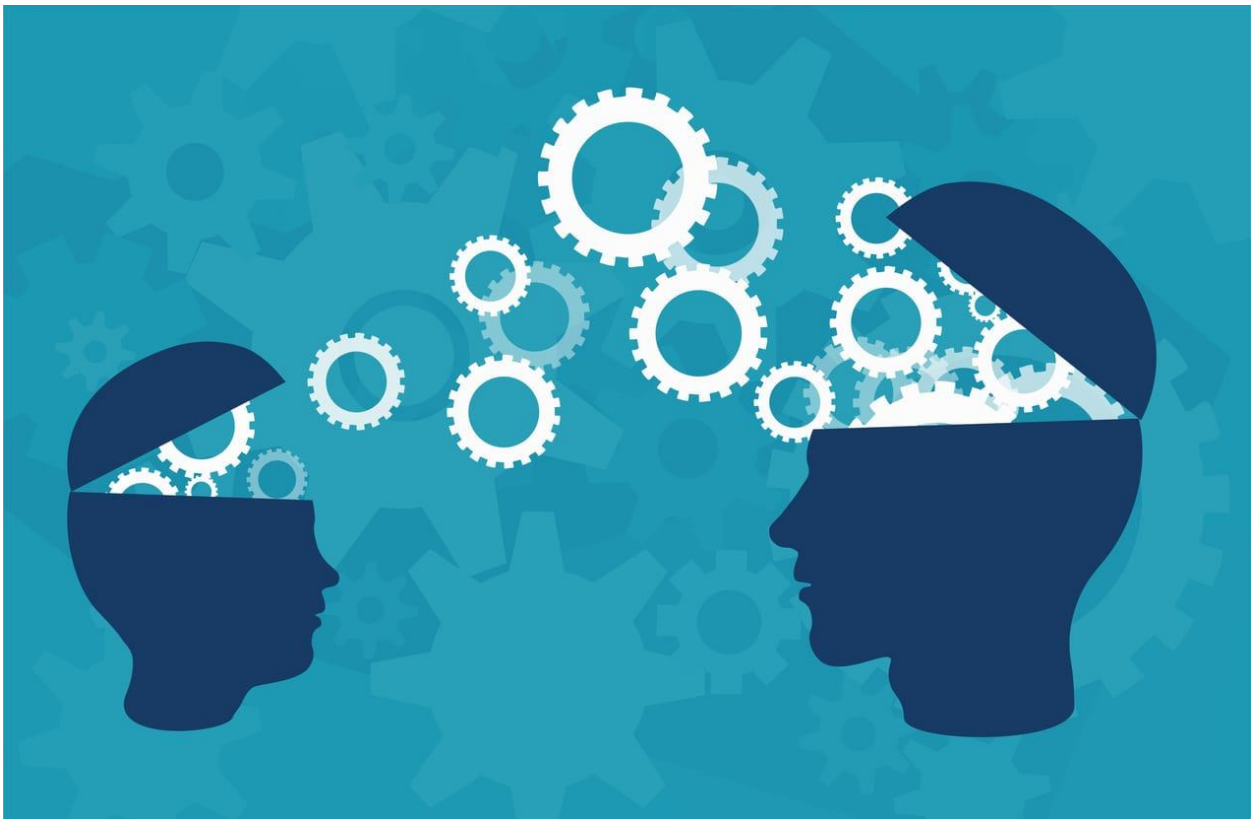
Facial Recognition

Transfer Learning Modeling



Main Objective of Analysis

Deep learning models trained on large datasets have gained traction recently and sophisticated generative transformer-based models with billions of parameters such as DALL-E and ChatGPT showcase the power of such architectures. The outputs of this analysis are more rudimentary in nature, they score and label an image of a face with its expression using deep learning models. Transfer learning was utilized with the FastAI library, where large, established neural networks pre-trained on big data provided most of the model and a single dense layer was added at the end with facial images as input and different facial expression categories as output. Two convolutional neural network (CNN) architectures, ResNet-34 and VGG19 were used in transfer learning. Image recognition models typically use CNN's. Accuracy and loss scores were assessed to determine which model performed best.

