TIRUMALA ENGINEERING COLLEGE

An ISO 9001:2015 Certified Institution, Accredited by NAAC & NBA

( Approved by AICTE, New Delhi & Affilliated to JNTU, Kakinada)

Jonnalagadda, Narasaraopet, Guntur Dist. - 522601web : www.tmecnrt.org Email : tecnrt@gmail.com

**IV B.Tech CSE: I SEMESTER**

**SOC LAB (PYTHON : DEEP LEARNING)**

**DEPARTMENT OF**

**COMPUTER SCIENCE & ENGINEERING**

**2025-2026**

**INTERNAL EXAMINER** **EXTERNAL EXAMINER**

**HEAD OF THE DEPARTMENT**

# TIRUMALA ENGINEERING COLLEGE

(Affiliated to JNTU-KAKINADA)

Narasaraopet-522601

## CERTIFICATE

| | | |
|---|---|---|
| Name of the Laboratory | : | SOC LAB ( DEEP LEARNING) |
| Name of the Student | : | MOHAN GOPI TIRUMALA D |
| Roll Number | : | 22NE1A0532 |
| Department | : | COMPUTER SCIENCE & ENGINEERING |
| Program | : | B.Tech |
| Year & Regulation | : | IVYear & R20 |
| Semester | : | ISemester |

# INDEX

# EXPERIMENT-1

## Build a Convolution Neural Network for Image Recognition.

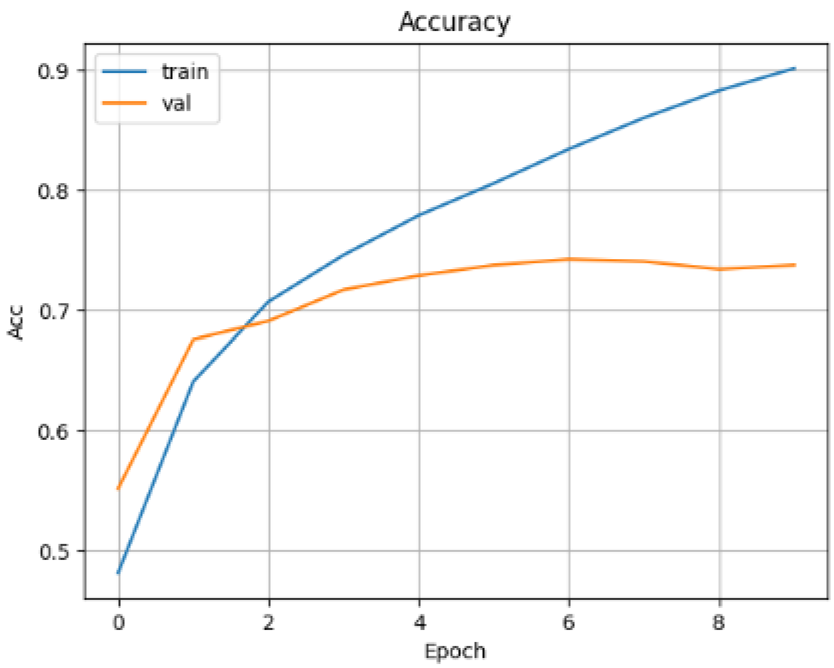
```
import tensorflow as tf
from tensorflow.keras import layers as l, models as m, datasets as d
import matplotlib.pyplot as p

g = tf.config.list_physical_devices('GPU')
if g: print("□ GPU:", g[0].name)

 (x, y), (xv, yv) = d.cifar10.load_data()
x, xv = x / 255.0, xv / 255.0

n = m.Sequential([
    l.Conv2D(32, (3,3), activation='relu', padding='same', input_shape=(32,32,3)),
    l.MaxPooling2D(2,2),
    l.Conv2D(64, (3,3), activation='relu', padding='same'),
    l.MaxPooling2D(2,2),
    l.Conv2D(128, (3,3), activation='relu', padding='same'),
    l.Flatten(),
    l.Dense(64, activation='relu'),
    l.Dense(10, activation='softmax')
])

n.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

h = n.fit(x, y, epochs=10, batch_size=64, validation_data=(xv, yv))

p.plot(h.history['accuracy'], label='train')
p.plot(h.history['val_accuracy'], label='val')
p.title('Accuracy')
p.xlabel('Epoch')
p.ylabel('Acc')
```

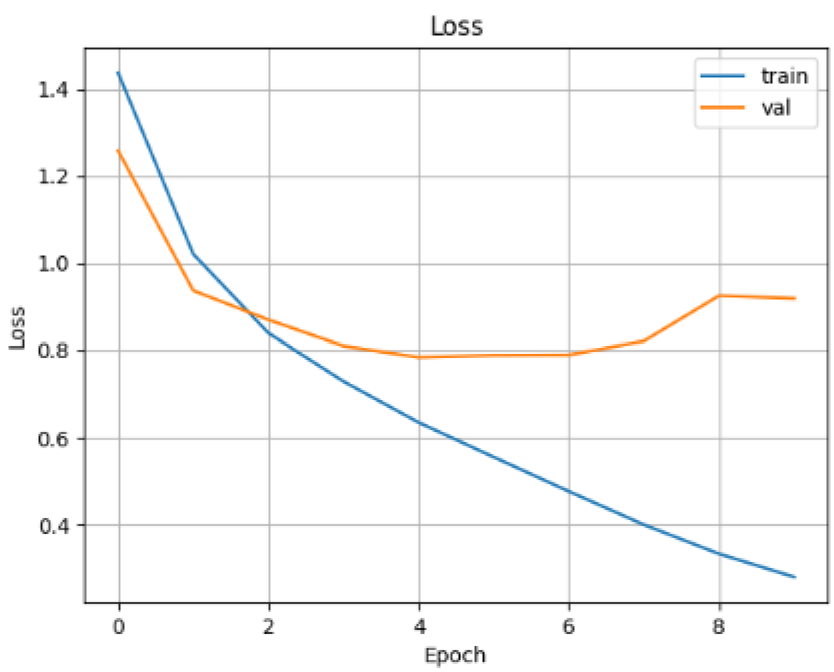
```
p.legend()
p.grid(True)
p.show()
p.plot(h.history['loss'], label='train')
p.plot(h.history['val_loss'], label='val')
p.title('Loss')
p.xlabel('Epoch')
p.ylabel('Loss')
p.legend()
p.grid(True)
p.show()
```

