Facial Expression Recognition with Convolutional Neural Networks

Arushi Raghuvanshi
Stanford University
arushir@cs.stanford.edu

Vivek Choksi
Stanford University
vchoksi@cs.stanford.edu

Abstract

Facial expression recognition systems have attracted much research interest within the field of artificial intelligence. Many established facial expression recognition (FER) systems apply standard machine learning to extracted image features, and these methods generalize poorly to previously unseen data. This project builds upon recent research to classify images of human faces into discrete emotion categories using convolutional neural networks (CNNs). We experimented with different architectures and methods such as fractional max-pooling and finetuning, ultimately achieving an accuracy of 0.48 in a seven-class classification task.

1. Introduction

Artificial intelligence systems to recognize human emotion have attracted much research interest, and potential applications of such systems abound, spanning domains such as customer-attentive marketing, health monitoring, and emotionally intelligent robotic interfaces. In light of the important role that facial expression plays in communicating emotion in humans, there has been substantial research interest in computer vision systems to recognize human emotion.

Certain facial expressions have universal meaning. In a 1971 paper titled ”Constants Across Cultures in the Face and Emotion”, Ekman et al. identified six facial expressions that are universal across all cultures: anger, disgust, fear, happiness, sadness, and surprise [4]. These are the same emotions that modern facial expression researchers aim to identify using computer vision. Research challenges such as Emotion Recognition in the Wild (EmotiW) and Kaggle's Facial Expression Recognition Challenge present these emotions, along with the addition of a seventh, neutral emotion, for classification.

The recent success of convolutional neural networks (CNNs) in tasks such as object classification extends to the problem of facial expression recognition. In the following sections, we will present an overview of our problem, a literature review, and a report of our work.

The objective of this project is to classify images of human faces into discrete emotion categories. Many established facial expression recognition (FER) systems use standard machine learning and extracted features, which do not have significant performance when applied to previously unseen data [1]. Within the past few months a few papers have been published that use deep learning for FER [2] [13] which have been successful at achieving about .60 accuracy on the EmotiW and other publicly available data sets. Noting the success of CNNs in this domain, our objective is to experiment with both new and existing network architectures to achieve similar results on a new data set.

2. Related Work

Yu and Zhang achieved state-of-the-art results in EmotiW in 2015 using CNNs to perform FER. They used an ensemble of CNNs with five convolutional layers each [14]. Among the insights from their paper was that randomly perturbing the input images yielded a 2-3% boost in accuracy. Specifically, Yu and Zhang applied transformations to the input images at train time. At test time, their model generated predictions for multiple perturbations of each test example and voted on the class label to produce a final answer. Also interesting is that they used stochastic pooling rather than max pooling because of its good performance on limited training data.

Kim et al. achieved a test accuracy of .61 in EmotiW2015 by using an ensemble based method with varying network architectures and parameters [2]. They used a hierarchical decision tree and an exponential rule to combine decisions of different networks rather than simply using a simply weighted average, and this improved their results. They initialized weights by training networks on other FER data sets and using these weights for fine-tuning.

Mollahosseini et al. have also achieved state of the art results in FER [1]. Their network consisted of two convolutional layers, max-pooling, and 4 Inception layers as introduced by GoogLeNet. The proposed architecture was tested on many publically available data sets. It received a lower test accuracy of 0.47 on the EmotiW data set but state-of-

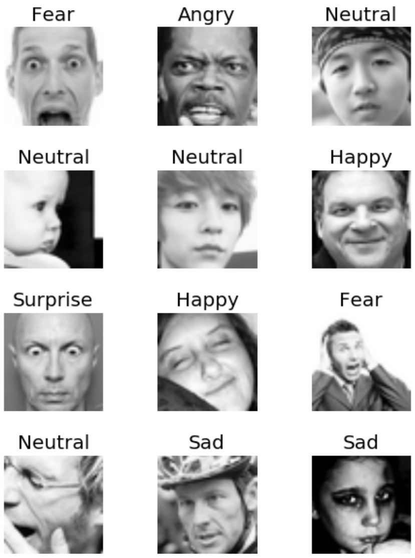

Figure 1. A sample of images from the data set, labeled with their corresponding emotions.

the-art test accuracies on other data sets (i.e. 0.93 on CK+). When compared to an AlexNet architecture, their proposed architecture improved results by 1-3 percent on most data sets.

