AI/ML BOOTCAMP

William Phelps

Christopher Newport University/Jefferson Lab

CHRISTOPHER NEWPORT UNIVERSITY

Jefferson Lab
Thomas Jefferson National Accelerator Facility

Course Topics*

1 Semester DS/AI Course in 4 lectures
45 hours in 8 hours

- **Lectures 1 & 2**
  - **Lecture 1**
    - Introduction to workflow and tools (git/anaconda/pycharm)
    - Introduction to Data Science
    - Python Review with jupyter notebooks
  - **Lecture 2**
    - Data Visualization
    - Principles of data visualization from Edward Tufte
    - Pandas introduction
- **Lectures 3 & 4**
  - **Lecture 3**
    - Machine Learning Introduction
    - Regression/Curvefit
    - Overfitting/Underfitting
    - kNN classifier
    - MNIST/Handwritten digit classifier
  - **Lecture 4**
    - Overfitting/Underfitting
    - Early Stopping
    - Hyperparameter optimization
    - Convolutional Neural Networks
    - CIFAR 10 datasets

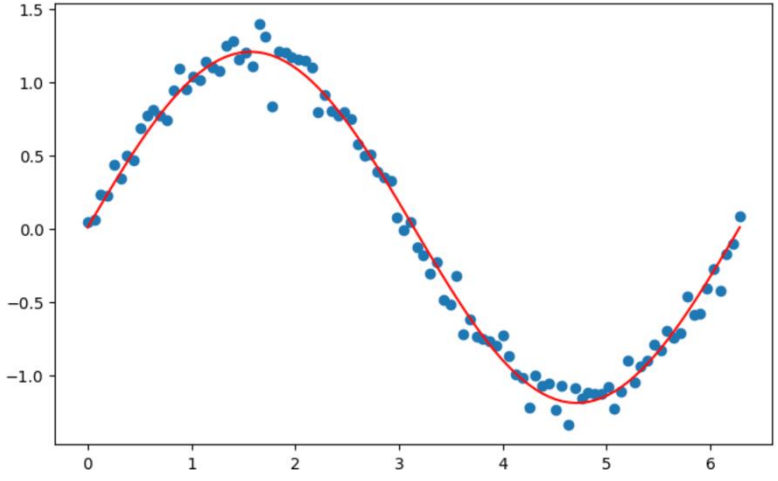
*Tentative

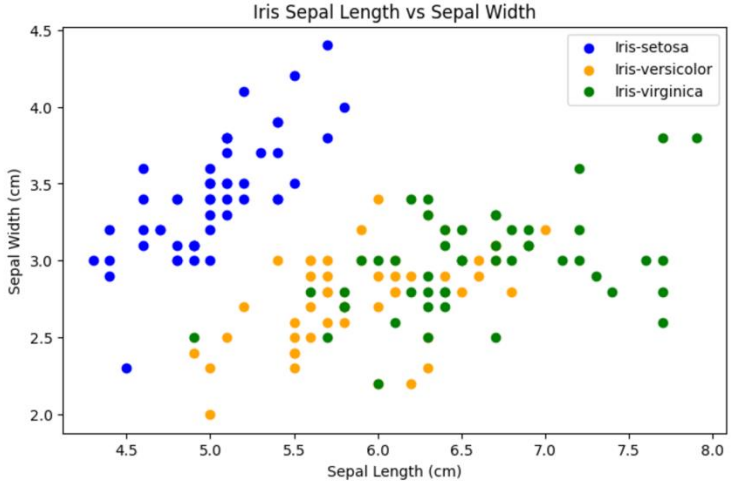
RECAP/MNIST

Recap – Regression and kNN

```
# Let's fit with curve fit and extract parameters
from scipy.optimize import curve_fit
def func(x, a, b):
    return a*np.sin(x) + b
popt, pcov = curve_fit(func, x, y)
```

```
# Plot function on top of dataset
plt.plot(x, y, 'o')
plt.plot(x, func(x, *popt), 'r-');
```

