# Transfer learning for Facial Expression Recognition

David Orozco[1], Christopher Lee[2], Yevgeny Arabadzhi[3], and Deval Gupta[4]

*Abstract*— **A great deal of progress has been made in object recognition with deep convolutional neural networks, thanks to the availability of large-scale datasets. However, these networks can prove difficult to train from scratch when the amount of available data is limited. The purpose of this project was to classify images of peoples facial expressions into 8 distinct emotion classes using transfer learning on a relatively small dataset. The Cohn-Kanade AU-Coded Facial Expression Database (C.K.)[1] and the Japanese Female Facial Expression (J.A.F.F.E.)[2] Databases were joined and preprocessed. AlexNet, VGG and ResNet architectures were pre-trained on ImageNet, a relatively large dataset containing 1.2 million images with 1000 categories. All of the weights of the networks were then fine tuned using transfer learning. We ultimately achieved an average 90% accuracy across all classes during testing.**

## I. INTRODUCTION

Today, deep-learning (DL) is the primary approach to analysis and classification problems. From the introduction of novel architecture design such as the Recurrent Neural Network (RNN), Convolutional Neural Network (CNN), and many others, spawned an array of variations and improvements to these designs, such as AlexNet, VGG, and ResNet.

These new networks have been increasing in complexity, and in performance. They are, typically, trained on many classes. AlexNet, for example, was trained on 1000 different classes of images. This makes the model, in theory, develop universal features, and better at differentiating between objects.

Emotion classification introduces a new sets of challenges, since the model needs to be able to differentiate intra-facial features. We use DL for, both, feature selection and expression classification. AlexNet provides a good start for Transfer Learning. Because of the many classes the net was trained on, there is an expectation that universal features had been learned by the network.

We will train, or tune, the model using 224x224 color images as input. Then, we will classify each image as one of eight different mood, or emotion, classes. See Fig. 1 for the visualization of the Input/Output relationship. AlexNet is abstracted in the figure.

## II. Related Work

Happy, George, and Routray used local binary patterns (LBP) to classify facial expression grayscale images into 6 different emotions: happiness, sadness, disgust, fear, surprise and anger. Similar to our approach, the initial data preprocessing involved facial detection using a Haar Cascade Face detection algorithm followed by a resizing of the image. Happy, George, and Routray used LBP to find features in minor changes in facial expression, which is then followed

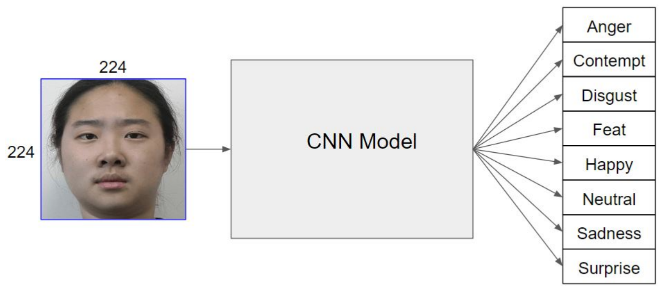


Fig. 1: Input/Output of Model.

by a dimensionality reduction by using PCA. The training result is stored to classify features. Their approach to use LBP and PCA is different from our transfer learning using convolutional networks, but they also encounter the problem of generalizability results.

Arumugam used Radial Basis Function network (RBFN) for classification and Fishers Linear Discriminant (FLD), Singular Value Decomposition (SVD) for feature selection in order to classify facial images into emotions. Instead of the Haar Cascade method, the Radial Basis Function network was used to detect faces from images. Furthermore, the dataset was made to only include three emotions: happy, anger and disgust. This will inherently increase the accuracy of an classifier. On the raw images, FLD and SVD were applied to because of their manageability, especially for large data such as images.

In both approaches, real time images of subjects were taken; instead of downloading databases of existing face images. This was justified because of ease of capturing real time images from a camera, and the author's ability to prelect images used for training and testing. Furthermore, the authors were able to create large datasets of images of themselves as testing and training data.

## III. Dataset and Features

Our combined data consists of two datasets. Each contained a degree of labels, and other meta-data. Before use, and preprocessing, the datasets needed to undergo a pre-selection process. Each image, then, underwent a standard array of preprocessing and data augmentation that is recommended for use in CNN.

### *A. Pre-Selection*

Our combined data consists of two datasets. Each contained a degree of labels, and other meta-data. Before use, and preprocessing, the datasets needed to undergo a pre-selection process. Each image, then, underwent a standard array of preprocessing and data augmentation that is recommended for use in a Convolutional Neural Network CNN.

