



Here is how ChatGPT
built my code.

A step by step process.



Santiago ✅
@svpino

...

Disclaimer:

I build Machine Learning models for a living. I know what questions to ask, and I can spot mistakes in ChatGPT's suggestions.

ChatGPT is an incredible tool, but you must understand how to use it properly.

Let's get started!



Santiago ✅

@svpino

...

I want to build a neural network to solve the MNIST problem using Keras.

I asked ChatGPT to start things off for me.

The result worked flawlessly without having to change anything.

This was a great start!



Sure, here's an example of a simple neural network built using Keras to classify handwritten digits from the MNIST dataset:

Copy code

```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.utils import to_categorical

# load the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# flatten the 28x28 images to a 784-dimensional vector
X_train = X_train.reshape((60000, 784))
X_test = X_test.reshape((10000, 784))

# normalize the pixel values (0-255) to be between 0 and 1
X_train = X_train.astype("float32") / 255
X_test = X_test.astype("float32") / 255

# convert the labels from integers to categorical
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# define the model architecture
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

# compile the model
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy'])

# train the model
model.fit(X_train, y_train, batch_size=128, epochs=20, validation_data=
(X_test, y_test))
```



Santiago ✅

@svpino

...

ChatGPT used Categorical CrossEntropy as the loss function. I asked it to use a different one.

The updated code also works.

Notice how it changed the loss function, but it also dropped the categorical transformation of labels. This is key for the new loss function work!

SV

Rewrite the code using sparse caterorical crossentropy



Sure, here's an updated version of the code that uses

Like Dislike

`sparse_categorical_crossentropy` as the loss function instead of
`categorical_crossentropy`:

Copy code

```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout

# load the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# flatten the 28x28 images to a 784-dimensional vector
X_train = X_train.reshape((60000, 784))
X_test = X_test.reshape((10000, 784))

# normalize the pixel values (0-255) to be between 0 and 1
X_train = X_train.astype("float32") / 255
X_test = X_test.astype("float32") / 255

# define the model architecture
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

# compile the model
model.compile(optimizer='rmsprop',
              loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# train the model
model.fit(X_train, y_train, batch_size=128, epochs=20, validation_data=
(X_test, y_test))
```



Santiago ✅

@svpino

...

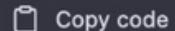
ChatGPT's code uses fully-connected layers.

But I want to use convolutional layers.

ChatGPT modified the code and added a Conv2D and a MaxPooling layer.

sv Use convolutional layers instead

 Sure, here's an example of how you can modify the previous code to use convolutional layers instead of fully connected layers:

 Copy code

```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D

# load the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# normalize the pixel values (0-255) to be between 0 and 1
X_train = X_train.astype("float32") / 255
X_test = X_test.astype("float32") / 255

# define the model architecture
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

# compile the model
model.compile(optimizer='rmsprop',
              loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# train the model
model.fit(X_train, y_train, batch_size=128, epochs=20, validation_data=(X_test, y_test))
```



Santiago ✅

@svpino

...

I need to evaluate the final model, so I asked ChatGPT to write code using the test set.

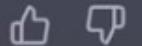
The result and explanation look great!

sv

write code to evaluate this model on the test set



Once you have trained a model, you can use the `evaluate` method to measure its performance on the test set. Here's an example of how you can use `evaluate` to measure the accuracy of the model on the test set:



Copy code

```
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
```

The `evaluate` method takes the test data and the test labels as arguments and returns the loss and the metrics specified during model compilation (in this case, accuracy). The above code snippet prints the test accuracy, which should give you an idea of how well the model generalizes to new data.



Santiago ✅

@svpino

...

The model uses the entire test set as validation data during training.

Let's ask ChatGPT to change that to 20% of the train data.

This is good stuff!

(By the way, every answer from ChatGPT comes with a full explanation. I'm not including it here to keep screenshots simple.)

sv

use 20% of the test dataset as validation during training



During the training process, it's a good practice to use a separate validation set to monitor the performance of the model and prevent overfitting. One way to do this is to split the test set into two parts: a validation set and a final test set.