In [5], Graham proposes a specific type of stochastic pooling, called fractional max-pooling, that achieves the regularization effect of standard max-pooling without discarding as much spatial information at each pooling step. As this method is most suited for data sets with small images, we experimented with it on our data set.

Instead of reimplementing published networks, we decided to take the key insights from these papers and experiment with different networks. All of the proposed papers used a network about 5 - 7 layers deep and image perturbation.

3. Data

We trained and tested our models on the data set from the Kaggle Facial Expression Recognition Challenge, which comprises 48-by-48-pixel grayscale images of human faces, each labeled with one of 7 emotion categories: anger, disgust, fear, happiness, sadness, surprise, and neutral. We used a training set of 28,709 examples, a validation set of 3,589 examples, and a test set of 3,589 examples.

As illustrated in Figure 1, the data set's images vary considerably in scale, viewpoint, and illumination.

We note that that all of the images are preprocessed so they form a bounding box around the face region. However, there are differences in angle, lighting, and objects – for example, some faces are adorned with glasses or covered by long hair.

4. Technical Work

We implemented three different classifiers from scratch: (1) a baseline classifier with one convolutional layer, (2) a CNN with a fixed size of five convolutional layers, and (3) a deeper CNN with a paramaterizable number of convolutional layers, filter dimensions, and number of filters. For each of these models, we tuned parameters including learning rate, regularization, and dropout. We also experimented with using batch normalization [6] and fractional max-pooling [5].

Finally, we implemented multiple classifiers using fine-tuning with variations on the number of layers retained, the number of layers backpropagated through, and the initial network used. We experimented with fine-tuning using two existing models: (1) VGG16, the model from Caffe's Model Zoo which was trained on ImageNet, and (2) VGGFace, a network trained on a facial recognition data set.

4.1. Baseline classifier

We implemented a baseline softmax classifier using features from a single convolutional layer. The architecture was one convolutional layer, followed by one fully connected layer, and a final softmax layer. The initial baseline did not use any regularization, dropout, or batch normalization.

4.2. Five-layer CNN

We implemented a first-pass CNN with a fixed depth of five convolutional layers. The model was trained using the architecture outlined in Table 1 and was trained using the following characteristics.

- Parametrized dropout rate, learning rate, and l2 regularization
- Batch normalization (optional) after each layer
- Adam update rule
- Weight initialization for using ReLU nonlinearities as presented by He et al.
- 3x3 convolutional filters with stride 1 and zero-padding to preserve spatial size
- 2x2 max pools with a stride of 2

4.3. Deeper CNN

The architecture of our deeper CNN is outlined in Table 2. We use the same network characteristics as the five-layer CNN with additional network structure parameters. With parameterizable layer depths and filter sizes, this model has a greater possible capacity than the five-layer

CNN described above. Using this model, we also experimented with both traditional max-pooling and fractional max-pooling (described below).

4.4. Fractional max pooling

Traditional max-pooling is often applied using 2x2 regions with stride 2. This configuration discards 75% of the data, and such drastic reduction in spatial size can be undesirable, particularly in data sets with small images such as ours.

Fractional max-pooling (FMP) is "gentler" in that it allows for a reduction of the spatial size of a layer by any arbitrary fraction. When the fraction is between 1 and 2, pooling regions of size 1x1, 1x2, 2x1, and 2x2 are stochastically shuffled to achieve the desired output size.

More specifically, FMP is divides a matrix into either disjoint or overlapping pooling regions as follows, and as described in more detail in [5].

$P = [a_{i-1}, a_i - 1]$ x $[b_{j-1}, b_j - 1]$ or
$P_{i,j} = [a_{i-1}, a_i]$ x $[b_{j-1}, b_j]$.

We obtain the these sequences by taking a random permutation of an appropriate number of ones and twos which denote the types of pooling regions, and then pooling within these stochastically determined regions.

We expected FMP to be useful since our data sets images are small. Using FMP would allow us to form deeper networks without as rapid data loss, at the expense of more computation. We experimented with random, overlapping fractional max-pooling as described in [5].