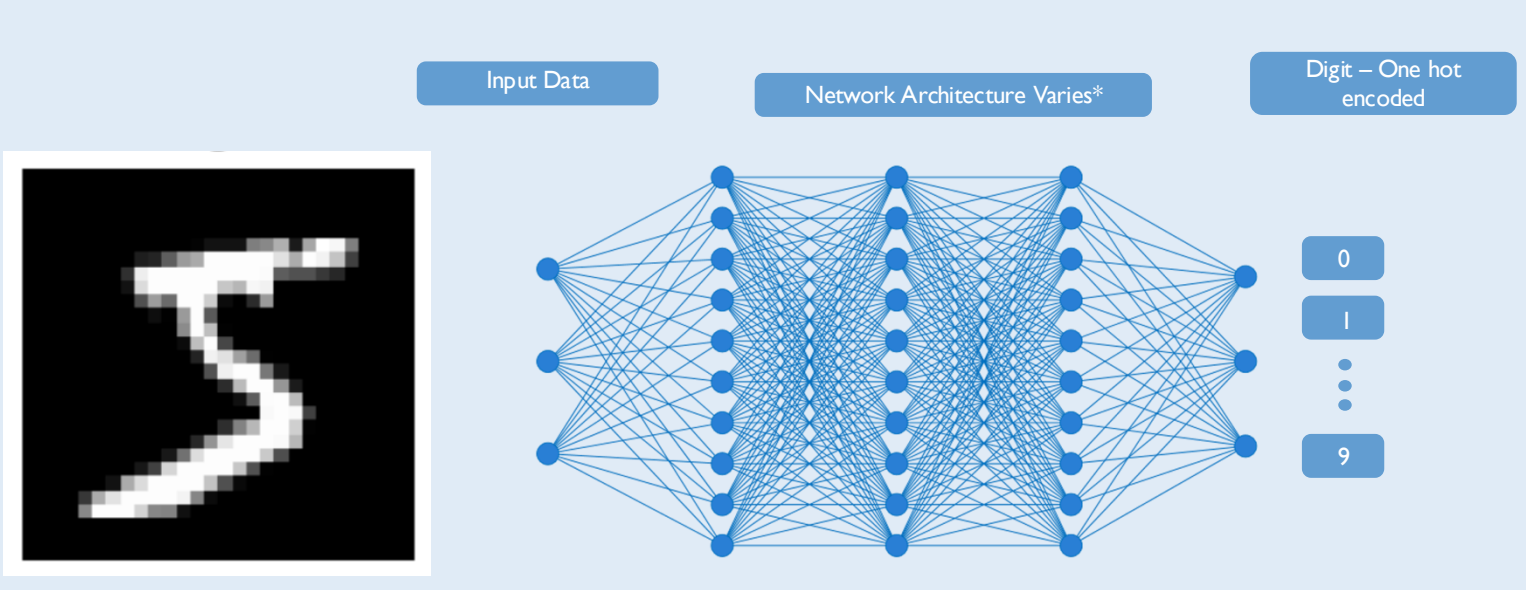





```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

MLP Example – MNIST

- Read in 28x28 images of handwritten digits
- Model Architecture: MLP, relu activation function

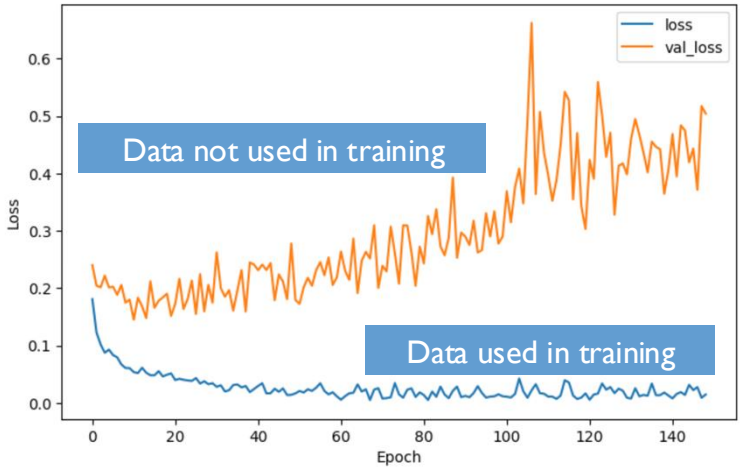


MNIST MLP

Is this the best we can do with a MLP?

