Facial Expression Recognition

Final Evaluation Report

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

In

Computer Science and Engineering

Major Project

Supervised By:

Dr. Virendra Prasad Vishwakarma,

Associate Professor

USICT, GGSIPU

Submitted By:

Mayank Bhandari

B. Tech. - C.S.E. (14-18)

8th Semester

02916403214

CERTIFICATE

This is to certify that project report entitled Facial Expression Recognition being submitted by Mayank Bhandari in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science and Engineering to University School of Information and Communication Technology is a record of bona fide work carried out by him under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree or diploma.

Mentor:

Dr. Virendra Prasad Vishwakarma,

Associate Professor

USICT, GGSIPU

ACKNOWLEDGEMENT

I express my sincere gratitude to all those who have provided me with invaluable assistance during this project and have helped me during the course of this project. I am sincerely thankful to my mentor who provided me with his valuable guidance and timely suggestions, and whose feedbacks greatly shaped this project. I would also like to take this opportunity to thank University School of Information and Communication Technology (USICT), Guru Gobind Singh Indraprastha University for providing me the opportunity to work on this project. Also, I am thankful to my friends for their advice and help. Also I would like to express my gratitude to my project coordinators and faculties and lab staff of my institution

Mayank Bhandari

(02916403214)

CONTENTS

1. Introduction

Facial expression are important cues for non-verbal communication among human beings. This is only possible because humans are able to recognize emotions quite accurately and efficiently. An automatic facial emotion recognition system is an important component in human machine interaction. Apart from the commercial uses of an automatic facial emotion recognition system it might be useful to incorporate some cues from the biological system in the model and use the model to develop further insights into the cognitive processing of our brain. A facial expression is a gesture executed with the facial muscles, which convey the emotional state of the subject to observers. An expression sends a message about a person's internal feeling. In Hebrew, the word for "face"- פָּנִים, has the same letters as the word represents "within" or "inside"- פְּנִים. That similarity implies about the facial expression most important role- being a channel of nonverbal communication.

Facial expressions are a primary means of conveying nonverbal information among humans, though many animal species display facial expressions too. Although human developed a very wide range and powerful of verbal languages, facial expression role in interactions remains essential, and sometimes even critical.

Expressions and emotions go hand in hand, i.e. special combinations of face muscular actions reflect a particular emotion. For certain emotions, it is very hard, and maybe even impossible, to avoid it’s fitting facial expression.
For example, a person who is trying to ignore his boss's annoying offensive comment by keeping a neutral expression might nevertheless show a brief expression of anger. This phenomenon of a brief, involuntary facial expression shown on the face of humans according to emotions experienced is called 'micro-expression'.
Micro-expressions express the seven universal emotions: **happiness, sadness, anger, surprise, contempt, fear** and **disgust**. Micro-expression is lasting only 1/25-1/15 of a second. Nonetheless, capturing it can illuminate one's real feelings, whether he wants it or not.

2. Overview

2.1 Convolutional Neural Networks(CNN)[1]

2.1.1 What is a CNN?

Convolutional Neural Network (**CNN**, or **ConvNet**) is a class of deep, feed-forward artificial neural network that has successfully been applied to analysing visual imagery.

CNNs use a variation of multilayer perceptron designed to require minimal pre-processing. They are also known as **shift invariant** or **space invariant artificial neural networks** (**SIANN**), based on their shared-weights architecture and translation invariance characteristics.

Convolutional networks were inspired by biological processes in which the connectivity pattern between neurons is inspired by the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification. This means that the network learns the filters that in traditional algorithms were hand engineered. This independence from prior knowledge and human effort in feature design is a major advantage. They have applications in image and video recognition, natural language processing.

2.1.2 Architecture of CNN

Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: **width, height, depth**.(*Depth* here refers to the third dimension of an activation volume, not to the depth of a full Neural Network). The neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. There are four constituents of CNN architecture namely Convolution Layers, Pooling/Subsampling Layers, Non-Linear Layers/activation functions, Fully-Connected Layers

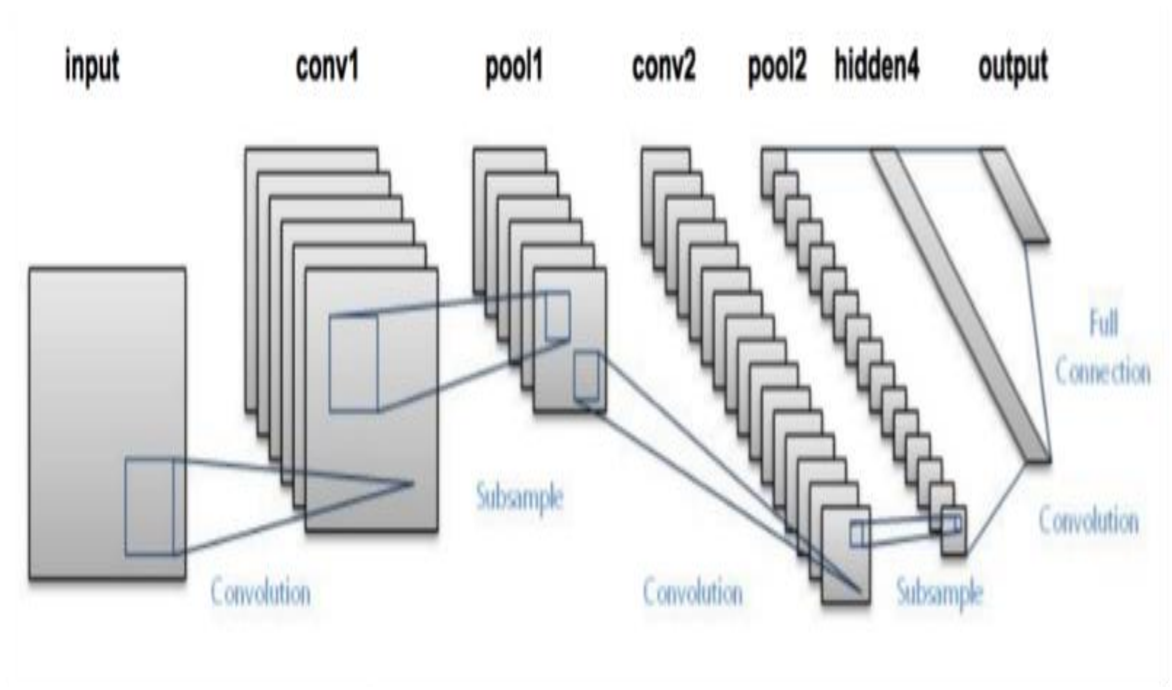


Fig 1. A typical architecture of Convolutional Neural Network

2.1.2.1 Convolutional Layers

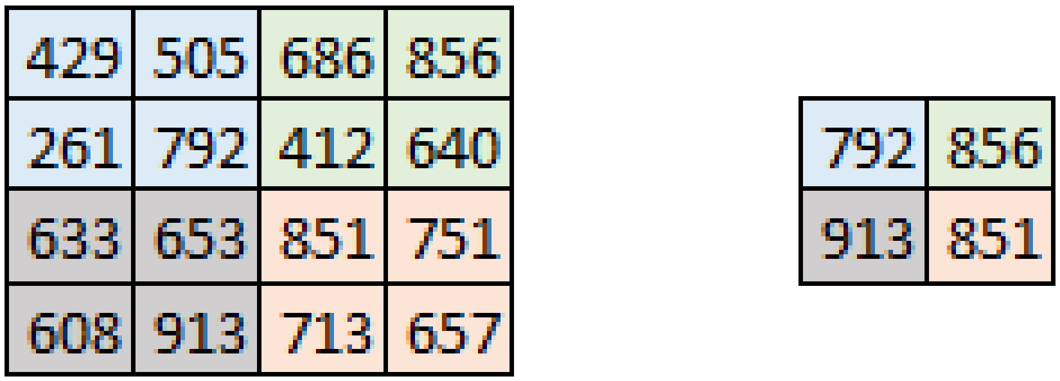
ConvNets derive their name from the convolution operator. The primary purpose of Convolution in case of a ConvNet is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.

In a CNN the input layer consisting of input image, is connected to a convolutional layer. Convolutional layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to the input value.

These layers try to find a particular pattern in the input image. The filters are also called feature maps.

2.1.2.2 Pooling layers

Pooling is used to subsample the data. The convolution is a costly process in terms of computation. So to reduce the size of the layers in between the input and hidden layer, we down-sample the data. There are various kinds of pooling, like average-pooling, min-pooling, max-pooling. It turns out that max-pooling works the best.



