Sr. No	Title	Pg.no	Date	Sign
1	Implement basic operations with Scalars, Vectors, Matrices, and Tensors using NumPy.			
1 a	Perform matrix multiplication and vector operations			
1 b	Compute norms of vectors and matrices.			
2	Explore issues like overflow, underflow, and poor conditioning in numerical computation.			
2 a	Implement Gradient-Based Optimization algorithms (e.g., Gradient Descent) for simple optimization problems.			
2 b	Solve Constraint Optimization problems using optimization techniques.			
3	Implementing deep neural network for performing binary classification task. Solving XOR problem using deep feed forward network.			
4	Build and train a Multilayer Perceptron (MLP) for a classification task using TensorFlow or PyTorch.			
4 a	Apply regularization techniques such as dropout and weight decay to prevent overfitting			
4 b	Experiment with different optimization algorithms (e.g., SGD, Adam) for training the model.			
5	Implement convolutional layers and pooling layers from scratch using NumPy.			
5 a	Construct a simple CNN architecture and train it on a dataset for image classification.			
5 b	Explore various CNN architectures and their applications through hands-on exercises			
6	Implement basic RNN cells and sequence modeling using TensorFlow or PyTorch.			
6 a	Train an RNN model for language modeling and text generation tasks			
6 b	Understand the challenges of training RNNs and explore solutions like LSTM and GRU.			
7	Implement a sequence-to-sequence model for machine translation using RNNs or Transformer architecture.			
7 a	Train the model on a dataset and evaluate its performance using appropriate metrics.			
7 b	Explore attention mechanisms and their role in improving sequence-to-sequence models			
8	Implement basic reinforcement learning algorithms such as Q-learning and policy gradients.			
8 a	Apply these algorithms to solve simple Markov decision processes (MDPs).			
8 b	Experiment with different reward structures and explore their impact on learning.			

9	Implement a Variational Autoencoder (VAE) architecture for learning latent representations			
9 a	Train the VAE model on a dataset and visualize the learned latent space			
9 b	Explore techniques for generating new data samples using the trained VAE model.			
10	Implement a basic GAN architecture (e.g., DCGAN) using TensorFlow or PyTorch.			
10 a	Train the GAN model on a dataset for image generation or style transfer tasks.			
10 b	Experiment with different loss functions and architectures to improve GAN performance.			
11	Use pre-trained GAN models for image generation and synthesis tasks.			
11 b	use of GANs for text generation tasks such as dialogue generation or story generation			

Practical 1

Aim:-Implement basic operations with Scalars, Vectors, Matrices, and Tensors using NumPy.

Perform matrix multiplication and vector operations

Compute norms of vectors and matrices.

Practical 2

Aim: Explore issues like overflow, underflow, and poor conditioning in numerical computation.

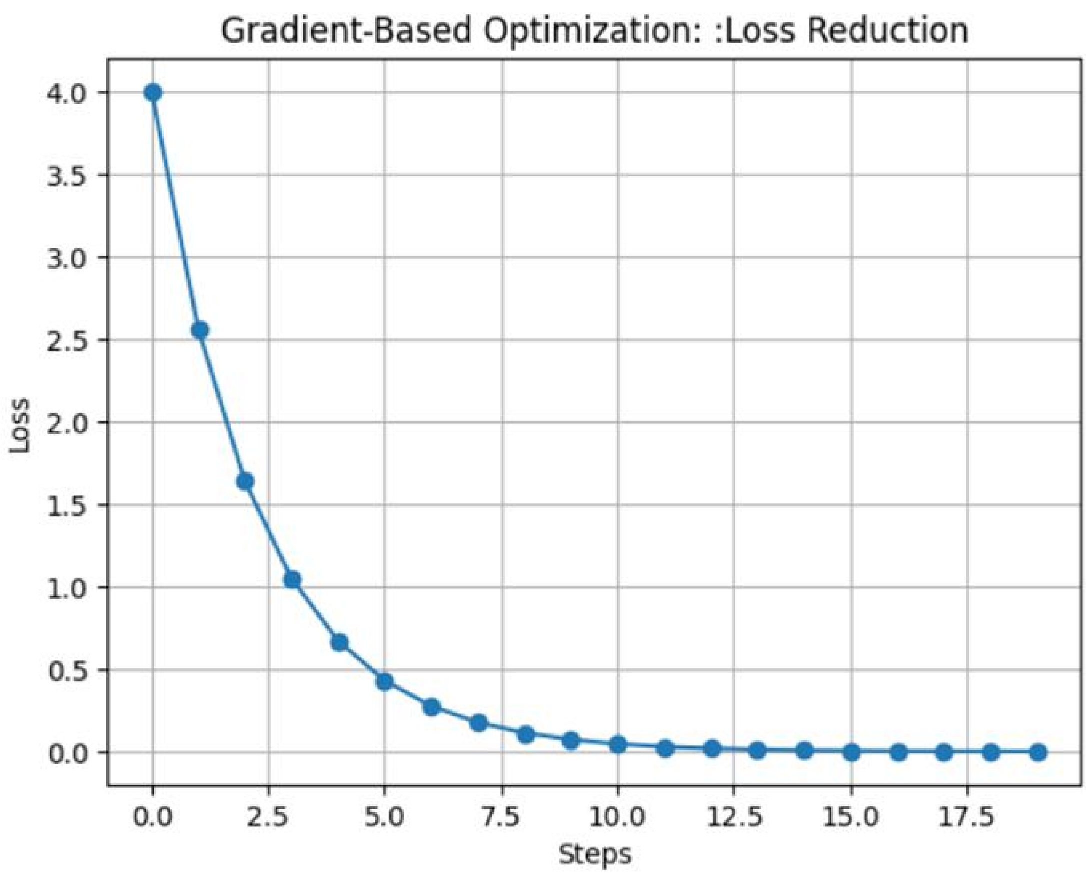
Implement Gradient-Based Optimization algorithms (e.g., Gradient Descent) for simple optimization problems.

Solve Constraint Optimization problems using optimization techniques.

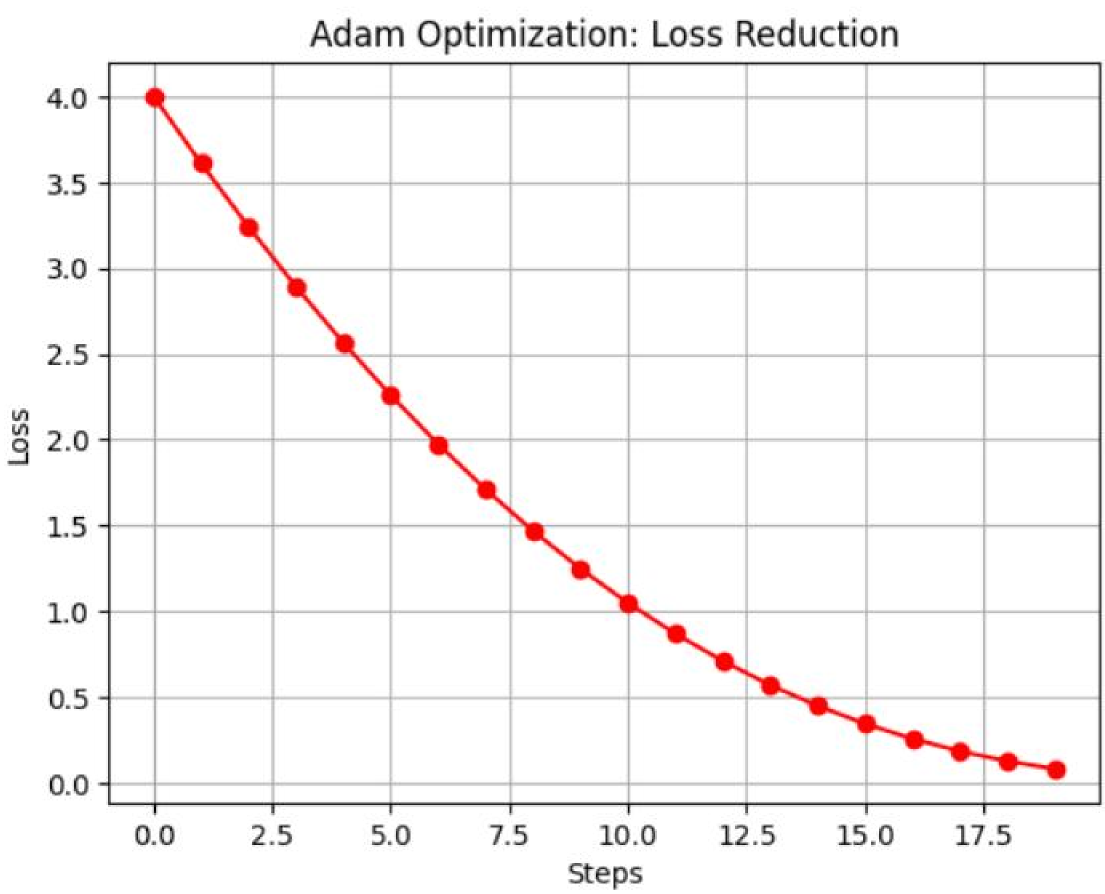
```
# Gradient Based Optimization
import tensorflow as tf
import matplotlib.pyplot as plt

def loss_function(x):
    return (x-3)**2

x = tf.Variable(initial_value = 5.0, trainable = True, dtype = tf.float32)
learning_rate = 0.1
steps = []
loss_values = []

for step in range(20):
    with tf.GradientTape() as tape:
        loss = loss_function(x)

    gradients = tape.gradient(loss, [x])

    x.assign_sub(learning_rate*gradients[0])

    steps.append(step)
    loss_values.append(loss.numpy())

    print(f"Step {step+1}: x = {x.numpy():.4f}, Loss = {loss.numpy():.4f}")

plt.plot(steps, loss_values, marker='o')
plt.title('Gradient-Based Optimization: :Loss Reduction')
plt.xlabel('Steps')
plt.ylabel('Loss')
plt.grid(True)
plt.show()
```

```
# Using Tensorflow Optimizers
x = tf.Variable(initial_value = 5.0, trainable = True, dtype = tf.float32)

optimizer = tf.keras.optimizers.Adam(learning_rate=0.1)
adam_steps = []
adam_loss_values = []

for step in range(20):
    with tf.GradientTape() as tape:
        loss = loss_function(x)

    gradients = tape.gradient(loss, [x])

    optimizer.apply_gradients(zip(gradients, [x]))

    adam_steps.append(step)
    adam_loss_values.append(loss.numpy())

    print(f"Step {step+1} (Adam): x = {x.numpy():.4f}, Loss = {loss.numpy():.4f}")

plt.plot(adam_steps, adam_loss_values, marker='o', color='red')
plt.title('Adam Optimization: Loss Reduction')
plt.xlabel('Steps')
plt.ylabel('Loss')
plt.grid('True')
plt.show()
print("Gayatri Kulkarni -53004230002")
```

```
Step 1: x = 4.6000, Loss = 4.0000
Step 2: x = 4.2800, Loss = 2.5600
Step 3: x = 4.0240, Loss = 1.6384
Step 4: x = 3.8192, Loss = 1.0486
Step 5: x = 3.6554, Loss = 0.6711
Step 6: x = 3.5243, Loss = 0.4295
Step 7: x = 3.4194, Loss = 0.2749
Step 8: x = 3.3355, Loss = 0.1759
Step 9: x = 3.2684, Loss = 0.1126
Step 10: x = 3.2147, Loss = 0.0721
Step 11: x = 3.1718, Loss = 0.0461
Step 12: x = 3.1374, Loss = 0.0295
Step 13: x = 3.1100, Loss = 0.0189
Step 14: x = 3.0880, Loss = 0.0121
Step 15: x = 3.0704, Loss = 0.0077
Step 16: x = 3.0563, Loss = 0.0050
Step 17: x = 3.0450, Loss = 0.0032
Step 18: x = 3.0360, Loss = 0.0020
Step 19: x = 3.0288, Loss = 0.0013
Step 20: x = 3.0231, Loss = 0.0008
```



```
Step 1 (Adam): x = 4.9000, Loss = 4.0000
Step 2 (Adam): x = 4.8002, Loss = 3.6100
Step 3 (Adam): x = 4.7006, Loss = 3.2406
Step 4 (Adam): x = 4.6015, Loss = 2.8921
Step 5 (Adam): x = 4.5030, Loss = 2.5648
Step 6 (Adam): x = 4.4051, Loss = 2.2589
Step 7 (Adam): x = 4.3082, Loss = 1.9744
Step 8 (Adam): x = 4.2123, Loss = 1.7114
Step 9 (Adam): x = 4.1177, Loss = 1.4698
Step 10 (Adam): x = 4.0246, Loss = 1.2493
Step 11 (Adam): x = 3.9331, Loss = 1.0498
Step 12 (Adam): x = 3.8435, Loss = 0.8707
Step 13 (Adam): x = 3.7560, Loss = 0.7115
Step 14 (Adam): x = 3.6709, Loss = 0.5716
Step 15 (Adam): x = 3.5884, Loss = 0.4501
Step 16 (Adam): x = 3.5086, Loss = 0.3462
Step 17 (Adam): x = 3.4319, Loss = 0.2587
Step 18 (Adam): x = 3.3585, Loss = 0.1866
Step 19 (Adam): x = 3.2886, Loss = 0.1285
Step 20 (Adam): x = 3.2225, Loss = 0.0833
```



Gayatri Kulkarni -53004230002

Practical 3

Aim: Implementing deep neural network for performing binary classification task.
Solving XOR problem using deep feed forward network.

Aim: Build and train a Multilayer Perceptron (MLP) for a classification task using TensorFlow or PyTorch.

Aim: Apply regularization techniques such as dropout and weight decay to prevent overfitting

Experiment with different optimization algorithms (e.g., SGD, Adam) for training the model.

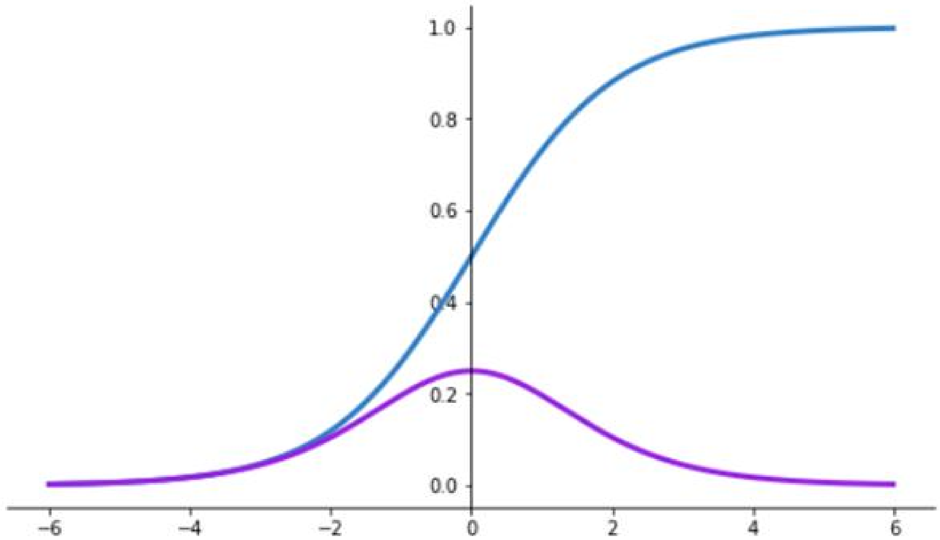
Practical 1

`1)Sigmoid:-

```
#Sigmoid Function
import matplotlib.pyplot as plt
import numpy as np
def sigmoid(x):
    s=1/(1+np.exp(-x))
    ds=s*(1-s)
    return s,ds
x=np.arange(-6,6,0.01)
sigmoid(x)
fig, ax = plt.subplots(figsize=(9, 5))
ax.spines['left'].set_position('center')
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
ax.plot(x,sigmoid(x)[0],color='#307EC7',linewidth=3, label="sigmoid")
ax.plot(x,sigmoid(x)[1],color='#9621e2',linewidth=3, label="derivative")
fig.show()
print("Gayatri Kulkarni - 53004230002")
```

Gayatri Kulkarni - 53004230002

```
C:\Users\admin\AppData\Local\Temp/ipykernel_13244/2117604978.py:18: UserWarnin
g: Matplotlib is currently using module://matplotlib_inline.backend_inline, whi
ch is a non-GUI backend, so cannot show the figure.
  fig.show()
```



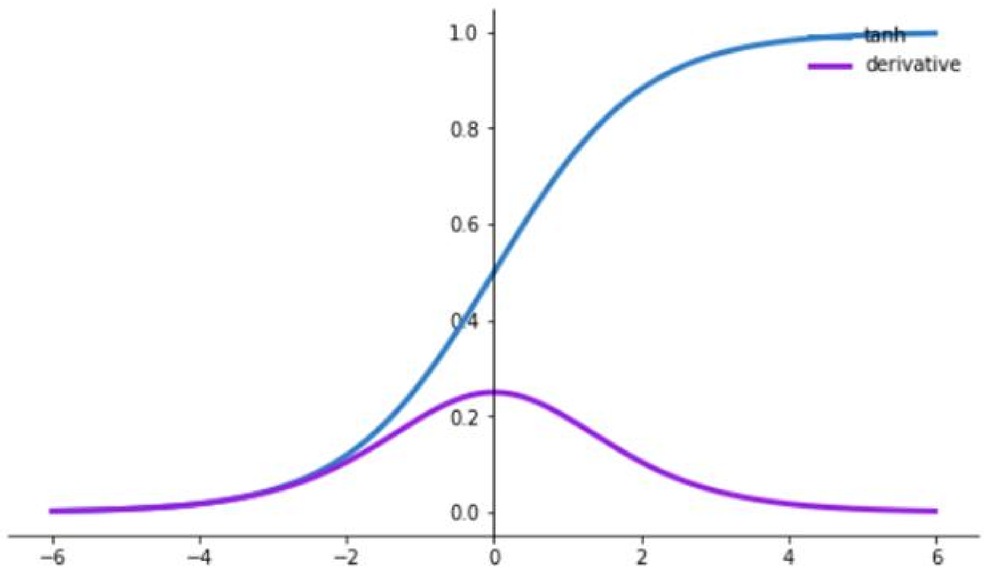
2)Tanh

```
#Tanh Function
import matplotlib.pyplot as plt
import numpy as np
def tanh(x):
    t=(np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))
    dt=1-t**2
    return t,dt
z=np.arange(-4,4,0.01)
tanh(z)[0].size,tanh(z)[1].size
fig, ax = plt.subplots(figsize=(9, 5))
ax.spines['left'].set_position('center')
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
ax.plot(x,sigmoid(x)[0],color='#307EC7',linewidth=3, label="tanh")
ax.plot(x,sigmoid(x)[1],color='#9621e2',linewidth=3, label="derivative")
```

ax.legend(loc="upper right", frameon=False)

plt.show()

print("Gayatri Kulkarni - 53004230002")

