-COS 429 Final Project Report-

Fine Tuning a Facial Emotion Identification CNN on Distorted and Undistorted Images

Sydney Berry ‘24 & Advika Srivastava ‘24

Abstract

For this project, we examined the test accuracy, test loss, confusion matrix and Grad-Cams for a facial emotion recognition CNN model from an Analytics Vidya blog post [1]. From our analysis, we concluded that the model was overfitting to the training data. We then experimented with various dropout percentages and numbers of layers in the CNN to see how they affected the CNN’s performance. We then created a final model with a 65% dropout rate and only two-fifths of the original model's layers. These changes greatly reduced the test loss for both undistorted and distorted images. In addition, we compared our final model to an EfficientNet transfer learning model. We found that our final model had a higher test accuracy and lower test loss on all image test sets than the EfficientNet model.

1. Introduction and Motivation

Facial expressions are very effective in communicating emotion, and analyzing them for emotion recognition is an important computer vision task with a lot of significant applications, such as aiding in lie detection, psychiatric observations, drunk driver recognition, and human-computer interaction [2]. Due to the significance of the facial emotion recognition task, it is important to ensure that we have models that generalize well on more realistic images, such as those that have blurs and occlusions, in addition to normal undistorted images. Especially after the pandemic, when most individuals have been wearing masks which hide their mouth, it is important to have

robust emotion recognition models that can recognize facial emotions even when a part of the face is covered. Most papers focus on analyzing their models on structured test sets such as FER-2013. These papers often do not analyze how their facial emotion recognition models perform on more realistic, distorted images. Moreover, research has shown that convolutional neural networks bring several added advantages to the facial emotion recognition task because of their ability to work well with irregularities such as excessive makeup and different facial poses [2]. Thus, we focus on the Convolutional Neural Network approach in this paper. Our goal is to pick a baseline CNN-based emotion recognition model, analyze it using quantitative and qualitative methods on both distorted and undistorted images, and improve its overall performance. We also implement a transfer learning approach using the EfficientNet for this task, and compare it to the baseline model, and our changed baseline model to determine the factors that allow for the best performance.

2. Related Work

There have been several approaches to the facial emotion recognition task with convolutional neural networks. Mehendale uses a two part convolutional neural network where the first part extracts the face from the background and the second part focuses on obtaining a facial feature vector or an expressional vector [2]. This vector is used to make the classification decision for one of five emotion classes [2]. Debnath et al. use a four-layer convolutional neural network with preprocessing and feature extraction. This allowed them to obtain their goal of reducing the number of training epochs needed and combatting very long runtimes. Some of Debnath et al.'s preprocessing techniques were similar to Mendales' in that the model started by acquiring the face present in the image through facial detection and head pose estimation. Next, Debnath et al.

used Local Binary Pattern (LBP) to extract the textures of the images, which allowed them to develop their model to be robust to fluctuations of lighting [3]. When researching how to make convolutional neural networks more robust to distortions, Hossain et al found success by incorporating discrete cosine transform within their baseline neural network [4]. Kuruvayil and Palaniswamy employed a meta-learning approach to create an emotion classification model that is robust to partial occlusions, varying head poses, and illumination levels [5]. These two approaches have shown to improve performance of facial emotion detection models on distortions like occlusions, blurs, and illumination differences.

Additionally, there is some research on the use of EfficientNet in the context of facial emotion recognition. Compared to other CNNs that are slower and require a lot of computation power, EfficientNet is especially useful due to its ability to achieve good accuracy levels while not being extremely deep. Utami et al. used EfficientNet as a feature extractor for their emotion classification model, in addition to other layers, and were able to increase accuracy. Therefore, EfficientNet has the potential to be useful in emotion recognition tasks [19].

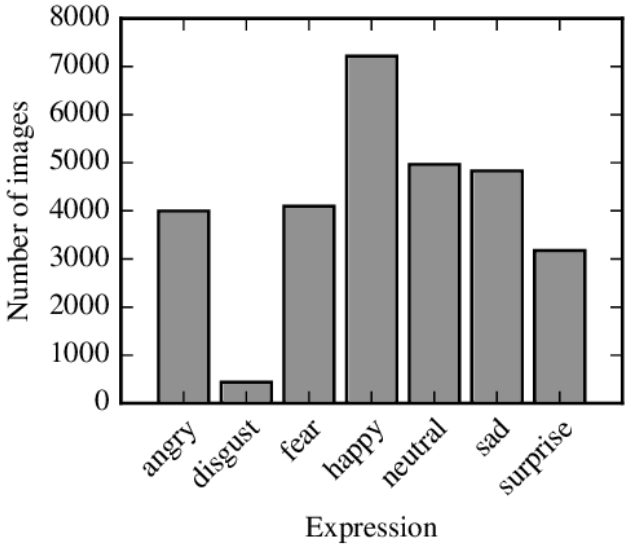
While all these researchers have improved emotion recognition, most of their projects were heavily dependent on preprocessing methods. These models would require preprocessing test data before making a prediction, which could increase runtimes on large test sets. In this project, we will explore if we can adjust parameters within the convolutional neural networks themselves to improve the models' performances without having the need to add additional preprocessing or feature extraction methods that these papers employ. Our baseline model, which came from a popular online blog titled Analytics Vidya [7], utilizes convolution layers, ReLu activation

functions, pooling layers, batch normalization, and a soft max layer as seen in Debnath et al.'s model [3].

3. Dataset

We will use the FER-2013 dataset for our training, validation, and testing sets. This dataset contains 48x48 pixel grayscale images of faces and is intended for the task of facial emotion detection. Each image has been labeled with one of the 7 categories - angry (0), disgust (1), fear (2), happy (3), sad (4), surprise (5) and neutral (6) [7].

Figure 1. Emotion distribution for the FER training dataset (Reprinted from ref 6. Copyright 2018.)



In each image, the face occupies about the same amount of space and the face is centered differently in each of the images. The training dataset contains 28709 images and the available testing dataset contains 3589 images. We created our own validation set by taking 15% of the training dataset. Thus, these were our final counts - training set = 24,402, validation set = 4307, and testing set = 3,589 [7]. Another testing dataset was created using the original FER-2013

testing set by applying distortions to the images in this set; this set is referred to as the "distorted testing set." The process to create this dataset is described in the *Implementation* section.

4. Approach

We first found a baseline model online on a blog titled Analytics Vidya [1]. We evaluated the baseline using our evaluation techniques outlined below on both our distorted and not distorted test sets. We will use our observations from our evaluations to design experiments to see how changing particular variables affects the accuracy of our models. We will use what we learn from these experiments to design our final model. We will then compare our final model to our baseline model and to the EfficientNet model we implemented.

4.1 Evaluation Techniques

Quantitative - Our main quantitative evaluation techniques are test accuracy and test loss. We also looked at the confusion matrices for our baseline and final models. The confusion matrix allowed us to visualize the models' performance on each emotion category and to analyze whether there were particular categories in which one model performed better on than another.

Qualitative - Our main qualitative evaluation technique was grad-cams. The grad-cam images are very helpful in understanding CNNs since the localization map visualizes regions in the image that were particularly important in making the model's prediction [8]. We implemented grad-cam by using Keras [9].

All these techniques were applied to both the original FER-2013 test dataset and the distorted test dataset (the one that we create).

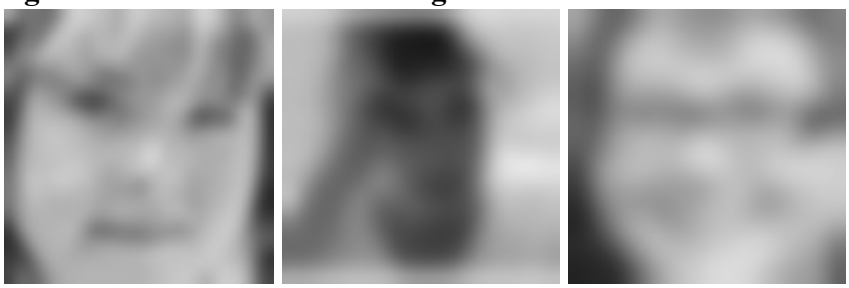
5. Implementation

5.1 Implementing the Distorted Images

The following 5 distortions were chosen for this project - gaussian blur, gaussian noise, salt and pepper blur, speckle blur and occlusion. These are some of the distortions used to evaluate the emotion detection model in [4] and thus, it made sense to use the same categories to evaluate our model. Additionally, testing on occlusions is especially relevant due to the usage of masks during the pandemic. We divided the testing dataset into 5 sections and applied one distortion to each section to obtain our distorted testing dataset. Each section contained either 717 or 718 images.

5.1.1 Gaussian blur - We applied a gaussian blur to images by using the cv2 gaussian blur method and a kernel size of 7. Since the cv2 gaussian blur method returns an image with two dimensions, we had to expand the image to 3 dimensions using the numpy expand dimensions method.

