CS256 HW 3

Experiments and Write-up

Experiment 1:

Hypothesis: Increase in the training set size and number of weights will have a positive effect on the accuracy of the model.

Experimenting:

Model is trained with data sets of different sizes of 10,000, 50,000, 100,000, and 200,000, and the accuracy is measured.

Constants: Mini batch size= 128,

Epoch = 30,

number of filters = 32,

Kernel size = 5 => weights = 832

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 60, 60, 32)        832
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 30, 30, 32)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 30, 30, 32)        128
_________________________________________________________________
flatten_5 (Flatten)          (None, 28800)             0
_________________________________________________________________
dense_10 (Dense)             (None, 256)               7373056
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_11 (Dense)             (None, 4)                 1028
=================================================================
Total params: 7,375,044
Trainable params: 7,374,980
Non-trainable params: 64
```

Data size	Accuracy
5000	Epoch 30/30 40/40 [==============================] - 1s 22ms/step - loss: 0.0121 - categorical_accuracy: 0.9796 - val_loss: 0.0083 - val_categorical_accuracy: 0.9870
10000	Epoch 30/30 79/79 [==============================] - 2s 21ms/step - loss: 0.0065 - categorical_accuracy: 0.9886 - val_loss: 0.0045 - val_categorical_accuracy: 0.9880

50000	Epoch 30/30 391/391 [==============================] - 8s 20ms/step - loss: 0.0015 - categorical_accuracy: 0.9983 - val_loss: 4.1605e-04 - val_categorical_accuracy: 1.0000
100,000	Epoch 30/30 782/782 [==============================] - 16s 21ms/step - loss: 5.7597e-04 - categorical_accuracy: 0.9996 - val_loss: 5.6294e-04 - val_categorical_accuracy: 0.9990

Observation: As the data set size increases, the accuracy is also increasing.
Conclusion: As the dataset size increases, we are covering more combinations in each iteration thereby improving the accuracy of the model.

Now, for the second part, we will keep the data set constant and change the weights to determine the effect of weights on the accuracy of the model.

Constants

Mini batch size= 128,

Epoch = 30,

Dataset = 10000

Case 1 :

Filters = 16

Kernel size = 2

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 63, 63, 16)        80
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 31, 31, 16)        0
_________________________________________________________________
batch_normalization_6 (Batch (None, 31, 31, 16)        64
_________________________________________________________________
flatten_9 (Flatten)          (None, 15376)             0
_________________________________________________________________
dense_18 (Dense)             (None, 256)               3936512
_________________________________________________________________
dropout_9 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_19 (Dense)             (None, 4)                 1028
=================================================================
Total params: 3,937,684
Trainable params: 3,937,652
Non-trainable params: 32
```

Case 2 :

Filters = 32

Kernel size = 2

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_10 (Conv2D)           (None, 63, 63, 32)        160       
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 31, 31, 32)        0         
_________________________________________________________________
batch_normalization_7 (Batch (None, 31, 31, 32)        128       
_________________________________________________________________
flatten_10 (Flatten)         (None, 30752)             0         
_________________________________________________________________
dense_20 (Dense)             (None, 256)               7872768   
_________________________________________________________________
dropout_10 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_21 (Dense)             (None, 4)                 1028      
=================================================================
Total params: 7,874,084
Trainable params: 7,874,020
Non-trainable params: 64
```

Case 3 :

Filters = 32

Kernel size = 4

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_11 (Conv2D)           (None, 61, 61, 32)        544
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 30, 30, 32)        0
_________________________________________________________________
batch_normalization_8 (Batch (None, 30, 30, 32)        128
_________________________________________________________________
flatten_11 (Flatten)         (None, 28800)             0
_________________________________________________________________
dense_22 (Dense)             (None, 256)               7373056
_________________________________________________________________
dropout_11 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_23 (Dense)             (None, 4)                 1028
=================================================================
Total params: 7,374,756
Trainable params: 7,374,692
Non-trainable params: 64
```

Case 4 :