```
# Build a MLP
model = keras.Sequential()
model.add(layers.Flatten(input_shape=(28, 28)))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.summary();
```

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




```
# Train the model
history = model.fit(train_images,
                    train_labels,
                    epochs=150,
                    batch_size=128,
                    validation_split=0.2)
```

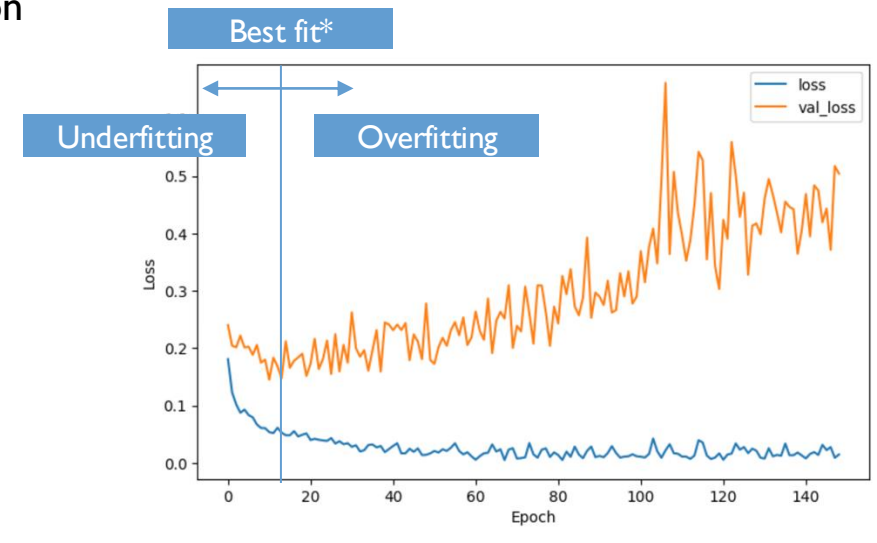
How many epochs?

How well did we do?

Early Stopping with Callbacks

- Callbacks monitor training and trigger actions during model.fit().
- Early stopping stops training when a chosen metric stops improving.
- Here, monitor='val_loss' tracks validation loss.
- patience=3 allows 3 epochs without improvement before stopping.
- This helps reduce overfitting and saves training time.