Here's an example of how you can use 20% of the test set as the validation set during training:

Copy code

```
import numpy as np

# split the test set into a validation set and a final test set
X_val = X_test[:2000]
y_val = y_test[:2000]
X_test = X_test[2000:]
y_test = y_test[2000:]

# train the model
model.fit(X_train, y_train, batch_size=128, epochs=20, validation_data=(X_val, y_val))
```



Santiago ✅

@svpino

...

I want smaller batches and run the code for fewer epochs.

Of course, I don't need ChatGPT for this, but I did not want to change my code without updating ChatGPT's context. That's why I asked.

sv

I want to use batches of 32 elements and train for 10 epochs



Sure, you can modify the `fit` method to use batches of 32 elements and train for 10 epochs. Here's an example of how you can do this:

Copy code

```
# train the model using batches of 32 elements for 10 epochs
model.fit(X_train, y_train, batch_size=32, epochs=10, validation_data=
(X_val, y_val))
```

By specifying `batch_size=32` in the `fit` method, Keras will update the model's weights after every 32 samples. And by specifying `epochs=10`, the model will be trained for 10 iterations over the entire training dataset.

It's worth noting that the batch size and number of epochs are hyperparameters of the model that can be adjusted to achieve the best performance. A smaller batch size may require more memory, but it can provide a more accurate estimate of the gradient, while a larger batch size can save memory, but it may take longer to converge. Similarly, the number of epochs should be chosen based on the complexity of the problem and the available computational resources.

Additionally, you can also monitor the training process by passing a validation dataset to the `validation_data` argument of the `fit` method. This will evaluate the model on the validation set after each epoch and print the validation loss and validation accuracy. You can use this information to determine when the model starts to overfit and stop the training early or use it to tune the model's hyperparameters.

Santiago ✅

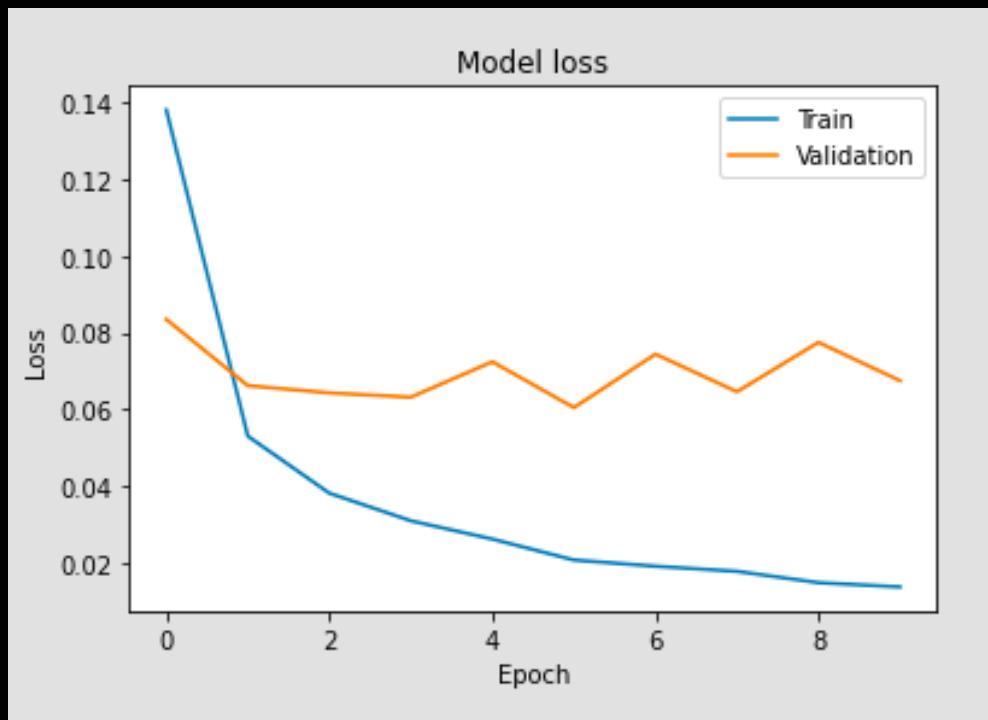
@svpino

...

