

```
# basic
import numpy as np
import pandas as pd

#visulas
import matplotlib.pyplot as plt
import seaborn as sns

#Scikit-learn
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split

# tensorflow
import tensorflow as tf
from tensorflow import keras
from keras.datasets import fashion_mnist
from keras.utils import to_categorical
# Instead of 'from keras.preprocessing.image import ImageDataGenerator'
from tensorflow.keras.preprocessing.image import ImageDataGenerator # Import ImageDataGenerator from tensorflow.keras.preprocessing.image
from tensorflow.keras import models
from keras.layers import Dense, MaxPool2D, Conv2D, BatchNormalization, Flatten, Dropout
```

✓ Binary Step Function

$$f(x) = 1, x \geq 0, x < 0$$

```
def binary_step(x):
    if x < 0:
        return 0
    else:
        return 1
binary_step(-1)
```

→ 0

✓ Linear Function

$$f(x) = ax$$

```
def linear_function(x):
    return 4*x
linear_function(4), linear_function(-2)
```

→ (16, -8)

✓ Sigmoid

The next activation function that we are going to look at is the Sigmoid function. It is one of the most widely used non-linear activation function. Sigmoid transforms the values between the range 0 and 1. Here is the mathematical expression for sigmoid- $f(x) = 1/(1+e^{-x})$

```
import numpy as np
def sigmoid_function(x):
    z = (1/(1 + np.exp(-x)))
    return z
sigmoid_function(7), sigmoid_function(-22)
```

→ (0.9990889488055994, 2.7894680920908113e-10)

✓ Tanh

The tanh function is very similar to the sigmoid function. The only difference is that it is symmetric around the origin. The range of values in this case is from -1 to 1. Thus the inputs to the next layers will not always be of the same sign. The tanh function is defined as-

$$\tanh(x) = 2\text{sigmoid}(2x) - 1$$

$$\tanh(x) = 2/(1+e^{-(2x)}) - 1$$

```
def tanh_function(x):
    z = (2/(1 + np.exp(-2*x))) - 1
```

```

    return z
tanh_function(0.5), tanh_function(-1)

↗ (0.4621171572600098, -0.7615941559557649)

```

✓ ReLU - Rectified Linear Unit

The ReLU function is another non-linear activation function. The main advantage is that it does not activate all the neurons at the same time. This means that the neurons will only be deactivated if the output of the linear transformation is less than 0. $f(x)=\max(0,x)$ Negative input values \rightarrow the result is zero, that means the neuron does not get activated. Since only a certain number of neurons are activated, the ReLU function is far more computationally efficient when compared to the sigmoid and tanh function.

```

def relu_function(x):
    if x<0:
        return 0
    else:
        return x
relu_function(7), relu_function(-7)

```

```

↗ (7, 0)

```

✓ Leaky ReLU

Leaky ReLU function is an improved version of the ReLU function. As we saw that for the ReLU function, the gradient is 0 for $x<0$, which would deactivate the neurons in that region. Leaky ReLU is defined to address this problem. Instead of defining the Relu function as 0 for negative values of x , we define it as an extremely small linear component of x .

Here is the mathematical expression- $f(x)= 0.01x, x<0 = x, x>=0$

```

def leaky_relu_function(x):
    if x<0:
        return 0.01*x
    else:
        return x
leaky_relu_function(7), leaky_relu_function(-7)

```

```

↗ (7, -0.07)

```

Parameterised ReLU

The parameterised ReLU introduces a new parameter as a slope of the negative part of the function.

Here's how the ReLU function is modified to incorporate the slope parameter- $f(x) = x, x>=0$

$= ax, x<0$

When the value of a is fixed to 0.01, the function acts as a Leaky ReLU function. However, in case of a parameterised ReLU function, ' a ' is also a trainable parameter . The network also learns the value of ' a ' for faster and more optimum convergence.