```
# Example of early stopping with callbacks
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=3)
history = model.fit(train_images,
                    train_labels,
                    epochs=50,
                    validation_split=0.2,
                    callbacks=[early_stopping])
```



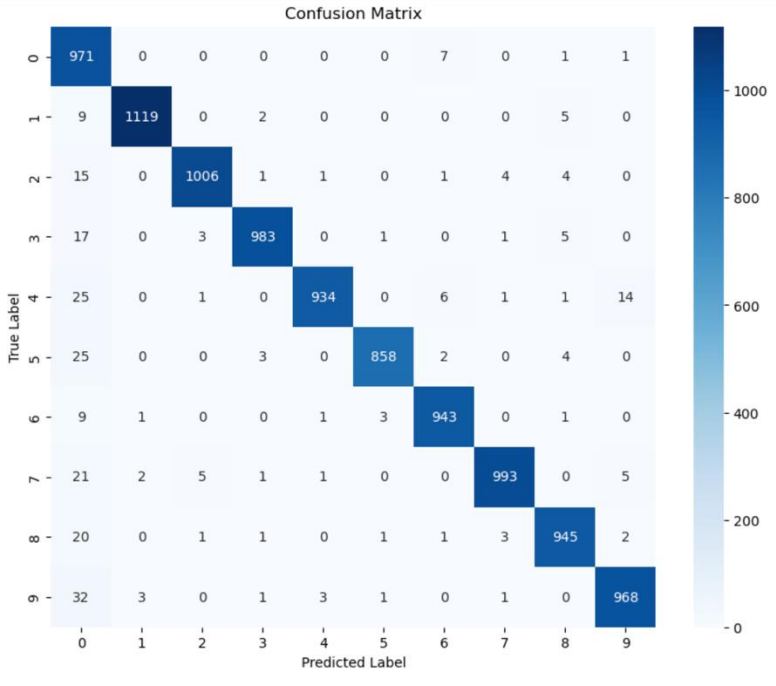
* Somewhat debatable in this example

Accuracy and Confusion Matrix

- model.evaluate(test_images, test_labels) measures performance on unseen test data.
- It returns the test loss and any metrics specified during model.compile(), such as accuracy.
- The test data should not be used in training!
- Next we will look at the confusion matrix

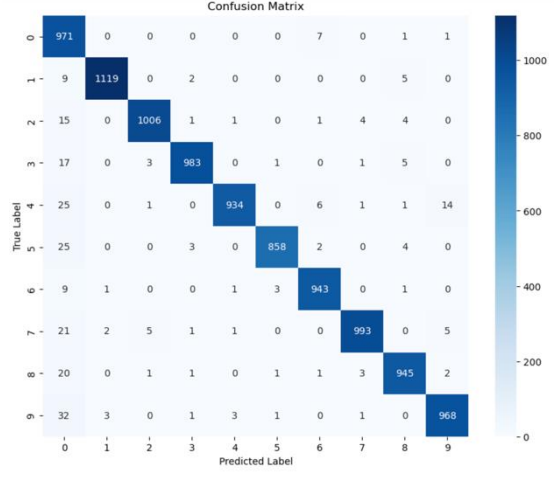
```
# Calculate accuracy on the test set
test_loss, test_acc = model.evaluate(test_images, test_labels)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.3390 - accuracy: 0.9720
```



Confusion Matrix Cont'd

- model.predict(test_images) generates class probabilities for each test image.
- argmax(axis=1) converts probabilities into predicted class labels.
- The confusion matrix compares true vs. predicted labels
- strong performance appears as large values along the diagonal.




```
# Plot confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns
predictions = model.predict(test_images)
predicted_labels = predictions.argmax(axis=1)
cm = confusion_matrix(test_labels, predicted_labels)
plt.figure(figsize=(10, 8))
sns.heatmap(cm,
            annot=True,
            fmt='d',
            cmap='Blues',
            xticklabels=range(10),
            yticklabels=range(10))
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show();
```

What is a hyperparameter?

- They are not learned directly from the data.
- They control the model architecture or training process.
- **Examples:** number of layers, number of neurons, learning rate, batch size, dropout rate, optimizer, and number of epochs.
- **Hyperparameter optimization** searches for settings that improve validation performance.
- In contrast, **model parameters are learned during training**, such as weights and biases.

Example Hyperparameters

```
# Build a MLP
model = keras.Sequential()
model.add(layers.Flatten(input_shape=(28, 28)))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.summary();

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

# Train the model
history = model.fit(train_images,
                    train_labels,
                    epochs=150,
                    batch_size=128,
                    validation_split=0.2)
```

Hyperparameter tuning Example

- Keras Tuner is used to compare fully connected neural networks with different hidden-layer widths and depths, selecting the model with the best validation accuracy.
- max_trials=5 tests only five hyperparameter combinations, while executions_per_trial=2 trains each selected combination twice from scratch to average out random training variation.
- grid search would be more appropriate since there are only 9 combinations in the parameter space

```
# Helper function that builds our model
def build_model(hp):

    neurons = hp.Choice("neurons", values=[256, 512, 1024]) # First hyperparameter
    n_layers = hp.Choice("n_layers", values=[1, 2, 3]) # Second hyperparameter

    model = keras.Sequential()
    model.add(layers.Flatten(input_shape=(28, 28)))  # Flatten the input images
    for i in range(n_layers):
      model.add(layers.Dense(neurons, activation='relu'))  # Hidden layer
    model.add(layers.Dense(10, activation='softmax'))  # Output layer with 10
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