Figure 2. Gaussian Blurred Images from our Distorted Test Set



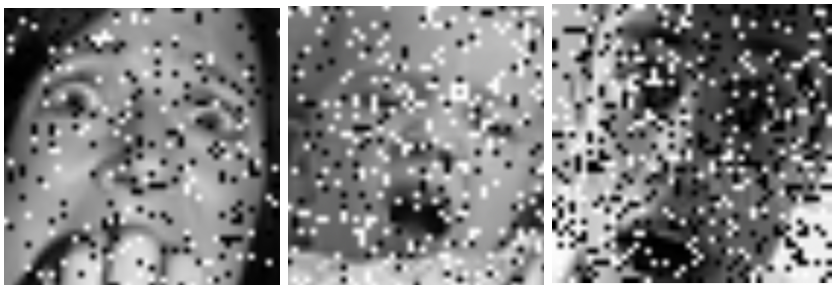
5.1.2 Gaussian noise - We based our gaussian noise implementation on a GitHub discussion centered around debugging a gaussian noise function definition [10]. In doing so, we had to debug Prasad9's implementation by changing the random function and modifying the code to work with grayscale images instead of colored images. After we got the function working, we experimented with different variance values for calculating the gaussian noise. We ultimately

chose a variance value of 4, because this was the value where we started to be able to see gaussian noise in our images with our naked eyes.

Figure 3. Images with Gaussian Noise from our Distorted Test Set

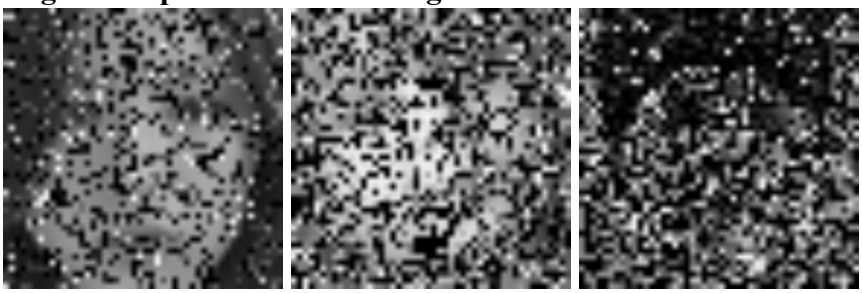
5.1.3 Salt & Pepper Blur - We used a salt and pepper blur definition from Geeks for Geeks to create this distorted dataset [11]. Since our pictures were significantly smaller than the pictures this function was designed for, we adjusted the number of pixels that were changed by the function to be 30 to 500 per color (white and black). The exact number of pixels changed was determined by a random number generator.

Figure 4. Salt & Pepper Blurred Images from our Distorted Test Set



5.1.4 Speckle Blur - We modified code from the AI learner to create a for loop that added a speckle blur to a set of images [12]. This adaptation was relatively straightforward and only involved switching and saving variables with the names we were using and creating the for loop.

Figure 5. Speckle Blurred Images from our Distorted Test Set



5.1.5 Occlusion - We wrote a function that used a random number generator to determine what quarter of a picture to cover with a cv2 black rectangle. The various rectangles we used are all shown in the images below.

Figure 6. Images with Occlusion from our Distorted Test Set

5.2 Implementing Baseline model

We chose to run all of our code using Google Colab. When training our models, we changed the runtime type to GPU to reduce how long it took to train the model. We based our baseline model on an emotion identification CNN that we found instructions on how to implement in an Analytics Vidhya article written by Thetechwriters [1]. When implementing, we had to make some minor substitutions to the original code since there were two functions that would not properly load in Google Colab. In addition, we used a Google Colab patch for the cv2 show method in order to see the pictures. The code uses pandas to read in the fer2013.csv that we mounted to colab using our google drives. The code then parsed through the dataset and added

pixel values and labels into arrays, which resorted and expanded the data to work with the model. The baseline model implemented data augmentation, which would randomly rescale, rotate by up to 10 degrees, and horizontally flip or shift images. The baseline model was implemented using Tensor flow and had the following layers:

```
# first input model
  visible = Input(shape=input_shape, name='input')
  num_classes = 7

  #the 1-st block
  conv1_1 = Conv2D(64, kernel_size=3, activation='relu', padding='same', name = 'conv1_1')(visible)
  conv1_1 = BatchNormalization()(conv1_1)
  conv1_2 = Conv2D(64, kernel_size=3, activation='relu', padding='same', name = 'conv1_2')(conv1_1)
  conv1_2 = BatchNormalization()(conv1_2)
  pool1_1 = MaxPooling2D(pool_size=(2,2), name = 'pool1_1')(conv1_2)
  drop1_1 = Dropout(0.3, name = 'drop1_1')(pool1_1)

  #the 2-nd block
  conv2_1 = Conv2D(128, kernel_size=3, activation='relu', padding='same', name = 'conv2_1')(drop1_1)
  conv2_1 = BatchNormalization()(conv2_1)
  conv2_2 = Conv2D(128, kernel_size=3, activation='relu', padding='same', name = 'conv2_2')(conv2_1)
  conv2_2 = BatchNormalization()(conv2_2)
  conv2_3 = Conv2D(128, kernel_size=3, activation='relu', padding='same', name = 'conv2_3')(conv2_2)
  conv2_3 = BatchNormalization()(conv2_3)
  pool2_1 = MaxPooling2D(pool_size=(2,2), name = 'pool2_1')(conv2_3)
  drop2_1 = Dropout(0.3, name = 'drop2_1')(pool2_1)

  #the 3-rd block
  conv3_1 = Conv2D(256, kernel_size=3, activation='relu', padding='same', name = 'conv3_1')(drop2_1)
  conv3_1 = BatchNormalization()(conv3_1)
  conv3_2 = Conv2D(256, kernel_size=3, activation='relu', padding='same', name = 'conv3_2')(conv3_1)
  conv3_2 = BatchNormalization()(conv3_2)
  conv3_3 = Conv2D(256, kernel_size=3, activation='relu', padding='same', name = 'conv3_3')(conv3_2)
  conv3_3 = BatchNormalization()(conv3_3)
  conv3_4 = Conv2D(256, kernel_size=3, activation='relu', padding='same', name = 'conv3_4')(conv3_3)
  conv3_4 = BatchNormalization()(conv3_4)
  pool3_1 = MaxPooling2D(pool_size=(2,2), name = 'pool3_1')(conv3_4)
  drop3_1 = Dropout(0.3, name = 'drop3_1')(pool3_1)
```

```
#the 4-th block
conv4_1 = Conv2D(256, kernel_size=3, activation='relu', padding='same', name = 'conv4_1')(drop3_1)
conv4_1 = BatchNormalization()(conv4_1)
conv4_2 = Conv2D(256, kernel_size=3, activation='relu', padding='same', name = 'conv4_2')(conv4_1)
conv4_2 = BatchNormalization()(conv4_2)
conv4_3 = Conv2D(256, kernel_size=3, activation='relu', padding='same', name = 'conv4_3')(conv4_2)
conv4_3 = BatchNormalization()(conv4_3)
conv4_4 = Conv2D(256, kernel_size=3, activation='relu', padding='same', name = 'conv4_4')(conv4_3)
conv4_4 = BatchNormalization()(conv4_4)
pool4_1 = MaxPooling2D(pool_size=(2,2), name = 'pool4_1')(conv4_4)
drop4_1 = Dropout(0.3, name = 'drop4_1')(pool4_1)

#the 5-th block
conv5_1 = Conv2D(512, kernel_size=3, activation='relu', padding='same', name = 'conv5_1')(drop3_1)
conv5_1 = BatchNormalization()(conv5_1)
conv5_2 = Conv2D(512, kernel_size=3, activation='relu', padding='same', name = 'conv5_2')(conv5_1)
conv5_2 = BatchNormalization()(conv5_2)
conv5_3 = Conv2D(512, kernel_size=3, activation='relu', padding='same', name = 'conv5_3')(conv5_2)
conv5_3 = BatchNormalization()(conv5_3)
conv5_4 = Conv2D(512, kernel_size=3, activation='relu', padding='same', name = 'conv5_4')(conv5_4)
conv5_4 = BatchNormalization()(conv5_4)
pool5_1 = MaxPooling2D(pool_size=(2,2), name = 'pool5_1')(conv5_4)
drop5_1 = Dropout(0.3, name = 'drop5_1')(pool5_1)

#Flatten and output
flatten = Flatten(name = 'flatten')(drop5_1)
output = Dense(num_classes, activation='softmax', name = 'output')(flatten)
```

The dataset was then trained on the model using 100 epochs with a batch size of 64.

5.3 Implementing Auxilarity Code to Aid Testing

EfficientNet - We implemented the EfficientNet transfer learning model by using Keras as a reference [13]. The last layer of the EfficientNet model was not used, and we instead decided to connect it to a dense layer that uses softmax activation for the final 7 category emotion

classification. We picked the EfficientNetb0 model because of the low resolution of the FER-2013 images since EfficientNetb0 works better than all the other EfficientNet models when dealing with low resolution images [13]. Additionally, our adapted EfficientNet model takes in images whose last coordinate has 3 dimensions, like RGB images. Therefore, it was not fully compatible with FER-2013's grayscale images. After reading discussions online on Stackoverflow and Github, we decided to expand the dimensions of the images to 3 by copying values of the first dimension to the second and third [14]. We compared our final model to our EfficientNet transfer learning model's results.

Image Generator and Separate Validation Set - We decided to create a random image generator to replace the data augmentation generator for some of our experiments. Our generator creates batches of training and validation images randomly. Additionally, the baseline model that we obtained from Analytics Vidhya uses the FER-2013 testing set as both the validation and testing set. We decided to take 15% of the training set to create a separate validation set. This would ensure that we test on images that the model has never seen before to get a valid estimate of the model's generalization ability and its performance on unseen data.

