Introduction to ANNs and Limitations in Image Processing

Course:
INFO-6152 Deep Learning with Tensorflow & Keras 2

Developed by:
Mohammad Noorchenarboo

September 9, 2025

Current Section

1. Course Introduction and Marking Scheme
2. Artificial Neural Networks (ANNs)
3. Using ANNs for Image Data (28×28)
4. Loss of Spatial Structure in ANNs
5. High Number of Parameters in ANNs
6. Lack of Translation Invariance in ANNs

Course Goal

This course **INFO-6152 Deep Learning with TensorFlow & Keras 2** focuses on advanced concepts, architectures, and applications of deep learning. We build upon the foundations of TensorFlow and Keras to explore modern models and their deployment in real-world contexts.

Main Objectives

- Extend understanding of deep learning beyond the basics
- Explore advanced architectures: CNNs, RNNs, LSTMs, GRUs, Attention, Transformers, GANs
- Learn strategies for improving model performance and generalization
- Apply transfer learning and fine-tuning in TensorFlow/Keras
- Understand generative models and their applications
- Gain practical skills in deploying and scaling deep learning systems

Course Goal

Outcome

By the end of this course, students will be able to:

- **Design and train** advanced neural network architectures
- **Implement and fine-tune** models for vision, sequence, and generative tasks
- **Apply** TensorFlow/Keras in solving real-world problems
- **Deploy and optimize** models for efficiency in production settings
- **Critically evaluate** deep learning solutions and trade-offs

Course Assessment Overview

Marking Scheme (100 points total):

Component	Weight (%)
Quizzes	20
In-class Activity	10
Assignments	10
Final Exam	30
Final Project	30

In-class Activities (10%)

Rules

- Students respond to questions after lectures. Each clear and complete answer earns **5 points**.
- Across the course, each student must answer exactly two times.
- One answer must occur **before the Reading Week**, and one **after**.
- Students may volunteer; otherwise, the instructor will select randomly.

Quizzes (20%)

Structure

- Multiple-choice format
- No cheat sheets allowed
- Each quiz has **10 questions**, each worth **0.5 points**
- Total = **5 points per quiz**

Assignments (10%)

Description

- Short, coding-based or written tasks on deep learning topics
- Students complete **two practical in-class activities** applying TensorFlow/Keras
- Each assignment must be **completed and submitted during class hours**
- Activities reinforce advanced deep learning concepts such as transfer learning, regularization, and generative models

Final Exam (30%)

Format

- Multiple-choice questions
- **30 questions**, each worth **1 point**
- One page (two-sided) cheat sheet allowed

Final Project (30%)

Requirements

- Students work **independently** (no groups)
- Must apply TensorFlow and **advanced deep learning techniques** on a real dataset
- Includes both implementation and a short presentation
- Emphasis on creativity, correctness, application, and clarity

Important Notes

Reminder

- All dates for quizzes, assignments, and exams are specified in the course outline.

Current Section

Artificial Neural Networks (ANNs)

Imagine predicting whether a patient has diabetes based on multiple medical features (glucose level, BMI, age, etc.). Traditional linear regression may fail to capture nonlinear interactions.
Central Question: How can we build a model that **learns nonlinear patterns** automatically?

Concept: Artificial Neural Networks

An Artificial Neural Network (ANN) is a **function approximator** inspired by the human brain. It is composed of layers of interconnected computational units called **neurons**.
The mathematical form of a single neuron is:

$$z = \sum_{i=1}^{n} w_i x_i + b, \quad a = f(z)$$

where: x_i = input features, w_i = weights, b = bias, $f(\cdot)$ = activation function introducing nonlinearity.
Common choices of $f(\cdot)$:

- ReLU: $f(z) = \max(0, z)$ (most hidden layers)
- Sigmoid: $f(z) = \frac{1}{1+e^{-z}}$ (binary outputs)
- Softmax: $f(z)_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$ (multiclass outputs)