429	505	686	856
261	792	412	640
633	653	851	751
608	913	713	657

792	856
913	851

Fig. 2: A 2*2 max-pooling, with strides of 2,2

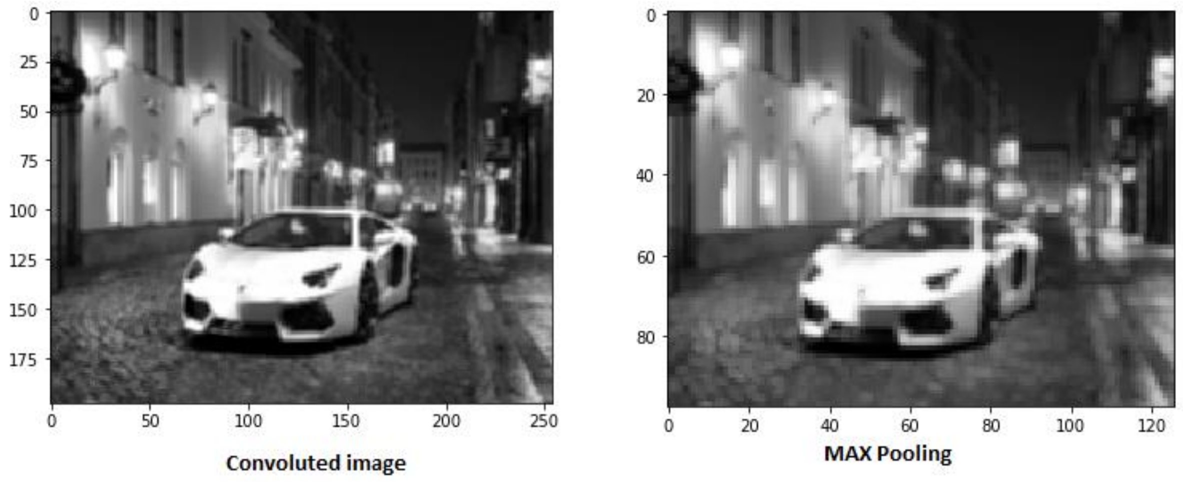


Fig. 3: Max pooling on a real image

2.1.2.3 ReLU

Neural Networks are different from linear functions, they are very powerful instead. They can learn even identify difficult patterns which are tedious to either identify or hand-code. This is possible due to the non-linearity provided by the activation-functions. There are several activation functions like sigmoid, tanh, ReLU and more. For deep networks the ReLU works better than sigmoid and tanh as the other two raise the issue of vanishing gradient**.**

$$\mathbf{ReLU(x) = max(0, x)}$$

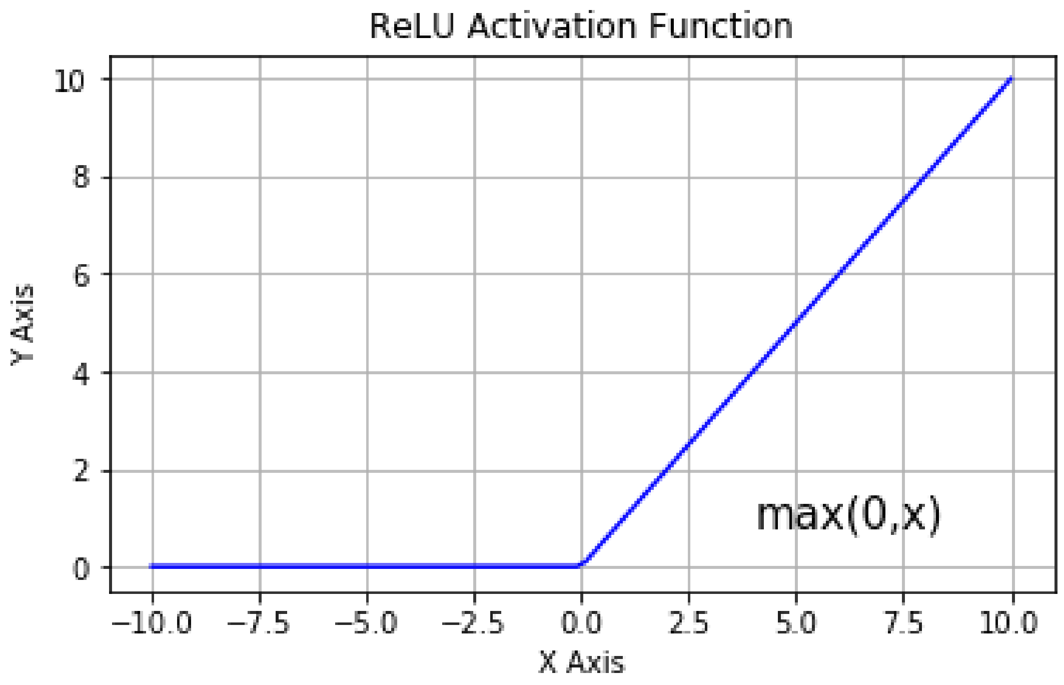


Fig 4: ReLU activation function

2.1.2.4 Fully-Connected Layers

Few(less than 4 generally) fully connected layers are added before the final output layer of the network. The last convolutional layer is flattened and connected densely to the first fully connected layer. More layers can be stacked on one another, or it can be connected to the output layer.

2.2 Why Convolutional Neural Networks?

It might be appealing to use normal-feed forward deep neural networks, however this doesn’t work. Second appealing option is using RNNs but this option doesn’t work either. For the normal-feed forward neural networks, the issue is localization of patterns, i.e. a pattern is local, so connecting a neuron to inputs from all neurons in a previous layer is a waste of memory as well as training time. Instead we use filters/kernels that run over a small portion of a layer and try to identify the patterns in the image. Second big issue is the flattening of the image to feed it as input in a simple feed-forward deep neural networks. Flattening the image destroys a large information about the shape of the image, and images are majorly a shape, while convolutional neural networks don’t do this.

RNNs have a different issue, if the kernel is used for different layers, the weights will be same for one kernel and therefore it cannot be used to identify a different pattern in the image in different layers.

2.3 Approaches

Multiple approaches have been practiced consisting of CNN, ensemble learning CNN, transfer learning with Resnet.

General CNN uses the basic architecture of CNN, while ensemble learning CNN uses multiple CNN architectures at the same time to predict the emotions. Transfer learning uses a very different approach. It takes a pre-trained model and uses the features learnt before to predict the outputs on the new dataset. The pretrained model used is RESNET50.

2.4 Technologies Used

2.4.1 Tensorflow[2]

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows one to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well

2.4.2 Keras[3]

Keras is a high-level neural networks API, written in Python and capable of running on top of Tensorflow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Keras is useful as it :

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

The project uses Sequential and InputModel from keras.models, Dense, Dropout, Flatten, normalization from keras.layers, and np.utils from keras.utils.

2.4.3 Numpy[4]

NumPy is an acronym for "Numeric Python" or "Numerical Python". It is an open source extension module for Python, which provides fast precompiled functions for mathematical and numerical routines. Furthermore, NumPy enriches the programming language Python with powerful data structures for efficient computation of multi-dimensional arrays and matrices. The implementation is even aiming at huge matrices and arrays. Besides that the module supplies a large library of high-level mathematical functions to operate on these matrices and arrays.

2.4.4 Pandas[5]

Pandas is a software library written for the Python programming language. It is used for data manipulation and analysis. It provides special data structures and operations for the manipulation of numerical tables and time series. Pandas is free software released under the three-clause BSD license. There are 2 basic data structures in pandas namely,

1. Series
2. DataFrame

2.4.5 Matplotlib[6]

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python an IPython shell, the jupyter notebook, web application servers, and four graphical user interface toolkits. It can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code.

2.5 Dataset[7]

The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centered 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 in to one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral).

train.csv contains two columns, "emotion" and "pixels". The "emotion" column contains a numeric code ranging from 0 to 6, inclusive, for the emotion that is present in the image. The "pixels" column contains a string surrounded in quotes for each image. The contents of this string a space-separated pixel values in row major order. test.csv contains only the "pixels" column.

The training set consists of 28,709 examples. The public test set used for the leader-board consists of 3,589 examples. The final test set, which was used to determine the winner of the competition, consists of another 3,589 examples.

This dataset was prepared by Pierre-Luc Carrier and Aaron Courville.

3. Implementation

3.1 Models

Multiple models have been tried and tested.

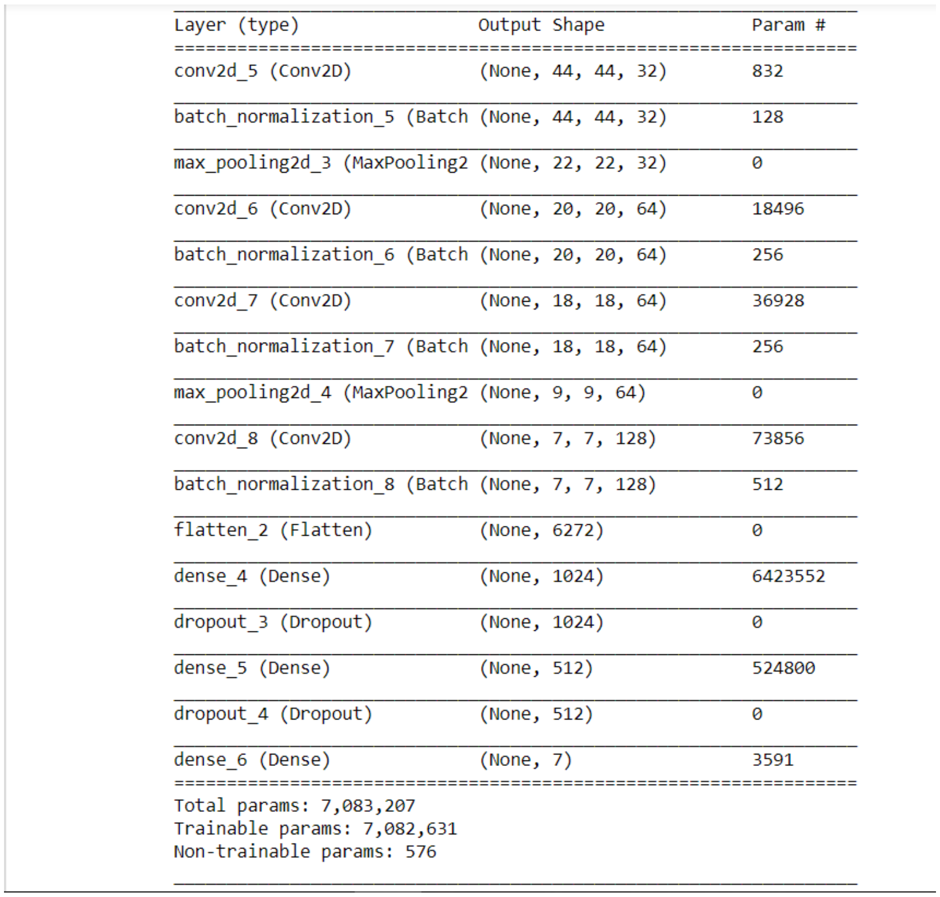
3.1.1 CNN model

In general a model consists of CNN, where the first few layers are convolutional layers, and last few layers are densely connected layers. This was applied using keras Sequential model. Filters of size 5x5, 7x7, 3x3 are used.