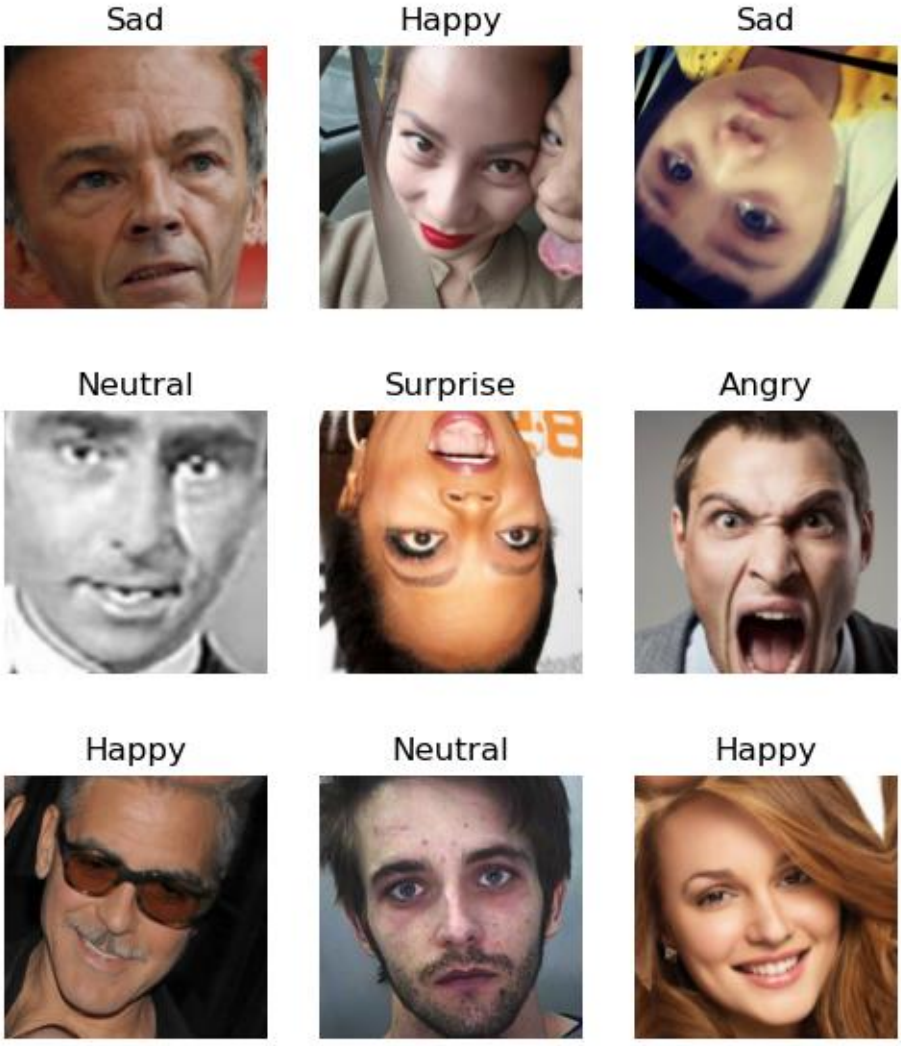
Description of Dataset

The dataset consisted of JPEG images of faces of various pixel dimensions and labeled “angry,” “happy,” “neutral,” “sad,” “surprise” to correspond to the facial expressions of each face. The counts of images within each category of facial expressions in the dataset:

- Angry – 1,313
- Happy – 3,740
- Neutral – 4,027
- Sad – 3,934
- Surprise – 1,235

Data Cleaning and Feature Engineering

The images were resized to 224 x 224 dimensions and the validation set was 20% of the total dataset. In addition, data augmentation was performed to expand the number of images and further refine training. The data augmentations were: vertical flipping, rotation, zoom, lighting changes, warping. A sample of the images after processing are shown below.

