Summary for FER2013 facial emotion recognition task:

Without data augmentation or synthetic images

> Started with a simple *1 layer CNN* architecture for the multi-class (7) classification. However, the accuracy of correct classification achieved was not high (limited to only **36%**). Could have pursued some parameter changes in the same 1 layer CNN architecture, however, the accuracy won't have increased much. So, the next step decided was to build a different architecture and include more layers in the new architecture.

> For the second CNN architecture, initially started with a *four* convolutional layers, *three* maxpooling layers (each layer between two conv2d layers), and *two* dense layers (at the end with dropout). As expected, the accuracy showed an increase to around **56%**. Changing the network parameters (such as batch size, number of epochs) did not affect the accuracy much. Adding *zero padding* to the convolutional layers increased the accuracy marginally (0.5%). Changing the *pool size* of the maxpooling layers did not affect the accuracy much too.

> For achieving a further higher accuracy, thought of making some changes to the architecture by following a research paper based on multi class classification of chest radio-graph images (https://arxiv.org/pdf/1701.08816.pdf). In this paper, the authors have mentioned to add regularizers to the dense layer and try and experiment with the regularizer rate. Following the suggestion, added *kernel regularizers* with *l2 regularization* firstly in the dense layers only and experimented with three different rates (0.1, 0.01, 0.001) -> **0.001** gave the best result with an increase in accuracy of almost **1%**.

> Following the CNN architecture suggestions in the paper, realized that the architecture contains an extra maxpooling layer. So, removed one of the three maxpooling layers and the new architecture then contained two maxpooling layers, each followed by two convolutional layers. The change in accuracy was a marginal increase of around **0.5%**.

> Following the CNN architecture suggestions listed in the paper, as the network goes deep, more *detailed patterns in the image* are extracted by the layers in the network. So, thought of replacing the second maxpooling layer with an average pooling layer so that the network can learn the detailed patterns more efficiently. Implemented the same, and it showed an increase in the accuracy by around **2-2.5%**, which was quite significant.

> Concerning the maxpooling and the average pooling layers, introduced *strides* in the layer. However, this introduction of strides of size (2, 2) made the network execution slow as well as adversely affected the metric of accuracy. The accuracy *decreased* by **1%**.

> As suggested in the paper, tried *Gaussian dropout* in the last two dense layers. However, the introduction of Gaussian dropout does not affect the accuracy significantly. So, chose to remain with the normal dropout with a rate of 0.25.

> Finally, also tried training the network with resizing the images (64 x 64), however, the accuracy is not affected.

The major architectural changes and their impact on the accuracy are summarized in the table below.

CNN architectural change	Accuracy / Impact on accuracy (%)
Simple 1 layer CNN	36%

4 conv2D, 3 maxpooling, 2 dense	56%
4 conv2D, 2 maxpooling, 2 dense	56.7% / increased by ~0.5%
4 conv2D, 1 maxpooling, 1 average pooling, 2 dense	58.9% / increased by ~2-2.5%
Adding kernel regularizers with l2 regularization (0.001) in the dense layer	**59.7%** / increased by ~1.0%
Included strides in the maxpooling layer	58.6% / decreased by ~1.0%
Replace normal dropout with Gaussian dropout	**59.7%** / almost remains the same
Resizing the image (64 x 64)	**59.7%** / almost remains the same
Changing the size and number of filters in Conv2D	**59.7%** / almost remains the same

> Hence the *final CNN* architecture comprises as below:
Conv2D -> Conv2D -> Maxpooling2D -> Conv2D -> Conv2D -> Averagepooling2D -> Flatten -> Dense -> Dropout -> Dense -> Dropout -> Dense (7)

For more details on the parameters of the layers (number of filters, neurons, pool size, etc.), please refer to the CNN model configuration and summary below:

