State Farm Distracted Driver Detection

Made By:

- Mohamed Khalifa
- Sara Samer
- Amira Adel
- Hadeer Mahdy
- Ahmed Gamal

Dataset

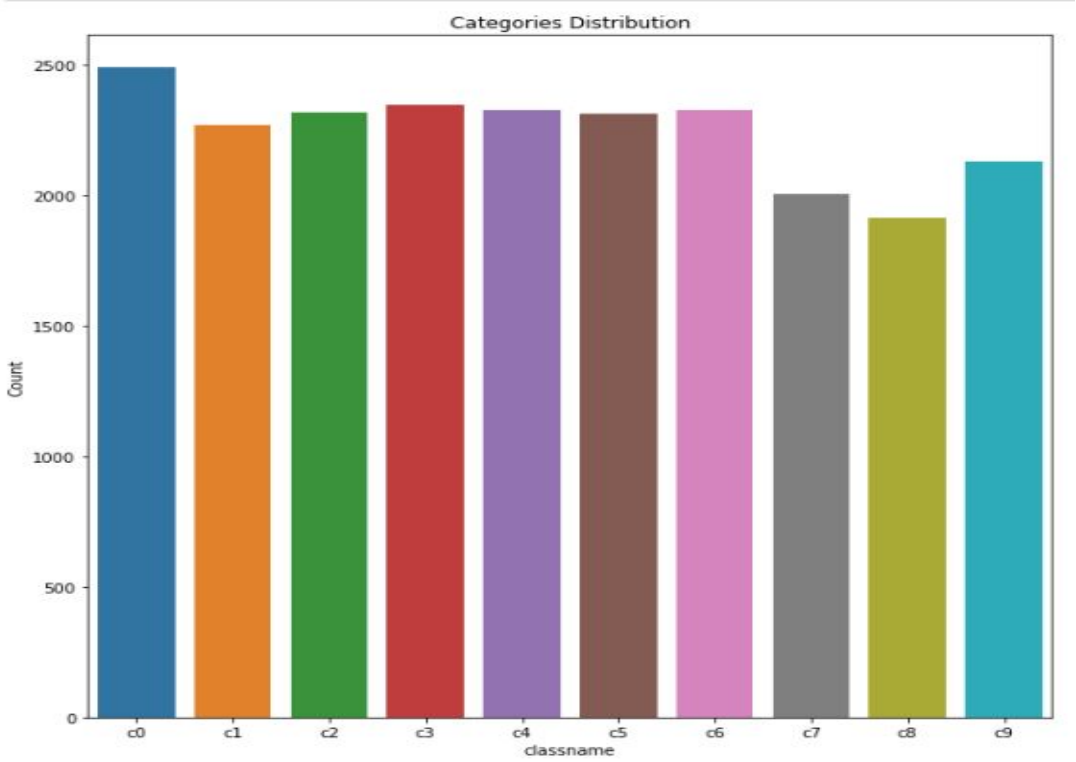
The dataset used in this project was provided by State Farm through a Kaggle competition, which is a set of images of drivers taken inside a car capturing their activities such as texting, talking on the phone, eating, reaching behind, putting on makeup, etc. As shown in Figure 1. These activities are classified into 10 classes as:

- c0: safe driving
- c1: texting — right
- c2: talking on the phone — right
- c3: texting — left
- c4: talking on the phone — left
- c5: operating the radio
- c6: drinking
- c7: reaching behind
- c8: hair and makeup
- c9: talking to a passenger

Exploratory Visualization

Data Exploration

There is one characteristic about the input data that may need to be addressed. I noticed that some categories have more images than the others as we can see from the chart below.



1. Data Preprocessing

These are the data preprocessing steps that positively impacted results and were applied on the data:

1. Images are converted into 3 channels, RGB – This preprocessing is done so models may use the 3 channels to learn features with the objective of improving accuracy and log loss.

2. Images are resized to 224 x 224 – Resizing images makes it easier to load on memory at the cost of losing some details.

3.The list of images are normalized and divided into a train set and validation set - This division is important so that we could validate if our model is improving or not when it is training.

