Detect human emotion through the images

**Adelene Ng, Wong Cheng Kwan, Perumal Srinivas**

# Motivation

Emotion is one of the very few words in the English language that do not have a [concrete definition](https://en.wikipedia.org/wiki/Emotion) and it is understandable. It is abstract. Yet almost every decision we have ever made in our lives is driven by emotion.

With working from home becoming more common these days, digital meetings or digital learnings are getting into its norm these days. Analysing emotion through communication platforms like Zoom, Teams, Ring Central etc., can give insights into the general sentiment of what people say or mean during virtual meetings.

As a part of this project, we aim to predict the emotional category from a recorded video clip or stills which can fall into one of the 7 categories – Angry, Disgust, Fear, Happy, Sad, Surprise and Neutral. We do this by fine-tuning 3 different convolutional neural networks for the tasks of emotion prediction.

The objective of this project is to develop a deep learning model to detect the human facial emotion through video clips or though still images.

# Datasets

For this project, we will use the Facial Expression Recognition - 2013 dataset. This can be downloaded from [Kaggle](https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data).

The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is centred and occupies about the same amount of space in each image.

The task is to categorize each face based on the emotion shown in the facial expression into one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). The training set consists of 28,709 examples and the public test set consists of 3,589 examples.

# Methods

We plan to split up the use the following deep learning architectures for this project. Each of us will work on one basic architecture and fine tune it further to achieve the best accuracy.

Once done, we will pick the best accuracy out of three, as the implementation candidate.

* VGGNet16 (Wong Cheng Kwan)
* MobileNetV1 (Perumal Srinivas)
* DenseNet121 (Adelene Ng)

# Preliminary Experiments

## Preliminary Experiments (Wong Cheng Kwan)

VGGNet is a classical convolutional neural network architecture used in large-scale image processing and pattern recognition. The VGG network introduced the concept of grouping multiple convolution layers with smaller kernel sizes instead of having one Conv layer with a large kernel size. This caused the number of features at the output to reduce and including multiple Relu layers instead of one, thus increasing learning instances.

### Experimental Run - Observations

Observation here is that this model has many weight parameters.

Also, the size of the model is considered huge. This might impact inference timing if response time is a concern.

Constant learning and relearning are problems with VGG which is why the loss seems to be so unpredictable (vanishing or exploding gradients).

### Experimental Run – VGG16 vs VGG19

VGG19 will have another three Convolutional layers with learnable weights, which also meant with deeper depth than VGG16.

Despite of this deep layers, where VGG19 is having 15.8% more learnable weights for computation, it only yields very slight accuracy improvement of 0.71% over VGG16. That is why VGG16 is chosen as the basic model.

### Experimental Run – with different Dropout Rate

A table created as below for recording the validation accuracy, with adjustment made to the parameters one at the time.

Although VGG16-1 is having the highest accuracy, the loss and accuracy graphs are showing there is a degrading of performance as number of epochs increases.

VGG16-1a is showing a better graph performance, where both loss and accuracy are showing traits of a learning model. Thus VGG16-1a will be used as the baseline for my subsequence improvement experiments.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Model | Batch Size | Dropout | Epochs with Early Stopping | Learning Rate | Optimizers | Validation accuracy |
| 1 | VGG16(base) | 64 | - | 25 / 200 | 1e-4 | Adam | 58.65% |
| 2 | VGG19 | 64 | - | 26 / 200 | 1e-4 | Adam | 59.07% |
| 3 | VGG16-1 | 64 | 0.5 | 37 / 200 | 1e-4 | Adam | 62.11% |
| 4 | VGG16-1a | 64 | 0.8 | 116 / 200 | 1e-4 | Adam | 60.69% |
| 5 | VGG16-1b | 64 | 0.2 | 27 / 200 | 1e-4 | Adam | 61.86% |

*Table 1 – VGGNet16 Baseline Model Parameters (Wong Cheng Kwan)*

### Next Steps

I will experiment with other parameters like Batch Size, Learning Rate, Optimizers to get the best possible accuracy, and compared them with other members best models.

## Preliminary Experiments (Perumal Srinivas)

Perumal Srinivas is based on MobileNetV1, has built the basic model as a baseline, It consists of 42 layers, with about 2 million parameters.

The MobileNet architecture was chosen as it offered the following advantages and are as follows:

1. MobileNet (Efficient Convolutional Neural Networks for Mobile Vision Applications) is an architecture that focuses on making the deep learning networks very small and having low latency.
2. The MobileNet models can be easily deployed on mobile and embedded edge devices.
3. MobileNets are based on a streamlined architecture that uses depth-wise separable convolutions to build light weight deep neural networks.
4. The MobileNet uses a Depthwise separable convolution instead of the usual convolution to reduce the computation time and parameters.

