Assignment 2: Convolution – Summary

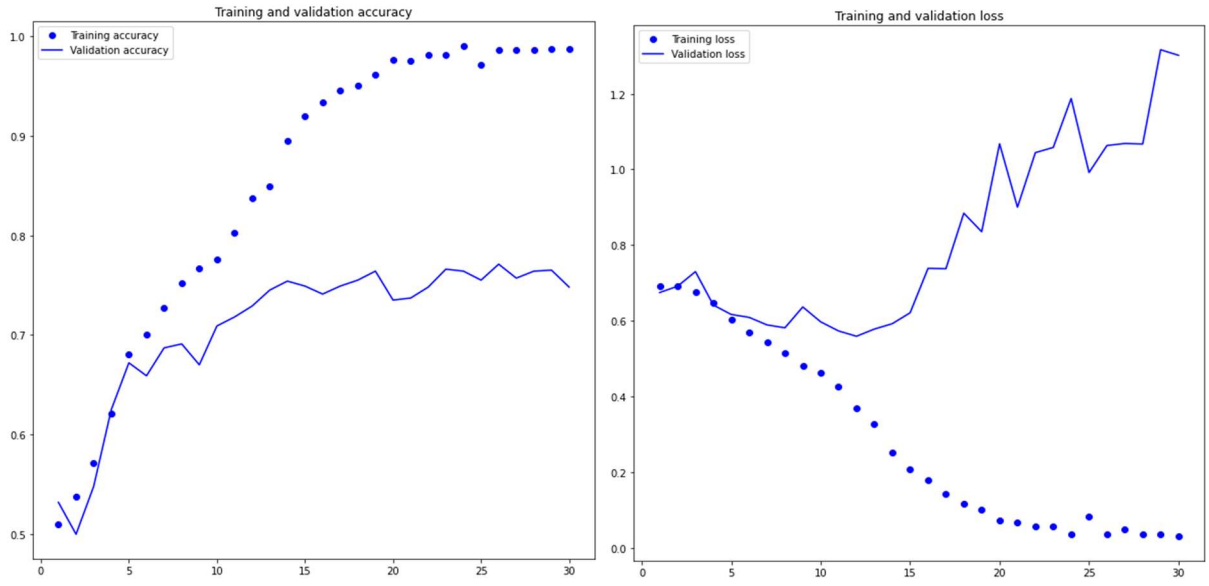
1. ***Started initially with a training sample of 1000, a validation sample of 500, and a test sample of 500 (like in the text).***

Data Processing -
Before being input into the model, the data is converted into preprocessed floatin g-point tensors as
the data is in JPEG format, the preprocessing stages are as follows:

1.Read the pictures 2. convert the JPEG content in to RGB grid of pixels 3. convert the RGB grid of pixels in to floating point tensors. 4. Resize them 5. Make them into batches.

Displaying curves of loss and accuracy during training

```
63/63 [==============================] - 5s 82ms/step - loss: 0.0369 - accuracy: 0.9900 - val_loss: 1.1873 - val_accuracy: 0.7640
Epoch 25/30
63/63 [==============================] - 5s 82ms/step - loss: 0.0833 - accuracy: 0.9710 - val_loss: 0.9918 - val_accuracy: 0.7550
Epoch 26/30
63/63 [==============================] - 5s 81ms/step - loss: 0.0376 - accuracy: 0.9860 - val_loss: 1.0632 - val_accuracy: 0.7710
Epoch 27/30
63/63 [==============================] - 5s 82ms/step - loss: 0.0503 - accuracy: 0.9855 - val_loss: 1.0686 - val_accuracy: 0.7570
Epoch 28/30
63/63 [==============================] - 5s 80ms/step - loss: 0.0366 - accuracy: 0.9860 - val_loss: 1.0671 - val_accuracy: 0.7640
Epoch 29/30
63/63 [==============================] - 5s 81ms/step - loss: 0.0362 - accuracy: 0.9870 - val_loss: 1.3160 - val_accuracy: 0.7650
Epoch 30/30
63/63 [==============================] - 5s 83ms/step - loss: 0.0314 - accuracy: 0.9870 - val_loss: 1.3014 - val_accuracy: 0.7480
```



```
[ ]  test_model = keras.models.load_model("convnet_from_scratch.keras")
     test_loss, test_acc = test_model.evaluate(test_dataset)
     print(f"Test accuracy: {test_acc:.3f}")

     32/32 [==============================] - 2s 40ms/step - loss: 0.5797 - accuracy: 0.7230
     Test accuracy: 0.723
```

