MoodMirror: Real-time Emotion Estimation

Daniel Paliulis, Tyler Hinrichs, Vihaan Shah, Gautam Pirthiani

CSE 5717: Big Data Analytics

Dr. Wei Wei

December 4, 2023

Table of Contents

Problem statement

Emotions play a crucial role in human interaction. Understanding them is fundamental in establishing meaningful relationships and productive communication. For many individuals, deciphering these emotional cues is challenging, especially for those with social or cognitive disorders. For example, some individuals with autism spectrum disorder (ASD) have varying degrees of difficulty in differentiating between neutral and emotional facial expressions. Such difficulty ultimately leads to the misclassification of emotional expressions (Eack et al., 2015). In addition to the challenges faced by individuals with social or cognitive disorders, the complexity of emotional interpretation is further intensified by the diverse range of cultural expressions and subtle nuances that vary across societies. This complexity makes it harder for some individuals to accurately read and respond to emotional signals. The impact of emotional misinterpretation is not confined to those with cognitive disorders; it applies to various aspects of daily life, influencing professional relationships, educational environments, and social interactions. Hence, there is a strong need to develop tools and technologies that can aid in bridging the emotional communication gap for a broader spectrum of individuals, thereby fostering inclusivity and understanding across diverse cultural and social contexts.

In the professional realm, the ability to comprehend and respond appropriately to colleagues' emotional states is crucial for team dynamics, collaboration, and overall workplace well-being. Miscommunication stemming from emotional misinterpretations can lead to misunderstandings, decreased productivity, and, in some cases, contribute to a less conducive work environment. Thus, addressing the challenges associated with emotional communication has far-reaching implications, not only on an individual level but also in promoting healthier and more efficient interactions within various societal domains. In the educational context, teachers' understanding of students' emotions is pivotal for creating an empathetic and supportive learning environment. Students who face difficulties in expressing or recognizing emotions may encounter barriers to effective communication with peers and educators. As such, integrating real-time emotional feedback systems in educational settings can contribute to fostering a more inclusive and accommodating atmosphere, optimizing the learning experience for all students.

The goal of our project is to develop an emotion estimation model that can be used to address these challenges posed by social, cognitive, cultural differences. Our primary aim is to create a model that enhances emotional communication, particularly for individuals facing challenges in interpreting and expressing emotions. We are developing a model that can efficiently classify facial expressions into specific emotional categories. To achieve an accurate emotion estimation model, we paid close attention to data preparation and model selection. Through systematic exploration of different models and incorporating advanced data preparation techniques, our goal is to get the model to excel in estimating diverse emotions.

Once a model has been finalized, we start building our interactive front-end. The front-end will utilize the user's webcam to capture facial images, these images will be then processed to obtain a cropped image of the person's face and then this image passes through our model to generate a prediction which according to our model will be one of these: Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral.

The evaluation of our model will be conducted through a set of well-defined performance measures ensuring a thorough understanding of the model's capabilities and limitations. These performance measures have been thoughtfully chosen to cover various facets of the model's performance, including accuracy, generalization, and its ability to discern nuanced emotional expressions. Our evaluation strategy incorporates both quantitative accuracy scores and qualitative visualizations. For quantitative metrics we will be using accuracy scores which includes training, validation, and test accuracy, and then loss metrics will be used. For qualitative metrics we will be using visualizations that offer insights into the model's learning patterns and potential overfitting issues.

Data set description

The dataset titled FER2013, is sourced from Kaggle and primarily consists of labeled grayscale images depicting facial expressions corresponding to distinct emotions. Each image has been centered and resized to occupy a uniform space, facilitating consistency in the dataset. The dataset is organized into training and testing sets, with a further division of the training set into training and validation subsets, maintaining an 80-20 ratio.

Data Set: https://www.kaggle.com/datasets/msambare/fer2013
Dataset Specifications:

- Number of Images: The FER2013 dataset comprises a total of 35,887 grayscale images.
- Image Attributes: Each image is 48 by 48 pixels, representing facial expressions in grayscale.
- Emotions Covered (Labels): The dataset encompasses seven emotional categories, each labeled from 0 to 6, corresponding to the emotions Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral, respectively.
- Dataset Split: The dataset is categorized into training and testing.

Example of an image from the "Happy" class (enlarged from 48x48 resolution):

Data preparation

Partitioning

When creating our initial implementation, we used the raw FER2013 dataset to train our model. This data was already partitioned into a training and testing set based on an 80-20 partition. That is, about 80% of the images for each class were partitioned for training, and the remaining 20% were reserved for testing. We used the testing partition in our initial implementation as a validation set in order to evaluate the model and choose the most accurate one. However, for subsequent models, we chose to expand upon this partitioning to include dedicated training, validation, and testing. This partitioning was based on a 70-10-20 split, where 70% of the data was used for training, 10% was used for testing, and 20% was used for validation. Having these sets separate from each other is imperative to maximizing the quality of our training. Fittingly, we train our model on the training data, whereas the validation data is used to check how the model performs on generalized data that it was not trained with. We can set parameters to prioritize maximizing validation accuracy to make sure our model is not overfitting to the training data. However, though the model is not directly trained on the validation data, the model constantly gets tuned based on its performance on the validation set, which can have the effect of the model overfitting to the validation set itself. The test set is completely isolated from the model during training, and can therefore give us a more accurate representation of how the model might perform on real data (Chollet, 2021, pp. 133-134).

Feature Scaling

Feature scaling is applied to our dataset through the form of a rescaling layer at the beginning of our neural network. This applies a 1/255 scale to every pixel in each image entering the neural network. This results in the value for each pixel in each image being within the range [0, 1]. We perform this scaling in order to normalize the pixel values for each image. During attempts to leverage fine tuning the VGG19 pre-trained model, we perform this scaling through the vgg19.preprocess_input function from the Keras library. For the EfficientNet pre-trained models, this process is done automatically without any need to call external-facing functions.

Alterations to the Dataset

One of the largest hurdles with this project was with issues in the dataset. The largest issue in particular being that the data was fairly unbalanced, with each class containing different amounts of images. This distribution was 4,953 angry images, 547 disgust images, 5,121 fear images, 8,989 happy images, 6,198 neutral images, 6,077 sad images, and 4,002 surprise images. The most troubling parts of this being that the amount of happy images was nearly double that of all the other classes, and the amount of disgust images was approximately 7 times smaller than the next smallest class of images. With so many happy images, the neural network will learn that its loss and accuracy are improved by preferring to classify images as happy images.

Additionally, with so few disgusted images, the neural network will learn to reluctantly classify images as disgust. It is important to note that upon further research on this dataset, it appeared that it was somewhat notorious for producing poor results. Most models that are considered "well performing" would have test results within the range of 60% to 70%. The historically best model was achieved by Stanford University researchers, which achieved a test accuracy of 75.8% (Khanzada et al., 2020). We leveraged several methods to mitigate the balancing issues within the dataset.

Our initial approach was to reduce the happy dataset and to increase the size of the disgust dataset. We reduced the happy dataset by randomly selecting images from the raw dataset such that the size of the happy dataset matched that of the neutral dataset. The disgusted dataset was expanded through downloading "disgusted face" google image results and leveraging the Haar-Cascade feature detection algorithm to crop out the faces of the image. This approach was ultimately abandoned given that the results for google images were not reliably matching the "disgust" classification. Additionally, there were not enough google image results to scale up the number of disgust images to be close to the other classes.

The next approach was to remove the disgusted class altogether. This altered the project to be a classification model on the 6 classes of angry, fear, happy, neutral, sad, and surprise. In this approach, the data was balanced further by randomly removing images from the angry, fear, happy, neutral, and sad classes to be a size of 4,002 images each. This improved the reliability of the model on classifying each of the 6 emotions with no bias to one class. During a fit with this data, our model achieved 68% training accuracy and 61% validation accuracy. While this testing accuracy met the benchmark for most models trained under the FER2013 dataset, removing training data was ultimately a bad decision for the model. This is because removing data lowers the model's ability to generalize its predictions.

Our final approach involved leveraging data augmentation to inflate the data. Under this approach, we reintroduced the disgust class and reverted all classes to their original sizes. The data augmentation was performed using the Albumentations library. With this library, the following transformations were performed on the images within each class at random with a probability of 0.7: horizontal image flipping, rotation within the range [-30°, 30°], zooming at a factor within the range [0, 55%], vertical and horizontal translation within the range [-10%, 10%], change in brightness within the range [-30%, 30%], and change in contrast with the range [-30%, 30%]. These transformations were randomly applied evenly to the images in each class until the number of images within each class was 8,989 images. This successfully inflated the size of the dataset such that each class had equal representation of images during training.

Loading and Preprocessing the Data

Given that we are working with a large amount of image data (in the magnitude of tens of thousands of images), we decided to store our data in a zipped directory stored in the cloud through Google Drive. Then, we can access our data by connecting to Google Drive, unzipping the appropriate directories, and then importantly, using the image_dataset_from_directory() API from keras in order to preprocess our data.

Image_dataset_from_directory() returns a Keras Dataset object, an iterator that returns preprocessed image data from a directory that we specify. There are several parameters that we must specify in order to properly preprocess the data for our use case. In addition to the directory, we can also specify the image size (in our case, using the tuple (48, 48) to indicate that our images are 48x48 pixels). Furthermore, we set label_mode to "categorical" to indicate that the data is categorical; i.e., there are subdirectories holding images from different classes that should be parsed accordingly. This automatically one-hot encodes the classes, a technique that assigns each data point a tensor of the length of the number of classes, with all values 0 except a 1 in the spot that corresponds for the specific class that the data point is a part of. This ensures that class representations equidistant tensors to each other, avoiding false associations that can occur from classes being listed in an incremental format. The latter method is known as sparse encoding, where classes a, b, c might correspond to the values 1, 2, 3, etc.; this can cause neural networks to falsely pick up on associations that classes with similar numerical representation are more closely related than pairs with more different representations. In our example of facial emotions, we assume class independence and hence use one-hot encoding.

In most cases, we set the color_mode to "grayscale" to correspond to our dataset's native format. However, when using pre-trained models such as VGG19 and EfficientNetB0, these models accept colors with rgb values instead of grayscale values; in this case, we can set color_mode to "rgb". In terms of how the model's will accept this image data, a color image the tuple (48, 48, 3) represents RGB whereas (48, 48, 1) represents grayscale, as 3 stands for the three color channels of RGB, whereas 1 stands for the single color channel of grayscale. Finally, we set batch_size to 32, which is the number of images that are processed per iteration within each epoch (Chollet, 2021, p. 35).

Model and/or technique selection

Convolutional Neural Networks Background

Convolutional Neural Networks (CNNs) are the primary model used for processing large images efficiently. CNNs use convolutional layers, which have properties that are far more useful to the task of computer vision than other models. For one, convolutional layers provide a means to learn local patterns in an image rather than global patterns. These learned patterns are

translation invariant, meaning that they can be recognized anywhere in an image regardless of location. CNNs also maintain the notion of spatial hierarchy, where more complex and abstract concepts are constructed by simpler patterns. Hence, CNNs focus on low-level features in early layers, which they combine into higher-level features in later layers. This hierarchical structure is a common feature of actual images, making the algorithm apt for image processing. CNNs also provide a means to process data in the second dimension. This preserves spatial relationships in the image which may be a determining factor during evaluation.

Convolutions operate on rank 3 tensors, called feature maps. These tensors contain two spatial axes for the height and width of the input, and one depth axis for the channels of the input. After a convolutional layer, the tensor depth is expanded based on the number of filters in that convolutional layer.

In a CNN, neuron's weights are called filters, or convolutional kernels. These filters are small matrix patches that contain a pattern learned from the training data. The filters are applied to an image by sliding across the image at strides(Chollet, 2021, pp. 226-262). At each overlapping region, a dot product is performed, resulting in a one dimensional vector of the size of the output depth. Each of these vectors are combined to create a new tensor. In our models, we used filters of size 3x3 pixels at a stride of 1 pixel. We also use "same" padding to ensure that the outer border of the image is not lost in the resulting tensor (Chollet, 2021, pp. 226-262).