It works by applying convolution to each channel of the image instead of as a block n times, and then using 1x1 convolution to get n filters.

The first step was to create a baseline model. In run 1, the following parameters were used:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Run | Description | Batch size | # Epochs | Learning rate | Optimizer | Parameters | ES – Patience | Loss | Accuracy |
| 1 | Baseline | 256 | 108/200 | 0.0001 | Adam | 2,089,159 | 20 | 1.5 | 42.49% |

Basic Mobilenet Architecture

|  |  |
| --- | --- |
| Parameters | Shape |
| *Conv/s2*  *Conv dw/s1*  *Conv /s1*  *Conv dw/s2*  *Conv /s1*  *conv dw /s1*  *Conv / s1*  *Conv dw / s2*  *Conv / s1*  *Conv dw / s1*  *Conv / s1*  *Conv dw /s2*  *Conv /s1*  *5 \* Conv dw/s1, Conv /s1*  *Conv dw / s2*  *Conv / s1*  *Avg Pool /s1*  *FC /s1*  *Softmax /S1* | *32*  *32*  *64*  *64*  *128*  *128*  *256*  *256*  *256*  *256*  *512*  *512*  *512*  *512*  *512*  *1024*  *1024*  *1024*  *Classifier* |

### Experimental Run – Varying Optimizer, Batch size, Layers, Data Augmentation values

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Run | Batch size | # Epochs | Learning rate | Parameteters | Optimizer | ES – Patience | Loss | Accuracy |
| 2 | 256 | 192/200 | 0.0001 | 82,823 | NAdam | 20 | 1.1 | 61.1% |
| 3 | 128 | 196/200 | 0.0001 | 57,863 | Adam | 20 | 1.1 | 57.45% |
| 4 | 128 | 119/200 | 0.001 | 57,863 | NAdam | 20 | 1.1 | 61.0% |
| 5 | 256 | 130/200 | 0.001 | 57,863 | NAdam | 20 | 1.2 | 61.38% |
| 6 | 64 | 133/200 | 0.001 | 57,863 | NAdam | 20 | 1.1 | 61.41% |

*Table 2 – Results from varying the Optimizer, Batch Size, Layers, Data Augmentation*

The objective of “ReduceLROnPlateau” is to reduce the learning rate when a metric has stopped improving. If no improvement is seen for a “patience” number of epochs, the learning rate is reduced by factor value (new\_lr = lr \* factor), where factor = 0.2.

The results demonstrate that reducing the size of the layers for the Depthwise seperable convolution and the resized Mobilenet Architecture has improved the accuracy.

## Preliminary Experiments (Adelene Ng)

The DenseNet architecture was chosen because it offered several advantages and these are:

1. reduces the vanishing-gradient problem
2. strengthens feature propagation
3. encourages feature reuse
4. substantially reduces the number of parameters

Feature propagation and reuse are reinforced because for each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps of the current layer are used as inputs into all subsequent layers.

The downside to DenseNet is that it is memory hungry because of the need for multiple concatenation operations because data is copied multiple times

The first step was to create a baseline model. In run 1, the following parameters were used:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Run | Description | Batch size | # Epochs | Learning rate | Optimizer | Dropout | ES – Patience | Loss | Accuracy |
| 1 | Baseline | 128 | 108/200 | 0.0001 | Adam | 0.4 | 20 | 1.5 | 59.1% |

*Table 3 – Baseline hyperparameters used in DenseNet121 training*

To prevent overfitting, the early stopping (ES) callback (column 8 Table 3) with a patience set to 20 was used.

The default DenseNet121 parameter settings used in the baseline model is shown in the table below:

|  |  |
| --- | --- |
| Parameters | Value |
| num\_blocks | 3 |
| num\_layers\_per\_block | 4 |
| growth\_rate | 16 |
| dropout\_rate | 0.4 |
| compress\_factor | 0.5 |
| eps | 1.1e-5 |
| num\_filters | 16 |

*Table 4 – DenseNet121 Baseline Model Default Parameters*

The DenseNet121 baseline model had 92 layers with a total of 2,377,958 parameters.

