Tim Miller
15.S04.SP22
Homework #1

PROBLEM 1A

Updating to 32 filters for convolution layers and 512 nodes:

```
input = keras.Input(shape=train_faces.shape[1:])
x = keras.layers.Rescaling(1./255)(input) #normalizing

# updating to 32 filters for convolution layer
# x = keras.layers.Conv2D(16, kernel_size=(2, 2), activation="relu", name="Conv_1")(x) # convolutional layer!
x = keras.layers.Conv2D(32, kernel_size=(2, 2), activation="relu", name="Conv_1")(x)
x = keras.layers.MaxPool2D()(x) # pooling layer

# updating to 32 filters for convolution layer
# x = keras.layers.Conv2D(16, kernel_size=(2, 2), activation="relu", name="Conv_2")(x) # convolutional layer!
x = keras.layers.Conv2D(32, kernel_size=(2, 2), activation="relu", name="Conv_2")(x)
x = keras.layers.MaxPool2D()(x) # pooling layer

# updating to 32 filters for convolution layer
# x = keras.layers.Conv2D(16, kernel_size=(2, 2), activation="relu", name="Conv_3")(x) # convolutional layer!
x = keras.layers.Conv2D(32, kernel_size=(2, 2), activation="relu", name="Conv_3")(x)
x = keras.layers.MaxPool2D()(x) # pooling layer
x = keras.layers.Flatten()(x)

#updating to 512 nodes for dense layer
# x = keras.layers.Dense(256, activation="relu")(x)
x = keras.layers.Dense(512, activation="relu")(x)
output = keras.layers.Dense(7, activation="softmax")(x)

model = keras.Model(input, output, name='CNN_model')
```

After these updates, we get **422,375 parameters**.

```
Model: "CNN_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 48, 48, 3)]       0

 rescaling_1 (Rescaling)     (None, 48, 48, 3)         0

 Conv_1 (Conv2D)             (None, 47, 47, 32)        416

 max_pooling2d_3 (MaxPooling  (None, 23, 23, 32)       0
 2D)

 Conv_2 (Conv2D)             (None, 22, 22, 32)        4128

 max_pooling2d_4 (MaxPooling  (None, 11, 11, 32)       0
 2D)

 Conv_3 (Conv2D)             (None, 10, 10, 32)        4128

 max_pooling2d_5 (MaxPooling  (None, 5, 5, 32)         0
 2D)

 flatten_1 (Flatten)         (None, 800)               0

 dense_2 (Dense)             (None, 512)               410112

 dense_3 (Dense)             (None, 7)                 3591

=================================================================
Total params: 422,375
Trainable params: 422,375
Non-trainable params: 0
```

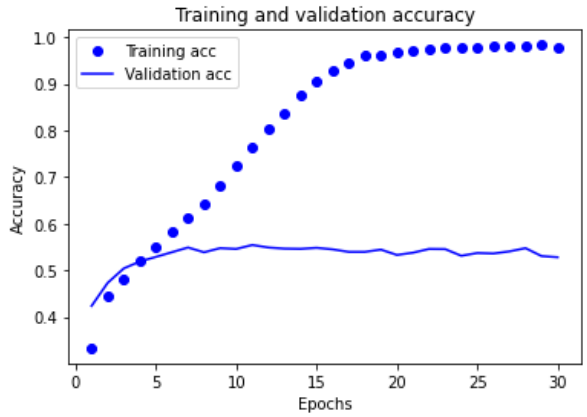
PROBLEM 1B

We find a model accuracy of ~52.51%

```
score = model.evaluate(test_faces, test_emotions)
print("Test accuracy:", score[1])
```

```
225/225 [==============================] - 1s 3ms/step - loss: 3.9358 - accuracy: 0.5251
Test accuracy: 0.5250766277313232
```

PROBLEM 1C



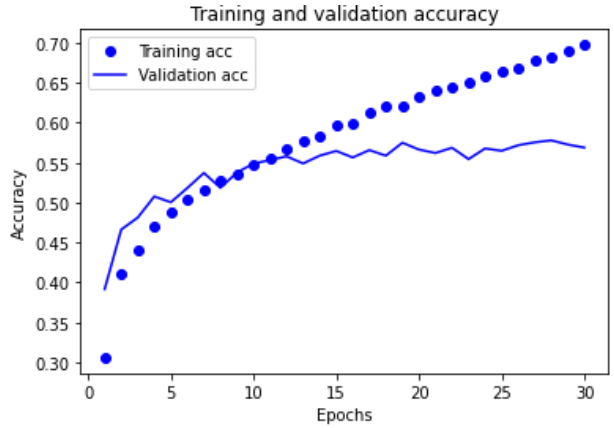
We can see the training accuracy approaches 100% as the number of epochs increases. However, validation accuracy reaches an asymptote of ~55%. Likely the model is overfitting on noise in the training set, which is why the accuracy is not replicated in the validation set.