4.5. Fine tuning

We fine-tuned using the model VGG16 as listed in Caffe's Model Zoo and described in [7]. VGG16 was trained on ImageNET [11], which has data that is quite different than ours in content. VGG16 was trained to perform object detection and localization on images of various objects (not faces). Since the objective was different from ours but the subset of the ImageNET data set it was trained on was in the same order of magnitude as the size of our data set, we tried the following two configurations of fine-tuning: (1) using the lower few layers of the VGG16-trained network, and (2) using the entire VGG16-trained network. With both of these configurations, we trained two fully connected layers at the end (on top) of the network.

In addition, we found a network trained on face images – VGGFace – available from the Visual Geometry Group at the University of Oxford [12]. This network was trained on a very large-scale data set (2.6M images, 2.6k people) for the task of face recognition. Since the data set was trained for a similar application but on a much larger data set than ours, we tried fine-tuning with the following configuration: using the full VGGFace-trained network without the final fully connected layers. We then trained two fully connected layers at the end of this network.

Five-layer CNN
INPUT (48x48x1)
CONV3-64
BATCHNORM
RELU
CONV3-64
BATCHNORM
RELU
CONV3-64
BATCHNORM
RELU
MAX-POOL
DROPOUT
CONV3-128
BATCHNORM
RELU
CONV3-128
BATCHNORM
RELU
MAX-POOL
DROPOUT
FC-512
BATCHNORM
RELU
DROPOUT
SOFTMAX

Table 1. The architecture of the five-layer CNN.

4.6. Implementation details

All of our classifiers were implemented in Keras [3] and trained on Stanford FarmShare computers using CPUs. This configuration allowed us to experiment with many different model configurations many runs in parallel, distributed to multiple cluster machines; however, computation speed was constrained since we were using shared machines and were not using GPUs. We experimented with GPU-optimized code on AWS, but the speed gains were not significant enough to make running 20 jobs in sequence faster than running 20 jobs in parallel on different machines.

With each of our models, we performed feature-wise mean subtraction and normalization of the input data. We also performed data augmentation in the form of random horizontal shifts, random vertical shifts, and random horizontal flips.

We trained for 10 epochs on each run, saving full run history and saving the model's best weights in HDF5 format. Initially, we were tuning parameters on 10% of the data. However, we realized that these parameters were not gener-

Deeper CNN
INPUT (48x48x1)
CONVn-FILT1
BATCHNORM
RELU
n times
CONVn-FILT1
BATCHNORM
RELU
CONVn-FILT1
BATCHNORM
RELU
(FRACTIONAL) MAX-POOL
DROPOUT
m times
CONVn-FILT2
BATCHNORM
RELU
CONVn-FILT2
BATCHNORM
RELU
(FRACTIONAL) MAX-POOL
DROPOUT
FC-512
BATCHNORM
RELU
DROPOUT
FC-256
BATCHNORM
RELU
DROPOUT
FC-128
BATCHNORM
RELU
DROPOUT
SOFTMAX

Table 2. The architecture of the deeper CNN, with parameterized depths and filter sizes, and optional use of fractional or traditional max-pooling

alizing well to the full data set, possibly because it was not well shuffled, so the last half of our parameter tuning was done on the full data set.

For fractional max pooling, we adapted code for a fractional max pooling layer implementation in Lasange to Keras [10].

For fine-tuning with VGG16 we loaded in the weights from a .hdf5 weight file and replicated the network in Keras using a prototxt file as reference. For fine-tuning with VGGFace, we used Caffe [8] to load in model weights and replicated the network in Keras using the prototxt file [9].

Model	train acc	val acc	test acc
Baseline	0.25	0.25	0.24
Five-layer CNN	0.46	0.37	0.39
Deeper CNN	0.60	0.47	0.48
VGG16 fine-tuned CNN	-	0.29	-
VGGFace fine-tuned CNN	0.37	0.38	-

Table 3. A comparison of the training, validation, and test accuracies of our different models when using the best parameters we found. Due to time constraints. We omitted some fine-tuning results that we were not able to run due to time constraints.

5. Results

5.1. Comparison of accuracies

We evaluated results against Kaggle's validation and test sets. A comparison of the accuracy results from each model is shown in Table 3.

Given 7 emotion categories, random classification would give an accuracy of 0.14. All our models outperformed random, and our deeper CNN achieved the best accuracy of 0.48 on the test set.