- **C.K.** is a facial expression dataset which contains video transitions of emotions (or moods). Each video shows a transition from a neutral facial expression to one of seven other moods. We are, however, performing image classification. So, each video needed to be divided into frames of two classes. This was done with a simple percentage threshold. See Fig. 2 for an example of the dataset. The dataset consisted of 120 participants expressing seven emotions (plus neutral), with a total of 5877 frames (images).

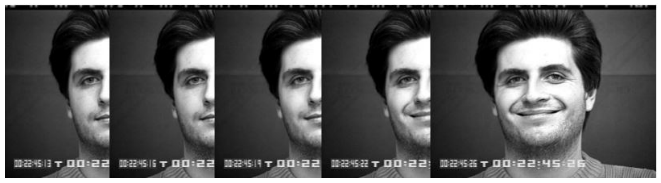


Fig. 2: Cohn Kanade example.

- The **J.A.F.F.E.** dataset contains 210 emotion images. These are generated by ten Japanese women, each expressing three images per class. There was no pre-selection needed for this data set, other than sorting into appropriate directories, as this was the most well prepared dataset. See Fig. 3 for an example of the dataset.

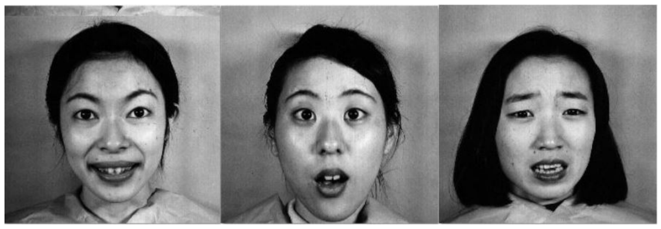


Fig. 3: J.A.F.F.E. example.

The pre-selection process reduces the overall datasets size, slightly, and changed the number of classes. Table I shows this change. Fig. 4 shows the distribution of data among the classes. The data is not as uniformly distributed as we'd like, but further 'shaving-off' data points is not desirable with our dataset size.

TABLE I: Dataset size change.

| Dataset Name | Initial Size | Size After Selection | Initial # of Classes | # of Classes After Selection |
|---|---|---|---|---|
| C.K. | 5900 | 4500 | 7 | 8 |
| J.A.F.F.E. | 210 | 210 | 8 | 8 |

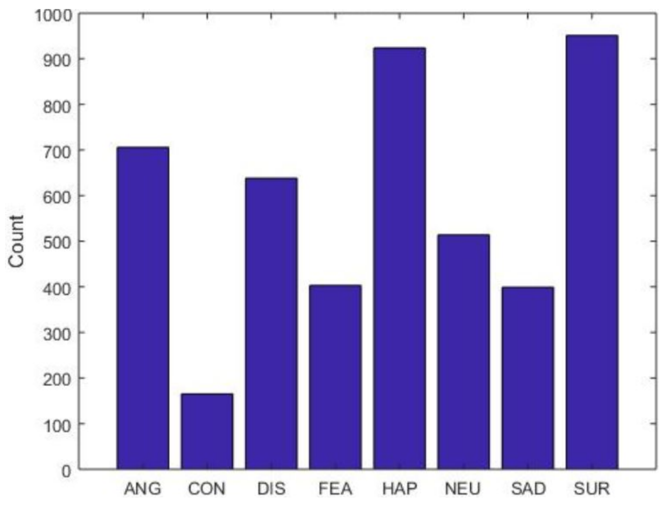


Fig. 4: Class Size Distribution.

## B. Preprocessing

The images varied greatly in the size ratio of face to image; in some images the face would take up most of the image, and in others only a quarter. For consistency, we decided to use a prebuilt Haar cascade function, from OpenCV, for face detection and cropping. The crop box was increased by 20% for some augmentations to be possible, described in the next segment. See Fig. 5 for visualization of this processing step.

Our complete dataset contained both GrayScale and RGB images. AlexNet architecture was, however, built for RGB images, only. To account for this, we simply replicated the GrayScale images into three identical layers.

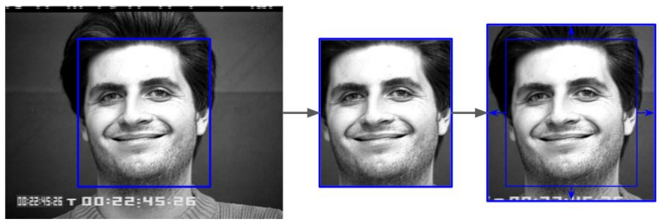


Fig. 5: Cropping Example Visualization.

