CNN and Vision Transformer Pre-trained Models for Facial Emotion Recognition

Group 2
Luoning Zhang, Tom Gao, William Chu, Montek Kalsi, and Grant Liu

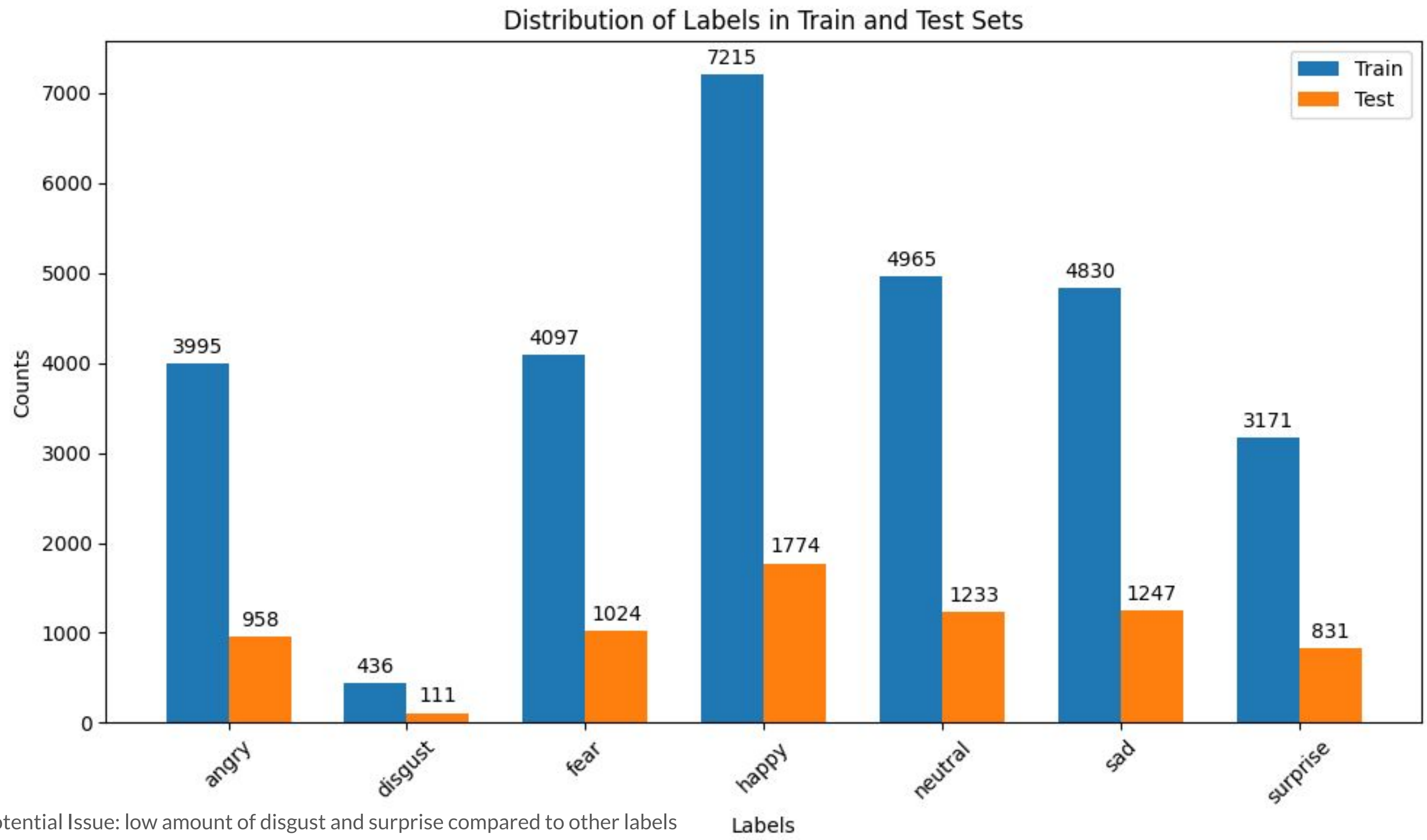
Introduction

Our project explores two distinct architectures, CNNs and Vision Transformers, to recognize facial emotions.

Why emotion recognition? It’s a useful and relevant problem for many modern day purposes

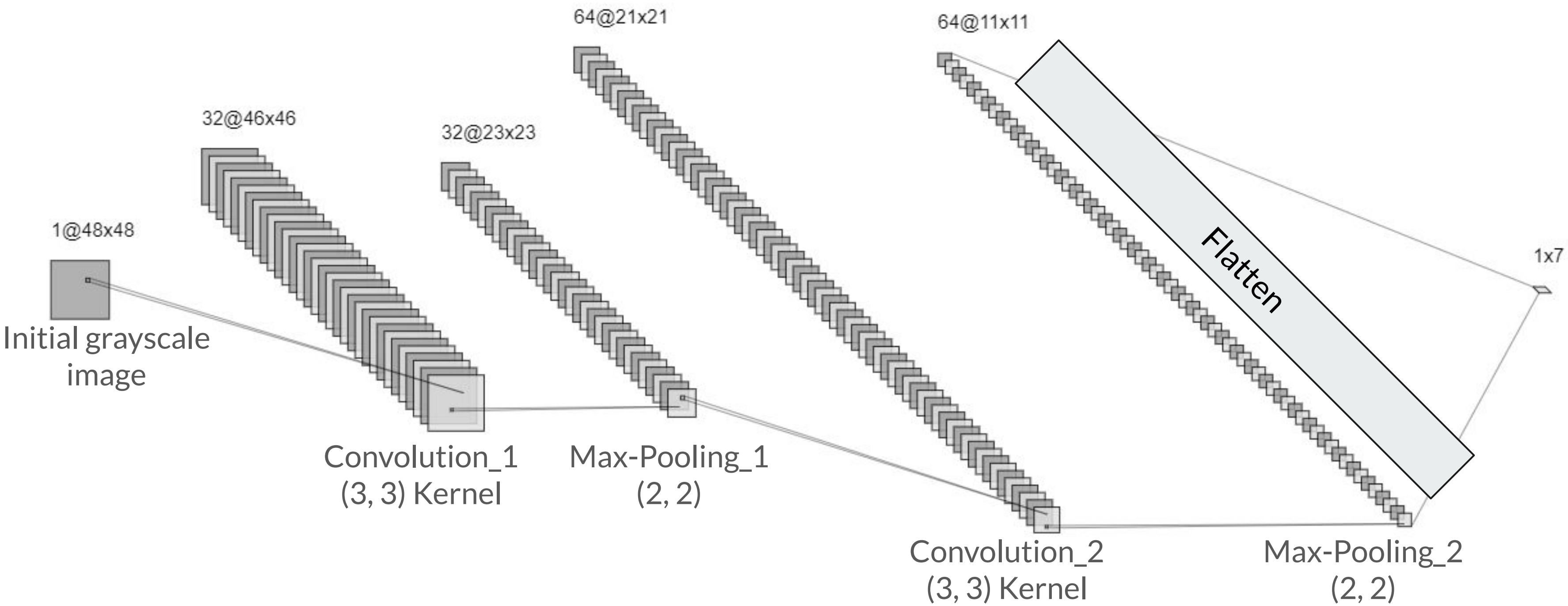
- Human-computer interaction (ex: face id)
- Applicable to mental health tools (detecting depression)
- Helpful with threat detection (security applications)

Both techniques are well documented approaches for facial recognition, and we will explore the strengths and weaknesses of both.