### Experimental Run – Apply “ReduceLROnPlateau”

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Run | Batch size | # Epochs | Learning rate | Optimizer | Dropout | ES – Patience | Loss | Accuracy |
| 2 | 128 | 91/200 | 0.0001 | Adam | 0.4 | 20 | 1.5 | 57.4% |

*Table 5 – Results from applying ReduceLROnPlateau callback*

The objective of “ReduceLROnPlateau” is to reduce the learning rate when a metric has stopped improving. If no improvement is seen for a “patience” number of epochs, the learning rate is reduced by factor value (new\_lr = lr \* factor), where factor = 0.2. The resulting accuracy is less than the baseline model. This may be due to the training stopping earlier compared to the baseline model (91 vs. 108).

### Experimental Run – Varying Dropout Rates

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Run | Batch size | # Epochs | Learning rate | Optimizer | Dropout | ES – Patience | Loss | Accuracy |
| 3 | 128 | 59/200 | 0.0001 | Adam | 0.8 | 20 | 2.3 | 25.5% |
| 4 | 128 | 42/200 | 0.0001 | Adam | 0.2 | 20 | 1.2 | 61.6% |
| 5 | 128 | 40/200 | 0.0001 | Adam | 0.1 | 20 | 1.3 | 60.1% |

*Table 6 – Results from varying the dropout rates*

The results demonstrate that a high dropout rate resulted in low accuracy. Setting a high dropout rate to 0.8 could mean that many of the useful features and correlations previously learned was discarded and the resulting loss in information resulted in a drastic drop in accuracy. It is observed that reducing the dropout increases the model accuracy, which is most likely due to less information from previous layers being discarded.

### Experimental Run – Applying Grid Search

I applied Grid Search to the following experimental runs to determine:

* Best optimizer to use
* Best Learning Rate
* Best Batch Size

Grid search made it easier to keep track of hyperparameters tested. However, having said that, since I used the default value of “None” for CV (cross validation), this defaults to using the 5-fold cross validation.

Although grid search could have been applied using a search space as follows:

params = {

‘optimizers’: [‘SGD’, ‘Adam’],

‘learning\_rate’: [0.001, 0.0001, 0.00001],

‘batch\_size’: [32, 64, 128]

}

which results in 2x3x3=18 different parameter combinations. Since a default cross-validation of 5-fold was used, this would mean the grid search code would train the model 18 x 5 = 90 times. I chose not to do it this way because of the long computational times.

| Run | Description | Batch Size | Learning Rate | Optimizer | Drop Out | ES – Patience | Best Parameters | Best Score |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | Vary Optimizers | 128 | 0.0001 | SGD, Adam | 0.4 | 20 | Adam | 58.5% |
| 7 | Vary Learning Rates | 128 | 0.001, 0.0001, 0.00001 | Adam | 0.4 | 20 | 0.001 | 60.0% |
| 8 | Vary Batch Sizes | 32, 64, 128 | 0.0001 | Adam | 0.4 | 20 | 64 | 56.5% |

*Table 7 – Applying “GridSearchCV” to find the best parameters*

### Next Steps

After running the different tuning experiments for DenseNet121, I will use the best parameters from each run to create the model and check if there is any improvement in the accuracy. If the improvement is minimal, I will then try data augmentation to see if I can squeeze better accuracy out from the model.

# Project Management

Allocation of Tasks

|  |  |  |
| --- | --- | --- |
| No | Tasks | Team Members |
| 1 | Understanding of Business Case | All |
| 2 | Data Understanding | All |
| 3 | Data Preparation | All |
| 4 | * Modeling VGGNet16 (Wong Cheng Kwan) * MobileNetV2 (Perumal Srinivas) * DenseNet121 (Adelene Ng) | Individual |
| 5 | Evaluation | Individual |
| 6 | Project Presentation | All |
|  |  |  |

Project Development Steps

|  |  |  |  |
| --- | --- | --- | --- |
| Steps | Development | Plan End Date | Actual End Date |
| 1 | Project Team Selection | 13/1/2022 | 13/1/2022 |
| 2 | Selection of Project Title | 13/1/2022 | 13/1/2022 |
| 3 | Submission of Project Proposal | 20/1/2022 | 20/1/2022 |
| 4 | Confirmation on Data Sets to be used | 25/1/2022 | 25/1/2022 |
| 5 | Explore and confirm which Models to use | 25/1/2022 | 25/1/2022 |
| 6 | Completion of base model | 10/2/2022 | 10/2/2022 |
| 7 | Submission of Project Milestone | 17/2/2022 | 21/2/2022 |
| 8 | Completion of Model Tuning | 21/2/2022 |  |
| 9 | Select the Best model for deployment | 28/2/2022 |  |
| 10 | Final Presentation | 2/3/2022 |  |
| 11 | Final Report submission | 2/3/2022 |  |