Introduction to Machine Learning with Keras

NSF HDR ML Hackathon
Max Cohen

Penn a3d3 imageomics BIO ML iHARP Support by NSF

Introduction to Neural Networks

Goals:

1) Give you a good intuition for how neural networks work, and what they're doing

2) Go over basic implementations in python using TensorFlow / Keras

Introduction to Neural Networks

Question: Is it possible to approximate the outputs of arbitrary functions if we have enough tunable parameters?

Examples: f(x) = y

- A function which takes in CT scan images, and outputs 1 when the patient has cancer, 0 otherwise

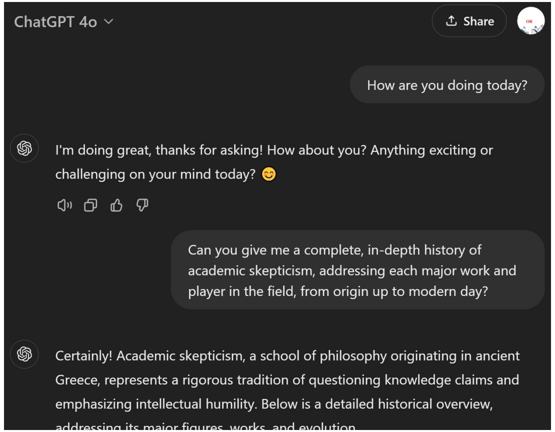
- A function which takes in arbitrary text as inputs, and outputs the next word (which can be looped over and over)

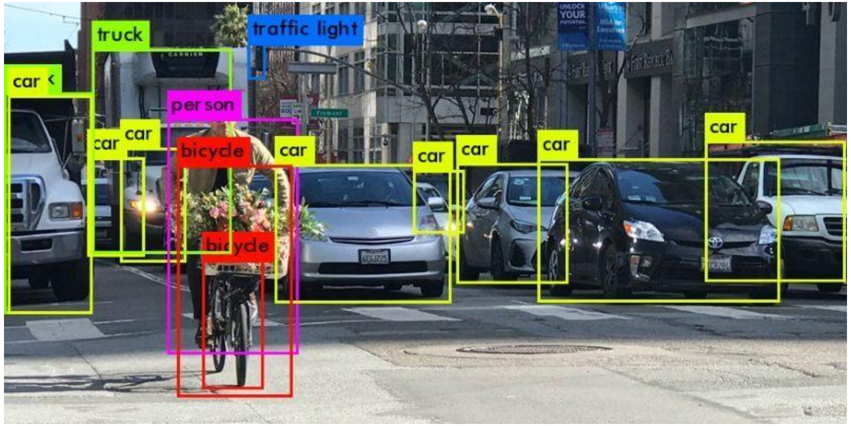
- A function which takes in images as inputs, and outputs the names of each object in the image

Introduction to Neural Networks

- **Answer:** Yes!

Visualized predictions of the first 5 images with actual and predicted classes





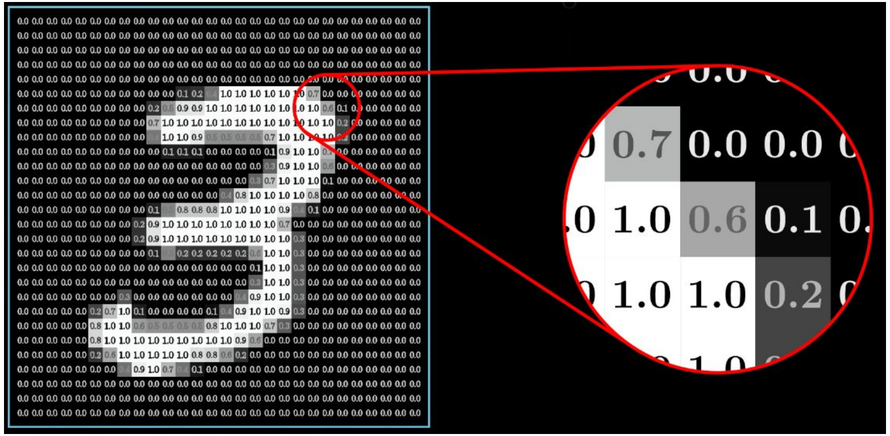
Introduction to Neural Networks

- **<u>So, how does this work?</u>**

f(x) = y

Step 1: Turn your inputs (x) into numbers

- For pictures, give each color an RGB value
- For text, associate each word to a different number
- For scientific data, the inputs are often already numerical