Potential Issue: low amount of disgust and surprise compared to other labels

Convolutional Neural Network (CNN) V1

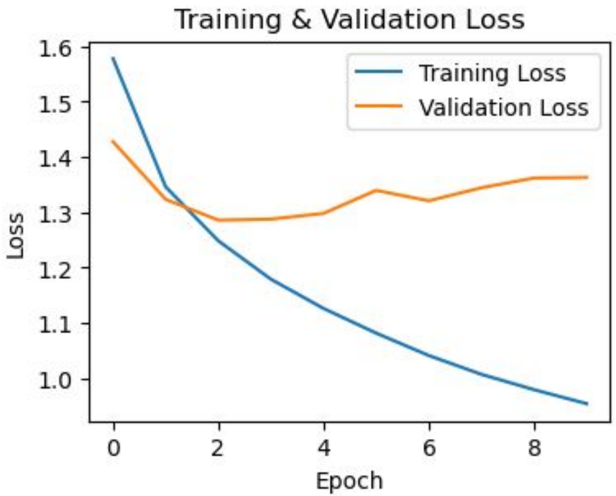


Data Preprocessing

```python
transform = transforms.Compose([
    transforms.Grayscale(),
    transforms.Resize((48, 48)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5])
])

train_dataset = datasets.ImageFolder(root='data/train', transform=transform)
test_dataset = datasets.ImageFolder(root='data/test', transform=transform)

class_counts = {"angry": 3995, "disgust": 436, "fear": 4097, "happy": 7215, "neutral": 4965, "sad": 4830, "surprise":3171}

# Sampled with weights inversely proportional to the number of images in each class
weights = [1.0 / counts for counts in class_counts.values()]
labels = train_dataset.classes
sample_weights = [weights[label] for _, label in train_dataset.samples]

# Create sampler and DataLoader for training
sampler = WeightedRandomSampler(sample_weights, len(sample_weights))

train_loader = DataLoader(dataset=train_dataset, batch_size=32, sampler=sampler, shuffle=False)
test_loader = DataLoader(dataset=test_dataset, batch_size=32, shuffle=False)
```

CNN V1

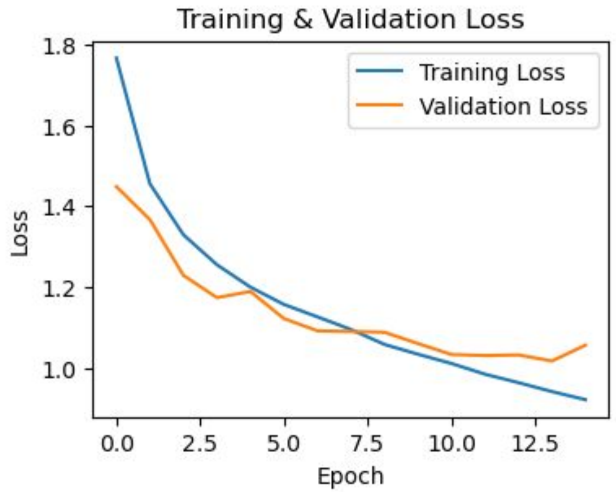
- Model does not learn past Epoch 2
- Underfitting? Overfitting?
- Validation Accuracy: 0.5181
- F1 Score: 0.5220



CNN V2

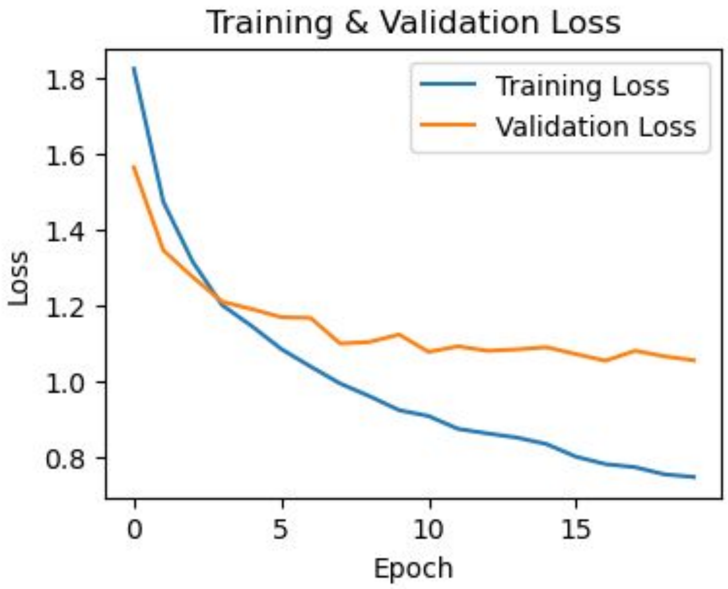
- Additional Convolutional Layers
- Batch Normalization
- Initially resulted in overfitting:
  - Dropout Layers
  - Increased dropout percentages
- Validation Accuracy: 0.6067
- F1 Score: 0.5887

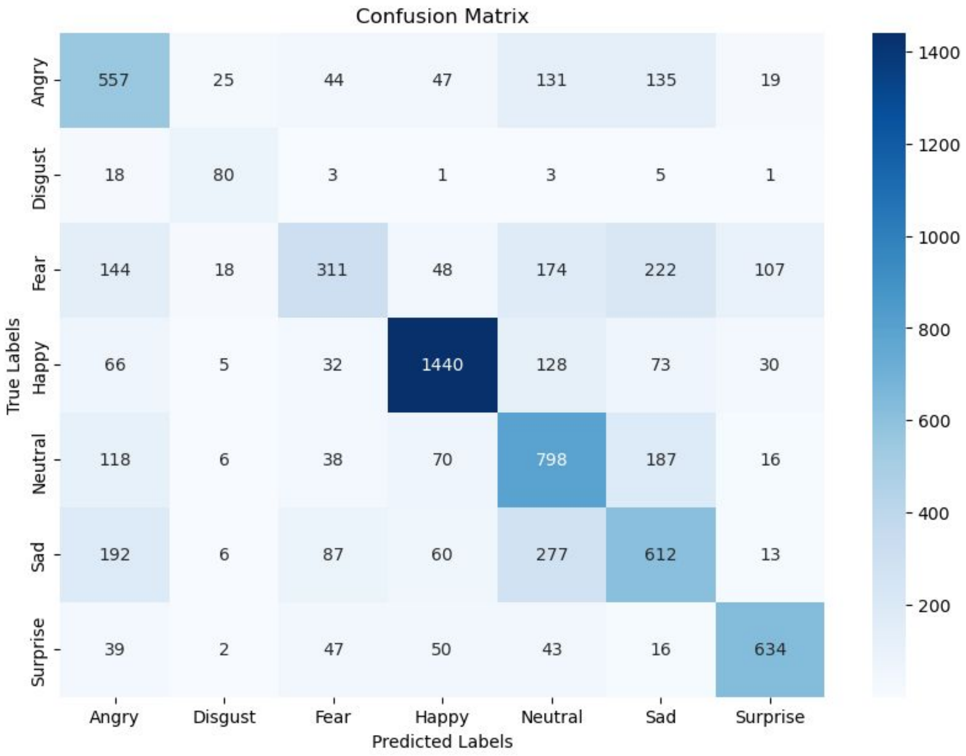
```
FERModel(
  (cnn_layers): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU()
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Dropout(p=0.5, inplace=False)
    (9): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU()
    (12): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (13): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (14): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (15): ReLU()
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Dropout(p=0.5, inplace=False)
  )
  (output_layer): Linear(in_features=2304, out_features=7, bias=True)
)
```



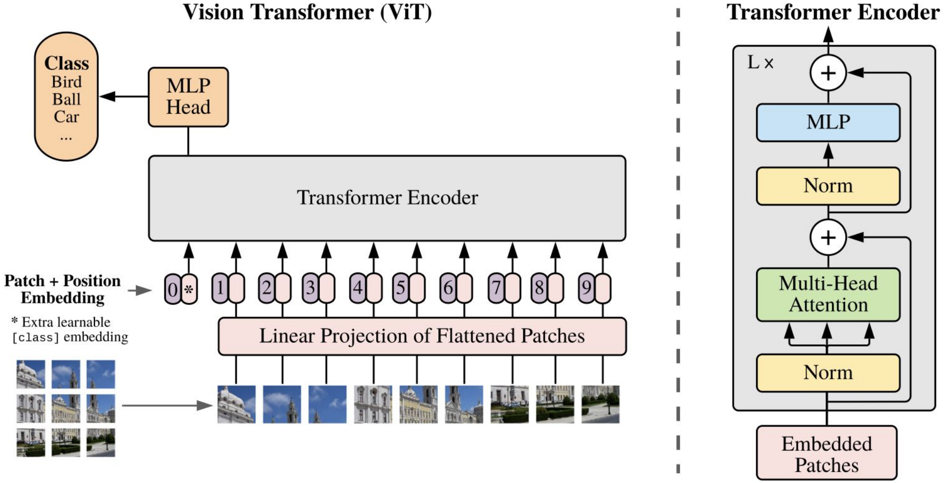
CNN V2.1 (Final)