5.4 Implementing evaluation methods

We used Keras' plt_metric definition to create plots of the training and validation sets' accuracies and losses.We used tensor flow's evaluate method to get the test accuracy and test loss for our models. We then saved the models to our google drives, so we could then reload them into our other testing files.

In one of our testing files, we wrote a for loop that used np where to read our labels and the output of tensor flow's prediction method to create 1-dimensional arrays that stored the actual labels and predicted values. We used these arrays to get a confusion matrix for the test set. In addition, we implemented grad-cam using a StackOverflow thread as a jumping off point [15]. We had to modify the provided code to work with the different sizes and dimensions of our input.

We created a second testing file to create confusion matrices and grad cam images for our distorted dataset.

6. The Baseline Model

6.1 Evaluating the baseline model

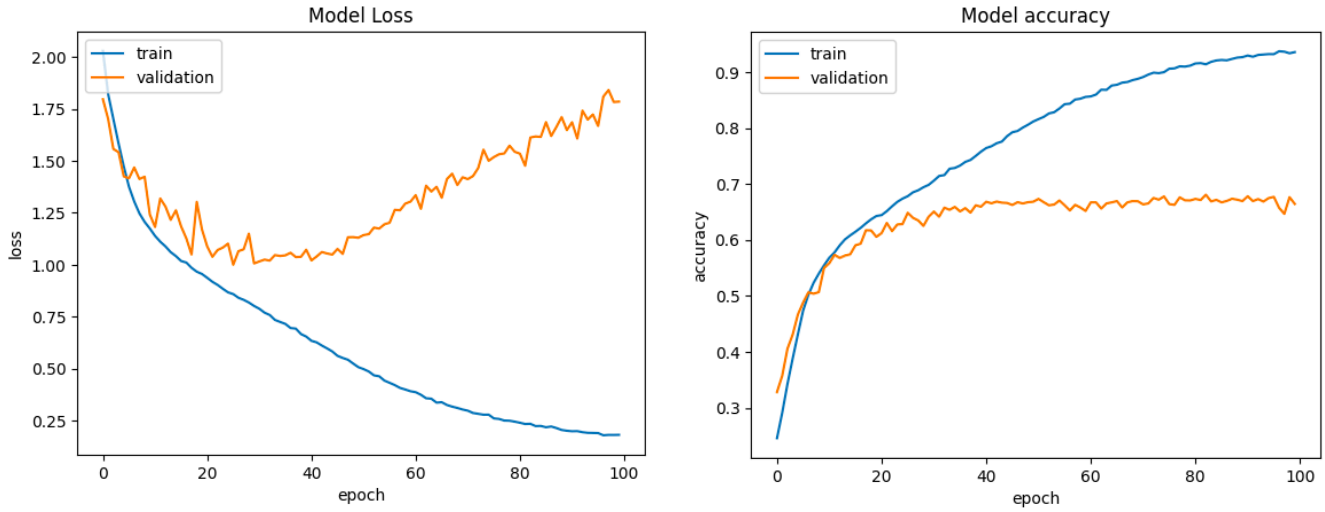
6.1.1 Training and Validation

The baseline model obtained from Analytics Vidya was first evaluated using our evaluation techniques to see how it could be improved. The hyperparameters used by the model were learning rate=0.0001, decay=1e-6, and number of epochs=100. The data augmentation part of the model was slightly edited to remove rescaling; everything else in the model including the hyperparameters was kept the same.

As can be seen in Figure 7 on the following page, the model is able to obtain a very high training accuracy and very low training loss after 100 epochs. However, the validation accuracy plateaued after about 20-30 epochs and the validation loss started increasing after that point. The

final validation loss was noticeably higher than the final training loss. Overall, this indicates that the model is overfitting to the training images.

Figure 7: Training and validation losses and accuracy for baseline model



6.1.2 Performance on Non-distorted Images

The overall test accuracy was 68.7%, while the overall test loss was 1.36. Additionally, the following confusion matrix was obtained:

```
[[ 59  11  55 126  90  69 101]
 [  9   0   4  13   5   8  10]
 [ 64   7  67 136  80  68  94]
 [116   9 110 216 177 113 162]
 [ 88   3  63 173 103  73  94]
 [ 54   7  46  89  71  53  61]
 [ 79  10  73 147 122  77 124]]
```

This showcases that the baseline model has a decent test accuracy and is able to generalize well on test data. However, one thing to note is that the baseline model uses the same set for testing and validation (the FER-2013 test set), therefore, this might not be the best indication of the baseline model's ability to generalize on unseen data. Another interesting observation from the

confusion matrix is how the baseline model is unable to categorize any of the images from category 1 (disgust) accurately.

The grad-cam in Figure 8 was obtained by looking at the final convolutional layer (in this case, layer conv5_4). The grad-cam showcases that the model is focusing on some of the important regions, like part of the mouth, but is not able to capture all important features to predict emotion. For instance, in the image below, the primary focus is on one side of the woman's face and part of her mouth, but other important features like her eyes and eyebrows are not captured.

Figure 8 : Grad-Cam on Conv5_4 for baseline model

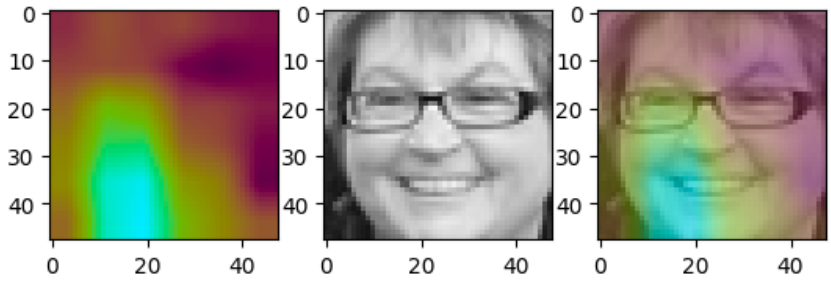
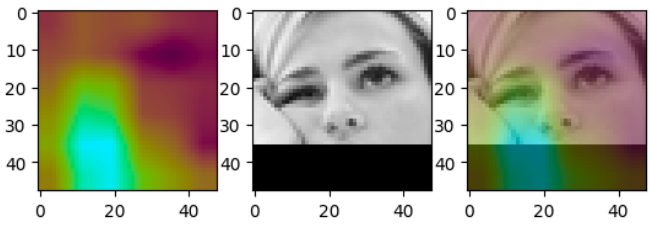


Table 1: Test accuracy and loss of baseline model on various distortions for baseline model

Distortion Type	Test Loss	Test Accuracy
Gaussian blur	3.55	0.360
Gaussian noise	3.57	0.425
Salt and pepper blur	8.28	0.171
Speckle blur	7.23	0.185
Occlusion	3.27	0.404
Overall	4.86	0.355

Overall, the test accuracy for the distorted images is 0.35 and the test loss is 4.86, with the model being the most accurate for the gaussian noise set and having the lowest test loss for the occlusion image set. The model does poorly on the salt and pepper blur set, where the test accuracy is only 17.1%. This is only very slightly higher than a random guess.

Figure 9: Grad-Cam on Conv5_4 for baseline model with occlusion



Additionally, with distortions like occlusion, the model seems to be focusing on the part that is distorted instead of focusing on important features, like the eyes in the image above. Therefore, it seems that the model is unable to capture all the important features in the face. This is one thing that the model needs to be improved on. Additionally, we can see how the two grad cams for the two different images with different target classes are very similar. This might indicate that the model is not able to understand each image and capture its unique important features.

6.1.3 Training with a separate validation dataset

When we created a separate validation set from the training set instead of using the test set, the test accuracy slightly decreased and the test loss decreased. The new test accuracy was 67.1% and the new test loss was 1.34. These numbers give a better indication of how the baseline model generalizes on unseen data. The new confusion matrix is shown on the following page.

```
[[ 57  10  72 135  81  59  97]
 [  8   0   7  11   5   6  12]
 [ 63  12  79 129  67  59 107]
 [104   8 133 227 162  98 171]
 [ 88   3  67 168  94  67 110]
 [ 55   5  53  98  64  43  63]
 [ 70   8  77 150 124  66 137]]
```

Table 2: Test accuracy and loss of baseline model on various distortions for baseline model with proper validation set

Distortion Type	Test Loss	Test Accuracy
Gaussian blur	4.02	0.289
Gaussian noise	3.01	0.435
Salt and pepper blur	7.38	0.181
Speckle blur	9.94	0.192
Occlusion	3.00	0.373
Overall	5.17	0.337

Since these numbers give a better indication of how well the model generalizes on unseen data, we will be using these as our baseline results for test accuracy and losses on the undistorted and distorted test datasets. The trends in these results are similar to the ones that we observed before- the model has the highest test accuracy for gaussian noise and has the lowest loss for the occlusion set.

7. Hypotheses

As we saw in the evaluation of the baseline, the model seems to be overfitting. To employ other ways of reducing overfitting besides data augmentation, we looked at reducing model complexity and increasing dropout. We only wanted to see the impact of changing these features

so we decided to remove data augmentation in our model for now, and replaced it with our random image generator.

Hypothesis #1 - The baseline model has 5 blocks of layers. We hypothesize that there would be a midpoint where the model is complex enough to effectively fit our training data but not too complex that it overfits. Our prediction is that 3 blocks of layers would lead to the highest test accuracy on the original test dataset and the distorted test dataset.