Introduction to Neural Networks

- **So, how does this work?**

- Step 2: matrix multiplication

Input data

$$\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} & w_{0,4} \\ w_{1,0} & w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,0} & w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} \end{bmatrix} \cdot \begin{bmatrix} i_0 \\ i_1 \\ i_2 \\ i_3 \\ i_4 \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

Output data

$$= \begin{bmatrix} w_{0,0} \cdot i_0 + w_{0,1} \cdot i_1 + w_{0,2} \cdot i_2 + w_{0,3} \cdot i_3 + w_{0,4} \cdot i_4 + b_0 \\ w_{1,0} \cdot i_0 + w_{1,1} \cdot i_1 + w_{1,2} \cdot i_2 + w_{1,3} \cdot i_3 + w_{1,4} \cdot i_4 + b_1 \\ w_{2,0} \cdot i_0 + w_{2,1} \cdot i_1 + w_{2,2} \cdot i_2 + w_{2,3} \cdot i_3 + w_{2,4} \cdot i_4 + b_2 \end{bmatrix}$$

Introduction to Neural Networks

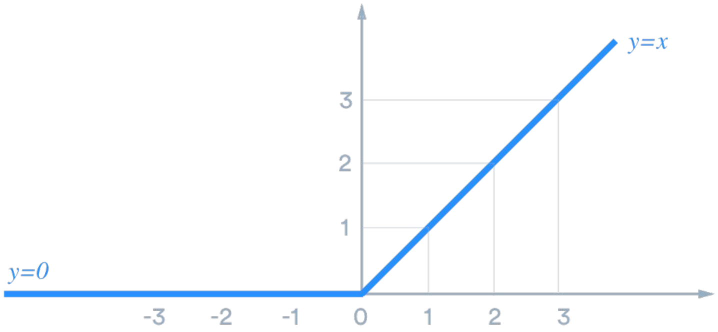
- **So, how does this work?**

- Step 3: Nonlinearity

$$G\left(\begin{bmatrix} w_{0,0}\cdot i_0 + w_{0,1}\cdot i_1 + w_{0,2}\cdot i_2 + w_{0,3}\cdot i_3 + w_{0,4}\cdot i_4 + b_0 \\ w_{1,0}\cdot i_0 + w_{1,1}\cdot i_1 + w_{1,2}\cdot i_2 + w_{1,3}\cdot i_3 + w_{1,4}\cdot i_4 + b_1 \\ w_{2,0}\cdot i_0 + w_{2,1}\cdot i_1 + w_{2,2}\cdot i_2 + w_{2,3}\cdot i_3 + w_{2,4}\cdot i_4 + b_2 \end{bmatrix}\right)$$

New output data

G =



ReLU activation

Introduction to Neural Networks

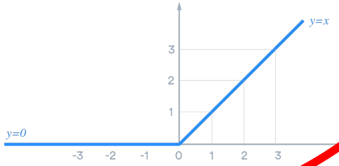
- **So, how does this work?**

- Step 4: Repeat

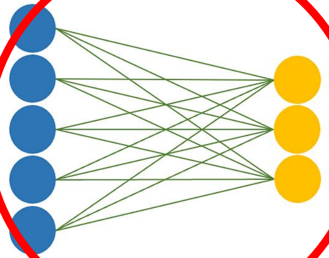
1 Network **Layer**

$$\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & w_{0,3} & w_{0,4} \\ w_{1,0} & w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,0} & w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} \end{bmatrix} \cdot \begin{bmatrix} i_0 \\ i_1 \\ i_2 \\ i_3 \\ i_4 \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

$$= \begin{bmatrix} w_{0,0} \cdot i_0 + w_{0,1} \cdot i_1 + w_{0,2} \cdot i_2 + w_{0,3} \cdot i_3 + w_{0,4} \cdot i_4 + b_0 \\ w_{1,0} \cdot i_0 + w_{1,1} \cdot i_1 + w_{1,2} \cdot i_2 + w_{1,3} \cdot i_3 + w_{1,4} \cdot i_4 + b_1 \\ w_{2,0} \cdot i_0 + w_{2,1} \cdot i_1 + w_{2,2} \cdot i_2 + w_{2,3} \cdot i_3 + w_{2,4} \cdot i_4 + b_2 \end{bmatrix}$$