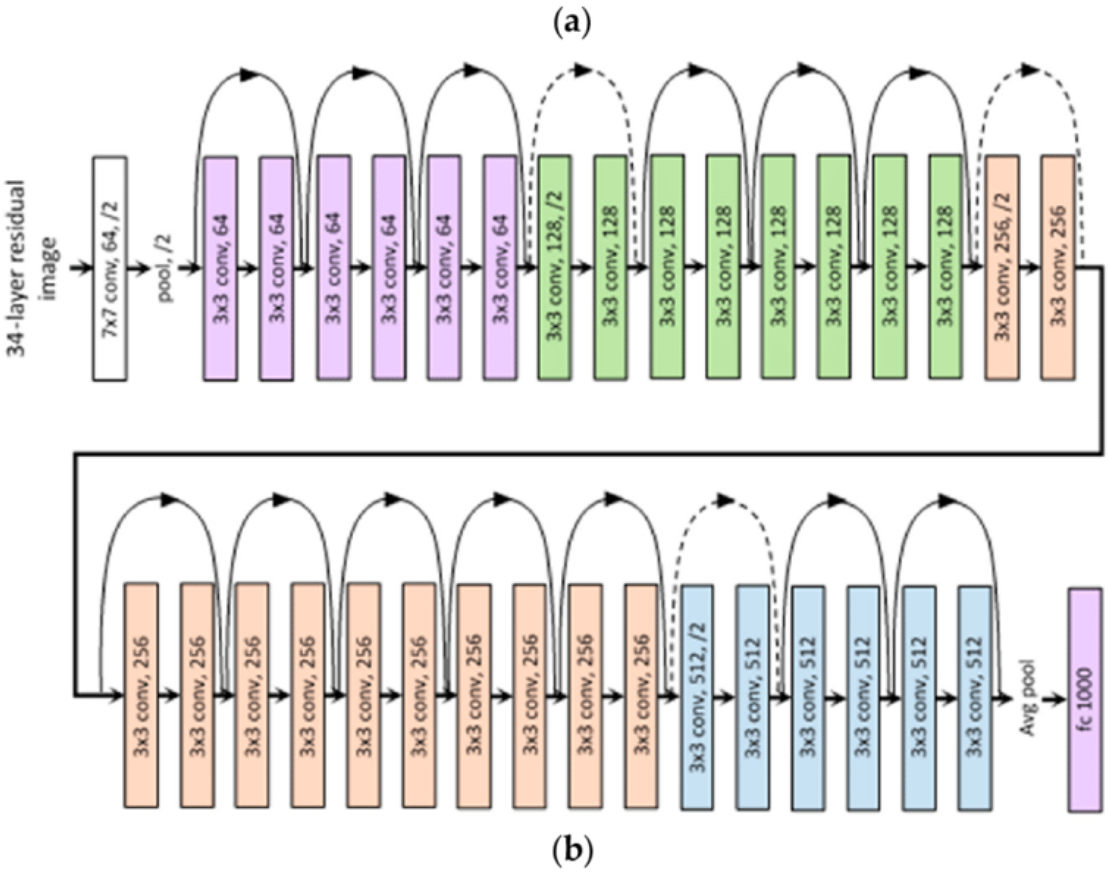


Transfer Learning Models

ResNet-34

A highly effective and efficient architecture for image classification tasks, leveraging residual connections to enable the training of deeper networks without suffering from the vanishing gradient problem. Residual connections are the identity shortcuts that add the input of a block to its output, allowing the gradient to flow more easily through the network during backpropagation.