Let's now draw a plot of the training and testing losses during the training process.

ChatGPT explained that I need to change the line that fits the model to capture its result value (history.) That was neat!

The plot works and looks good!



SV draw a chart of the training and test losses using matplotlib

You can use the `history` object returned by the `fit` method to plot the training and validation loss over time using Matplotlib. Here's an example of how you can do this:

```
Copy code
```

```
import matplotlib.pyplot as plt

# train the model
history = model.fit(X_train, y_train, batch_size=32, epochs=10,
validation_data=(X_val, y_val))

# plot the training and validation loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```

Santiago ✅ @svpino · 1h

Replying to @svpino

I need to show a few examples from the dataset, so let's ask ChatGPT to write code to print a combination of images and labels.

...

The code was also perfect.

You can see the plot with 20 images attached.



write code that displays 20 random images and labels from the train set. Use a single figure with 4 rows and 5 columns.



You can use the Matplotlib library to display random images and labels from the train set. ⌂ ↴

Here's an example of how you can do this:

Copy code

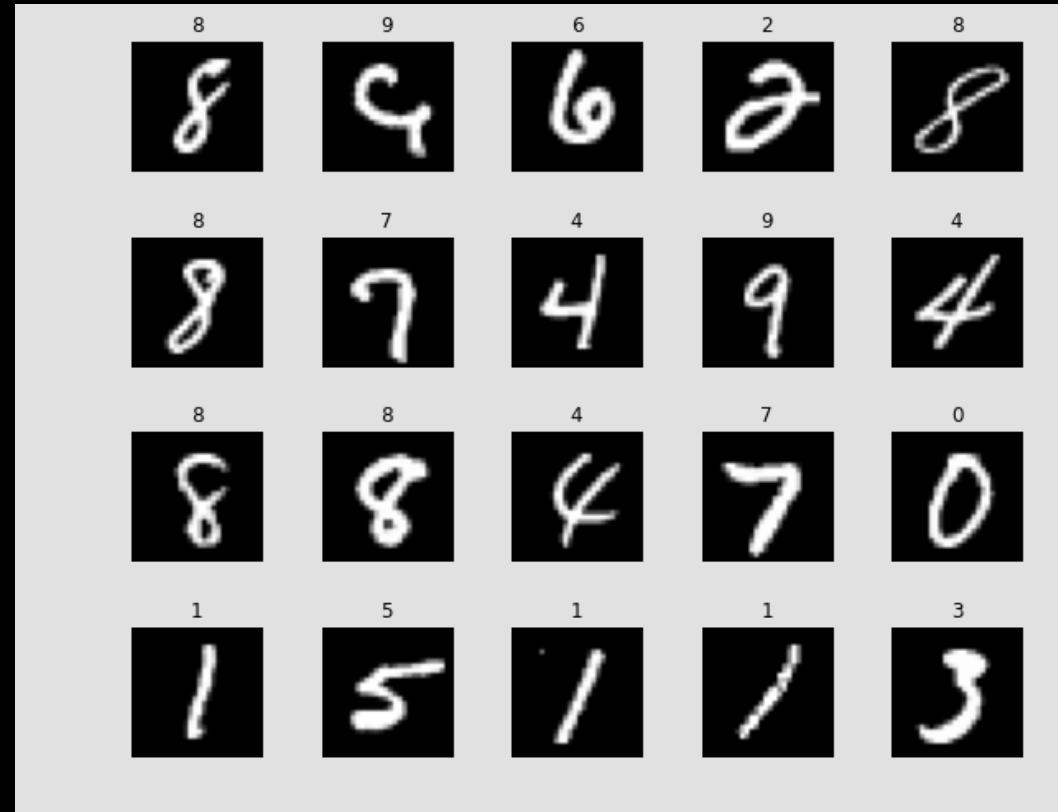
```
# Select 20 random indexes from the train set
random_indices = np.random.randint(0, len(X_train), 20)