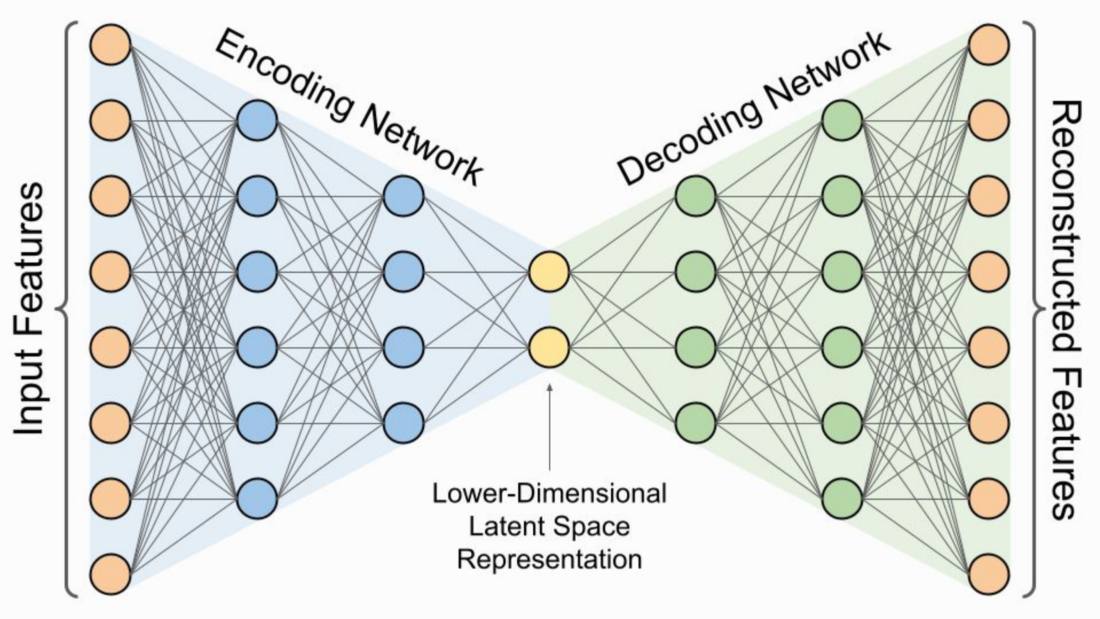
=



Introduction to Neural Networks

- **So, how does this work?**

- Step 4: Repeat



Introduction to Neural Networks

f(x) = y

- **Question:** How do we choose the network parameters (values in the matrices) such that the outputs actually approximate our function of interest?

Introduction to Neural Networks

f(x) = y

- **Question:** How do we choose the network parameters (values in the matrices) such that the outputs actually approximate our function of interest?

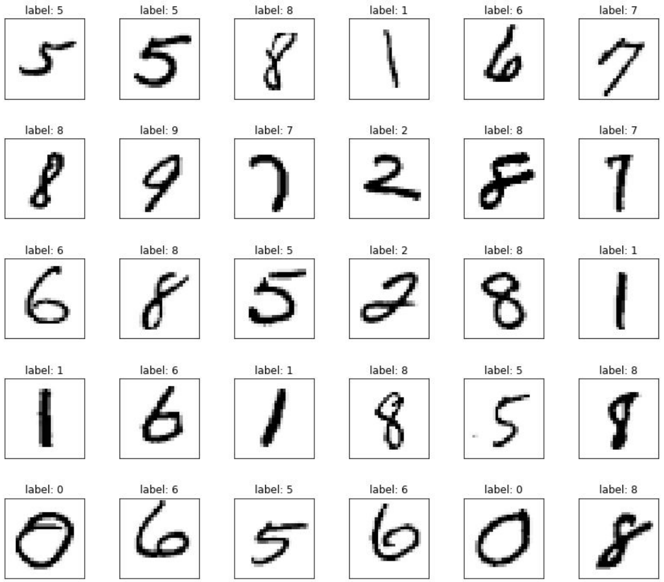
- **Answer:** Gradient Descent with Backpropagation