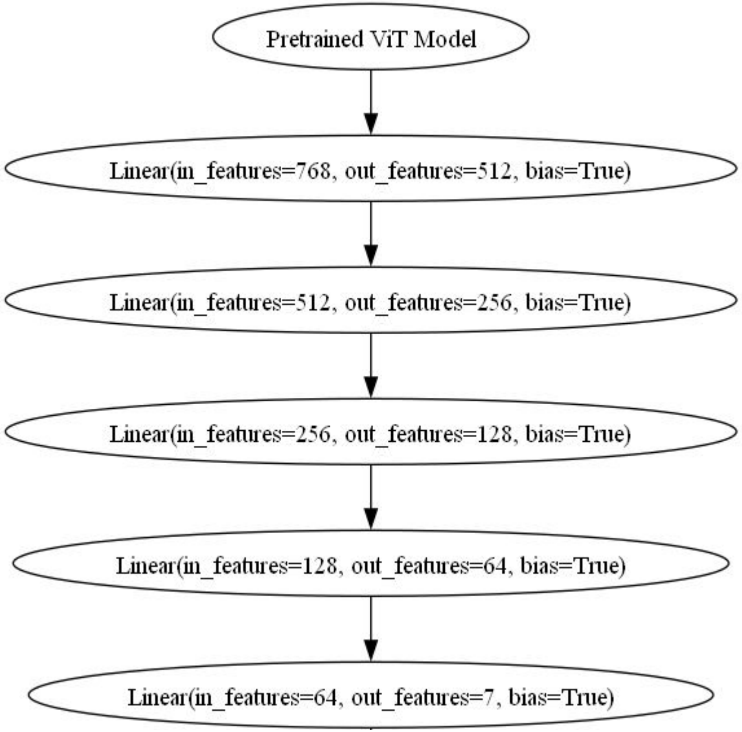
- Same Architecture as V2
- Retrained on resampled





ViT Architecture

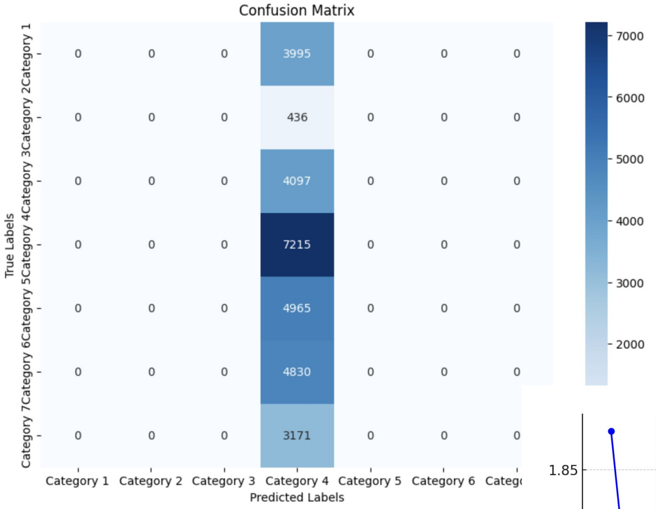




Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby "**An Image is Worth 16x16 Words**: Transformers for Image Recognition at Scale"

Initial attempt:
Train a ViT model from zero

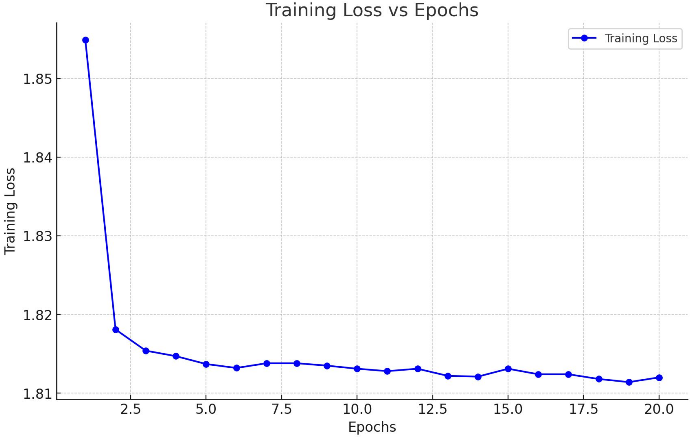
```
Using device: cuda
Epoch 1/20: 100%|██████████| 898/898 [01:34<00:00,  9.49batch/s]
Epoch [1/20], Loss: 1.8549, Time: 94.64 sec
Epoch 2/20: 100%|██████████| 898/898 [00:20<00:00, 43.93batch/s]
Epoch [2/20], Loss: 1.8181, Time: 20.44 sec
Epoch 3/20: 100%|██████████| 898/898 [00:20<00:00, 44.23batch/s]
Epoch [3/20], Loss: 1.8154, Time: 20.31 sec
Epoch 4/20: 100%|██████████| 898/898 [00:20<00:00, 43.81batch/s]
Epoch [4/20], Loss: 1.8147, Time: 20.50 sec
Epoch 5/20: 100%|██████████| 898/898 [00:20<00:00, 44.13batch/s]
Epoch [5/20], Loss: 1.8137, Time: 20.35 sec
Epoch 6/20: 100%|██████████| 898/898 [00:20<00:00, 43.27batch/s]
Epoch [6/20], Loss: 1.8132, Time: 20.76 sec
Epoch 7/20: 100%|██████████| 898/898 [00:22<00:00, 39.99batch/s]
Epoch [7/20], Loss: 1.8138, Time: 22.46 sec
Epoch 8/20: 100%|██████████| 898/898 [00:21<00:00, 41.09batch/s]
Epoch [8/20], Loss: 1.8138, Time: 21.85 sec
Epoch 9/20: 100%|██████████| 898/898 [00:24<00:00, 37.31batch/s]
Epoch [9/20], Loss: 1.8135, Time: 24.07 sec
Epoch 10/20: 100%|██████████| 898/898 [00:22<00:00, 40.22batch/s]
Epoch [10/20], Loss: 1.8131, Time: 22.33 sec
Epoch 11/20: 100%|██████████| 898/898 [00:22<00:00, 40.45batch/s]
Epoch [11/20], Loss: 1.8128, Time: 22.20 sec
Epoch 12/20: 100%|██████████| 898/898 [00:22<00:00, 39.60batch/s]
Epoch [12/20], Loss: 1.8131, Time: 22.68 sec
Epoch 13/20: 100%|██████████| 898/898 [02:20<00:00,  6.38batch/s]
Epoch [13/20], Loss: 1.8122, Time: 140.79 sec
Epoch 14/20: 100%|██████████| 898/898 [00:25<00:00, 34.62batch/s]
Epoch [14/20], Loss: 1.8121, Time: 25.94 sec
Epoch 15/20: 100%|██████████| 898/898 [00:19<00:00, 44.99batch/s]
```



Architecture: Convolutional Layer + Transformer Encoder + Fully Connected Layer

Result:

- Training loss didn't decrease (~1.8)
- Accuracy: 0.25



Reason: ViT doesn't fit small datasets

Using Pre-trained ViT Model

google/vit-base-patch16-224

Vision Transformer (ViT) model pre-trained on ImageNet-21k (14 million images, 21,843 classes) at resolution 224x224, and fine-tuned on ImageNet 2012 (1 million images, 1,000 classes) at resolution 224x224.

[google/vit-base-patch16-224 · Hugging Face]

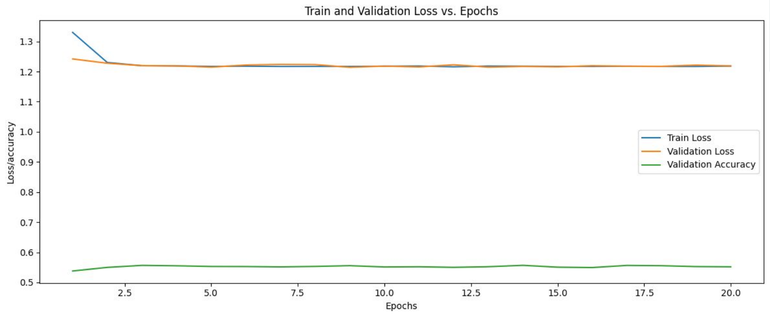
Data Preprocessing

The resolution of our test data images don't fit the ViT model, so we upscale the 48x48 images to 224x224. This may result in lower accuracies.