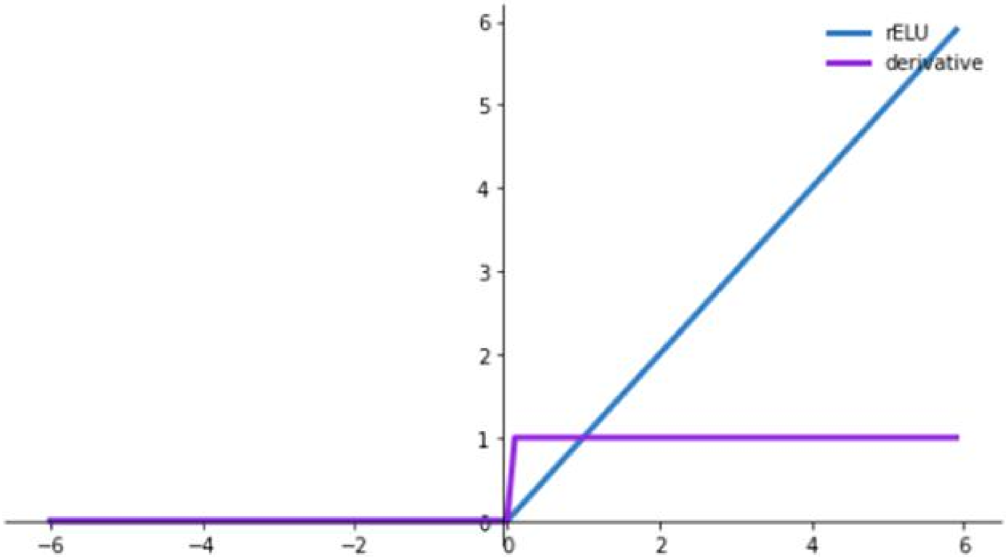


Gayatri Kulkarni - 53004230002

3)Relu

```
#Relu Function and its derivative
import matplotlib.pyplot as plt
import numpy as np
def relu(x):
    r = np.maximum(0, x)
    dr = np.where(x <=0, 0, 1)
    return r,dr


x=np.arange(-6, 6, 0.1)#Range for the x-axis
relu_values, relu_derivatives = relu(x) #Compute ReLU AND its derivative
fig, ax = plt.subplots(figsize=(9, 5))
ax.spines['left'].set_position('center')
ax.spines['bottom'].set_position(('data', 0))#set the x-axis at y=0
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
```

```
ax.yaxis.set_ticks_position('left')
ax.plot(x,relu_values,color='#307EC7',linewidth=3, label="rELU")
ax.plot(x,relu_derivatives,color='#9621e2',linewidth=3, label="derivative")
ax.legend(loc="upper right", frameon=False)
plt.show()
print("Gayatri Kulkarni - 53004230002")
```

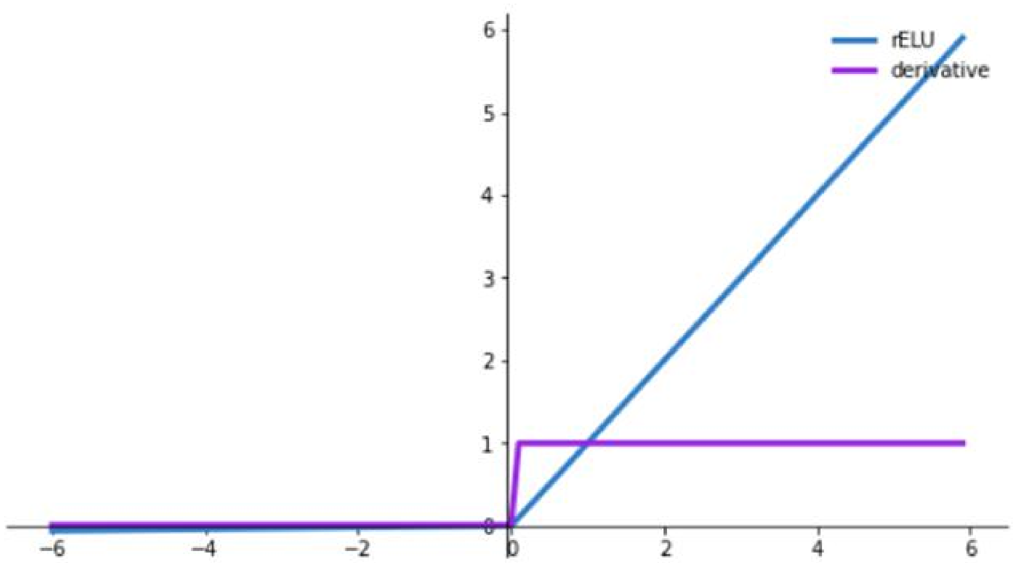



```
Gayatri Kulkarni - 53004230002
```

4)Leaky RELU

```
# lEAKY Rectified Linear Unit(Leaky Relu)
import matplotlib.pyplot as plt
import numpy as np
def leaky_relu(x, alpha=0.01):
    r = np.maximum(alpha * x, x)
    dr = np.where(x < 0, alpha, 1)
    return r,dr
#Generate x values
x=np.arange(-6, 6, 0.1)#Range for the x-axis
leaky_relu_values, leaky_relu_derivatives = leaky_relu(x)

# Create the plot
fig, ax = plt.subplots(figsize=(9, 5))
```

```
ax.spines['left'].set_position('center')
ax.spines['bottom'].set_position(('data', 0))#set the x-axis at y=0
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
ax.plot(x, leaky_relu_values, color='#307EC7',linewidth=3, label="rELU")
ax.plot(x,leaky_relu_derivatives, color='#9621e2',linewidth=3, label="derivative")
ax.legend(loc="upper right", frameon=False)
#Show the plot
plt.show()
print("Gayatri Kulkarni - 53004230002")
```

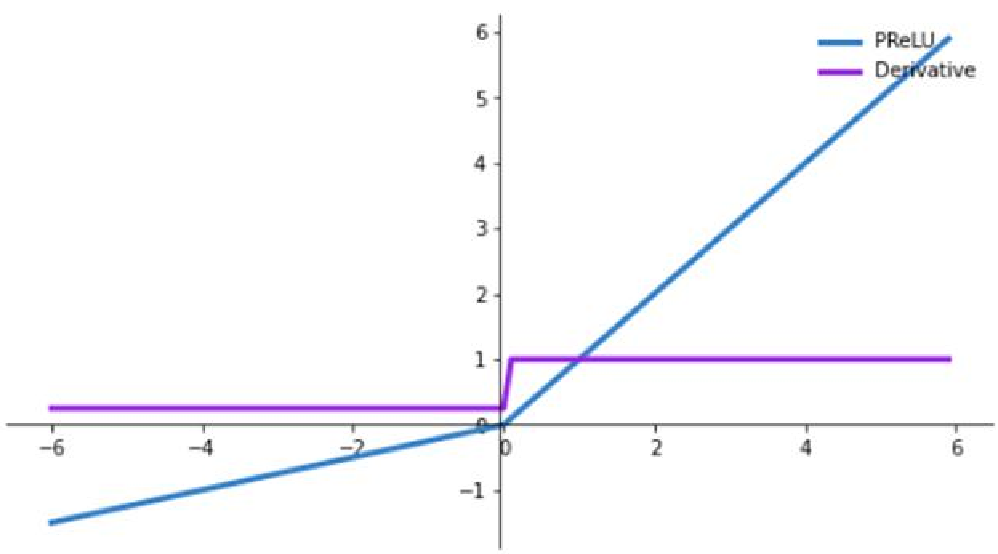



```
Gayatri Kulkarni - 53004230002
```

4)Prelu

```
# Parametric Rectified Linear Unit (PRelu)
import matplotlib.pyplot as plt
import numpy as np
def prelu(x, alpha=0.25):
    r = np.maximum(alpha * x, x)
    dr = np.where(x < 0, alpha, 1)
    return r,dr
```

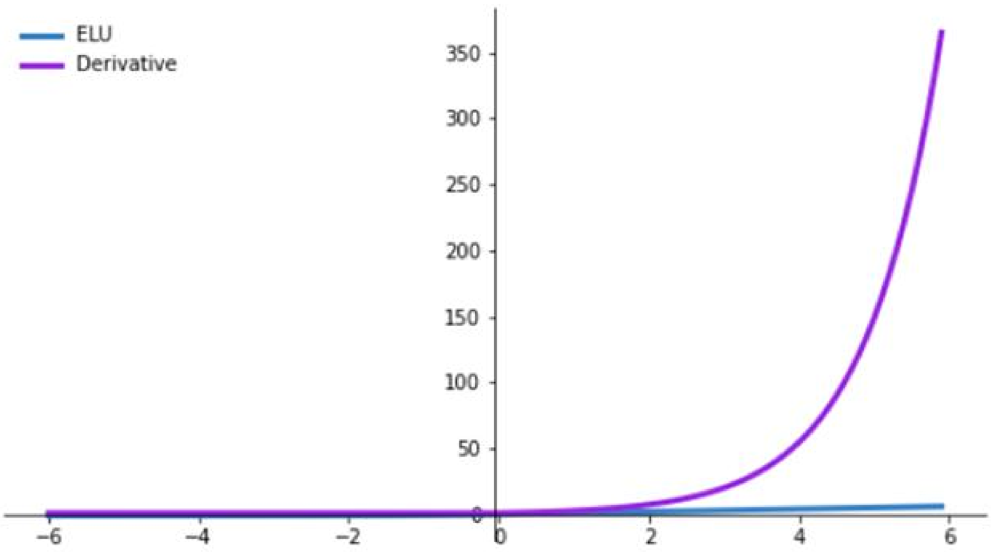
```
x=np.arange(-6, 6, 0.1)#Range for the x-axis
prelu_values, prelu_derivatives = prelu(x)

# Create the plot
fig, ax = plt.subplots(figsize=(9, 5))
ax.spines['left'].set_position('center')
ax.spines['bottom'].set_position(('data', 0))#set the x-axis at y=0
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')

#Plotting the PRelu and its derivative
ax.plot(x, prelu_values, color='#307EC7',linewidth=3, label="PReLU")
ax.plot(x,prelu_derivatives, color='#9621e2',linewidth=3, label="Derivative")
ax.legend(loc="upper right", frameon=False)
#Show the plot
plt.show()
print("Gayatri Kulkarni - 53004230002")
```




```
Gayatri Kulkarni - 53004230002
```

5)ELU

```
# Exponential Linear Unit (ELU)
import matplotlib.pyplot as plt
import numpy as np
def elu(x, alpha=1.0):
    r = np.where(x >= 0, x, alpha * (np.exp(x) - 1))
    dr = np.where(x <= 0, 1, alpha * np.exp(x))
    return r,dr

x=np.arange(-6, 6, 0.1)#Range for the x-axis
elu_values, elu_derivatives = elu(x)

# Create the plot
fig, ax = plt.subplots(figsize=(9, 5))
ax.spines['left'].set_position('center')
ax.spines['bottom'].set_position(('data', 0))#set the x-axis at y=0
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')

#Plotting the PRelu and its derivative
ax.plot(x, elu_values, color='#307EC7',linewidth=3, label="ELU")
ax.plot(x,prelu_derivatives, color='#9621e2',linewidth=3, label="Derivative")
ax.legend(loc="upper left", frameon=False)
#Show the plot
plt.show()
print("Gayatri Kulkarni - 53004230002")
```

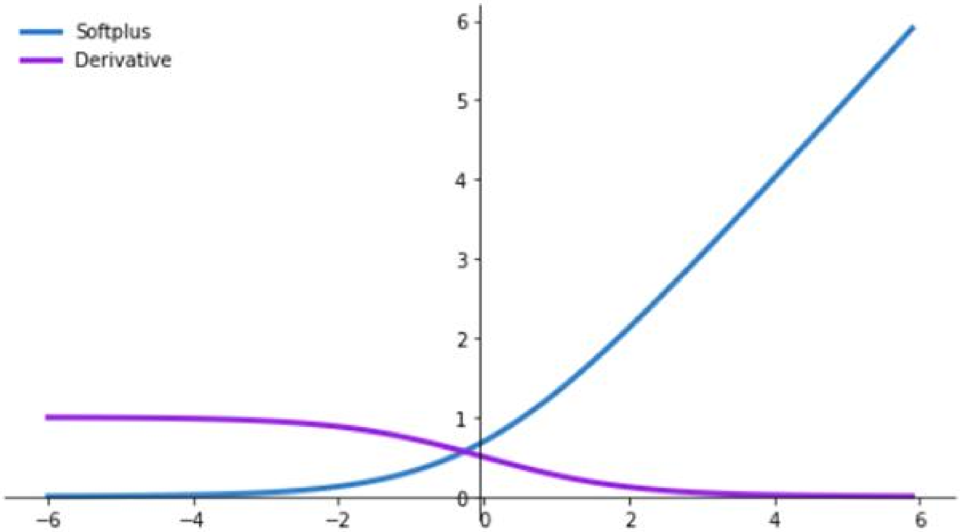


Gayatri Kulkarni - 53004230002

5)Softplus

```
import matplotlib.pyplot as plt
import numpy as np
def softplus(x):
    r = np.log(1 + np.exp(x))
    dr = 1/(1 + np.exp(x))
    return r,dr

x=np.arange(-6, 6, 0.1)#Range for the x-axis
softplus_values, softplus_derivatives = softplus(x)

fig, ax = plt.subplots(figsize=(9, 5))
ax.spines['left'].set_position('center')
ax.spines['bottom'].set_position(('data', 0))#set the x-axis at y=0
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')

#Plotting the PRelu and its derivative
```

```
ax.plot(x, softplus_values, color='#307EC7',linewidth=3, label="Softplus")
ax.plot(x,softplus_derivatives, color='#9621e2',linewidth=3, label="Derivative")
ax.legend(loc="upper left", frameon=False)
#Show the plot
plt.show()
print("Gayatri Kulkarni - 53004230002")
```

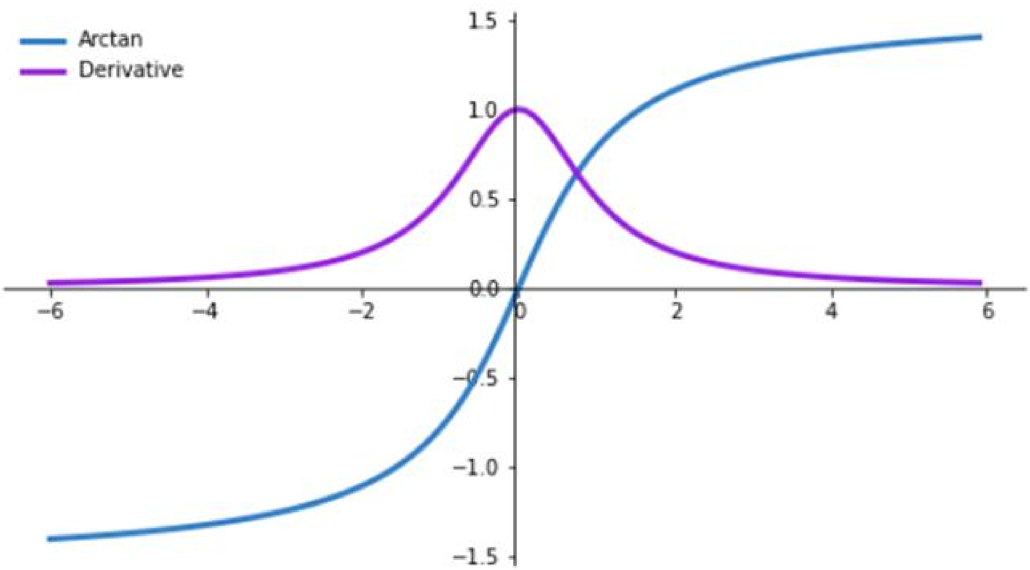


Gayatri Kulkarni - 53004230002

6)arctan

```
import matplotlib.pyplot as plt
import numpy as np

def arctan(x):
    r = np.arctan(x)
    dr = 1/(1 +  x**2)
    return r,dr

x=np.arange(-6, 6, 0.1)#Range for the x-axis
arctan_values, arctan_derivatives = arctan(x)

fig, ax = plt.subplots(figsize=(9, 5))
ax.spines['left'].set_position('center')
ax.spines['bottom'].set_position(('data', 0))#set the x-axis at y=0
```

```
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')

#Plotting the PRelu and its derivative
ax.plot(x, arctan_values, color='#307EC7',linewidth=3, label="Arctan")
ax.plot(x, arctan_derivatives, color='#9621e2',linewidth=3, label="Derivative")
ax.legend(loc="upper left", frameon=False)
#Show the plot
plt.show()
print("Gayatri Kulkarni - 53004230002")
```