Statistics

```
# Statistics
# Load the list of names
names = [item[17:19] for item in sorted(glob("../input/state-farm-distracted-driver-detection/imgs/train/*/"))]
test_files_size = len(np.array(glob(os.path.join('..', 'input', '../input/state-farm-distracted-driver-detection/imgs/test/', '*.jpg'))))
x_train_size = len(x_train)
categories_size = len(names)
x_test_size = len(x_test)
print('There are %s total images.\n' % (test_files_size + x_train_size + x_test_size))
print('There are %d training images.' % x_train_size)
print('There are %d total training categories.' % categories_size)
print('There are %d validation images.' % x_test_size)
print('There are %d test images.'% test_files_size)
```

```
There are 102150 total images.

There are 17939 training images.
There are 10 total training categories.
There are 4485 validation images.
There are 79726 test images.
```

2. Baseline Dense Model

Model summary:

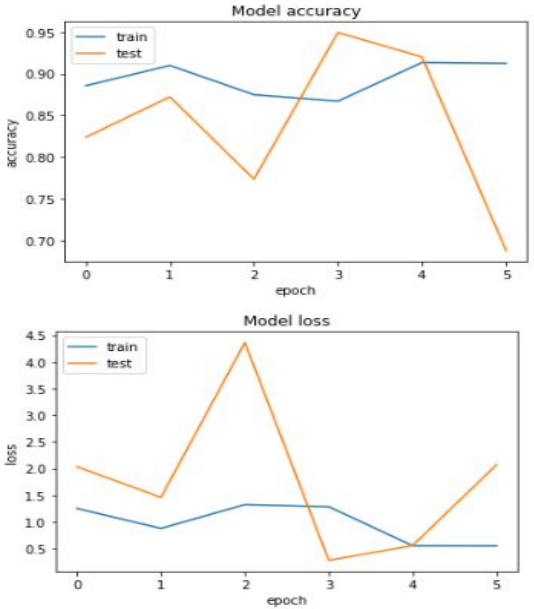
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               25690624
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 25,856,138
Trainable params: 25,856,138
Non-trainable params: 0
```

Model accuracy:

```
# Training the Dense Model version
history_v0 = model_v0.fit(x_train.reshape(17939,224*224*1), y_train,
          validation_data=(x_test.reshape(4485,224*224*1), y_test),
          callbacks=callbacks,
          epochs=nb_epoch, batch_size=batch_size, verbose=1)
```

```
Epoch 1/10
449/449 [==============================] - 4s 9ms/step - loss: 1.2520 - accuracy: 0.8857 - val_loss: 2.0341 - val_accuracy: 0.8239

Epoch 00001: val_loss did not improve from 1.98025
Epoch 2/10
449/449 [==============================] - 3s 8ms/step - loss: 0.8795 - accuracy: 0.9099 - val_loss: 1.4608 - val_accuracy: 0.8722

Epoch 00002: val_loss improved from 1.98025 to 1.46082, saving model to /saved_models/weights_best_vanilla.hdf5
Epoch 3/10
449/449 [==============================] - 3s 7ms/step - loss: 1.3258 - accuracy: 0.8749 - val_loss: 4.3618 - val_accuracy: 0.7735

Epoch 00003: val_loss did not improve from 1.46082
Epoch 4/10
449/449 [==============================] - 3s 7ms/step - loss: 1.2856 - accuracy: 0.8671 - val_loss: 0.2810 - val_accuracy: 0.9494

Epoch 00004: val_loss improved from 1.46082 to 0.28098, saving model to /saved_models/weights_best_vanilla.hdf5
Epoch 5/10
449/449 [==============================] - 3s 8ms/step - loss: 0.5544 - accuracy: 0.9135 - val_loss: 0.5580 - val_accuracy: 0.9202

Epoch 00005: val_loss did not improve from 0.28098
Epoch 6/10
449/449 [==============================] - 3s 7ms/step - loss: 0.5534 - accuracy: 0.9124 - val_loss: 2.0716 - val_accuracy: 0.6881

Epoch 00006: val_loss did not improve from 0.28098
Epoch 00006: early stopping
```

```
plot_train_history(history_v0)
```




```
score = model_v0.evaluate(x_test.reshape(4485,224*224*1), y_test, verbose=1)
print('Score: ', score)
```