Architecture diagram:



Model:

```
Sequential(
  (0): Sequential(
    (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (2): BasicBlock(
```

```
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (5): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
```

```
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (3): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (6): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (3): BasicBlock(
```

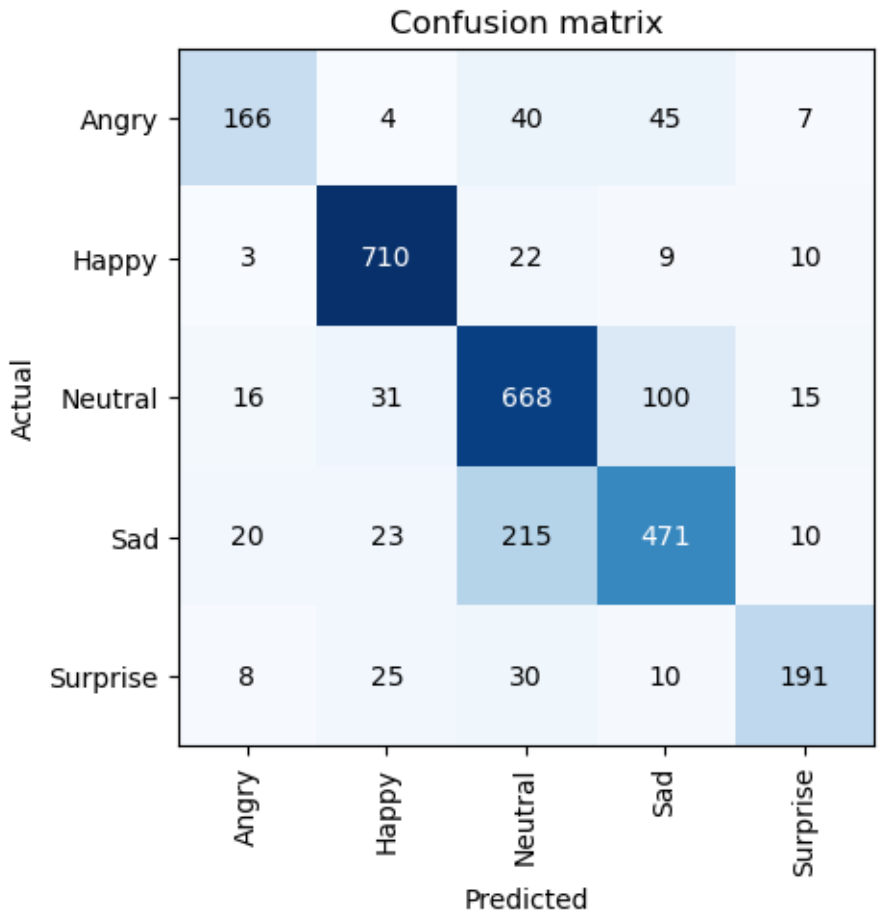
```
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (4): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (5): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (7): Sequential(
```