Activation function used is ReLU, optimizers used are SGD and Adam.

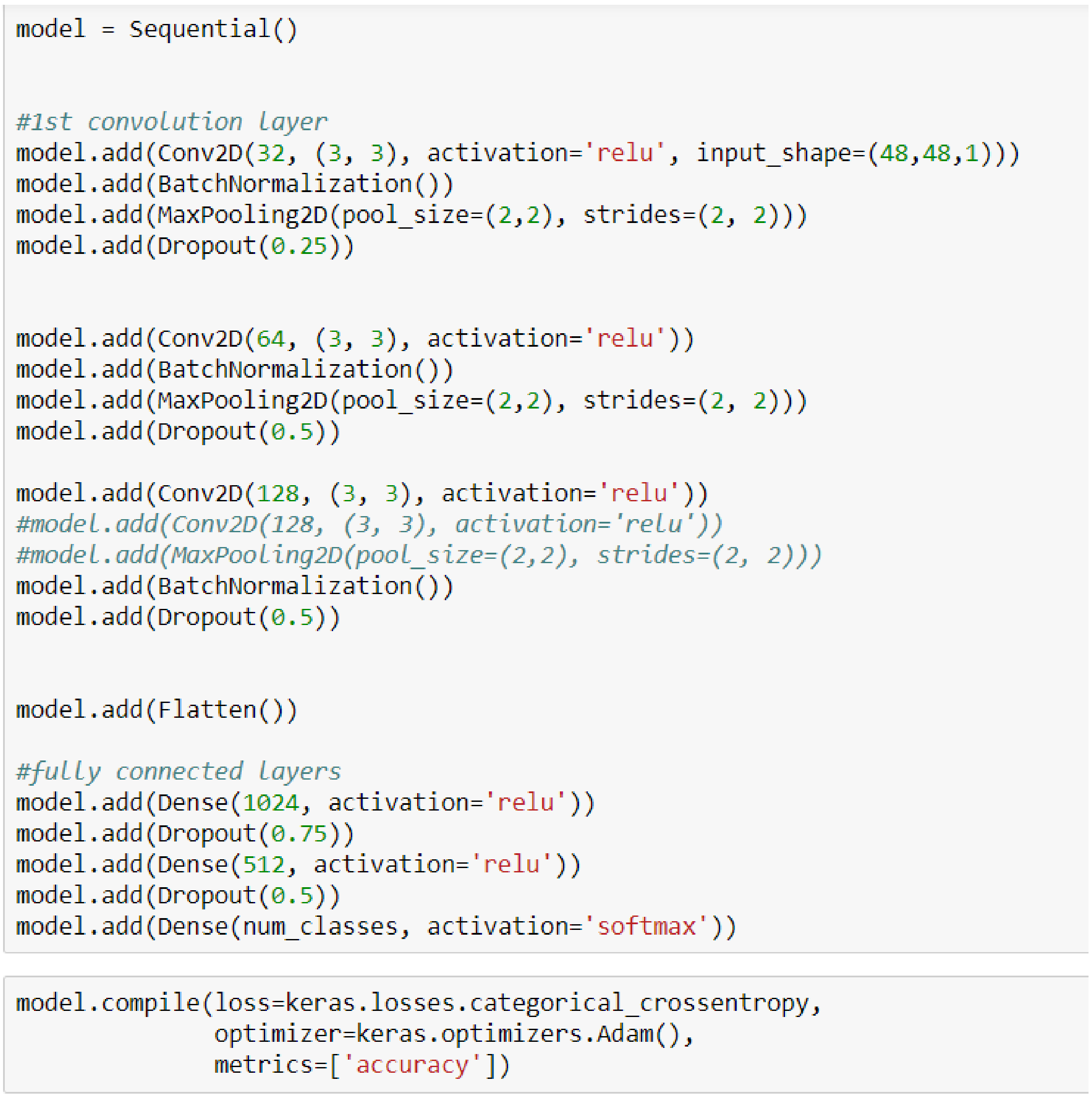
Pooling applied is Max-Pooling

Regularization applied is Dropout(on dense layers)



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 44, 44, 32)        832
_________________________________________________________________
batch_normalization_5 (Batch (None, 44, 44, 32)        128
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 22, 22, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 20, 20, 64)        18496
_________________________________________________________________
batch_normalization_6 (Batch (None, 20, 20, 64)        256
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 18, 18, 64)        36928
_________________________________________________________________
batch_normalization_7 (Batch (None, 18, 18, 64)        256
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 9, 9, 64)          0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 7, 7, 128)         73856
_________________________________________________________________
batch_normalization_8 (Batch (None, 7, 7, 128)         512
_________________________________________________________________
flatten_2 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_4 (Dense)              (None, 1024)              6423552
_________________________________________________________________
dropout_3 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 7)                 3591
=================================================================
Total params: 7,083,207
Trainable params: 7,082,631
Non-trainable params: 576
_________________________________________________________________
```

Fig. 5: A sample model



```
model = Sequential()


#1st convolution layer
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48,48,1)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model.add(Dropout(0.25))


model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(128, (3, 3), activation='relu'))
#model.add(Conv2D(128, (3, 3), activation='relu'))
#model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.5))


model.add(Flatten())

#fully connected layers
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.75))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(),
              metrics=['accuracy'])
```

Code Snippet 1: CNN single network implementation

3.1.2 Transfer Learning with ResNet50

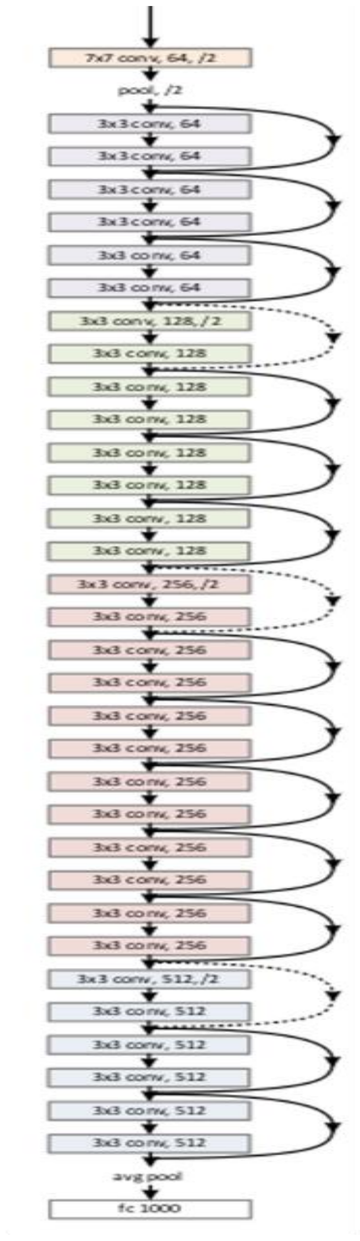
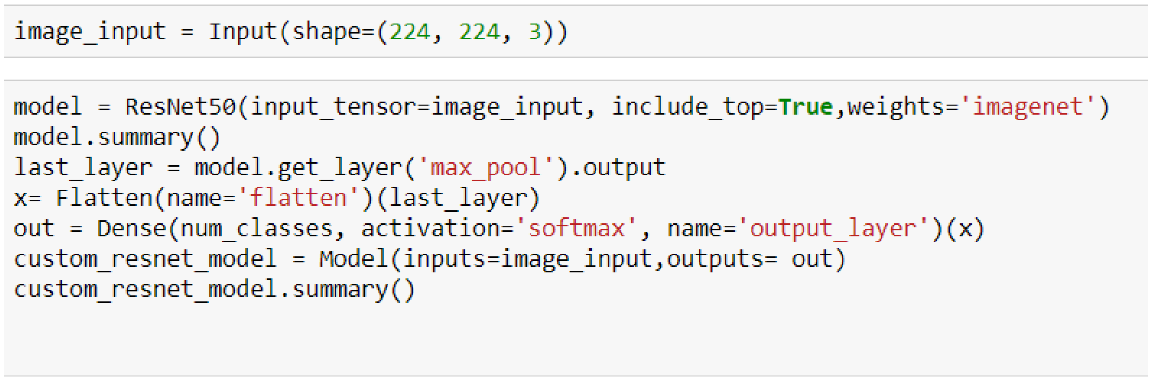


Fig. 6: ResNet50 architecture

The last layer is augmented to the default architecture of ResNet50, with number of neurons equal to the number of classes and only the last layer of custom model is allowed to update. Rest of the model is frozen.



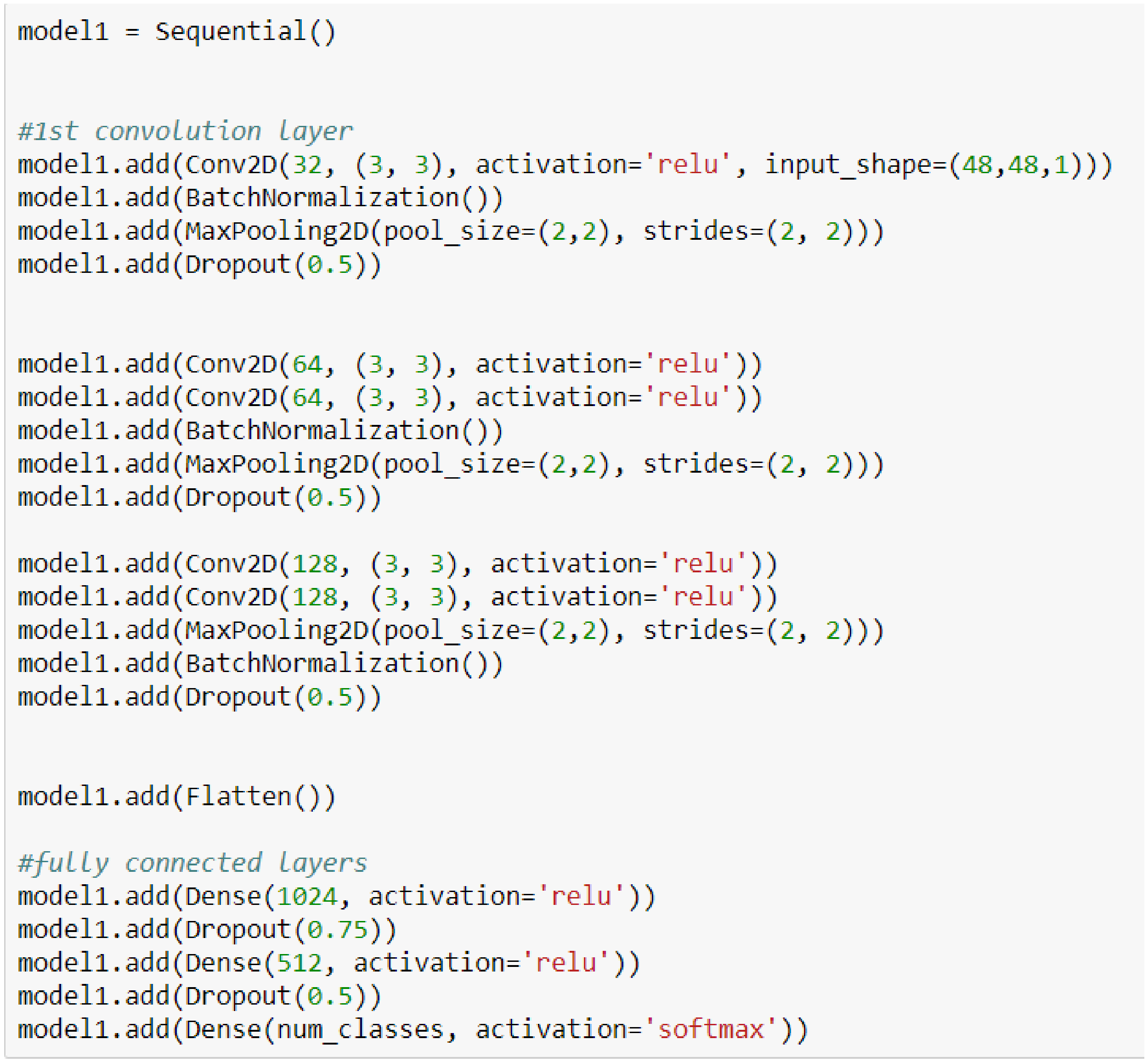
```python
image_input = Input(shape=(224, 224, 3))
```