Artificial Neural Networks (ANNs)

Real-World Examples

- **Medical diagnosis**: Predicting diabetes, heart disease
- **Finance**: Stock price prediction, fraud detection
- **Energy**: Load forecasting in power grids
- **Computer vision**: Recognizing handwritten digits or faces

Common Mistakes

- Using too many neurons → Apply regularization (dropout, L2) to prevent overfitting
- Forgetting to normalize inputs → Scale features for stable convergence
- Using sigmoid in deep hidden layers → Prefer ReLU or its variants to avoid vanishing gradients
- Forgetting softmax for multiclass classification → Recommended to use softmax in the output layer for multiclass problems

Artificial Neural Networks (ANNs)

Caveats

- Neural networks are less interpretable than linear models
- Require large datasets and significant computational resources
- ReLU can cause "dead neurons" (weights stuck at 0)

Neurons as Light Switches with Dimmers

Think of each neuron like a light switch. Inputs are wires carrying signals. The weight decides how much current flows, the bias sets the starting brightness, and the activation function acts as a dimmer knob, deciding whether the light is dim, medium, or fully on.

Get more info → **TensorFlow Sequential API Documentation**

Get more info → **TensorFlow Activation Functions**

Summary: Artificial Neural Networks (ANNs)

Concept/Aspect	**Key Explanation**
Neuron formula	$z = \sum w_i x_i + b$, $a = f(z)$
ANN definition	Function approximator inspired by biological neurons
Role of activation	Introduces nonlinearity (ReLU, sigmoid, softmax)
Real-world examples	Diabetes prediction, fraud detection, energy load forecasting
Pitfalls	Overfitting, poor scaling, vanishing gradients, dead neurons

Current Section

ANNs with Image Inputs

Suppose we want to classify handwritten digits (0–9) from grayscale images of size 28×28. Each image has 784 pixels, each pixel value $\in [0, 255]$.

Central Question: How do we adapt ANNs to process image inputs effectively?

Input Representation

For an image $I \in \mathbb{R}^{28\times 28}$:

$$x = \text{flatten}(I) \in \mathbb{R}^{784}$$

If I is the digit "5", then:

$$x = [x_1, x_2, \ldots, x_{784}]$$

where x_i are pixel intensities.

ANNs with Image Inputs

Define ANN for 28×28 Images

```python
import tensorflow as tf

# Define ANN for MNIST images
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.summary()
# Expected output: Flatten -> Dense(128) -> Dense(10)
```

ANNs with Image Inputs

Numerical Example

- Image: 28×28 pixels $\rightarrow$ 784 inputs
- ANN structure:
  - Hidden layer with 128 neurons (ReLU)
  - Output layer with 10 neurons (softmax)
- Output formula:

$$\hat{y} = \text{softmax}(W_2 \cdot f(W_1 x + b_1) + b_2)$$

  where $f(\cdot)$ is ReLU activation.

Compile the Model

```
# Compile model with optimizer and loss function
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Expected: model is ready for training
```