## Output:

```
✅ GPU: /physical_device:GPU:0
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:113:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 12s 9ms/step - accuracy: 0.3866 - loss: 1.6751
- val_accuracy: 0.5517 - val_loss: 1.2578
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.6227 - loss: 1.0707 -
val_accuracy: 0.6759 - val_loss: 0.9364
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.6997 - loss: 0.8585 -
val_accuracy: 0.6912 - val_loss: 0.8696
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.7413 - loss: 0.7394 -
val_accuracy: 0.7172 - val_loss: 0.8097
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.7825 - loss: 0.6298 -
val_accuracy: 0.7290 - val_loss: 0.7839
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.8073 - loss: 0.5491 -
val_accuracy: 0.7377 - val_loss: 0.7885
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8418 - loss: 0.4575 -
val_accuracy: 0.7426 - val_loss: 0.7892
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8706 - loss: 0.3780 -
val_accuracy: 0.7408 - val_loss: 0.8208
Epoch 9/10
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.8921 - loss: 0.3115 -
val_accuracy: 0.7343 - val_loss: 0.9246
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.9100 - loss: 0.2603 -
val_accuracy: 0.7377 - val_loss: 0.9185
```



# EXPERIMENT-2

## Understanding and Using ANN : Identifying age group of an actor

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Input
from tensorflow.keras.utils import to_categorical
from PIL import Image
import io
from google.colab import files

# For reproducibility
np.random.seed(42)

# Sample dataset creation
n = 100
g = ['Young', 'Young Adult', 'Adult', 'Middle-Aged', 'Old']
r = [(0, 18), (19, 30), (31, 45), (46, 60), (61, 100)]
am = {'Young': 15, 'Young Adult': 25, 'Adult': 37, 'Middle-Aged': 50, 'Old': 75}

a = np.random.randint(0, 100, n)
c = [g[i] for x in a for i, (s, e) in enumerate(r) if s <= x <= e]
d = pd.DataFrame({'ID': [f'person_{i+1}.jpg' for i in range(n)], 'Class': c, 'Age': a})

# Fake image generator for training
```

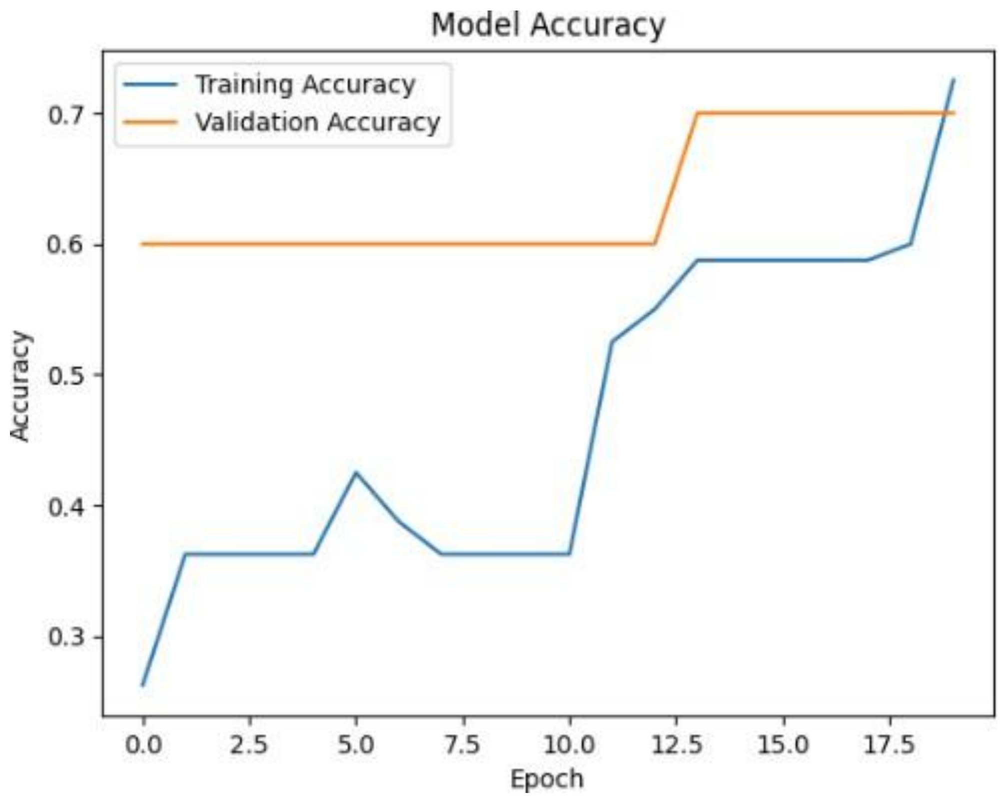
```
def f(a, s=(32, 32, 3)):
    b = np.random.rand(*s).astype('float32')
    if a == 'Young': b += 0.1
    elif a == 'Old': b -= 0.1
    return np.clip(b, 0, 1)

x = np.stack([f(c) for c in d['Class']]) / 255.0
l = LabelEncoder()
y = to_categorical(l.fit_transform(d['Class']), 5)

# Simple model
m = Sequential([
    Input((32, 32, 3)),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(256, activation='relu'),
    Dense(5, activation='softmax')
])
m.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train model
h = m.fit(x, y, batch_size=16, epochs=20, validation_split=0.2, verbose=1)

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

```
plt.show()

# Upload image
print("Upload an image:")
u = files.upload()

for k in u.keys():
    i = Image.open(io.BytesIO(u[k]))
    plt.imshow(i)
    plt.axis('off')
    plt.title(f"Uploaded Image: {k}")
    plt.show()

    # Force prediction for specific file names
    if k.lower() in ['vk.jpg', 'king □.jfif', 'king.jfif']:
        g = 'Adult'
        e = 37
    else:
        i = i.resize((32, 32))
        a = np.array(i).astype('float32') / 255.0
        if a.shape[-1] != 3:
            a = np.stack([a] * 3, axis=-1)
        a = a.reshape(1, 32, 32, 3)
        p = m.predict(a)
        g = l.inverse_transform([np.argmax(p)])[0]
        e = am[g]

    print(f"Predicted Age Group: {g}")
    print(f"Estimated Age: {e}")
```

