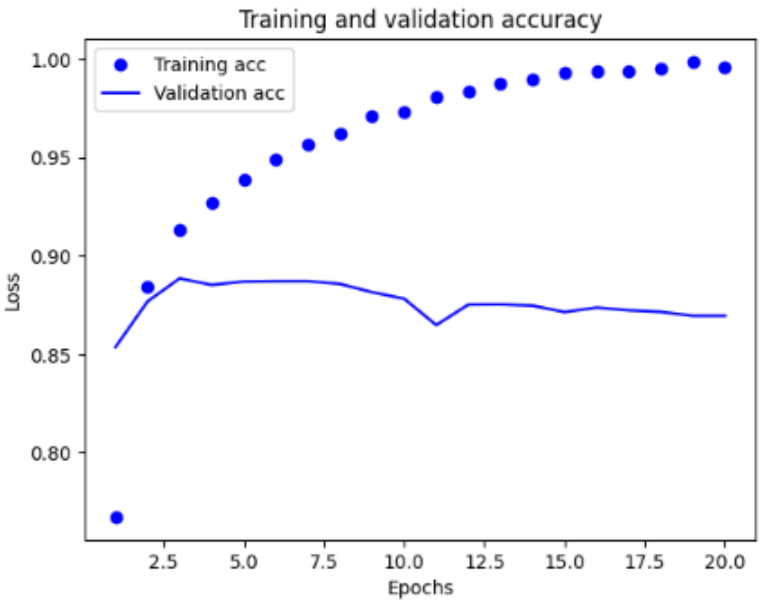
Sania Bibi

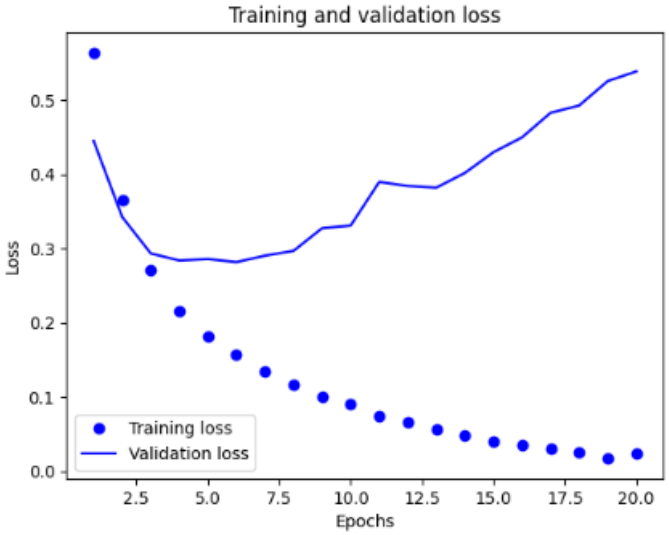
Task # 20

Binary classification

```
acc_values = history_dict['acc']
acc = history_dict['acc']
val_acc_values = history_dict['val_acc']
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc_values, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



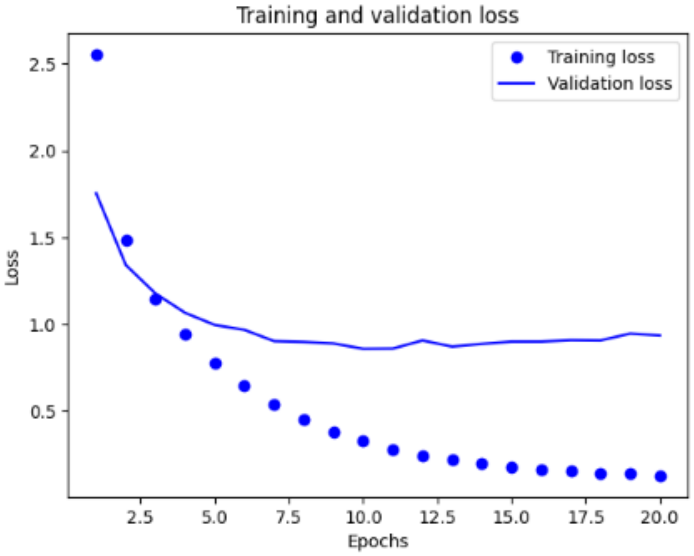
```
[21] val_loss_values = history_dict['val_loss']
     epochs = range(1, len(acc) + 1)
     plt.plot(epochs, loss_values, 'bo', label='Training loss')
     plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
     plt.title('Training and validation loss')
     plt.xlabel('Epochs')
     plt.ylabel('Loss')
     plt.legend()
     plt.show()
```



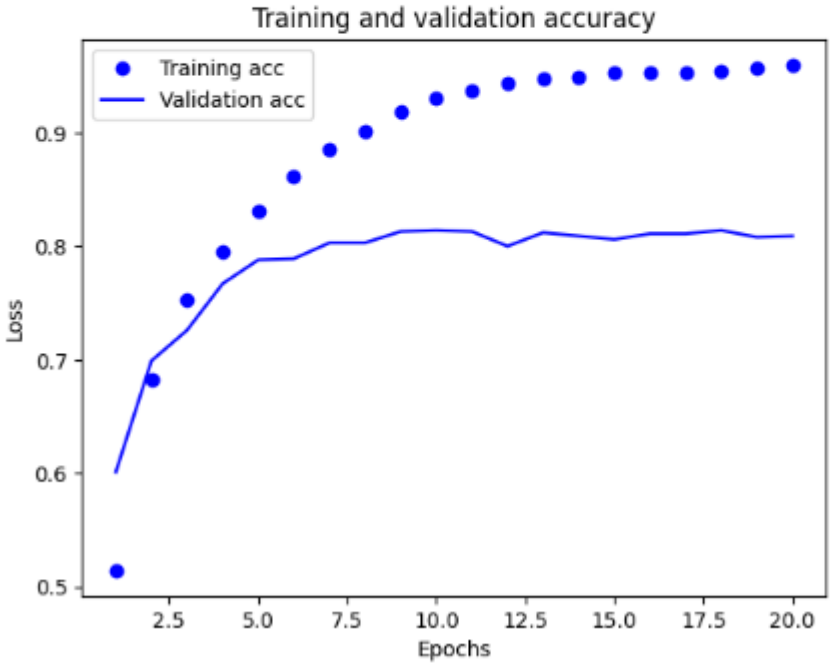
Multiclass classification

+ Code + Text