Hypothesis #2 - We hypothesize that there will similarly be a point with the dropout percentage where the model will be general enough to learn general trends of sentiment expression while not overfitting to minute details in the training set. We hypothesize that when we increase the dropout rate, the test loss will decrease, because the model will not overfit to small features.

8. Results and Evaluation

Our models provided results that explored two main topics. How does the complexity of a CNN and the number of convolution and pooling layers it has affect overall accuracy on normal and distorted images? How does the use of various dropout percentages affect overall accuracy on normal and distorted images? We kept the hyperparameters the same while running these experiments, with learning rate=0.0001, decay=1e-6, and number of epochs=100, to see the effect of only changing the dropout and the number of layers.

8.1 Number of Layers

When analyzing the baseline model, we determined that it was divided into 5 natural blocks. For the purpose of this paper, we will define a block of layers as the culmination of all layers that end

in a dropout layer. Following this definition, every block in our CNN network contains 2 to 4 convolution layers and 1 pooling layer. The full model from the Analytics Vidhya blog originally contained 5 blocks. We experimented with removing blocks and seeing how this impacted test accuracies. We decided to do this because the initial model was seen to be overfitting and one way to reduce overfitting is to reduce the complexity of the model; removing layers is an effective way to reduce model complexity [16]. Table 3 shows the test accuracies and test losses for undistorted images on models containing various numbers of blocks of layers. Tables 4 through 8 show the test accuracies and test losses for images with various distortions on models containing various numbers of blocks of layers.

Table 3: Test Accuracies and Losses for various numbers of blocks of layers on not distorted images

Blocks of layers in model	Test Accuracy	Test Loss
1	0.509	1.79
2	0.622	1.22
3	0.631	1.86
4	0.552	3.01
5	0.614	2.31

From this table, we concluded that a CNN containing 3 blocks of layers was the most accurate when examining undistorted data. It was only slightly more accurate than a CNN containing 2 layers, by 0.009. However, the CNN containing 2 layers had the lowest test loss.

Table 4: Test Accuracies and Losses for gaussian blurred images for various numbers of blocks of layers

Blocks of layers in model	Test Accuracy	Test Loss
1	0.338	1.83
2	0.283	2.07
3	0.307	3.24
4	0.272	5.66
5	0.286	3.51

From this table, we concluded that the CNN model that had the highest accuracy and lowest loss on images distorted using a gaussian blur only contained 1 block of layers. This was the model that was the least accurate on undistorted images.

Table 5: Test Accuracies and Losses for images containing gaussian noise for various numbers of blocks of layers

Blocks of layers in model	Test Accuracy	Test Loss
1	0.426	1.94
2	0.429	1.80
3	0.454	3.30
4	0.416	4.60
5	0.462	3.28

From this table, we found that the CNN model that had the highest test accuracy on images containing gaussian noise contained 5 blocks of layers. However, the model with 2 blocks of layers had the lowest test loss.

Table 6: Test Accuracies and Losses for salt and pepper blurred images for various numbers of block of layers

Blocks of layers in model	Test Accuracy	Test Loss
1	0.248	12.6
2	0.182	4.06
3	0.170	10.4
4	0.162	9.12
5	0.213	5.91

From this table, we concluded that the CNN model that had the highest test accuracy on images distorted using a salt and pepper blur only contained 1 block of layers. However, the model with 2 blocks of layers had the lowest test loss, while the test loss of the model with 1 block of layers was almost 3 times the test loss of the model with 2.

Table 7: Test Accuracies and Losses for speckle blurred images for various numbers of block of layers

Blocks of layers in model	Test Accuracy	Test Loss
1	0.234	19.4
2	0.203	3.34
3	0.177	15.4
4	0.149	7.80
5	0.199	6.66

From this table, we concluded that the CNN model that had the highest test accuracy on images distorted using a speckle blur only contained 1 block of layers. However, the model with 2 blocks of layers had the lowest test loss, while the test loss of the model with only one layer was drastically higher.

Table 8: Test Accuracies and Losses for images with occlusions for various numbers of block

Blocks of layers in model	Test Accuracy	Test Loss
1	0.337	3.72
2	0.409	2.35
3	0.329	4.99
4	0.326	4.89
5	0.383	3.94

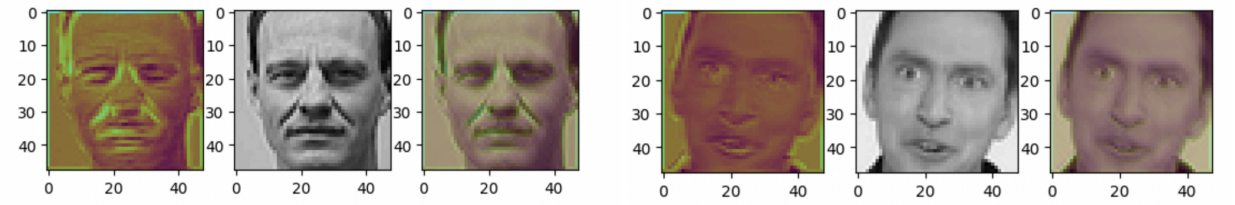
From this table, we concluded that the CNN model that performed the best on images containing occlusions contained 2 blocks of layers. It had the highest test accuracy and lowest test loss.

From these test accuracies, we found that the CNN model containing 1 block was the most accurate on all the blurred images, the CNN model containing 2 blocks was most accurate with images containing occlusions, the CNN model containing 3 blocks was most accurate with the not distorted images, the CNN model containing 5 blocks was most accurate with the images containing Gaussian noise. For all test sets except the gaussian blurred image set, the lowest test loss was associated with the model with two blocks of layers. Furthermore, the model with two blocks of layers had the second lowest test loss for gaussian blurred images. Overall, it appears that simpler models are better at identifying images that contain distortions. This makes sense since simpler models allow for more generalization. In addition, test accuracy was improved on normal images by decreasing complexity but only to a certain extent. The model containing 4 and 5 levels started learning nuances of the training set and thus the model started becoming slightly less accurate. Our hypothesis that the model with 3 blocks of layers would be the most accurate was only somewhat true. The model with 3 blocks of layers had the highest test accuracy for undistorted images, and the second highest test accuracy for Gaussian blur and

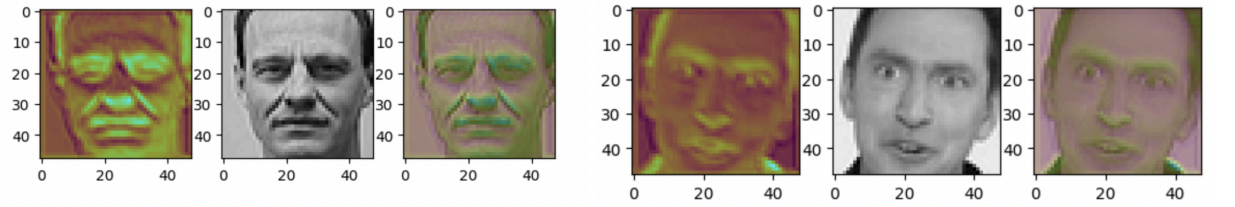
Gaussian noise. It appears that the model with 2 blocks of layers might work better as it often had the lowest test loss.

In order to determine what work each layer was doing, we decided to examine the grad-cam images for each layer. In order to do this, we had to temporarily remove the batch normalization from every convolutional layer. With batch normalization, the grad-cam images only included vague squares of where faces were normally found by the Conv1_2 layer. Without batch normalization, the features in the images were apparent in the grad-cam through many more layers. Thus, this allowed us to gain a better understanding of what each layer was learning. The grad-cam images we got for each convolutional layer was as follows:

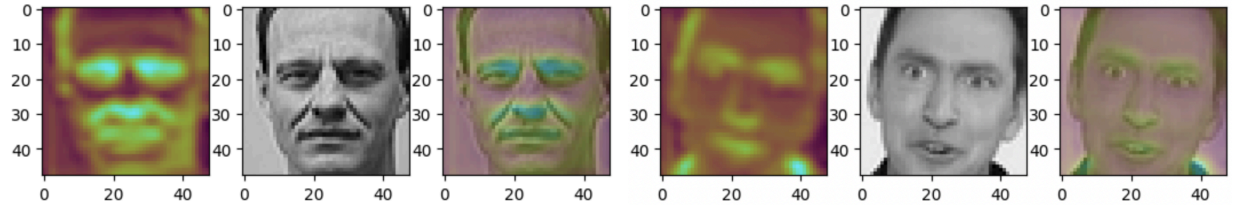
Conv1_1:



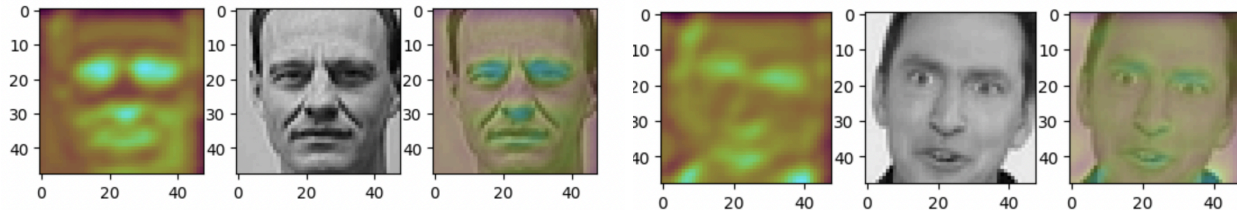
Conv1_2:



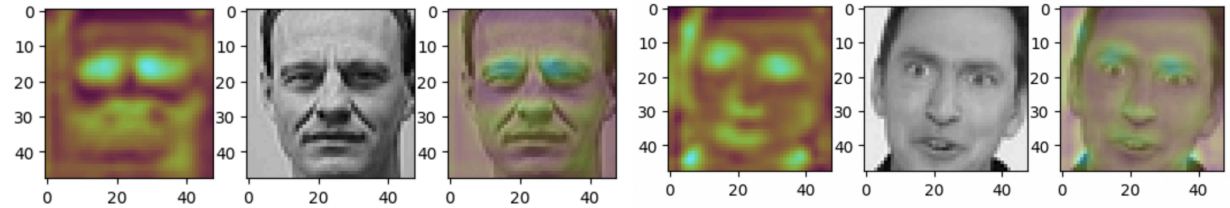
Conv 2_1:



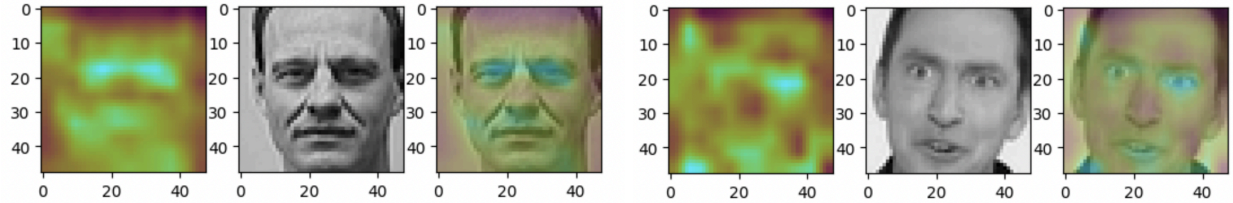
Conv 2_2:



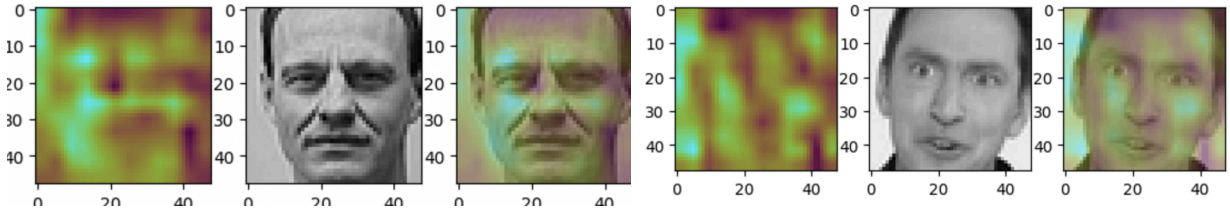
Conv 2_3:



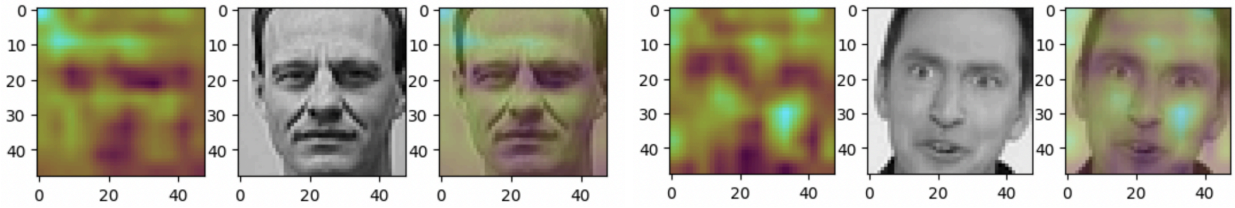
Conv 3_1:



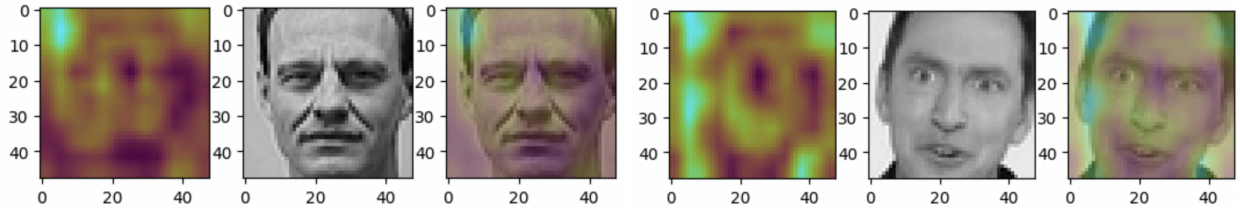
Conv 3_2:



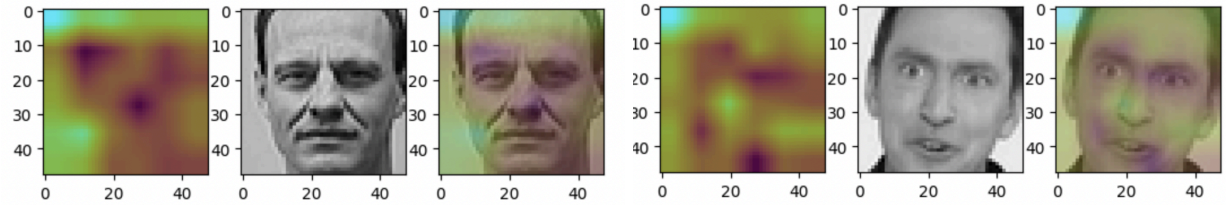
Conv 3_3:



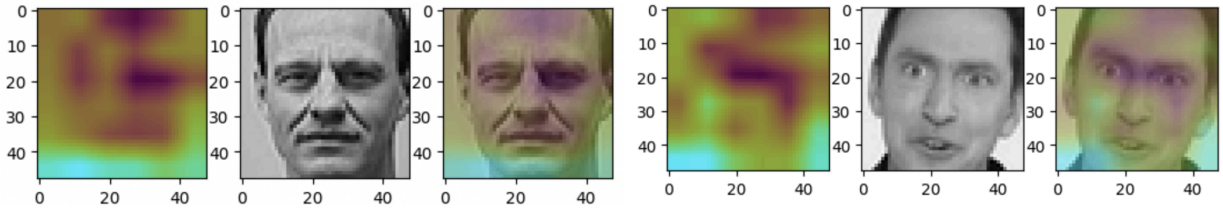
Conv 3_4:



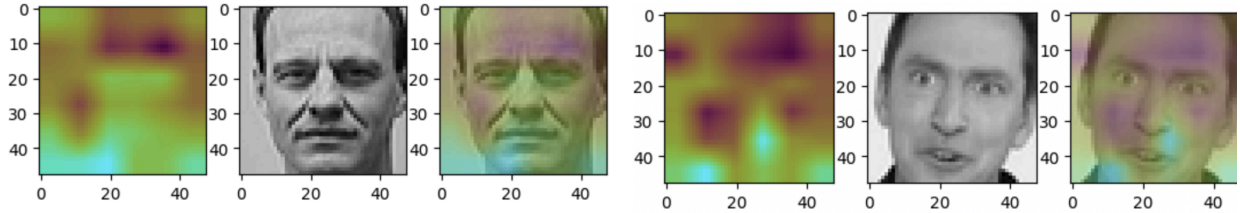
Conv 4_1:



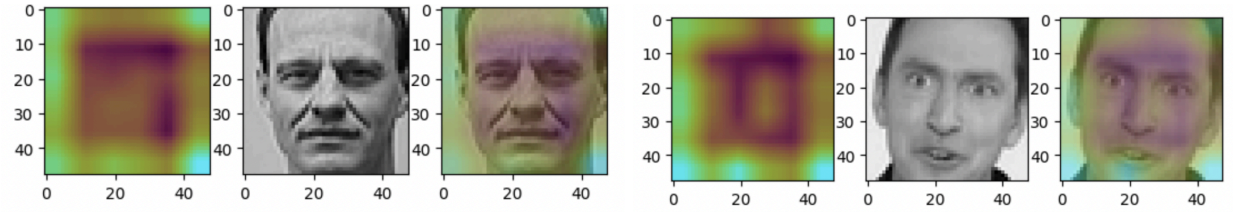
Conv 4_2:



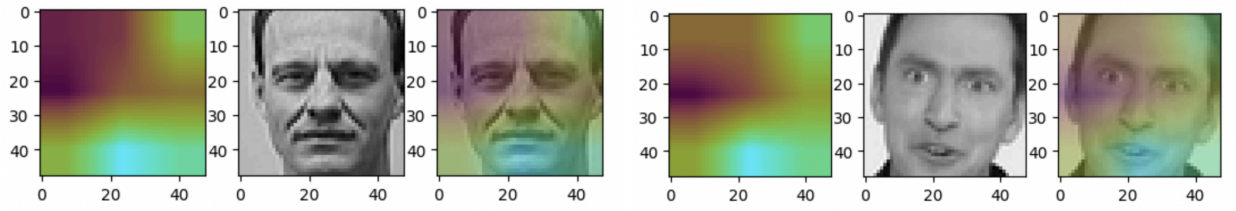
Conv 4_3:



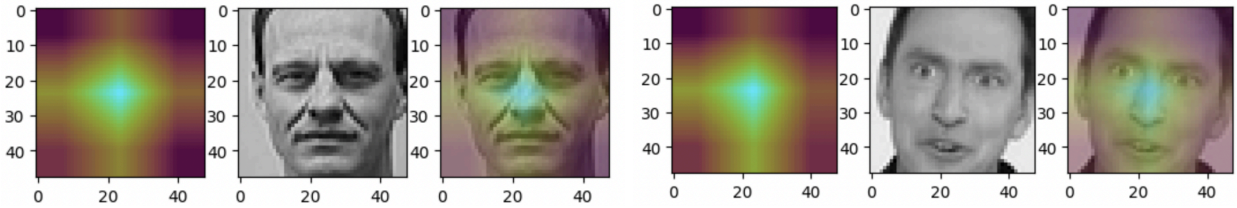
Conv 4_4:



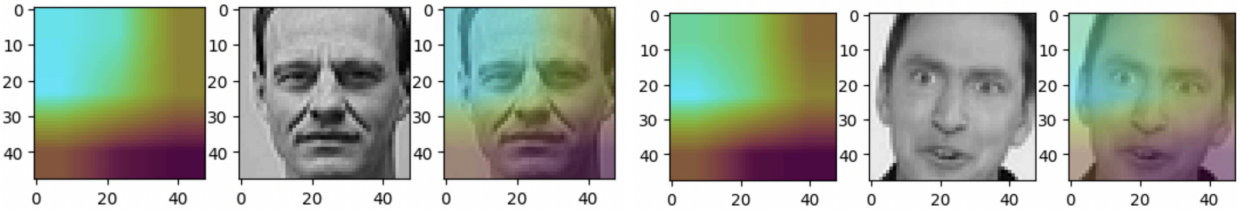
Conv 5_1:



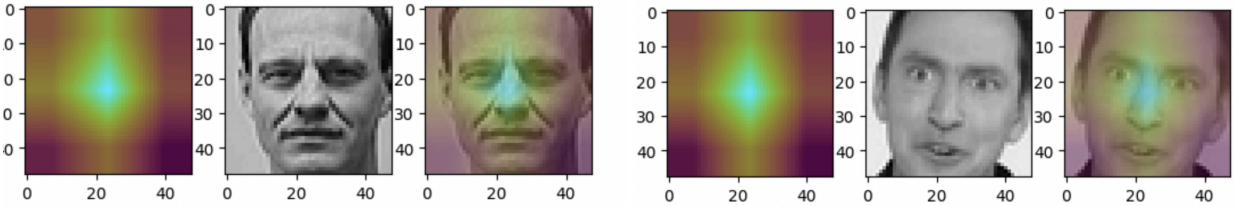
Conv 5_2:



Conv 5_3:



Conv 5_4:



The heatmaps from the layers in the first block appear to be outlining all the lines that make up the faces in both images. The heatmaps from the second block of layers appear to be focusing on the faces' eyes, mouths, and noses. We expect these regions to contain information regarding the emotion the person is currently experiencing. The heatmaps from the third block of layers become progressively more vague and move away from these key regions. This trend continues

into the heatmaps from the fourth block of layers. All the heatmaps in the fifth block of layers appear to be the same regardless of the image. From this, it appears that the model is losing its ability to generalize at some point during the third and fourth block of layers. Thus, the grad-cams suggest that the model with two blocks of layers might work better with identifying emotions than the original model with 5 blocks.

8.2 DropOut

In the following experiments, we used the same four blocks of layers which we left unaltered throughout the experiments except for changing the dropout percentage at the end of every block to either 30, 50, or 65%. Table 9 displays the test loss and accuracy for undistorted images for various dropout percentages. Tables 10 through 14 display test accuracies and test losses for images with various distortions on models containing various dropout percentages.

Table 9: Test Accuracies and Losses for not distorted images for various dropout percentages

Percentage of Dropout	**Test Accuracy**	**Test Loss**
30%	0.552	3.01
50%	0.627	2.42
65%	0.622	1.18

The highest test accuracy for not distorted images was for the 50% dropout model by a 0.005 margin over the 65% dropout model. The lowest test loss was for 65% dropout model, which was less than half the test loss for the other two models.

Table 10: Test Accuracies and Losses for gaussian blurred images for various dropout percentages

Percentage of Dropout	Test Accuracy	Test Loss
30%	0.272	5.66
50%	0.273	4.85
65%	0.244	2.31

The results from the gaussian blurred images were similar to the not distorted images in that the 50% dropout model had the highest accuracy while the 65% dropout model had the lowest loss.

Table 11: Test Accuracies and Losses for images containing gaussian noise for various dropout percentages

Percentage of Dropout	Test Accuracy	Test Loss
30%	0.416	4.60
50%	0.428	3.75
65%	0.428	1.67

The results for the images containing gaussian noise had a tie for the highest test accuracy between the model with 50% dropout and the model with 65% dropout. The lowest test loss was for the model with 65% dropout, which was less than half the test loss for the other two models.

Table 12: Test Accuracies and Losses for salt and pepper blurred images for various dropout percentages

Percentage of Dropout	Test Accuracy	Test Loss
30%	0.162	9.12
50%	0.184	5.46
65%	0.192	2.83

The highest test accuracy and lowest test loss for salt and pepper blurred images were both associated with the model with 65% dropout.

Table 13: Test Accuracies and Losses for speckle blurred images for various dropout percentages

Percentage of Dropout	Test Accuracy	Test Loss
30%	0.149	7.80
50%	0.191	6.57
65%	0.173	2.84

The highest test accuracy for speckle blurred images was for the 50% dropout model. The lowest test loss was for the 65% dropout model, which was less than half the test loss for the other two models.

Table 14: Test Accuracies and Losses for images containing occlusions for various dropout percentages

Percentage of Dropout	Test Accuracy	Test Loss
30%	0.326	4.89
50%	0.348	4.33
65%	0.387	2.14

Our results for occlusion found that our model with a 65% dropout rate was 4.2% more accurate on the occlusion test set than the model with 50% dropout. The model with 65% dropout still had the lowest test loss, and its test loss is more than half as small as both the 30% and 50% dropout models.

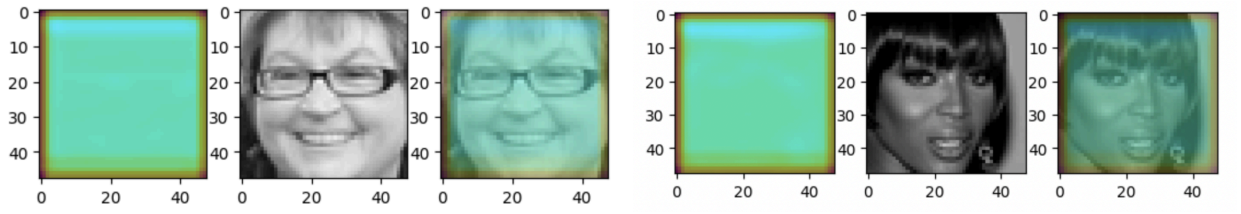
While the model with 50% dropout had a slightly higher test accuracy for some of the image sets, the model with 65% dropout had the lowest test loss by a significant margin for all image sets. The low test loss signals that predictions are getting close to the target values. Loss values improving while test accuracy remaining comparable signals that the model is becoming more

robust [17]. Thus, our hypothesis that when we increase the dropout rate, the test loss will decrease appears to be holding true.

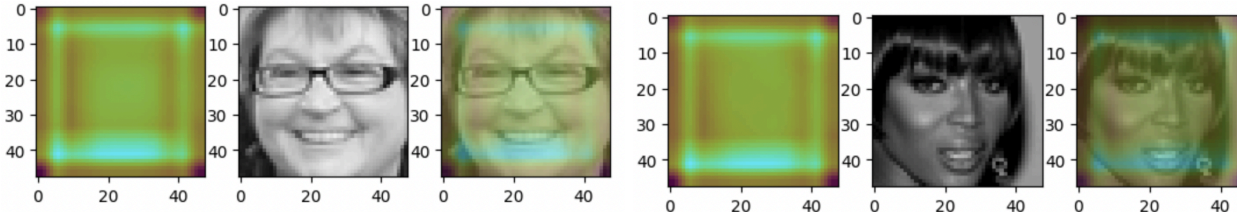
The following are the grad-cams from the last convolutional layer for the second, third, and fourth block of layers for each dropout model. Since these models were trained using batch normalization, the grad-cams only give us a general idea of the area the models are examining.