## C. Augmentation

During data collection, researchers often focus on consistency, for cleaner datasets. That means images are taken from a similar distance and the same angles. Although the consistency makes the data easier to work with and process, it does not necessarily translate to better model performance. This is because, when training, the model becomes biased towards some features. These might be the angle of the object, or the lighting, etc. To make our model robust to these changes, we implemented five types of data augmentations:

- **Resizing**. All images are, first, resized to a 256xN image. N is based on the dimension ratio of the image, after cropping. For example, if, after cropping, the image size was 512x200, then N=100. So, both dimensions are resized evenly.

- **Cropping**. Now, we take random crops, of the 256xN image, of size 224x224. As mentioned in the *Preprocessing* section, the cropping rectangle was enlarged by 20%. The enlargement, combined with this step's cropping, creates images where small parts of the face are missing.
- **Horizontal Flip**. Flipping random images, horizontally, does not necessarily increase the model's robustness, but it is an easy way to increase the dataset size. It was an appropriate augmentation, since human faces and facial expressions are fairly symmetric.
- **Rotation**. Another possible variation, during classification, is the rotation of the object in the image. To make our model robust to this change, we rotate random images by $\pm 1^o$. The augmentation is limited to changes of only $1^o$ because of the datasets size limit. In order to train the model to be robust to bigger changes, we would need much more data.
- **GrayScale**. Next, a random set of 10% of the images are converted from RGB to GrayScale. Since the features we want the network to learn are not necessarily in color, but in the facial expression, this will add further robustness to our model.
- **Normalization**. Finally, each image was normalized using *transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])*, which scales the image values in the range of [0, 1].

It is worth noting that augmentations such as these produce a two-fold benefit. Primarily, the model becomes robust to changes. Secondarily, by augmenting an image, we create an augmented copy of that image, and in turn, this produces more data. See Fig. 6 for a visualization of the **Cropping / Horizontal Flip / Rotation / GrayScale**augmentations performed.

## IV. Methods

Three pre-trained networks (VGG, AlexNet, and ResNet) were pre-tained on the ImageNet dataset using the process known as transfer learning. First we explain the network architectures and how they were adapted for the specific task of emotion recognition. We then go into detail on the learning algorithms used to train these networks on our new datasets. Lastly, we include information of how our datasets were augmented to increase the performance of our networks and how we went about testing and training.

### A. Network Architectures

Before explaining the structure and size of each architecture we will give a generalized explanation on the subcomponents used in these and other common networks, how they work and why they are used for image recognition.

*1) CNN components:*

- **Convolutional Layers.** This is the defining feature of CNNs that separates them from other common neural network architectures such as Multi-layer perceptions. For a given input layer a number of 2 dimensional ”feature map” or image stacked on a 3rd dimension or