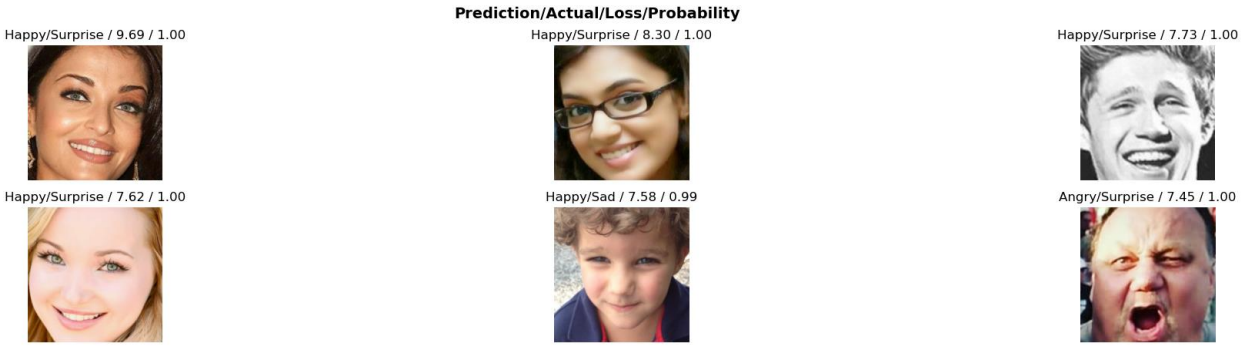
```
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
```

```
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (1): Sequential(
    (0): AdaptiveConcatPool2d(
      (ap): AdaptiveAvgPool2d(output_size=1)
      (mp): AdaptiveMaxPool2d(output_size=1)
    )
    (1): fastai.layers.Flatten(full=False)
    (2): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (3): Dropout(p=0.25, inplace=False)
    (4): Linear(in_features=1024, out_features=512, bias=False)
    (5): ReLU(inplace=True)
    (6): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (7): Dropout(p=0.5, inplace=False)
    (8): Linear(in_features=512, out_features=5, bias=False)
  )
)
```

The Resnet-34 model was trained for 3 epochs, as computational resources were limited. Early stopping callback based on accuracy was used to end the training if further epochs would not improve performance. Training proceeded for all 3 epochs and the loss for the validation set was less than for the training set, suggesting that the model did not overfit and would generalize well for new image data. According to the confusion matrix, the model performed with best accuracy on happy and neutral faces. But looking at the images with the highest loss scores, faces in the surprise category were mislabeled as happy. However, inspecting the actual images with highest loss, these surprise faces subjectively appear happy.

epoch	train_loss	valid_loss	accuracy	time
0	1.035992	0.812797	0.684451	51:40
1	0.785070	0.620490	0.760969	47:25
2	0.671556	0.565982	0.774307	1:23:15

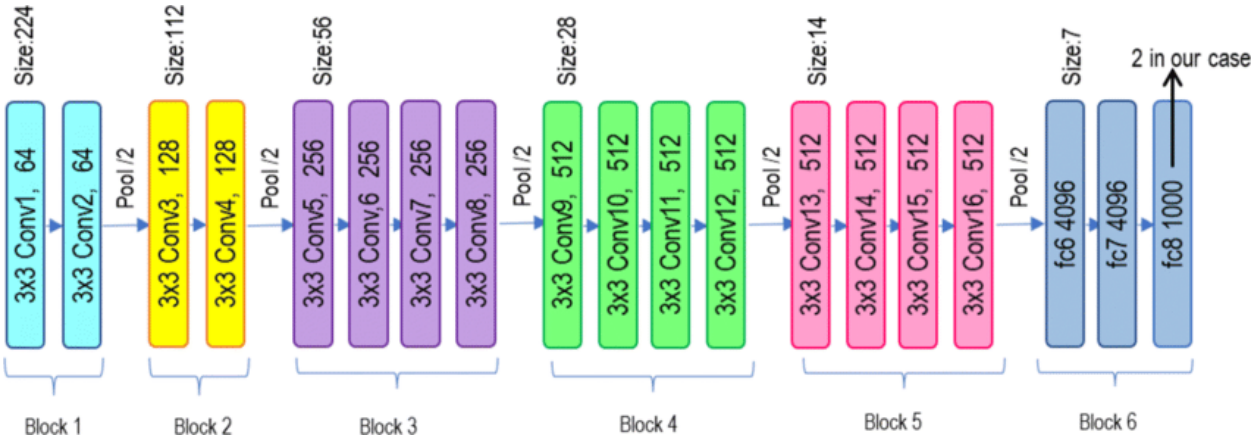

Confusion matrix
Angry
Happy
Neutral
Sad
Surprise
Actual
166 4 40 45 7
3 710 22 9 10
16 31 668 100 15
20 23 215 471 10
8 25 30 10 191
Angry
Happy
Neutral
Sad
Surprise
Predicted