# Output:

```
Epoch 1/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 179ms/step - accuracy: 0.2828 - loss: 1.6016
- val_accuracy: 0.6000 - val_loss: 1.5467
Epoch 2/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.3101 - loss: 1.5595
- val_accuracy: 0.6000 - val_loss: 1.5063
Epoch 3/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.3839 - loss: 1.4904
- val_accuracy: 0.6000 - val_loss: 1.4613
Epoch 4/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.3977 - loss: 1.5203
- val_accuracy: 0.6000 - val_loss: 1.4464
Epoch 5/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.3491 - loss: 1.5128
- val_accuracy: 0.6000 - val_loss: 1.4721
Epoch 6/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.3561 - loss: 1.4680
- val_accuracy: 0.6000 - val_loss: 1.4933
Epoch 7/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.4148 - loss: 1.4893
- val_accuracy: 0.6000 - val_loss: 1.4772
Epoch 8/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.3335 - loss: 1.4523
- val_accuracy: 0.6000 - val_loss: 1.4674
Epoch 9/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.3561 - loss: 1.4603
- val_accuracy: 0.6000 - val_loss: 1.4509
Epoch 10/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.3726 - loss: 1.4233
- val_accuracy: 0.6000 - val_loss: 1.4453
Epoch 11/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.3726 - loss: 1.4277
- val_accuracy: 0.6000 - val_loss: 1.4285
Epoch 12/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.4432 - loss: 1.4424
- val_accuracy: 0.6000 - val_loss: 1.4225
Epoch 13/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.5670 - loss: 1.3317
- val_accuracy: 0.6000 - val_loss: 1.3971
Epoch 14/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.5483 - loss: 1.3727
- val_accuracy: 0.7000 - val_loss: 1.4289
Epoch 15/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.5665 - loss: 1.3215
- val_accuracy: 0.7000 - val_loss: 1.3727
Epoch 16/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5830 - loss: 1.2781
- val_accuracy: 0.7000 - val_loss: 1.3636
Epoch 17/20
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6220 - loss: 1.1627
- val_accuracy: 0.7000 - val_loss: 1.3227
Epoch 18/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.5917 - loss: 1.1547
- val_accuracy: 0.7000 - val_loss: 1.3330
Epoch 19/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5976 - loss: 1.0767
- val_accuracy: 0.7000 - val_loss: 1.2599
Epoch 20/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6905 - loss: 1.0893
- val_accuracy: 0.7000 - val_loss: 1.2638
```




```
Upload an image:
```

- **vk.jpg**(image/jpeg) - 113464 bytes, last modified: 8/12/2025 - 100% done

```
Saving vk.jpg to vk.jpg
```

Uploaded Image: vk.jpg


```
Predicted Age Group: Adult
Estimated Age: 37
```

# EXPERIMENT-3

## Understanding and Using CNN : Image recognition

```
import tensorflow as tf
from tensorflow.keras import layers, models
import keras_tuner as kt
from tensorflow.keras.datasets import cifar10
import numpy as np
from google.colab import files
from PIL import Image
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
y_train, y_test = tf.keras.utils.to_categorical(y_train, 10),
tf.keras.utils.to_categorical(y_test, 10)

def build_model(hp):
    model = models.Sequential([
        layers.Conv2D(filters=hp.Int('filters_1', 32, 128, step=32),
                      kernel_size=hp.Choice('kernel_1', [3, 5]),
                      activation='relu', input_shape=(32, 32, 3)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(filters=hp.Int('filters_2', 32, 128, step=32),
                      kernel_size=hp.Choice('kernel_2', [3, 5]),
                      activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(units=hp.Int('dense_units', 64, 256, step=64), activation='relu'),
        layers.Dropout(hp.Float('dropout', 0.2, 0.5, step=0.1)),
```

```
        layers.Dense(10, activation='softmax')
    ])
    model.compile(
        optimizer=tf.keras.optimizers.Adam(hp.Float('learning_rate', 1e-4, 1e-2,
sampling='log')),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )
    return model

tuner = kt.RandomSearch(
    build_model,
    objective='val_accuracy',
    max_trials=2,
    executions_per_trial=1,
    directory='tuner_dir',
    project_name='cnn_tuning'
)
tuner.search(x_train, y_train, epochs=3, validation_data=(x_test, y_test))
best_model = tuner.get_best_models(num_models=1)[0]
best_model.fit(x_train, y_train, epochs=3, validation_data=(x_test, y_test))

uploaded = files.upload()

for filename in uploaded.keys():
    img = Image.open(filename)
    plt.imshow(img)
    plt.axis('off')
    plt.title("Uploaded Image")
    plt.show()
```

```
img_resized = img.resize((32, 32))
img_array = np.array(img_resized) / 255.0
img_array = np.expand_dims(img_array, axis=0)
predictions = best_model.predict(img_array)
predicted_class = np.argmax(predictions[0])
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
print(f"Predicted Class: {class_names[predicted_class]}")
```

# Output:

```
Trial 2 Complete [00h 00m 37s]
val_accuracy: 0.4803999960422516

Best val_accuracy So Far: 0.6779000163078308
Total elapsed time: 00h 01m 07s
/usr/local/lib/python3.11/dist-packages/keras/src/saving/saving_lib.py:802: UserWarning: Skipping variable loading for optimizer 'adam', because it has 2 variables whereas the saved optimizer has 18 variables.
  saveable.load_own_variables(weights_store.get(inner_path))
Epoch 1/3
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 6ms/step - accuracy: 0.6992 - loss: 0.8571 - val_accuracy: 0.6912 - val_loss: 0.8944
Epoch 2/3
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.7356 - loss: 0.7551 - val_accuracy: 0.7042 - val_loss: 0.8507
Epoch 3/3
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.7655 - loss: 0.6712 - val_accuracy: 0.7088 - val_loss: 0.8744
```

- **Find Puppies for Sale at Puppies_com.jfif**(image/jpeg) - 51824 bytes, last modified: 8/12/2025 - 100% done

```
Saving Find Puppies for Sale at Puppies_com.jfif to Find Puppies for Sale at Puppies_com.jfif
```