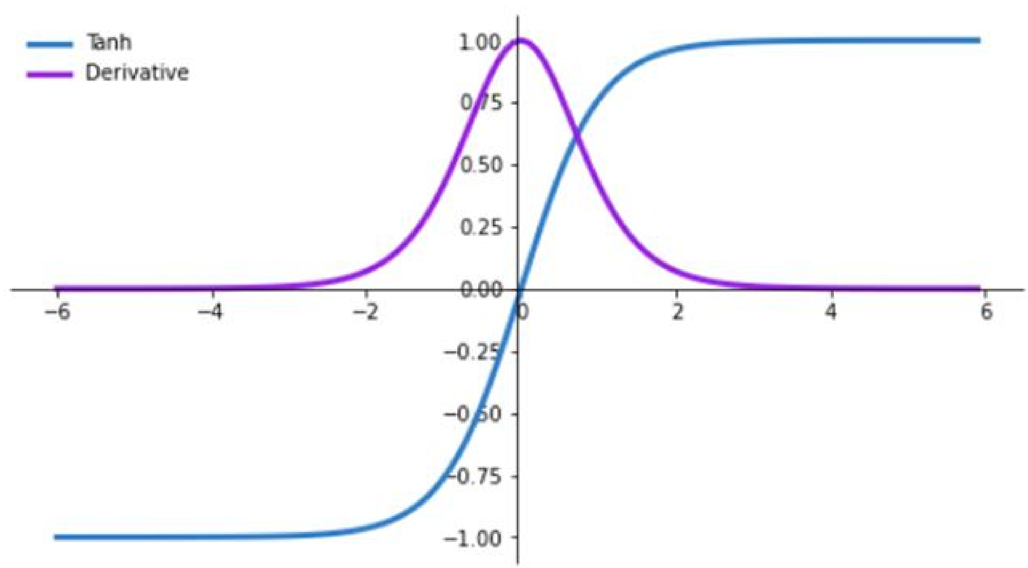

```
Gayatri Kulkarni - 53004230002
```

---

7)tanh

```
import matplotlib.pyplot as plt
import numpy as np

def tanh(x):
    r = np.tanh(x)
    dr = 1- r**2
    return r,dr
```

```
x=np.arange(-6, 6, 0.1)#Range for the x-axis
tanh_values, tanh_derivatives = tanh(x)

fig, ax = plt.subplots(figsize=(9, 5))
ax.spines['left'].set_position('center')
ax.spines['bottom'].set_position(('data', 0))#set the x-axis at y=0
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')

#Plotting the PRelu and its derivative
ax.plot(x, tanh_values, color='#307EC7',linewidth=3, label="Tanh")
ax.plot(x, tanh_derivatives, color='#9621e2',linewidth=3, label="Derivative")
ax.legend(loc="upper left", frameon=False)
#Show the plot
plt.show()
print("Gayatri Kulkarni - 53004230002")
```




```
Gayatri Kulkarni - 53004230002
```

meet.google.com/dkf-kwae-fzs?pli=1

Rajesh Maurya (Presenting)

localhost:8890/notebooks/Practical/1-Activation%20functions/Activation%20functions%20and%20derivatives.ipynb

jupyter Activation functions and derivatives Last Checkpoint: 6 months ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

[23]:

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt

# Data preparation
X, y = make_moons(n_samples=1000, noise=0.2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Model definition
model = Sequential([
    Dense(10, input_shape=(2,), activation='tanh'),
    Dense(10, activation='elu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer=Adam(learning_rate=0.01), loss='binary_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=100, verbose=1)

def plot_decision_boundary(model, X, y):
```

Rajesh viraj SAP- 0

17:00 | dkf-kwae-fzs

Type here to search

Meet – dkf-kwae-fzs

meet.google.com/dkf-kwae-fzs?pli=1

Rajesh Maurya (Presenting)

localhost:8890/notebooks/Practical/1-Activation%20functions/Activation%20functions%20and%20derivatives.ipynb

jupyter Activation functions and derivatives Last Checkpoint: 6 months ago

File Edit View Run Kernel Settings Help Not Tru

Code JupyterLab Python 3 (ipyke

```python
model.compile(optimizer=Adam(learning_rate=0.01), loss='binary_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=100, verbose=1)

def plot_decision_boundary(model, X, y):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor='k')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('Decision Boundary and Test Data Points')

plt.figure(figsize=(10, 6))
plot_decision_boundary(model, np.vstack((X_train, X_test)), np.hstack((y_train, y_test)))
plt.show()

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Training Performance')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```

17:01 | dkf-kwae-fzs

Practical 4

Aim: Build and train a Multilayer Perceptron (MLP) for a classification task using TensorFlow or PyTorch.

Apply regularization techniques such as dropout and weight decay to prevent overfitting

Experiment with different optimization algorithms (e.g., SGD, Adam) for training the model.

```
 import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
#-----------------part 3
print("=====part3:Dataset Loading and preprocessing===")
    #3.1 load datasets (eg. MNIST, CIFAR-10)
    #load  mnist datasets
(mnist_train, mnist_train_labels), (mnist_test, mnist_test_labels) = tf.keras.datasets.mnist.load_data()
#normalise
mnist_train = mnist_train/255.0
mnist_test = mnist_test/255.0

#expand dimensu=ions (eg. MNIST images are grayscale)
mnist_train = mnist_train[..., tf.newaxis]
mnist_train = mnist_train[..., tf.newaxis]

print(f"MINST Train Shape: {mnist_train.shape}, MNIST Test Shape: {mnist_test.shape}")

#3.2 Create a custom dataset using tf.data Datasets
batch_size = 32
train_dataset = tf.data.Dataset.from_tensor_slices((mnist_train, mnist_train_labels))
train_dataset = train_dataset.shuffle(buffer_size=10000).batch(batch_size)

test_dataset = tf.data.Dataset.from_tensor_slices((mnist_test, mnist_test_labels))
test_dataset = test_dataset.batch(batch_size)

#3.3 perform Data Augmentation using TensorFlow operations
```

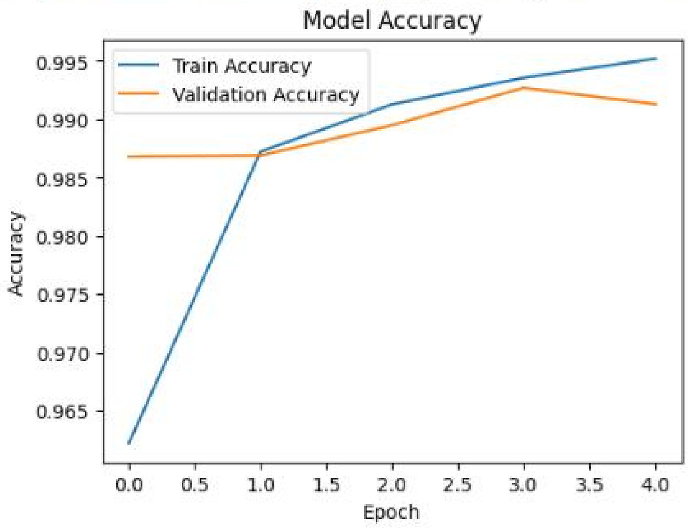
```
def augment(image, label):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
    return image, label
    augment_train_dataset = train_dataset.map(augment)
```

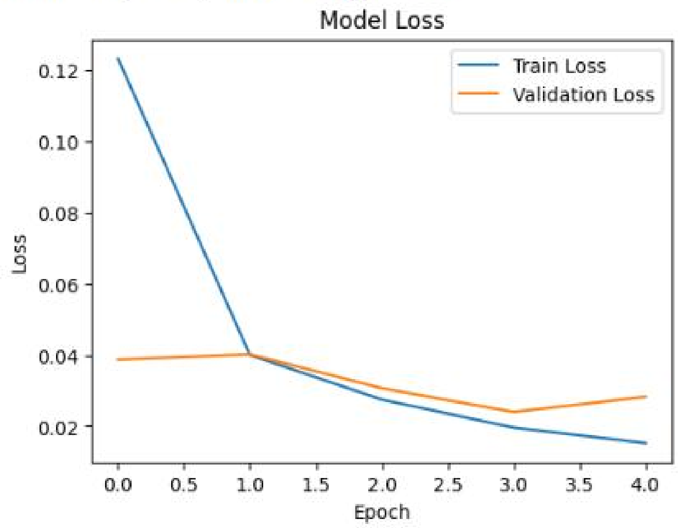
```
=====part3:Dataset Loading and preprocessing===
MINST Train Shape: (60000, 28, 28, 1, 1), MNIST Test Shape: (10000, 28, 28)
```

```
#------------------------------------ Part 4:Build and train simple model---------------#
print("¥n== Part 4:Build and Train a Simple Model==")
#4.1 Create a Neural Network to Classify MNIST using the Sequential API
model = models.Sequential([
    layers.Input(shape=(28, 28, 1)),
    layers.Conv2D(64, kernel_size=(3,3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2,2)),
    layers.Conv2D(64, kernel_size=(3,3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2,2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax') # 10 classes for MNIST digits
])

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

# 4.2 Train the Model
history = model.fit(train_dataset, epochs=5, validation_data=test_dataset)
```

```
#4.3 Evaluate any Visualise Model Performance
plt.figure(figsize=(12, 4))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.histroy['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
print("Gayatri Kulkarni -5300423002")
```

```
== Part 4:Build and Train a Simple Model==
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 18s 9ms/step - accuracy: 0.9110 - loss: 0.2835 - val_accuracy: 0.9868 - val_loss: 0.0387
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 17s 9ms/step - accuracy: 0.9856 - loss: 0.0456 - val_accuracy: 0.9869 - val_loss: 0.0402
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 18s 10ms/step - accuracy: 0.9903 - loss: 0.0299 - val_accuracy: 0.9895 - val_loss: 0.0307
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 18s 10ms/step - accuracy: 0.9934 - loss: 0.0208 - val_accuracy: 0.9927 - val_loss: 0.0240
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 19s 10ms/step - accuracy: 0.9952 - loss: 0.0146 - val_accuracy: 0.9913 - val_loss: 0.0283
```





Gayatri Kulkarni -5300423002

Practical 5

Aim: Implement convolutional layers and pooling layers from scratch using NumPy

Construct a simple CNN architecture and train it on a dataset for image classification.

Explore various CNN architectures and their applications through hands-on exercises

Practical 6

Aim: Implement basic RNN cells and sequence modeling using TensorFlow or PyTorch

Train an RNN model for language modeling and text generation tasks

Understand the challenges of training RNNs and explore solutions like LSTM and GRU.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN, LSTM
from sklearn.model_selection import train_test_split
import seaborn as sns
#load
url='https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers.csv'
df=pd.read_csv(url,usecols=['Passengers'])
df.head()
```

	Passengers
0	112
1	118
2	132
3	129
4	121

```
#Data Preprocessing :Normalize the dataset for better performance of the neural network
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(df)

#create
def create_dataset(data, time_step=1):
    X, Y = [], []
    for i in range(len(data)-time_step-1):
        a = data[i:(i+time_step), 0]
        X.append(a)
        Y.append(data[i + time_step, 0])
    return np.array(X), np.array(Y)
#prepare
time_step = 10
X, Y = create_dataset(scaled_data, time_step)

#SPLIT
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

#reshape
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

```
#Build and train thr RNN Model
model = Sequential()
model.add(SimpleRNN(50, return_sequences=True, input_shape=(time_step, 1)))
model.add(SimpleRNN(50, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

#Train the model
model.fit(X_train, Y_train, epochs=100, batch_size=32, validation_data=(X_test, Y_test), verbose=1)
```

```
Epoch 1/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 110ms/step - loss: 0.1251 - val_loss: 0.0881
Epoch 2/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.1027 - val_loss: 0.0120
Epoch 3/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.0316 - val_loss: 0.0215
Epoch 4/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0236 - val_loss: 0.0248
Epoch 5/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0291 - val_loss: 0.0079
Epoch 6/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0118 - val_loss: 0.0129
Epoch 7/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0168 - val_loss: 0.0059
Epoch 8/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0096 - val_loss: 0.0079
Epoch 9/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 0.0089 - val_loss: 0.0054
Epoch 10/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0083 - val_loss: 0.0062
Epoch 11/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0064 - val_loss: 0.0048
Epoch 12/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0068 - val_loss: 0.0047
Epoch 13/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0058 - val_loss: 0.0046
Epoch 14/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0058 - val_loss: 0.0040
Epoch 15/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.0058 - val_loss: 0.0038
Epoch 16/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0046 - val_loss: 0.0036
Epoch 17/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.0050 - val_loss: 0.0035
Epoch 18/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0043 - val_loss: 0.0035
Epoch 19/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0043 - val_loss: 0.0033
Epoch 20/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.0040 - val_loss: 0.0031
Epoch 21/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0041 - val_loss: 0.0030
Epoch 22/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0039 - val_loss: 0.0029
Epoch 23/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0042 - val_loss: 0.0029
Epoch 24/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.0038 - val_loss: 0.0029
Epoch 25/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0042 - val_loss: 0.0028
```

```
Epoch 26/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0037 - val_loss: 0.0028
Epoch 27/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.0034 - val_loss: 0.0028
Epoch 28/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.0034 - val_loss: 0.0027
Epoch 29/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0031 - val_loss: 0.0027
Epoch 30/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0036 - val_loss: 0.0027
Epoch 31/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0034 - val_loss: 0.0027
Epoch 32/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0037 - val_loss: 0.0026
Epoch 33/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0033 - val_loss: 0.0028
Epoch 34/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0028 - val_loss: 0.0031
Epoch 35/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0036 - val_loss: 0.0024
Epoch 36/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0033 - val_loss: 0.0023
Epoch 37/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0028 - val_loss: 0.0024
Epoch 38/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0029 - val_loss: 0.0023
Epoch 39/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0028 - val_loss: 0.0023
Epoch 40/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0030 - val_loss: 0.0023
Epoch 41/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0029 - val_loss: 0.0022
Epoch 42/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0030 - val_loss: 0.0023
Epoch 43/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0027 - val_loss: 0.0022
Epoch 44/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0026 - val_loss: 0.0022
Epoch 45/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0028 - val_loss: 0.0022
Epoch 46/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0026 - val_loss: 0.0025
Epoch 47/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0029 - val_loss: 0.0024
Epoch 48/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0027 - val_loss: 0.0021
Epoch 49/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 0.0028 - val_loss: 0.0022
Epoch 50/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0026 - val_loss: 0.0022
```

```
Epoch 50/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0026 - val_loss: 0.0022
Epoch 51/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0025 - val_loss: 0.0021
Epoch 52/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 0.0025 - val_loss: 0.0021
Epoch 53/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0022 - val_loss: 0.0021
Epoch 54/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0024 - val_loss: 0.0024
Epoch 55/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0030 - val_loss: 0.0023
Epoch 56/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0029 - val_loss: 0.0023
Epoch 57/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0025 - val_loss: 0.0024
Epoch 58/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0031 - val_loss: 0.0021
Epoch 59/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 0.0027 - val_loss: 0.0027
Epoch 60/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0036 - val_loss: 0.0021
Epoch 61/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0022 - val_loss: 0.0020
Epoch 62/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0022 - val_loss: 0.0020
Epoch 63/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0023 - val_loss: 0.0019
Epoch 64/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0022 - val_loss: 0.0021
Epoch 65/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0024 - val_loss: 0.0019
Epoch 66/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0023 - val_loss: 0.0021
Epoch 67/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0024 - val_loss: 0.0028
Epoch 68/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0035 - val_loss: 0.0026
Epoch 69/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0032 - val_loss: 0.0020
Epoch 70/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0022 - val_loss: 0.0021
Epoch 71/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0024 - val_loss: 0.0022
Epoch 72/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0026 - val_loss: 0.0017
Epoch 73/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0022 - val_loss: 0.0018
Epoch 74/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0021 - val_loss: 0.0018
Epoch 75/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 0.0022 - val_loss: 0.0023
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0021 - val_loss: 0.0018
Epoch 75/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 0.0022 - val_loss: 0.0023
Epoch 76/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - loss: 0.0029 - val_loss: 0.0025
Epoch 77/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0029 - val_loss: 0.0018
Epoch 78/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 0.0021 - val_loss: 0.0017
Epoch 79/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0020 - val_loss: 0.0016
Epoch 80/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0021 - val_loss: 0.0017
Epoch 81/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0021 - val_loss: 0.0017
Epoch 82/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 0.0019 - val_loss: 0.0027
Epoch 83/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 0.0030 - val_loss: 0.0022
Epoch 84/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 0.0029 - val_loss: 0.0023
Epoch 85/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 0.0023 - val_loss: 0.0021
Epoch 86/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 0.0027 - val_loss: 0.0022
Epoch 87/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 0.0023 - val_loss: 0.0018
Epoch 88/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0022 - val_loss: 0.0018
Epoch 89/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0021 - val_loss: 0.0016
Epoch 90/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 0.0019 - val_loss: 0.0015
Epoch 91/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 0.0019 - val_loss: 0.0016
Epoch 92/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 0.0021 - val_loss: 0.0017
Epoch 93/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 0.0019 - val_loss: 0.0017
Epoch 94/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 0.0020 - val_loss: 0.0018
Epoch 95/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0021 - val_loss: 0.0014
Epoch 96/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 0.0019 - val_loss: 0.0023
Epoch 97/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - loss: 0.0032 - val_loss: 0.0017
Epoch 98/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - loss: 0.0024 - val_loss: 0.0017
Epoch 99/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - loss: 0.0019 - val_loss: 0.0015
Epoch 100/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - loss: 0.0016 - val_loss: 0.0016
: <keras.src.callbacks.history.History at 0x18b0197b9e0>
```