Observation, from the above result we can conclude: These two graphs are characterized by overfitting as the model is training well on the training set (98.70%) while validation (74.80%) is not improving. Evaluated the model on the test set (72.30%). Model accuracy is right in line with the validation accuracy of model prediction. More epochs would increase model performance since it hasn't overfit yet. We can use these three techniques namely, 1. Data augmentation, 2. Regularization, 3. Dropout to improve our validation and test accuracy and prevent overfitting.

2. ***Increased training sample size and applied data augmentation & dropout technique to reduce overfitting.***

Increasing the training dataset to 1500 per assignment and Using Data augmentation.

What exactly do we do in Data Augmentation-

In the simplest sense, we tend to flip, rotate, scale, crop, translate (moving image along x and y axis), Gaussian Noise (way of distorting high frequency by adding some noise to them). For our Neural Network, we will only use flipping, rotation and zooming.

Here, there are samples of 9 images that have been flipped, zoomed, and rotated.

After Defining a new convnet that includes image augmentation and dropout we get below results in terms of accuracy.

```
Epoch 44/50
63/63 [==============================] - 7s 103ms/step - loss: 0.3139 - accuracy: 0.8650 - val_loss: 0.4366 - val_accuracy: 0.8170
Epoch 45/50
63/63 [==============================] - 8s 116ms/step - loss: 0.2930 - accuracy: 0.8745 - val_loss: 0.4319 - val_accuracy: 0.8250
Epoch 46/50
63/63 [==============================] - 7s 107ms/step - loss: 0.2989 - accuracy: 0.8750 - val_loss: 0.4466 - val_accuracy: 0.8200
Epoch 47/50
63/63 [==============================] - 7s 106ms/step - loss: 0.2786 - accuracy: 0.8775 - val_loss: 0.4611 - val_accuracy: 0.8200
Epoch 48/50
63/63 [==============================] - 7s 103ms/step - loss: 0.2870 - accuracy: 0.8755 - val_loss: 0.4552 - val_accuracy: 0.8310
Epoch 49/50
63/63 [==============================] - 7s 101ms/step - loss: 0.2911 - accuracy: 0.8765 - val_loss: 0.4351 - val_accuracy: 0.8250
Epoch 50/50
63/63 [==============================] - 7s 105ms/step - loss: 0.2856 - accuracy: 0.8810 - val_loss: 0.4258 - val_accuracy: 0.8260
```

Evaluating the model on the test set

```
[ ]  test_model = keras.models.load_model(
         "convnet_from_scratch_with_augmentation.keras")
     test_loss, test_acc = test_model.evaluate(test_dataset)
     print(f"Test accuracy: {test_acc:.3f}")

     32/32 [==============================] - 2s 40ms/step - loss: 0.4579 - accuracy: 0.8140
     Test accuracy: 0.814
```

Test Accurcay noted - 81.4% Training Accuracy - 87.55% Validation Accuracy - 83.10%. As we can see that our test accuracy has already improved alot by using data augmentation and dropout and increasing the training sample size. However, we do have to train the model for more epochs than usual.

Hence, we can say that by using Data Augmentation, dropout and Regularization we can somewhat mitigated the effects of Overfitting.

By Increasing the training sample size with augmentation helped in getting better accuracy.

3. *Increasing the training sample size to 2000 while maintaining the same va lidation and test sets as before 500 samples*

```
Epoch 44/50
63/63 [==============================] - 7s 103ms/step - loss: 0.6932 - accuracy: 0.4850 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 45/50
63/63 [==============================] - 7s 101ms/step - loss: 0.6932 - accuracy: 0.4645 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 46/50
63/63 [==============================] - 7s 100ms/step - loss: 0.6932 - accuracy: 0.4780 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 47/50
63/63 [==============================] - 7s 102ms/step - loss: 0.6932 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 48/50
63/63 [==============================] - 7s 99ms/step - loss: 0.6932 - accuracy: 0.5000 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 49/50
63/63 [==============================] - 6s 98ms/step - loss: 0.6932 - accuracy: 0.4840 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 50/50
63/63 [==============================] - 6s 95ms/step - loss: 0.6932 - accuracy: 0.4970 - val_loss: 0.6931 - val_accuracy: 0.5000
```