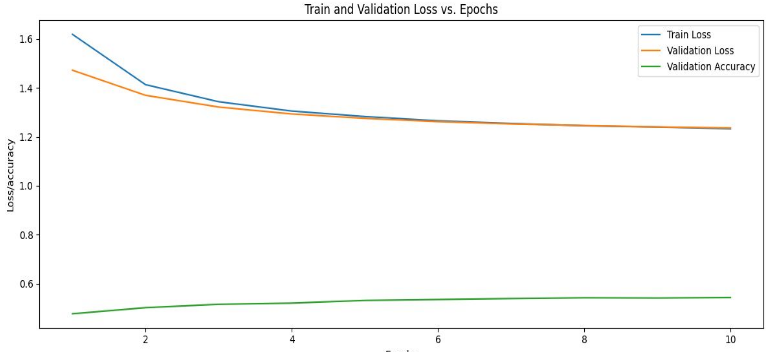
```
transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),
])

train_dataset = datasets.ImageFolder(root='archive/train', transform=transform)
test_dataset = datasets.ImageFolder(root='archive/test', transform=transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

Learning Rate

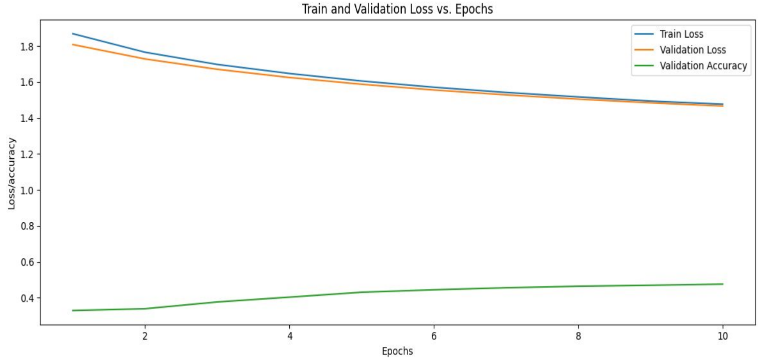
1e-3 doesn’t converge



1e-4 just right



1e-5 converge too slowly



Fine-tuning - V1.3

Replace classifier layer in pretrained model with "CustomClassifier"

Accuracy: 0.59

10 epochs

```
class CustomClassifier(nn.Module):
    def __init__(self, input_dim, num_classes):
        super(CustomClassifier, self).__init__()
        self.fc1 = nn.Linear(input_dim, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, num_classes)
        self.dropout = nn.Dropout(0.5)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc3(x)
        return x
```

Fine-tuning - V1.5

Accuracy: 0.61

Add 2 more fully connected layers

10 epochs

```
class CustomClassifier(nn.Module):
    def __init__(self, input_dim, num_classes):
        super(CustomClassifier, self).__init__()
        self.fc1 = nn.Linear(input_dim, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 128)
        self.fc4 = nn.Linear(128, 64)
        self.fc5 = nn.Linear(64, num_classes)
        self.dropout = nn.Dropout(0.5)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc3(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc4(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc5(x)
        return x
```

Fine-tuning - V1.6

Accuracy: 0.625

24 epochs

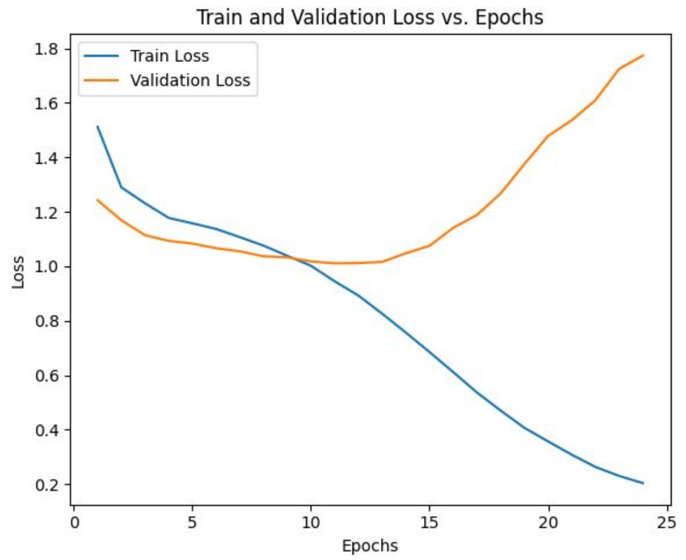
Gradually Unfreeze

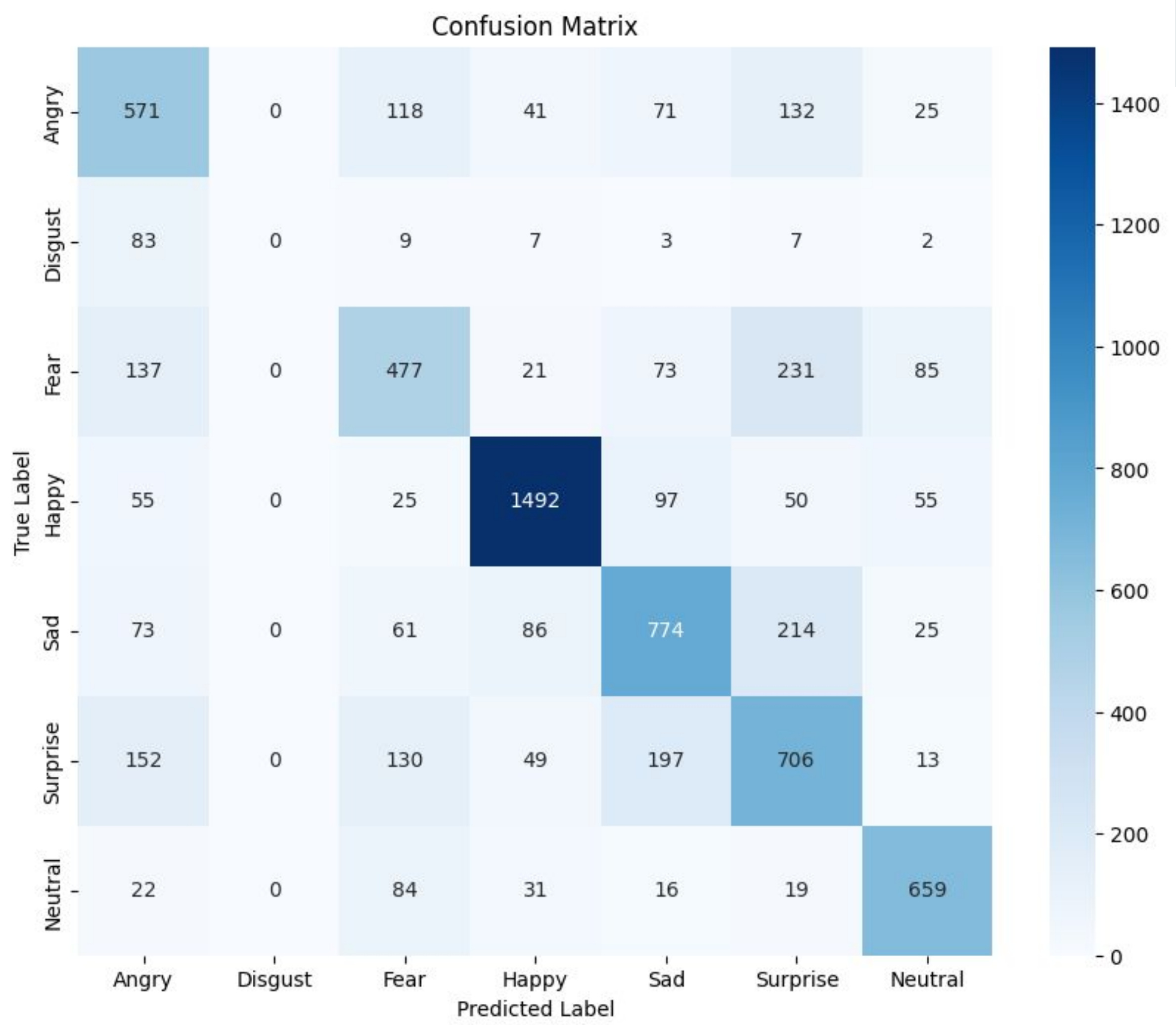
- Don’t update pretrained layers until achieving better accuracy

Notice overfitting on later epochs

Early stopping used