After the convolutions, an activation function is applied to the output to aid in determining if the pattern was present in the image. In all of our models, we use the rectified linear unit (ReLU) function, which sets all negative pixels to zero. The ReLU function also preserves non-linearity, allowing for such a complex evaluation (Kumar, 2020). During the training, CNNs learn the best filters for accurate predictions. Patterns become more complex the deeper into the model, forming the hierarchical structure (Chollet, 2021, pp. 226-262).

Along with the convolutional layers in a CNN, we employ pooling layers to transform the data. The purpose of the pooling layer is to decrease the dimension of the feature map and eliminate redundant details (Ramya et al., 2022). This is important considering that convolutional layers record the precise position of the features in the input. Thus, alterations to the image, such as shifting or rotation, will result in a different feature map. The pooling layers can mitigate this through downsampling, where the spatial dimensions of the input are decreased (Chollet, 2021, pp. 226-262). This eliminates the fine details while maintaining the important structural elements. Pooling occurs after the ReLU has been applied to the feature map output of the convolutional layer. With max pooling, the window moves along the input usually at a stride of 2, extracting the largest value as it builds the output tensor. This implementation decreases the size of the spatial axes in the feature map by 2. Pooling is an operation that is specified rather than learned within the model (Brownlee).

We took two main approaches when implementing the CNN for our task. One approach was creating a neural network from scratch for which we would implement our own architecture and train from random weights. The other approach was to leverage a pre-trained model, fine tuning its top-most layers to predict on our data.

Original "From Scratch" Model

Our first model, which we will call the basic model, comprises multiple convolutional layers of increasing numbers of filters for each subsequent layer. There are four convolutional layers in total, with the number of filters being 32, 64, 256, and 512 respectively. This allows the model to learn simple patterns at first, gradually accounting for more complexity as time goes on. There are also max-pooling layers to reduce spatial dimensions and mitigate overfitting. Each max pooling layer has a window size of 2x2 pixels. To enhance the network's generalization capabilities, dropout layers have been added. We have three dropout layers dropping a fraction of 0.1 units, and one dropout layer dropping a fraction of 0.2 units. Generalization from dropout allows for random neurons to be ignored, helping with avoiding overfitting. After the convolutions, we have a layer to flatten the data. This outputs to two fully connected dense layers. The first layer has 512 neurons with ReLU activation. The final player contains seven neurons, each corresponding to a distinct emotion class, with the softmax activation function to output a probability distribution. We chose the Adam optimizer for its efficiency, and used categorical cross-entropy as the loss function to train the model for multi-class classification. This model was trained for 50 epochs, and the model was saved at the end of training.

Advanced Model Implementation

Building off of the basic model, we built the advanced model. This model takes advantage of the CNN architecture principles outlined by François Chollet in "Deep Learning with Python Second Edition". One convnet best practice that was implemented in this model was modularizing the network into blocks of convolutional layers and max pooling layers. These blocks would consist of two convolutional layers with the same number of filters, and one max pooling layer. Modularization provides the ability to reuse the blocks with different hyperparameters, keeping the model organized as the hierarchy deepens (Chollet, 2021, pp. 274).

Modularizing the network into blocks allows us to create the pyramid-shaped feature hierarchy in this model. This involves organizing the convnet such that the number of filters grows with the depth of the network, while the size of the feature maps decreases. Using multiple scales of feature maps provides a means to capture information at different levels of abstraction. This allows the network to learn more efficiently by reducing the number of parameters that need to be learned while accurately identifying relevant patterns (Chollet, 2021, pp. 275). Our feature hierarchy involved increasing the feature depth with the number of filters at 32, 64, 128, 256, and 512. We also used a max pooling layer with a 3x3 pixel window moving at a stride of 2 in order

to decrease the feature map size. This created a deep and narrow model which is beneficial in encouraging feature reuse (Chollet, 2021, pp. 274).

One issue presented by creating such a deep model is vanishing gradients. At each iteration of training a neural network, each weight receives an update proportional to the partial derivative of the error function with respect to the current weight, the gradient. The problem is that in some cases, the gradient will be very small and even vanish from the model, effectively preventing the weight from changing its value during backpropagation. Due to vanishing gradients, increasing the complexity of an CNN causes information to vanish in the model, hindering learning in early layers of the neural network (Chollet, 2021, pp. 276-277).

We mitigated the issue of vanishing gradients by introducing residual connections to our model. Residual connections involve retaining a noiseless version of the information contained in the input before passing the data through destructive blocks like those that perform dropout and ReLU. This residual is then added back to the data after it travels through the block of layers. In turn, this forces the operations to be nondestructive, allowing error gradient information from early layers to propagate noiselessly through the CNN (Chollet, 2021, pp. 277). It is important to note that, in order to add the residual with the data output from the block, we need to reshape the residual to match the dimensions of the output. This involves leveraging a convolutional layer with a stride value greater than 1 to downsample the residual connection to match that of the output from the max pooling layer (Chollet, 2021, pp. 278). In our model, since the max pooling layer moves at a stride of 2, we set this convolutional layer to move at a stride of 2 in order to reduce the dimensions by half.

One other component within the advanced model was the batch normalization layers. Normalization is a technique for standardizing the data by centering it around zero by subtraction by the mean and dividing by the standard deviation, creating the assumption of a gaussian distribution. This aids in training speed and in generalizing the model. The difference with deep learning is that we want to normalize the data after every transformation operated by the network. Thus, the goal of batch normalization is to normalize the data as the mean and variance of the data change throughout training. This is done through evaluating the mean and variance of the current batch of data in order to normalize the samples (Chollet, 2021, pp. 280-281). Similar to residual connections, this aids in alleviating the vanishing gradient issue when creating deep neural networks. It is important to note that since batch normalization centers the data around zero, a bias vector is not needed when performing convolutions. Hence, we set the “use_bias” parameter to “False” (Chollet, 2021, pp. 280-281).

Another change that we made when moving from the basic model to the advanced model was changing the convolutional layers to depthwise separable convolutional layers. Depthwise separable convolutions process data such that the channel features and spatial features are

learned separately. This relies on the assumption that spatial locations are highly correlated, while different channels are highly independent from each other (Chollet, 2021, pp. 282-283), which is generally the case with image data. Thus, depthwise separable convolutions perform a spatial convolution on every channel in the input independently, then combining the input channels into a pointwise convolution. Note that a pointwise convolution is one with a kernel of size 1x1 pixels. These convolutional layers provide the benefit of requiring fewer computations and parameters while being as good as, if not better than, regular convolutional layers. Due to the smaller parameter size, models with separable convolutional layers will tend to converge faster and overfit less (Chollet, 2021, pp. 280-281).

Apart from the advanced computer vision hierarchy, we also added a data augmentation layer to the beginning of the advanced model. This layer performs random transformations on the incoming data with the goal of diversifying it. Our data augmentation layer performs random horizontal flips, random rotations at a factor of 0.1, and random zoom at a factor of 0.2. Performing this data augmentation has the benefit of improving the generalization and reducing overfitting during training.

Altogether, the advanced model is composed of several layers, creating a deep neural network with a pyramid structure. This begins with a data augmentation layer, which applies random transformations to diversify the data. A rescaling layer then alters the image pixel values to be within the range [0, 1]. We then have a convolutional layer to preprocess any highly correlated features present in the data (this is more relevant for images with color). This is followed by 5 blocks of batch normalization, separable convolutional layers, and additions of residual connections. Within these blocks, we perform the ReLU activation function to achieve nonlinearity. The separable convolutional layers within these blocks have the filter amounts of 32, 64, 128, and 512 respectively. There is finally a global average pooling layer, followed by a dropout layer for regularization. The output layer of our model is a dense layer with 7 neurons, each corresponding to the emotion classes, using a softmax activation function. Note that of all of the models discussed in this paper, the advanced model had the highest test accuracy at 66.68%.

Pre-trained Model Approach

Another technique that we investigated was applying pre-trained models to the task, both “as-is” through feature extraction as well as fine-tuning them in order to get as high-quality results as possible. The process involves using deep learning models with predefined architectures that have been previously trained on large datasets (Keras, 2023). Using Keras through TensorFlow, we can get immediate access to these models and can apply techniques for feature extraction and fine-tuning.

There are several major benefits of using pre-trained models. First, many pre-trained models use highly researched and effective architectures, several of which we will explore later in the context of VGG19 and EfficientNet. Beyond just high-quality architectures, many pre-trained models have been trained on large datasets, so they already have learned visual hierarchies that can be applied to new problems. The ability to use and apply previously-learned features can often make training more efficient and effective, especially in the context of smaller to mid-sized datasets (Chollet, 2021, pp. 224-225).

There are numerous pre-trained models at our disposal through Keras. For example, Keras lists 38 available models that can be accessed through the keras.applications API (Keras, 2023). When previously using our own model and training from scratch, we used an architecture similar to the architecture of the pretrained model Xception, both of which use convolutional layers of increasing sizes amongst many other architectural choices. Ultimately, we decided to look at several new pretrained models in order to holistically search for the best model. All of these models have been trained on the ImageNet dataset, a huge dataset that has representations of many common objects. ImageNet corresponds to WordNet, a huge dataset of words grouped into synonym groups called “synsets”. ImageNet has tens of millions of images, with thousands for each of the noun synsets in WordNet, meaning that it attempts to capture general image information for the world and allow large neural networks to be trained with it (ImageNet, 2021).

VGG19

One of the first models we employed was VGG19, a pre-trained model with architecture based on a small filter size, alongside a depth of 19 layers that includes 16 convolutional layers and 3 fully connected layers (Simonyan & Zisserman, 2015). At the time of creation, this model leveraged a deeper structure than had been previously seen with similar architectures in the past. It was possible for this structure to leverage a high depth of layers due to the fact that the convolutional filters were small, and overall, other parameters were fixed; this included having standard increasing filter numbers (64, 128, 256, 512), max pooling layers, and dense layers (Simonyan & Zisserman, 2015). Interestingly, while the model architecture was introduced as having an increased depth compared to past state-of-the-art models, it actually has far fewer layers than many newer models. VGG19 has 19 layers (as its name subtly suggests), whereas many of the other available models through the keras.applications API have significantly more, even as high as 533 for NASNetLarge (Keras, 2023). However, VGG19 has one of the highest number of total parameters at about 143.7 million, which is in large part due to its fully connected layers at the top of the model; for instance, two of its dense layers contain 4096 neurons. In practice, we used the model with include_top set to false, which removes the densely connected layers. This reduces the total number of parameters from 143.7 million to about 20 million. Ultimately, while not being the newest model, this model had a simple yet powerful architecture that seemed worthwhile to attempt fine-tuning and evaluating.

EfficientNet

While VGG19 works well with certain datasets, we did not want to limit our evaluation to just one pre-trained image classification model, especially one that is considerably older than other notable recent models. When training with VGG19, we found that there was potential overfitting occurring (this will be discussed later). As a result, we wanted to try a new model (or perhaps multiple) that could address shortcomings with VGG19 while taking advantage of a new architecture. One set of models that stood out was the EfficientNet family, which seemed promising due to its usage of architectures that uses a revolutionary scaling method that improves efficiency while still maintaining a high level of accuracy on major datasets including ImageNet (Tan, M., & Le, Q. V., 2020). There are 8 models in the family, ranging from EfficientNetB0 to EfficientNetB7; EfficientNetB7 has the most parameters and hence the most complexity, but while still being 8.4x smaller and 6.1x faster than the best CNN applied to ImageNet, was able to achieve 84.3% top-1 accuracy on ImageNet (Tan, M., & Le, Q. V., 2020), a score that already puts it far beyond VGG19 at 71.3% (Keras, 2023). EfficientNet purposeful minimization of parameters alongside its prioritization of maintaining complexities makes them generalize well on various image datasets, making them potentially suitable for our facial emotion recognition task. Having been trained on the vast ImageNet, these models have seen a wide variety of patterns that can be reused when training on our facial image dataset.

