gen ai

p1---

```
# P1: Activation functions + derivatives (plots for each)

import numpy as np, matplotlib.pyplot as plt

x=np.linspace(-6,6,2000)

acts={

 "sigmoid":(lambda x:(s:=1/(1+np.exp(-x)),s*(1-s))),

 "tanh":   (lambda x:(t:=np.tanh(x),1-t**2)),

 "relu":   (lambda x:(r:=np.maximum(0,x),(x>0).astype(float))),

 "leaky":  (lambda x,a=0.025:(np.where(x>0,x,a*x),np.where(x>0,1,a))),

 "prelu":  (lambda x,a=0.25:(np.where(x>0,x,a*x),np.where(x>0,1,a))),

 "elu":    (lambda x,a=1.0:(np.where(x>=0,x,a*(np.exp(x)-1)),np.where(x>=0,1,a*np.exp(x)))),

 "softplus":(lambda x:(np.log1p(np.exp(x)),1/(1+np.exp(-x))))

 "leaky": (lambda: x,a=0.25:(np.where(x>0,x,a*x))

}

for name, func in acts.items():

    y, dy = func(x)

    plt.plot(x, y)

    plt.plot(x, dy)

    plt.axhline(0, color='black')

    plt.axvline(0, color='black')

    plt.title(name.upper())

    plt.show()
```

p2--

```
import tensorflow as tf

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Flatten, Dense

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import classification_report, confusion_matrix


# 1. Load and prepare the data

(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
# 2. Build the model
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

# 4. Train the model
model.fit(x_train, y_train, epochs=5)

# 5. Evaluate the model
print("\nEvaluating on test data:")
model.evaluate(x_test, y_test)

# 6. Make predictions and generate reports
predictions = model.predict(x_test)
y_pred = np.argmax(predictions, axis=1)

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# 7. Visualize results
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.imshow(x_test[i], cmap='gray')
    plt.title(f"Pred: {y_pred[i]}, Act: {y_test[i]}")
```

```
    plt.axis('off')
plt.tight_layout()
plt.show()

p3----------------------------------------------------------
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
import matplotlib.pyplot as plt
import numpy as np

# 1. Load and prepare the data
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Define the optimizers you want to compare
optimizers_to_test = ['sgd', 'adam', 'rmsprop', 'adagrad']
results = {}
history_dict = {} # ADDED: Dictionary to store training history for the plot

# 2. Loop through each optimizer
for optimizer_name in optimizers_to_test:
    print(f"\n--- Training with {optimizer_name.upper()} optimizer ---")

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

    model.compile(optimizer=optimizer_name,
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    history = model.fit(x_train, y_train,
```

```
                epochs=4, validation_split=0.1)

    # ADDED: Store the history for this optimizer
    history_dict[optimizer_name] = history

    # Evaluate the model and store the final test accuracy
    loss, accuracy = model.evaluate(x_test, y_test)
    results[optimizer_name] = accuracy
    print(f"Final Test Accuracy: {accuracy}")

# 3. Print a final summary
print("\n--- Final Results ---")
best_optimizer = max(results, key=results.get)
best_optimizer

# Simplified plot focusing on the main comparison metric
plt.figure(figsize=(10, 6))
for name, history in history_dict.items():
    plt.plot(history.history['val_accuracy'], label=name)

plt.title('Optimizer Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```

p4---

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from tensorflow.keras.callbacks import EarlyStopping

# 1. Load and prepare the MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

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


# 2. Build a simple Convolutional Neural Network (CNN)
model = tf.keras.models.Sequential([
    # Add a channel dimension for the CNN and define the input shape
    tf.keras.layers.Input(shape=(28, 28, 1)),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax') # 10 classes for digits 0-9
])
model.summary()
# 3. Compile the model
# Use 'sparse_categorical_crossentropy' because labels are integers (not one-hot encoded)
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
earlystop = EarlyStopping(monitor='val_loss',patience=3,restore_best_weights=True)


# 4. Train the model
print("--- Starting Model Training ---")
history = model.fit(x_train, y_train, epochs=3, validation_split=0.1,callbacks=[earlystop])
print("--- Training Finished ---")


# 5. Evaluate the model and print a classification report
print("\n--- Evaluating Model ---")
loss, acc = model.evaluate(x_test, y_test, verbose=0)
print(f"Test Accuracy: {acc:.4f}")


y_pred_probs = model.predict(x_test)
y_pred = np.argmax(y_pred_probs, axis=1)


print("\n--- Classification Report ---")
print(classification_report(y_test, y_pred))
```

```
# 6. Show sample predictions
for i in range(10):
    plt.subplot(3, 4, i+1)
    plt.imshow(x_test[i], cmap='gray')
    plt.title(f"Pred: {y_pred[i]} | True: {y_test[i]}")
    plt.axis('off')
plt.tight_layout()
plt.show()

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['accuracy','val_accuracy','loss','val_loss'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
```

