- ***Creating the model***
  - -> getting the data in the right form / shape is the hardest part
  - -> this is creating the model
  - ***-> creating the model using keras***
    - -> keras is used to create neural networks in Python
    - -> each line of this code is a layer in the neural network
  - ***-> using keras to pass through the layers sequentially -> i.e one by one***
    - -> taking the 28x28 structure which stores the images and flattening them into a neural network
    - -> the second line, layer 2 -> is a dense layer
      - -> all of the neurones in the previous layer are connected to all of the neurones in the next
      - -> and there is an activation function being used -> to normalise those values and make the algorithm easier to run
        - -> sigmoid, tanh -> there are different activation layers we can use
    - ***-> then the last layer***
      - -> there are 10 neurones in this layer
      - -> there are 10 different items of clothing which the images inputted into the model could be
        - -> softmax <- to make sure that the values for each of those nodes sum to one
        - -> each of the values at those nodes are the probabilities that that node will be that specific item of clothing

```python
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),  # input layer (1)
    keras.layers.Dense(128, activation='relu'),  # hidden layer (2)
    keras.layers.Dense(10, activation='softmax') # output layer (3)
])
```

      - -> this is the architecture of the neural network
        - -> the number of neurones at each layer
        - -> the activation function
        - -> the type of connections
    - ***-> compiling the model***
      - -> picking the optimiser, the loss function and the metrics which we're going to use
        - the function we're going to use to optimise -> and the value we want to increase (the accuracy of the model)
      - -> the optimiser is the function which is going to perform the gradient descent
      - -> hyperparameters are things we can change -> e.g the number of nodes in the model or the activation function
        - -> hyperparameter tuning is changing these values to optimise the model
        - -> compiling the model
    - ***-> training the model***
      - -> fitting the model to the training data
      - -> this is the same as training the model
      - -> during each stage of the training, in this case it's printing out the success rates of each iteration as the model
      - -> it's being ran on the cloud -> google collaboratory
      - -> the amount of resources it takes to tain the model
      - -> the model has an accuracy when it's being trained -> the actual accuracy of the model is how it behaves when it's being ran on the data which it wasn't trained on
      - -> (below) this was the code used to train the model

```
4423680/4422102 [==============================] - 0s 0us/step

[3] model = keras.Sequential([
        keras.layers.Flatten(input_shape=(28, 28)),  # input layer (1)
        keras.layers.Dense(128, activation='relu'),  # hidden layer (2)
        keras.layers.Dense(10, activation='softmax') # output layer (3)
    ])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=10)

... Train on 60000 samples
```

- -> the model has been fit

- ***-> finding the true accuracy of the model***
  - -> testing it on the testing data

```
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=1)

print('Test accuracy:', test_acc)
```

  - -> verbose dictates how much data is being printed out when we run the model
  - -> overfitting -> this is when the accuracy of the model is high on the training data and low on the testing data <- because it wasn't being given enough data when training the model
    - -> we need to make sure that the model generalises properly
    - -> overfitting and hyper-parameter tuning
    - -> evaluating the model is how we print its accuracy
    - -> the accuracy can be suspiciously high, in which case it may have been overfit
    - -> less epochs can improve the accuracy of the model -> more epochs means less information per epoch which means the model can be overfit and the accuracy can be too high
      - -> then when it comes to testing the model on data it hasn't seen before the accuracy drops

- ***-> using the model to make predictions***

```
predictions = model.predict([test_images[0]])
```

  - -> test_images[0] <- this is an array which you want the model to predict
  - -> it is an array of different pixels for an image
  - -> and we are running the model on it
  - -> the predictions it returns is an array
    - -> that array is an array of probabilities
    - -> then using numpy to return the maximum value of that list -> aka the highest probability
    - -> which means it's a boot
  - ***-> what it's done is***
    - -> the image is represented by an array of pixels
    - -> the model was ran on it
    - -> then it's returned an array of probabilities

- Next is (continued)
    - -> each element in that array contains the probability that the image is a picture of a different item of clothing
    - -> the largest number (max) in that output array is the item of clothing which that image is most likely to contain <- in other words the prediction for the model
    - ***-> making predictions on any entry that we want***
        - -> making predictions on the model
        - -> he then he's running the code in another cell where you input a random number
        - -> (below) this is the code which does this

```python
COLOR = 'white'
plt.rcParams['text.color'] = COLOR
plt.rcParams['axes.labelcolor'] = COLOR

def predict(model, image, correct_label):
  class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
  prediction = model.predict(np.array([image]))
  predicted_class = class_names[np.argmax(prediction)]

  show_image(image, class_names[correct_label], predicted_class)


def show_image(img, label, guess):
  plt.figure()
  plt.imshow(img, cmap=plt.cm.binary)
  plt.title("Excpected: " + label)
  plt.xlabel("Guess: " + guess)
  plt.colorbar()
  plt.grid(False)
  plt.show()


def get_number():
  while True:
    num = input("Pick a number: ")
    if num.isdigit():
      num = int(num)
      if 0 <= num <= 1000:
        return int(num)
    else:
      print("Try again...")

num = get_number()
image = test_images[num]
label = test_labels[num]
predict(model, image, label)
```

- ***Next is***
    - -> convolutional neural networks
    - -> deep computer vision / object recognition and detection
    - -> building a model of dense layers