✓ Exponential Linear Unit - ELU

Exponential Linear Unit a variant of Rectified Linear Unit (ReLU) that modifies the slope of the negative part of the function. Unlike the leaky relu and parametric ReLU functions, instead of a straight line, ELU uses a log curve for defining the negative values. It is defined as

$f(x) = x, x>=0 = a(e^x-1), x<0$

```

def elu_function(x, a):
    if x<0:
        return a*(np.exp(x)-1)
    else:
        return x
elu_function(5, 0.1), elu_function(-5, 0.1)

```

```

↗ (5, -0.09932620530009145)

```

✓ Swish

```
def swish_function(x):
    return x/(1+np.exp(-x))
swish_function(-67), swish_function(4)
```

→ (5.349885844610276e-28, 4.074629441455096)

▼ Softmax

```
def softmax_function(x):
    z = np.exp(x)
    z_ = z/z.sum()
    return z_
softmax_function([0.8, 1.2, 3.1])
```

→ array([0.08021815, 0.11967141, 0.80011044])

▼ All Activation Functions:

ReLU (Rectified Linear Unit)

Sigmoid

Tanh

Softmax

Leaky ReLU

ELU (Exponential Linear Unit)

Swish

```
# load data
(train,train_label),(test,test_label) = fashion_mnist.load_data()
```

→ Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz> 29515/29515 — 0s 0us/step
 Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz> 26421880/26421880 — 0s 0us/step
 Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz> 5148/5148 — 0s 0us/step
 Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz> 4422102/4422102 — 0s 0us/step

```
print('The shape of train',train.shape)

print('The shape of train_label',train_label.shape)

print('The shape of test',test.shape)

print('The shape of test_label',test_label.shape)
```

→ The shape of train (60000, 28, 28)
 The shape of train_label (60000,)
 The shape of test (10000, 28, 28)
 The shape of test_label (10000,)

```
plt.figure(figsize=(20,20)) # specifying the overall grid size
plt.subplots_adjust(hspace=0.4)

labels = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

for i in range(100):

    plt.subplot(10,10,i+1) # the number of images in the grid is 5*5 (25)
    plt.imshow(train[i],cmap='Blues')
    plt.title(labels[int(train_label[i])],fontsize=12)
    plt.axis('off')


plt.show()
```



```
label,count = np.unique(train_label,return_counts=True)
uni = pd.DataFrame(data=count,index=labels,columns=['Count'])

plt.figure(figsize=(14,4),dpi=200)
```

```
sns.barplot(data=uni,x=uni.index,y='Count',palette='icefire',width=0.5).set_title('Class distribution in training set',fontsize=15)
plt.show()
```

 <ipython-input-5-a701c8cdf6d9>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le`

```
sns.barplot(data=uni,x=uni.index,y='Count',palette='icefire',width=0.5).set_title('Class distribution in training set',fontsize=15)
```