#Predict and Evaluation

train_predict = model.predict(X_train)

test_predict = model.predict(X_test)

#Inverse transform to get the actual values

train_predict = scaler.inverse_transform(train_predict)

```
test_predict = scaler.inverse_transform(test_predict)
#Y_train = scaler.inverse_transform([Y_train])
#Y_test = scaler.inverse_transform([Y_test])
Y_train = scaler.inverse_transform(Y_train.reshape(-1, 1))
Y_test = scaler.inverse_transform(Y_test.reshape(-1, 1))

# Calculate RMSE
train_rmse = np.sqrt(mean_squared_error(Y_train, train_predict[:,0]))
test_rmse = np.sqrt(mean_squared_error(Y_test, test_predict[:,0]))

print(f'Train RMSE: {train_rmse}')
print(f'Train RMSE: {test_rmse}')
```

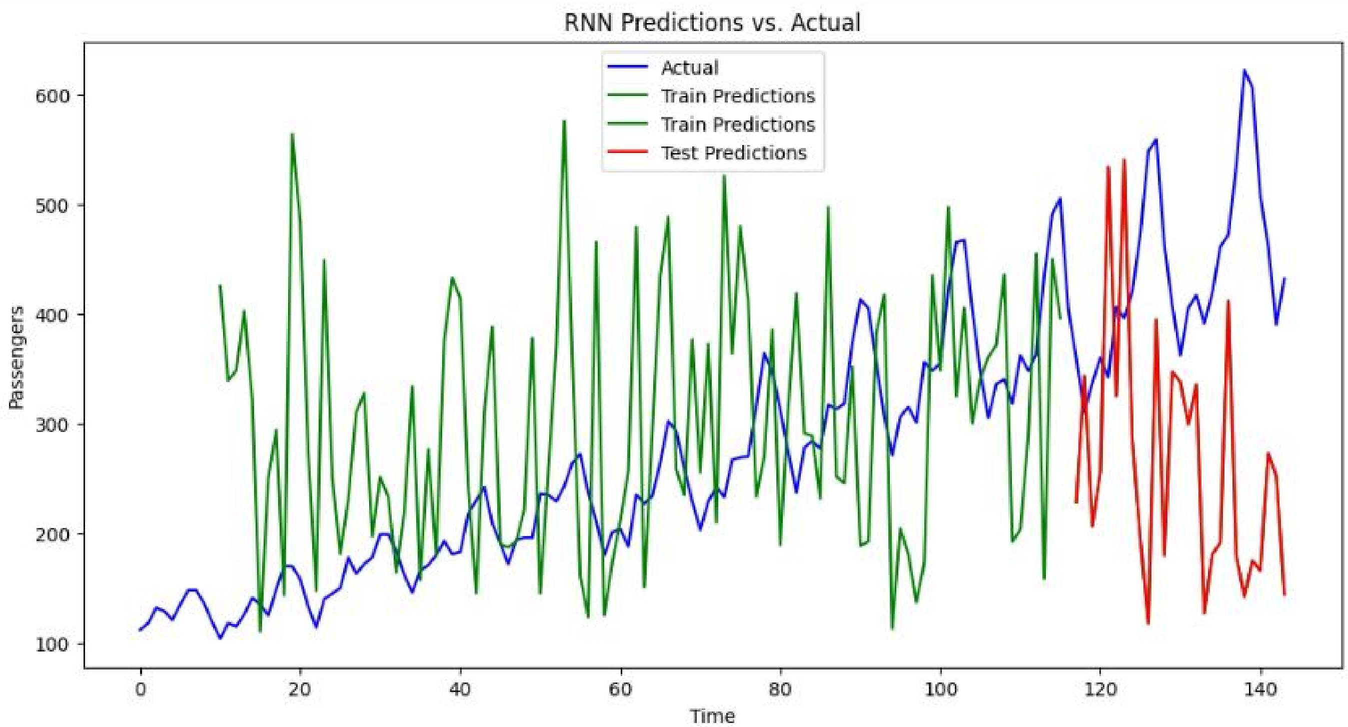
```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
Train RMSE: 21.52355107828995
Train RMSE: 20.813798677468384
```

```
#Compare with a Traditional Time Series Model (ARIMA)
from statsmodels.tsa.arima.model import ARIMA

# Fit the ARIMA model
model_arima = ARIMA(df, order=(5,1,0))
model_arima_fit = model_arima.fit()

#Forecast
arima_pred = model_arima_fit.forecast(steps=len(X_test))

#Calculate RMSE for ARIMA
arima_rmse = np.sqrt(mean_squared_error(df[-len(arima_pred):], arima_pred))
print(f'ARIMA RMSE: {arima_rmse}')
```

```
ARIMA RMSE: 96.63626284391589
```

```
#Visualization : RNN Predications vs. Actual Vlues
plt.figure(figsize=(12,6))
```

```
#plot the actual data
plt.plot(df, label='Actual', color='blue')

#Plot the training predictions
plt.plot(range(time_step, time_step + len(train_predict)), train_predict, color='green', label='Train Predictions')

#Plot the testing predictions
plt.plot(range(len(df) - len(test_predict), len(df)), test_predict, color='green', label='Train Predictions')

#Plot the testing predictions
plt.plot(range(len(df) - len(test_predict), len(df)), test_predict, color='red', label='Test Predictions')

plt.legend()
plt.title('RNN Predictions vs. Actual')
plt.xlabel('Time')
plt.ylabel('Passengers')
plt.show()

print("Gayatri Kulkarni - 53004230002")
```



Gayatri Kulkarni - 53004230002

Practical 7

Aim: Implement a sequence-to-sequence model for machine translation using RNNs or Transformer architecture.

Train the model on a dataset and evaluate its performance using appropriate metrics.

Explore attention mechanisms and their role in improving sequence-to-sequence models

```
import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

import os

import zipfile

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import VGG16


#Download

url = "https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip"

#filename = os.path.join(os.getcwd(), "cats_and_dogs_filtered.zip")

cache_dir = os.getcwd()
```

```
filename = "cats_and_dogs_filtered.zip"
file_path = tf.keras.utils.get_file(fname=filename, origin=url, cache_dir=cache_dir, extract=False)
tf.keras.utils.get_file(filename, url)
#with zipfile.ZipFile("cats_and_dogs_filtered.zip", "r") as zip_ref:
  # zip_ref.extractall()
with zipfile.ZipFile(file_path, "r") as zip_ref:
   zip_ref.extractall()
#define
train_dir = os.path.join(os.getcwd(), "cats_and_dogs_filtered", "train")
validation_dir = os.path.join(os.getcwd(), "cats_and_dogs_filtered", "validation")

train_datagen = 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)
validation_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size=(150,150),
                                                    batch_size=20,
                                                    class_mode="binary")
validation_generator = validation_datagen.flow_from_directory(validation_dir,
                                                    target_size=(150,150),
                                                    batch_size=20,
                                                    class_mode="binary")


#Load
conv_base = VGG16(weights="imagenet",
              include_top=False,
```

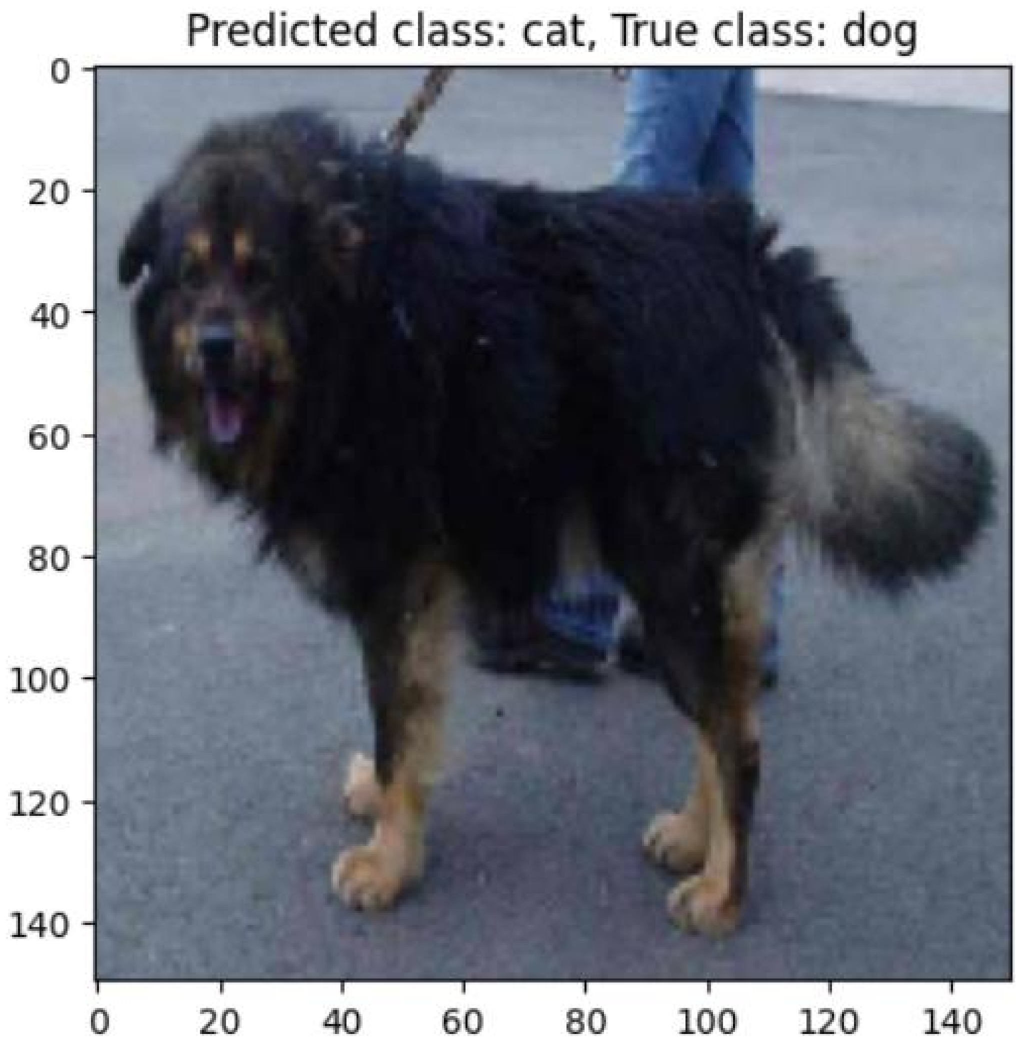
```
                 input_shape=(150, 150, 3))
#freeze
conv_base.trainable = False
#build
model = tf.keras.models.Sequential()
model.add(conv_base)
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256, activation="relu"))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(1, activation="sigmoid"))

#compile
model.compile(loss="binary_crossentropy",
          optimizer=tf.keras.optimizers.RMSprop(learning_rate=2e-5),
          metrics=["accuracy"])
#train
history = model.fit(train_generator,
                 steps_per_epoch=100,
                 epochs=30,
                 validation_data=validation_generator,
                 validation_steps=50)
#show
x, y_true = next(validation_generator)
y_pred = model.predict(x)
class_names = ['cat', 'dog']
for i in range(len(x)):
   plt.imshow(x[i])
   plt.title(f'Predicted class: {class_names[int(round(y_pred[i][0]))]}, True class: {class_names[int(y_true[i])]}')
   plt.show()
#plot
acc = history.history["accuracy"]
val_acc = history.history["val_accuracy"]
```

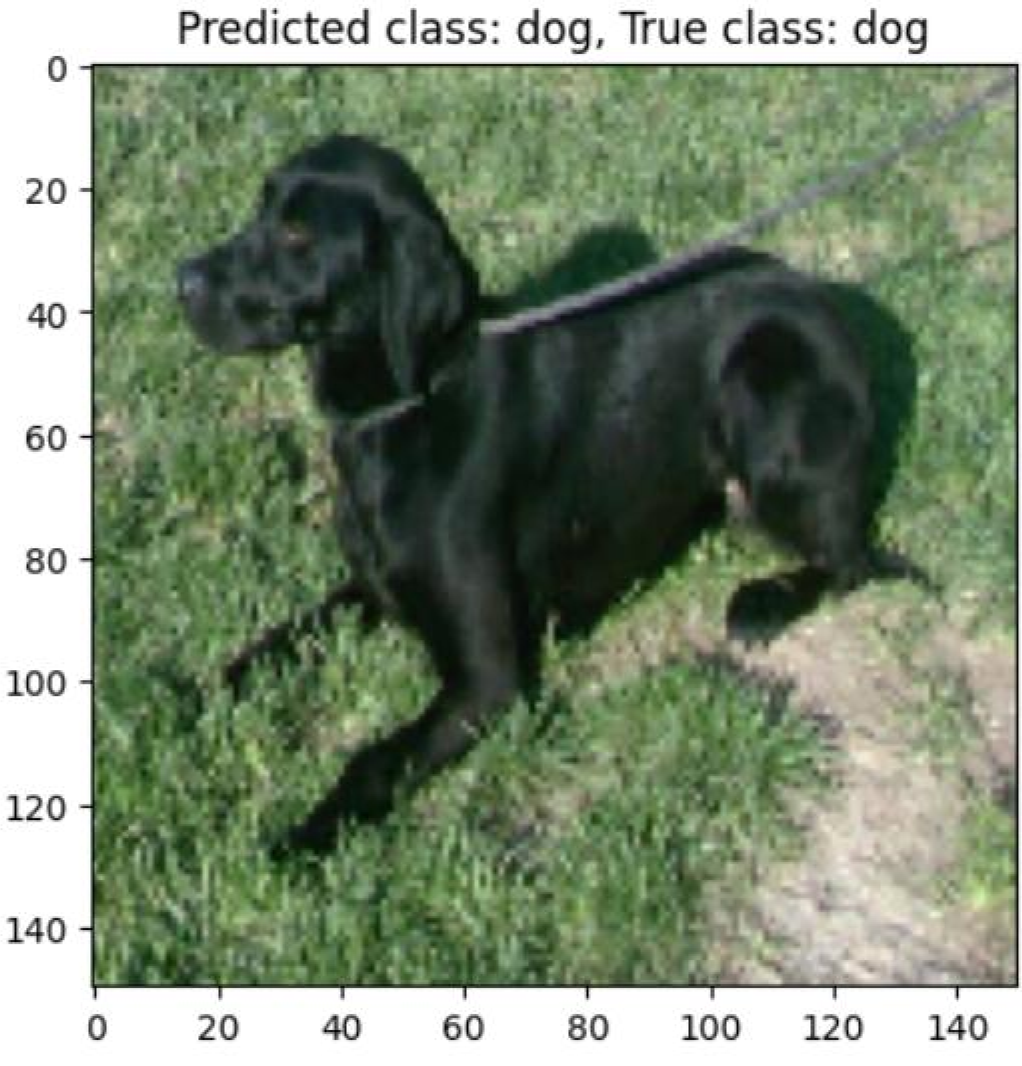
```
loss = history.history["loss"]
val_loss = history.history["val_loss"]

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.legend()

plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b",label="Validation loss")
plt.title("Training and validation loss")
plt.legend()

plt.show()
print("Gayatri kulkarni-53004230002")
```