```
141/141 [==============================] - 0s 3ms/step - loss: 2.0716 - accuracy: 0.6881
Score:  [2.0716123580932617, 0.6880713701248169]
```

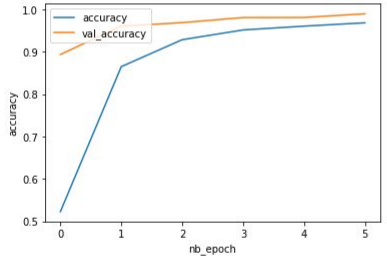
3. Baseline CNN

Model summary:

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 62, 62, 32)        320
_________________________________________________________________
batch_normalization_14 (Batc (None, 62, 62, 32)        128
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 62, 62, 32)        9248
_________________________________________________________________
batch_normalization_15 (Batc (None, 62, 62, 32)        128
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 31, 31, 32)        0
_________________________________________________________________
dropout_10 (Dropout)         (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 31, 31, 64)        18496
_________________________________________________________________
batch_normalization_16 (Batc (None, 31, 31, 64)        256
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 31, 31, 64)        36928
_________________________________________________________________
batch_normalization_17 (Batc (None, 31, 31, 64)        256
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
dropout_11 (Dropout)         (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 16, 16, 128)       73856
_________________________________________________________________
batch_normalization_18 (Batc (None, 16, 16, 128)       512
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 16, 16, 128)       147584
_________________________________________________________________
batch_normalization_19 (Batc (None, 16, 16, 128)       512
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
dropout_12 (Dropout)         (None, 8, 8, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_14 (Dense)             (None, 512)               4194816
_________________________________________________________________
batch_normalization_20 (Batc (None, 512)               2048
_________________________________________________________________
dropout_13 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_15 (Dense)             (None, 128)               65664
_________________________________________________________________
dropout_14 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_16 (Dense)             (None, 10)                1290
=================================================================
Total params: 4,552,042
Trainable params: 4,550,122
Non-trainable params: 1,920
```

Model accuracy:

```
plt.plot(history_v2.history["accuracy"])
plt.plot(history_v2.history['val_accuracy'])

plt.ylabel('accuracy')
plt.xlabel('nb_epoch')
plt.legend(['accuracy', 'val_accuracy'], loc='upper left')

plt.show()
```



+ Code + Markdown

```
score1 = model.evaluate(x_test, y_test, verbose=1)
print('Loss: ', score1[0])
print('Accuracy: ', score1[1]*100, ' %')
```

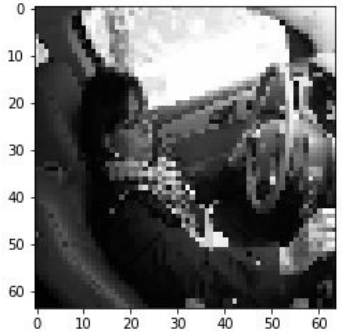
```
141/141 [==============================] - 15s 103ms/step - loss: 0.0351 - accuracy: 0.9897
Loss:  0.035131312906742096
Accuracy:  98.97435903549194  %
```

```
history_v2 = model.fit(x_train, y_train,
          validation_data=(x_test, y_test),
          epochs=nb_epoch, batch_size=batch_size, verbose=1)
```

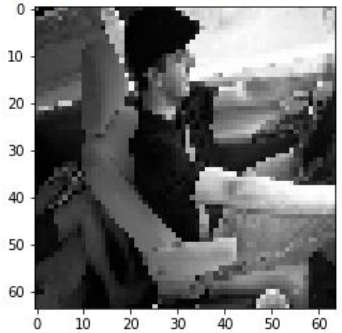
```
Epoch 1/6
281/281 [==============================] - 220s 771ms/step - loss: 1.5443 - accuracy: 0.4926 - val_loss: 0.4259 - val_accuracy: 0.8658
Epoch 2/6
281/281 [==============================] - 213s 757ms/step - loss: 0.4442 - accuracy: 0.8537 - val_loss: 0.1538 - val_accuracy: 0.9532
Epoch 3/6
281/281 [==============================] - 209s 745ms/step - loss: 0.2455 - accuracy: 0.9216 - val_loss: 0.0702 - val_accuracy: 0.9808
Epoch 4/6
281/281 [==============================] - 209s 745ms/step - loss: 0.1742 - accuracy: 0.9451 - val_loss: 0.0656 - val_accuracy: 0.9824
Epoch 5/6
281/281 [==============================] - 212s 755ms/step - loss: 0.1293 - accuracy: 0.9589 - val_loss: 0.0574 - val_accuracy: 0.9815
Epoch 6/6
281/281 [==============================] - 212s 755ms/step - loss: 0.0985 - accuracy: 0.9686 - val_loss: 0.0421 - val_accuracy: 0.9882
```

Prediction Samples:

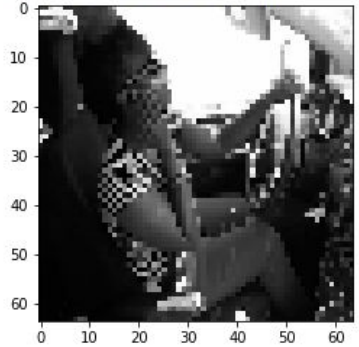
```
1/1 [==============================] - 0s 370ms/step
Y prediction: [[1.9079283e-09 8.7641758e-16 2.4082774e-11 5.5070032e-10 2.4784116e-10
  9.9999964e-01 3.3204529e-12 8.7869495e-10 9.7563579e-08 2.1145745e-07]]
Predicted: Operating the radio
```




```
1/1 [==============================] - 0s 50ms/step
Y prediction: [[9.6494182e-07 1.5600721e-09 6.6973621e-07 4.2936104e-09 2.5467530e-06
  9.9985135e-01 4.9533831e-08 5.3989105e-08 1.6404088e-06 1.4281916e-04]]
Predicted: Operating the radio
```




```
1/1 [==============================] - 0s 51ms/step
Y prediction: [[1.5194789e-06 6.5872609e-06 6.5887538e-03 6.2759011e-09 2.4896980e-07
  1.8446251e-08 2.3642656e-07 6.7874826e-06 9.9319708e-01 1.9877782e-04]]
Predicted: Hair and makeup
```




```
1/1 [==============================] - 0s 45ms/step
Y prediction: [[1.5350214e-03 1.5164029e-07 1.6971776e-06 5.8196956e-01 2.9925475e-01
  9.9514835e-02 1.1400979e-07 2.0523319e-06 3.2206574e-03 1.4501143e-02]]
Predicted: Texting - left
```

4. Data Augmentation

Data generator:

```
train_datagen = ImageDataGenerator(rescale = 1.0/255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True,
                                   validation_split = 0.2)

test_datagen = ImageDataGenerator(rescale=1.0/ 255, validation_split = 0.2)
```

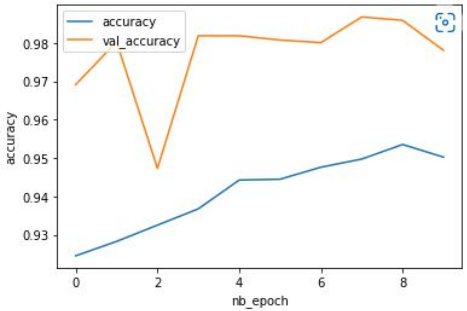
Model Accuracy:

```
history_v1 = model_1.fit_generator(training_generator,
                        steps_per_epoch = nb_train_samples // batch_size,
                        epochs = 10,
                        verbose = 1,
                        callbacks=callbacks,
                        validation_data = validation_generator,
                        validation_steps = nb_validation_samples // batch_size)
```

```
/opt/conda/lib/python3.7/site-packages/keras/engine/training.py:1972: UserWarning:

`Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.

Epoch 1/10
280/280 [==============================] - 219s 781ms/step - loss: 0.2360 - accuracy: 0.9245 - val_loss: 0.1119 - val_accuracy: 0.9692
Epoch 2/10
280/280 [==============================] - 222s 793ms/step - loss: 0.2268 - accuracy: 0.9283 - val_loss: 0.0595 - val_accuracy: 0.9804
Epoch 3/10
280/280 [==============================] - 220s 786ms/step - loss: 0.2088 - accuracy: 0.9325 - val_loss: 0.1863 - val_accuracy: 0.9473
Epoch 4/10
280/280 [==============================] - 219s 781ms/step - loss: 0.1983 - accuracy: 0.9368 - val_loss: 0.0495 - val_accuracy: 0.9819
Epoch 5/10
280/280 [==============================] - 218s 778ms/step - loss: 0.1776 - accuracy: 0.9443 - val_loss: 0.0545 - val_accuracy: 0.9819
Epoch 6/10
280/280 [==============================] - 223s 796ms/step - loss: 0.1712 - accuracy: 0.9445 - val_loss: 0.0641 - val_accuracy: 0.9808
Epoch 7/10
280/280 [==============================] - 220s 786ms/step - loss: 0.1645 - accuracy: 0.9476 - val_loss: 0.0604 - val_accuracy: 0.9801
Epoch 8/10
280/280 [==============================] - 218s 780ms/step - loss: 0.1560 - accuracy: 0.9498 - val_loss: 0.0425 - val_accuracy: 0.9868
Epoch 9/10
280/280 [==============================] - 219s 782ms/step - loss: 0.1563 - accuracy: 0.9536 - val_loss: 0.0476 - val_accuracy: 0.9859
Epoch 10/10
280/280 [==============================] - 225s 803ms/step - loss: 0.1555 - accuracy: 0.9503 - val_loss: 0.0638 - val_accuracy: 0.9781
```

Model Evaluation:

```
plt.plot(history_v1.history["accuracy"])
plt.plot(history_v1.history['val_accuracy'])

plt.ylabel('accuracy')
plt.xlabel('nb_epoch')
plt.legend(['accuracy', 'val_accuracy'], loc='upper left')

plt.show()
```