EfficientNetB0

The first pre-trained model from the EfficientNet family we used was EfficientNetB0. It is the most lightweight EfficientNet model in the family, an efficient convolutional neural network architecture that has a good balance between accuracy and computational efficiency. We chose EfficientNetB0 to start for its simplicity compared to larger variants while still maintaining good accuracy, something that we thought could mitigate previous overfitting. Without its topmost dense layers, it only has about 4 million parameters, as opposed to 20 million for VGG19 without its respective top. As will be explored later, this dataset seemed to underfit, so we decided to proceed to a more complex model in its family in order to learn more complex patterns in our data.

EfficientNetB5

To counter underfitting, we went with EfficientNetB5 to explore a more complex architecture while still maintaining a level of computational efficiency. We chose this to investigate whether a more intricate architecture can capture finer details and nuances present in facial emotion expressions. While more complex than B0, EfficientNetB5 strikes a balance by being less resource-intensive than larger variants like B6 and B7 (Keras, 2023). This model has about 28 million parameters, putting it even larger than VGG19. Interestingly, while the depth of VGG19 is only 19 layers, EfficientNetB5 has 312 layers (whereas EfficientNetB0 had 132, still many more than VGG19) (Keras, 2023). Results of this model's effectiveness will be discussed later.

Feature Extraction and Fine-tuning with Pre-trained Models

There are several techniques that can be used to leverage the power of pre-trained models. It is first worth noting some key characteristics about pre-trained models and CNNs themselves. First, these models have been trained on the ImageNet dataset, which is vast in both the total number of images as well as the number of classes it has. All image classification CNNs have a series of convolutional layers (also including down-sampling, normalization, etc.), followed by a densely connected classifier at the top of the model (Chollet, 2021, p. 225). At more shallow layers, patterns found are more general in nature, and get specific as layers get deeper; the dense layers are the most specific to a given dataset, with the dense layers losing the spatial information that the convolutional layers provide (Chollet, 2023, pp. 225-226). This makes them less applicable to a problem when the image data the model is trained on is not contained within the dataset that the convnet was trained on; in our case, the ImageNet dataset does not contain comprehensive facial images, so we decided to remove the topmost, dense layers from all models that we used.

The first technique that we employed was fast feature extraction. From a high level, we use a pre-trained model as a convolutional base and train our data with it, after passing the output from the pre-trained model through our own dense layers (including dropout). This technique essentially works by passing each image through the pre-trained base once, as we are simply predicting the outputs based on the pre-trained model, and then training just our dense classifier (Chollet, 2021, p. 228). This is the most efficient method, but can be potentially limiting; it doesn't allow us to use data augmentation in the model itself, and doesn't allow for fine-tuning. Still, this gives us a good idea on how the pre-trained model works on the data with minimal adjustments or additions.

The next technique we employed was fine-tuning the pre-trained models. There are several main steps that are important to follow in this technique, and they are as follows:

1.) Create a new CNN and add it on top of a pre-trained model.
2.) Freeze the pre-trained model.
3.) Train just the newly added CNN.
4.) Unfreeze parts of the pre-trained CNN, allowing them to be trained.
5.) Train both the new CNN we added alongside the unfrozen layers of the pre-trained model.

(Chollet, 2021, p. 234)

In step 5, we give a small degree of freedom for the pre-trained model to be modified, alongside our added CNN, which we have already trained; this process allows for fine-tuning our added layers alongside some of the layers of the pre-trained model to our dataset itself, leading to better results in many cases. We employed this strategy on all pre-trained models used.

Evaluation

We will use several techniques to investigate the approaches used. First and foremost, we will observe various accuracy scores that represent the model's training. We will have three accuracy scores readily available to us: training accuracy, validation accuracy, and test accuracy. As mentioned previously, these accuracy scores all serve unique purposes. Training accuracy refers to how accurate the model is on the data it is training on; by itself, this score does not serve much purpose, but in the context of the validation accuracy, it has a great deal of use. The validation accuracy measures how accurately the model makes predictions on a set that it is not trained on, and these predictions occur every epoch. Finally, since the model makes adjustments after every epoch based on its success on the validation set, validation accuracy is not a purely independent accuracy score, so following training, we make predictions on a test set that is completely untouched by the model.

Our goal is to maximize all three of these accuracy scores, but ultimately, we want to maximize test accuracy, as well as minimize the difference between training accuracy and validation accuracy (while also maximizing validation accuracy). By maximizing test accuracy, we are quite simply finding a better model, one that can generalize well to unseen data. By ensuring that the training accuracy is as similar as possible to the validation accuracy, we are seeking to minimize overfitting, the phenomenon where a model becomes extremely accurate when predicting the data it is trained on, but worse when evaluating unseen data.

Beyond just the scores though, we will evaluate with several visualizations and other metrics. First, we will plot training and validation accuracy over a period of time (represented in epochs). This will indicate in a visual format the trends of the two accuracies, and can indicate overfitting if the curve for training accuracy has a significant gap above the curve for validation accuracy. We will also plot validation loss and training loss; loss is a metric that measures the cost of a model's predictions, finding a summation of the errors that a model had when predicting on the validation and training sets (Chollet, 2021, p. 9). It can be useful to view both as they are unique metrics, as accuracy measures a frequency of correct predictions, whereas loss measures a sum of costs of false predictions (AI Wiki, 2023). We will also view the numerical value for loss to get a holistic view of the model's evaluation on the unseen test data.

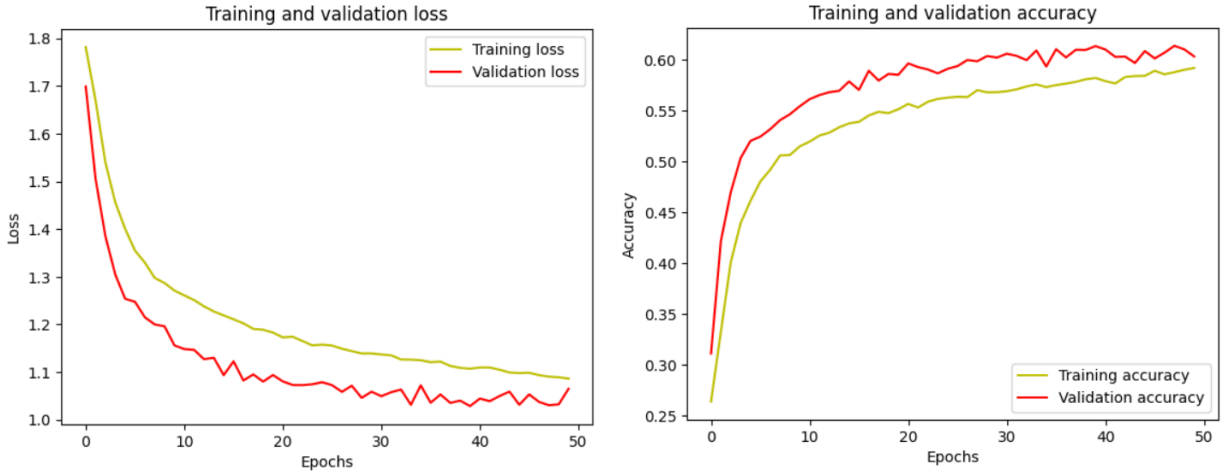
We will also evaluate the model with precision, recall, and F1-scores. Precision calculates the ratio of the number of correct predictions of a specific class (aka true positives) divided by the total number of predictions of that class (the sum of true positives and false positives). Recall calculates true positives divided by the sum of true positives and false negatives, or otherwise, the total number of actual instances of a class (versus the number of *predictions* for said class) (Géron, 2019, p. 139). F1-score is the harmonic mean of the two, which we will seek to maximize as well (Géron, 2019, p. 140).

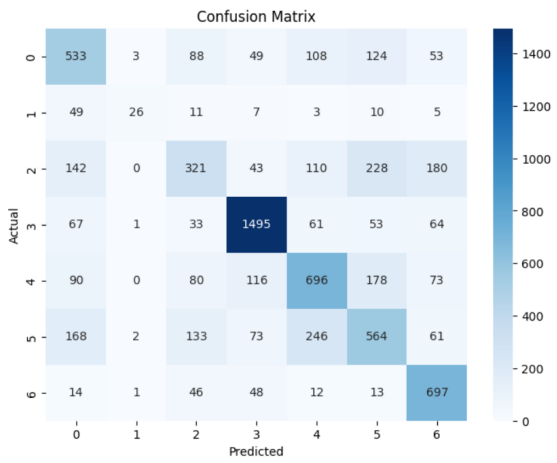
Furthermore, viewing the confusion matrix gives us valuable insights into whether values have been predicted correctly or not; the confusion matrix is a square grid with one axis representing predicted class values, and the other representing actual class values. Each square in the matrix represents the rate of "confusion" with the corresponding row and column it is in; that is, given an actual class x, what is the number of times that the points from class x are predicted as class y (class x can equal class y). Ideally, we will see higher values when class x equals class y, so predictions were correct. The confusion matrix can help visualize this phenomenon, for example, shading darker when values are higher to give a meaningful visual representation of the distribution of class predictions versus actual class values.

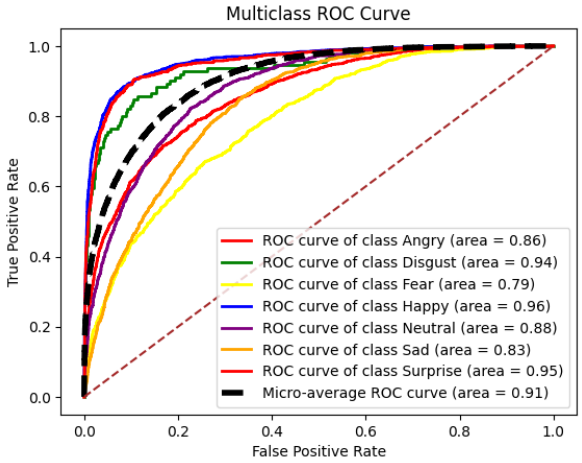
One other evaluation metric we will be using is the receiver operating characteristic (ROC) curve and the area under the curve. The ROC curve is a plot of the true positive rate (TPR) against the false positive rate (FPR) of a classifier. The FPR is the ratio of all negative instances incorrectly classified as positive, while the TPR is the ratio of all positive instances correctly classified as positive (Géron, 2019, p. 146). With the ROC curve, we will be looking to maximize the area under the curve (AUC) when evaluating a good classifier. For multiclass classification, we implement the ROC curve by plotting the micro average of the ROC curves for all of the classes.