```python
model = ResNet50(input_tensor=image_input, include_top=True,weights='imagenet')
model.summary()
last_layer = model.get_layer('max_pool').output
x= Flatten(name='flatten')(last_layer)
out = Dense(num_classes, activation='softmax', name='output_layer')(x)
custom_resnet_model = Model(inputs=image_input,outputs= out)
custom_resnet_model.summary()
```

Code Snippet 2: Custom model using ResNet50 architecture

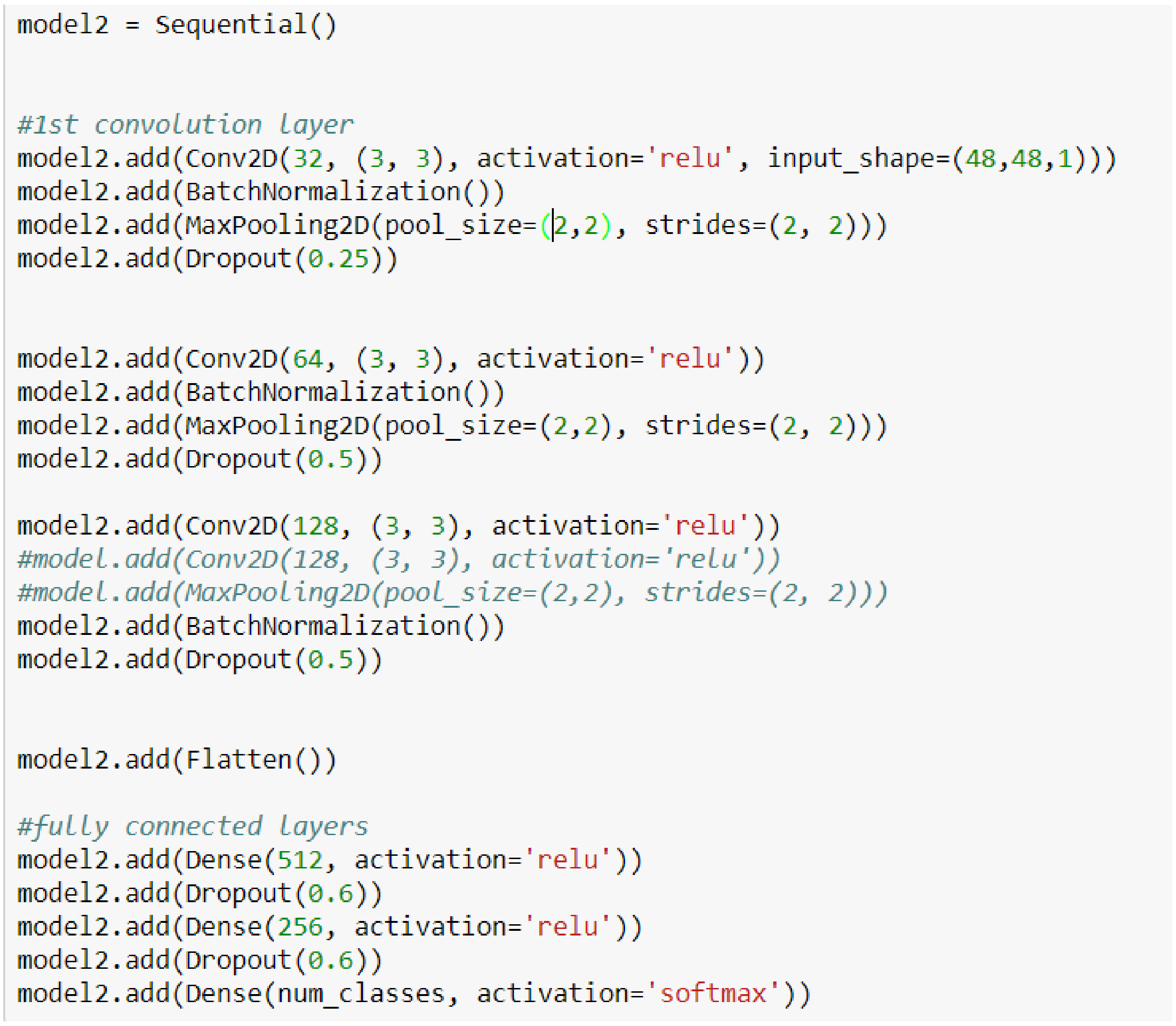
3.1.3 Ensemble learning with CNN

Multiple CNN architectures have been used to perform ensemble learning. In the current model we have 6 CNN architectures.



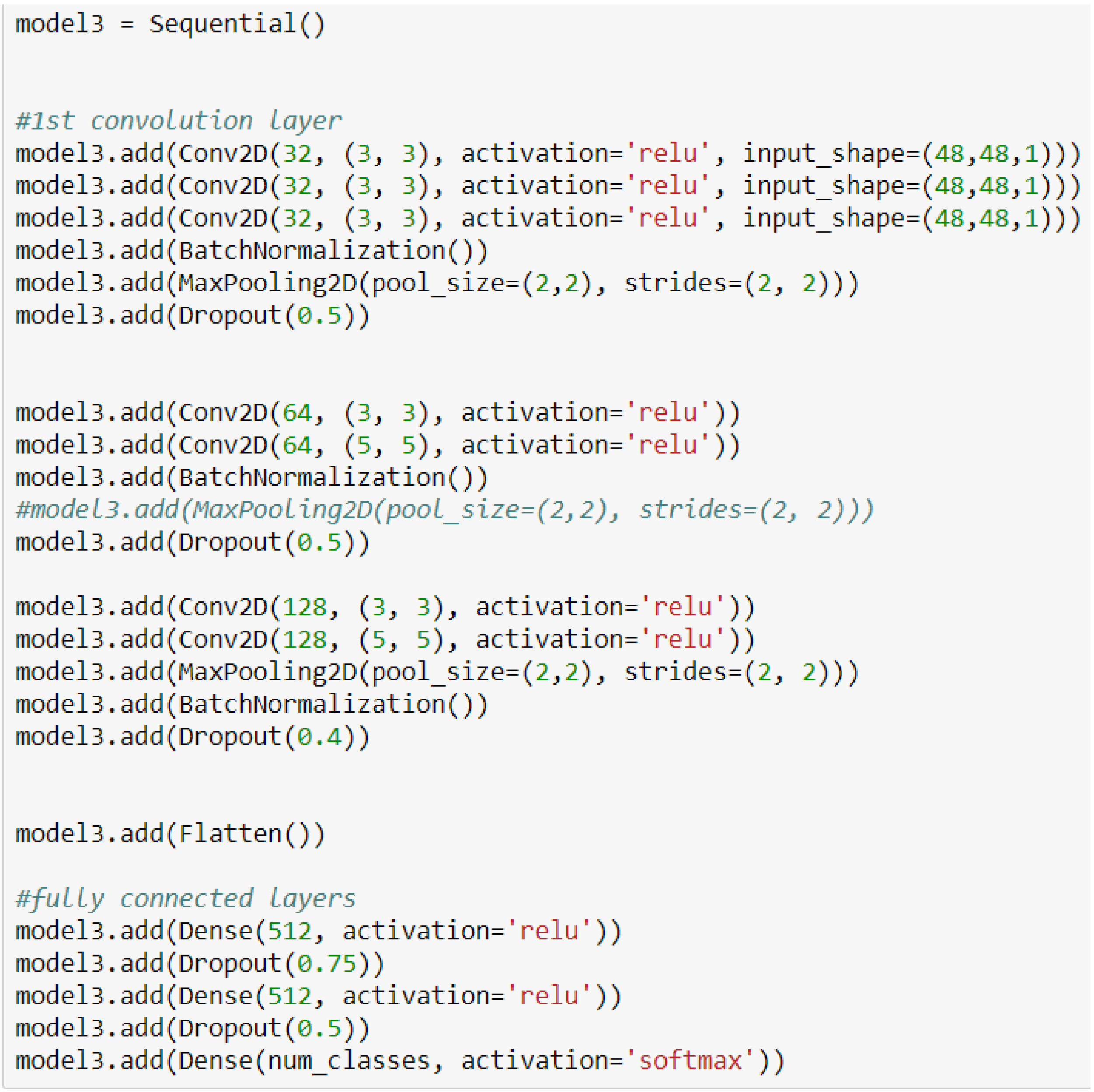
```
model1 = Sequential()