Introduction to Gradient Descent and Backpropagation

f(x) = y

Step 1:

Collect your **training data:** Compile a large number of input, output pairs



Introduction to Gradient Descent and Backpropagation

f(x) = y

Step 2:

Define a **loss function:**

MSE (mean squared error):

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Negative Log Likelihood:

$$L = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

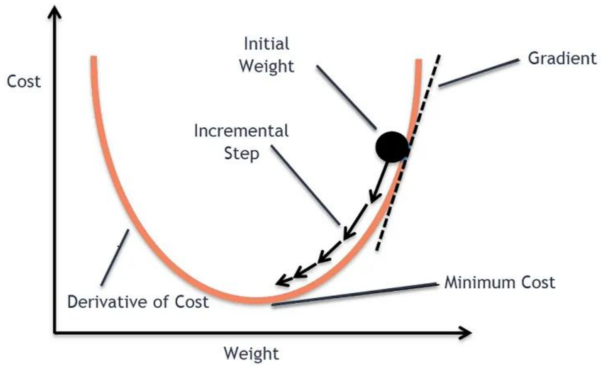
Introduction to Gradient Descent and Backpropagation

f(x) = y

Step 3:

Minimize the loss function!

- We know that functions are extremized when the derivative is 0
- Accordingly, we take derivatives of the loss function with respect to the model parameters (matrix elements)
- Then, update the parameters in the direction of steepest descent (the direction of the negative gradient)
- Repeat until loss is minimized



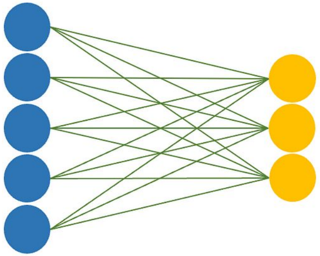
Introduction to Gradient Descent and Backpropagation

That's it!

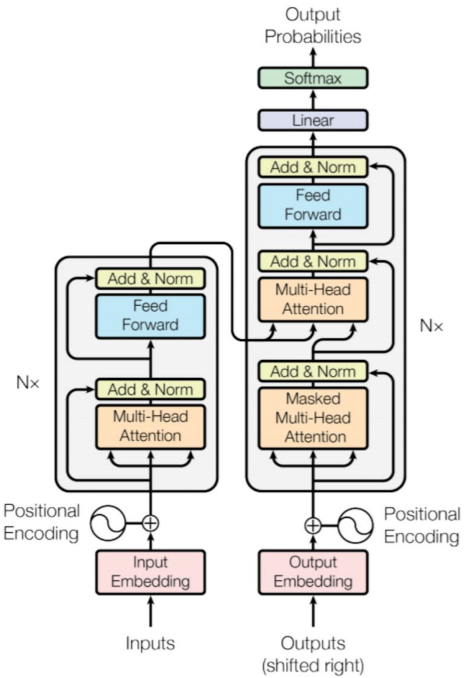
Now that the model parameters have been tuned correctly, our network will successfully approximate the output of the function we wanted!

One more detail: other types of layers

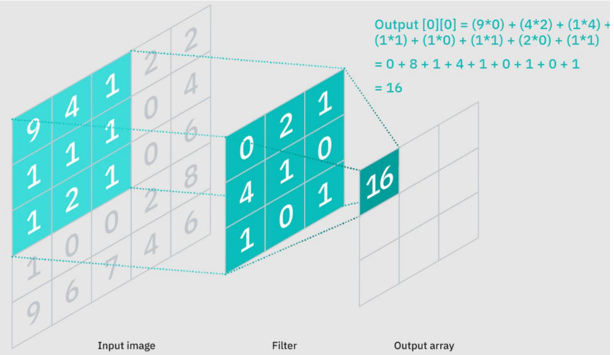
Dense Layer:



Transformer Layer:



Convolutional Layer:



Implementations in Keras

TensorFlow is a python package which handles the detailed calculations of neural networks, including gradient descent and tensor operations

Keras is a wrapper around TensorFlow which makes implementing neural networks very easy!

Keras

Simple. Flexible. Powerful.