Prediction/Actual/Loss/Probability
Happy/Surprise / 9.69 / 1.00
Happy/Surprise / 8.30 / 1.00
Happy/Surprise / 7.73 / 1.00
Happy/Surprise / 7.62 / 1.00
Happy/Sad / 7.58 / 0.99
Angry/Surprise / 7.45 / 1.00

VGG19

A powerful and deep CNN architecture that leverages small convolutional filters, deep layers, and a straightforward design to achieve high performance on image classification tasks. Its simplicity and effectiveness have made it a popular choice for various computer vision applications. All convolutional layers use 3x3 filters, which simplifies the architecture and ensures consistent learning across layers.

Architecture diagram:



Model:

```
Sequential(
  (0): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (5): ReLU(inplace=True)
```

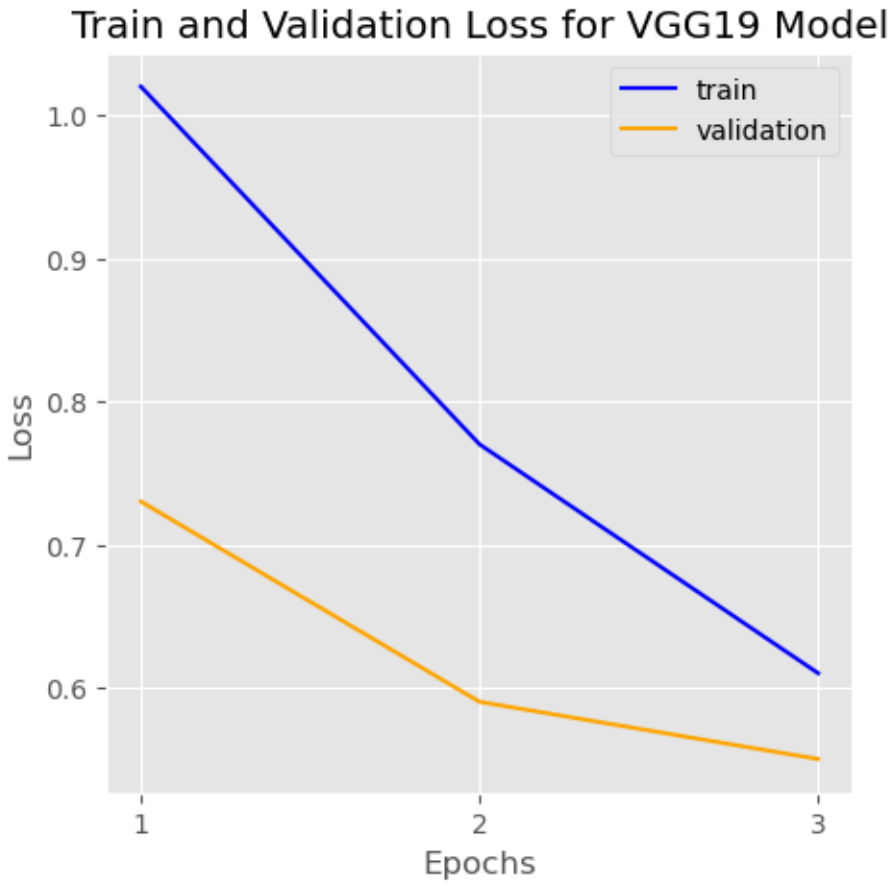
```
      (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

      (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

      (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

      (9): ReLU(inplace=True)

      (10): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

      (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

      (12): ReLU(inplace=True)

      (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)

      (14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

      (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

      (16): ReLU(inplace=True)

      (17): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

      (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

      (19): ReLU(inplace=True)

      (20): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

      (21): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

      (22): ReLU(inplace=True)

      (23): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

      (24): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

      (25): ReLU(inplace=True)

      (26): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)

      (27): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

      (28): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)

      (29): ReLU(inplace=True)
```