#1st convolution layer
model1.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48,48,1)))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model1.add(Dropout(0.5))


model1.add(Conv2D(64, (3, 3), activation='relu'))
model1.add(Conv2D(64, (3, 3), activation='relu'))
model1.add(BatchNormalization())
model1.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model1.add(Dropout(0.5))

model1.add(Conv2D(128, (3, 3), activation='relu'))
model1.add(Conv2D(128, (3, 3), activation='relu'))
model1.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model1.add(BatchNormalization())
model1.add(Dropout(0.5))


model1.add(Flatten())

#fully connected layers
model1.add(Dense(1024, activation='relu'))
model1.add(Dropout(0.75))
model1.add(Dense(512, activation='relu'))
model1.add(Dropout(0.5))
model1.add(Dense(num_classes, activation='softmax'))
```

Code Snippet 3: Model 1 of the ensemble



```
model2 = Sequential()


#1st convolution layer
model2.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48,48,1)))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model2.add(Dropout(0.25))


model2.add(Conv2D(64, (3, 3), activation='relu'))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model2.add(Dropout(0.5))

model2.add(Conv2D(128, (3, 3), activation='relu'))
#model.add(Conv2D(128, (3, 3), activation='relu'))
#model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model2.add(BatchNormalization())
model2.add(Dropout(0.5))


model2.add(Flatten())

#fully connected layers
model2.add(Dense(512, activation='relu'))
model2.add(Dropout(0.6))
model2.add(Dense(256, activation='relu'))
model2.add(Dropout(0.6))
model2.add(Dense(num_classes, activation='softmax'))
```

Code Snippet 4: Model 2 of the ensemble

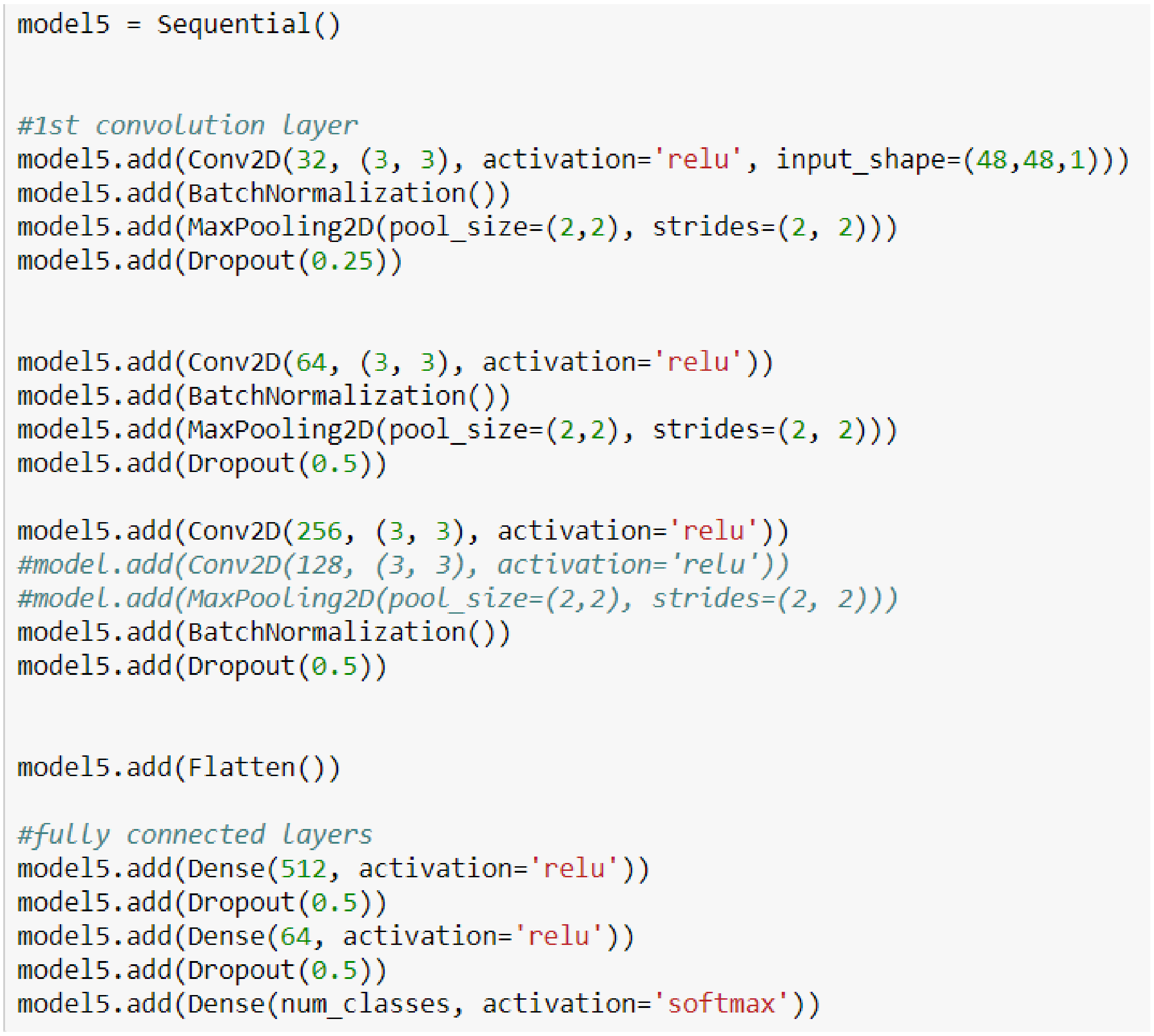


```python
model3 = Sequential()


#1st convolution layer
model3.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48,48,1)))
model3.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48,48,1)))
model3.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48,48,1)))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model3.add(Dropout(0.5))


model3.add(Conv2D(64, (3, 3), activation='relu'))
model3.add(Conv2D(64, (5, 5), activation='relu'))
model3.add(BatchNormalization())
#model3.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model3.add(Dropout(0.5))

model3.add(Conv2D(128, (3, 3), activation='relu'))
model3.add(Conv2D(128, (5, 5), activation='relu'))
model3.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model3.add(BatchNormalization())
model3.add(Dropout(0.4))


model3.add(Flatten())