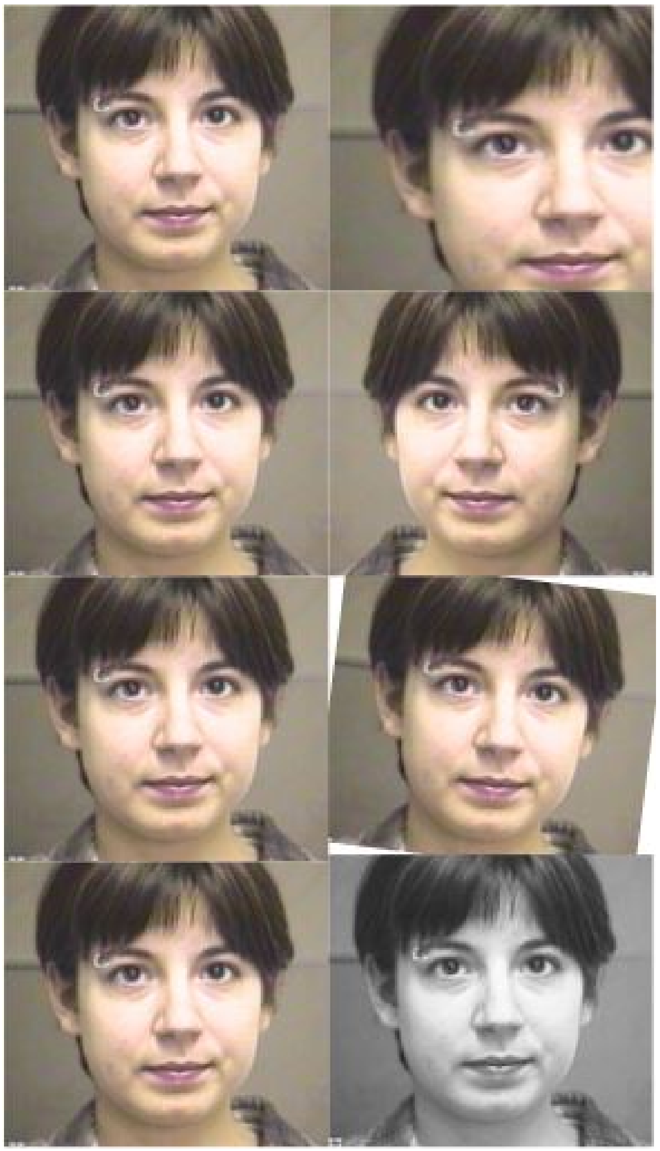


Fig. 6: Visualization of Augmentations.

”depth” is typically fed in and a smaller rectangular ”kernel” is convolved with the image. The explicit formulation of this process is governed by the equation below, where h is the input layer, and a is one kernel and the output is one layer of a feature map.

$$c[m,n] = a[m,n] \otimes h[m,n] = \sum_{j=0}^{J-1} \sum_{k=0}^{K-1} h[j,k]\, a[m-j, n-k]$$

There are multiple reasons for using this convolution operation on images. Since pixels in images typically are only related to pixels in nearby regions and independent of pixels far away, convolution gives pixel features ”temporal” independence. i.e. extracted features are only functions of pixels nearby. Another reason for using convolution is that the output feature map is

typically smaller than the original input, as expressed in the following equations for the output size relationships.
$W_o = (W_{in} - F_w + 2P)/S + 1$
$h_o = (h_{in} - F_h + 2P)/S + 1$
Where, $w_o$ is the output width,$w_in$ is the input image width,$f_w$ is the kernel width, p is the padding size, and s is the stride used. Likewise, height is expressed in the latter equation. This decrease in size allows for a method of dimensionality reduction which is important when reducing the number of features down for large images (curse of dimensionality). One last important reason for using convolutional kernels is that the inputs used shared weights. Since a single kernel will be used for a convolution over an entire image, it greatly reduces the require number of parameters of each ”pixel” or dimension. Given that these networks can reach parameter numbers in the order of $10^6$, reducing the number of weights used can reduce over fitting, and the number of required training images.

- **Rectified Linear Units (ReLU).** RelU is a non-linear activation function used at the output of each convolution layer. Clearly, just like other non-linearities used in neural networks, ReLU allows the network to learn non-linear functions. However, the reason for the choice of ReLU over other activation functions is that it reduces the ”vanishing and exploding” gradient problem caused by deep neural networks when back propagating. This unit allows for 6 times the training speed over tanh, with no cost to accuracy. The ReLU function and gradients are defined respectively below.

| | | |
|---|---|---|
| ReLu | $y_s = \max(0, x_s)$ | $\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \mathbb{I}\{x_s > 0\}$ |

- **Max Pooling.** Max pooling solves two problems for CNNs; it reduces the output size of each layer thus reducing dimensionality, and it gives the network translational invariance. The latter is important because if a feature is shifted slightly by a few pixels, it will be collected in a ”max pool” and gets routed to later layers of a larger perimeter. This has shown to have higher performance than other pooling techniques such as ”average pooling”. An example of how this is done is shown in figure 7 below.

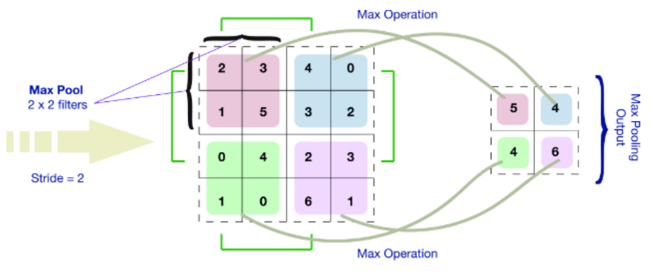


Fig. 7: Maxpooling Technique

- **Drop Out.** This technique is used in CNNs to reduce over fitting by further reducing the number of trainable parameters and temporarily blocking neural pathways. The idea is that neurons become specialized for specific features and that by randomly disabling weights forces the network to create new paths and feature representations throughout the network.