For the below confusion matrices the number to class mappings are 0: angry, 1: disgust, 2: fear, 3: happy, 4: neutral, 5: sad, 6: surprise

Basic Model







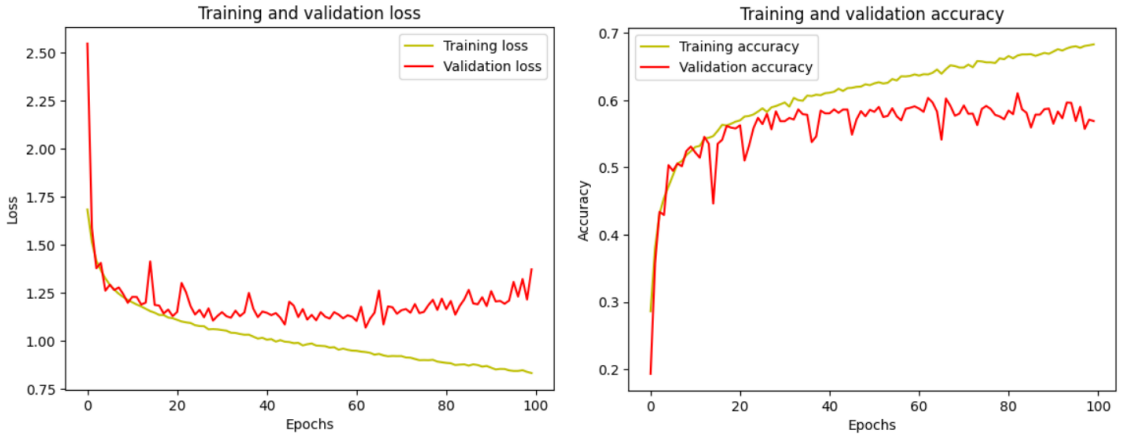
	precision	recall	f1-score	support
0	0.50	0.56	0.53	958
1	0.79	0.23	0.36	111
2	0.45	0.31	0.37	1024
3	0.82	0.84	0.83	1774
4	0.56	0.56	0.56	1233
5	0.48	0.45	0.47	1247
6	0.62	0.84	0.71	831
accuracy			0.60	7178
macro avg	0.60	0.54	0.55	7178
weighted avg	0.60	0.60	0.59	7178

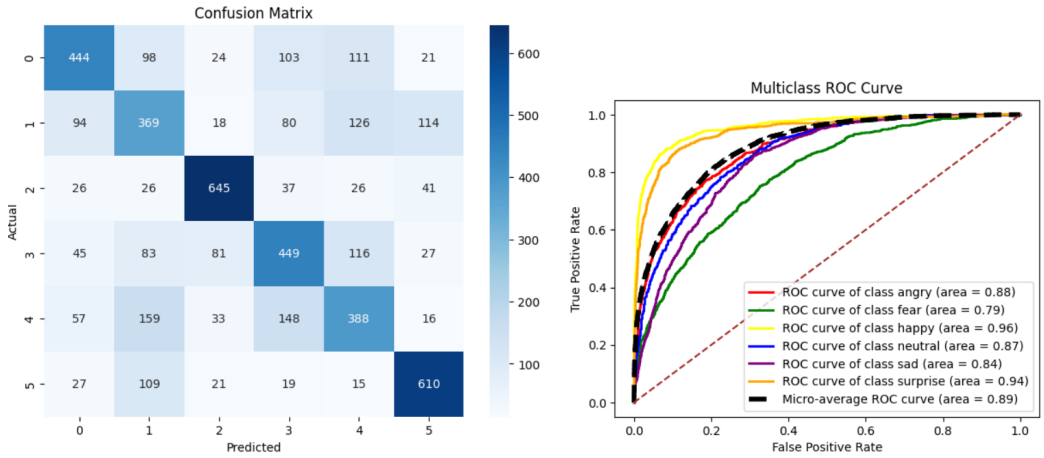
The above graphs show the evaluation metrics for the basic model, which was a from-scratch model trained on the raw FER2013 dataset. The train-validation loss and accuracy graphs show that the model's train and validation metrics were still trending in the same direction together. This leads us to conclude that the model can be trained for longer, as this is a sign that the model has not yet overfit to the training data. Thus, the convergence point may occur after a future epoch. The confusion matrix from this model represents the faults in using this particular dataset without altering the data. There is a high concentration of correct predictions on the happy class, while accurate predictions on all other classes are less prominent. In particular, the disgust class has significantly fewer correct classifications than the other classes. This is due to the severe imbalance in the dataset, where there were significantly more happy images than any other class, and significantly fewer sad images than any other class. The weighted average F1-score is at 0.59, which is a moderate score that should be improved upon. The weighted average precision and recall for this model are both 0.6, which also should be focused on for improvement in subsequent models. The test accuracy of this model was 60.9%, with a loss of 1.05. Though this is greater than random guess, which is about 14%, we will be improving on this accuracy.

The ROC curve displays differential predictive performance for the facial emotion detection model across various emotional expressions. The class "Disgust" and "Happy" show

high AUC values of 0.94 and 0.96, respectively, suggesting that the model is particularly effective in distinguishing these emotions from others. This is indicative of a strong true positive rate as opposed to the false positive rate for these emotions. On the other hand, the class "Fear" has the lowest AUC at 0.79, indicating that the model has more difficulty correctly identifying this emotion. The micro-average ROC curve, with an AUC of 0.91, reflects the model's overall ability to classify all emotions correctly. This high value suggests that, in general, the model can effectively distinguish between positive instances and negative instances across all classes.

Advanced Model Reduced Dataset

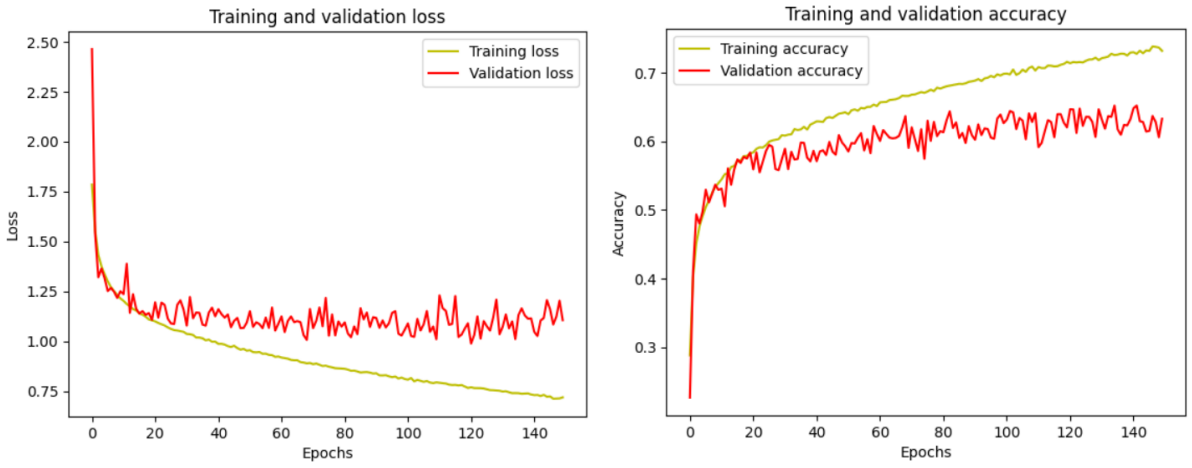


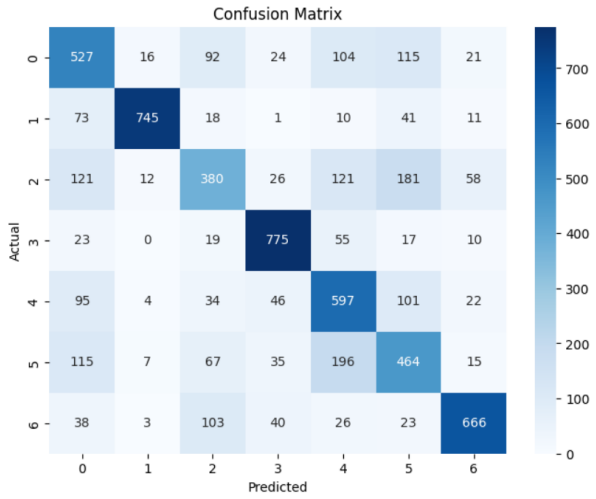


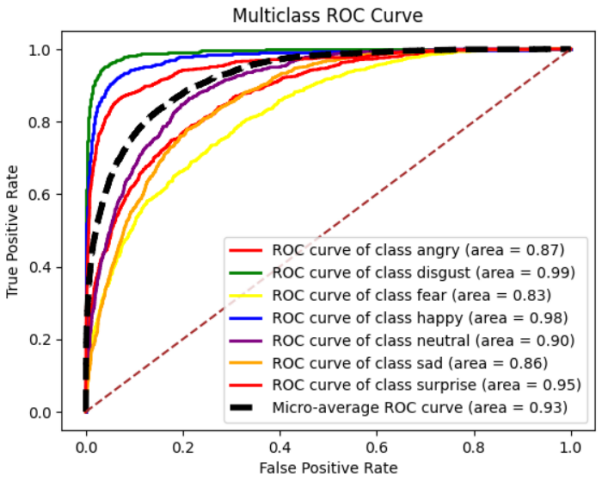
	precision	recall	f1-score	support
angry	0.63	0.56	0.59	801
fear	0.44	0.45	0.44	801
happy	0.78	0.81	0.79	801
neutral	0.55	0.56	0.55	801
sad	0.48	0.48	0.48	801
surprise	0.74	0.77	0.75	801
accuracy			0.60	4806
macro avg	0.60	0.60	0.60	4806
weighted avg	0.60	0.60	0.60	4806

The above visuals are evaluation metrics for the advanced model with the reduced dataset. Note, this is the data that has only the angry, fear, happy, neutral, sad, and surprise classes, all reduced to be the same number of images. These classes map to 0, 1, 2, 3, 4, and 5 on the confusion matrix respectively. As we can see, this model shows an improvement in its ability to classify all 6 classes, as the diagonal of the confusion matrix is far more prominent in comparison to the basic model. The confusion matrix does tell us that images classified as sad tend to be misclassified as fear. This matrix also shows that the model tends to misclassify neutral and sad with each other. The model also has trouble misclassifying fear as neutral, sad, and surprise. Thus, improvements can be made to improve distinction between classes. From the train-validation loss and train-validation accuracy plots, we can confirm that the validation converged around epoch 50, beginning to overfit on subsequent epochs. The AUC of the micro-average ROC curve for this model is 0.89, which is a deterioration from the previous model. This is still fairly high and shows that the average distinction between the classes in this model is generally good. When analyzing the weighted average of the precision, recall, and F1-score, we can see that all three values are 0.6. Finally, the testing accuracy of this model was 61.6%, which was a 1% improvement on the previous model. The loss for this model was about 1.05, which is the same as the previous model. Between this model and the previous model, this would suit our needs better as an emotion classifier due to the broader prediction ability of the other classes.

Advanced Model Adam





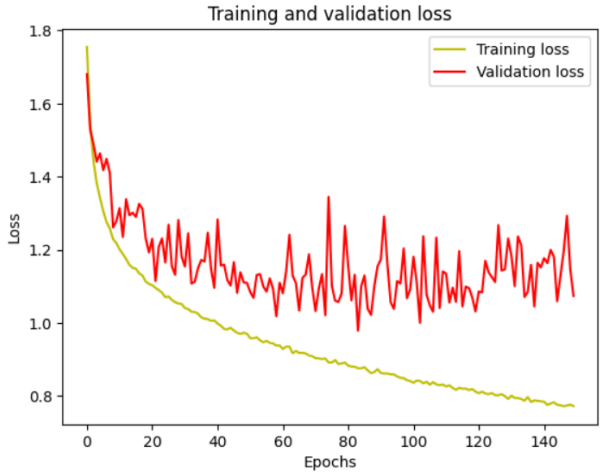


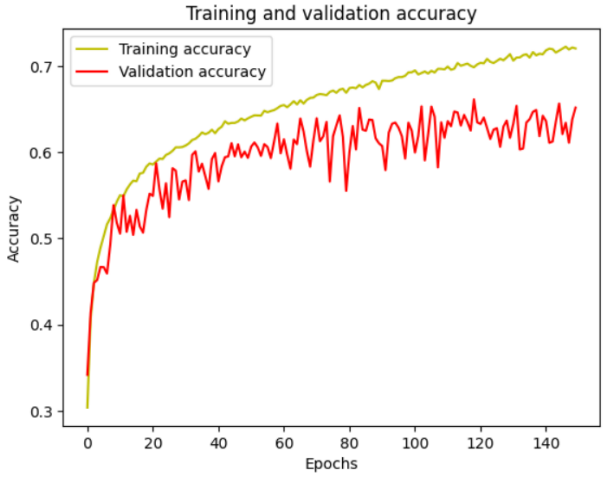
	precision	recall	f1-score	support
angry	0.54	0.60	0.57	899
disgust	0.95	0.83	0.89	899
fear	0.52	0.41	0.46	899
happy	0.82	0.86	0.84	899
neutral	0.54	0.66	0.59	899
sad	0.50	0.54	0.52	899
surprise	0.83	0.73	0.78	899
accuracy			0.66	6293
macro avg	0.67	0.66	0.66	6293
weighted avg	0.67	0.66	0.66	6293