ANNs with Image Inputs

Train and Evaluate ANN

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

# Normalize pixel values
x_train, x_test = x_train/255.0, x_test/255.0

# Train
model.fit(x_train, y_train, epochs=5, batch_size=32, verbose=1)

# Evaluate
loss, acc = model.evaluate(x_test, y_test, verbose=0)
print("Test Accuracy:", acc) # Expected output: Accuracy ~0.97
```

Flattening Like Reading a Book

Think of the 28×28 image as a page with lines. Flattening is like reading line by line and writing all the words in a single row. The ANN only sees a long sequence, not the spatial arrangement.

Get more info → ANN for Image Data

Summary: ANNs with Image Inputs

Concept/Aspect	**Key Explanation**
Flattening images	Convert 28×28 matrix into 784-length vector
ANN structure	Hidden layer (ReLU) + output softmax for 10 classes
Parameter count	For 128 hidden neurons: 100,480 parameters
Best practices	Normalize pixel values, use ReLU + softmax
Limitations	Spatial info lost when flattening
TensorFlow workflow	Define $\rightarrow$ Compile $\rightarrow$ Train $\rightarrow$ Evaluate

Current Section

Loss of Spatial Structure in ANNs

When an image is fed into a basic ANN, it is flattened into a vector of length 784 (for a 28×28 image).
Problem: Flattening destroys 2D relationships.

- A pixel from the left eye of a face may end up numerically next to a pixel from the chin.
- The model loses the fact that they were far apart in the original image.

Mathematical View

Flattening converts:

$$I \in \mathbb{R}^{28\times28}$$

into:

$$x = \text{flatten}(I) \in \mathbb{R}^{784}$$

The ANN processes x as:

$$h = f(Wx + b)$$

Here, W has no knowledge that x_{37} and x_{38} were neighbors in 2D, or that x_{37} and x_{742} were far apart. All distances and neighborhoods are lost.

Loss of Spatial Structure in ANNs

TensorFlow Shape Inspection

```python
import tensorflow as tf

# Example input: 28x28 image
sample_input = tf.random.normal([1, 28, 28])

# After flattening
flatten_layer = tf.keras.layers.Flatten()
flattened = flatten_layer(sample_input)

print("Original shape:", sample_input.shape)
print("Flattened shape:", flattened.shape)
# Expected:
# Original shape: (1, 28, 28)
# Flattened shape: (1, 784)
```

Loss of Spatial Structure in ANNs

Numerical Example

- Pixel $(10, 10)$ might represent part of an eye.
- Pixel $(10, 11)$ is its right neighbor, crucial for detecting the edge of the eye.

After flattening:

$$x_{(10,10)} \mapsto x_{130}, \quad x_{(10,11)} \mapsto x_{131}$$

But pixel $(11, 10)$ (just below the eye) becomes:

$$x_{158}$$

So the ANN treats it as unrelated, even though it is spatially connected in 2D.

Loss of Spatial Structure in ANNs

Mistakes

- Assuming ANNs can detect edges or shapes after flattening
- Believing proximity is preserved: only row-wise order is preserved, not 2D spatial locality

Warnings

- For small toy problems (like digit classification), ANNs might still work
- For complex images (faces, objects), this limitation makes ANNs ineffective

Pixels Like Scrambled Words

Imagine reading a book where all the words are taken out of order and placed in a single line. The ANN sees the "words" (pixels), but it cannot understand sentences (shapes/edges) because the order and grouping are gone.

Get more info → **CS231n: Convolutional Neural Networks for Visual Recognition**

Summary: Loss of Spatial Structure in ANNs

Concept/Aspect	Key Explanation
Flattening	Converts 28×28 into a 784-length vector
Loss of spatial structure	Neighbors in 2D may not stay neighbors in 1D
Effect	ANN cannot naturally detect edges, shapes, or patterns
Consequence	Poor scaling for larger/complex image tasks
TensorFlow demo	Flatten layer shows input shape $(28, 28) \rightarrow$ output shape (784)

Current Section

High Number of Parameters in ANNs

Images have thousands of pixels. In a fully connected ANN, each neuron in a layer connects to every neuron in the next.
Problem: This creates a massive number of parameters (weights + biases), which are expensive to train and prone to overfitting.

Parameter Count Formula

For a fully connected layer:

$$\#\text{params} = (\text{input\_units} \times \text{output\_units}) + \text{output\_units}$$

where the second term corresponds to biases.

High Number of Parameters in ANNs

TensorFlow Parameter Count Demo