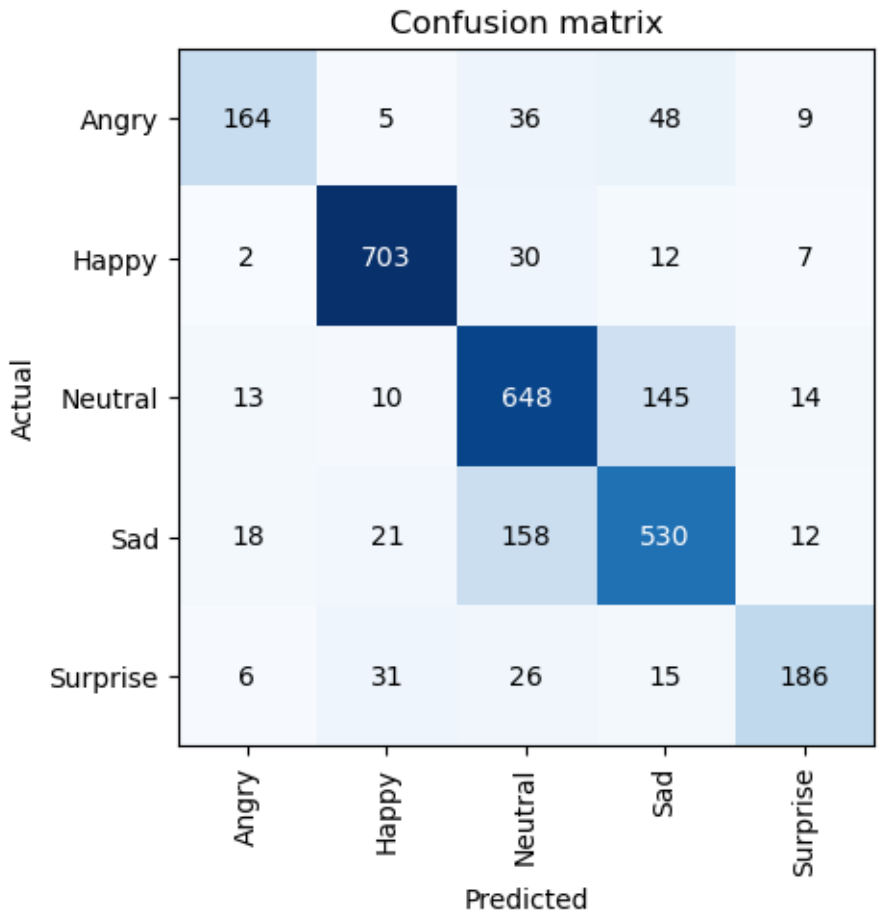
```
      (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (31): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (32): ReLU(inplace=True)
      (33): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (34): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (35): ReLU(inplace=True)
      (36): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (37): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (38): ReLU(inplace=True)
      (39): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
      (40): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (41): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (42): ReLU(inplace=True)
      (43): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (44): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (45): ReLU(inplace=True)
      (46): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (47): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (48): ReLU(inplace=True)
      (49): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (50): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (51): ReLU(inplace=True)
      (52): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    )
```