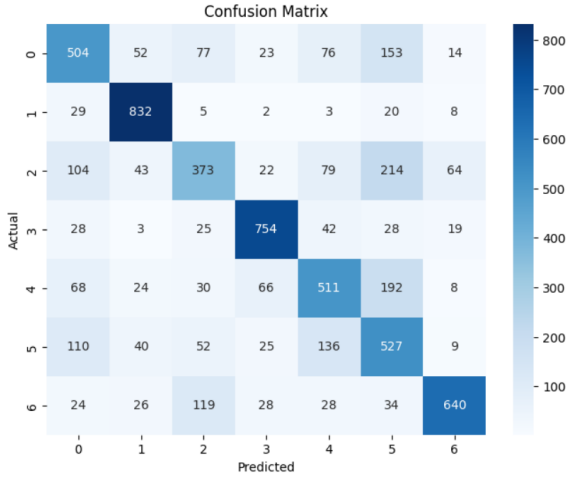
The above data shows the evaluation metrics for the advanced model trained on the dataset expanded with data augmentation. Other unique features of this model's hyperparameters are that it is trained on 150 epochs and that it uses the Adam optimization algorithm. As can be seen in the train-validation loss and train-validation accuracy curves, this model converges to the range of 60-65% validation accuracy. It also appears to overfit within the last 10 epochs as the validation accuracy begins to decrease and the validation loss begins to increase. This model has a stronger ability to distinguish between the different classes, as is shown by the prominent diagonal in the confusion matrix. The model still tends to have trouble distinguishing between neutral and sad images, but it is far improved from the previous model. Another notable misclassification tendency present in the confusion matrix is misclassifying fear as sadness. The ROC curve also shows that this model has improved its ability to distinguish between positive and negative classes, as the micro-average ROC AUC has increased to 0.93. This is also supported by the weighted average F1-score, which has increased to 0.66, with recall and average increasing to 0.66 0.67 respectively. Note however that an average F1-score of 0.67 means that there are still improvements to be made when it comes to distinguishing between the classes. This model also has a significantly improved test accuracy at 66.5%, with a loss of 0.96.

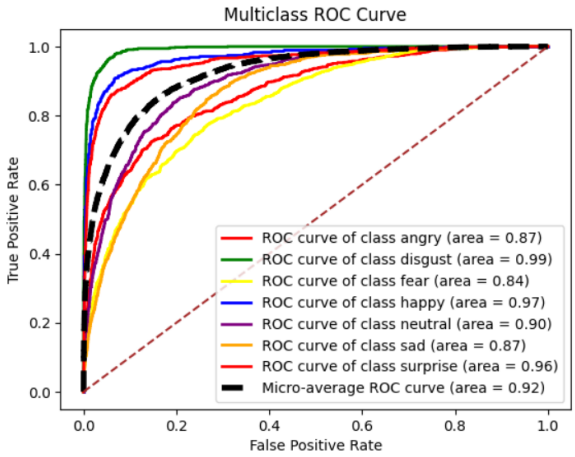
This makes the model lie well within our desired threshold of between 60% and 70% accuracy. Therefore, it is a strong candidate for use on the web application.

Advanced Model RMSProp







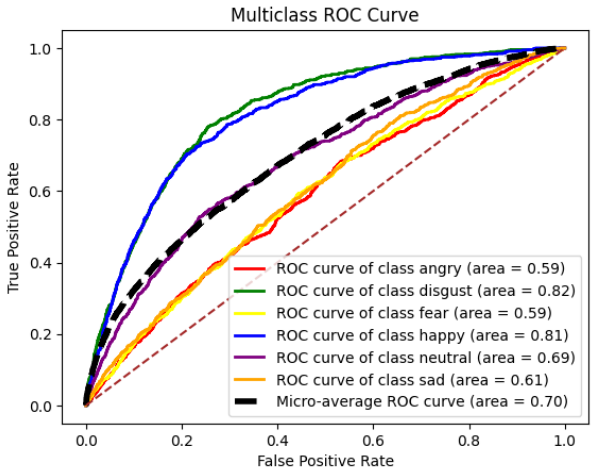


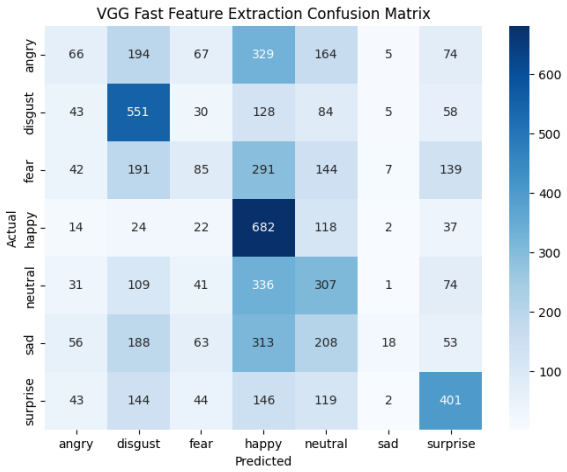
	precision	recall	f1-score	support
0	0.58	0.56	0.57	899
1	0.82	0.93	0.87	899
2	0.55	0.41	0.47	899
3	0.82	0.84	0.83	899
4	0.58	0.57	0.58	899
5	0.45	0.59	0.51	899
6	0.84	0.71	0.77	899
accuracy			0.66	6293
macro avg	0.66	0.66	0.66	6293
weighted avg	0.66	0.66	0.66	6293

The above graphics show the evaluation metrics of the advanced model trained on the expanded dataset with the RMSProp optimizer. Note that this model was obtained through saving the model with the highest validation accuracy during training. The train-validation loss and accuracy metrics show that the model converged at about 100 epochs and began to overfit slightly after subsequent epochs. One other item to note in these plots is the intense jittering in the test and loss graphs. This jittering is likely due to a higher learning rate presented by the RMSProp optimizer that causes it to overshoot over the minimum of the loss function. The confusion matrix tells us that this model had fairly strong distinctions between the classes. The model once again had difficulty distinguishing between neutral and sad images. Since this is common with all models, it is likely that the training images for sad and neutral are quite similar, leading to misclassification. There is also a strong misclassification of fear images as being the sad class. The micro-average ROC AUC of this model was 0.92. This AUC is high, but a slight deterioration from the previous model. This still indicates that the model does a good job at distinguishing between the classes. The weighted average F1-score, precision, and recall are all 0.66 for the model. This is a slight deterioration in the precision in comparison to the previous model, but is fairly comparable. The model trained with RMSProp has a test accuracy of 66.7% and a loss of 1.06. While the loss is fairly high, this model has the highest test accuracy of all the models we trained. Hence, it is also a strong contender for use on the web application.

Pre-trained Models

Fast Feature Extraction with VGG19





	precision	recall	f1-score	support
angry	0.22	0.07	0.11	899
disgust	0.39	0.61	0.48	899
fear	0.24	0.09	0.14	899
happy	0.31	0.76	0.44	899
neutral	0.27	0.34	0.30	899
sad	0.45	0.02	0.04	899
surprise	0.48	0.45	0.46	899
accuracy			0.34	6293
macro avg	0.34	0.34	0.28	6293
weighted avg	0.34	0.34	0.28	6293

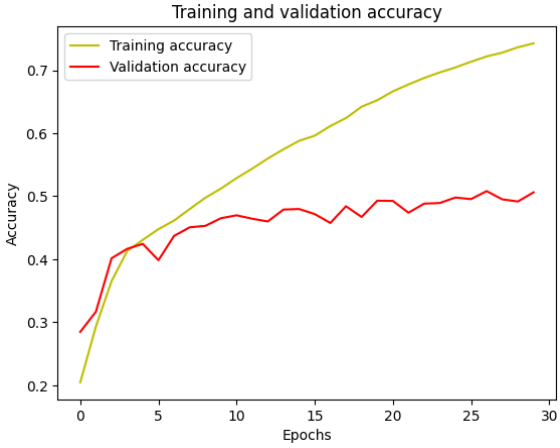
As can be shown above, when using VGG19 with little modification, loss plateaued at a high value for both training and validation sets, while training accuracy remained extremely low and validation accuracy was inconsistent, hovering around the same range of values just under 30%. As will be described later, there are several reasons as to why using a pre-trained model with very little modification (using the weights completely as is to make predictions, then feeding into a small densely connected network) has the potential for problems.

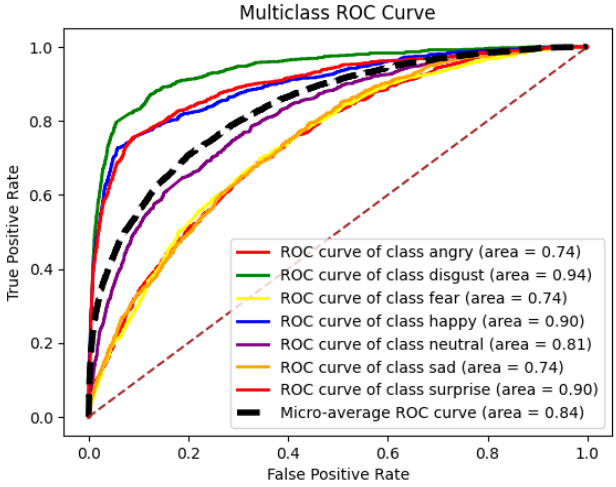
In terms of numerical metrics, this model was only able to achieve training accuracy of 27.77%, validation loss of 2.0388, and validation accuracy of 27.54%. After evaluating the test data, the achieved accuracy was 26.86%. For these reasons, we will stick with evaluating the results of fine-tuning the pre-trained models, as these were more suitable for our dataset.

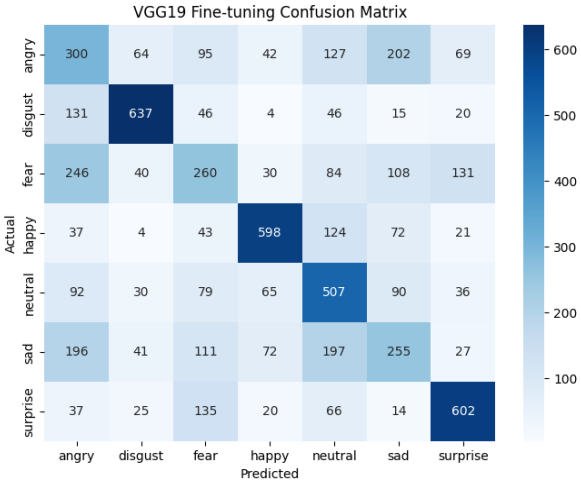
The classification report shows that for some emotions, for example ‘sad’, the model has a high precision at the cost of the recall. The precision for sad is 0.45 but its recall is 0.02, indicating that there’s a high rate of false negatives which then suggests that the decision boundary for ‘sad’ is too stringent or the feature space for ‘sad’ is not well-learned. Similarly, the recall is poor for ‘angry’ and ‘fear’ classes, but higher for ‘happy’ and ‘disgust’ classes. This clearly indicates that the model’s performance is quite varied and it cannot generalize across different emotions.

The inability of the model to generalize is also confirmed by the ROC curves. The 'happy' and 'disgust' classes have higher AUC values, implying a better true positive rate across various thresholds compared to 'angry' and 'fear', which are barely above the no-discrimination line (AUC ~ 0.5). This variability could be an indicator of the model's differential sensitivity to the feature distributions of each emotion class.