The state of the art test accuracy for 7 emotion categories using deep networks is .61, and the top Kaggle implementation received an accuracy of .71. We hoped to reach an accuracy that approached the state of the art implementation but fell short of this benchmark by about 0.13.

Our fine-tuned models did not perform as well as our deeper CNN. Of the two fine-tuned models we experimented with, the model fine-tuned on VGGFace performed better, with a validation accuracy of 0.38. This follows intuition, since VGGFace was trained on input data more similar to our data set.

5.2. Qualitative results

Figure 2 below shows a sample of images misclassified by our highest performing deeper CNN model. We were surprised that even we were unable to correctly classify some of the images, and the emotions on some of the faces seem ambiguous. For example, we made the same misclassification as our model, classifying the surprised face as a happy face in the leftmost column in Figure 2. Perhaps our human error speaks to the inherent difficulty of recognizing complex facial expressions as discrete emotions, especially when different people (and actors, in the case of our data set) perform emotions in visually different ways.

Apart from expressional ambiguity, these misclassified images have other characteristics that may make them difficult to classify. For example, occlusion: the image in the center of the top row only shows part of a face; lighting: the image on the bottom right occurs in much lower lighting than the other images; and viewpoint variation: the image in the center column of the third row shows a profile, whereas most other images show frontal views of faces.

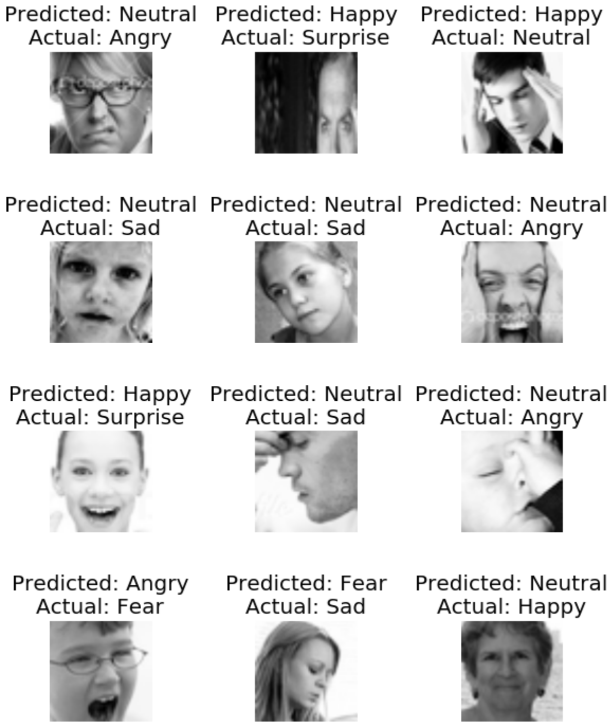

Figure 2. A sample of misclassified images with their predicted and actual labels.

6. Discussion

6.1. Baseline classifier

Our baseline classifier achieved the lowest accuracy of all our models, as expected.

6.2. Five-layer and deeper CNN

Since the five-layer CNN is a specific instance of the parameterizable deeper CNN, we will discuss both together.

6.2.1 Parameter experimentation

During the experimentation process, we tuned a large number of parameters including the following:

- Learning rate and regularization
- Dropout rate
- With vs without batch normalization
- Number of convolutional layers
- Number of fully connected layers
- Number of filters at each layer
- Fractional max-pooling vs normal max-pooling

From our incremental results, we came to the following conclusions.

Using fractional max-pooling instead of standard max pooling does not substantially affect the highest accuracy on this data set, and it increases the amount of time it takes to train a network (with identical architecture and parameters) by a factor of 1.5 due to a higher computational cost at each iteration. Also, increasing the number of fractional max-pooling layers significantly reduces the optimal L2 regularization parameter. This makes sense, because fractional max-pooling has a regularizing affect due to its stochasticity. Our optimal dropout parameters even with standard max-pooling were very low, so fractional-max pooling did not affect this parameter significantly. Figure 3 gives a direct comparison between the optimal L2 regularization weights with traditional and fractional max-pooling.

Increasing the number of fully connected layers from 1 to 4 increased our validation set accuracy by about 5 percent, from 0.42 to 0.47, indicating that we had been underfitting.

Increasing dropout decreases overfitting and increases the amount of training time to achieve accuracies achievable without dropout. This dynamic is illustrated in Figure 4.

