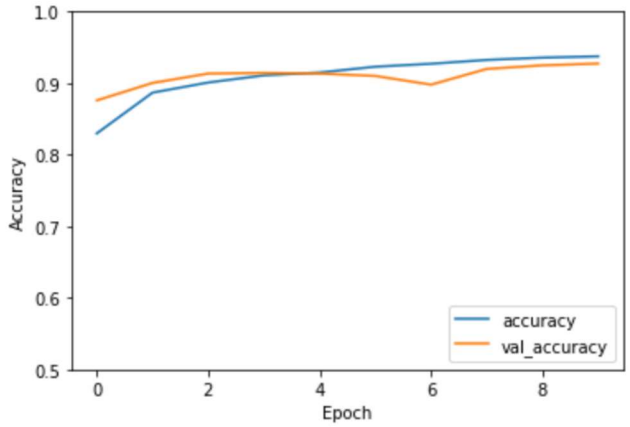
ECE 528 Homework Assignment 2

Network Architecture Search, Hyperparameter Tuning, and Transfer Learning

Q1. A) In this section, autokeras has been installed but not used for the classification of the fashion mnist dataset. The model built in this section will be much similar to the model used in the q1.ipynb of the Homework Assignment 1. The test accuracy achieved in the classification of the fashion mnist dataset is 92.74%.

```
313/313 - 1s - loss: 0.2058 - accuracy: 0.9274
```

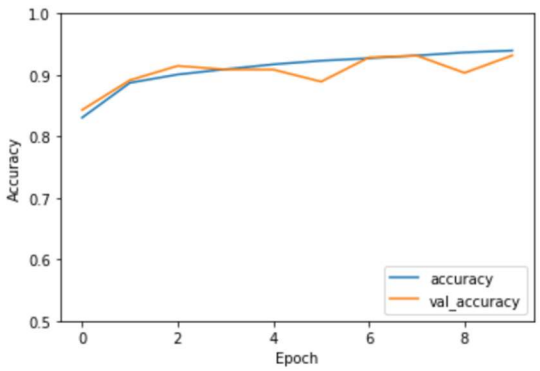



```
In [14]: print('\nTest accuracy:', test_acc)
```

```
Test accuracy: 0.9273999929428101
```

Q1. B) In this section, autokeras has been used for the classification of the fashion mnist dataset. For comparison purpose, the model has been trained without autokeras and with autokeras. From which, it has been found that image classification using autokeras gives less test accuracy 91.84% compared to that of the image classification without autokeras on the fashion mnist dataset where the test accuracy is 93.18% in this section.

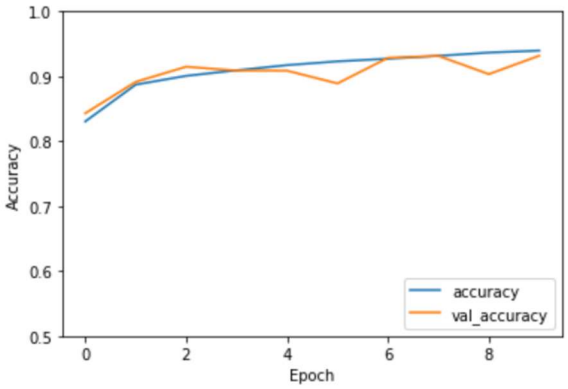
```
313/313 - 1s - loss: 0.2346 - accuracy: 0.9184
```




```
In [16]: print('\nTest accuracy:', test_acc)
```

```
Test accuracy: 0.91839998960495
```

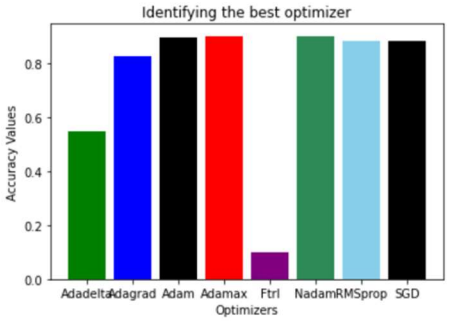
```
313/313 - 1s - loss: 0.1941 - accuracy: 0.9318
```




```
In [12]: print('\nTest accuracy:', test_acc)

Test accuracy: 0.9318000078201294
```