Uploaded Image

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 557ms/step
Predicted Class: dog
```

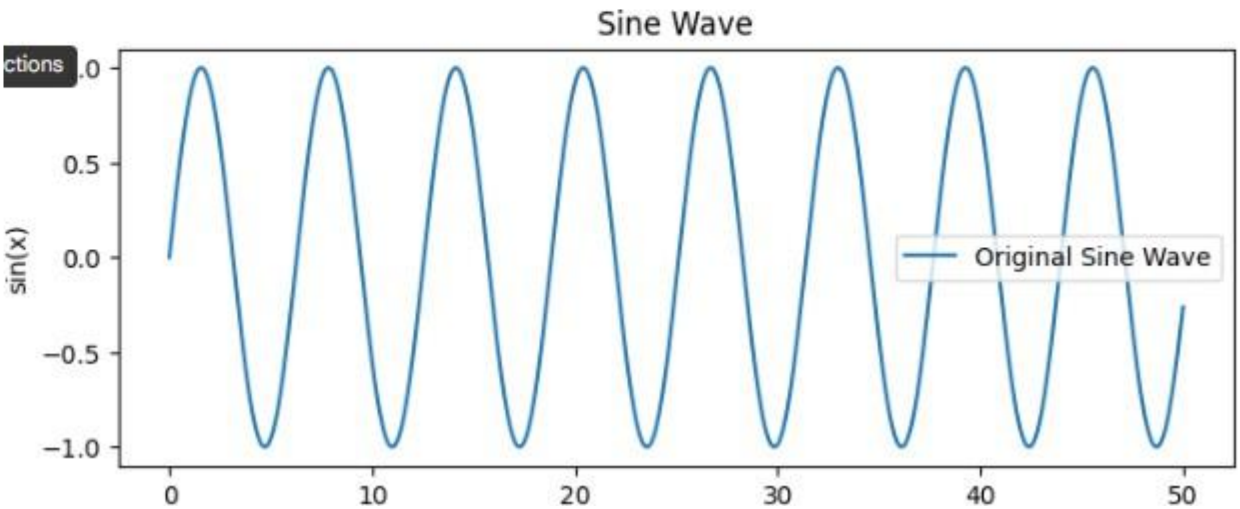
# EXPERIMENT-4

## Module name : Predicting Sequential Data

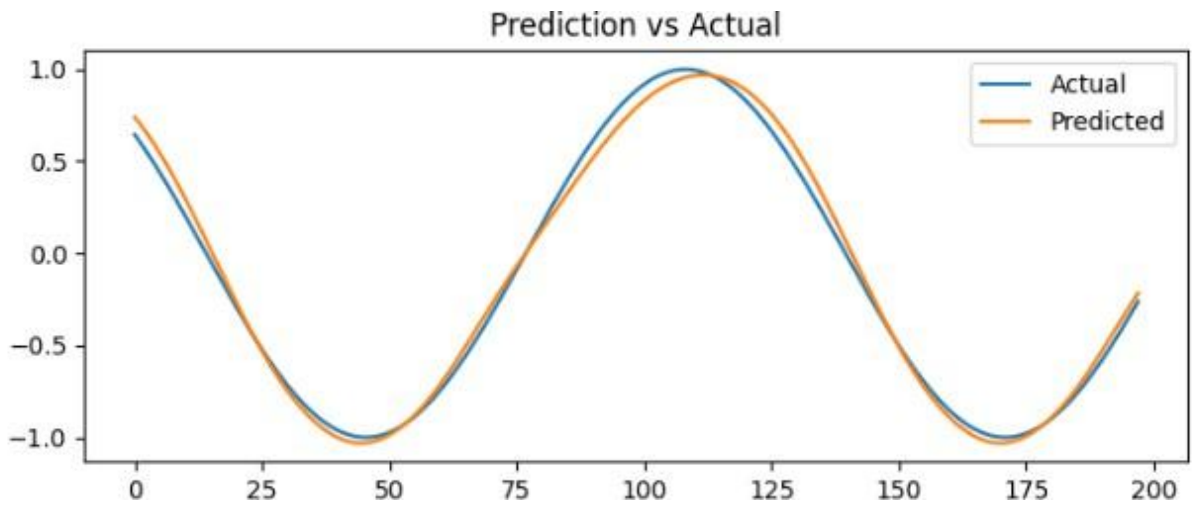
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
from sklearn.preprocessing import MinMaxScaler

sl, tp = 10, 1000
x = np.linspace(0, 50, tp)
y = np.sin(x)

plt.figure(figsize=(8, 3))
plt.plot(x, y, label="Original Sine Wave")
plt.title("Sine Wave")
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.legend()
plt.show()

sc = MinMaxScaler((0, 1))
yc = sc.fit_transform(y.reshape(-1, 1))

X = np.array([yc[i:i + sl] for i in range(tp - sl)])
y = np.array([yc[i + sl] for i in range(tp - sl)])
X = X.reshape(X.shape[0], sl, 1)

sp = int(len(X) * 0.8)
Xt, Xv, yt, yv = X[:sp], X[sp:], y[:sp], y[sp:]
```

```
m = Sequential([
    SimpleRNN(32, activation='tanh', input_shape=(sl, 1)),
    Dense(1)
])

m.compile(optimizer='adam', loss='mse')
m.fit(Xt, yt, epochs=10, batch_size=32, verbose=1)

loss = m.evaluate(Xv, yv, verbose=0)
print(f"Test loss: {loss:.4f}")

pr = sc.inverse_transform(m.predict(Xv))
ar = sc.inverse_transform(yv)

print(f"Predicted: {pr[0][0]:.4f}")
print(f"Actual: {ar[0][0]:.4f}")

plt.figure(figsize=(8, 3))
plt.plot(ar, label="Actual")
plt.plot(pr, label="Predicted")
plt.title("Prediction vs Actual")
plt.legend()
plt.show()
```

# Output:




```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:199: UserWarning: Do
not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
25/25 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - loss: 0.4952
Epoch 2/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0368
Epoch 3/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0089
Epoch 4/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0033
Epoch 5/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0017
Epoch 6/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0016
Epoch 7/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0014
Epoch 8/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0013
Epoch 9/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0011
Epoch 10/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0010
Test loss: 0.0008
```