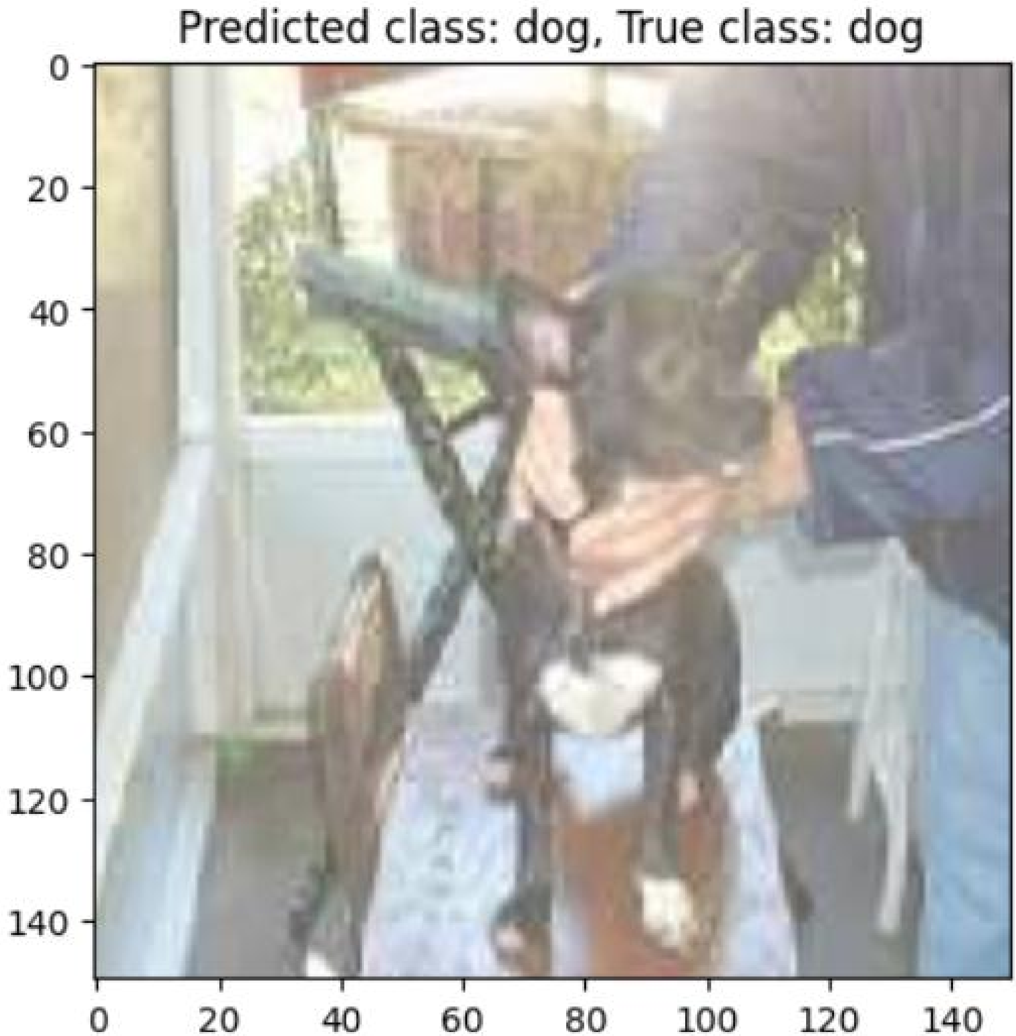
```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Epoch 1/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 193s 2s/step - accuracy: 0.5729 - loss: 0.7041 - val_accuracy: 0.8090 - val_loss: 0.4944
Epoch 2/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 188s 2s/step - accuracy: 0.6999 - loss: 0.5629 - val_accuracy: 0.8300 - val_loss: 0.4217
Epoch 3/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 124s 1s/step - accuracy: 0.7726 - loss: 0.4817 - val_accuracy: 0.8280 - val_loss: 0.3885
Epoch 4/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 127s 1s/step - accuracy: 0.7806 - loss: 0.4596 - val_accuracy: 0.8470 - val_loss: 0.3512
Epoch 5/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 128s 1s/step - accuracy: 0.8122 - loss: 0.4338 - val_accuracy: 0.8610 - val_loss: 0.3313
Epoch 6/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 129s 1s/step - accuracy: 0.8047 - loss: 0.4178 - val_accuracy: 0.8620 - val_loss: 0.3181
Epoch 7/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 130s 1s/step - accuracy: 0.8153 - loss: 0.3823 - val_accuracy: 0.8690 - val_loss: 0.3064
Epoch 8/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 130s 1s/step - accuracy: 0.8443 - loss: 0.3569 - val_accuracy: 0.8720 - val_loss: 0.2975
Epoch 9/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 132s 1s/step - accuracy: 0.8293 - loss: 0.3777 - val_accuracy: 0.8690 - val_loss: 0.3025
Epoch 10/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 131s 1s/step - accuracy: 0.8446 - loss: 0.3544 - val_accuracy: 0.8730 - val_loss: 0.2888
Epoch 11/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 132s 1s/step - accuracy: 0.8265 - loss: 0.3804 - val_accuracy: 0.8720 - val_loss: 0.2928
Epoch 12/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 132s 1s/step - accuracy: 0.8303 - loss: 0.3672 - val_accuracy: 0.8760 - val_loss: 0.2907
Epoch 13/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 131s 1s/step - accuracy: 0.8420 - loss: 0.3543 - val_accuracy: 0.8760 - val_loss: 0.2853
Epoch 14/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 132s 1s/step - accuracy: 0.8539 - loss: 0.3267 - val_accuracy: 0.8740 - val_loss: 0.2750
Epoch 15/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 131s 1s/step - accuracy: 0.8602 - loss: 0.3221 - val_accuracy: 0.8800 - val_loss: 0.2725
Epoch 16/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 133s 1s/step - accuracy: 0.8543 - loss: 0.3292 - val_accuracy: 0.8820 - val_loss: 0.2711
Epoch 17/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 131s 1s/step - accuracy: 0.8470 - loss: 0.3374 - val_accuracy: 0.8820 - val_loss: 0.2677
Epoch 18/30
```


```
Epoch 17/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 131s 1s/step - accuracy: 0.8470 - loss: 0.3374 - val_accuracy: 0.8820 - val_loss: 0.2677
Epoch 18/30
```

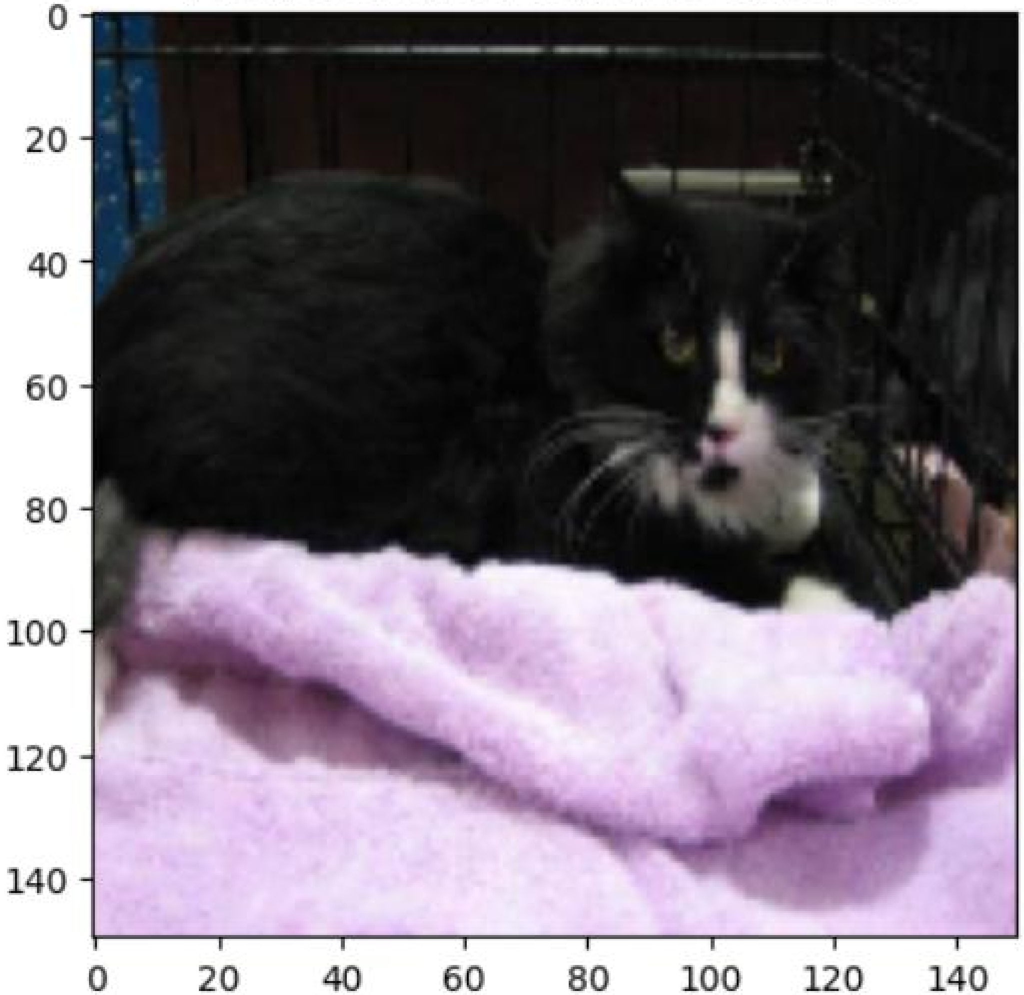

```
100/100 ━━━━━━━━━━━━━━━━━━━━ 131s 1s/step - accuracy: 0.8481 - loss: 0.3185 - val_accuracy: 0.8790 - val_loss: 0.2696
Epoch 19/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 132s 1s/step - accuracy: 0.8702 - loss: 0.2956 - val_accuracy: 0.8830 - val_loss: 0.2718
Epoch 20/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 131s 1s/step - accuracy: 0.8602 - loss: 0.3240 - val_accuracy: 0.8850 - val_loss: 0.2736
Epoch 21/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 132s 1s/step - accuracy: 0.8687 - loss: 0.3232 - val_accuracy: 0.8720 - val_loss: 0.2729
Epoch 22/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 131s 1s/step - accuracy: 0.8595 - loss: 0.3172 - val_accuracy: 0.8810 - val_loss: 0.2626
Epoch 23/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 132s 1s/step - accuracy: 0.8659 - loss: 0.3095 - val_accuracy: 0.8870 - val_loss: 0.2676
Epoch 24/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 132s 1s/step - accuracy: 0.8739 - loss: 0.3086 - val_accuracy: 0.8840 - val_loss: 0.2647
Epoch 25/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 131s 1s/step - accuracy: 0.8902 - loss: 0.2802 - val_accuracy: 0.8810 - val_loss: 0.2597
Epoch 26/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 132s 1s/step - accuracy: 0.8800 - loss: 0.2985 - val_accuracy: 0.8800 - val_loss: 0.2595
Epoch 27/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 131s 1s/step - accuracy: 0.8613 - loss: 0.3291 - val_accuracy: 0.8860 - val_loss: 0.2603
Epoch 28/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 133s 1s/step - accuracy: 0.8743 - loss: 0.2973 - val_accuracy: 0.8840 - val_loss: 0.2567
Epoch 29/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 131s 1s/step - accuracy: 0.8615 - loss: 0.3243 - val_accuracy: 0.8830 - val_loss: 0.2552
Epoch 30/30
100/100 ━━━━━━━━━━━━━━━━━━━━ 132s 1s/step - accuracy: 0.8827 - loss: 0.2882 - val_accuracy: 0.8830 - val_loss: 0.2548
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 978ms/step
```

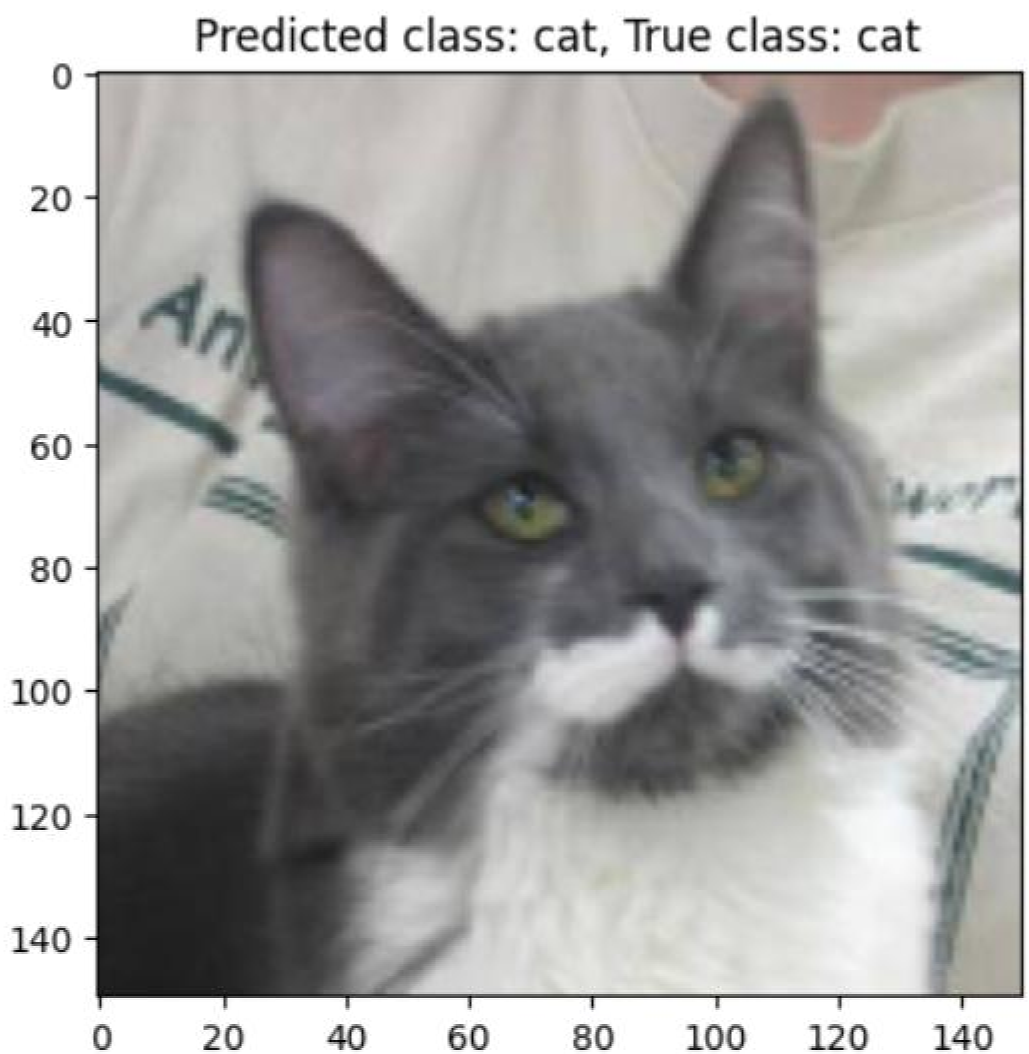

Predicted class: cat, True class: dog
0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

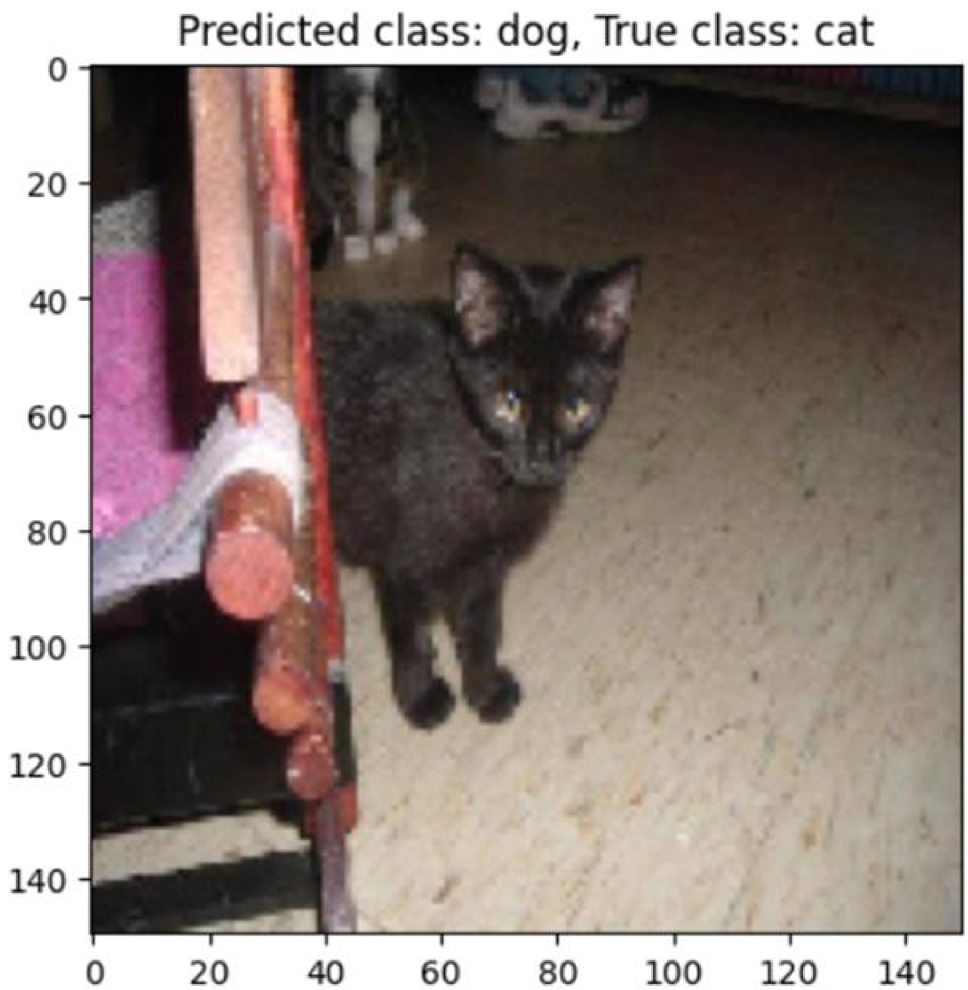

Predicted class: dog, True class: dog
0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

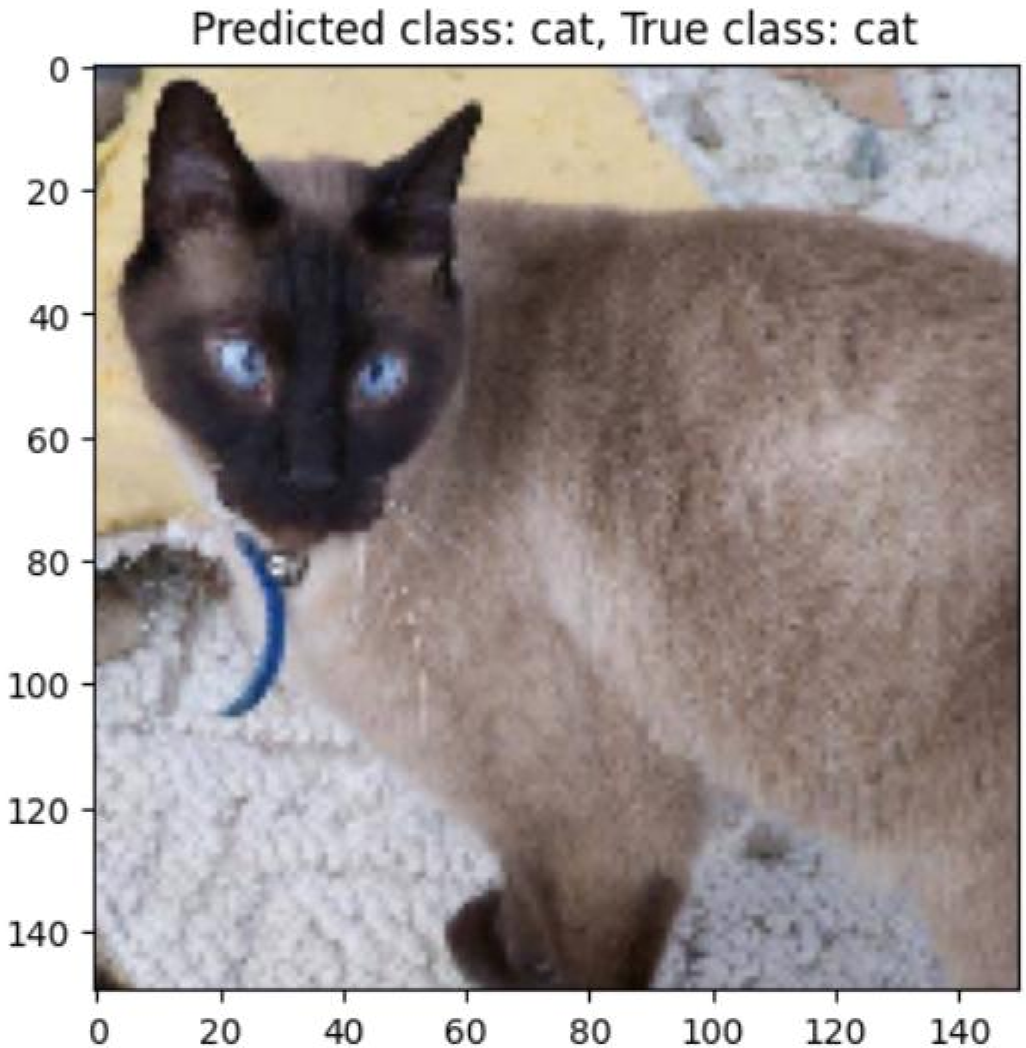

Predicted class: dog, True class: dog
0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