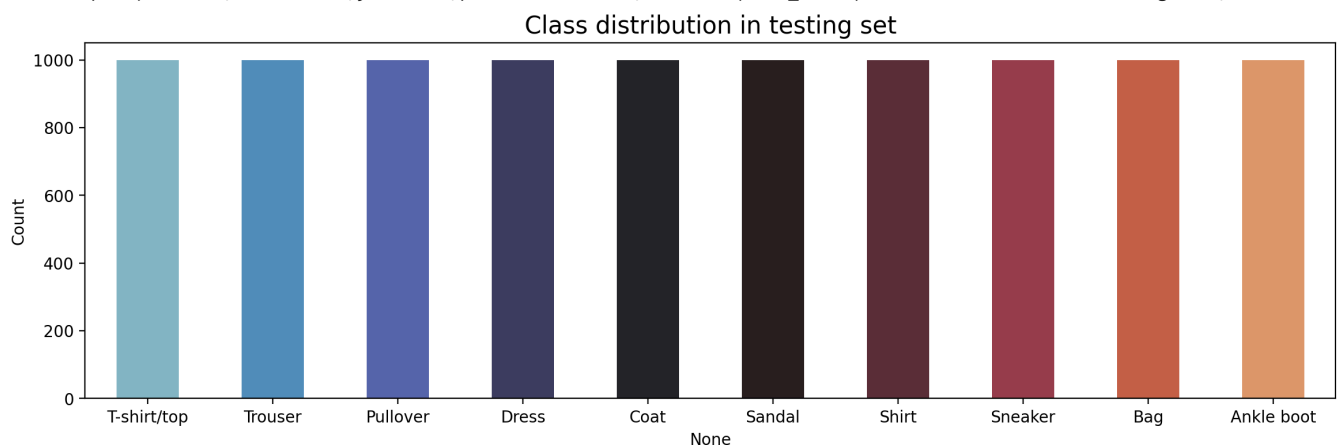
```
# count for Testing dataset
label,count = np.unique(test_label,return_counts=True)
uni = pd.DataFrame(data=count,index=labels,columns=['Count'])
```

```
plt.figure(figsize=(14,4),dpi=200)
sns.barplot(data=uni,x=uni.index,y='Count',palette='icefire',width=0.5).set_title('Class distribution in testing set',fontsize=15)
plt.show()
```