*2) Architectures & Adaptations:* All of these networks were initialized using pretrained weights, adjusted for our specific task, and then fine tuned on our new datasets. The task specific adjustments are also explained for each network below. **Note: ”fc” and ”dense” in all architectures stands for a standard fully connected network.**

- **VGG 19.** This is a deep network consisting of 19 layers. A visualization of the original architecture follows in figure 8. Since the pre-trained network was originally trained on the 1000 classes of ImageNet we can see that the final layer consists of a 4096x1000 fully connected layer. This needed to be replaced by a randomly initialized 4096x8 fc layer, and had to be entirely trained from scratch.

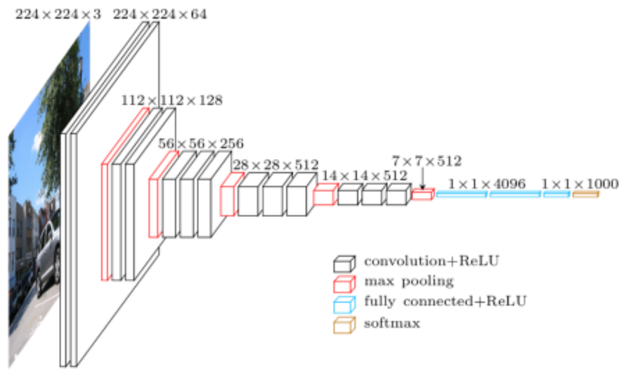


Fig. 8: VGG 19

- **ResNet.** This is the deepest network that we trained and tested. It is made up of small ”residual blocks” shown in figure 9, the purpose for the direct connection with the input to the output is to give a direct path for the error to back propagate through the entire network. With such a deep network the vanishing gradient can pose a real problem and this was designed with this in mind. We expect this network to have the lowest error because the depth allows the network to learn more complicated representations and features. We can see in figure 10 that the final layer has similarly been changed to have a final 8 class output fc layer.

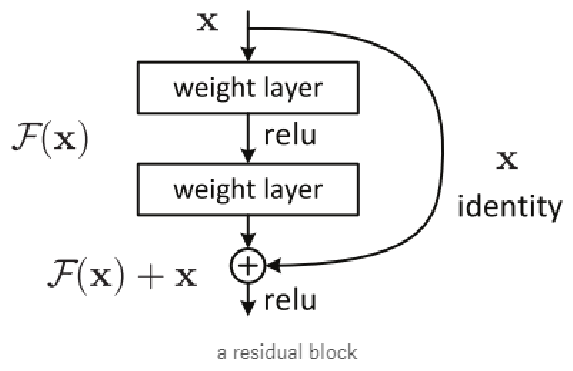

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x})+\mathbf{x}$
relu
a residual block

Fig. 9: Residual Block for ResNet

## B. Learning Algorithm

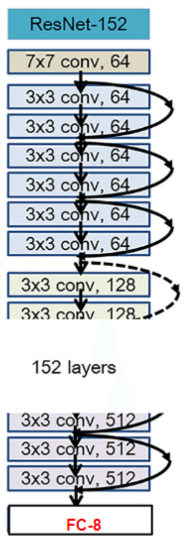


Fig. 10: ResNet architecture

- **AlexNet.** At only 8 layers deep this is the shallowest network we trained and tested. The entire architecture and adaption can be seen in figure 11. This demonstrates the same type of layer removal described for the prior two networks.

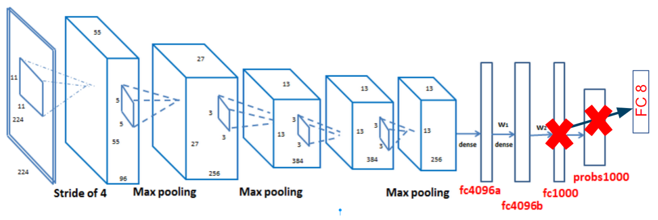


Fig. 11: AlexNet Architecture Change

## C. Learning Algorithm

*1) Problem Formulation:* First we define an image as a vector $x$ from our training set of size N, where $x\epsilon\{x_1,..,x_N\}$. We then split our dataset into batches $x^t$ of size K. i.e. $x^t\epsilon\{x^1,...,x^K\}$. Our CNN takes in as input a batch of examples, and produces a probability distribution representing the liklelihood that an example $x$ came from a given class. Thus, the output is a probability distribution of dimension C, where C is the number of classes. The final layer of the CNN is a "soft-max" function that produces these probabilities governed by the following equation,

$$P(y=j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^{\mathsf{T}}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^{\mathsf{T}}\mathbf{w}_k}}$$

where the vector $x$ is the features produced by the final fc layer of the CNN, and $w$ are the learned weights associated with these feature, and the output is the probability that this vector belongs to class j. Therefor, we can think of the final layer of the CNN as a linear classifier separating the C classes based on the learned features, and the output of the softmax as the probability distribution of what class the example belongs to. It can be noted that the softmax function enforces the requirement that the probability distributions must sum up to 1.