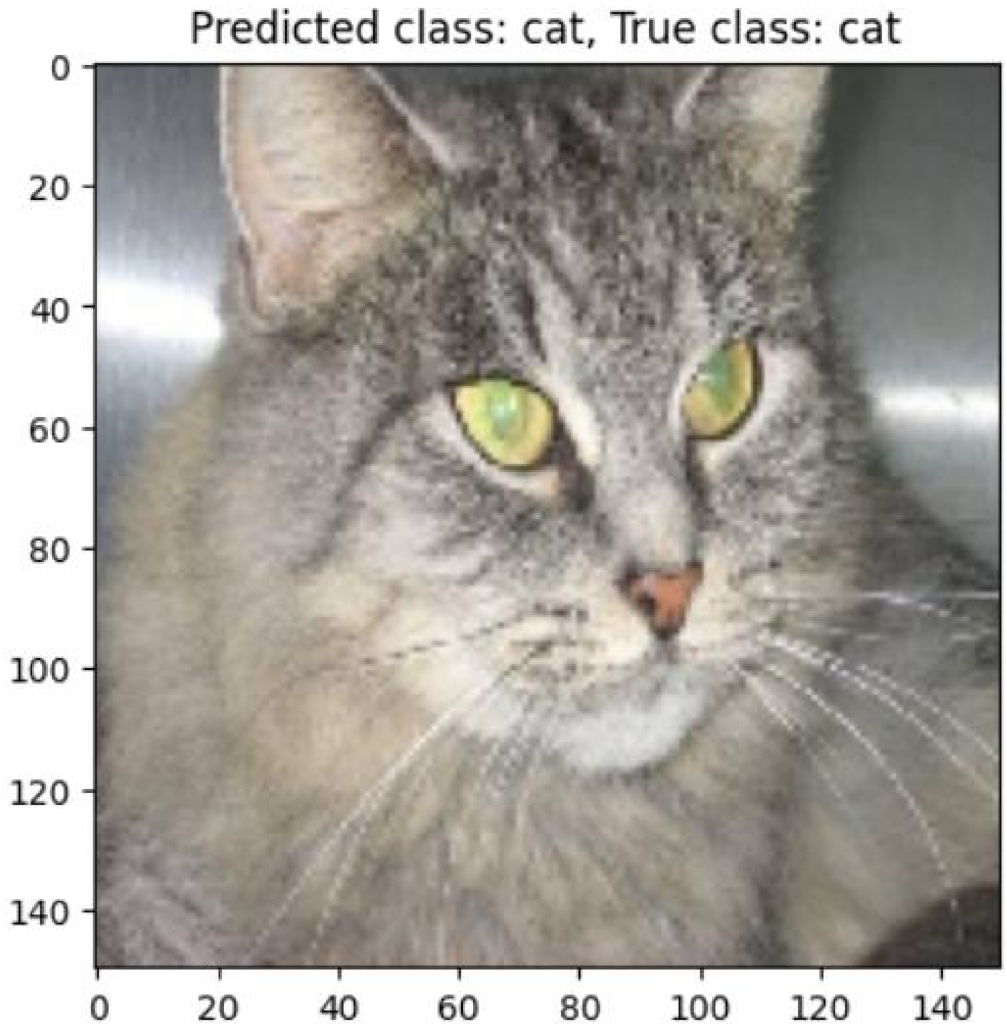
Predicted class: cat, True class: cat

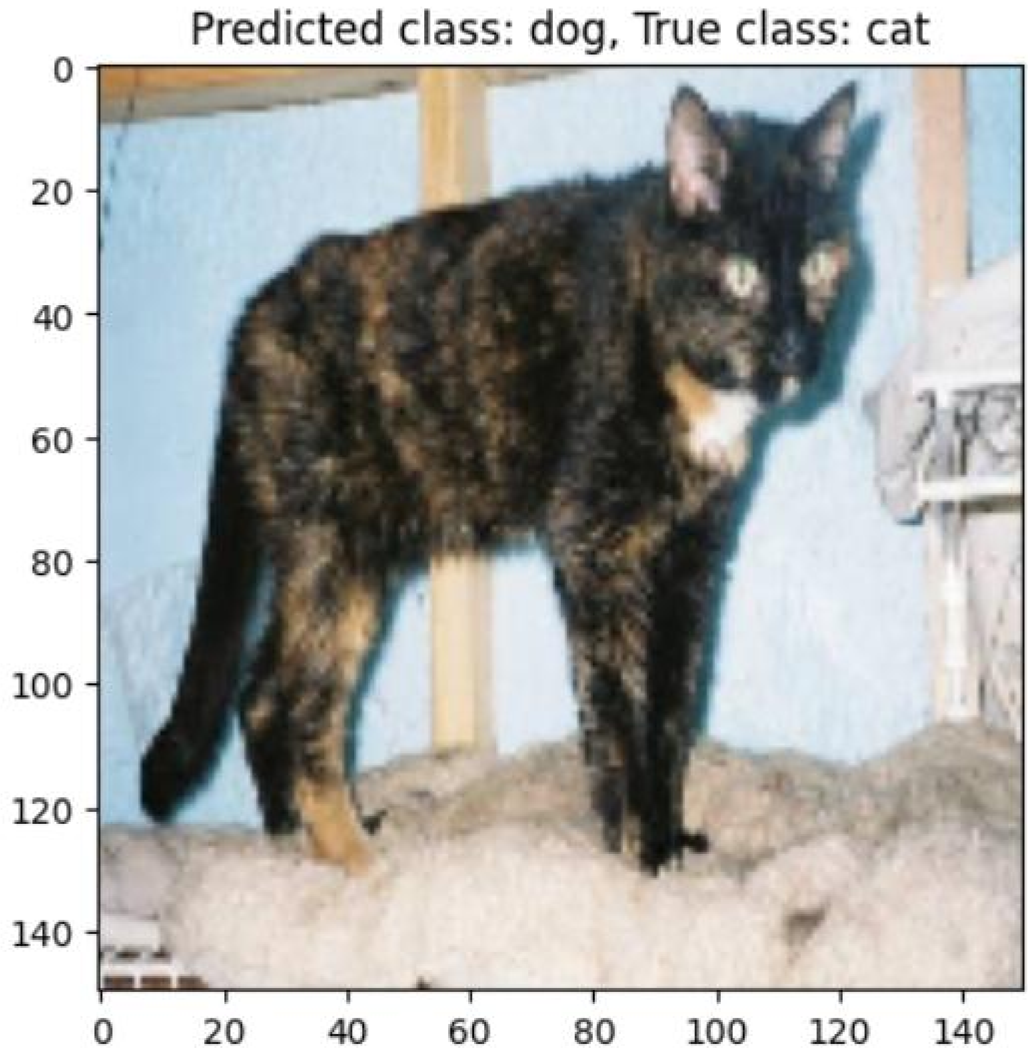


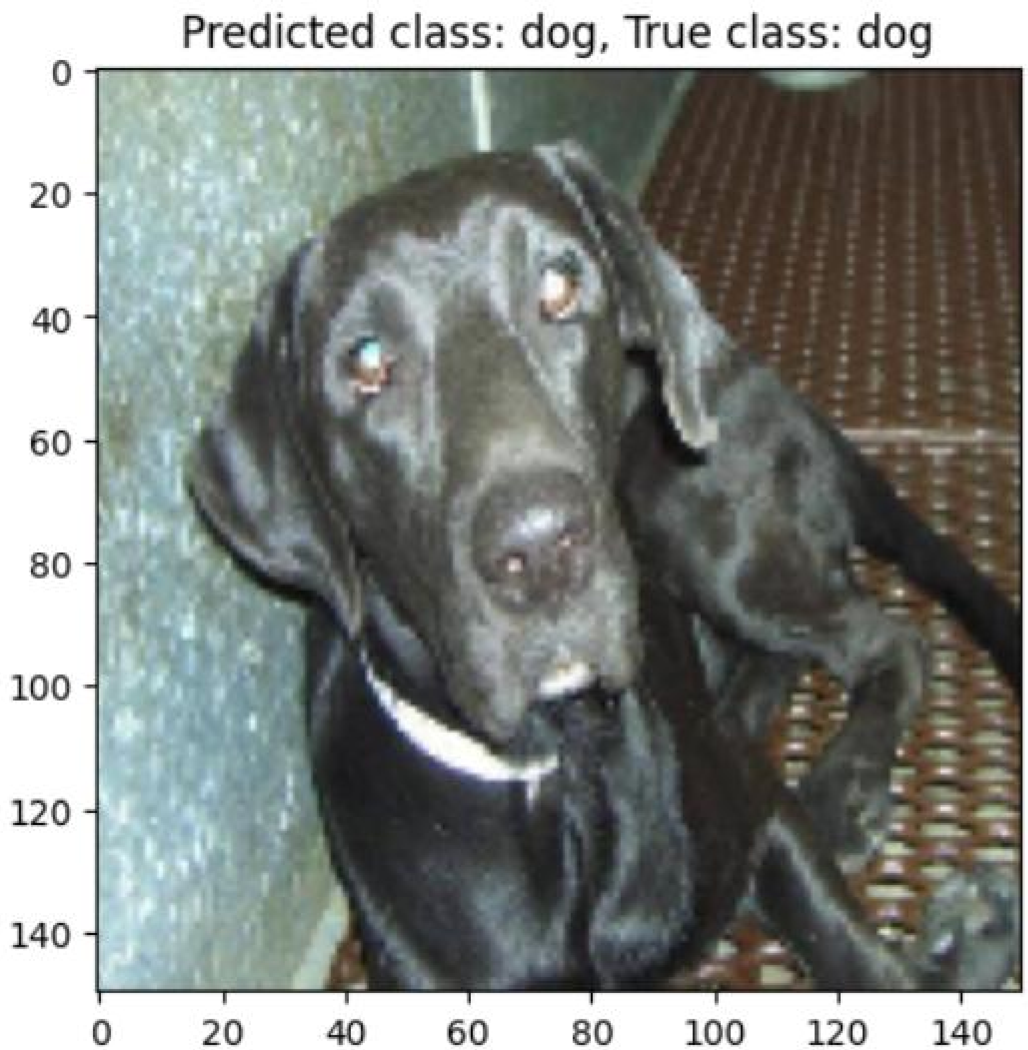

Predicted class: cat, True class: cat

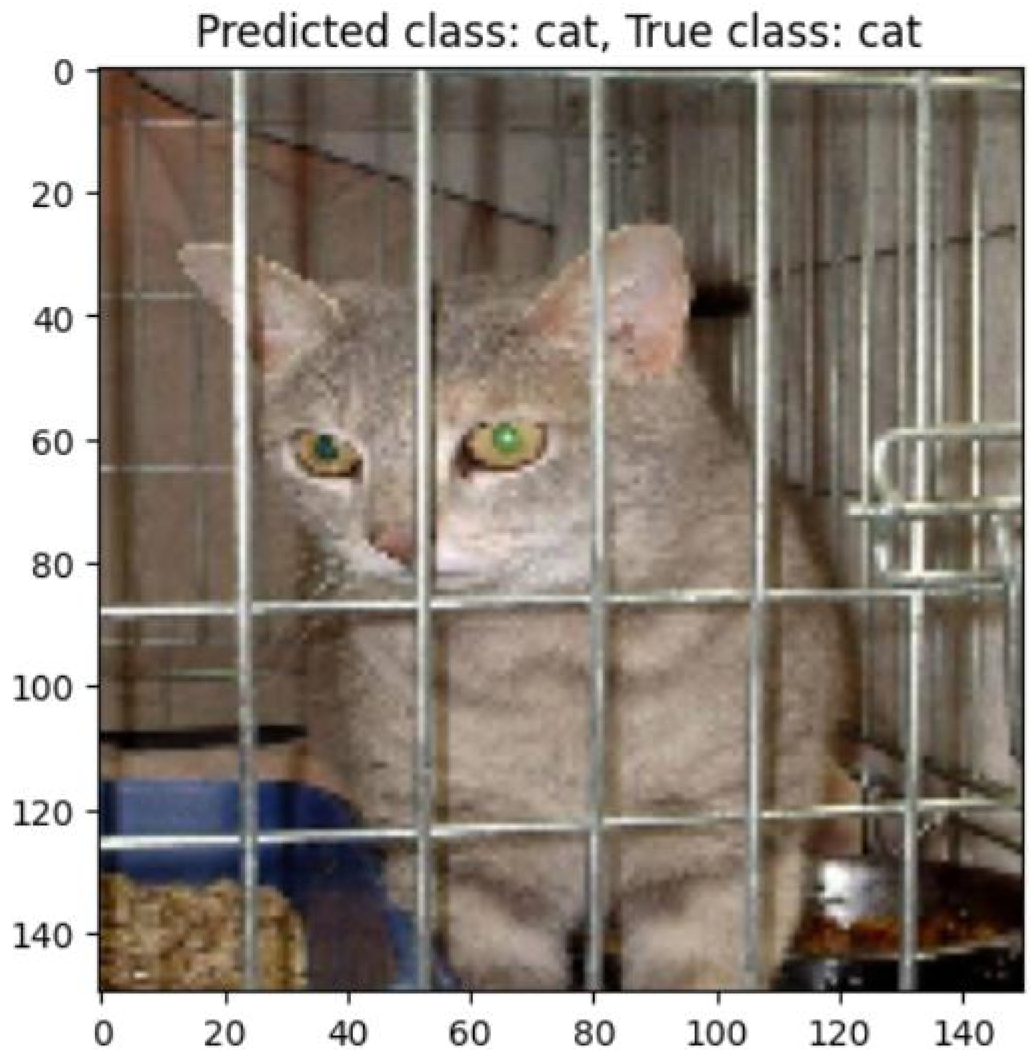

Predicted class: dog, True class: cat
0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