#fully connected layers
model3.add(Dense(512, activation='relu'))
model3.add(Dropout(0.75))
model3.add(Dense(512, activation='relu'))
model3.add(Dropout(0.5))
model3.add(Dense(num_classes, activation='softmax'))
```

Code Snippet 5: Model 3 of the ensemble

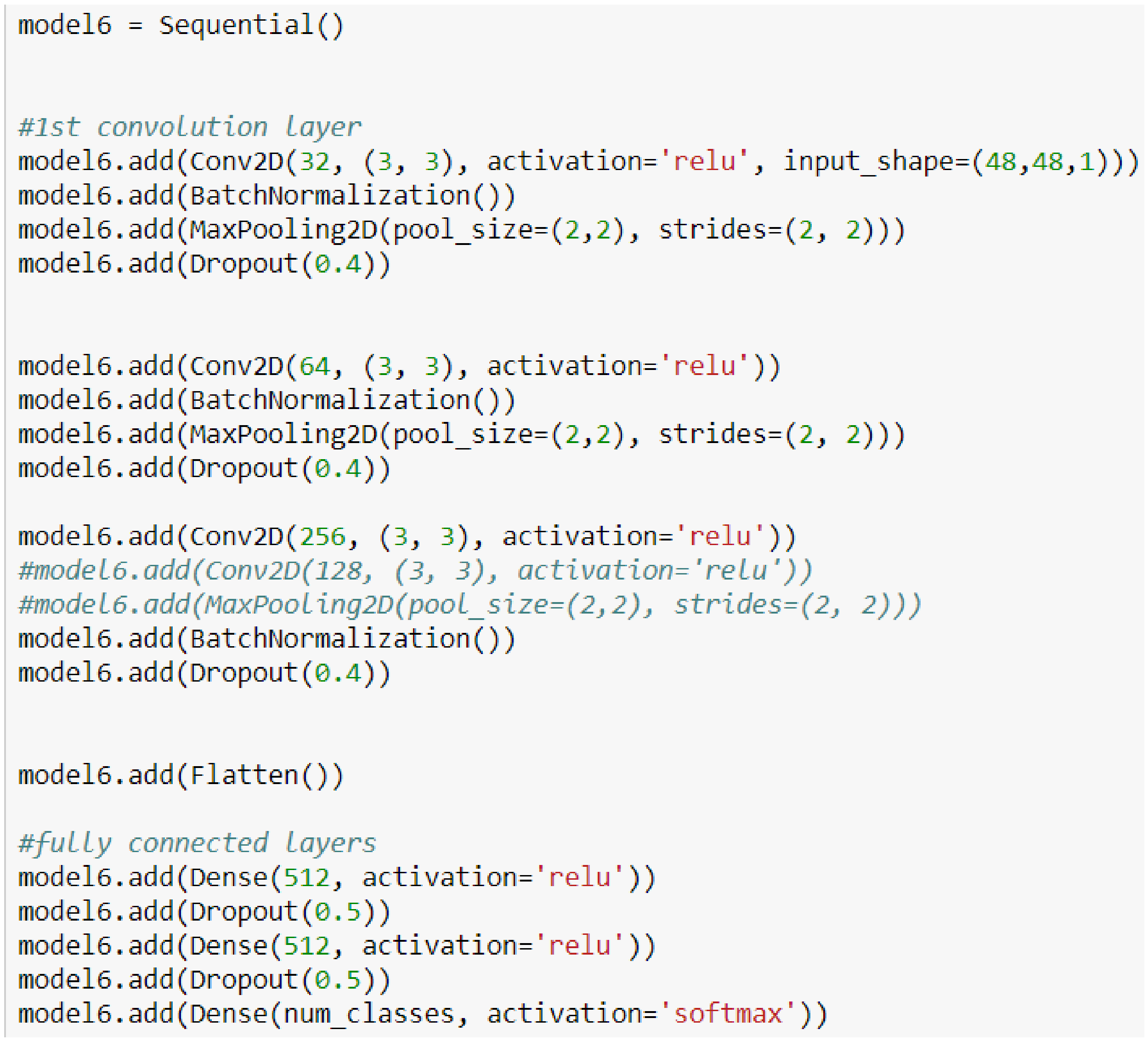
```
model4 = Sequential()


#1st convolution layer
model4.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48,48,1)))
model4.add(BatchNormalization())
model4.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model4.add(Dropout(0.25))


model4.add(Conv2D(64, (3, 3), activation='relu'))
model4.add(BatchNormalization())
model4.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model4.add(Dropout(0.5))

model4.add(Conv2D(128, (5, 5), activation='relu'))
model4.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model4.add(BatchNormalization())
model4.add(Dropout(0.5))


model4.add(Flatten())

#fully connected layers
model4.add(Dense(512, activation='relu'))
model4.add(Dropout(0.75))
model4.add(Dense(128, activation='relu'))
model4.add(Dropout(0.5))
model4.add(Dense(num_classes, activation='softmax'))
```

Code Snippet 6: Model 4 of the ensemble



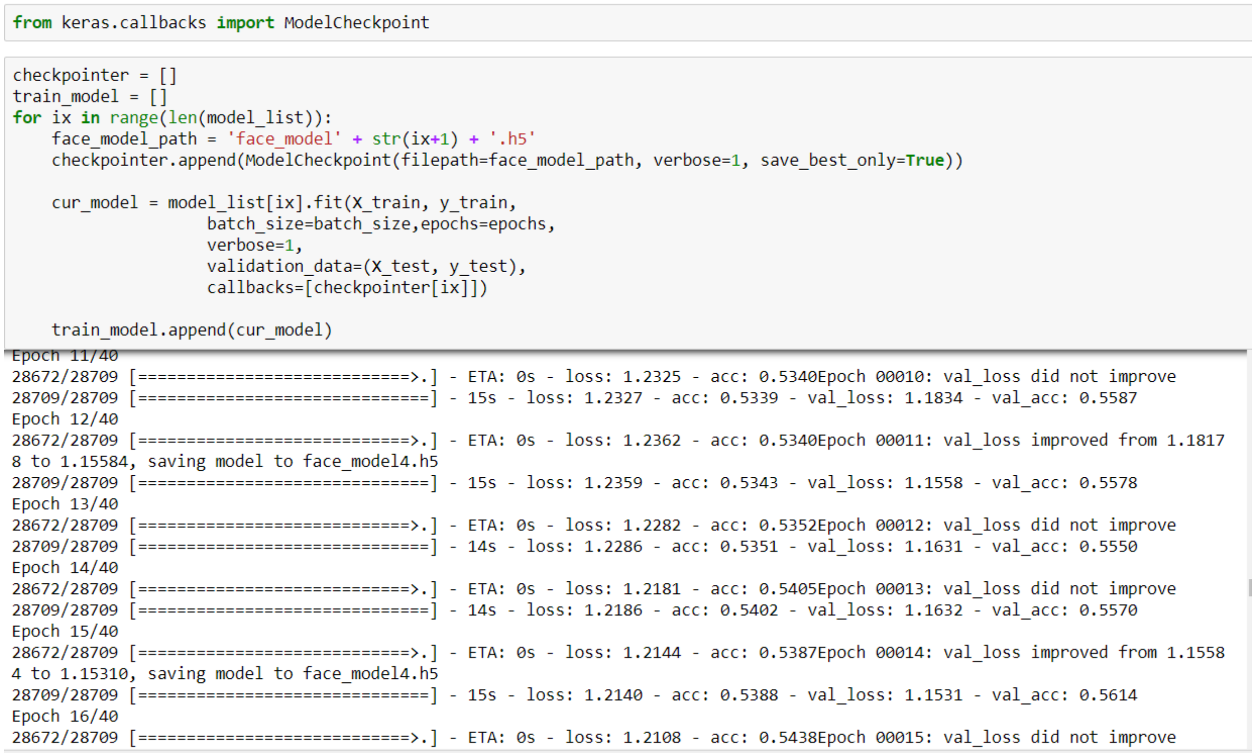
```
model5 = Sequential()


#1st convolution layer
model5.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48,48,1)))
model5.add(BatchNormalization())
model5.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model5.add(Dropout(0.25))


model5.add(Conv2D(64, (3, 3), activation='relu'))
model5.add(BatchNormalization())
model5.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model5.add(Dropout(0.5))

model5.add(Conv2D(256, (3, 3), activation='relu'))
#model.add(Conv2D(128, (3, 3), activation='relu'))
#model.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model5.add(BatchNormalization())
model5.add(Dropout(0.5))


model5.add(Flatten())

#fully connected layers
model5.add(Dense(512, activation='relu'))
model5.add(Dropout(0.5))
model5.add(Dense(64, activation='relu'))
model5.add(Dropout(0.5))
model5.add(Dense(num_classes, activation='softmax'))
```

Code Snippet 7: Model 5 of the ensemble



```python
model6 = Sequential()


#1st convolution layer
model6.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48,48,1)))
model6.add(BatchNormalization())
model6.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model6.add(Dropout(0.4))


model6.add(Conv2D(64, (3, 3), activation='relu'))
model6.add(BatchNormalization())
model6.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model6.add(Dropout(0.4))

model6.add(Conv2D(256, (3, 3), activation='relu'))
#model6.add(Conv2D(128, (3, 3), activation='relu'))
#model6.add(MaxPooling2D(pool_size=(2,2), strides=(2, 2)))
model6.add(BatchNormalization())
model6.add(Dropout(0.4))


model6.add(Flatten())

#fully connected layers
model6.add(Dense(512, activation='relu'))
model6.add(Dropout(0.5))
model6.add(Dense(512, activation='relu'))
model6.add(Dropout(0.5))
model6.add(Dense(num_classes, activation='softmax'))
```

Code Snippet 8: Model 6 of the ensemble



```python
from keras.callbacks import ModelCheckpoint
```

```python
checkpointer = []
train_model = []
for ix in range(len(model_list)):
    face_model_path = 'face_model' + str(ix+1) + '.h5'
    checkpointer.append(ModelCheckpoint(filepath=face_model_path, verbose=1, save_best_only=True))

    cur_model = model_list[ix].fit(X_train, y_train,
                    batch_size=batch_size,epochs=epochs,
                    verbose=1,
                    validation_data=(X_test, y_test),
                    callbacks=[checkpointer[ix]])