PROBLEM 2A

Deep learning models require a lot of data to be trained. Data augmentation is useful because it helps us create a larger data set to train our models without necessarily having to go out and find additional images of faces (or whatever we are training our models). In this case we flipped the image and zoomed in / out providing additional data for training. But we did it with the data we already had.

PROBLEM 2B

Below is a graph for accuracy vs. epochs for the CNN augmented model.



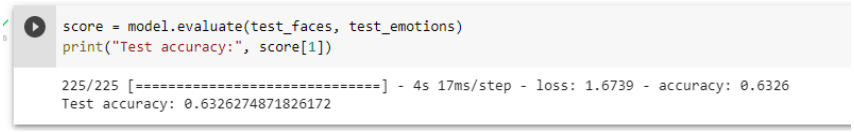
PROBLEM 2C

Compared to problem 1C, we see two things

- The training accuracy in 2B does not increase in accuracy as rapidly as 1C
- The difference between the training and validation accuracy lines is small

The likely explanation for this that data augmentation helps reduce overfitting. As we said in 1C, the reason the training accuracy approaches 100% is because the model is likely overfitting. That is why it does not perform as well on the validation set. But in 2B, overfitting is reduced, so the model does not increase in accuracy as rapidly. Then accordingly the accuracy for the training and validation models is closer.

PROBLEM 3A



The accuracy for this model is ~63.3%.

PROBLEM 3B

Transfer learning is useful here because we have access to VGG19 which has been trained on the >1million images in ImageNet. Given the size of this dataset, and the fact it is much bigger than the set of images we are analyzing, we are comfortable that VGG19 is not at risk of overfitting. Furthermore, VGG19 has been trained for image recognition and feature classification – like the end use case we are running with our sample data.

Transfer learning can be necessary when the you have a small amount of data in the sample you are testing. Given deep learning models require a large amount of training data, your results will likely not be great with small sample of data you have. Instead, you can leverage the work already done with another model and tweak the weights using your sample data. In many cases, this allows you to achieve impressively high accuracy levels even with a small amount of data. However, as stated above we need to make sure the model used for transfer learning has much more data than what we are testing to prevent overfitting.

PROBLEM 3C

Confusion matrix

```
df = pd.DataFrame({'Predictions': predictions, 'Actuals': actuals})
pd.crosstab(df.Predictions, df.Actuals)
```

Actuals	angry	disgust	fear	happy	neutral	sad	surprise
Predictions							
angry	473	19	72	31	60	126	18
disgust	4	62	0	0	0	3	0
fear	135	11	516	36	63	149	71
happy	35	0	25	1444	62	60	42
neutral	170	9	197	187	915	403	33
sad	108	9	133	33	103	476	12
surprise	33	1	81	43	30	30	655

If we looked at the true positivity, we find the following

Emotion	True Positivity Value
angry	0.49
disgust	0.56
fear	0.50
happy	0.81
neutral	0.74
sad	0.38
surprise	0.79

Clearly, the values for negative emotions like anger, disgust, fear, and sadness are much lower than positive or neutral emotions. The simplest answer would be to add in additional datapoints (images) for angry, disgust, fear, and, sad emotions to train the model better. Or we could find a transfer learning model that is specifically good at looking at negative emotions to add to the transfer learning model we already built.

BONUS

I successfully loaded my image into the notebook but encountered an error when I ran the prediction. I think this has something to do with the way my image is formatted as I enter it into the prediction model.

▾ BONUS Question #4 - Importing my own image

```
[54] from PIL import Image
     from numpy import asarray
```

```
[81] # load image
     load_img_rz = np.array(Image.open('tatte.jpg').resize((48,48)))
     print("After resizing:",load_img_rz.shape)

     After resizing: (48, 48, 3)
```

```
[ ] # predict emotion
     model.predict(test_faces)
```