Prediction vs Actual
Actual
Predicted
1.0
0.5
0.0
−0.5
−1.0
0
25
50
75
100
125
150
175
200

# EXPERIMENT-5

## Module Name: Removing noise from the images

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D,
UpSampling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from google.colab import files
from PIL import Image

uploaded = files.upload()

img_arr = []
original_imgs = []
for fname in uploaded.keys():
    img = Image.open(fname)
    original_imgs.append(np.array(img) / 255.0)
    img_resized = img.resize((32, 32))
    img_arr.append(np.array(img_resized) / 255.0)

img_arr = np.array(img_arr)
print("Dataset shape:", img_arr.shape)

noise_factor = 0.2
noisy_imgs = img_arr + noise_factor * np.random.normal(loc=0.0, scale=1.0,
size=img_arr.shape)
noisy_imgs = np.clip(noisy_imgs, 0., 1.)
```

```
def build_autoencoder():
    input_img = Input(shape=(32, 32, 3))
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    encoded = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(64, (3, 3), activation='relu', padding='same')(encoded)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
    x = UpSampling2D((2, 2))(x)
    decoded = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)
    model = Model(input_img, decoded)
    return model

autoencoder = build_autoencoder()
autoencoder.compile(optimizer=Adam(), loss='mse')

history = autoencoder.fit(
    noisy_imgs, img_arr,
    epochs=10,
    batch_size=1,
    verbose=1
)

pred_imgs = autoencoder.predict(noisy_imgs)

n = min(5, len(img_arr))
plt.figure(figsize=(12, 6))
for i in range(n):
    ax = plt.subplot(3, n, i+1)
    plt.imshow(original_imgs[i])
```

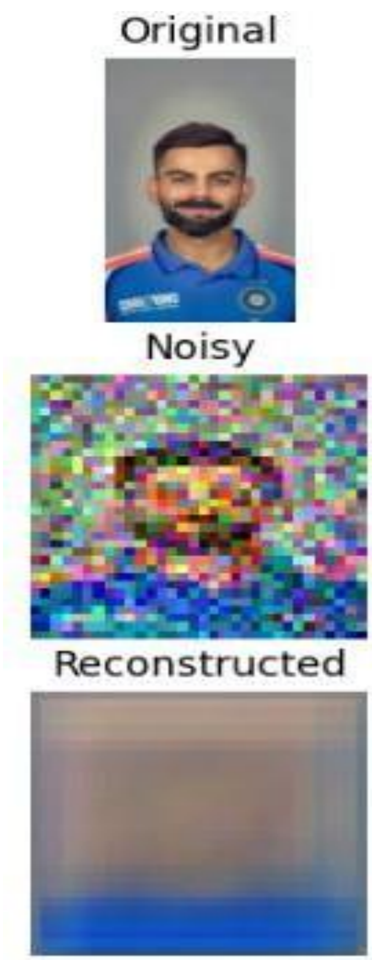
```
    ax.set_title("Original")
    plt.axis("off")

    ax = plt.subplot(3, n, i+1+n)
    plt.imshow(noisy_imgs[i])
    ax.set_title("Noisy")
    plt.axis("off")

    ax = plt.subplot(3, n, i+1+2*n)
    plt.imshow(pred_imgs[i])
    ax.set_title("Reconstructed")
    plt.axis("off")

plt.show()

plt.plot(history.history['loss'], label='Training Loss')
plt.xlabel("Epochs")
plt.ylabel("Loss (MSE)")
plt.legend()
plt.show()
```

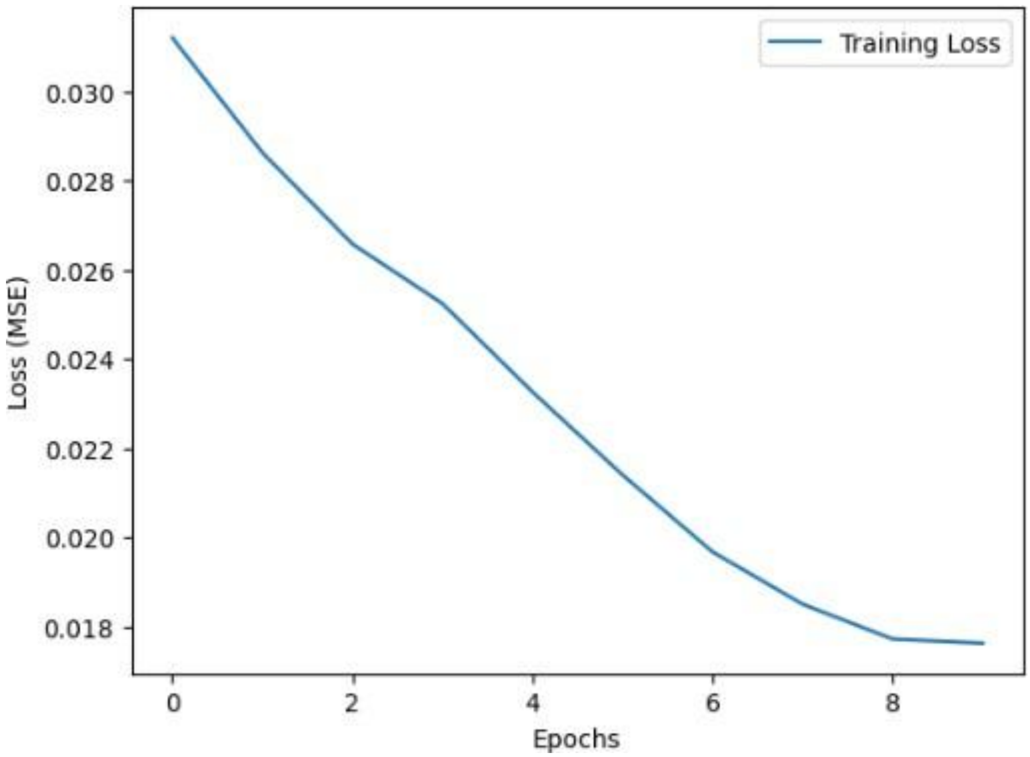
## Output:

- **vk.jpg**(image/jpeg) - 113464 bytes, last modified: 8/12/2025 - 100% done