```
# Define a schedule to unfreeze layers gradually
unfreeze_schedule = {
    2: 'vit.encoder.layer.11',
    4: 'vit.encoder.layer.10',
    6: 'vit.encoder.layer.9',
    8: 'vit.encoder.layer.8',
    10: 'vit.encoder.layer.7',
    12: 'vit.encoder.layer.6',
    14: 'vit.encoder.layer.5',
    16: 'vit.encoder.layer.4',
    18: 'vit.encoder.layer.3',
}
```




Confusion Matrix
True Label
Predicted Label
Angry
Disgust
Fear
Happy
Sad
Surprise
Neutral
571 0 118 41 71 132 25
83 0 9 7 3 7 2
137 0 477 21 73 231 85
55 0 25 1492 97 50 55
73 0 61 86 774 214 25
152 0 130 49 197 706 13
22 0 84 31 16 19 659
1400
1200
1000
800
600
400
200
0

Fine-tuning - V1.6.6

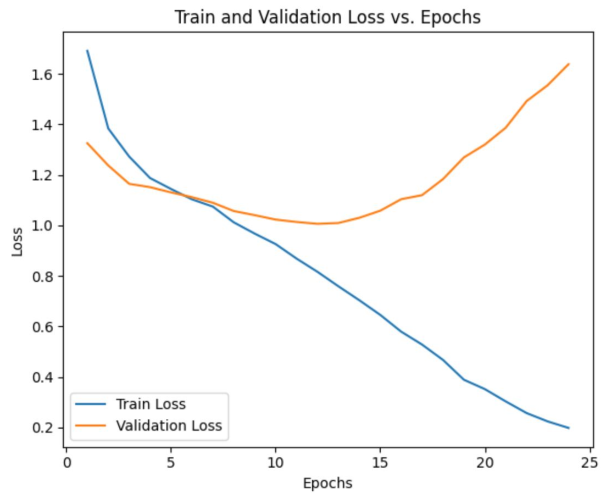
```
train_dataset = datasets.ImageFolder(root='archive/train', transform=transform)
test_dataset = datasets.ImageFolder(root='archive/test', transform=transform)

class_counts = {"angry": 3995, "disgust": 436, "fear": 4097, "happy": 7215, "neutral": 4965, "sad": 4830, "surprise":3171}

weights = [1.0 / counts for counts in class_counts.values()]
labels = train_dataset.classes
sample_weights = [weights[label] for _, label in train_dataset.samples]