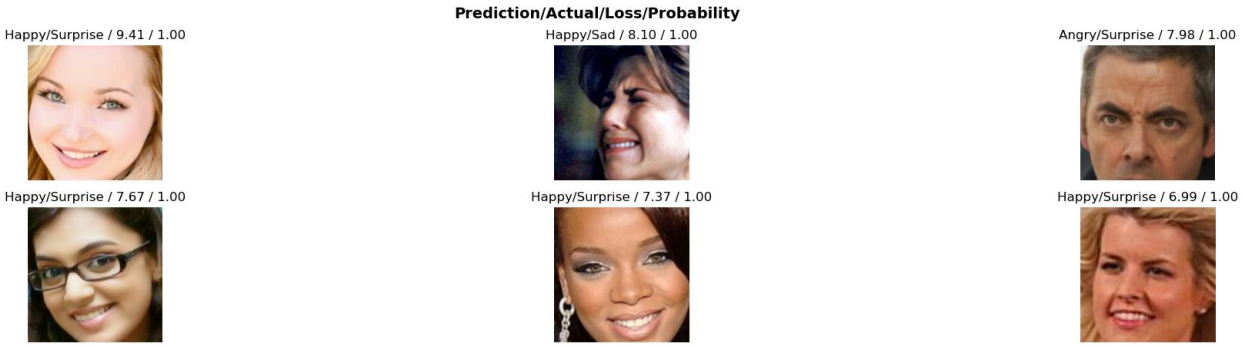
```
  )
  (1): Sequential(
    (0): AdaptiveConcatPool2d(
      (ap): AdaptiveAvgPool2d(output_size=1)
      (mp): AdaptiveMaxPool2d(output_size=1)
    )
    (1): fastai.layers.Flatten(full=False)
    (2): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (3): Dropout(p=0.25, inplace=False)
    (4): Linear(in_features=1024, out_features=512, bias=False)
    (5): ReLU(inplace=True)
    (6): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (7): Dropout(p=0.5, inplace=False)
    (8): Linear(in_features=512, out_features=5, bias=False)
  )
)
```

The VGG19 model was trained for 3 epochs, as computational resources were limited. Early stopping callback based on accuracy was used to end the training if further epochs would not improve performance. Training proceeded for all 3 epochs and the loss for the validation set was less than for the training set, suggesting that the model did not overfit and would generalize well for new image data. According to the confusion matrix, the model performed with best accuracy on happy and neutral faces. But looking at the images with the highest loss scores, faces in the surprise category were mislabeled as happy. Two of these images also appeared in highest loss for the ResNet-34 model. Inspecting the actual images with highest loss, aside from one, these surprise faces subjectively seemed happy.

epoch	train_loss	valid_loss	accuracy	time
0	1.025988	0.737682	0.713233	3:17:59
1	0.774489	0.595573	0.759916	10:15:29
2	0.619777	0.555484	0.783082	3:44:58




Confusion matrix
Angry
Happy
Neutral
Sad
Surprise
Actual
164 5 36 48 9
2 703 30 12 7
13 10 648 145 14
18 21 158 530 12
6 31 26 15 186
Angry
Happy
Neutral
Sad
Surprise
Predicted


Prediction/Actual/Loss/Probability
Happy/Surprise / 9.41 / 1.00
Happy/Sad / 8.10 / 1.00
Angry/Surprise / 7.98 / 1.00
Happy/Surprise / 7.67 / 1.00
Happy/Surprise / 7.37 / 1.00
Happy/Surprise / 6.99 / 1.00

Key Findings and Insights

Both the ResNet-34 and VGG19 transfer learning models were trained for the same number of epochs on the same image data. VGG19 took significantly longer to train than ResNet-34 as the depth and parameters of VGG19 are larger than ResNet-34. Both had higher loss scores for the training set compared to the validation set, suggesting they do not overfit. VGG19 with a final accuracy score of 0.78 performed slightly better than ResNet34 with a final accuracy score of 0.77. Subjectively inspecting the images with the highest error score, the labels were misleading.

Next Steps

Due to limitations in computational resources, the training for both models only ran for 3 epochs each. Utilizing processing resources from a cloud service such as AWS or Google Cloud would enable more epochs and perhaps higher performance. Other CNN architectures could be used as well for transfer learning models. A much larger image dataset would be preferable, but a hurdle with the dataset used in this analysis was that some of the images seemed to be incorrectly labeled. A model is only as good as the data it is trained on and images with better labeling could enhance the accuracy of the models after training. Also, the dataset was not balanced between categories and either undersampling or more data could help with performance.