```
Saving vk.jpg to vk (2).jpg
Dataset shape: (1, 32, 32, 3)
Epoch 1/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - loss: 0.0312
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - loss: 0.0286
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.0266
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.0252
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - loss: 0.0233
Epoch 6/10
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - loss: 0.0214
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - loss: 0.0197
Epoch 8/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 0.0185
Epoch 9/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - loss: 0.0177
Epoch 10/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - loss: 0.0176
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 643ms/step
```




Training Loss
0.030
0.028
0.026
0.024
0.022
0.020
0.018
Loss (MSE)
0
2
4
6
8
Epochs

# EXPERIMENT-6

## Module name : Advanced Deep Learning Architectures

```
from ultralytics import YOLO
from google.colab import files
from PIL import Image
from IPython.display import display

uploaded = files.upload()
filename = next(iter(uploaded))

model = YOLO('yolov8n.pt')

# run inference with lower confidence threshold
results = model.predict(filename, conf=0.25)

# show annotated image
annotated_image = results[0].plot()
display(Image.fromarray(annotated_image))

# print all detected objects
print("\nDetected Objects:")
for box in results[0].boxes:
    cls_id = int(box.cls[0].item())
    conf = float(box.conf[0].item())
    xyxy = box.xyxy[0].tolist()
    print(model.names[cls_id], f"{conf:.2f}", [round(x,2) for x in xyxy])
```

## Output:

- **baby.jpg**(image/jpeg) - 171345 bytes, last modified: 8/19/2025 - 100% done

```
Saving baby.jpg to baby (2).jpg

image 1/1 /content/baby (2).jpg: 384x640 1 person, 10.1ms
Speed: 3.5ms preprocess, 10.1ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)
```


```
Detected Objects:
person 0.89 [341.18, 25.73, 642.64, 530.77]
```

# EXPERIMENT-7

## Module Name: Optimization of Training in Deep Learning

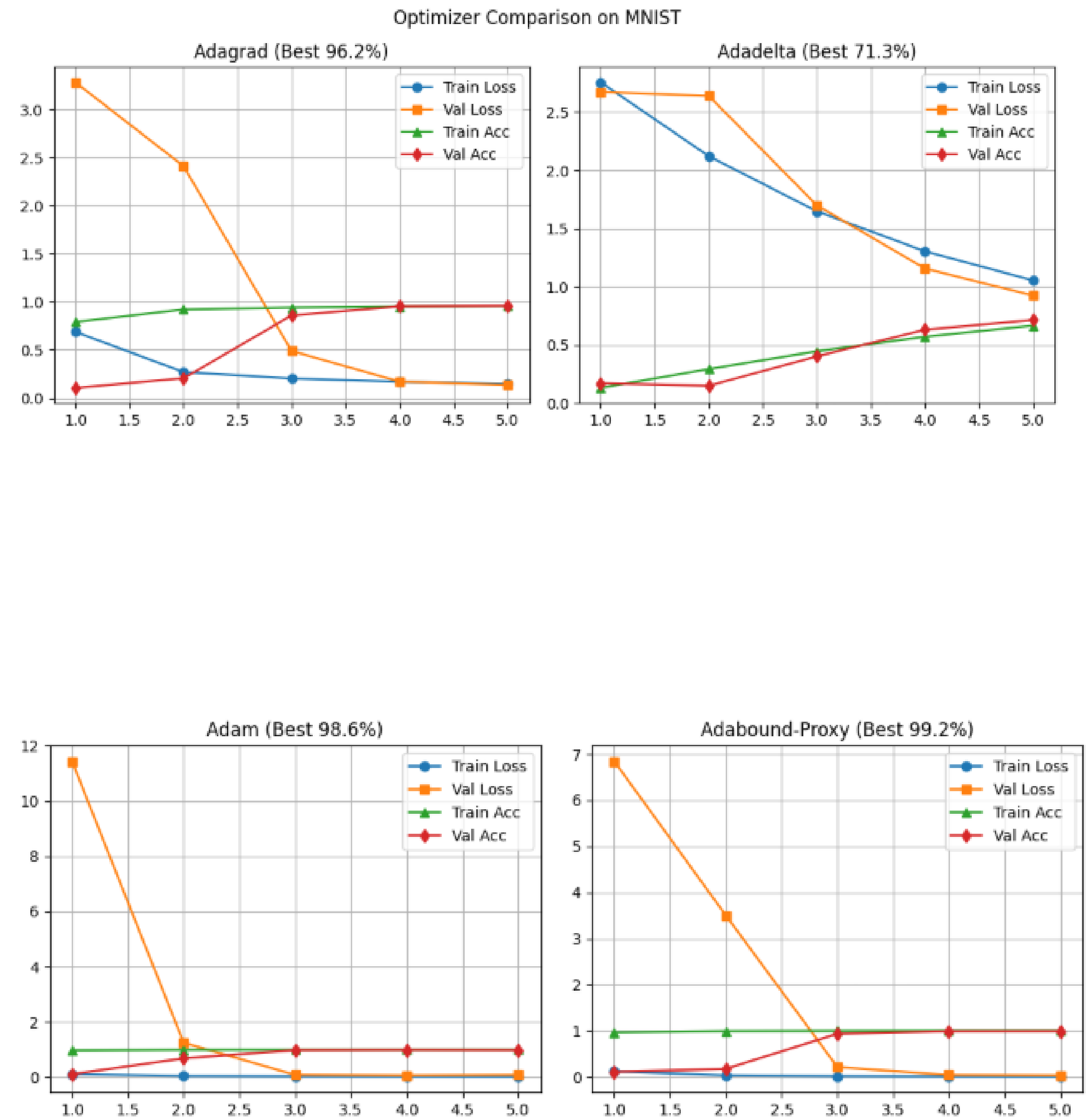
```
import tensorflow as tf, numpy as np, matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D,
MaxPooling2D, BatchNormalization


tf.random.set_seed(42); np.random.seed(42)
batch, epochs, classes, val_split, input_shape = 250, 5, 10, 0.2, (28, 28, 1)

print("=== Optimizer Comparison on MNIST ===")