Fine-tuning VGG19







	precision	recall	f1-score	support
angry	0.33	0.39	0.36	899
disgust	0.75	0.72	0.74	899
fear	0.31	0.29	0.30	899
happy	0.72	0.66	0.69	899
neutral	0.43	0.55	0.49	899
sad	0.37	0.28	0.32	899
surprise	0.68	0.67	0.67	899
accuracy			0.51	6293
macro avg	0.51	0.51	0.51	6293
weighted avg	0.51	0.51	0.51	6293

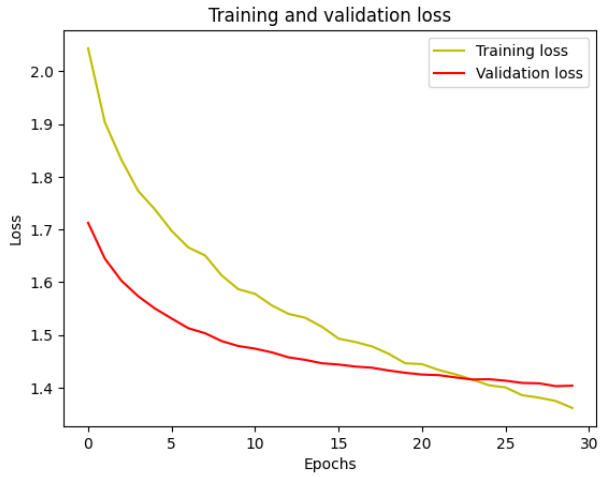
When fine-tuning the VGG19 pre-trained model, the results changed significantly from the fast feature extraction step with the same model. First, the training accuracy reached higher levels than most other models we tried; on the most recent execution of the model, training accuracy reached 74.26% by epoch 30, with previous executions achieving even higher accuracy on the training data (up to 79.99%). The training loss was also fairly low in the context of this problem at 0.6695. On the other hand, validation accuracy and loss were much lower; validation accuracy reached a significantly lower 0.5061, with validation loss at a much higher 2.0737; alongside the visualization of the divergence of both the training/validations loss and the training/validation accuracy, it is clear that the model was overfitting to the training data. This was confirmed by the testing accuracy being 0.4806 and the testing loss being 2.3074.

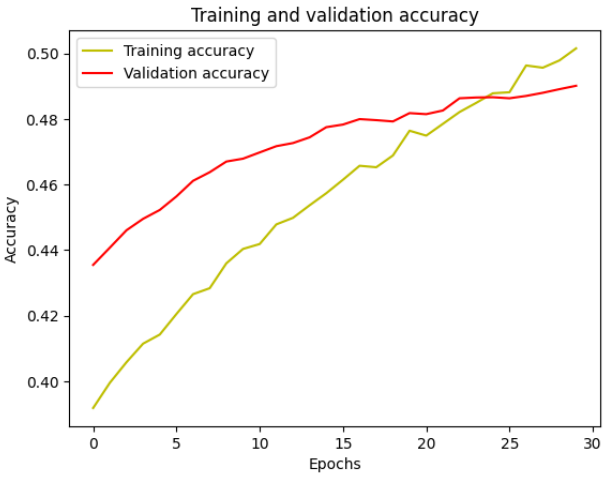
The confusion matrix, calculated on predictions on the test set, indicates a reasonable degree of accuracy in practice, with a darker gradient across the upper-left to bottom-right diagonal, indicating that the classifications with the highest frequencies were generally correct. However, as a percentage of the total number of predictions, some of these values were not as high; for example, even though 300 angry predictions were correctly made when images actually were part of the angry class, there were well over 300 incorrect guesses. This occurred with several classes, and gives a more detailed indication on how the accuracy for this model was generally lower than some of the other classes.

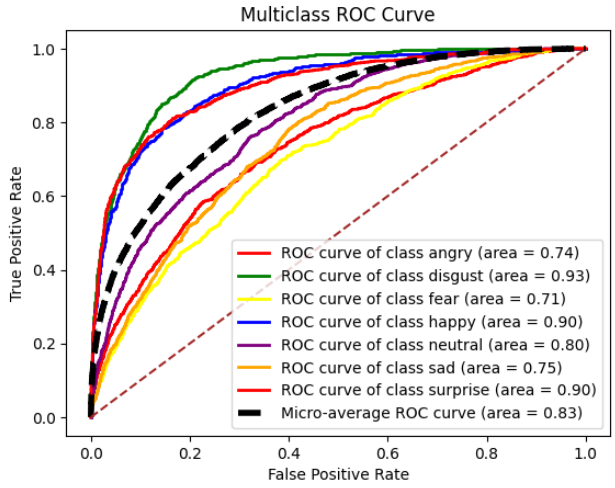
The classification report indicates some imbalance in the model's performance across different classes. For example, 'disgust' has high precision and recall, suggesting that the model's features and decision boundary for this class are well-defined. This is as expected since the data for 'disgust' was inflated using data augmentation. In contrast, 'fear' and 'sad' have lower metrics, indicating that these classes are not as well learned.

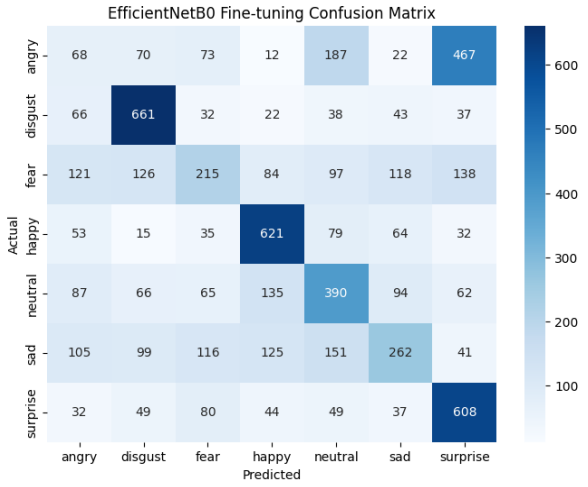
The ROC curves show that the model has reasonably good ability to distinguish between most classes as indicated by the AUC values, which are significantly above 0.5 for all classes. The micro-average of 0.84 suggests that when all classes' performance is combined, the model is performing decently.

Fine-tuning EfficientNetB0









	precision	recall	f1-score	support
angry	0.38	0.32	0.35	899
disgust	0.58	0.73	0.65	899
fear	0.35	0.25	0.29	899
happy	0.56	0.70	0.62	899
neutral	0.43	0.42	0.43	899
sad	0.35	0.31	0.33	899
surprise	0.62	0.66	0.64	899
accuracy			0.49	6293
macro avg	0.47	0.49	0.47	6293
weighted avg	0.47	0.49	0.47	6293

The model that fine-tuned EfficientNetB0 was our attempt at mitigating the overfitting seen from our model that fine-tuned VGG19. However, results went potentially too far in the other direction, with observed underfitting occurring with this model. The training accuracy was much lower at 50.16%, though the validation accuracy stayed close to it at 49.01% after 30 epochs. Loss from both was likewise similar, around the 1.4 range; this value is greater than the previous model's training loss, but less than the previous model's validation loss. It appears that

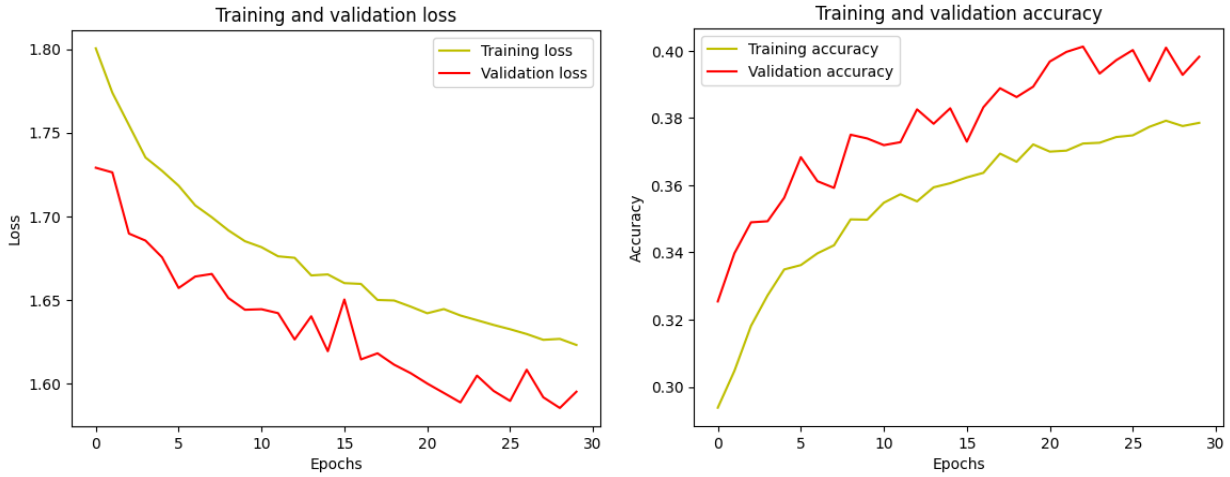
while the previous model may not have been generalizing, this model was unable to pick up on some of the complexities in the training data. As will be explored later, this could be a result of the relatively fewer parameters in this model that may not have been able to pick up on the nuances of important facial features, especially given potential drawbacks of the ImageNet dataset that all of our models were trained on.

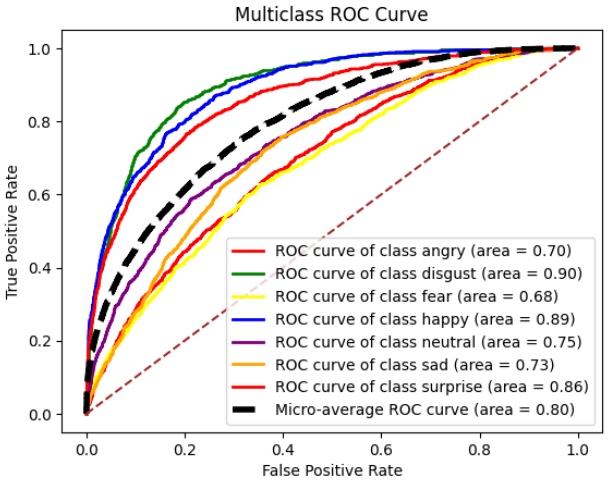
The confusion matrix for this model indicates that predictions of the correct class generally occurred more frequently than any individual incorrect class in most cases, but there is clear indication that this model more often got predictions wrong rather than correct. Also, for certain emotions, the model was completely unable to generalize on the test set; for example, when images were labeled as angry, the model was more likely to predict that the image was an instance of surprised or neutral rather than angry.

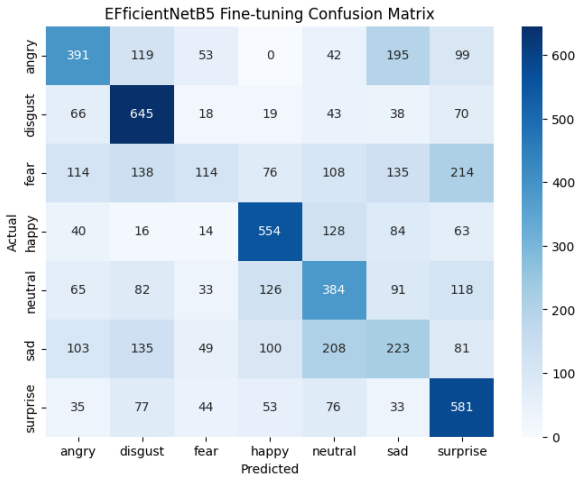
The precision, recall, and F1-score from the classification report for different emotions indicate variability in performance. For instance, the model performs relatively well on 'disgust' and 'surprise' but struggles with 'fear' and 'sad'. Although there is variability it has slightly better overall performance in comparison to the fine-tuned VGG19 model.

The AUC values for individual classes are fairly consistent, above 0.7, suggesting that the model has a moderate ability to distinguish between classes.