Evaluating the model on test set

```
[ ] test_model = keras.models.load_model(
        "convnet_from_scratch_with_augmentation1.keras")
    test_loss, test_acc = test_model.evaluate(test_dataset)
    print(f"Test accuracy: {test_acc:.3f}")

    32/32 [==============================] - 2s 39ms/step - loss: 0.6922 - accuracy: 0.5110
    Test accuracy: 0.511
```

I began with the training a sample convnet on the 1,000 training samples without any optimization to which resulted in the Test accuracy was around 72.30% and the Overfitting was recognized as the main problem. After applying data augmentation and other optimization strategies dropout .Then I raised the model's test accuracy to 81% and validation accuracy raised to 83% from 74%.

After that I looked for the best training sample to improve accuracy. The best approach to avoid overfitting have been discovered by manipulating the training sample and using the optimization techniques

a. Getting more training samples not always practical. Our test accuracy has reduced to 51% by increasing training sample.

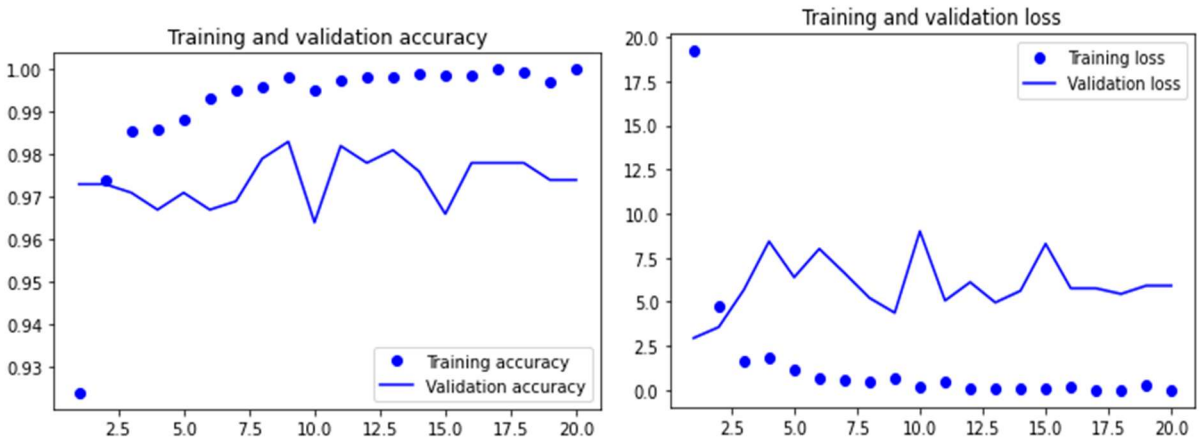
b. Reducing the capacity of the work: Overfitting is significantly reduced when the model's size is reduced, i.e., the number of learnable parameters in the model, which is effectively the number of layers and the number of units in layers.

c. Adding weight regularization: Limiting the complexity of a network by restricting the weights to accept only tiny values, which helps to regularize the distribution of the weight values and so prevents or minimizes overfitting.

d. Adding dropout- Overfitting is reduced by zeroing out several the layer's output characteristics during training. The percentage of features that are zeroed out is known as the dropout rate.

4. ***We will use a big convnet trained on the ImageNet dataset in this scenario (1.4 million labeled images and 1,000 different classes). We will use the VGG16 architecture, although there are a variety of other architectures to choose from, including VGG, ResNet, Inception, Xception, and so on.***

Feature Extraction with a pretrained model-Feature extraction is the process of extracting important features from new samples using the representations acquired by a previously trained model (in our instance, ImageNet). These characteristics are then fed into a new classifier that has been trained from the ground up.