 (x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train[..., None]/255.0, x_test[..., None]/255.0
print(f"Train: {x_train.shape}, Test: {x_test.shape}\n")

def create_model():
   return Sequential([
      Conv2D(32, (3,3), activation='relu', input_shape=input_shape),
BatchNormalization(),
      MaxPooling2D(2,2), BatchNormalization(),
      Conv2D(64, (3,3), activation='relu'), BatchNormalization(),
      MaxPooling2D(2,2), BatchNormalization(),
      Flatten(), Dense(256, activation='relu'), BatchNormalization(),
      Dense(classes, activation='softmax')
   ])

def train_and_eval(model, name, optimizer,
loss='sparse_categorical_crossentropy'):
   model.compile(loss=loss, optimizer=optimizer, metrics=['accuracy'])
```

```
    history = model.fit(
        x_train, y_train,
        batch_size=batch, epochs=epochs,
        verbose=0, validation_split=val_split
    )
    loss_val, acc_val = model.evaluate(x_test, y_test, verbose=0)
    print(f"{name}: Loss={loss_val:.4f}, Acc={acc_val:.4f}")
    return history


optimizers = {
    'Adagrad': tf.keras.optimizers.Adagrad(1e-3),
    'Adadelta': tf.keras.optimizers.Adadelta(1e-3),
    'Adam': tf.keras.optimizers.Adam(1e-2),
    'Adabound-Proxy': tf.keras.optimizers.Adam(1e-3)
}

histories = {name: train_and_eval(create_model(), name, opt) for name,
opt in optimizers.items()}

epochs_range = range(1, epochs+1)
fig, ax = plt.subplots(2, 2, figsize=(10, 8))
for (name, hist), axis in zip(histories.items(), ax.ravel()):
    axis.plot(epochs_range, hist.history['loss'], 'o-', label='Train Loss')
    axis.plot(epochs_range, hist.history['val_loss'], 's-', label='Val Loss')
    axis.plot(epochs_range, hist.history['accuracy'], '^-', label='Train Acc')
    axis.plot(epochs_range, hist.history['val_accuracy'], 'd-', label='Val Acc')
    axis.set_title(f'{name} (Best {max(hist.history["val_accuracy"]):.1%})')
    axis.legend(); axis.grid(True)
plt.suptitle('Optimizer Comparison on MNIST'); plt.tight_layout();
plt.show()

def bi_tempered_loss(y_true, y_pred, t=0.7, k=0.7, ls=0.1):
    y_pred = tf.clip_by_value(y_pred, 1e-7, 1-1e-7)
    pt = tf.exp(tf.math.log(y_pred)/t)
    pt /= tf.reduce_sum(pt, -1, keepdims=True)
```

```
    log_pt = tf.math.log(tf.clip_by_value(pt, 1e-7, 1-1e-7))/k
    y_onehot = tf.one_hot(tf.cast(y_true, tf.int32), classes)
    return -tf.reduce_mean(tf.reduce_sum(y_onehot*(log_pt +
ls*tf.math.log(1-y_pred)/(1-ls)), axis=-1))

hist_bi = train_and_eval(create_model(), 'Bi-Tempered (Adam)',
tf.keras.optimizers.Adam(1e-2), loss=bi_tempered_loss)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,4))
ax1.plot(epochs_range, histories['Adam'].history['val_loss'],'o-
',label='Adam')
ax1.plot(epochs_range, hist_bi.history['val_loss'],'s-',label='Bi-Temp');
ax1.legend(); ax1.grid(True)
ax2.plot(epochs_range, histories['Adam'].history['val_accuracy'],'o-
',label='Adam')
ax2.plot(epochs_range, hist_bi.history['val_accuracy'],'s-',label='Bi-Temp');
ax2.legend(); ax2.grid(True)
plt.suptitle('Bi-Tempered vs Adam'); plt.show()
```

# OUTPUT:

```
=== Optimizer Comparison on MNIST ===
Train: (60000, 28, 28, 1), Test: (10000, 28, 28, 1)

Adagrad: Loss=0.1344, Acc=0.9598
Adadelta: Loss=0.9088, Acc=0.7235
Adam: Loss=0.0817, Acc=0.9784
Adabound-Proxy: Loss=0.0291, Acc=0.9911
```




```
Bi-Tempered (Adam): Loss=0.3115, Acc=0.9888
```

# EXPERIMENT-8

## Module name: Advanced CNN

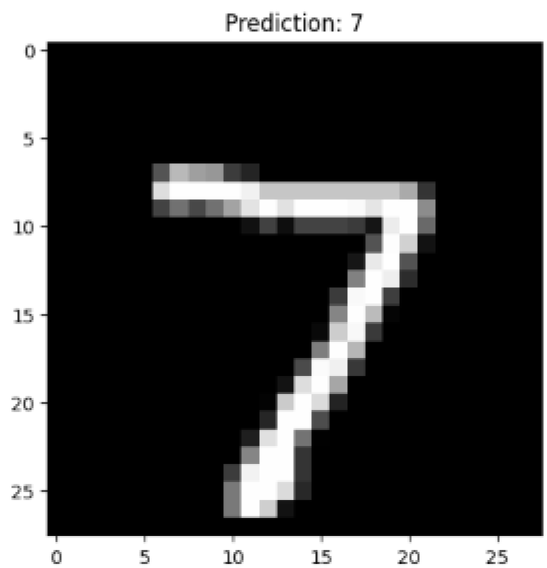
```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from keras.models import Sequential
from keras.utils import to_categorical
from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Flatten,
BatchNormalization

(xtr, ytr), (xte, yte) = mnist.load_data()

xtr = xtr.reshape(-1, 28, 28, 1).astype('float32') / 255
xte = xte.reshape(-1, 28, 28, 1).astype('float32') / 255

ytr = to_categorical(ytr, 10)
yte = to_categorical(yte, 10)

m = Sequential()
m.add(Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)))
m.add(MaxPooling2D((2,2)))
m.add(BatchNormalization())
m.add(Conv2D(64, (3,3), activation='relu'))
m.add(MaxPooling2D((2,2)))
m.add(BatchNormalization())
m.add(Conv2D(128, (3,3), activation='relu'))
m.add(BatchNormalization())
m.add(Flatten())
m.add(Dense(256, activation='relu'))
m.add(Dropout(0.4))
m.add(BatchNormalization())
m.add(Dense(128, activation='relu'))
m.add(Dropout(0.4))
```