# Extract the 20 random images and labels from the train set
random_images = X_train[random_indices]
random_labels = y_train[random_indices]

# Plot the 20 random images
fig, axes = plt.subplots(4, 5, figsize=(10, 8))
axes = axes.ravel()

for i, ax in enumerate(axes):
    ax.imshow(random_images[i], cmap="gray")
    ax.axis('off')
    ax.set_title(str(random_labels[i]))

plt.subplots_adjust(hspace=0.5)
plt.show()
```





Santiago ✅
@svpino

...

Something useful whenever you build a model is to take a look at its structure.

Let's ask ChatGPT how I can do that.

sv I want to print the structure of the model

You can use the `summary` method of the model to print the structure of a model in Keras.
Here's an example of how you can use it:

```
model.summary()
```

Copy code

This method will print a summary of the model's architecture, including the number of layers, the number of parameters in each layer, and the shape of the input and output tensors. It is a useful tool to check the architecture of the model, and the number of parameters in it.



Santiago ✅

@svpino

...

Okay, the summary of the model is helpful, but I'd rather see a plot showing the model's structure.

Let's ask:

sv

Plot the structure of the model in a diagram



You can use the `plot_model` function from the `keras.utils` module to plot the structure of a Keras model as a diagram. Here's an example of how you can use it:

Copy code

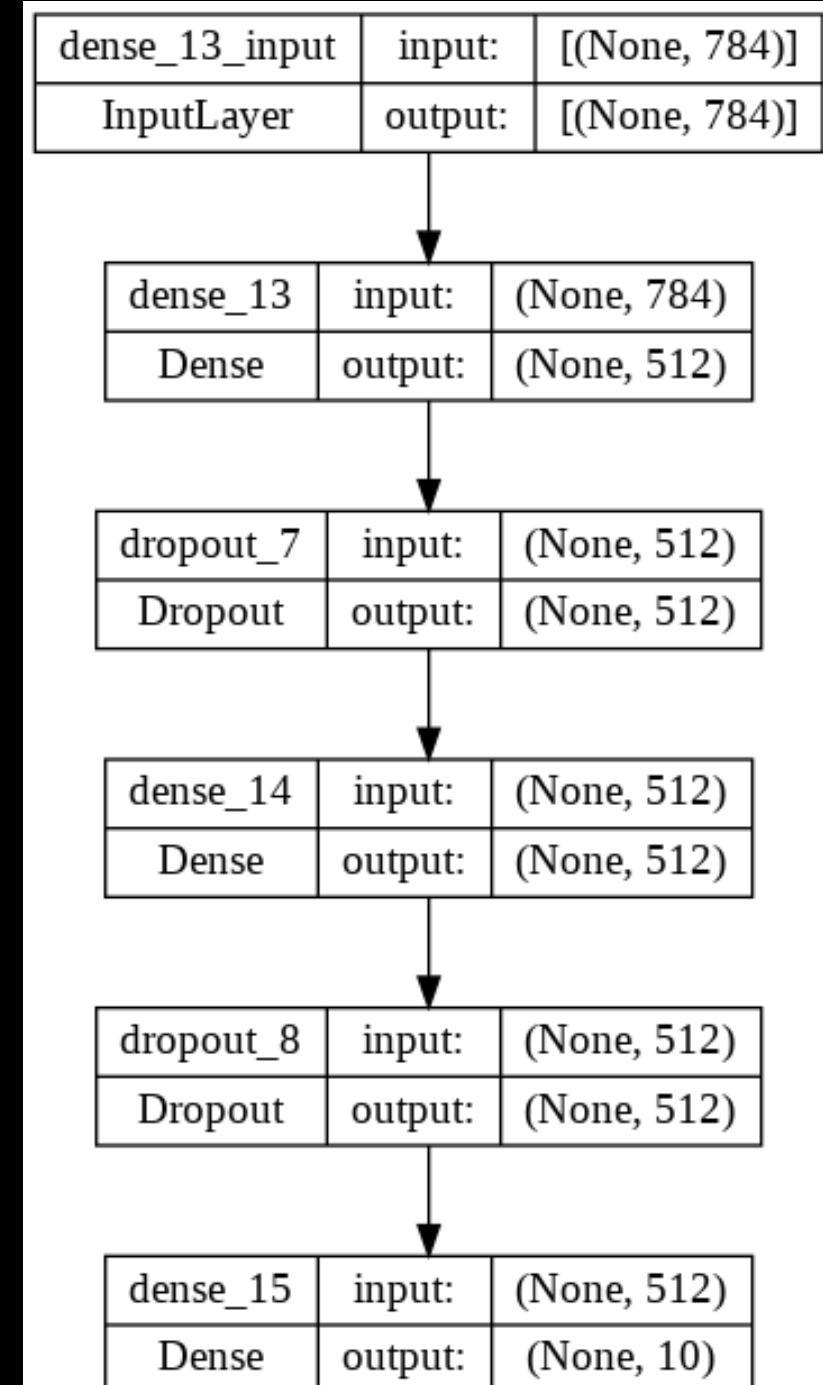
```
from keras.utils import plot_model
plot_model(model, to_file='model.png', show_shapes=True,
show_layer_names=True)
```

This will create a PNG image of the model architecture, where the layers are represented as nodes and the connections between layers are represented as edges. The `show_shapes` parameter allows you to show the output shape of each layer in the diagram and `show_layer_names` parameter is used to show the name of each layer.

You can also specify `rankdir='TB'` to make the diagram layout vertical.

This can be very helpful to visualize and understand the architecture of your model. It also can be used to communicate the model architecture to others and to spot any error or inconsistencies in the architecture design.

Please note that to use this function, you need to have Graphviz library installed and it's also required that the graphviz executables are in the PATH for the system.





Santiago ✅

@svpino

...

This is looking good! Let's now get ready to deploy this model.

Let's ask ChatGPT to save the model to disk.

sv

save the model to disk



You can use the `save` method of the model to save the model's architecture, its weight values, and its training configuration to disk. Here's an example of how you can use it:

Copy code