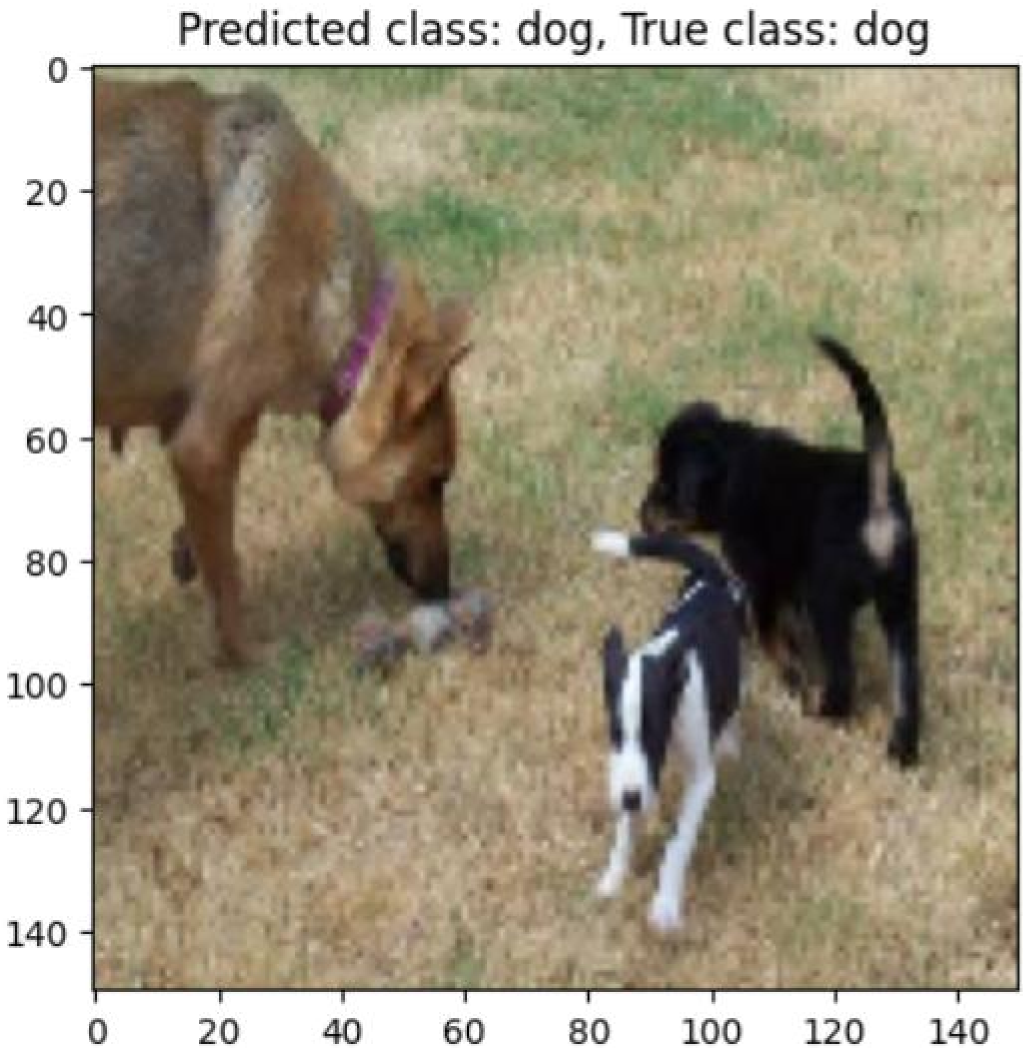

Predicted class: cat, True class: cat
0
20
40
60
80
100
120
140

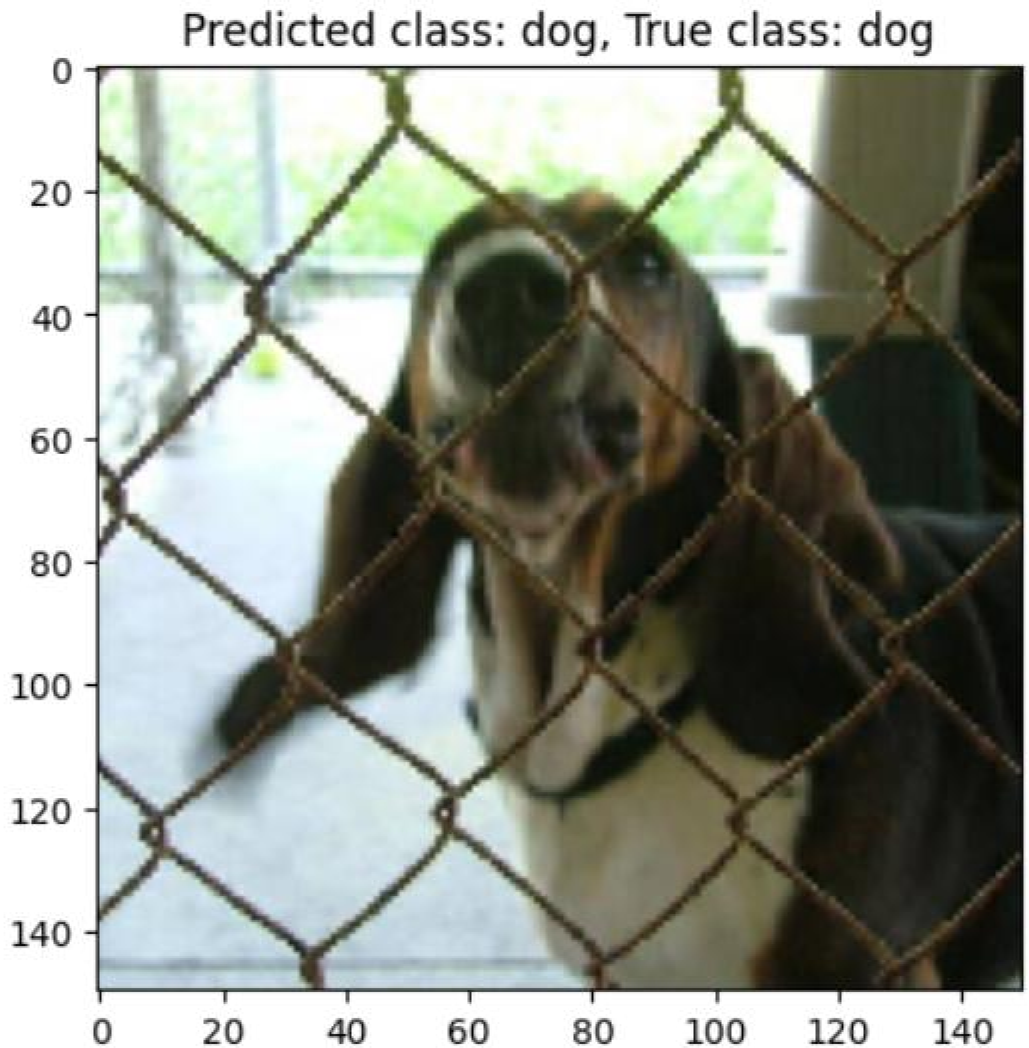

Predicted class: cat, True class: cat
0
20
40
60
80
100
120
140


Predicted class: dog, True class: cat
0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

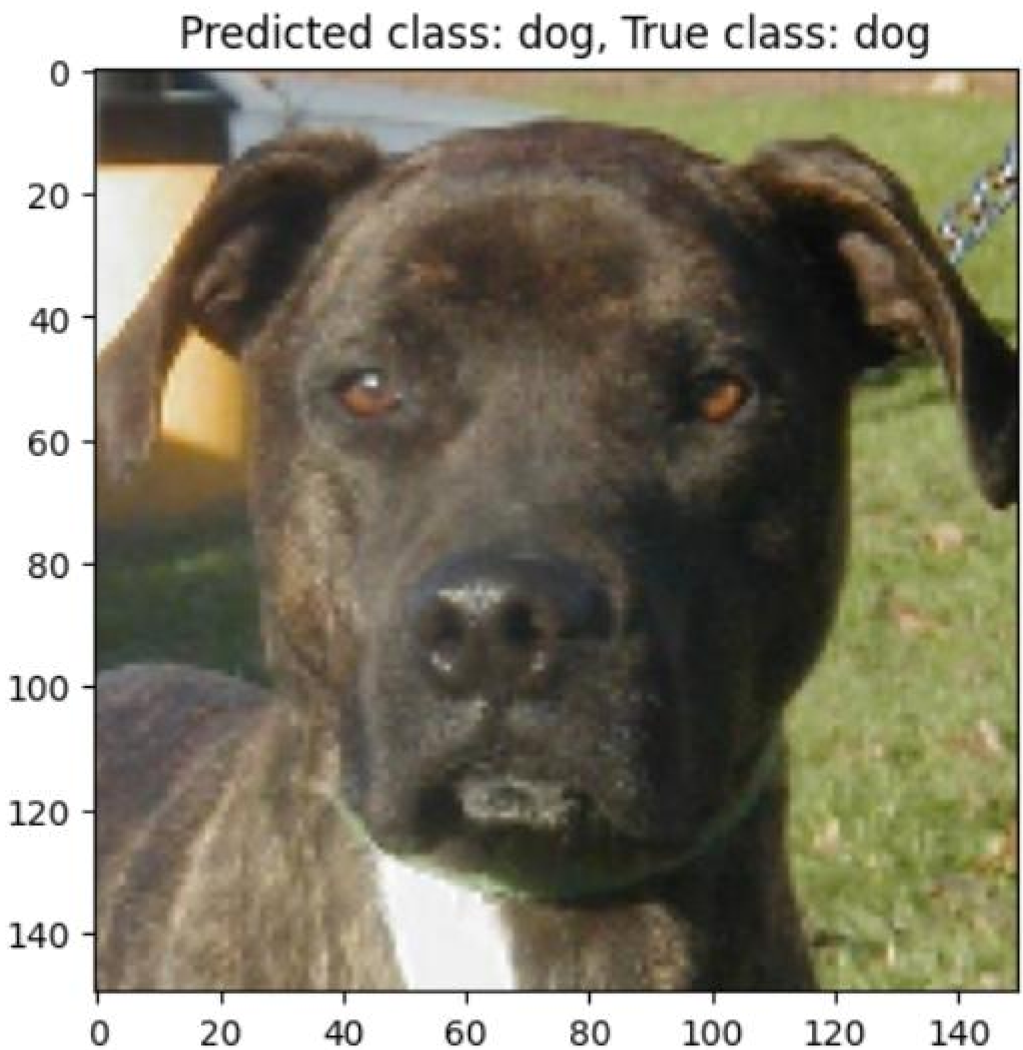

Predicted class: dog, True class: dog
0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

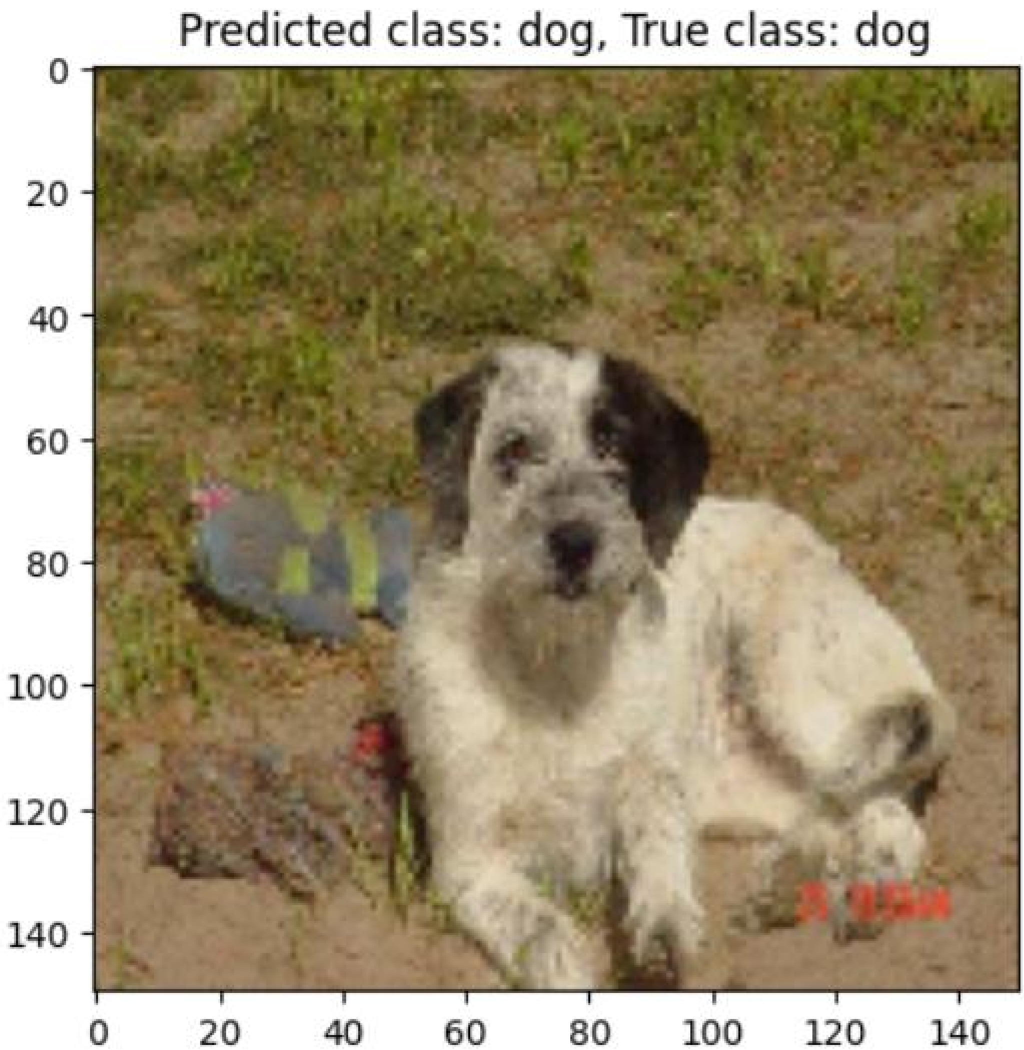

Predicted class: cat, True class: cat
0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


Predicted class: dog, True class: dog
0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

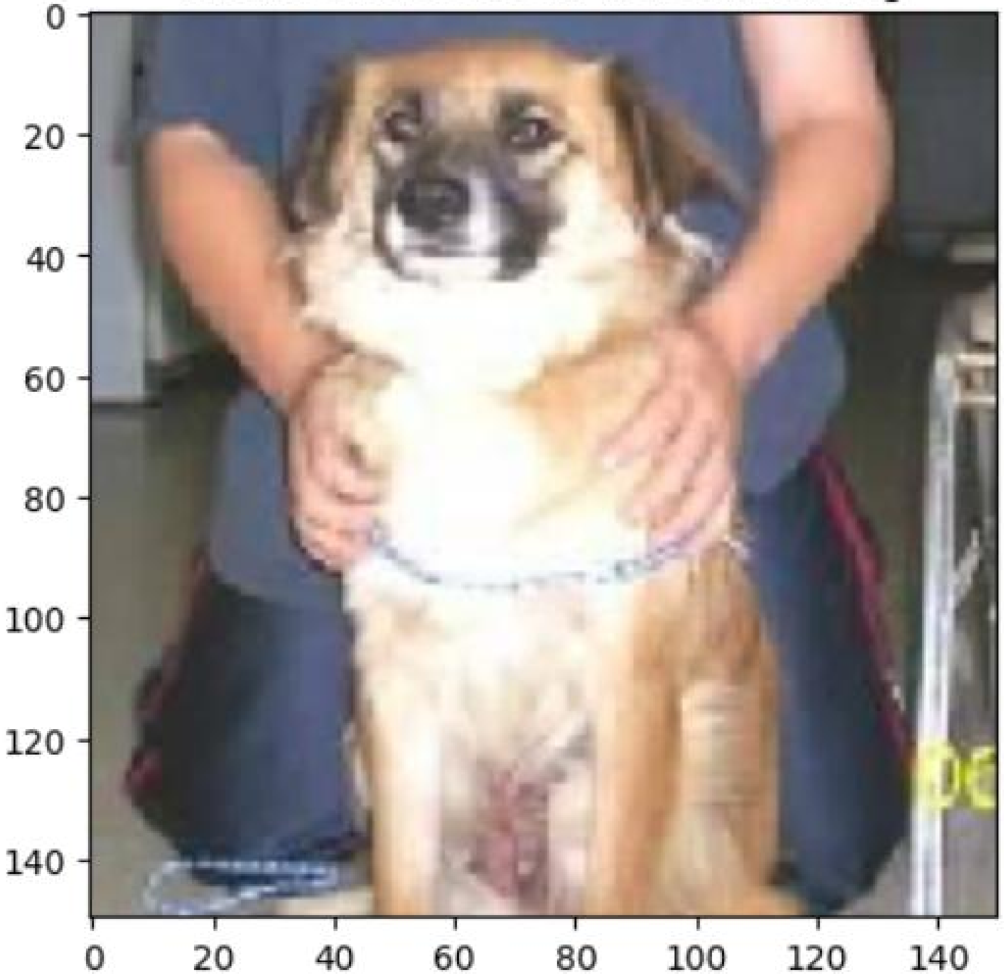

Predicted class: dog, True class: dog
0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