Having a larger number of filters in the deeper layers of the network led to higher accuracies. To reduce the number of tunable variables, we separated the number of filters into two parameters such that we could specify a different number of filters for the first repeated block in Table 2 than the second repeated block, while maintaining the same number of filters for all convolutional layers within the same block. We found, for example, that using 32 filters in the first part of the network 64 filters in the second led to higher accuracies than if all convolutional layers used 32 filters. We also found that increasing the number of filters at the first layers of the network increased computation time (as expected) without significantly increasing accuracy. For example in our initial random sweep of parameters, we found that using 77 filters in the first layers did not improve accuracy as compared to using 32 filters in the first layers. This structure of using more filters in later convolutional layers is consistent with the structure of the VGG16 network.

6.2.2 Interpretation

Our CNN models underperformed compared to state-of-the-art. One weakness in our models is that they overfit the training data, as shown in Table 3 by the high training set accuracies compared to lower validation and test set accuracies. We were not able to explore this due to time constraints, but to combat this overfitting, we could have experimented with models that use more data augmentation, such

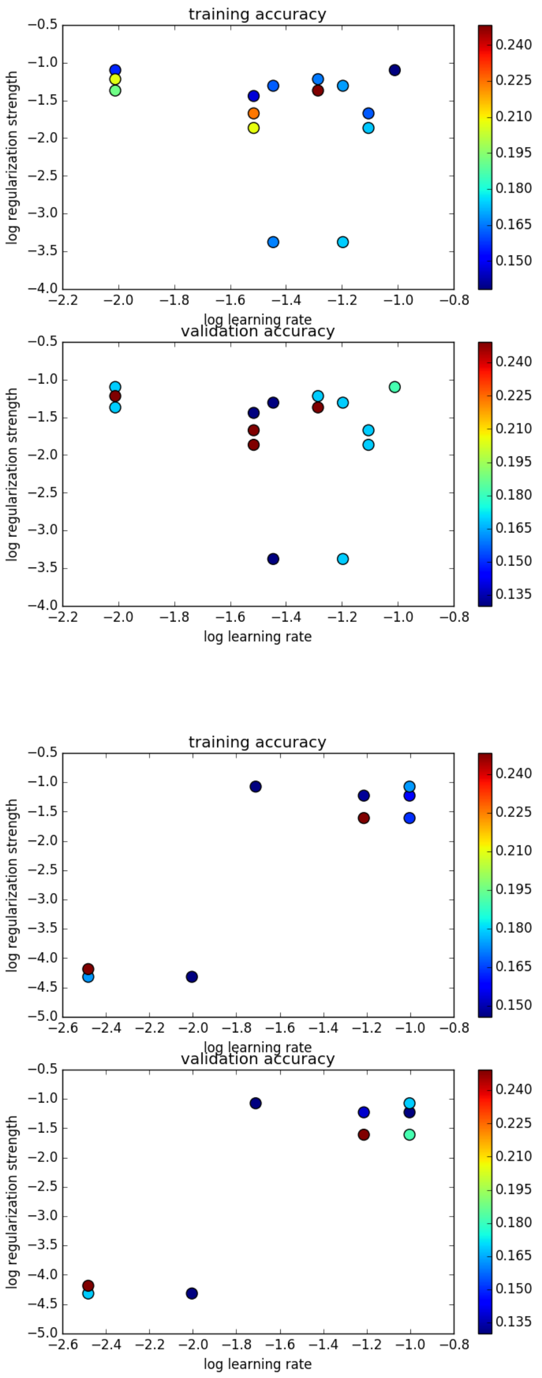

Figure 3. The upper two graphs represent the training and validation accuracy for the the five-layer CNN during learning rate and regularization tuning. The accuracies are shown by the color scale depicted to the right. The lower two graphs represent the five-layer CNN with fractional max-pooling. As depicted, the optimal L2 regularization weights with fractional max pooling are lower than those for max-pooling.