# Create sampler and DataLoader for training
sampler = WeightedRandomSampler(sample_weights, len(sample_weights))

train_loader = DataLoader(dataset=train_dataset, batch_size=32, sampler=sampler, shuffle=False)
test_loader = DataLoader(dataset=test_dataset, batch_size=32, shuffle=False)
```

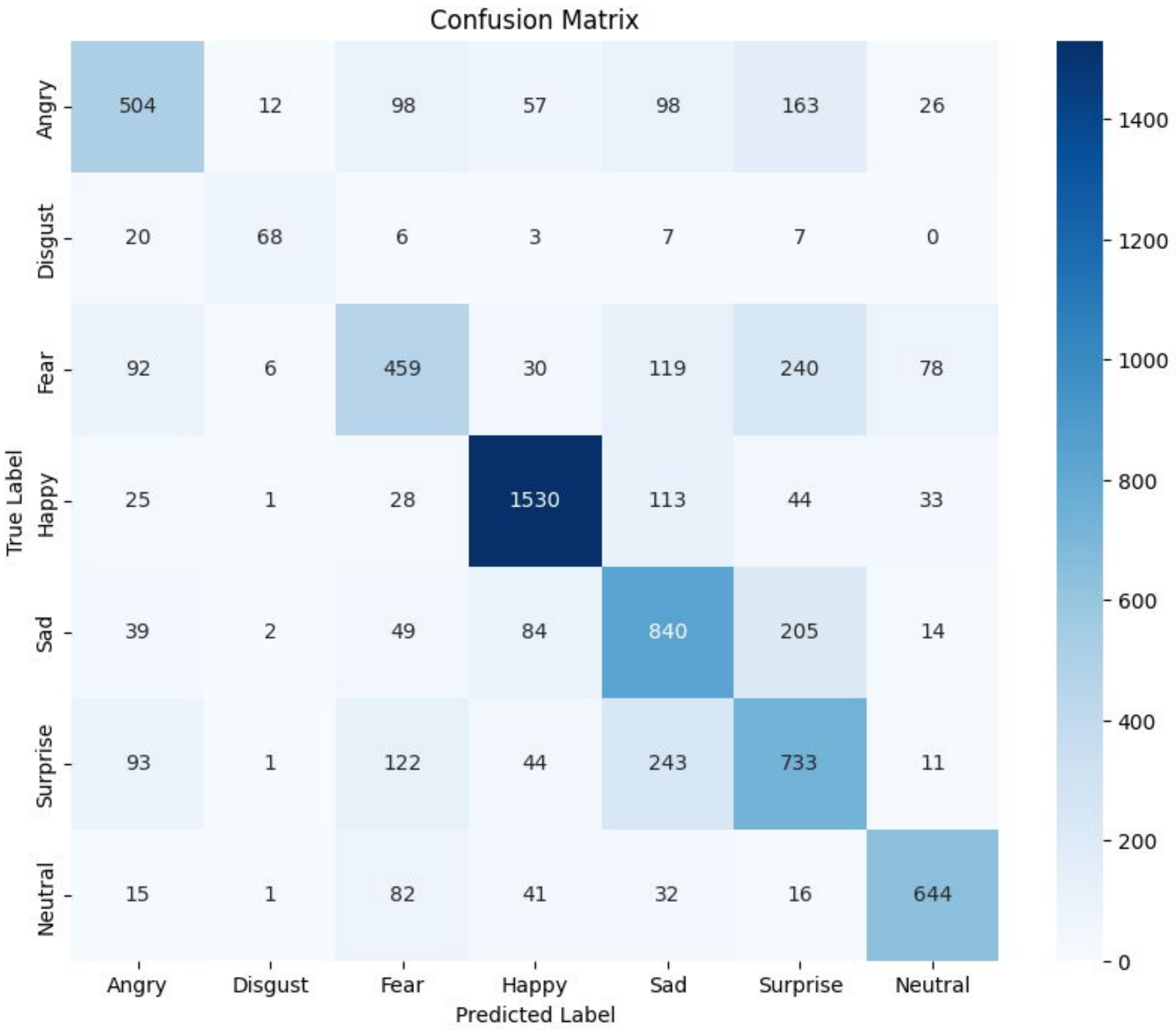
Accuracy: 0.634

24 epochs

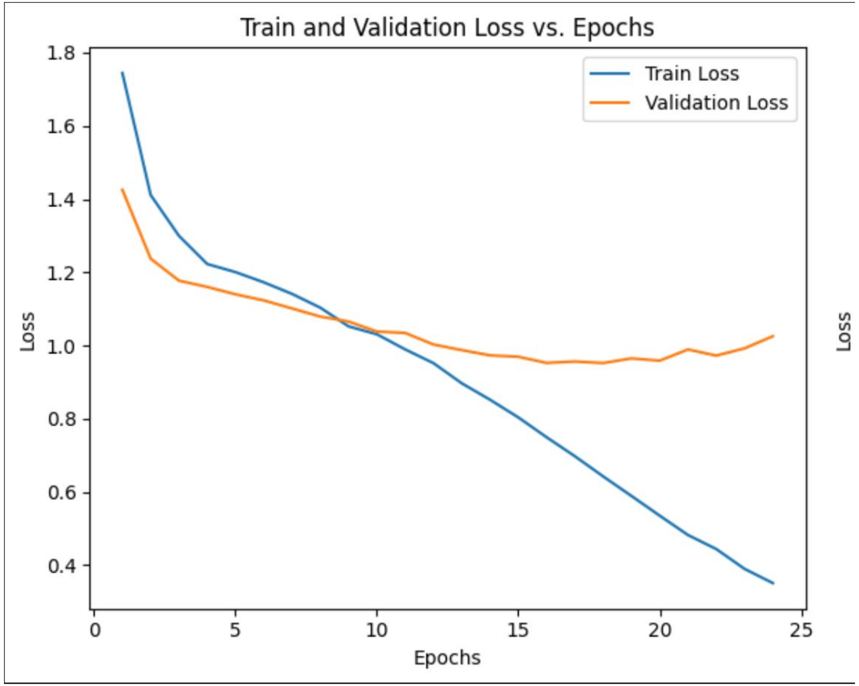
Used dataset upscale

Predicts “disgust” much better




Confusion Matrix
True Label
Predicted Label
Angry
Disgust
Fear
Happy
Sad
Surprise
Neutral
504 12 98 57 98 163 26
20 68 6 3 7 7 0
92 6 459 30 119 240 78
25 1 28 1530 113 44 33
39 2 49 84 840 205 14
93 1 122 44 243 733 11
15 1 82 41 32 16 644
1400
1200
1000
800
600
400
200
0

Fine-tuning - V1.6.7 (Final version)

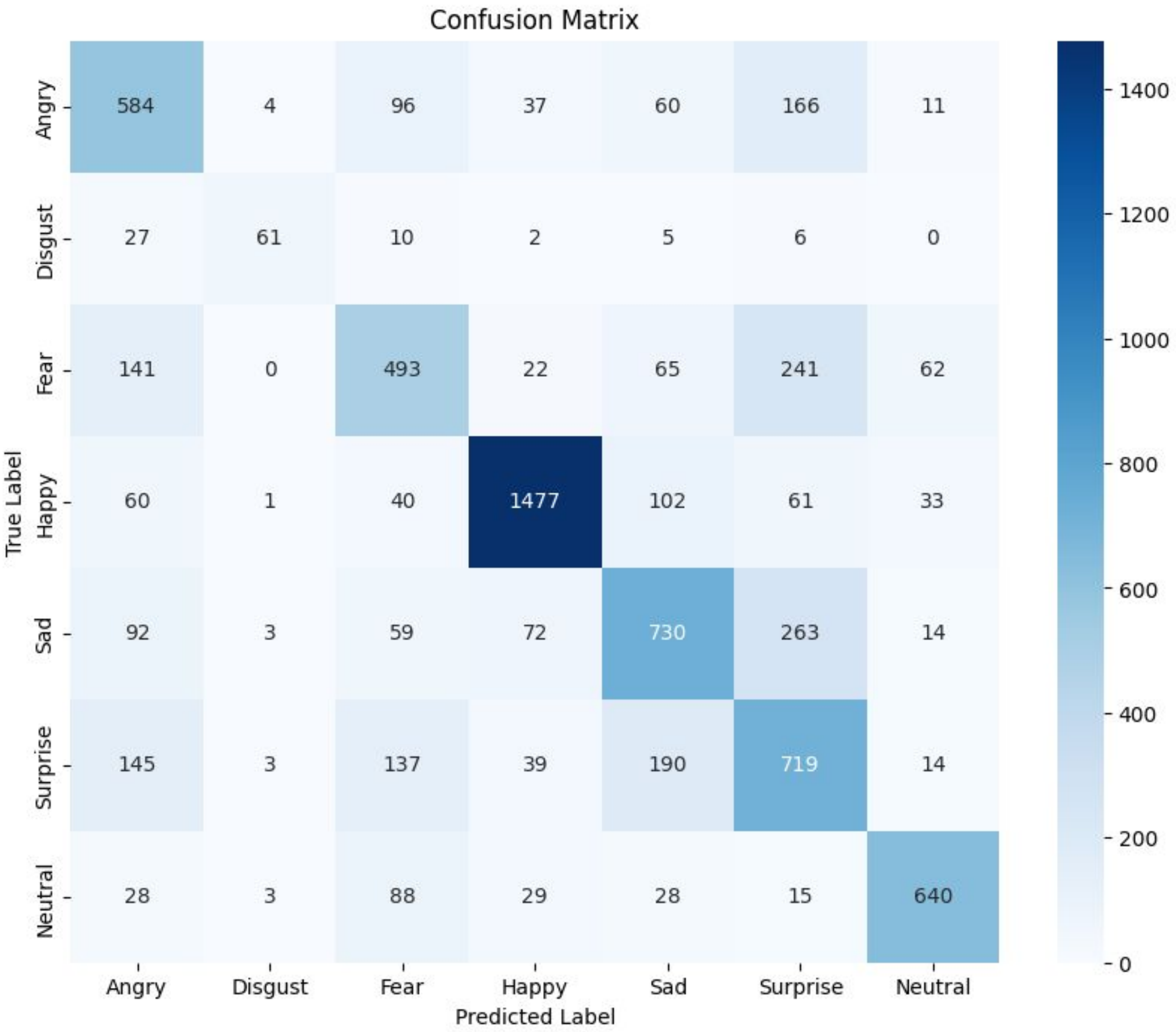


Accuracy: 0.663

24 epochs

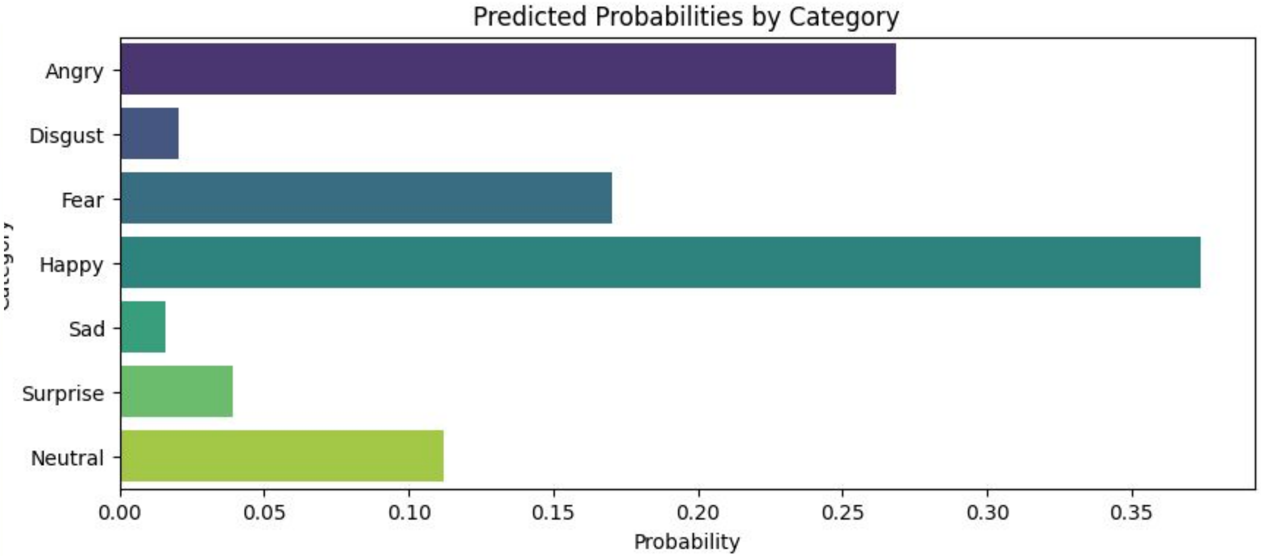
Added batch normalization layers

```python
class CustomClassifier(nn.Module):
    def __init__(self, input_dim, num_classes):
        super(CustomClassifier, self).__init__()
        self.fc1 = nn.Linear(input_dim, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 128)
        self.fc4 = nn.Linear(128, 64)
        self.fc5 = nn.Linear(64, num_classes)
        self.bn0 = nn.BatchNorm1d(input_dim)
        self.bn1 = nn.BatchNorm1d(512)
        self.bn2 = nn.BatchNorm1d(256)
        self.bn3 = nn.BatchNorm1d(128)
        self.bn4 = nn.BatchNorm1d(64)
        self.dropout = nn.Dropout(0.5)
        self.relu = nn.ReLU()

    def forward(self, x):