30% dropout:

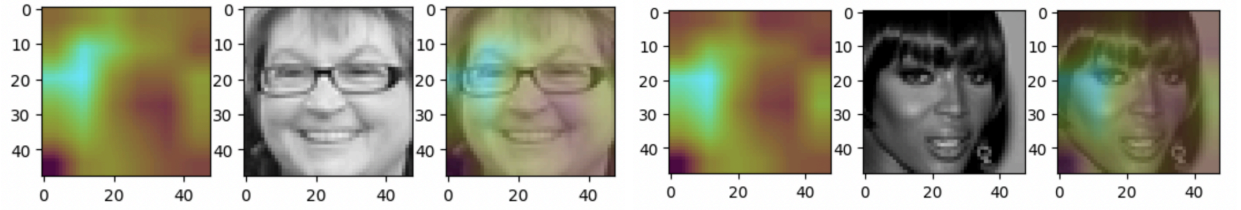
Conv2_3:



Conv3_4:

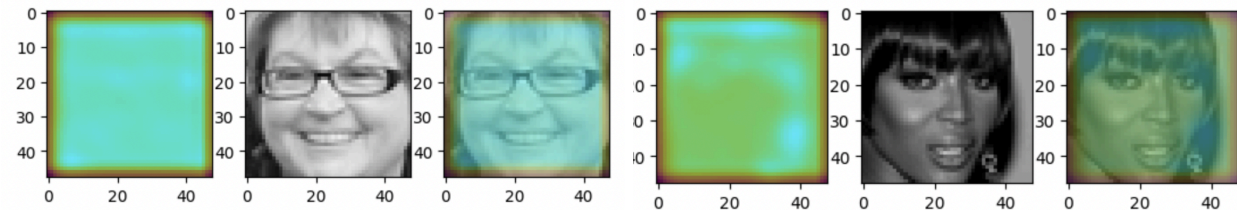


Conv4_4:

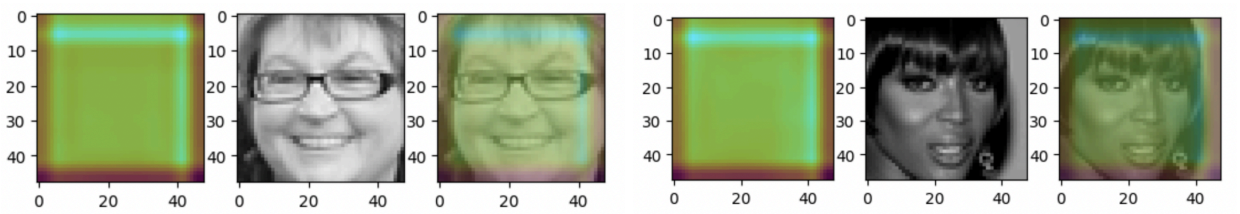


50% dropout:

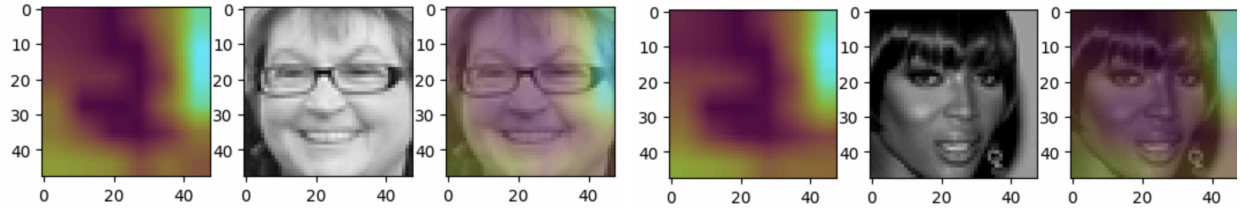
Conv2_3:



Conv3_4:

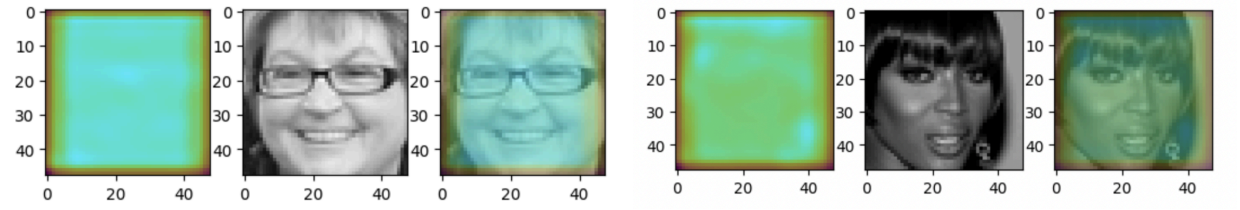


Conv4_4:

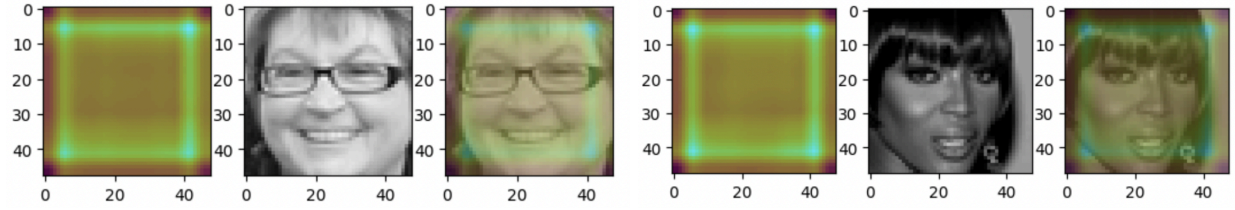


65% dropout:

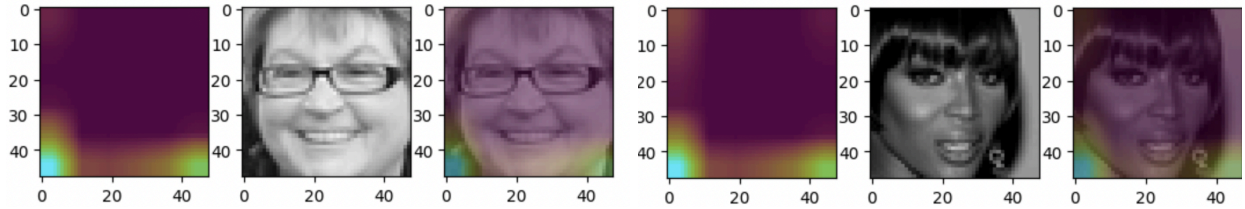
Conv2_3:



Conv3_4:



Conv4_4:



The grad-cams appear to be focusing on the majority of the picture in Conv2_3 and Conv3_4 regardless of the dropout rate. Conv2_3 is consistently looking at almost the entire picture while Conv3_4 puts a little more emphasis on the border in which the faces fall within. Conv4_4 appears to be focusing on areas that are not often within people's faces. It is not clear which dropout model is doing the best at picking out pictures from these grad-cams. Thus, we will rely more heavily on our quantitative analysis to draw conclusions about dropout rates' impact on the CNN's ability to classify emotions.

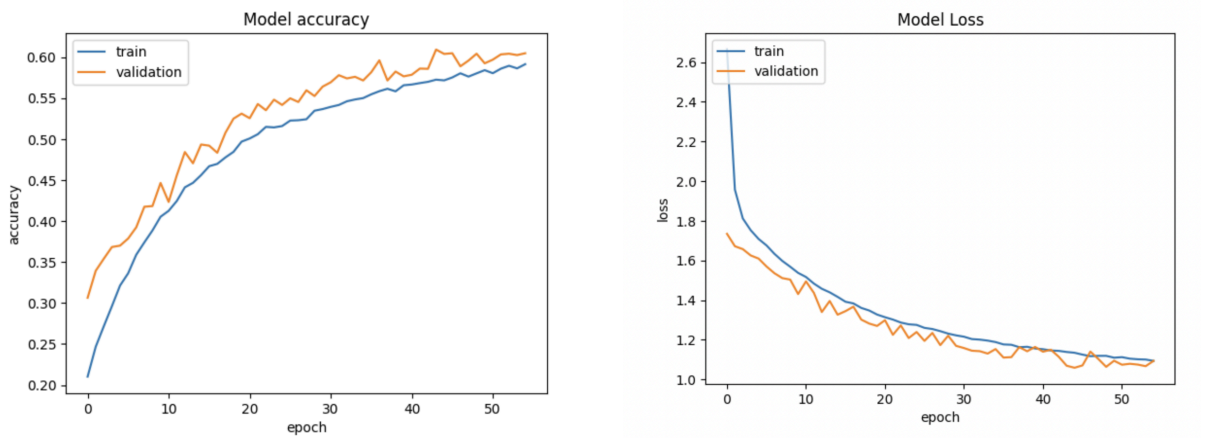
9. Final Model

From our blocks of layers' quantitative evaluations, we determined that the model with 2 blocks of layers consistently almost always had the lowest test loss. In addition, the grad-cams appeared to be picking out features such as eyes and mouths at the end of the second block and appeared to lose track of these features in the layers contained in block three. Thus, we decided to make our final model have two blocks of layers. From our dropout quantitative analysis, we determined that the model with 65% dropout consistently had the lowest test losses. Since our

qualitative analysis was non-conclusive, we decided to rely on our quantitative analysis and use 65% dropout for our final model. Similar to the baseline model, we used image augmentation when training our final model as it should make our model more robust to overfitting. Our experiments made us realize that the validation loss tended to go up after approximately 55 epochs for our final model, indicating that our model was overfitting after that. Therefore, we decided to stop at 55 epochs for training. The graphs showing the final model's training and validation accuracies and losses for different epochs are shown in Figure 10 on the following page.

Overall, we expected our updates to the model to cause our final model to have lower test losses on distorted and undistorted images than our baseline model. Since our test accuracies had less distinct trends in our experiments, we were unsure how these updates will affect the test accuracies.

Figure 10 : Training and validation losses and accuracy for final model



As can be seen above, our final model seems to reduce overfitting from the baseline model to a great extent. The training and validation losses are both quite low and about the same level.