Fine-tuning EfficientNetB5







	precision	recall	f1-score	support
angry	0.31	0.21	0.25	899
disgust	0.51	0.72	0.59	899
fear	0.36	0.12	0.18	899
happy	0.54	0.62	0.58	899
neutral	0.35	0.43	0.38	899
sad	0.30	0.24	0.27	899
surprise	0.47	0.64	0.54	899
accuracy			0.43	6293
macro avg	0.40	0.43	0.40	6293
weighted avg	0.40	0.43	0.40	6293

The EfficientNetB5 model was meant to be a step up from the previous model, EfficientNetB0, which seemed to underfit. This model's accuracy was generally lower than most other models we tested, at only 37.86% on the training data and 39.82% on the validation data after 30 epochs. Test accuracy was only 37.81%, indicating a relatively low generalization. Once again, the confusion matrix indicated moderate levels of accuracy, but with some emotions yielding incorrect results much of the time, such as fear and sad. As will be explored later, this model may have gotten stuck in a local minima.

The precision and recall scores show significant variability across different emotions. 'Disgust' has a relatively high precision and recall, indicating that the model is picking up on its features well. In contrast, 'fear' has both low precision and recall, suggesting that its features are not being learned effectively. The low F1-score for 'fear' and 'sad' reflects the model's difficulty in predicting these emotions accurately.

The AUC values for each class show how well the model can distinguish between each class and the rest. While some emotions like 'disgust' and 'happy' have higher AUC values, indicating better distinguishability, 'fear' and 'angry' have lower AUC values, suggesting difficulty in differentiation, but a micro-average AUC score of 0.8 is good.

Based on our evaluation of the above models, we ultimately decided that the advanced model trained on the expanded dataset with the Adam optimization algorithm was the best candidate for use on the web application. While the advanced model using the RMSProp optimization algorithm had a higher test accuracy at 66.7%, we ultimately decided that the Adam optimizer model was better. This is due to the fact that it has a slightly higher weighted average precision score at 0.67, the ROC AUC is higher at 0.93, and from the confusion matrix result. The confusion matrix shows that the Adam model tends to have more correct classifications across all classes, while the RMSProp model has select classes that it is stronger at classifying. It is possible that this random partition of the test set has better performance on the RMSProp model, but adding further test data would show that the Adam model is a more powerful model. Therefore, the Adam model is our best candidate.

Error Analysis

From-Scratch Models

Basic Model

The basic model's main source of error was in the dataset. Though the accuracy of the model was at 60.9%, which is within the desired threshold, this is not fully representative of the behavior of the model. Since there were so many happy images and so few disgust images, the model likely learned preferences in the data. For instance, the model likely learned to prefer predicting that an image was in the happy class, and prefer against predicting that an image was in the disgust class. This model also appears to underfit, evident in the trends for training and validation moving in the same direction by the end of the 50 epochs. Additionally, the validation accuracy tended to be greater than the training accuracy during training. It would be beneficial to increase the number of epochs to observe at which epoch the model begins to overfit.

Advanced Model on Reduced Dataset

The advanced model's errors lie in overfitting. In the validation and accuracy plots it is clear that the model begins to overfit after 50 epochs. This is evident in the fact that the validation loss tends to trend upwards after 50 epochs, and the validation accuracy tends to trend downwards after 50 epochs. This is likely due to the significantly minimized data set, since there is less information for the data to learn generalized patterns from. To alleviate this overfitting issue it would be beneficial to use a fine tuned pre-trained model, or to increase the data such that the model can generalize better.

Advanced Model on Expanded Dataset with Adam Optimizer

The error advanced model trained on the expanded dataset with the adam optimizer is mostly attributed to overfitting. This is evident in the training and validation charts. While the train loss and train accuracy continue to decrease and increase respectively with each epoch, the

validation data plateaus. That is, the validation accuracy maintains around 60% to 64%, while the validation loss maintains between 1.0 and 1.25. This overfitting is likely due to flaws in the dataset, as is further explained below under "*All From-Scratch Models*". Adding further regularization methods to the model may be beneficial in improving these metrics.

Advanced Model on Expanded Dataset with RMSProp Optimizer

The error in the advanced model with the RMSProp optimizer and the expanded data can also be attributed to overfitting. This is shown in the train and validation loss graph, where the validation loss begins to trend upwards after epoch 100. This overfitting is likely due to flaws in the dataset, as is further explained below under "*All From-Scratch Models*". This model also appears to have too high of a learning rate. This would cause the optimizer to jump over the minima of the cost function instead of approaching it. This is observable through the severe jitter in both the loss and accuracy charts. Decreasing the learning rate would improve this issue. It would be additionally beneficial to apply further regularization to this model to fight overfitting.

All From-Scratch Models

It is important to note that all of the from-scratch models displayed patterns in the misclassification of their confusion matrices. For instance, the sad and neutral classes were often misclassified as each other. Additionally the fear class tended to be predicted as the sad class. Given that this is common in all the models and by our investigation in the dataset, this is likely attributed to the data. The sad images and the neutral images tend to look similar, and the fear images tend to look like the sad images. With this data, the model would have trouble distinguishing between these classes, resulting in the confusion matrix metrics that we see in our models. Another important note about the data is the effect that expanding a small dataset had on our models. While we did leverage data augmentation when balancing the class sizes in order to diversify the dataset, this still presented similar patterns to the model across multiple images. It would likely improve the performance of the models if each image was unique and each class had an equal number of images. Compare this to having the balanced classes contain transformed versions of their original images. It is also possible that during the data augmentation, the images were transformed so severely that key features were lost in the transformation, further damaging the model's ability to learn key features.

Pre-trained Models

Pre-trained models have the potential to leverage a high amount of power, but also may not always be perfectly suited to a task at hand. As was previously seen, accuracy was consistently lower for pre-trained techniques when compared to the "from-scratch" techniques taken. Though these models have a high amount of generalization power, and have been trained on large amounts of image data, this does not always imply that they will generalize well to our particular problem.

The pre-trained models that are offered through Keras are all trained on the ImageNet dataset, which has a wide range of images as well as a high depth of images per class. However, a relatively recent update from ImageNet indicates that due to privacy concerns, all face data was blurred from the ImageNet dataset (ImageNet, 2021). The models trained with ImageNet still have generalization power of commonly seen patterns and features, but removing face data means that specific features of faces may not be contained in the models that we were trying to leverage. This could indicate why our models did not perform exceptionally well on the face data; we did not tweak many layers from each model (only the last 3 layers), so especially for models with greater depths, we did not modify many of the original weights. However, these modifications did show significant improvement from no modifications, as fast feature extraction yielded almost exclusively poor results.

VGG19

The VGG19 pretrained model was originally trained with 224x224 pixel rgb image inputs (Simonyan & Zisserman, 2015). As has been mentioned previously, our dataset had 48x48 grayscale images. This indicates that the original model was trained on images that have both more base complexity, as well as a completely different format. Larger images have the potential for more complex patterns, and RGB values allow for a further degree of complexity beyond the more simple grayscale format. We had to import our image data in RGB format so that the model could read the data. While this didn't necessarily worsen results, results were likely worse than if our images were larger and had RGB information. We likely were unable to fully leverage the power of this model and other models due to the nature of our data.

This model's overfitting was evident from visualizations in training versus validation accuracy, where training accuracy was able to achieve around 74% accuracy, with validation accuracy remaining around 50%; to compare, the best version of the advanced model had train accuracy of around 67%, with validation accuracy of about 65%. Though there was still a small amount of overfitting evident, this generalization was much better on unseen data. Something we must consider in terms of VGG19 overfitting to the test data is that in addition to just these format inconsistencies is the actual architecture of VGG19 itself. The architecture had more convolutional layers with less dropout and pooling than our advanced model that was created from scratch, which could explain why the VGG19 fine-tuned model overfit whereas the advanced model did not. The poor accuracies with fast feature extraction indicated that as is, the model was not equipped to accurately make predictions on our dataset without tuning.

EfficientNetB0

We decided to use EfficientNetB0 in order to reduce the overfitting that we saw from fine-tuning the VGG19 model. This model's simpler architecture did seem to reduce overfitting, but reduction in params could be too drastic, as underfitting seemed to occur with generalizations never truly being made. As mentioned before, training and validation accuracy hovered around

50%, which is significantly lower than better versions of the model that we tried. This could be in part due to the combination of EfficientNetB0 having only about ⅛ the number of parameters as VGG19 (4 million vs. 20 million without their dense top layers respectively). This, in combination with the EfficientNet family models we used being trained on ImageNet, were likely simply insufficient to fully learn the facial features in our data.

EfficientNetB5

Our next attempt was using EfficientNetB5; the motivation behind this model was being part of the EfficientNet family, but being more complex than EfficientNetB0, which seemed to underfit. The attempt was to find a balance between overfitting and underfitting, though results unfortunately were not ideal; the model still seemed to underfit, getting only as high as around 42% for training accuracy, and slightly higher at around 44% for validation accuracy. Evaluations on the test set yielded only around 40% accuracy. It was observed that these accuracy values seemed to plateau early, after starting somewhat promisingly at around 40% val accuracy on the first epoch (and only climbing to around 44% after 30 epochs). Unfortunately, it seems as if the architecture for the EfficientNet family had a difficult time finding good general representations of our original dataset.

However, the fact that improvements to accuracy started to significantly reduce quickly indicated a possible plateau, where the error function will remain on the same value for a large amount of time (Géron, 2019, p. 423). Neural networks rarely plateau, meaning that given enough epochs, we may have started to see eventual improvements, but unfortunately, with our resource constraints, we were unable to use a large enough amount of epochs to observe this change.

Furthermore, with both EfficientNet models we used, there were batch normalization layers that were in the final three layers which we unfroze; it may have yielded better results to leave the batch normalization layers frozen, and unfreeze only convolutional layers for both models, something that could have led to better results.

Efforts to Improve the results

From Scratch Model

As discussed above, while the basic model had a test accuracy of 62.53%, which was within our desired range, the results were misleading. As seen in the confusion matrix, the model heavily prefers predicting towards the “happy” class. With this realization, we decided to make efforts to improve both the data and the model. This began with employing the data engineering techniques mentioned above for balancing the data within each class. We began by removing the disgust class altogether. With the class having about 500 images altogether, it was orders of magnitudes smaller than the other classes. Attempts were made to expand the dataset through

Google Images results, but this process was unreliably accurate and only increased the dataset to 1,000 images. We also chose to decrease the sizes of the other classes until they each contained 4,002 images. We changed the model by moving from our “basic model” architecture to our “advanced model” architecture. One aspect of this architecture was deepening the model to increase its complexity and feature extraction at different levels of abstraction. With this deepening of the model we introduced residuals to alleviate the issue of vanishing gradients. We also added batch normalization to aid in the generalization of the model and to aid in training. Additionally, we changed to use depth wise separable convolutional layers instead of normal convolutional layers. These layers provide the advantage of being faster to train. Depthwise separable convolutional layers also support the concept of highly correlated spatial locations and highly independent channels which is present in most images. Finally, we applied a data augmentation layer to diversify the incoming data into the model. This was done to improve the generalization of the model and reduce overfitting. The number of epochs to train this model was also increased from 50 to 100 to ensure that the model trains for long enough. While fitting for too long may result in overfitting, we used a callback to save the model with the lowest validation loss. So, even if the model did overfit by the end of training, we would have saved the model with the minimized loss metric. These changes resulted in the model getting a 61.5% accuracy. While this is only about a 0.6% improvement on the results of the basic model, this model is far more preferred for our goal. This is because, when comparing the confusion matrices between the basic model and the advanced model with the reduced dataset, the advanced model has a far higher breadth of predictions across the 6 classes. This is observed through the fact that the diagonal from the top left to the bottom right on the advanced model is significantly darker than that of the basic model. Therefore, we decided to move forward using the advanced model architecture.