        #x = self.bn0(x)

        x = self.fc1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.dropout(x)

        x = self.fc2(x)
        x = self.bn2(x)
        x = self.relu(x)
        x = self.dropout(x)

        x = self.fc3(x)
        x = self.bn3(x)
        x = self.relu(x)
        x = self.dropout(x)

        x = self.fc4(x)
        x = self.bn4(x)
        x = self.relu(x)
        x = self.dropout(x)

        x = self.fc5(x)
        return x
```

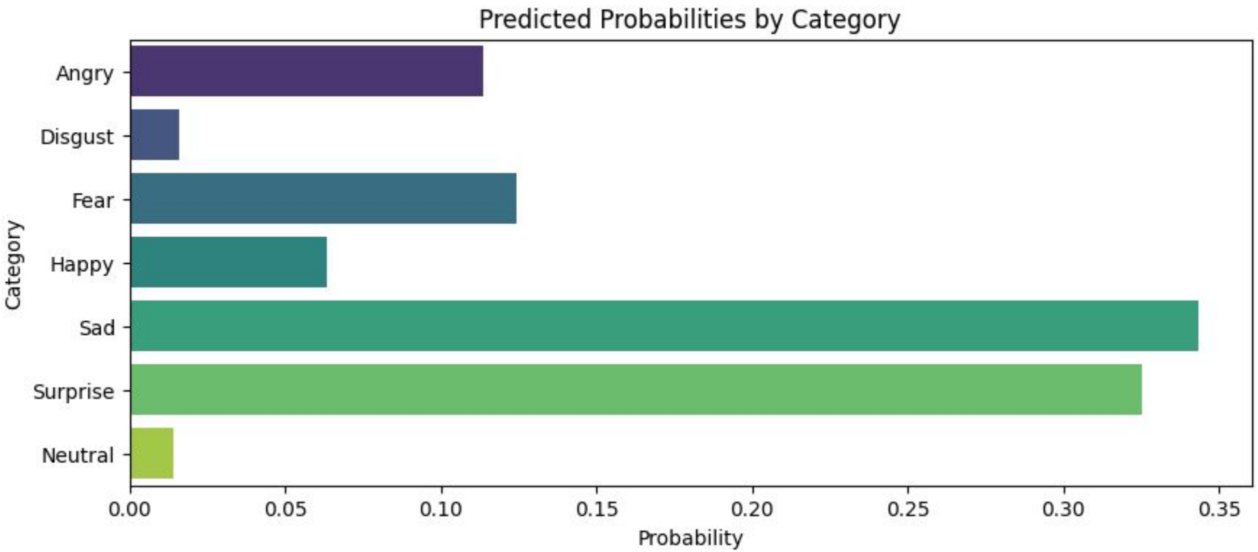

Confusion Matrix
True Label
Angry
Disgust
Fear
Happy
Sad
Surprise
Neutral
584 4 96 37 60 166 11
27 61 10 2 5 6 0
141 0 493 22 65 241 62
60 1 40 1477 102 61 33
92 3 59 72 730 263 14
145 3 137 39 190 719 14
28 3 88 29 28 15 640
Angry
Disgust
Fear
Happy
Sad
Surprise
Neutral
Predicted Label
1400
1200
1000
800
600
400
200
0

Generalization

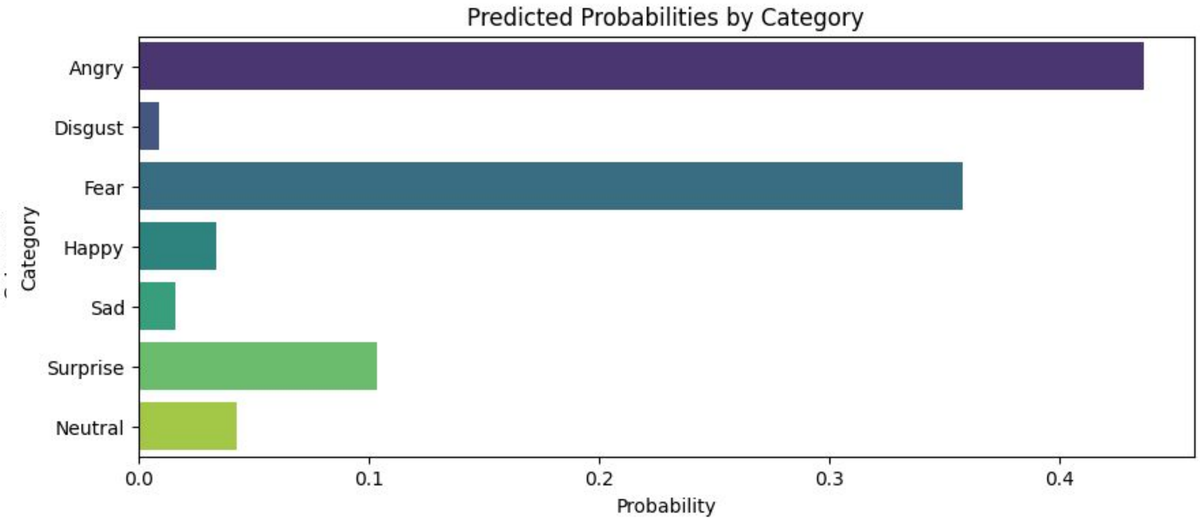
Input Image



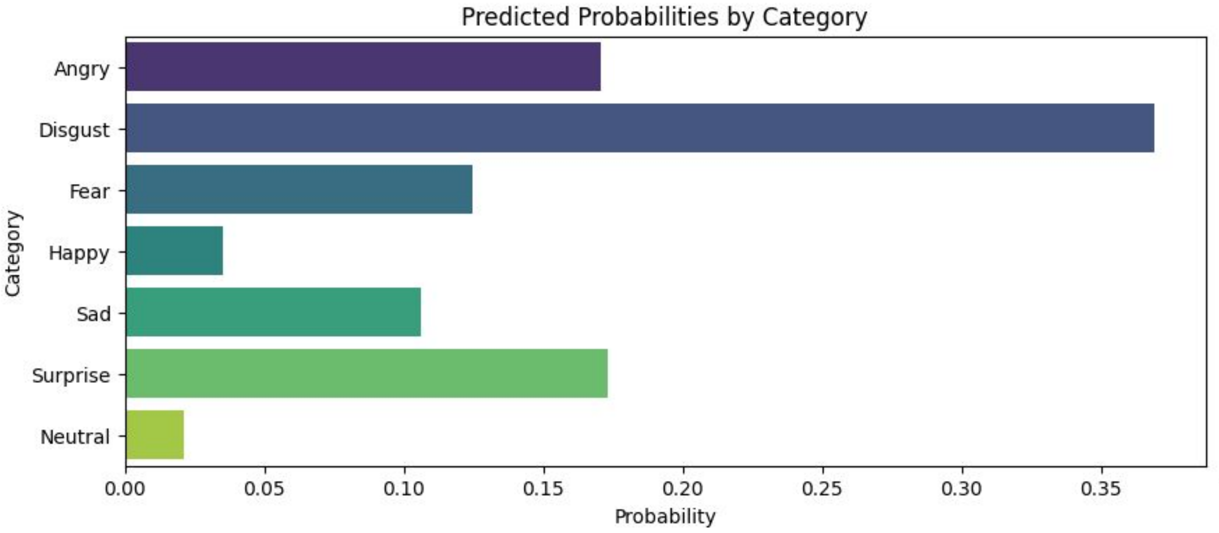
Input Image


Predicted Probabilities by Category
Angry
Disgust
Fear
Happy
Sad
Surprise
Neutral
Category
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Probability

Input Image


Predicted Probabilities by Category
Angry
Disgust
Fear
Happy
Sad
Surprise
Neutral
Category
0.0
0.1
0.2
0.3
0.4
Probability

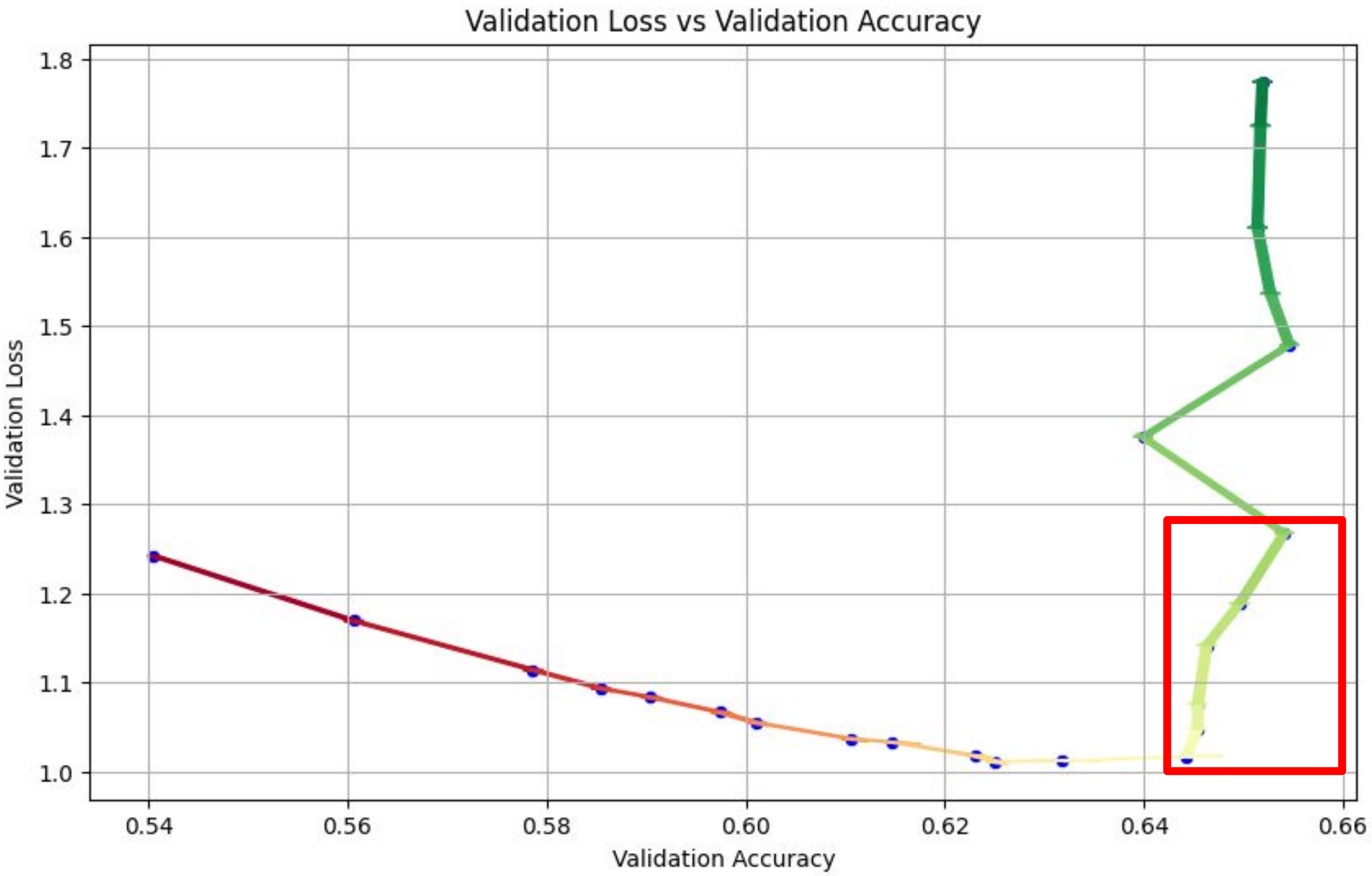
Input Image



One potential application of the model:

Understand your partner :-)

Interesting Discovery



“Loss can be seen as a **distance** between the true values of the problem and the values predicted by the model. The larger the loss, the larger the errors you made on the data.