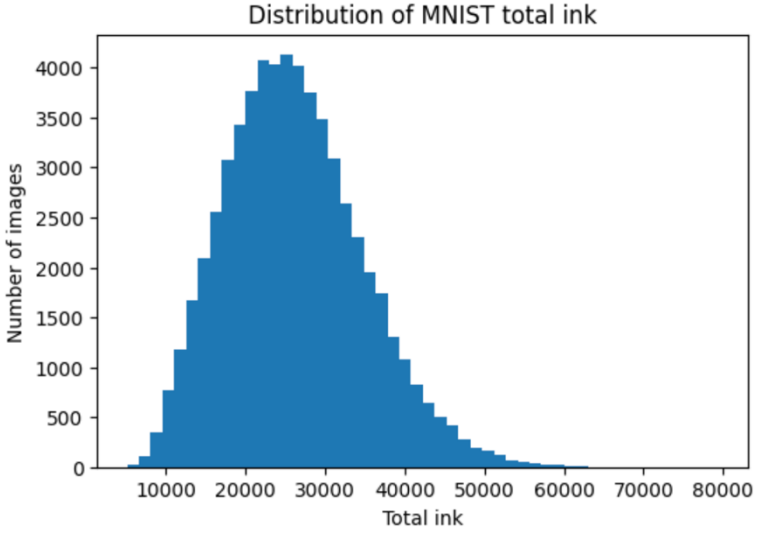
```
import keras_tuner as kt

tuner = kt.RandomSearch(
    build_model,
    objective="val_accuracy",
    max_trials=5,
    executions_per_trial=2,
    directory=TUNING_DIR,
    overwrite=False,
    project_name="n_neuron_search")

tuner.search(train_images,
             train_labels,
             epochs=10,
             validation_split=0.2,
             verbose=1)

best_hp = tuner.get_best_hyperparameters(1)[0]

print(f"Best # neurons per layer: {best_hp}")
```

Classification → Regression

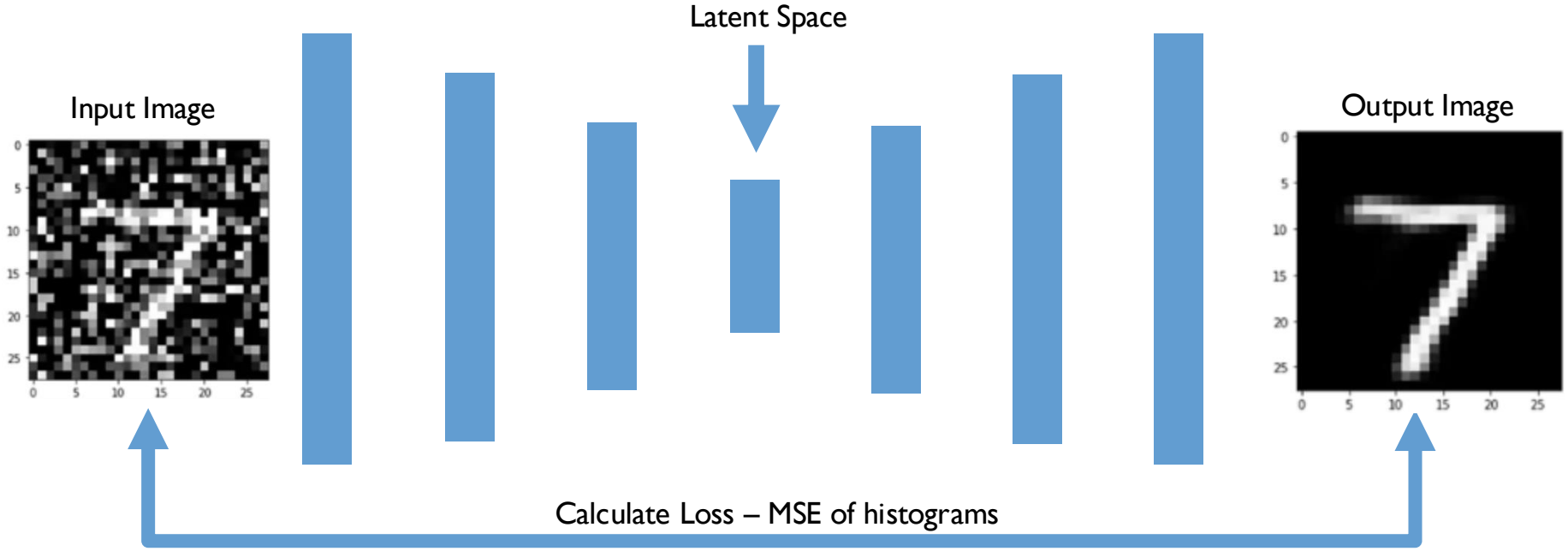
- For classification the loss that we used was cross entropy
  - Classification Target → Class ID
- For regression and other methods such as denoising you will want to use MSE or others
  - Regression Target → Continuous Value