CNN model configuration

Layer	Hyperparameters
Convolutional	*32* filters, *3 x 3*, L2 regularization (*0.001*), activation: relu, zeropadding (*2, 2*)
Convolutional	*64* filters, *3 x 3*, activation: relu, zeropadding (*1, 1*)
Pooling	Maxpool (*5 x 5)*
Convolutional	*128* filters, *3 x 3*, activation: relu, zeropadding (*1, 1*)
Convolutional	*128* filters, *3 x 3*, activation: relu, zeropadding (*1, 1*)
Pooling	Average pool (3 *x 3)*
Dense	1024
Dropout	0.25
Dense	1024
Dropout	0.25
Softmax	Fully connected

CNN model summary

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 46, 46, 32)        320
_________________________________________________________________
batch_normalization_1 (Batch (None, 46, 46, 32)        128
```

```
_________________________________________________________________
p_re_lu_1 (PReLU)            (None, 46, 46, 32)        67712
_________________________________________________________________
zero_padding2d_1 (ZeroPaddin (None, 50, 50, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 48, 48, 64)        18496
_________________________________________________________________
p_re_lu_2 (PReLU)            (None, 48, 48, 64)        147456
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 9, 9, 64)          0
_________________________________________________________________
zero_padding2d_2 (ZeroPaddin (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 11, 11, 128)       73856
_________________________________________________________________
p_re_lu_3 (PReLU)            (None, 11, 11, 128)       15488
_________________________________________________________________
zero_padding2d_3 (ZeroPaddin (None, 15, 15, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 128)       147584
_________________________________________________________________
p_re_lu_4 (PReLU)            (None, 13, 13, 128)       21632
_________________________________________________________________
zero_padding2d_4 (ZeroPaddin (None, 17, 17, 128)       0
_________________________________________________________________
average_pooling2d_1 (Average (None, 5, 5, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              3277824
_________________________________________________________________
p_re_lu_5 (PReLU)            (None, 1024)              1024
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              1049600
_________________________________________________________________
p_re_lu_6 (PReLU)            (None, 1024)              1024
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 7175
_________________________________________________________________
activation_1 (Activation)    (None, 7)                 0
=================================================================
Total params: 4,829,319
Trainable params: 4,829,255
Non-trainable params: 64
```

With data augmentation or synthetic images

As was suggested by enpuku, the CNN architecture can be trained with some augmented images rather than just the normal images. In order to train the above CNN architecture with the augmented images, firstly produced some synthetic images with rotation, flip, blur, etc. These images were generated with the *ImageDataGenerator* provided by *keras*. The above operations were applied

to each of the *training images* in the dataset of FER2013 and the CNN architecture was trained with *datagen.flow* provided by *keras*. Some of the synthetic images generated for the original image in the training data set are shown below. More images can be found in the folder Synthetic images of the project repository.

Original image in the dataset.

Synthesized image of the above original image.

Original image in the dataset.

Synthesized image of the above original image.

As described above, the CNN architecture is then trained with the synthesized images instead of the original training images in the dataset. The parameters used for generating the synthesized images are listed below:

Parameters	**Accuracy**
rotation_range=20, width_shift_range=0.1, height_shift_range=0.1, shear_range=0.1, zoom_range=0.1, horizontal_flip=True	**63.14%**
rotation_range=40, width_shift_range=0.2, height_shift_range=0.2, shear_range=0.2, zoom_range=0.2, horizontal_flip=True	60.62%
rotation_range=90, width_shift_range=0.4, height_shift_range=0.4,	48.55%

shear_range=0.4, zoom_range=0.4, horizontal_flip=True	

From the results in the above table, it can be concluded that if the augmentation effects are too high, it destroys the original image to a large extent and thus the actual image patterns to be learned by the network are also destroyed. Hence the augmentation effects should be selected appropriately. The best accuracy obtained with the augmentation is with the SET 1 parameters in the above table and the accuracy attained is **63.14%**, depicting a rise of greater than **3%** when compared to the accuracy when the CNN is trained only with the original training images contained in the FER2013 dataset.