Each example $x$ has a target true one-hot encoding "label" associated with it depending on what facial expression it comes from. For example, if image $j$ comes from class 8, then the target label $t_j$ is the one-hot encoding $\{0,0,0,...,1\}$. This label represents the target probability distribution for this example. i.e. the probability that this example comes from class 8 is 1.

We can now define a loss function for our loss function because we have target labels for each image in a batch $t_n\epsilon\{t_1,...,t_K\}$ and output distribution we will define as $Y$ where $y_n\epsilon\{y_1,...,y_K\}$. This becomes a simple supervised learning problem and the objective becomes to minimize the cross entropy loss function for each batch of images denoted below.

$\Sigma_{j=1}^{K}\Sigma_{i=1}^{C} - y_{j,i} * log(p_j, i)$

where j represents an observation, and i represents a given class.Thus $y_{i,j}$ is the true label for the jth observation and ith class, and p is the probability that the network predicted this example belonged to that class. C and K are the number of classes and size of a batch respectively.

*2) Stochastic Gradient Descent and Momentum:* For our means of minimizing the loss function we use stochastic gradient descent. That is if we define our loss function as before, but use only one training example $j$ as an approximation for the the true loss function, we will arrive at the following definition for the loss function Q(w).

Q(w) = $\Sigma_{i=1}^{C} - y_{j,i} * log(p_j, i)$

We then minimize this loss function for the current training example, and repeat this process iteratively for random examples in the batch until we have done this for our entire dataset. There are two main reasons to use stochastic gradient descent over other gradient based optimization methods. First, when the size of datasets is large, it may be computationally expensive or difficult to calculate the entire "true" gradient, so stochastic gradient descent samples a subset of summand functions at every training step. The second reason is that if a examples in a training set are highly correlated or similar, the gradients from different examples tend to cancel out and then add up in a single direction when you when averaged over the entire dataset. What this means in a practical sense is that similar training examples provide little or no additional information when averaged together, yet to take a single step, you must have finished an entire ”epoch”. So using stochastc gradient descent with highly correlated data means that you can essentially take more gradient steps using less epochs, and therefore is typically more efficient by number of iterations required.

To minimize this loss function we take the gradient of the loss with respect to the network parameters $w$, and adjust our weights by ”taking a step” in the direction that minimizes the loss. This process is described by the following update step, the parameter $\eta$, is the learning rate which is typically adjusted iteratively can be thought of as a function of epoch $t$, i.e. $\eta$ = $\eta(t)$

$$w := w - \eta \nabla Q_i(w).$$

To improve training we also use the proposed momentum method which gets added to our gradient step. The idea behind momentum is that we use a weighted sum of our previous weight updates so that our gradient descent builds ”momentum” in a certain direction. This should dampen oscillations in our gradient step keeping the average direction of our weight updates to be in a certain direction and thus less effected by changes in the gradient. The final form of the weight update is defined as follows.

$$w := w - \eta \nabla Q_i(w) + \alpha \Delta w$$

### D. *Training and Testing*

Training data was augmented to ensure robust training. The methods employed included: 240 pixel resize, random 224 pixel crop, random horizontal flip, 10% random gray scale, +/- 1% angle deviation, and gray scale pictures to RGB. A early stopping method using 10% of the data was also implemented. Another 10% of the data was used solely for testing purposes. Finally, images of one team members faces were also tested. The data augmentation on the training data was further detailed in the Data Augmentation section of Dataste And Features.

## V. Experimental Results

### A. *Choosing a network*

To classify the facial expressions we decided to run a preliminary test on three well-known networks: AlexNet, VGG-19, and ResNet. Each network was trained over a constant 30 epochs to observe how well each trains per epoch.