Quick Regression Example

- Take MNIST dataset
  - Calculate some continuous value that we can fit
  - Position, rms, etc.
    - These are already done by the "M" part of MNIST
  - Let's do this with total ink used

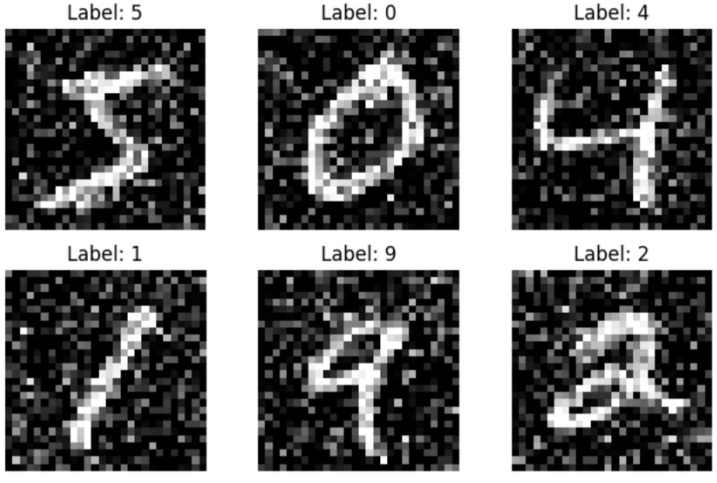


Autoencoder Example



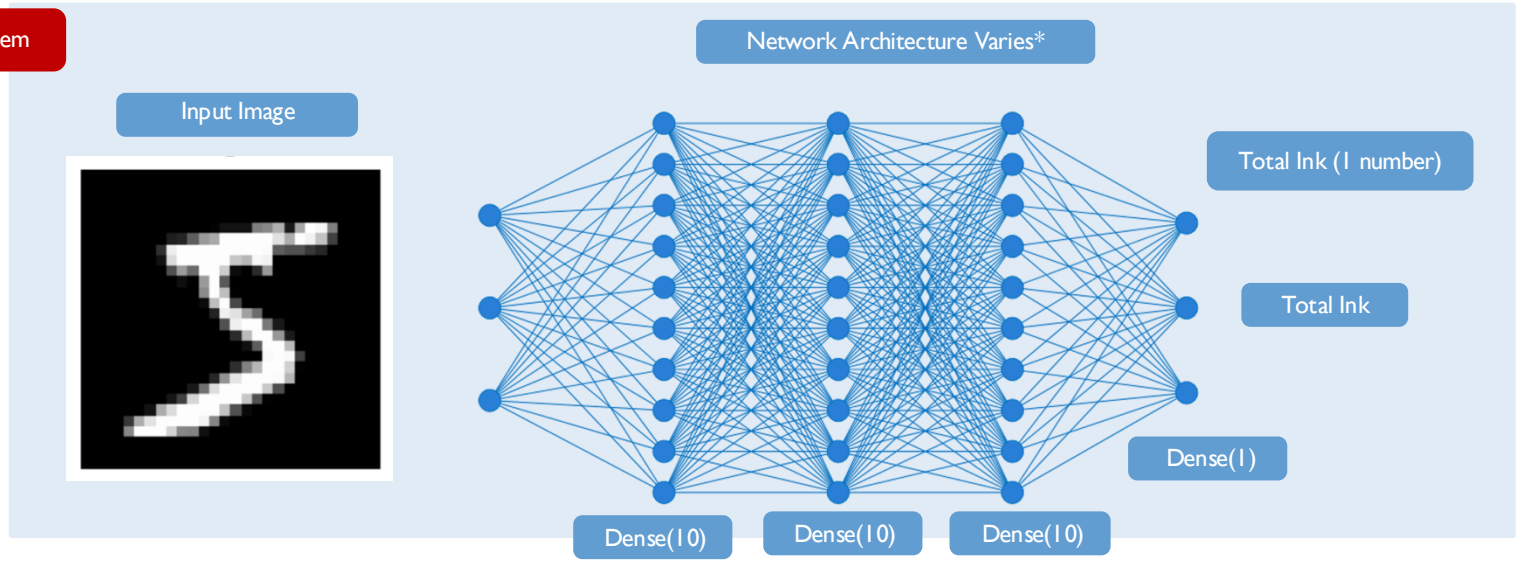
Denoising example

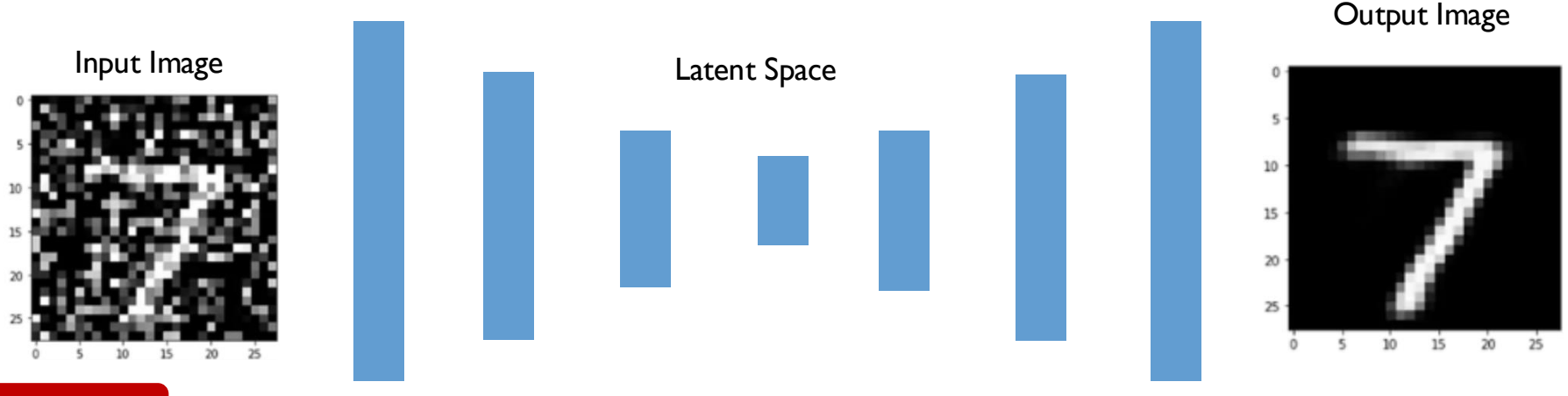
- Take MNIST dataset
- Add some noise
- Noisy Image → Clean Image




```
# Let's add some noise
clean_images = train_images.astype("float32") / 255.0
clean_images = clean_images[..., None]
clean_test_images = test_images.astype("float32") / 255.0
clean_test_images = clean_test_images[..., None]

noise_sigma = 0.3
noisy_images = clean_images + noise_sigma * np.random.normal(size=clean_images.shape)
noisy_images = np.clip(noisy_images, 0, 1)
noisy_test_images = clean_test_images + noise_sigma * np.random.normal(size=clean_test_images.shape)
noisy_test_images = noisy_test_images[..., None]
```

Regression Problem





Denoising Problem

Calculate Loss – MSE of histograms