Filters = 32

Kernel size = 5

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 60, 60, 32)        832
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 30, 30, 32)        0
_________________________________________________________________
batch_normalization_10 (Batc (None, 30, 30, 32)        128
_________________________________________________________________
flatten_13 (Flatten)         (None, 28800)             0
_________________________________________________________________
dense_26 (Dense)             (None, 256)               7373056
_________________________________________________________________
dropout_13 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_27 (Dense)             (None, 4)                 1028
=================================================================
Total params: 7,375,044
Trainable params: 7,374,980
Non-trainable params: 64
```

Case 5 :

Filters = 64

Kernel size = 4

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 61, 61, 64)        1088
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 30, 30, 64)        0
_________________________________________________________________
batch_normalization_9 (Batch (None, 30, 30, 64)        256
_________________________________________________________________
flatten_12 (Flatten)         (None, 57600)             0
_________________________________________________________________
dense_24 (Dense)             (None, 256)               14745856
_________________________________________________________________
dropout_12 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_25 (Dense)             (None, 4)                 1028
=================================================================
Total params: 14,748,228
Trainable params: 14,748,100
Non-trainable params: 128
```

Cases	**Accuracy**
Case 1	Epoch 30/30 79/79 [==============================] - 1s 15mF/step - loss: 0.0136 - categorical_accuracy: 0.9742 - val_loss: 0.0105 - val_categorical_accuracy: 0.9770
Case 2	Epoch 30/30 79/79 [==============================] - 2s 24ms/step - loss: 0.0076 - categorical_accuracy: 0.9877 - val_loss: 0.0051 - val_categorical_accuracy: 0.9890
Case 3	Epoch 30/30 79/79 [==============================] - 2s 21ms/step - loss: 0.0074 - categorical_accuracy: 0.9878 - val_loss: 0.0072 - val_categorical_accuracy: 0.9820

Case 4	Epoch 30/30 79/79 [==============================] - 2s 22ms/step - loss: 0.0075 - categorical_accuracy: 0.9872 - val_loss: 0.0040 - val_categorical_accuracy: 0.9940
Case 5	Epoch 30/30 79/79 [==============================] - 3s 37ms/step - loss: 0.0051 - categorical_accuracy: 0.9916 - val_loss: 0.0026 - val_categorical_accuracy: 0.996

Observation

From the above table, we can observe that as the weights increase, the accuracy is also gradually increasing.

Conclusion

This may be the case as we are increasing the weights, we might end up covering more combinations in each iteration thereby improving the accuracy.

Experiment 2:

Hypothesis: The well-chosen samples should give a better performance compared to the random samples.

Experimenting: We will run experiments on a set of Random samples and then on a set of well-chosen samples and compare the accuracy.

Constants : Mini batch size= 128,
Epoch = 30,
number of filters = 32,
Kernel size = 5 => weights = 832

Training:

Random samples	**Well Chosen**
8/8 [==============================] - 0s 13ms/step - loss: 0.0088 - categorical_accuracy: 0.9820	8/8 [==============================] - 0s 13ms/step - loss: 0.0063 - categorical_accuracy: 0.9890

total number of items tested on 1000	total number of items tested on 1000

Observation: well-chosen should have better accuracy.

Testing:

Random samples	Well Chosen
79/79 [==============================] - 1s 11ms/step - loss: 0.0100 - categorical_accuracy: 0.9840 total number of items tested on 10000	79/79 [==============================] - 1s 12ms/step - loss: 0.0115 - categorical_accuracy: 0.9758 total number of items tested on 10000

Observation: We see a minor difference in accuracy.

Conclusion: This difference may be due to the difference in the combination of images. However, Ideally, there should be no difference in the accuracy in testing. But the Well Chosen data performs better for training.

Experiment 3 :

Hypothesis: Performing cross-validation gives a better performance compared to the separate test data.

Experimenting: We will run experiments to compare the accuracy of cross-validation and use separate test data.

Constants : Mini batch size= 128,
Epoch = 30,
number of filters = 32,
Kernel size = 5 => weights = 832

Cases	Separate Test Data	Cross Validation
Data size = 5000	**Training :** Epoch 30/30	**Testing :** 8/8

	40/40 [============================] - 1s 22ms/step - loss: 0.0121 - categorical_accuracy: 0.9796 - val_loss: 0.0083 - val_categorical_accuracy: 0.9870 **Testing :** 8/8 [============================] - 0s 13ms/step - loss: 0.0134 - categorical_accuracy: 0.9700 **total number of items tested on 1000**	[============================] - 0s 10ms/step - loss: 0.0189 - categorical_accuracy: 0.9600 total number of items tested on 1000

Observation: Accuracy of the model decreased in Cross-Validation testing.

Conclusion: The accuracy decreased in cross-validation because it is avoiding the overfitting of data by taking the average of data over K splits. Therefore, cross fitting is better for training the model.