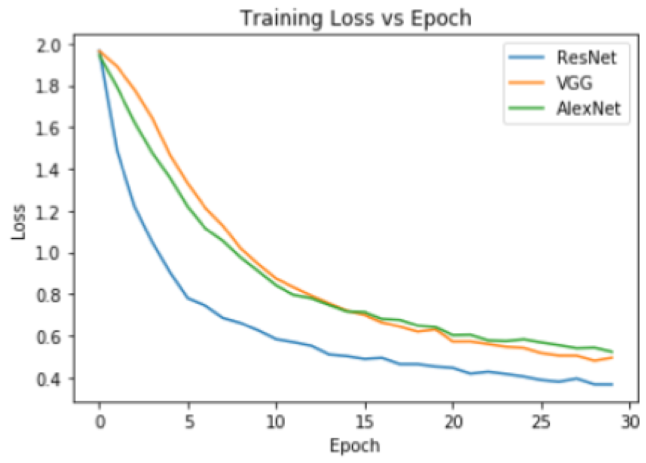


Fig. 12: Training loss vs epoch for each network

As shown in Fig. 12, AlexNet and VGG appear to have similar training curves, with ResNet training significantly faster for the same amount of epochs. This is primarily due to the greater complexity of ResNet (over 150 layers vs 8 layers for AlexNet and 19 for VGG 19). Although VGG and AlexNet had similar training curves AlexNet appeared to run in less time than VGG per epoch of training. Coupling that with the simpler architecture relative to ResNet we decided to use AlexNet to further our testing.

### B. *AlexNet on test set*

After training AlexNet for 50 epochs we achieved the following accuracies across all classes:

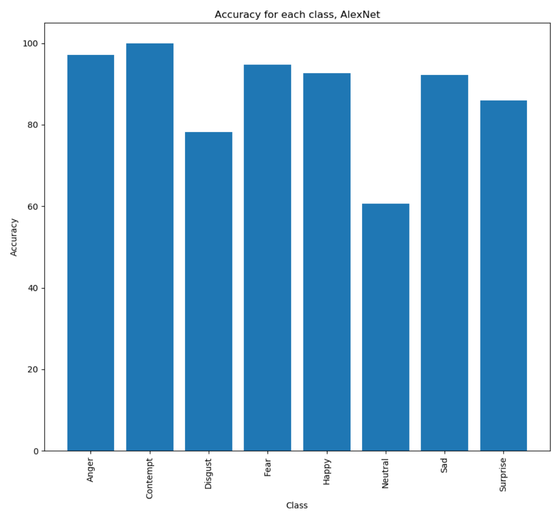


Fig. 13: Accuracy of AlexNet for each class

In general, our network had an average of 90% accuracy across all classes, with the highest accuracy in contempt and the lowest in neutral.

### C. *Overfitting*

To better understand how the network is classifying images the confusion matrix is displayed below:

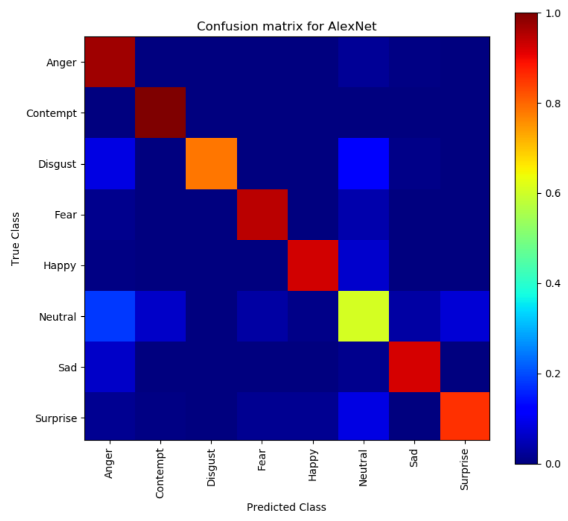
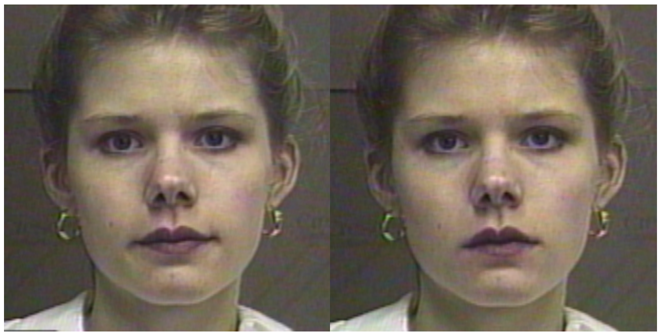


Fig. 14: Confusion Matrix for AlexNet

From the matrix in Fig 14 the test set faces from the neutral class seemed to have false-positives as well as some false negatives. This might be due to some of our data being inconsistent. For instance some images from the C.K. dataset may lean toward neutral. This is due to how the images from this set were extracted - stills from a video displaying a range of expression from neutral to the given expression. Hence there could be too many images in other classes that should belong in the neutral class.