In addition to evaluating our final model against the baseline, we decided to see how it performed compared to our EfficientNet transfer learning model. Transfer learning that uses EfficientNet with pre-trained weights from Imagenet should lead to a high accuracy and low loss. There is a plethora of research on EfficientNet's ability to reach "state-of-the-art accuracy on both imagenet and common image classification transfer learning tasks"[18]. Our experiments made us realize that the validation loss tended to go up after approximately 20 epochs for EfficientNet, indicating that the model was overfitting after that. Therefore, when adapting EfficientNet, we decided to stop at 20 epochs for training. Table 15 shows the test accuracies and test losses for not distorted images on our baseline, final, and EfficientNet models. Tables 16 through 20 show the test accuracies and test losses for images with various distortions on the same models.

Table 15: Test Accuracies and Losses for not distorted images using various models

Type of Model	**Test Accuracy**	**Test Loss**
Baseline	0.671	1.34
Final Model	0.596	1.10
Efficient Net	0.590	1.63

Our final model has the lowest test loss of all three models. Our final model has a slightly higher test accuracy than our EfficientNet model. However, it is 7.5% less accurate than our baseline model. The confusion matrices for our baseline and final models are shown on the following page.

Baseline Model:

```
[[ 57  10  72 135  81  59  97]
 [  8   0   7  11   5   6  12]
 [ 63  12  79 129  67  59 107]
 [104   8 133 227 162  98 171]
 [ 88   3  67 168  94  67 110]
 [ 55   5  53  98  64  43  63]
 [ 70   8  77 150 124  66 137]]
```

Final Model:

```
[[ 66   9  39 151  48  64 134]
 [  5   1   3  17   5   7  11]
 [ 66   3  35 156  70  61 125]
 [123   4  84 262 107 110 213]
 [ 84   0  49 177  62  85 140]
 [ 61   2  32 111  54  51  70]
 [ 89   4  43 171  91  77 157]]
```

By examining the confusion matrices' diagonals, we can see how many correct identifications the models made for each class. The green highlights show the highest number of correctly identified images for each class between the two models. Our final model had the highest number of correctly identified images for 5 of the classes while the baseline had the highest for only two. Thus, it appears that our baseline model's higher accuracy is mainly due to it correctly identifying fear and sad emotions, while our final model has more correct predictions for all the other categories.

Table 16: Test Accuracies and Losses for gaussian blurred images using various models

Type of Model	Test Accuracy	Test Loss
Baseline	0.289	4.02
Final Model	0.319	1.86
Efficient Net	0.298	2.81

Our final model has the highest test accuracy and the lowest test loss when run on our gaussian blurred image test set.

Table 17: Test Accuracies and Losses for images containing gaussian noise using various models

Type of Model	Test Accuracy	Test Loss
Baseline	0.435	3.01
Final Model	0.405	2.11
Efficient Net	0.372	2.51

Our final model has the lowest test loss of all three models when examining our test set containing gaussian noise. Our final model has a 3.3% higher test accuracy than our EfficientNet model. However, it is 3% less accurate than our baseline model.

Table 18: Test Accuracies and Losses for salt and pepper blurred images using various models

Type of Model	Test Accuracy	Test Loss
Baseline	0.181	7.38
Final Model	0.189	2.19
Efficient Net	0.185	3.48

Our final model has the highest test accuracy and the lowest test loss when run on our salt and pepper blurred image test set.

Table 19: Test Accuracies and Losses for speckle blurred images using various models

Type of Model	Test Accuracy	Test Loss
Baseline	0.192	9.94
Final Model	0.187	2.72
Efficient Net	0.152	3.44

Our final model has the lowest test loss of all three models when testing on speckle blurred images. Our final model has a 3.5% higher test accuracy than our EfficientNet model. However, it is 0.5% less accurate than our baseline model.

Table 20: Test Accuracies and Losses for images containing occlusions using various models

Type of Model	Test Accuracy	Test Loss
Baseline	0.373	3.00
Final Model	0.443	1.78
Efficient Net	0.372	2.45

Our final model has the highest test accuracy and the lowest test loss when run on an occlusion image test set. Notably, the test accuracy is 7% higher than the two other models and the test loss is almost half of the EfficientNet test loss and more than half of the baseline test loss.

10. Conclusion

Overall, our final model had the lowest test loss on all six testing sets. As we had hypothesized, our choices of a 2 layer model and a dropout rate of 65% led the model's test loss to be reduced for all testing sets and thus, we had a more robust model. In addition, it always had a higher test accuracy than EfficientNet. It also had a higher test accuracy than the baseline model for gaussian blurred, salt and pepper blurred, and occluded images. Thus, we are confident that we have reduced the overfitting we observed in the baseline model and feel that our model could possibly generalize well to images not in the FER-2013 dataset. Our training and validation plots also showcase how the problem of overfitting that we were facing with the baseline model has now been addressed.

11. Honor Code

This paper represents our own work in accordance with University regulations. /s/ Sydney Berry

& Advika Srivastva

12. References

[1] Thetechwriters. “Facial Emotion Detection Using CNN.” Analytics Vidhya, 5 Sept. 2022, www.analyticsvidhya.com/blog/2021/11/facial-emotion-detection-using-cnn/.
[2] Mehendale, Ninad. “Facial Emotion Recognition Using Convolutional Neural Networks (FERC) - SN Applied Sciences.” *SpringerLink*, 18 Feb. 2020, link.springer.com/article/10.1007/s42452-020-2234-1.
[3] Debnath, Tanoy, et al. “Four-Layer ConvNet to Facial Emotion Recognition with Minimal Epochs and the Significance of Data Diversity.” Nature News, 28 Apr. 2022, www.nature.com/articles/s41598-022-11173-0#Fig8.
[4] Hossain, Md Tahmid, et al. “Distortion Robust Image Classification Using Deep Convolutional Neural Network with Discrete Cosine Transform.” IEEE Xplore, 26 Aug. 2019, ieeexplore.ieee.org/document/8803787/.
[5] Kuruvayil, Soumya, and Suja Palaniswamy. “Emotion Recognition from Facial Images with Simultaneous Occlusion, Pose and Illumination Variations Using Meta-Learning.” *Journal of King Saud University - Computer and Information Sciences*, 22 June 2021, www.sciencedirect.com/science/article/pii/S1319157821001452.
[6] Yeom, Sonja. “FER2013 Training Data Distribution. | Download Scientific Diagram.” FER2013 Training Data Distribution, Dec. 2018, www.researchgate.net/figure/FER2013-training-data-distribution_fig1_329948265.
[7] Sambare, Manas. “Fer-2013.” Kaggle, 19 July 2020, www.kaggle.com/datasets/msambare/fer2013.
[8] Selvaraju, Ramprasaath R., et al. “Grad-Cam: Visual Explanations from Deep Networks via Gradient-Based Localization.” arXiv.Org, 3 Dec. 2019, arxiv.org/abs/1610.02391.
[9] Team, Keras. “Keras Documentation: Grad-Cam Class Activation Visualization.” *Keras*, 26 Apr. 2020, keras.io/examples/vision/grad_cam/.
[10] V. “Python Code to Add Random Gaussian Noise on Images.” GitHub Gist, 2019, gist.github.com/Prasad9/28f6a2df8e8d463c6ddd040f4f6a028a.
[11] apurva__007. “Add a ‘Salt and Pepper’ Noise to an Image with Python.” GeeksforGeeks, 3 Jan. 2023, www.geeksforgeeks.org/add-a-salt-and-pepper-noise-to-an-image-with-python/.
[12] Kang, and Atul. “Add Different Noise to an Image.” TheAILearner, 7 May 2019, theailearner.com/2019/05/07/add-different-noise-to-an-image/.
[13]Fu, Yixing. “Keras Documentation: Image Classification via Fine-Tuning with EfficientNet.” *Keras*, 30 June 2020, keras.io/examples/vision/image_classification_efficientnet_fine_tuning/.
[14] “How Can I Use a Pre-Trained Neural Network with Grayscale Images?” *Stack Overflow*, stackoverflow.com/questions/51995977/how-can-i-use-a-pre-trained-neural-network-with-grayscale-images. Accessed 9 May 2023.

[15] Haque, Rezuana HaqueRezuana. “Grad Cam Outputs for All the Images Are the Same.” Stack Overflow, 1 Oct. 1969, stackoverflow.com/questions/75272745/grad-cam-outputs-for-all-the-images-are-the-same.
[16] Lin, David Chuan-En. “8 Simple Techniques to Prevent Overfitting.” Medium, 10 Apr. 2023, towardsdatascience.com/8-simple-techniques-to-prevent-overfitting-4d443da2ef7d#:~:text=As%20mentioned%20in%20L1%20or,in%20the%20fully%2Dconnected%20layers.
[17] ANTOREEPJANA. “Accuracy vs Loss Conflict: Data Science and Machine Learning.” Kaggle, 2021, www.kaggle.com/general/220823.
[18] Gurion, David Ben. “Image Classification Transfer Learning and Fine Tuning Using Tensorflow.” Medium, 17 Nov. 2021, towardsdatascience.com/image-classification-transfer-learning-and-fine-tuning-using-tensorflow-a791baf9dbf3.
[19] Utami, Pipit, et al. “The EfficientNet Performance for Facial Expressions Recognition.” *Ieee*, 7 Mar. 2023, ieeexplore.ieee.org/Xplore/guesthome.jsp.