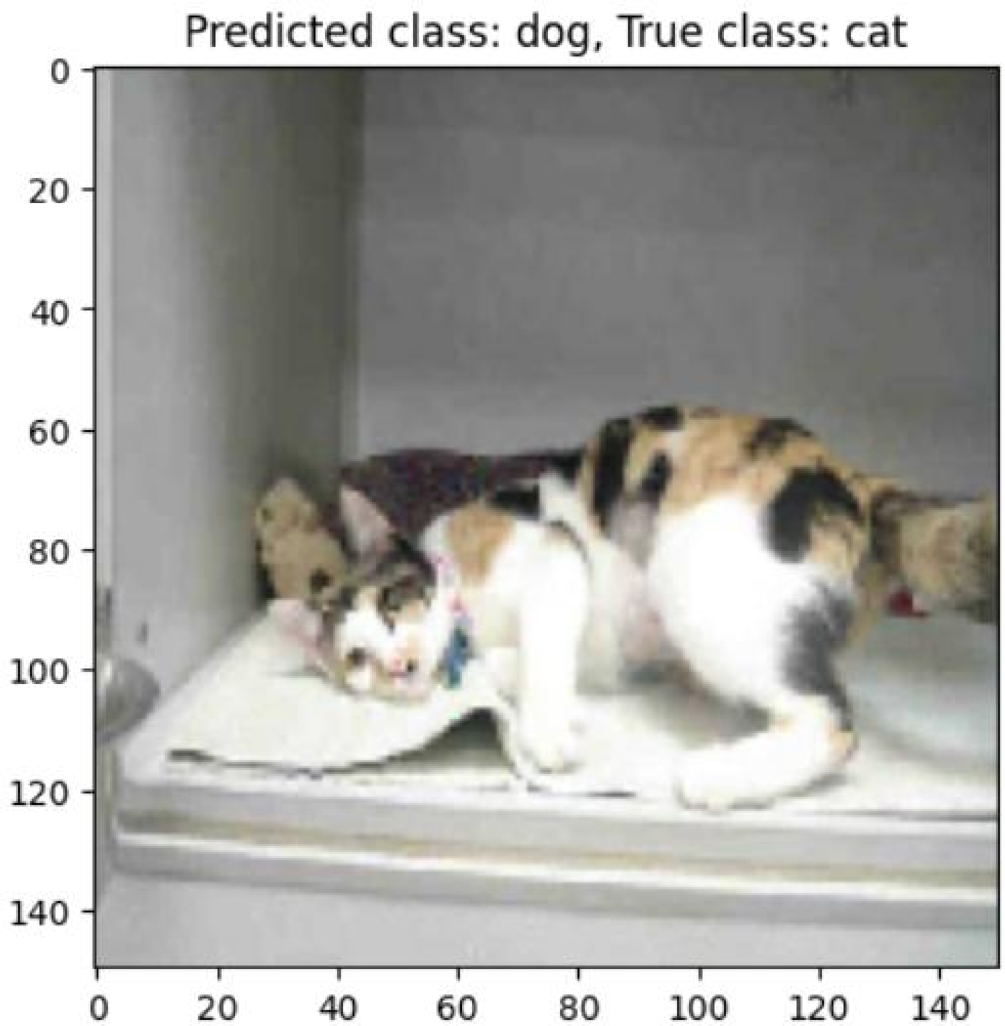

Predicted class: cat, True class: cat


Predicted class: dog, True class: dog
0
20
40
60
80
100
120
140

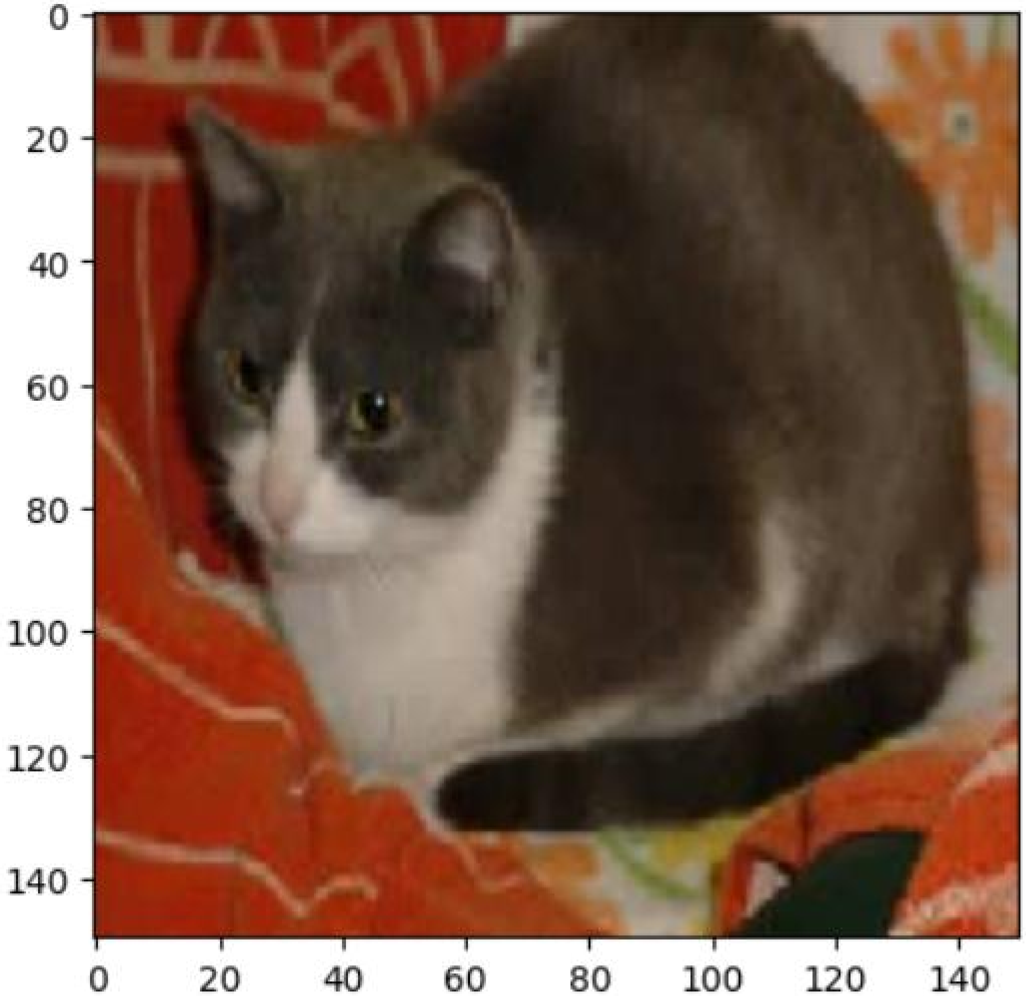

Predicted class: dog, True class: dog
0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

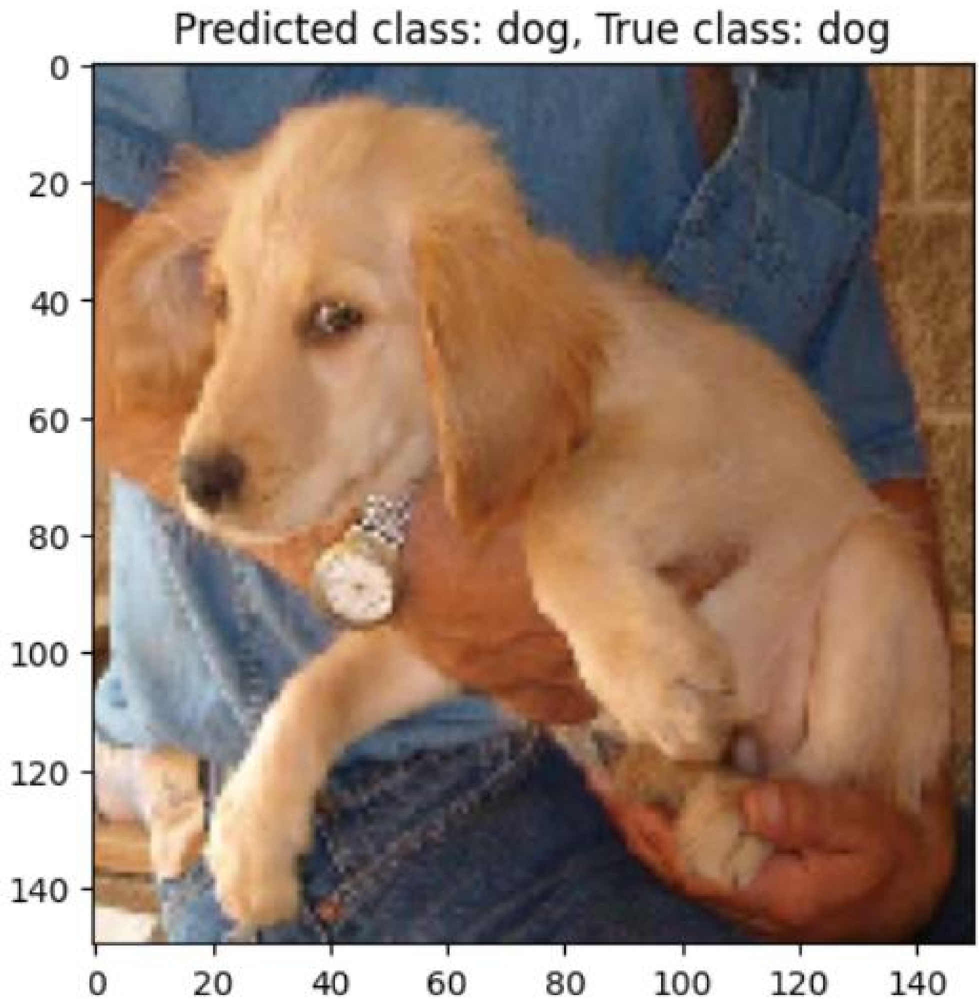
Predicted class: cat, True class: dog

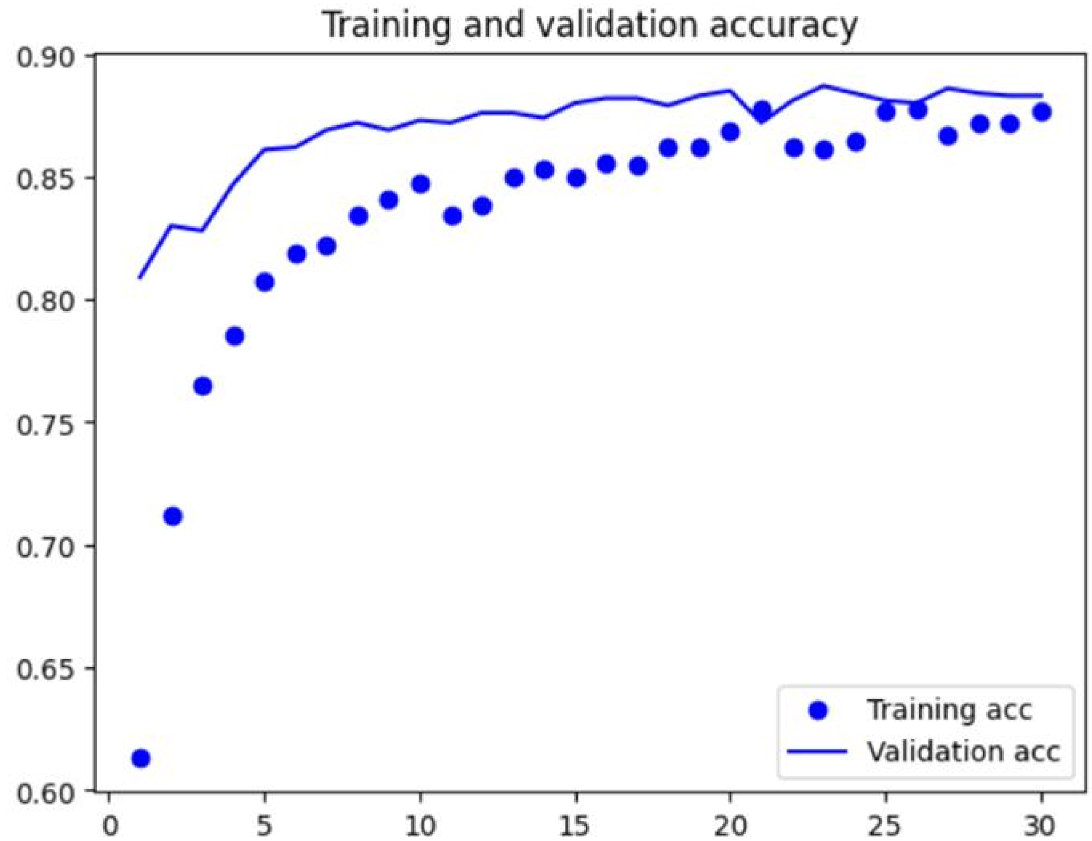


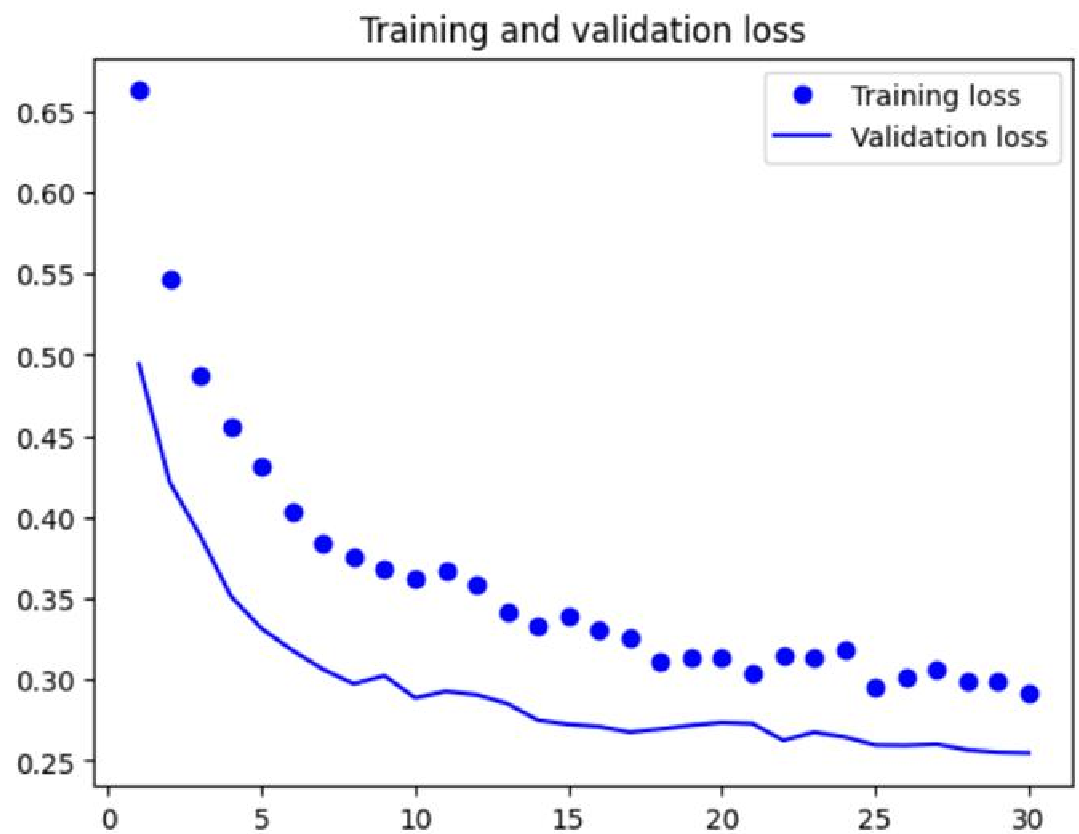

Predicted class: dog, True class: cat
0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

Predicted class: cat, True class: cat




Predicted class: dog, True class: dog
0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140





Gayatri kulkarni-53004230002

Practical 8

Aim: Implement basic reinforcement learning algorithms such as Q-learning and policy gradients.

Apply these algorithms to solve simple Markov decision processes (MDPs).

Experiment with different reward structures and explore their impact on learning.

Practical 9

Aim: Implement a Variational Autoencoder (VAE) architecture for learning latent representations

Train the VAE model on a dataset and visualize the learned latent space

Explore techniques for generating new data samples using the trained VAE model

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist

#load
(x_train, _), (x_test, _) = mnist.load_data()

#normalize
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train),np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test),np.prod(x_test.shape[1:])))

#size
```

```
encoding_dim = 32
#input
input_img = Input(shape=(784,))
#encoded
encoded = Dense(encoding_dim, activation='relu')(input_img)
#decoded
decoded = Dense(784, activation='sigmoid')(encoded)
#this model maps its reconstruction
autoencoder = Model(input_img, decoded)
#this model maps its  encoded representation
encoder = Model(input_img, encoded)
#create
encoded_input = Input(shape=(encoding_dim,))
#retrive
decoder_layer =autoencoder.layers[-1]
#create
decoder = Model(encoded_input, decoder_layer(encoded_input))

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
#encode
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
```

```
#use
n=10
plt.figure(figsize=(20,4))
for i in range(n):
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
print("Gayatri Kulkarni -53004230002")
```

```
Epoch 1/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.3821 - val_loss: 0.1891
Epoch 2/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.1800 - val_loss: 0.1538
Epoch 3/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.1494 - val_loss: 0.1342
Epoch 4/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.1318 - val_loss: 0.1218
Epoch 5/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1207 - val_loss: 0.1138
Epoch 6/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.1135 - val_loss: 0.1084
Epoch 7/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.1082 - val_loss: 0.1043
Epoch 8/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.1047 - val_loss: 0.1009
Epoch 9/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 0.1015 - val_loss: 0.0982
Epoch 10/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0989 - val_loss: 0.0963
Epoch 11/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0973 - val_loss: 0.0949
Epoch 12/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.0957 - val_loss: 0.0941
Epoch 13/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0952 - val_loss: 0.0935
Epoch 14/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.0948 - val_loss: 0.0932
Epoch 15/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.0945 - val_loss: 0.0929
Epoch 16/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0943 - val_loss: 0.0927
Epoch 17/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0941 - val_loss: 0.0925
Epoch 18/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0938 - val_loss: 0.0924
Epoch 19/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0938 - val_loss: 0.0923
Epoch 20/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0937 - val_loss: 0.0922
Epoch 21/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0937 - val_loss: 0.0922
Epoch 22/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.0935 - val_loss: 0.0921
Epoch 23/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0932 - val_loss: 0.0920
Epoch 24/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0935 - val_loss: 0.0920
Epoch 25/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0931 - val_loss: 0.0920
```

```
Epoch 26/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0932 - val_loss: 0.0919
Epoch 27/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0932 - val_loss: 0.0919
Epoch 28/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0929 - val_loss: 0.0920
Epoch 29/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0930 - val_loss: 0.0918
Epoch 30/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0931 - val_loss: 0.0918
Epoch 31/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0929 - val_loss: 0.0918
Epoch 32/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0930 - val_loss: 0.0918
Epoch 33/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0928 - val_loss: 0.0918
Epoch 34/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0929 - val_loss: 0.0918
Epoch 35/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0929 - val_loss: 0.0918
Epoch 36/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0928 - val_loss: 0.0918
Epoch 37/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0926 - val_loss: 0.0917
Epoch 38/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0931 - val_loss: 0.0917
Epoch 39/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0929 - val_loss: 0.0916
Epoch 40/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0926 - val_loss: 0.0917
Epoch 41/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0928 - val_loss: 0.0917
Epoch 42/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0928 - val_loss: 0.0916
Epoch 43/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.0927 - val_loss: 0.0917
Epoch 44/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.0927 - val_loss: 0.0916
Epoch 45/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.0927 - val_loss: 0.0917
Epoch 46/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0929 - val_loss: 0.0916
Epoch 47/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0927 - val_loss: 0.0916
Epoch 48/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - loss: 0.0927 - val_loss: 0.0915
Epoch 49/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.0924 - val_loss: 0.0916
Epoch 50/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0928 - val_loss: 0.0916
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

```
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.0924 - val_loss: 0.0916
Epoch 50/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - loss: 0.0928 - val_loss: 0.0916
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

```
Gayatri Kulkarni -53004230002
```

Practical 10

Aim: Implement a basic GAN architecture (e.g., DCGAN) using TensorFlow or PyTorch.

Train the GAN model on a dataset for image generation or style transfer tasks.

Experiment with different loss functions and architectures to improve GAN performance.

Practical 11

Aim: Use pre-trained GAN models for image generation and synthesis tasks.

use of GANs for text generation tasks such as dialogue generation or story generation

Tensorfloiw operations