(a) Contempt (b) Neutral

Fig. 15: Example of similarities between classes

One way we tried to mitigate this was to implement early stopping on the training to prevent over fitting. However there was no noticeable improvement in the network's performance regardless of architecture. Thus another option to explore would be to "clean up" the training data a bit more, removing any images that are too ambiguous.

### D. *Generalizing*

After having relative success with the test set we decided to see how well the network generalized to new faces, specifically one of ours. To limit the amount of variability we only used one face to provide various expressions. The network had an average accuracy of 34.7% which is much lower than previously stated with the previous test set.

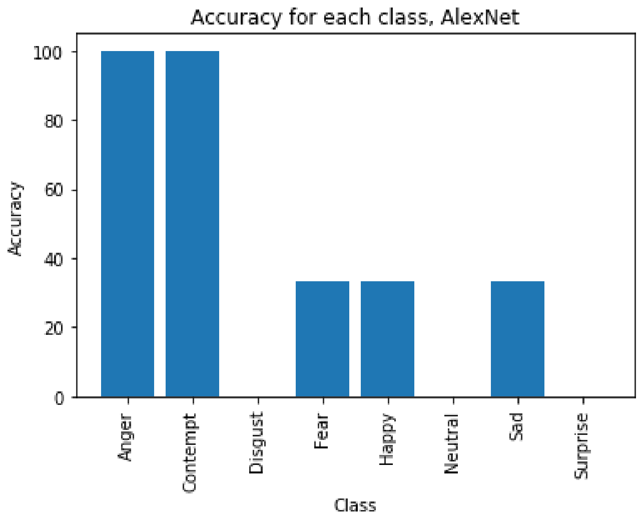
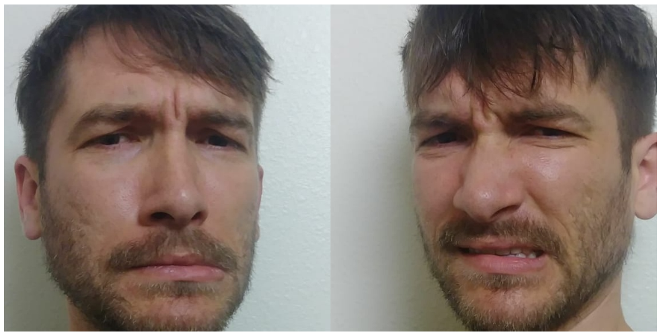


Fig. 16: Classification accuracy for a new face

We believe that due to the new lighting and photography angles of the new expressions the network was not able to generalize to the new data. This also explains why the network did so well on the previous test set, which was composed of images from the same dataset as the training images (although the same images were not used to both test and train; the set was portioned).



(a) Sadness (b) Disgust

Fig. 17: Example of angle and lighting variance for new face

## VI. CONCLUSION

Given the accuracy of the network on the test set it is clear that the classifier shows promise. The only pitfalls we observed were most likely from the portioned dataset we trained on. Thus in order to improve this model there are a few options to consider. One possible way to improve the networks performance would be to clean up more of the training data, making sure that there are no mislabeling of images. Coupling this with early stopping while training can improve performance and generalize to new faces and environments.

## REFERENCES

[1] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews, The Extended Cohn-Kanade Dataset (CK ): A complete dataset for action unit and emotion-specified expression, 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops, 2010.

[2] The Japanese Female Facial Expression (JAFFE) Database, Facial Expression Database: Japanese Female Facial Expression (JAFFE) Database. [Online]. Available: http://www.kasrl.org/jaffe.html. [Accessed: 16-Jun-2018].

[3] Mahmoud Afifi and Abdelrahman Abdelhamed, "AFIF4: Deep gender classification based on an AdaBoost-based fusion of isolated facial features and foggy faces". arXiv:1706.04277, arXiv 2017.

[4] W. Bainbridge, 10k US Adult Faces Database, Wilma Bainbridge, Ph.D. [Online]. Available: http://www.wilmabainbridge.com/facememorability2.html. [Accessed: 16-Jun-2018].

[5] OpenCV: Face Detection using Haar Cascades, [Online]. http://docs.opencv.org [Accessed: 16-Jun-2018].

[6] How transferable are features in deep neural networks?, https://arxiv.org/pdf/1411.1792.pdf

[7] S. L. Happy, A. George, and A. Routray, A real time facial expression classification system using Local Binary Patterns, 2012 4th International Conference on Intelligent Human Computer Interaction (IHCI), 2012.

[8] D. Arumugam and D. S., Emotion Classification Using Facial Expression, International Journal of Advanced Computer Science and Applications, vol. 2, no. 7, 2011.