TensorFlow

The Core of Keras: the Model and Defining Layers

The model is defined as an instance of the pre-defined “model” keras class

We define layers with the following syntax:

- The input data has shape (784,), which means it is a vector with 784 elements
- Next, we define 3 dense layers
  - The number written (e.g. 128) is the output dimension from that layer
  - The activation function (nonlinearity) is defined by the ‘activation’ argument

```
model = models.Sequential([
    layers.Input(shape=(784,)),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

The Core of Keras: the Model and Defining Layers

For convolutional layers, we can write:

```
cnn_model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

- Here, the input shape is a 2d picture, e.g. 28 by 28 with 1 channel
- We define convolution layers (with a 3 by 3 filter), alternating with max pooling layers
- Pictures are then flattened into a vector before two dense layers

The full training / testing pipeline: Imports and load data

```python
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
import numpy as np
```

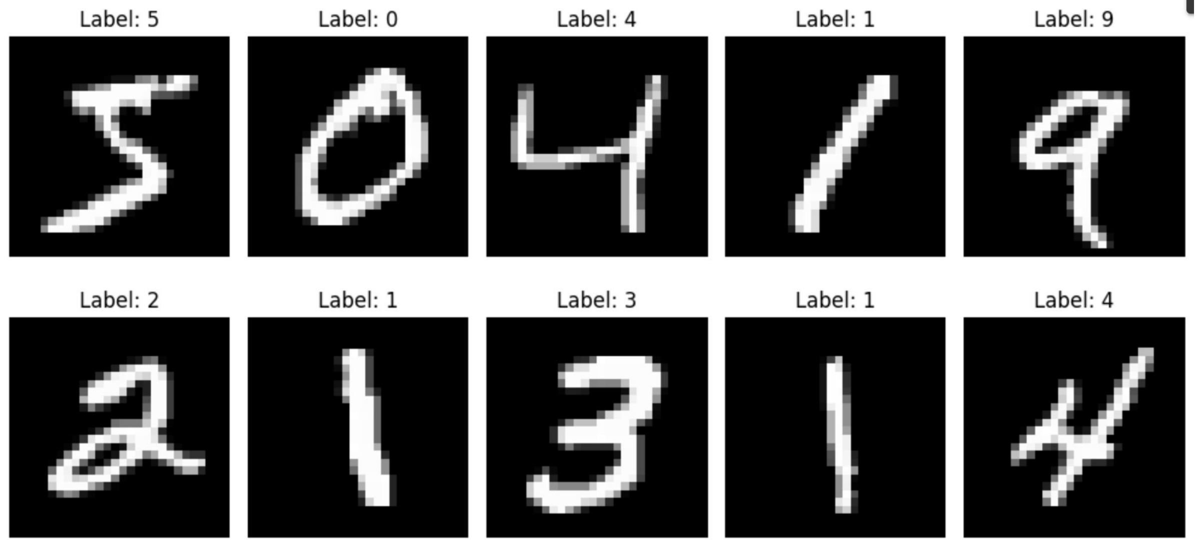
```python
# Load MNIST dataset from TensorFlow datasets
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Print dataset shape
print("Training set shape:", x_train.shape)
print("Test set shape:", x_test.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Training set shape: (60000, 28, 28)
Test set shape: (10000, 28, 28)
```

Plot a few sample images

```
# Visualize sample images
plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(x_train[i], cmap='gray')
    plt.title(f"Label: {y_train[i]}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```



Preprocessing

```
# Normalize pixel values to range [0, 1]
x_train = x_train / 255.0
x_test = x_test / 255.0

# Flatten the images for the DNN
x_train_flat = x_train.reshape(x_train.shape[0], -1)
x_test_flat = x_test.reshape(x_test.shape[0], -1)

print("Flattened training shape:", x_train_flat.shape)
print("Flattened test shape:", x_test_flat.shape)
```

```
Flattened training shape: (60000, 784)
Flattened test shape: (10000, 784)
```

Build and compile the model

- After we build the model, we have to **compile** it
  - Define the optimizer (the algorithm which controls gradient descent)
  - Define the loss function

```
# Build the DNN model
model = models.Sequential([
    layers.Input(shape=(784,)),  # Input layer (28x28 flattened)
    layers.Dense(128, activation='relu'),  # Hidden layer 1
    layers.Dense(64, activation='relu'),   # Hidden layer 2
    layers.Dense(10, activation='softmax')  # Output layer (10 classes)
])