+ Code + Markdown

```
score2 = model_1.evaluate_generator(validation_generator, nb_validation_samples // batch_size)
print('Loss: ', score2[0])
print('Accuracy: ', score2[1]*100, ' %')
```

```
Loss:  0.06373465061187744
Accuracy:  97.81249761581421  %
```

5. Transfer Learning

Model Selection:

Model development

```
[9]  base = VGG16(weights='imagenet', include_top=False, input_shape=(256, 256, 3))
     base.trainable = False
```

Model Summary:

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0

 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080

 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0

 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160

 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0

 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
```

Added Layers:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 8, 8, 512)         14714688

 flatten (Flatten)           (None, 32768)             0

 dropout (Dropout)           (None, 32768)             0

 dense (Dense)               (None, 256)               8388864

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 10)                1290

=================================================================
Total params: 23,137,738
Trainable params: 8,423,050
Non-trainable params: 14,714,688
_________________________________________________________________
```

Model Accuracy:

Adam Opt.

```
history = model.fit(
    train_generator,
    steps_per_epoch=560,
    epochs=5,
    validation_data=val_generator,
    validation_steps=140
)
```

```
Epoch 1/5
560/560 [==============================] - 181s 312ms/step - loss: 0.3464 - acc: 0.9185 - val_loss: 0.0675 - val_acc: 0.9799
Epoch 2/5
560/560 [==============================] - 174s 310ms/step - loss: 0.0453 - acc: 0.9863 - val_loss: 0.0465 - val_acc: 0.9875
Epoch 3/5
560/560 [==============================] - 173s 309ms/step - loss: 0.0320 - acc: 0.9900 - val_loss: 0.0351 - val_acc: 0.9893
Epoch 4/5
560/560 [==============================] - 171s 305ms/step - loss: 0.0372 - acc: 0.9889 - val_loss: 0.0498 - val_acc: 0.9850
Epoch 5/5
560/560 [==============================] - 172s 307ms/step - loss: 0.0395 - acc: 0.9872 - val_loss: 0.0158 - val_acc: 0.9955
```

Model Accuracy:

RMSprop Opt.

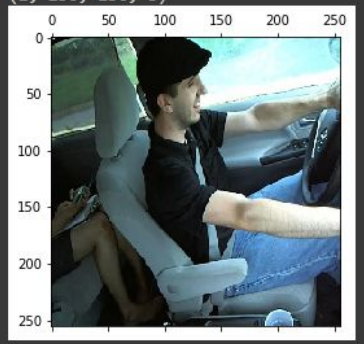
```
[ ] history = model2.fit(
        train_generator,
        steps_per_epoch=560,
        epochs=7,
        validation_data=val_generator,
        validation_steps=140
    )
```

```
Epoch 1/7
560/560 [==============================] - 173s 308ms/step - loss: 0.5192 - acc: 0.8390 - val_loss: 0.0348 - val_acc: 0.9913
Epoch 2/7
560/560 [==============================] - 173s 310ms/step - loss: 0.1117 - acc: 0.9700 - val_loss: 0.0305 - val_acc: 0.9911
Epoch 3/7
560/560 [==============================] - 173s 309ms/step - loss: 0.0809 - acc: 0.9812 - val_loss: 0.3306 - val_acc: 0.9228
Epoch 4/7
560/560 [==============================] - 171s 305ms/step - loss: 0.0575 - acc: 0.9867 - val_loss: 0.0269 - val_acc: 0.9935
Epoch 5/7
560/560 [==============================] - 175s 311ms/step - loss: 0.0630 - acc: 0.9882 - val_loss: 0.1178 - val_acc: 0.9714
Epoch 6/7
560/560 [==============================] - 173s 308ms/step - loss: 0.0430 - acc: 0.9913 - val_loss: 0.0223 - val_acc: 0.9953
Epoch 7/7
560/560 [==============================] - 176s 315ms/step - loss: 0.0436 - acc: 0.9927 - val_loss: 0.0691 - val_acc: 0.9862
```