Q2. A) The fashion mnist dataset and the model from Q1 has been used here without autokeras module, and hparams plugin from the tensorboard module has been imported to proceed further. Hparams has been used on all 8 optimizers to explore which optimizers gives the best accuracy. On doing so, it's been found that Nadam gives the best accuracy among the eight optimizers.

```
In [10]: print(accuracy_values)
         print(optimizers)

[0.5501999855041504, 0.8282999992370605, 0.8978000283241272, 0.8996000289916992, 0.10000000149011612, 0.9024999737739563, 0.
8859000205993652, 0.8819000124931335]
['Adadelta', 'Adagrad', 'Adam', 'Adamax', 'Ftrl', 'Nadam', 'RMSprop', 'SGD']

In [11]: import matplotlib.pyplot as plt
         plt.bar(optimizers, accuracy_values, color=['green', 'blue', 'black', 'red','purple', 'seagreen', 'skyblue', 'black'])
         plt.xlabel('Optimizers')
         plt.ylabel('Accuracy Values')
         plt.title('Identifying the best optimizer')
         plt.show()
```



Q2. B) The best optimizer function from Q2. A) i.e., Nadam has been used in the compiling the model that's been built to explore the use of hparams on 81 different combinations of the activation functions 'relu', 'selu', 'elu'. On training the fashion mnist dataset for all the 81 combinations, it's found that the combination no: 65 (selu, relu, elu, relu) gives the test accuracy 93.41% among all.

61	combination62	selu	elu	selu	relu	0.9285
62	combination63	selu	elu	selu	selu	0.9301
63	combination64	selu	relu	elu	elu	0.9264
64	combination65	selu	relu	elu	relu	0.9341
65	combination66	selu	relu	elu	selu	0.9260
66	combination67	selu	relu	relu	elu	0.9291

```
In [12]: print("The best model combination is displayed as follows:\n")
         df.iloc[max_value,:]
```

```
The best model combination is displayed as follows:
```

```
Out[12]: combination_no:    combination65
         Activation1                 selu
         Activation2                 relu
         Activation3                  elu
         Activation4                 relu
         Accuracy                  0.9341
```

Q2. c) The best optimizer function from Q2. A) i.e., Nadam has been used in the compiling the model that's been built to explore the use of hparams on 81 different combinations of the number of units in the dense layers [32, 64, 128]. On training the fashion mnist dataset for all the 81 combinations, it's found that the combination no: 51 (64, 128, 64, 128) gives the test accuracy 93.89% among all.

46	combination47	64	128	32	64	0.9238
47	combination48	64	128	32	128	0.9282
48	combination49	64	128	64	32	0.9281
49	combination50	64	128	64	64	0.9283
50	combination51	64	128	64	128	0.9389
51	combination52	64	128	128	32	0.9277
52	combination53	64	128	128	64	0.9298
53	combination54	64	128	128	128	0.9306
54	combination55	128	32	32	32	0.9240
55	combination56	128	32	32	64	0.9316

```
In [11]: print("The best model combination is displayed as follows:\n")
         df.iloc[max_value,:]
```

```
The best model combination is displayed as follows:
```

```
Out[11]: combination_no:    combination51
         Num_of_units_1                64
         Num_of_units_2               128
         Num_of_units_3                64
         Num_of_units_4               128
         Accuracy                  0.9389
```

Q3. A) In this section, the pre-trained MobileNetV2 has been used for the classification of images in the Oxford flowers 102 dataset. In which, the test accuracy before tuning is 67.52% and the test accuracy after tuning is 76.1% where RMSprop optimizer function has been used with a much lower learning rate 0.0001 for compiling the model.

In [ ]:
```
loss, accuracy = model.evaluate(train_batches)
print('\nTest accuracy before tuning:', accuracy)
```

```
13/13 [==============================] - 2s 36ms/step - loss: 2.3889 - accuracy: 0.6752

Test accuracy before tuning: 0.6752451062202454
```

In [ ]:
```
loss, accuracy = model.evaluate(train_batches)
print('\nTest accuracy after tuning :', accuracy)
```

```
13/13 [==============================] - 2s 36ms/step - loss: 1.1045 - accuracy: 0.7610

Test accuracy after tuning : 0.7610294222831726
```

Q3. B) In this section, the pre-trained DenseNet201 has been used for the classification of images in the Oxford flowers 102 dataset. In which, the test accuracy before tuning is 94.24% and the test accuracy after tuning is 100% where RMSprop optimizer function has been used with a much lower learning rate 0.0001 for compiling the model.

In [16]:
```
loss, accuracy = model.evaluate(train_batches)
print('\nTest accuracy before tuning:', accuracy)
```

```
13/13 [==============================] - 3s 100ms/step - loss: 0.7313 - accuracy: 0.9424

Test accuracy before tuning: 0.9424019455909729
```

In [22]:
```
loss, accuracy = model.evaluate(train_batches)
print('\nTest accuracy after tuning :', accuracy)
```

```
13/13 [==============================] - 3s 100ms/step - loss: 0.0021 - accuracy: 1.0000

Test accuracy after tuning : 1.0
```