Accuracy can be seen as the **count** of mistakes/ misclassifications you made on the data. The larger the accuracy, the fewer misclassifications you made on the data.”

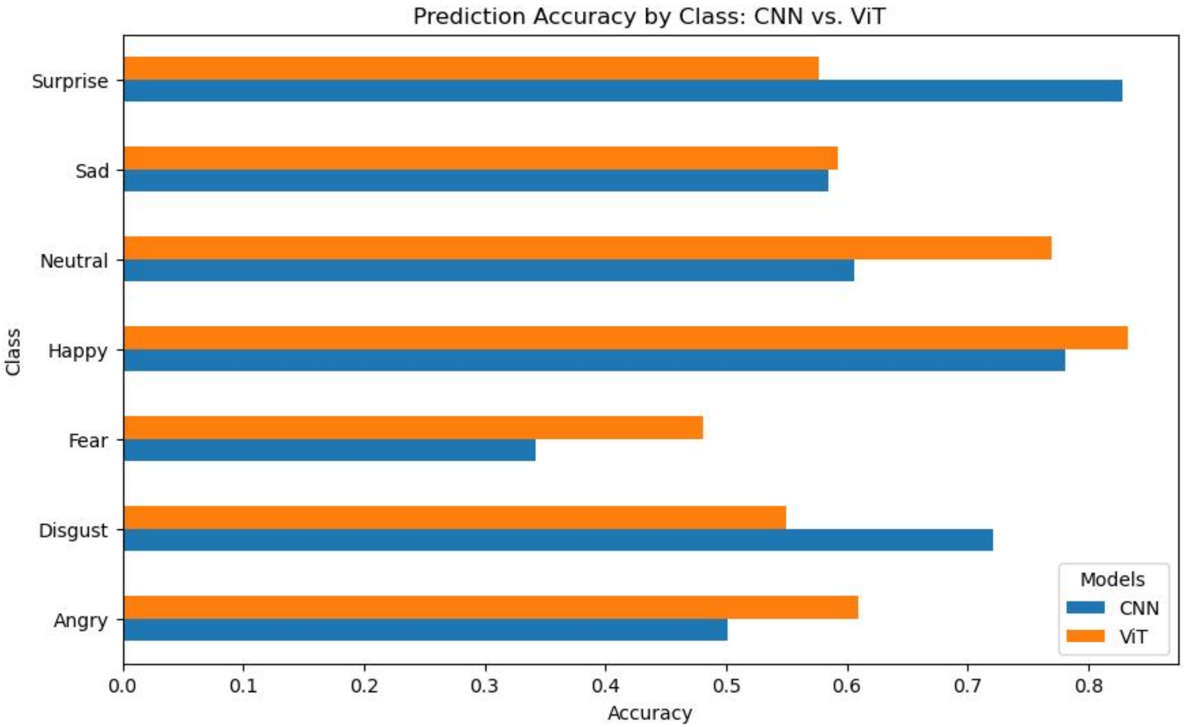
https://datascience.stackexchange.com/questions/42599/what-is-the-relationship-between-the-accuracy-and-the-loss-in-deep-learning

Conclusion

We can see that the ViT performs better in 5 out of the 7 categories

However, the CNN does better in categories with fewer training samples (e.g. disgust)

Overall, it may be better to use a CNN when there is a small amount of training data/small image resolution, and a ViT otherwise



Note: The ViT is pre-trained on color images, while the test dataset is grayscale, this might be the cause of lower accuracies

Conclusion Continued

Potential problems with results:

- Google's ViT is pretrained and fine tuned on more data than our CNN architecture
    - unfair conclusion to say Vision Transformers are better than CNNs for this task

- The ViT might be performing worse since it was pretrained on higher resolution, colored images

Note: The ViT is pre-trained on color images, while the test dataset is grayscale, this might be the cause of lower accuracies

Future Works for ViT Model

How to combat overfitting?

- Freeze the pre-trained layers
- Use L2 regularization (weight decay)
- Adjusting the dropping out rate
- Use other pretrained models

Future Works

- Compare pre-trained CNN models to our ViT for a more fair comparison

- Find colored, upscaled image dataset for emotion recognition and rerun experiments

Q&A