Instantiating and freezing the VGG16 convolutional base

```
63/63 [==============================] - 13s 207ms/step - loss: 0.6470 - accuracy: 0.9880 - val_loss: 1.5173 - val_accuracy: 0.9840
Epoch 44/50
63/63 [==============================] - 13s 206ms/step - loss: 0.4980 - accuracy: 0.9900 - val_loss: 1.6341 - val_accuracy: 0.9860
Epoch 45/50
63/63 [==============================] - 13s 204ms/step - loss: 0.3625 - accuracy: 0.9880 - val_loss: 1.5096 - val_accuracy: 0.9810
Epoch 46/50
63/63 [==============================] - 13s 205ms/step - loss: 0.5690 - accuracy: 0.9890 - val_loss: 1.8508 - val_accuracy: 0.9770
Epoch 47/50
63/63 [==============================] - 13s 206ms/step - loss: 0.4992 - accuracy: 0.9900 - val_loss: 1.9741 - val_accuracy: 0.9790
Epoch 48/50
63/63 [==============================] - 13s 207ms/step - loss: 0.9090 - accuracy: 0.9820 - val_loss: 1.5734 - val_accuracy: 0.9820
Epoch 49/50
63/63 [==============================] - 13s 206ms/step - loss: 0.6179 - accuracy: 0.9920 - val_loss: 1.5265 - val_accuracy: 0.9820
Epoch 50/50
63/63 [==============================] - 13s 205ms/step - loss: 0.8396 - accuracy: 0.9835 - val_loss: 1.7538 - val_accuracy: 0.9730
```

Evaluating the model on the test set

```
[ ] test_model = keras.models.load_model(
        "feature_extraction_with_data_augmentation.keras")
    test_loss, test_acc = test_model.evaluate(test_dataset)
    print(f"Test accuracy: {test_acc:.3f}")

    32/32 [==============================] - 4s 112ms/step - loss: 1.6130 - accuracy: 0.9800
    Test accuracy: 0.980
```

A pretrained VGG16 model with Fine-tuning

Finetuning involves unfreezing a few of the top layers of a frozen model base use for feature extraction and training both the newly added element of the model (in this case, the fully connected classifier) and these top layers at the same time.

Finetuning is the process of slightly adjusting the more abstract representations o f the model that are being reused to make them more relevant to the task at hand

```
63/63 [==============================] - 14s 222ms/step - loss: 0.0860 - accuracy: 0.9950 - val_loss: 1.2401 - val_accuracy: 0.9810
Epoch 24/30
63/63 [==============================] - 14s 224ms/step - loss: 0.0863 - accuracy: 0.9975 - val_loss: 1.9418 - val_accuracy: 0.9730
Epoch 25/30
63/63 [==============================] - 14s 221ms/step - loss: 0.0837 - accuracy: 0.9950 - val_loss: 1.0830 - val_accuracy: 0.9820
Epoch 26/30
63/63 [==============================] - 14s 222ms/step - loss: 0.0806 - accuracy: 0.9970 - val_loss: 1.3669 - val_accuracy: 0.9750
Epoch 27/30
63/63 [==============================] - 14s 218ms/step - loss: 0.0557 - accuracy: 0.9960 - val_loss: 1.1413 - val_accuracy: 0.9810
Epoch 28/30
63/63 [==============================] - 14s 218ms/step - loss: 0.1279 - accuracy: 0.9970 - val_loss: 1.0964 - val_accuracy: 0.9820
Epoch 29/30
63/63 [==============================] - 14s 218ms/step - loss: 0.0135 - accuracy: 0.9990 - val_loss: 1.0566 - val_accuracy: 0.9800
Epoch 30/30
63/63 [==============================] - 15s 234ms/step - loss: 0.1314 - accuracy: 0.9940 - val_loss: 1.1709 - val_accuracy: 0.9830
```

```
] model = keras.models.load_model("fine_tuning.keras")
  test_loss, test_acc = model.evaluate(test_dataset)
  print(f"Test accuracy: {test_acc:.3f}")

  32/32 [==============================] - 5s 116ms/step - loss: 1.2238 - accuracy: 0.9790
  Test accuracy: 0.979
```

Here we can clearly see the changes in validation accuracy from the previous mod el. It's almost 0.98,

which is close to the testing accuracy of 0.9940. So, we can conclude that it's not overfitting. Test accuracy noted as 0.979 which is in line with validation accuracy.

In orderto finetune a pretrained model, we need to adjust the learning rate of the model. The learning rate has an impact on finding local minimums and maximum s. Large adjustments to the learning rate will have drastic impacts on the model.