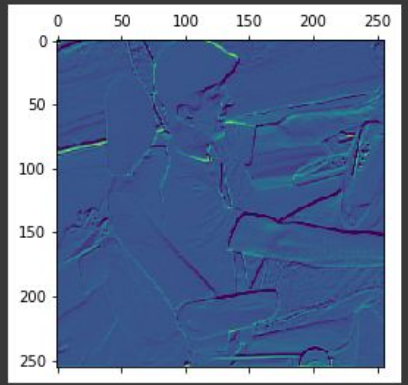
6. CNN Visualization

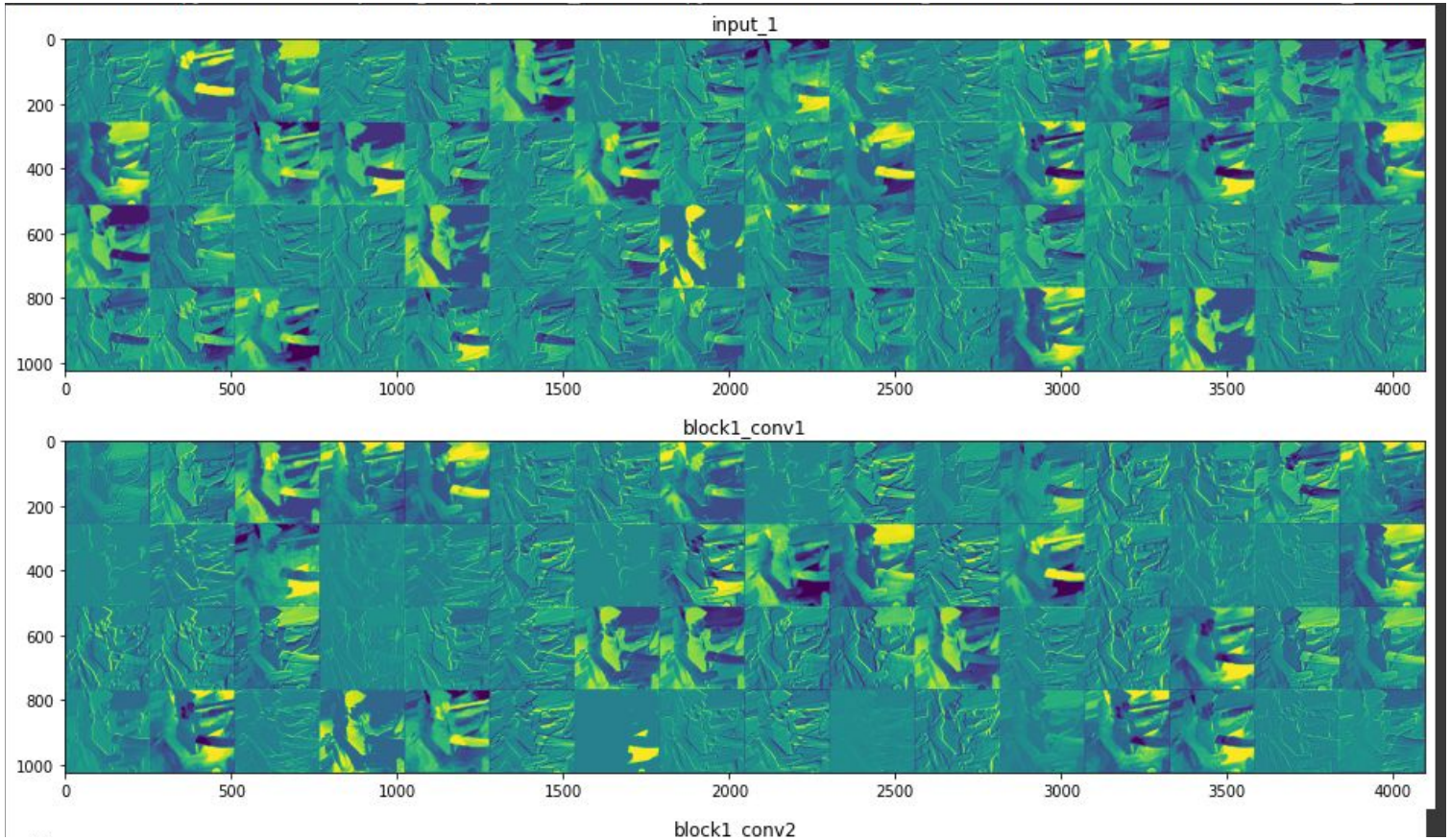
```
[ ]  img = image.load_img('/content/gdrive/MyDrive/DL/test.jpg', target_size=(256, 256))
     img_tensor = image.img_to_array(img)
     img_tensor = np.expand_dims(img_tensor, axis=0)
     img_tensor /= 255.
     print(img_tensor.shape)
     plt.matshow(img)
     plt.show()
```