 <ipython-input-6-36c2ae3643c0>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le`

```
sns.barplot(data=uni,x=uni.index,y='Count',palette='icefire',width=0.5).set_title('Class distribution in testing set',fontsize=15)
```



```
# function to change the data type and normalize the data
# scaling the images
def pre_process(data):
    return data/255
```

```
# function to chnagne the labels into categorical data
def to_cat(data):
    return to_categorical(data,num_classes=10)
```

```

train = pre_process(train)
test = pre_process(test)

print('The shape of train_label before one hot encoding',train_label.shape)

train_label = to_cat(train_label)

print('The shape of y_train after one hot encoding',train_label.shape)

↩ The shape of train_label before one hot encoding (60000,)
  The shape of y_train after one hot encoding (60000, 10)

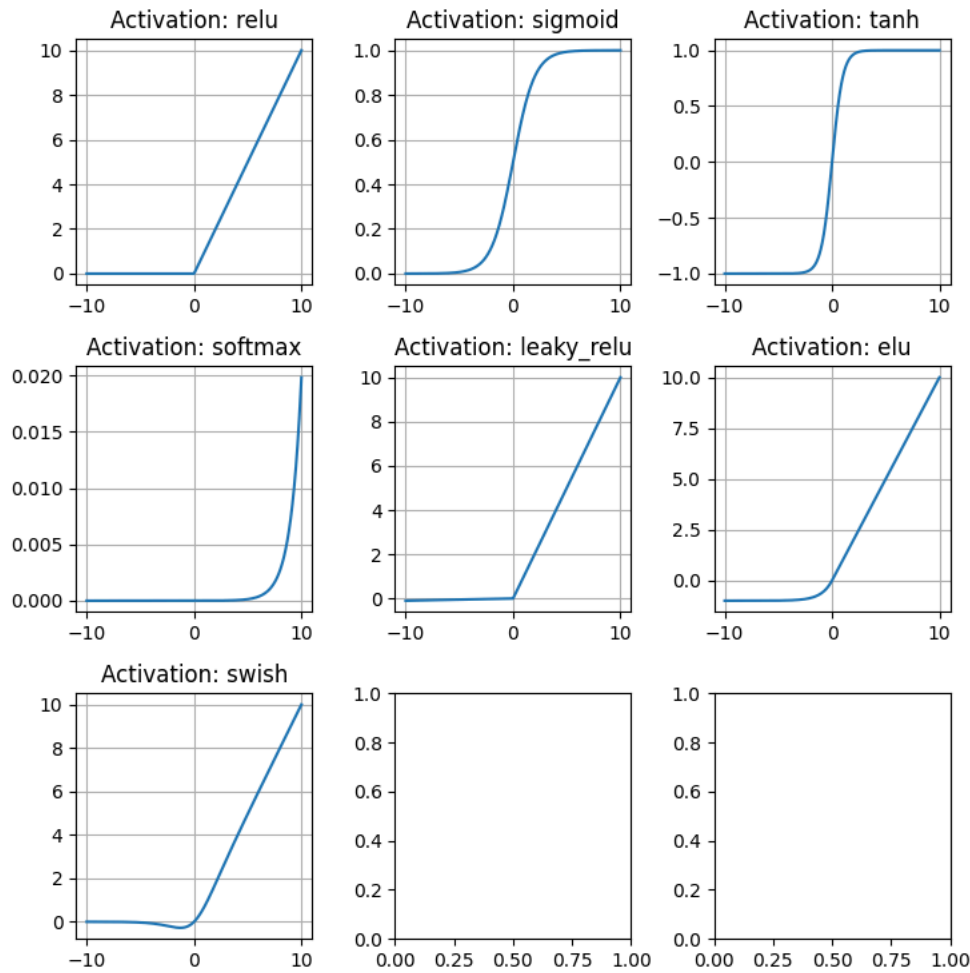
train = train.reshape(train.shape[0], 28, 28, 1).astype('float32')
test = test.reshape(test.shape[0], 28, 28, 1).astype('float32')

# Define the model
def create_model(activation_function):
    model = models.Sequential()
    # First hidden layer with the specified activation function
    model.add(layers.Dense(64, activation=activation_function, input_shape=(784,)))
    # Second hidden layer with the same activation function
    model.add(layers.Dense(64, activation=activation_function))
    # Output layer (softmax for classification)
    model.add(layers.Dense(10, activation='softmax')) # 10 classes for MNIST
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    return model

# Activation functions to demonstrate
activation_functions = ['relu', 'sigmoid', 'tanh', 'softmax', 'leaky_relu', 'elu', 'swish']
# Create a subplot to visualize the activation functions
fig, axes = plt.subplots(3, 3, figsize=(7.5,7.5))
axes = axes.flatten()
# Plot and demonstrate each activation function
for idx, activation in enumerate(activation_functions):
    ax = axes[idx]
    # Generate an input range from -10 to 10 for plotting
    x = np.linspace(-10, 10, 1000)
    if activation == 'relu':
        y = np.maximum(0, x) # ReLU function
    elif activation == 'sigmoid':
        y = 1 / (1 + np.exp(-x)) # Sigmoid function
    elif activation == 'tanh':
        y = np.tanh(x) # Tanh function
    elif activation == 'softmax':
        # Softmax is not typically used in isolation for a single input, so we simulate it
        y = np.exp(x) / np.sum(np.exp(x)) # Softmax function (simplified)
    elif activation == 'leaky_relu':
        y = np.where(x > 0, x, 0.01 * x) # Leaky ReLU function
    elif activation == 'elu':
        y = np.where(x > 0, x, np.exp(x) - 1) # ELU function
    elif activation == 'swish':
        y = x * (1 / (1 + np.exp(-x))) # Swish function
    # Plot the activation function
    ax.plot(x, y)
    ax.set_title(f"Activation: {activation}")
    ax.grid(True)
plt.tight_layout()
plt.show()