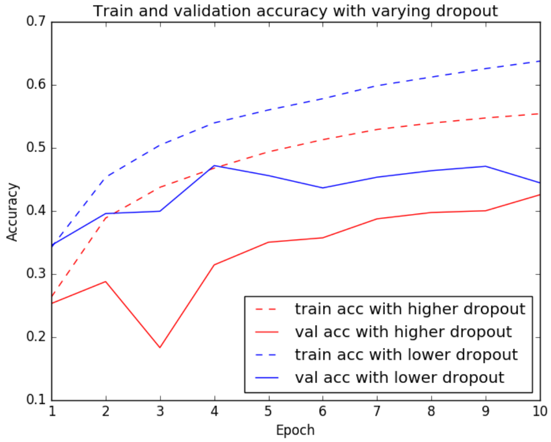

Figure 4. Higher dropout results in less overfitting but a longer time to reach accuracies achieved using the same parameters and lower dropout.

as random image rotations and more aggressive horizontal and vertical shifts.

Although our highest-accuracy model underperforms compared to the state-of-the-art, it achieves accuracies much higher than random and is training in a meaningful way. Figure 5 shows visualizations of filters from the convolutional layers in our best model. The filters show clear patterns in all layers. As a comparison point, Figure 6 shows visualizations from the convolutional layers of the VGG16 network we used for fine-tuning, at the same layer depths.

6.3. Fine-tuned CNNs

In fine-tuning using VGG16, we tried both (1) keeping only the bottom layers and training two fully connected layers on top of that and (2) keeping the whole network except for the final classification and training two fully connected layers on top of that. The first method gave a validation accuracy of 0.249 while the second gave a validation accuracy of 0.290. It was surprising that keeping the full network gave a higher accuracy than keeping just the lower layers, because we would expect the lower layers to generalize better to very different data sets. It is possible that we retained too few layers from VGG16. However, since the validation accuracies were much lower than the networks trained from scratch, we decided not experiment much further with fine-tuning on VGG16.

For fine-tuning with VGGFace we kept the full network except for the final layer and added two fully connected layers that we trained on top of that. Fine-tuning on VGGFace worked comparatively well, yielding a validation accuracy of 0.38. However, this accuracy is still than our deeper CNN's accuracy of 0.47 on the validation set, and fine-tuning VGGFace took about 3 times as long to train.

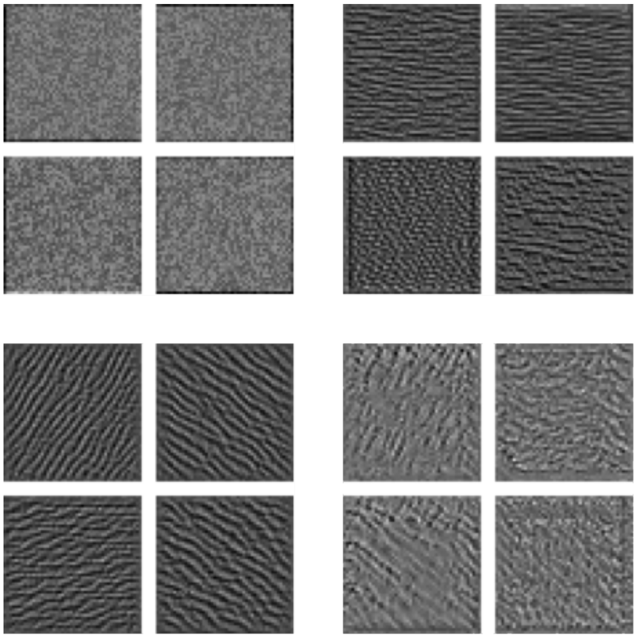

Figure 5. Visualizations of filters from the deeper CNN model. The visualizations represent filters from the first (top left), third (top right), fifth (bottom left), and seventh (bottom right) convolutional layers. These visualizations were generated by performing gradient ascent to generate an image that maximizes the activations at various layers in the network.