    train_model.append(cur_model)
```

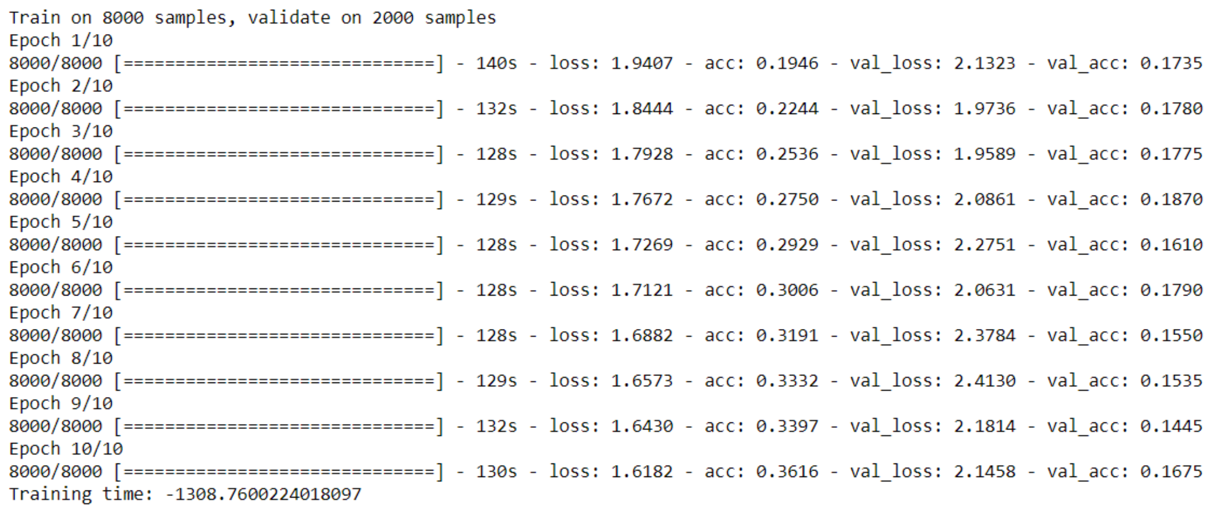
```
Epoch 11/40
28672/28709 [============================>.] - ETA: 0s - loss: 1.2325 - acc: 0.5340Epoch 00010: val_loss did not improve
28709/28709 [==============================] - 15s - loss: 1.2327 - acc: 0.5339 - val_loss: 1.1834 - val_acc: 0.5587
Epoch 12/40
28672/28709 [============================>.] - ETA: 0s - loss: 1.2362 - acc: 0.5340Epoch 00011: val_loss improved from 1.1817
8 to 1.15584, saving model to face_model4.h5
28709/28709 [==============================] - 15s - loss: 1.2359 - acc: 0.5343 - val_loss: 1.1558 - val_acc: 0.5578
Epoch 13/40
28672/28709 [============================>.] - ETA: 0s - loss: 1.2282 - acc: 0.5352Epoch 00012: val_loss did not improve
28709/28709 [==============================] - 14s - loss: 1.2286 - acc: 0.5351 - val_loss: 1.1631 - val_acc: 0.5550
Epoch 14/40
28672/28709 [============================>.] - ETA: 0s - loss: 1.2181 - acc: 0.5405Epoch 00013: val_loss did not improve
28709/28709 [==============================] - 14s - loss: 1.2186 - acc: 0.5402 - val_loss: 1.1632 - val_acc: 0.5570
Epoch 15/40
28672/28709 [============================>.] - ETA: 0s - loss: 1.2144 - acc: 0.5387Epoch 00014: val_loss improved from 1.1558
4 to 1.15310, saving model to face_model4.h5
28709/28709 [==============================] - 15s - loss: 1.2140 - acc: 0.5388 - val_loss: 1.1531 - val_acc: 0.5614
Epoch 16/40
28672/28709 [============================>.] - ETA: 0s - loss: 1.2108 - acc: 0.5438Epoch 00015: val_loss did not improve
```

Code Snippet 9: Training of all the models in the ensemble

3.2 Model Training

The model was trained on 28709 images, out of which 22967 were used to train model and 5742 were used for cross-validation on a GTX GeForce 1050ti 4GB GPU. The model was tested on 3589 images.

To prevent overfitting high dropouts were applied on larger models.

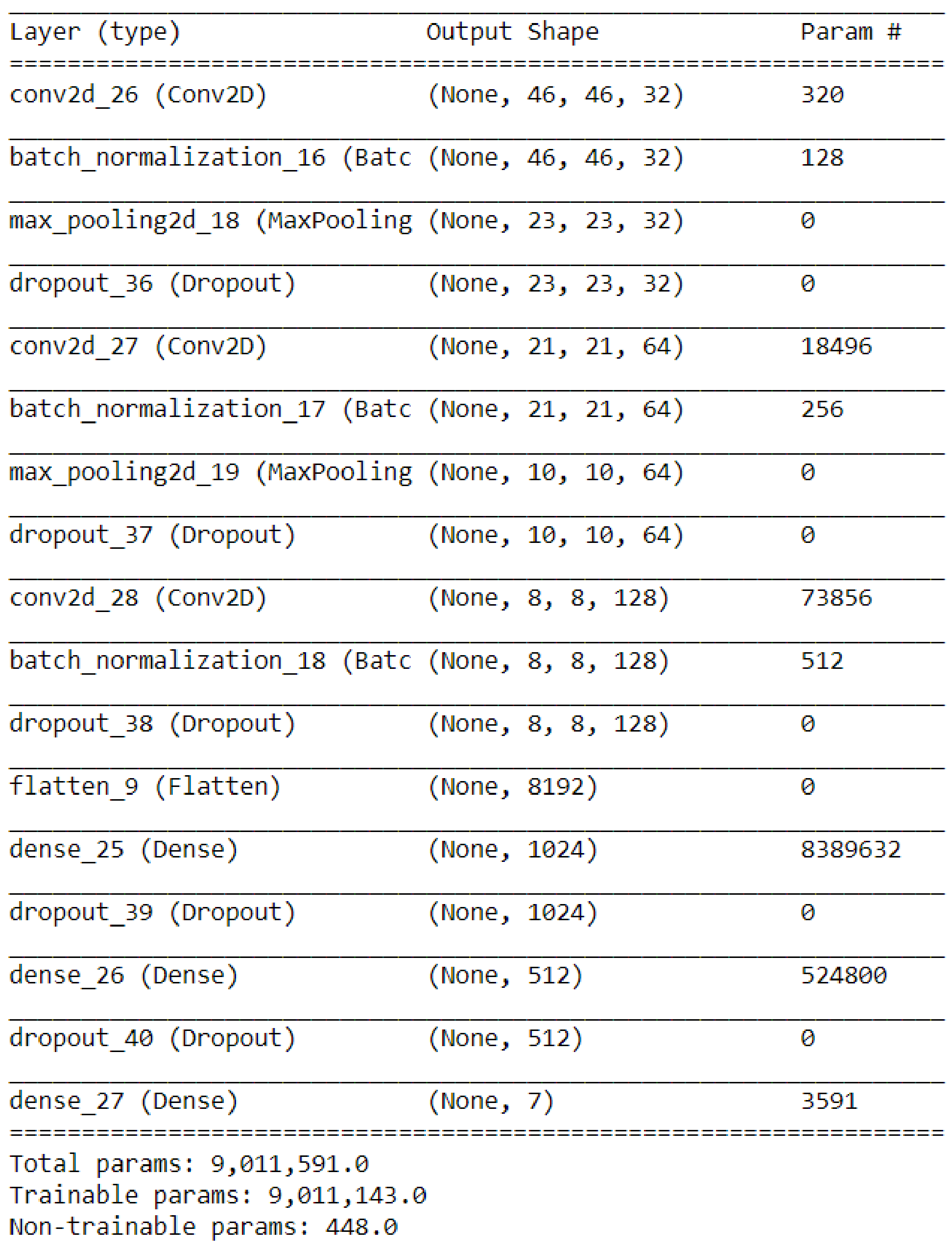


```
Train on 8000 samples, validate on 2000 samples
Epoch 1/10
8000/8000 [==============================] - 140s - loss: 1.9407 - acc: 0.1946 - val_loss: 2.1323 - val_acc: 0.1735
Epoch 2/10
8000/8000 [==============================] - 132s - loss: 1.8444 - acc: 0.2244 - val_loss: 1.9736 - val_acc: 0.1780
Epoch 3/10
8000/8000 [==============================] - 128s - loss: 1.7928 - acc: 0.2536 - val_loss: 1.9589 - val_acc: 0.1775
Epoch 4/10
8000/8000 [==============================] - 129s - loss: 1.7672 - acc: 0.2750 - val_loss: 2.0861 - val_acc: 0.1870
Epoch 5/10
8000/8000 [==============================] - 128s - loss: 1.7269 - acc: 0.2929 - val_loss: 2.2751 - val_acc: 0.1610
Epoch 6/10
8000/8000 [==============================] - 128s - loss: 1.7121 - acc: 0.3006 - val_loss: 2.0631 - val_acc: 0.1790
Epoch 7/10
8000/8000 [==============================] - 128s - loss: 1.6882 - acc: 0.3191 - val_loss: 2.3784 - val_acc: 0.1550
Epoch 8/10
8000/8000 [==============================] - 129s - loss: 1.6573 - acc: 0.3332 - val_loss: 2.4130 - val_acc: 0.1535
Epoch 9/10
8000/8000 [==============================] - 132s - loss: 1.6430 - acc: 0.3397 - val_loss: 2.1814 - val_acc: 0.1445
Epoch 10/10
8000/8000 [==============================] - 130s - loss: 1.6182 - acc: 0.3616 - val_loss: 2.1458 - val_acc: 0.1675
Training time: -1308.7600224018097
```

Fig. 7: A sample model training

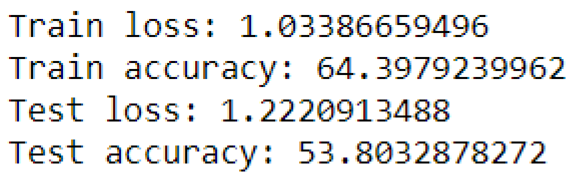
4. Results

CNN Model: The best accuracy we get is around 55% on single CNN network.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_26 (Conv2D)           (None, 46, 46, 32)        320       
_________________________________________________________________
batch_normalization_16 (Batc (None, 46, 46, 32)        128       
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 23, 23, 32)        0         
_________________________________________________________________
dropout_36 (Dropout)         (None, 23, 23, 32)        0         
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 21, 21, 64)        18496     
_________________________________________________________________
batch_normalization_17 (Batc (None, 21, 21, 64)        256       
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 10, 10, 64)        0         
_________________________________________________________________
dropout_37 (Dropout)         (None, 10, 10, 64)        0         
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 8, 8, 128)         73856     
_________________________________________________________________
batch_normalization_18 (Batc (None, 8, 8, 128)         512       
_________________________________________________________________
dropout_38 (Dropout)         (None, 8, 8, 128)         0         
_________________________________________________________________
flatten_9 (Flatten)          (None, 8192)              0         
_________________________________________________________________
dense_25 (Dense)             (None, 1024)              8389632   
_________________________________________________________________
dropout_39 (Dropout)         (None, 1024)              0         
_________________________________________________________________
dense_26 (Dense)             (None, 512)               524800    
_________________________________________________________________
dropout_40 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_27 (Dense)             (None, 7)                 3591      
=================================================================
Total params: 9,011,591.0
Trainable params: 9,011,143.0
Non-trainable params: 448.0
```