```



```
import tensorflow as tf
from tensorflow.keras.layers import Activation

# ReLU (Rectified Linear Unit)
def relu_layer(input_layer):
    return Activation('relu')(input_layer)

# Sigmoid
def sigmoid_layer(input_layer):
    return Activation('sigmoid')(input_layer)

# Tanh
def tanh_layer(input_layer):
    return Activation('tanh')(input_layer)

# Softmax
def softmax_layer(input_layer):
    return Activation('softmax')(input_layer)

# Leaky ReLU
def leaky_relu_layer(input_layer, alpha=0.01):
    return tf.keras.layers.LeakyReLU(alpha=alpha)(input_layer)

# ELU (Exponential Linear Unit)
def elu_layer(input_layer, alpha=1.0):
    return tf.keras.layers.ELU(alpha=alpha)(input_layer)

# Swish
def swish_layer(input_layer):
    return Activation(tf.nn.swish)(input_layer)

from tensorflow.keras.layers import Input, Dense

# Example input layer
input_layer = Input(shape=(784,)) # Example shape, adjust as needed

# Apply ReLU activation
x = relu_layer(input_layer)

# Add other layers and apply other activations
```

```

x = Dense(128)(x)
x = sigmoid_layer(x)

# Final output layer with softmax (e.g., for multi-class classification)
output_layer = softmax_layer(x)

# Build model
model = tf.keras.Model(inputs=input_layer, outputs=output_layer)
model.summary()

```

Model: "functional"

| Layer (type) | Output Shape | Param # |
|----------------------------|--------------|---------|
| input_layer_1 (InputLayer) | (None, 784) | 0 |
| activation (Activation) | (None, 784) | 0 |
| dense (Dense) | (None, 128) | 100,480 |
| activation_1 (Activation) | (None, 128) | 0 |
| activation_2 (Activation) | (None, 128) | 0 |

Total params: 100,480 (392.50 KB)
 Trainable params: 100,480 (392.50 KB)

```

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.layers import Dense, Flatten, Activation, LeakyReLU, ELU
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential

# Load and preprocess data
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0 # Normalize
train_labels = to_categorical(train_labels, 10)
test_labels = to_categorical(test_labels, 10)

# Define a function to create models with different activation functions
def create_model(activation):
    model = Sequential([
        Flatten(input_shape=(28, 28)),
        Dense(128),
        activation,
        Dense(64),
        activation,
        Dense(10, activation='softmax') # Final layer with softmax for multi-class classification
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Define a dictionary of activation layers
activation_layers = {
    'ReLU': Activation('relu'),
    'Sigmoid': Activation('sigmoid'),
    'Tanh': Activation('tanh'),
    'Softmax': Activation('softmax'),
    'Leaky ReLU': LeakyReLU(alpha=0.01),
    'ELU': ELU(alpha=1.0),
    'Swish': Activation(tf.nn.swish)
}

# Train and evaluate each model
results = {}
for name, activation in activation_layers.items():
    print(f"\nTraining model with {name} activation...")
    model = create_model(activation)
    model.fit(train_images, train_labels, epochs=5, batch_size=128, verbose=0)
    test_loss, test_accuracy = model.evaluate(test_images, test_labels, verbose=0)
    results[name] = test_accuracy
    print(f"Test accuracy with {name}: {test_accuracy:.4f}")

# Plotting the test accuracy of each activation function
plt.figure(figsize=(10, 6))
plt.bar(results.keys(), results.values(), color='skyblue')
plt.xlabel('Activation Function')
plt.ylabel('Test Accuracy')
plt.title('Test Accuracy Comparison for Different Activation Functions')
plt.ylim(0, 1)

```



```
plt.xticks(rotation=45)  
plt.show()
```



```
Training model with ReLU activation...  
Test accuracy with ReLU: 0.8691
```

```
Training model with Sigmoid activation...  
Test accuracy with Sigmoid: 0.8617
```

```
Training model with Tanh activation...  
Test accuracy with Tanh: 0.8790
```

```
Training model with Softmax activation...  
Test accuracy with Softmax: 0.6612
```

```
Training model with Leaky ReLU activation...  
Test accuracy with Leaky ReLU: 0.8443
```

```
Training model with ELU activation...  
Test accuracy with ELU: 0.8684
```

```
Training model with Swish activation...  
Test accuracy with Swish: 0.8748
```