```
model.save('model.h5')
```

This will save the model to a single HDF5 file with the name "model.h5" in your current working directory. The saved model can later be loaded using the `keras.models.load_model` function.

Copy code

```
from keras.models import load_model  
loaded_model = load_model("model.h5")
```

Saving the model allows you to reuse the trained model later for inference, for fine-tuning on new data, or for continued training. The saved model can also be easily shared with others and used on other machines.

The `save` method also allows to save the model in multiple formats, for example TensorFlow SavedModel and JSON. This can be done by passing the `save_format` parameter with the appropriate format.



Santiago ✅

@svpino

...

Now, I want to create a class that uses the saved model to make predictions.

This is an interesting prompt, and the solution is perfect.

sv

create a class Predictor with a function that loads the model from disk if it hasn't been loaded and predicts the image supplied as parameter. the function should return the predicted label.



Here's an example of how you can create a `Predictor` class with a `predict` method that loads the model from disk if it hasn't been loaded and predicts the image supplied as parameter. The function should return the predicted label.

Copy code

```
from keras.models import load_model

class Predictor:
    def __init__(self, model_path):
        self.model_path = model_path
        self.model = None

    def predict(self, image):
        # load the model from disk if it hasn't been loaded
        if self.model is None:
            self.model = load_model(self.model_path)

        # predict the image
        prediction = self.model.predict(image)

        # return the predicted label
        return np.argmax(prediction)
```



Santiago 

@svpino

...

Let's now write an example that uses the Predictor class to predict the label of 10 random images:

sv

use the predictor class to predict 10 random images from the test set



Here's an example of how you can use the `Predictor` class to predict the labels of 10 random images from the test set:

 Copy code