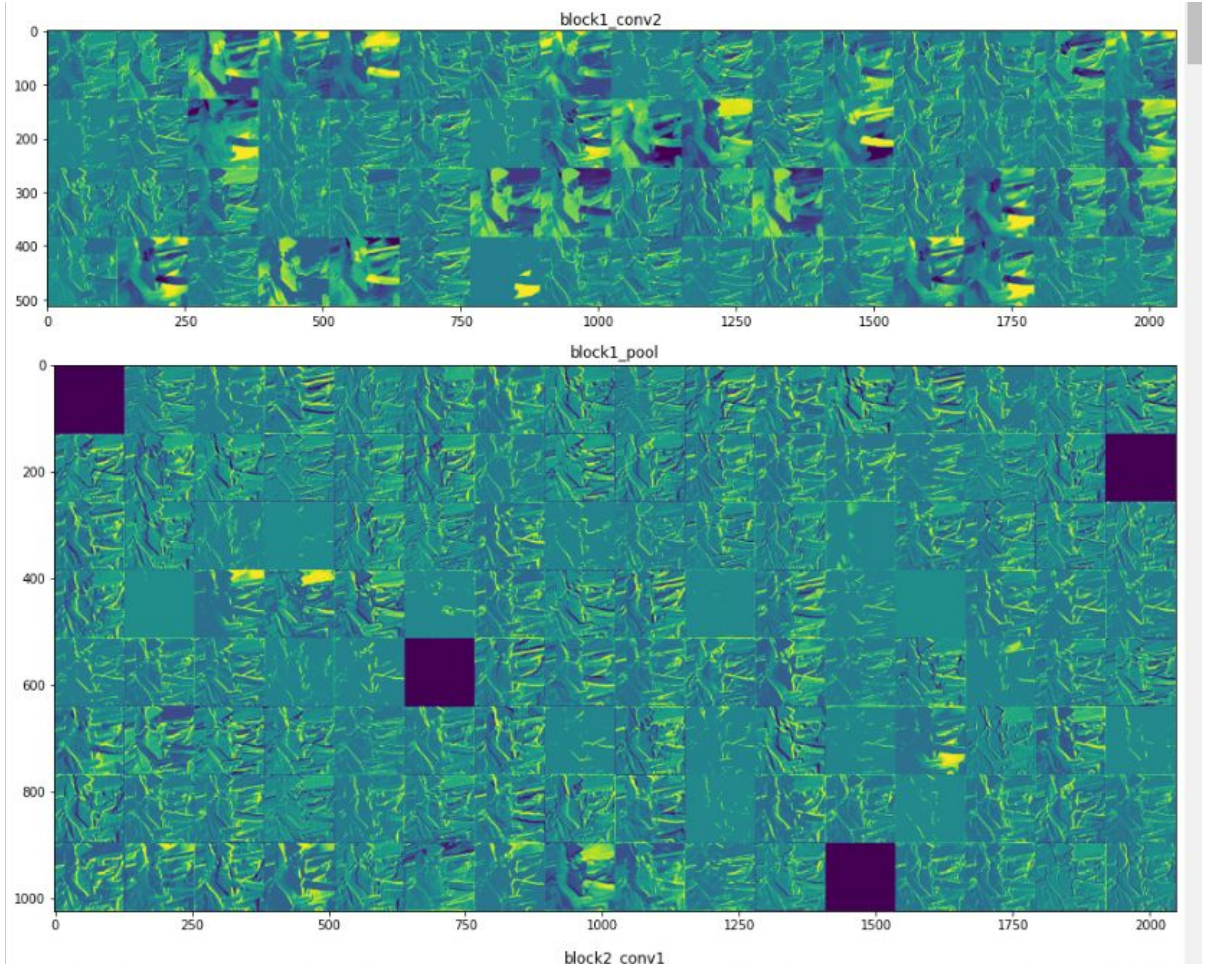
```
(1, 256, 256, 3)
```

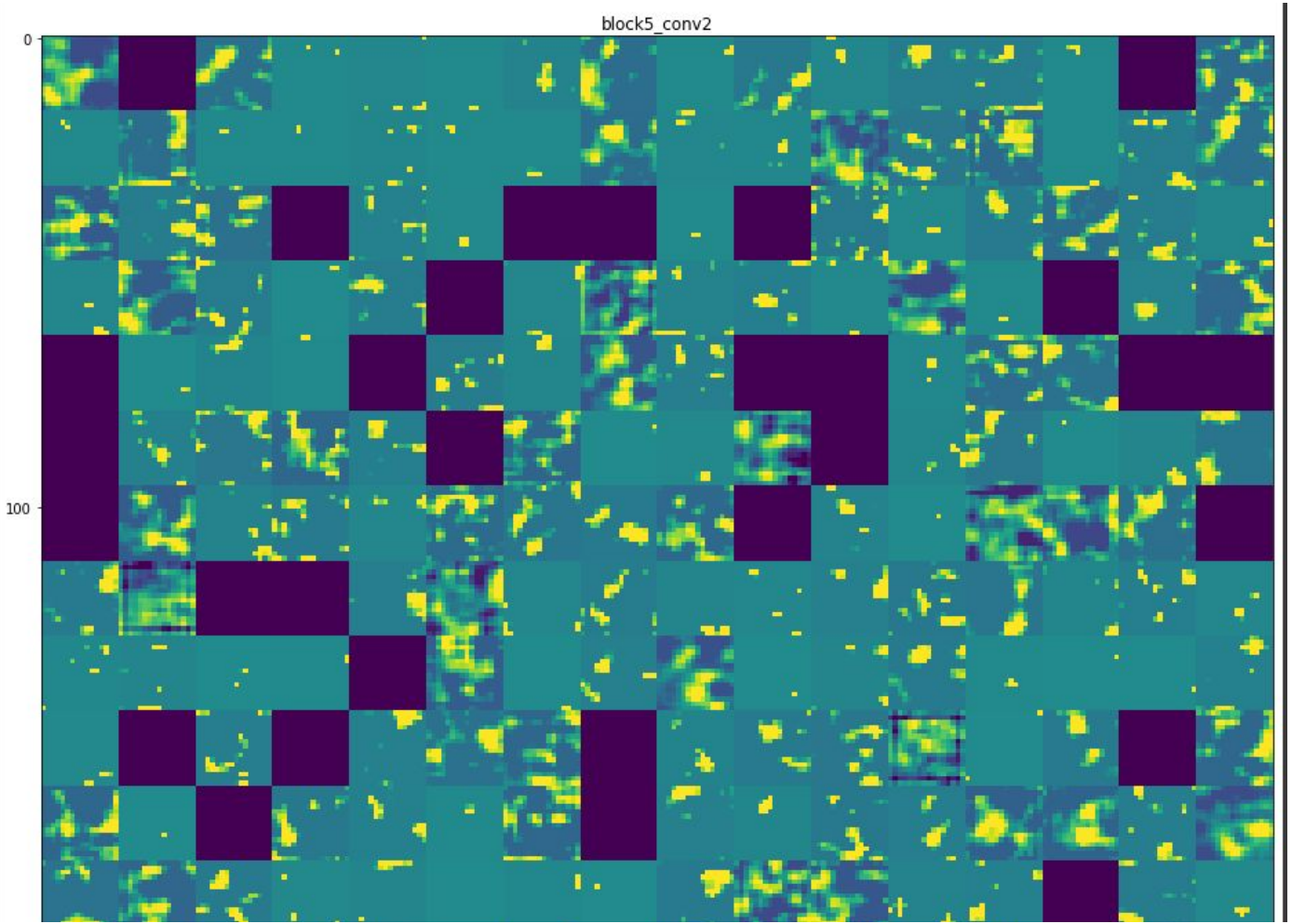



```
[ ]  activations = activation_model.predict(img_tensor)
     first_layer_activation = activations[0]
     plt.matshow(first_layer_activation[0, :, :, 3], cmap='viridis')
     plt.show()
```



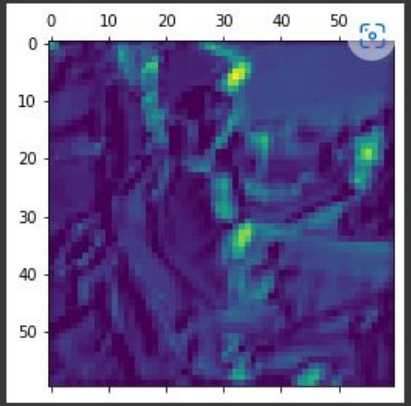

input_1
0
200
400
600
800
1000
0
500
1000
1500
2000
2500
3000
3500
4000
block1_conv1
0
200
400
600
800
1000
0
500
1000
1500
2000
2500
3000
3500
4000
block1_conv2


block1_conv2
0
100
200
300
400
500
0
250
500
750
1000
1250
1500
1750
2000
block1_pool
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750
2000
block2_conv1


block5_conv2
0
100

HeatMap

```
[86] import matplotlib.pyplot as plt
     heatmap = np.maximum(heatmap, 0)
     heatmap /= np.max(heatmap)
     plt.matshow(heatmap)
     plt.show()
```




```
[90] import matplotlib.pyplot as plt

     plt.imshow(img_tensor[0])
     plt.show()
```