p5--

```
import torch, torchvision
from torchvision import transforms
from PIL import Image
import matplotlib.pyplot as plt

# Models
seg = torchvision.models.segmentation.fcn_resnet50(weights='DEFAULT').eval()
det = torchvision.models.detection.fasterrcnn_resnet50_fpn(weights='DEFAULT').eval()

# Image
img = Image.open("/content/Google_AI_Studio_2025-08-31T07_30_03.481Z.png").convert("RGB")
t = transforms.ToTensor()(img)

# Predictions
with torch.no_grad():
    seg_out = seg(t.unsqueeze(0))['out'].argmax(1).squeeze()
```

```
    boxes = det([t])[0]['boxes']

# Plot
plt.imshow(img); plt.title("Original"); plt.axis("off"); plt.show()
plt.imshow(seg_out); plt.title("Segmentation"); plt.axis("off"); plt.show()
plt.imshow(img); plt.title("Detection"); plt.axis("off")
for x1, y1, x2, y2 in boxes:
    plt.gca().add_patch(plt.Rectangle((x1,y1),x2-x1,y2-y1,fill=False,color='r'))
plt.show()
```

p6--

```
#CATS AND DOG
import tensorflow as tf, os, zipfile, matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16

# Download & extract dataset
url="https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip"
path=tf.keras.utils.get_file("cats_and_dogs.zip",url,cache_dir=os.getcwd())
with zipfile.ZipFile(path,"r") as z: z.extractall(os.getcwd())

train_dir=os.path.join(os.getcwd(),"cats_and_dogs_filtered/train")
val_dir=os.path.join(os.getcwd(),"cats_and_dogs_filtered/validation")

# Data generators
train_gen=ImageDataGenerator(rescale=1./255,rotation_range=20,width_shift_range=0.2,
    height_shift_range=0.2,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
val_gen=ImageDataGenerator(rescale=1./255)

train_ds=train_gen.flow_from_directory(train_dir,batch_size=20,class_mode="binary",target_size=(150,150))
val_ds=val_gen.flow_from_directory(val_dir,batch_size=20,class_mode="binary",target_size=(150,150))

# Model
base=VGG16(weights="imagenet",include_top=False,input_shape=(150,150,3))
base.trainable=False
```

```
model=tf.keras.Sequential([
    base,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256,activation="relu"),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1,activation="sigmoid")
])
model.compile(loss="binary_crossentropy",optimizer=tf.keras.optimizers.RMSprop(2e-5),metrics=["accuracy"])

# Train
hist=model.fit(train_ds,steps_per_epoch=100,epochs=30,validation_data=val_ds,validation_steps=50)

# Predictions
x,y=next(val_ds)
preds=model.predict(x)
class_names=["cat","dog"]
plt.figure(figsize=(12, 12))
for i in range(len(x)):
    plt.subplot(4, 5, i+1)  # 4 rows × 5 cols = 20 images (batch_size=20)
    plt.imshow(x[i])
    plt.axis("off")
    pred = class_names[int(preds[i][0]>0.5)]
    true = class_names[int(y[i])]
    plt.title(f'P:{pred} T:{true}')
plt.tight_layout()
plt.show()

# Plot metrics
plt.plot(hist.history["accuracy"],label="Train Acc")
plt.plot(hist.history["val_accuracy"],label="Val Acc")
plt.plot(hist.history["loss"],label="Train Loss")
plt.plot(hist.history["val_loss"],label="Val Loss")
plt.legend()
plt.show()
```

p7---

```
import cv2

from ultralytics import YOLO

model = YOLO("yolov8n.pt")

# 2. Set OpenCV to use an optimized code path and set number of threads

cv2.setUseOptimized(True)

cv2.setNumThreads(4)

# 3. Run prediction on the webcam (source="0") and display the results.

print("Starting webcam feed. Press 'q' to quit.")

model.predict(source="0", show=True)

#YOLO

import cv2

from ultralytics import YOLO

model = YOLO("yolov8n.pt")

cap = cv2.VideoCapture(0)

while True:

    ret, frame = cap.read()

    results = model(frame)

    cv2.imshow("YOLO", results[0].plot())

    if cv2.waitKey(1) & 0xFF == ord('q'):

        break

cap.release()

cv2.destroyAllWindows()
```