The next step to improving the results was to increase the dataset rather than reduce it. While reducing the dataset provided more balanced representation of the classes, it eliminated valuable information for the model to learn from. Adding more data also will aid in generalization and reduce overfitting. However, we could not duplicate the images already provided in the dataset, as that would worsen overfitting to the training data. Instead, we leveraged the Albumentations library and used data augmentation to generate new images by applying random transformations to the old images. This involved writing a script to evenly select images from the classes to apply random horizontal flip, zoom, translation, rotation, contrast changes, and brightness changes. This expanded each class to contain 8,989 images each. Using data augmentation also allowed us to reintroduce the disgust class for all future iterations of the model. With this expanded balanced dataset, we retrained the advanced model to achieve an accuracy of 63.6%. This was an improvement of more than 2% from the advanced model on the reduced dataset.

When analyzing the training and validation accuracy graphs for the advanced model, we observed that there was a higher training accuracy than validation accuracy, meaning that the model was overfitting to the training data. By the end of training, the training accuracy was 69.8%, while the validation accuracy was 62.6%. To reduce the overfitting we decided to place a dropout layer within the convolution-convolution-pooling blocks of the model to apply regularization. This was a dropout layer that would remove a fraction of 0.25 of the units. Upon training the model again, however, the train accuracy never went above 59% and the validation accuracy never went above 52% after 100 epochs. Hence, this iteration of the model was underfitting in comparison to the previous iteration. Considering this, we decided to remove this dropout layer moving forward.

When observing the training and validation accuracy chart for the advanced model on the expanded dataset, it appeared that while the values were beginning to converge, both the train and the validation accuracy were slightly increasing still. Thus, we chose to increase the number of epochs to provide the model even more time to train to the data. We first increased the number of epochs to 115. Note that we were using a callback to save the model with the best loss validation. This resulted in a testing accuracy of 64.8%, which was more than a 1% improvement when training on only 100 epochs. Since the accuracy increased, we decided to increase the number of epochs to 150 for the next iteration. This resulted in a test accuracy of 66.5%, which was more than a 1.5% improvement on the model trained on 115 epochs. We decided not to attempt increasing the number of epochs any further, as the validation accuracy appeared to converge to 65%, with accuracy fluctuations only going below this value.

Considering the presence of the validation accuracy metric during training, we considered the possibility that the epoch with the greatest validation accuracy potentially would have the highest test accuracy. To test this, we fit the model on 150 epochs and added an additional callback to save the model with the highest validation accuracy. This resulted in two models being saved after training, one with the lowest validation loss and one with the highest validation accuracy. On training the model with these callbacks, the model with the lowest validation loss obtained a testing accuracy of 66.5%, while the model with the highest validation accuracy obtained a testing accuracy of 64.7%. Even though there was no improvement with saving the model with the highest validation accuracy, we maintained this callback for future iterations of the model.

Our final effort to improve the accuracy was in changing the optimizer used on the model. The optimization algorithm is an algorithm used to train the model's parameters such that the loss is minimized during training. We started by using the adaptive moment estimation (Adam) optimization algorithm (Doshi, 2019). The Adam optimizer works by combining the gradient descent methods of momentum and root mean squared propagation (RMSProp). Momentum is an optimization algorithm that considers an exponentially weighted average when

accelerating the gradient descent. This effectively accelerates the convergence in the relevant direction while reducing fluctuation in the irrelevant direction (Ajagekar, 2021). RMSProp is an adaptive optimization algorithm that performs gradient descent by dividing the learning rate with the root mean square of the gradients. This provides faster convergence speeds (Huang, 2020). The benefit of Adam is that it uses both of the other optimizers to build a more optimized gradient descent. It does this by maintaining the moving average of the gradient from momentum while using the gradient to scale the learning rate similar to RMSProp (Ajagekar, 2021). Considering this, we chose to train our model on the RMSProp optimizer as well to see if we could glean better results. Running the advanced model on the expanded dataset, training on 150 epochs, and saving the model with the best accuracy resulted in a model with 66.7% test accuracy. This was the best test accuracy of all of the models that we trained for this project.

Pre-trained Models

Throughout our process working with pre-trained models, our main efforts to improve our results revolved around varying the types of models that we used in attempts to remediate potential errors observed from previous model attempts. To start, while we found that our advanced model had many strengths, using a fine-tuned version of VGG19 was an attempt at gradually changing our model's architecture to observe improvements.

VGG19 had a somewhat similar structure to our advanced model in terms of increasing filter number in subsequent blocks of layers. However, VGG19 uses standard convolutional layers, and has less dropout and normalization between convolutional layers. It is also more complex overall, with more parameters; essentially, we looked to see if a more complex model could identify more complex features and hence improve accuracy.

After seeing that our fine-tuned VGG19 model did not generalize well and instead overfit on our data, we decided to pivot to a new model or family of models that could do so. The EfficientNet family stood out due to the fact that it used techniques to reduce the total number of parameters while still maintaining as high a level of accuracy as possible with accounting for that tradeoff. This technique reduced the total number of parameters significantly, potentially ensuring that more general patterns are observed, avoiding overfitting as much as possible. However, in these models, there is still the ability to observe complexity, especially given the larger number of layers than a model like VGG19 (which is balanced by the lower number of parameters). EfficientNetB0 was the most lightweight model, and we observed it to underfit potentially. Following this attempt, we decided to pivot to EfficientNetB5, which had more parameters and complexity, and potentially worked better with the required complexity of our dataset. Unfortunately, we seemed to have gotten stuck in a plateau that did not allow us to fully see the potential of the model, which would've required many more epochs.

Conclusion and Results

Results

This project is a multiclass image classification project where deep learning is used to predict the emotions of human faces. We leveraged convolutional neural networks trained on the FER2013 dataset to train a model. This project involved implementing several deep learning techniques, such as fine tuning pre-trained models, feature extraction, and advanced computer vision architecture. This project also involved data engineering principles, such as train-test-validation partitioning and using data augmentation to expand the limited dataset. We ultimately concluded that the best model was one trained on the expanded dataset with advanced computer vision architecture. This model was trained using the Adam optimization algorithm on 150 epochs and a callback to save the model with the best validation loss. This final model achieved a testing accuracy of 66.5% and a testing loss of 0.9623. We created an API that used this model to communicate with our front end web application and process images taken from a user's webcam.

Future Work

In order to improve effectiveness, one approach we will take in the future is working with a different dataset. Although FER2013 is considered a standard dataset to work with for facial emotion recognition tasks, past research has not yielded incredible results across the entire field of deep learning. One of the reasons for this is that each image is small and grayscale, so the information that each image can convey is limited. This can have effects on pretrained datasets such as VGG19 and EfficientNetB0, both of which have been trained on ImageNet, meaning larger images with RGB values present. In order to leverage the benefits of these pretrained models, we may want to look for a dataset that better fits these requirements.

Ultimately, our goal is to create a model that can be as accurate as possible when interfacing with common devices in order to indicate emotions of others during video calls and other online communication platforms. These practical use cases would also benefit from a different dataset; most current technologies have capabilities to capture high quality images with RGB values. These practical use cases also require the received data to be passed into our model. Say we have a webcam with a quality of 720p, meaning 1280x720 pixels; though a face might be much smaller, on a normal video call, it is still common for a person's face to be in the range of 100x100 px or larger. If we use a model that only takes grayscale images of 48x48, we immediately lose some detailed spatial information of the larger image, as well as color information. Training a model with a more representative dataset can help improve our results by allowing us to preserve information about the original image, and to actually use this information in the model itself.

Furthermore, our techniques of data augmentation sought to correct the inconsistencies in the amount of data for each dataset, where for example we had many more happy instances as compared to disgusted instances. Finding a better dataset could approach this issue of representation in each dataset. Our current approach used data augmentation to make the sizes of each of the datasets equivalent to each other, but it meant that smaller starting datasets could have more similar images and slightly less variation (though still no repeated images). One of the approaches that we considered was looking towards google image results and preprocessing those as needed to add additional images to our dataset.

We also have several areas which we would like to improve our current models themselves in the future. First, we would like to continue to improve the accuracy of our model; we achieved 66.5% testing accuracy, whereas state of the art models have achieved up to 75.8% test accuracy (Khanzada et al., 2020). The first course of action for future work will be adjusting various hyperparameters and altering the model to improve performance metrics, such as the optimizer type, learning rate, and various layers. To counteract overfitting, we may need to add regularizers to generalize estimations, including more pooling and dropout layers. We may also need to apply more preprocessing to the data before training and testing the model. Improving this performance will also include altering hyperparameters within the model. This will involve a process of trial and error to observe what set of hyperparameters maximizes recall, precision, and accuracy.

There are several ways we can improve the effectiveness of the pre-trained models used. One approach we can use is changing the amount of layers that we tune from the pre-trained model. This is especially pertinent with the EfficientNet models, as they have many more layers than VGG19 (hundreds in EfficientNet vs. 19 in VGG19). Furthermore, we could be more precise about what layers we want to unfreeze and hence tune; in our current methods, we always unfreeze the last three layers, which for both EfficientNet models includes batch normalization layers. In the future, we may want to only freeze convolutional layers for the most effectiveness in our fine-tuning process. We may also want to use a new model that includes properties from both our advanced model made from scratch in conjunction with a fine-tuned, pre-trained model.

Furthermore, we may want to find weights for each of the pre-trained models from training on a more relevant dataset. As was mentioned earlier, ImageNet recently removed all face data from its dataset for privacy reasons, so finding a dataset that has more face data and then using the weights acquired from either training VGG19, a model from the EfficientNet family, or another pre-trained model on this dataset could make our training on FER2013 better.

Once we find the ideal model and tune it to a desirable degree of accuracy, we will move on to improving our interactive frontend. Currently, our frontend uses the user's webcam to

capture an image of their face, preprocesses the image (including using an external model to capture a cropped image of the person's face), and then passes the image through our model to get an emotion prediction. One area where we may expand is creating a continuous stream of predictions that makes the product give constant feedback to the user. Ultimately, our results indicate that even on limited facial image data, it is possible to create a model that can be leveraged alongside an interactive frontend in order to give a user realtime information about facial emotions from faces in a live video.

Works Cited

AI Wiki. (2023). Accuracy and Loss. https://machine-learning.paperspace.com/wiki/accuracy-and-loss

Ajagekar, A. (2021, December 16). Adam. https://optimization.cbe.cornell.edu/index.php?title=Adam

Brownlee, J. (2019, July 5). *A gentle introduction to pooling layers for Convolutional Neural Networks*. MachineLearningMastery.com. https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

Chollet, F. (2021). *Deep Learning with Python* (Second Edition). Manning Publications.

Doshi, S. (2019, August 13). *Various Optimization Algorithms For Training Neural Network*. Medium. https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

Eack, S. M., Mazefsky, C. A., & Minshew, N. J. (2015, April). *Misinterpretation of facial expressions of emotion in verbal adults with autism spectrum disorder*. Autism : the international journal of research and practice. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4135024/

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (Second Edition). O'Reilly.

Huang, J. (2020, December 21). *RMSProp*. RMSProp - Cornell University Computational Optimization Open Textbook - Optimization Wiki. https://optimization.cbe.cornell.edu/index.php?title=RMSProp

ImageNet. (2021). *ImageNet*. https://www.image-net.org/

Keras. (2023). *Keras Applications*. https://keras.io/api/applications/

Khanzada, A., Bai, C., & Celepcikay, F. T. (2020). Facial Expression Recognition with Deep Learning. http://cs230.stanford.edu/projects_winter_2020/reports/32610274.pdf

Kumar, A. (2020, May 13). *How ReLU includes non linearity in neural network?*. Medium. https://medium.com/@kumaranupam2020/how-relu-includes-non-linearity-in-neural-network-b9f03fbcbac9

Ramya, R., & Venkatesh, S. (2022). *Pooling layer*. Pooling Layer - an overview | ScienceDirect Topics. https://www.sciencedirect.com/topics/mathematics/pooling-layer

Simonyan, K., & Zisserman, A. (2015, April 10). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv. https://arxiv.org/abs/1409.1556

Tan, M., & Le, Q. V. (2020, September 11). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. arXiv. https://arxiv.org/abs/1905.11946