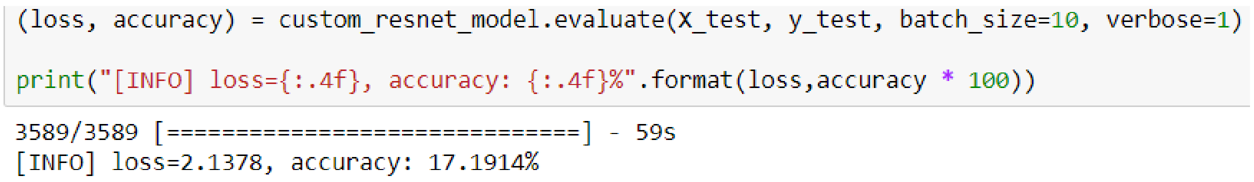
Fig. 8.1:Single CNN Model architecture



```
Train loss: 1.03386659496
Train accuracy: 64.3979239962
Test loss: 1.2220913488
Test accuracy: 53.8032878272
```

Fig 8.2: Loss and Accuracy for model on training and test data

Transfer learning with ResNet50 model: The accuracy of ResNet was very low even when compared to a single CNN.

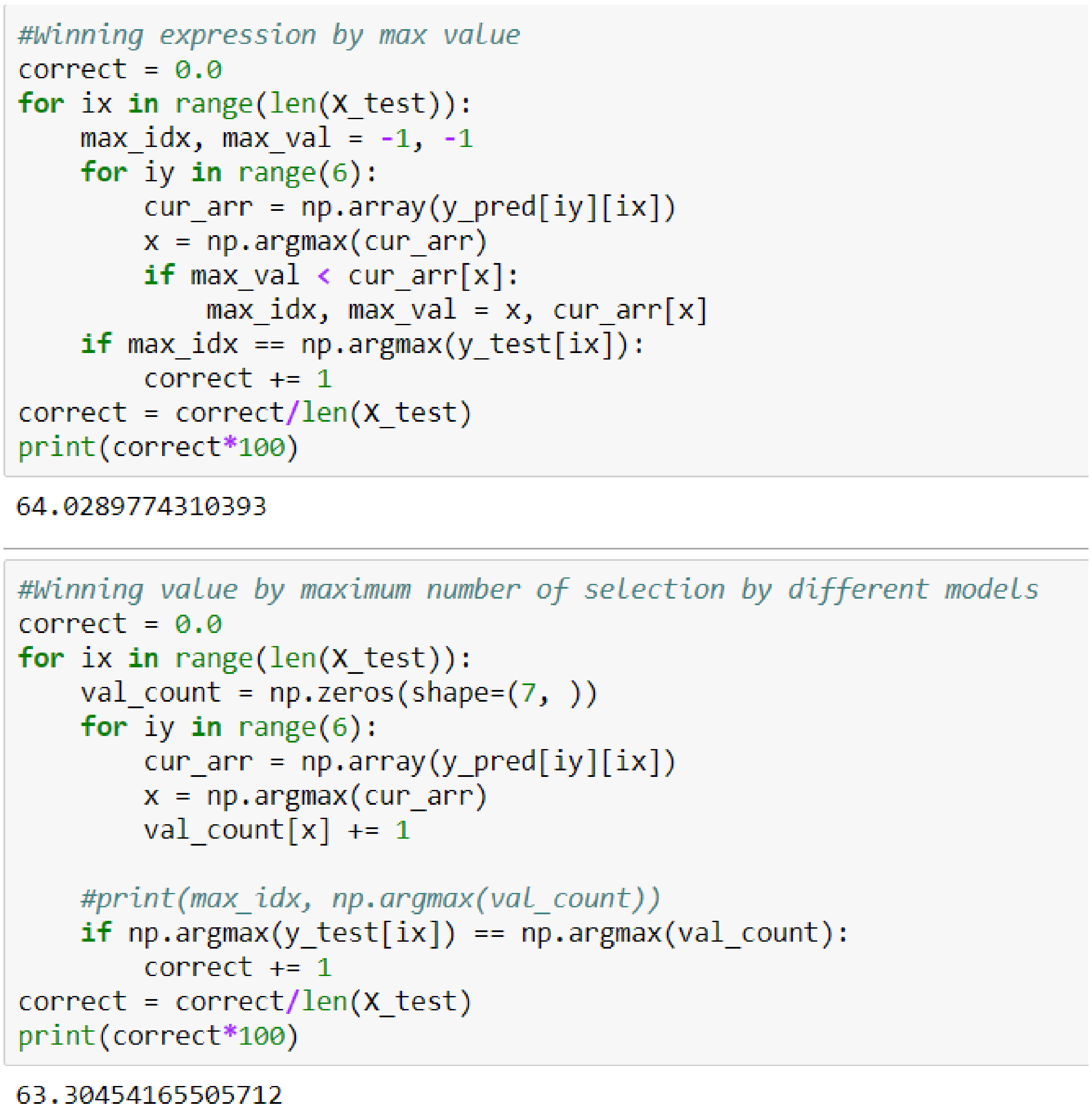


```
(loss, accuracy) = custom_resnet_model.evaluate(X_test, y_test, batch_size=10, verbose=1)

print("[INFO] loss={:.4f}, accuracy: {:.4f}%".format(loss,accuracy * 100))
```

```
3589/3589 [==============================] - 59s
[INFO] loss=2.1378, accuracy: 17.1914%
```

Fig 9: Loss and Accuracy for ResNet model on training and test data

Ensemble learning with CNN: The ensemble model give nearly 10% increase in accuracy than a single CNN subnet



```
#Winning expression by max value
correct = 0.0
for ix in range(len(X_test)):
    max_idx, max_val = -1, -1
    for iy in range(6):
        cur_arr = np.array(y_pred[iy][ix])
        x = np.argmax(cur_arr)
        if max_val < cur_arr[x]:
            max_idx, max_val = x, cur_arr[x]
    if max_idx == np.argmax(y_test[ix]):
        correct += 1
correct = correct/len(X_test)
print(correct*100)
```

```
64.0289774310393
```

```
#Winning value by maximum number of selection by different models
correct = 0.0
for ix in range(len(X_test)):
    val_count = np.zeros(shape=(7, ))
    for iy in range(6):
        cur_arr = np.array(y_pred[iy][ix])
        x = np.argmax(cur_arr)
        val_count[x] += 1

    #print(max_idx, np.argmax(val_count))
    if np.argmax(y_test[ix]) == np.argmax(val_count):
        correct += 1
correct = correct/len(X_test)
print(correct*100)
```

```
63.30454165505712
```

Fig 10: Accuracy of ensemble model on test data

5. Inference

In this project we have reviewed the state of the art in CNN based FER. We have observed that CNN performs better than ResNet50 model. The Microsoft's ResNet model was winner of ImageNet challenge in 2015. However, due to the difference in FER and ImageNet contest, the model doesn't work well and ends up with an accuracy of near-about 20%. Finally, we have demonstrated that an ensemble of such CNNs outperforms state of the art methods without the use of additional training data or requiring face registration. The increment in accuracy we get is overwhelming. The accuracy reaches up to human level which according to the experiments conducted by Ian J. Goodfellow is around 65%.

6. References

[1] Andrej Karpathy. CS231n Convolutional Neural Networks for Visual Recognition. URL http://cs231n.github.io/convolutional-networks

[2] TensorFlow. URL https://www.tensorflow.org/

[3] Keras Documentation. URL https://www.keras.io/

[4] Numpy. URL http://www.numpy.org/

[5] Pandas. URL https://www.python-course.eu/pandas.php

[6] Matplotlib. URL https://matplotlib.org/

[7] Challenges in Representation Learning: Facial Expression Recognition Challenge. URL https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge

[8] Ian J. Goodfellow, Dumitru Erhan , Pierre Luc Carrier, Aaron Courville, Mehdi Mirza, Ben Hamner, Will Cukierski, Yichuan Tang, David Thaler, Dong-Hyun Lee, Yingbo Zhou, Chetan Ramaiah, Fangxiang Feng, Ruifan Li, Xiaojie Wang, Dimitris Athanasakis, John Shawe-Taylor, Maxim Milakov, John Park, Radu Ionescu, Marius Popescu, Cristian Grozea, James Bergstra, Jingjing Xie, Lukasz Romaszko, Bing Xu, Zhang Chuang, and Yoshua Bengio. Challenges in Representation Learning: A report on three machine learning contests. Technical Report arXiv: 1307.0414, U. Montreal, 2012. URL https://arxiv.org/abs/1307.0414.

[9] Lukasz Romaszko. A deep learning approach with an ensemble-based neural network classifier for black box icml 2013 contest. Workshop on Challenges in Representation Learning, ICML, 2013.

[10] James Bergstra and David D. Cox. Hyperparameter optimization and boosting for classifying facial expressions: How good can a “null” model be? Workshop on Challenges in Representation Learning, ICML, 2013.

[11] Joshua Susskind, Adam Anderson, and Geoffrey E. Hinton. The Toronto face dataset. Technical Report UTML TR 2010-001, U. Toronto, 2010.