```
# select 10 random images from the test set
random_indices = np.random.randint(0, len(X_test), 10)
random_images = X_test[random_indices]
random_labels = y_test[random_indices]

predictor = Predictor("model.h5")

for i in range(10):
    predicted_label =
predictor.predict(random_images[i].reshape(1,28,28,1))
    print("True Label: ", random_labels[i], "Predicted Label: ",
predicted_label)
```

In this example, we first use the `random.randint()` function from the numpy library to generate 10 random indices from the test set, then extract the 10 random images from the test set corresponding to those indices.

We create an instance of the `Predictor` class with the path of the saved model. Then, in a for loop for each image, we use the `predict` method of the `Predictor` class to predict the label of that image, then we print the true label and the predicted label for that image.

It's important to note that the predictions of the model can be affected by various factors such as random initialization of weights, data shuffling, and others. So, it's a good practice to predict multiple times and take average results. Also the images in the dataset are in the shape of (28,28) while the model has the input shape of (28,28,1) so we reshape it before predicting.

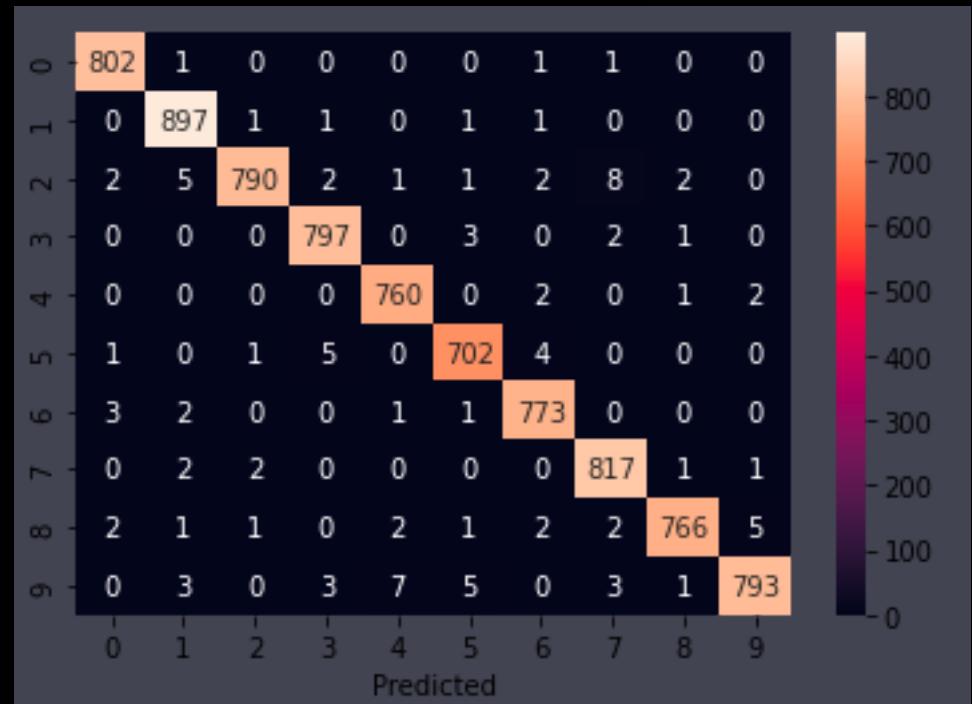


Santiago ✅

@svpino

To finish this, let's display a confusion matrix.

I like the styles that ChatGPT used!



sv

Use the model.predict to predict the test set and plot a confusion matrix



You can use the `predict` method of the model to predict the test set, and then use the `confusion_matrix` function from the `sklearn.metrics` module to create a confusion matrix. Here's an example of how you can do this:

Copy code

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Predict the test set
y_pred = model.predict(X_test)

# Convert the predicted probabilities to class labels
y_pred = np.argmax(y_pred, axis=1)

# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```