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

# Print the model summary
model.summary()
```

Train the model

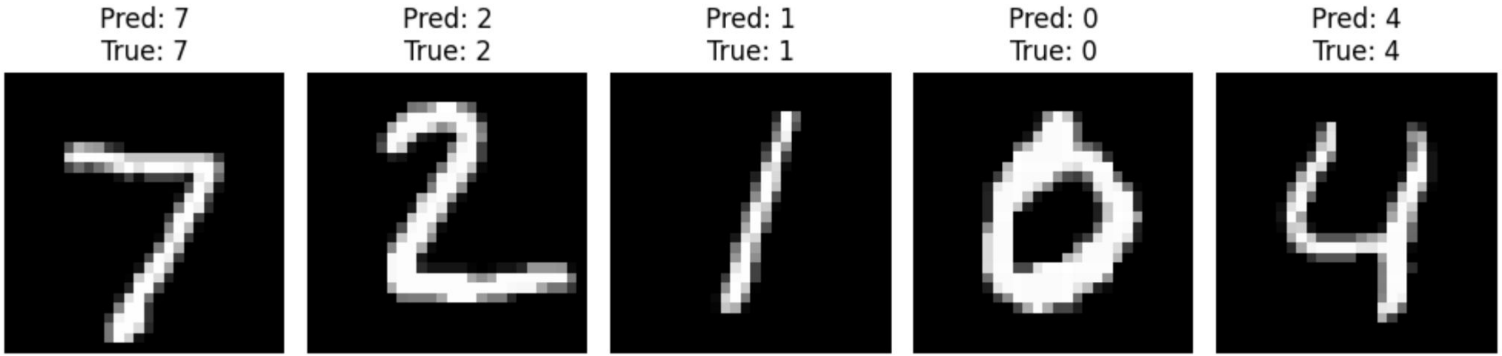
- Define the training data
- Specify the number of epochs, e.g. how long the network should train for
- Specify batch size e.g. how many samples are seen before weights update
- Specify what percentage of the data will be used for validation instead of training

```
# Train the model
history = model.fit(x_train_flat, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

Generate model predictions over the test data

```python
# Predict on test images
predictions = model.predict(x_test_flat)

# Visualize predictions
plt.figure(figsize=(10, 5))
for i in range(5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(x_test[i], cmap='gray')
    predicted_label = np.argmax(predictions[i])
    true_label = y_test[i]
    plt.title(f"Pred: {predicted_label}\nTrue: {true_label}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```

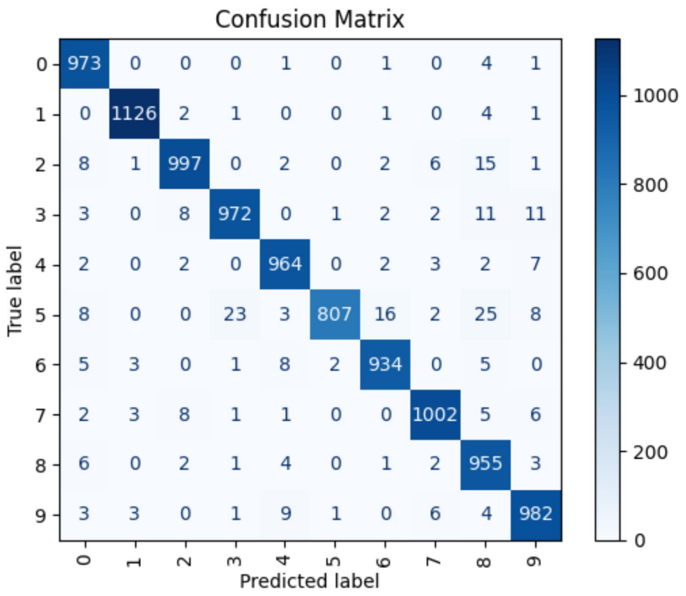


Plot confusion matrix

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Compute confusion matrix
y_pred = np.argmax(predictions, axis=1)
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=range(10))
disp.plot(cmap='Blues', xticks_rotation='vertical')
plt.title("Confusion Matrix")
plt.show()
```



Thank you for listening!

- Let me know if you have any questions
- The example notebook also trains a convolutional network, so feel free to take a look at the details!
- Good luck hacking!