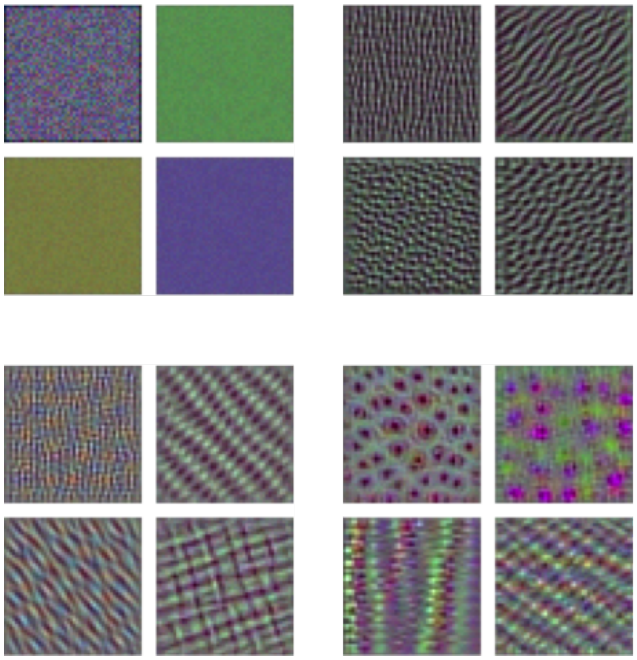

Figure 6. Visualizations of filters from VGG16. The visualizations represent filters from the first (top left), third (top right), fifth (bottom left), and seventh (bottom right) convolutional layers. These visualizations were generated by performing gradient ascent to generate an image that maximizes the activations at various layers in the network.

As a result, we decided not to continue experimenting with VGGFace, but using it in combination with other methods could be promising as future work.

It makes sense that fine-tuning with VGGFace performed much better in our FER task than fine-tuning with VGG16, because the network was trained on more similar images and for a more related objective.

Overall, however, training a model from scratch outperformed fine-tuning. This trend differs from what we have seen with other applications, but is understandable given that there has been relatively less work on face data sets compared to general image classification data sets, and given that state-of-the-art networks are less polished in this more specific task of FER. In addition, our application (FER) is not a sub-problem of the problem the network was trained to solve (e.g. classifying dog breeds vs. classifying animals); rather, FER is tangentially related to facial recognition, the task for which VGGFace was trained.

7. Conclusion

In this project, we addressed the task of facial expression recognition and aimed to classify images of faces into any of seven discrete emotion categories that represent universal human emotions. We experimented with various techniques, such as fine-tuning and fractional max-pooling, and achieved our highest accuracy (0.48) on a CNN trained from scratch with seven convolutional layers. Given more time, we would have liked to combat overfitting and approach state-of-the-art accuracies of around 0.61.

8. Acknowledgments

We thank the teaching team of CS231n for equipping us with a practical understanding of CNNs. We are also grateful to the developers of Theano and Keras and the creators of VGG16 and VGGFace for sharing their work with the community.

References

[1] D. C. Ali Mollahosseini and M. H. Mahoor. Going deeper in facial expression recognition using deep neural networks. *IEEE Winter Conference on Applications of Computer Vision*, 2016.

[2] S.-Y. D. Bo-Kyeong Kim, Jihyeon Roh and S.-Y. Lee. Hierarchical committee of deep convolutional neural networks for robust facial expression recognition. *Journal on Multimodal User Interfaces*, pages 1–17, 2015.

[3] F. Chollet. keras. `https://github.com/fchollet/keras`, 2015.

[4] P. Ekman and W. V. Friesen. Emotional facial action coding system. *Unpublished manuscript, University of California at San Francisco*, 1983.

[5] B. Graham. Fractional max-pooling. 2015.

[6] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *JMLR Proceedings*, 2015.

[7] M. S.-S. S. M. B. Z. L. X. S. B. P. J. Zhang, S. Ma and R. Mech. Salient object subitizing. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[8] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[9] Lab41. Misc, caffekeras_util. `https://github.com/Lab41/Misc/blob/master/blog/kerasvgg/caffekeras_util.py`, 2015.

[10] Lasange. Lasange, fractional pooling layer pull request. `https://github.com/Lasagne/Lasagne/pull/171`, 2015.

[11] H. S.-J. K. S. S. S. M. Z. H. A. K. A. K. M. B. A. C. B. O. Russakovsky, J. Deng and L. Fei-Fei. Imagenet large scale visual recognition challenge. *IJCV*, 2015.

[12] A. V. Omkar M. Parkhi and A. Zisserman. Deep face recognition. 2015.

[13] C. W. Pablo Barros and S. Wermter. Emotional expression recognition with a cross-channel convolutional neural network for human-robot interaction. *IEEE 15th International Conference on Humanoid Robots*, 2015.

[14] Z. Yu and C. Zhang. Image based static facial expression recognition with multiple deep network learning. *ICMI Proceedings*.