```python
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.summary()
# Expected:
# Flatten -> Dense(128): 100,480 params
# Dense(10): 1,290 params
# Total params: 101,770
```

High Number of Parameters in ANNs

Numerical Example: Two-Layer ANN

- Input: $28 \times 28 = 784$ pixels
- Hidden layer (128 neurons):

$$784 \times 128 + 128 = 100{,}480$$

- Output layer (10 neurons):

$$128 \times 10 + 10 = 1{,}290$$

- Total parameters:

$$100{,}480 + 1{,}290 = 101{,}770$$

- For higher resolutions (e.g., $256 \times 256 = 65{,}536$ inputs), parameters become astronomical.

High Number of Parameters in ANNs

Mistakes

- Ignoring parameter growth when scaling image size
- Training huge fully connected models on limited data $\rightarrow$ severe overfitting

Warnings

- Larger input images make fully connected ANNs computationally infeasible
- Even if training succeeds, such networks generalize poorly

Too Many Wires in a City

Imagine every house in a city being connected to every other house with its own wire. For a small village this might work, but for a big city the wiring becomes unmanageable. Similarly, in ANNs, as image size grows, the number of weights explodes.

Get more info $\rightarrow$ Deep Learning Book (Goodfellow et al.)

Summary: High Number of Parameters in ANNs

Concept/Aspect	Key Explanation
Parameter formula	(inputs $\times$ outputs) $+$ outputs
Example (28×28)	ANN with 128 hidden + 10 output neurons $\rightarrow$ 101,770 params
Problem	Parameter count grows linearly with input size
Consequence	Slow training, high memory use, overfitting risk
TensorFlow demo	`model.summary()` shows parameter explosion

Current Section

Lack of Translation Invariance in ANNs

A standard ANN treats each input pixel as independent, tied to its fixed position.
Problem: If the object shifts in the image, the ANN cannot recognize it as the same object.

Mathematical View

Suppose $x \in \mathbb{R}^{784}$ is the flattened 28×28 image.

- Pixel (i,j) maps to x_k where $k = i \cdot 28 + j$
- If the object shifts right by 1 pixel, each input moves to a different index x_{k+1}

The ANN computes:

$$h = f(Wx + b)$$

But W is position-specific. A pattern at x_{100} is treated completely differently from the same pattern at x_{101}.

Lack of Translation Invariance in ANNs

TensorFlow: Shifted Inputs Demonstration

```python
import tensorflow as tf

# Simulate a "digit" as a 1D tensor with a peak
x_center = tf.concat([tf.zeros(5), tf.ones(3), tf.zeros(20)], axis=0)
x_shifted = tf.concat([tf.zeros(6), tf.ones(3), tf.zeros(19)], axis=0)

print("Center index of peak:", tf.argmax(x_center).numpy())
print("Shifted index of peak:", tf.argmax(x_shifted).numpy())
# Expected:
# Center index of peak: 5
# Shifted index of peak: 6
```

Lack of Translation Invariance in ANNs

Numerical Example

- Digit “7” centered $\rightarrow$ activates pixels x_{300} to x_{330}
- Shifted right $\rightarrow$ activates x_{301} to x_{331}
- ANN must learn two different weight sets for the same digit

If there are k possible shifts, the ANN must effectively memorize k different versions of the same object.

Mistakes

- Assuming a trained ANN can recognize objects anywhere in the image
- Believing deeper layers automatically solve translation invariance

Lack of Translation Invariance in ANNs

Warnings

- Lack of translation invariance leads to data inefficiency
- The network must see the same object in many positions during training
- Especially harmful for larger images where objects can appear anywhere

Recognizing Faces Only at One Spot

Imagine teaching someone to recognize a face but only if it is printed in the center of a page. If you shift the photo slightly left, they claim it is a new, unfamiliar image. That is how ANNs treat translations.

Get more info → **Convolutions for Images and Translation Invariance**

Summary: Lack of Translation Invariance in ANNs

Concept/Aspect	**Key Explanation**
Translation invariance	ANN does not naturally recognize shifted objects
Mathematical reason	Different pixel indices $\rightarrow$ different weights
Effect	Network must learn multiple redundant patterns
Consequence	Poor generalization and inefficiency
TensorFlow demo	Shifted inputs show index changes of activations