```
[15] loss = history.history['loss']
     val_loss = history.history['val_loss']
     epochs = range(1, len(loss) + 1)
     plt.plot(epochs, loss, 'bo', label='Training loss')
     plt.plot(epochs, val_loss, 'b', label='Validation loss')
     plt.title('Training and validation loss')
     plt.xlabel('Epochs')
     plt.ylabel('Loss')
     plt.legend()
     plt.show()
```



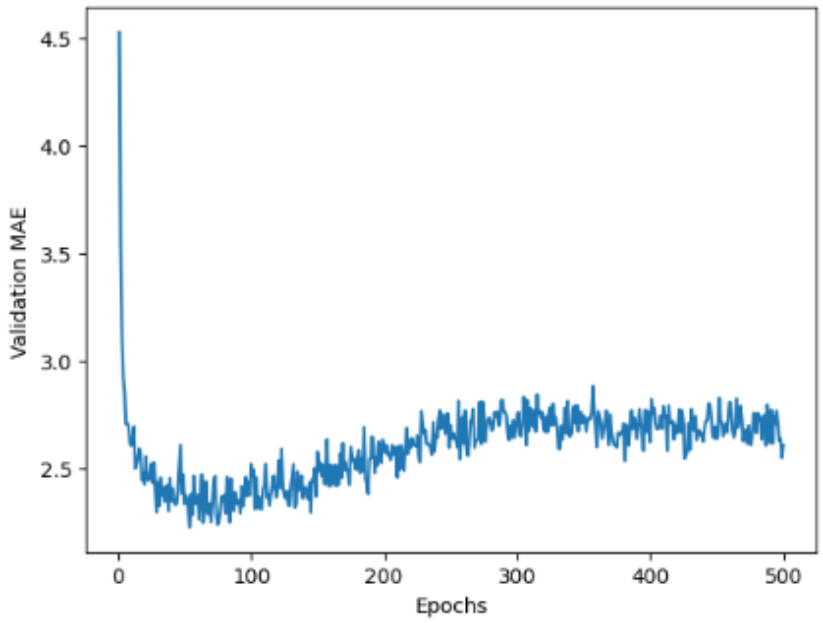
```
[22] plt.clf()
     acc = history.history['accuracy']
     val_acc = history.history['val_accuracy']
     plt.plot(epochs, acc, 'bo', label='Training acc')
     plt.plot(epochs, val_acc, 'b', label='Validation acc')
     plt.title('Training and validation accuracy')
     plt.xlabel('Epochs')
     plt.ylabel('Loss')
     plt.legend()
     plt.show()
```



Regression

```
np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)]
```

[18]
```
import matplotlib.pyplot as plt
plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



+ Code + Text

```
def smooth_curve(points, factor=0.9):
  smoothed_points = []
  for point in points:
    if smoothed_points:
        previous = smoothed_points[-1]
        smoothed_points.append(previous * factor + point * (1 - factor))
    else:
      smoothed_points.append(point)
  return smoothed_points
smooth_mae_history = smooth_curve(average_mae_history[10:])
plt.plot(range(1, len(smooth_mae_history) + 1), smooth_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```