```
m.add(BatchNormalization())
m.add(Dense(10, activation='softmax'))

m.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

h = m.fit(xtr, ytr, epochs=10, batch_size=64, validation_split=0.1,
verbose=2)

l, a = m.evaluate(xte, yte, verbose=0)
print(f"☐ Test Accuracy: {a*100:.2f}%")

p = m.predict(xte[:1])
print("Predicted Label:", np.argmax(p))
print("True Label:", np.argmax(yte[0]))

plt.imshow(xte[0].reshape(28,28), cmap="gray")
plt.title(f"Prediction: {np.argmax(p)}")
plt.show()
```

# OUTPUT:

```
Epoch 1/10
844/844 - 16s - 19ms/step - accuracy: 0.9471 - loss: 0.1826 - val_accuracy: 0.9892 - val_loss: 0.0362
Epoch 2/10
844/844 - 4s - 4ms/step - accuracy: 0.9811 - loss: 0.0648 - val_accuracy: 0.9885 - val_loss: 0.0415
Epoch 3/10
844/844 - 5s - 6ms/step - accuracy: 0.9860 - loss: 0.0473 - val_accuracy: 0.9895 - val_loss: 0.0381
Epoch 4/10
844/844 - 3s - 4ms/step - accuracy: 0.9901 - loss: 0.0349 - val_accuracy: 0.9913 - val_loss: 0.0324
Epoch 5/10
844/844 - 3s - 4ms/step - accuracy: 0.9915 - loss: 0.0287 - val_accuracy: 0.9902 - val_loss: 0.0360
Epoch 6/10
844/844 - 3s - 4ms/step - accuracy: 0.9920 - loss: 0.0252 - val_accuracy: 0.9893 - val_loss: 0.0387
Epoch 7/10
844/844 - 3s - 4ms/step - accuracy: 0.9931 - loss: 0.0220 - val_accuracy: 0.9908 - val_loss: 0.0350
Epoch 8/10
844/844 - 3s - 4ms/step - accuracy: 0.9946 - loss: 0.0181 - val_accuracy: 0.9893 - val_loss: 0.0424
Epoch 9/10
```

844/844 - 4s - 4ms/step - accuracy: 0.9955 - loss: 0.0157 - val_accuracy: 0.9910 - val_loss: 0.0376
Epoch 10/10
844/844 - 3s - 4ms/step - accuracy: 0.9960 - loss: 0.0129 - val_accuracy: 0.9923 - val_loss: 0.0352
☐ Test Accuracy: 99.18%
**1/1** ━━━━━━━━━━━━━━━━━━━━ **1s** 583ms/step
Predicted Label: 7
True Label: 7

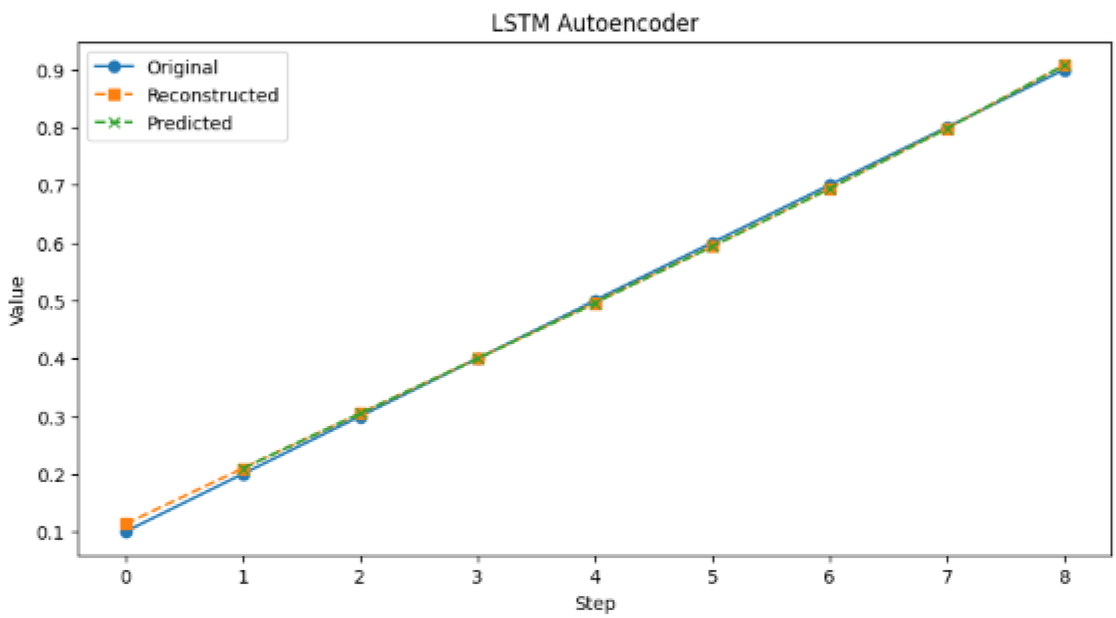


# EXPERIMENT-9

## Module name: Autoencoders Advanced

```
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import LSTM, RepeatVector, TimeDistributed, Dense

seq = np.array([0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9])
n = len(seq)
x = seq.reshape((1,n,1))

m = Sequential()
m.add(LSTM(100, activation='relu', input_shape=(n,1)))
m.add(RepeatVector(n))
m.add(LSTM(100, activation='relu', return_sequences=True))
m.add(TimeDistributed(Dense(1)))
m.compile(optimizer='adam', loss='mse')

m.fit(x,x, epochs=300, verbose=0)

r = m.predict(x, verbose=0)
p_full = m.predict(x, verbose=0)
p = p_full[0,1:,0]

plt.figure(figsize=(10,5))
plt.plot(seq, 'o-', label='Original')
plt.plot(r[0,:,0], 's--', label='Reconstructed')
plt.plot(np.arange(1,n), p, 'x--', label='Predicted')
plt.title('LSTM Autoencoder')
plt.xlabel('Step')
plt.ylabel('Value')
plt.legend()
plt.show()
```

# OUTPUT:


LSTM Autoencoder
Original
Reconstructed
Predicted
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
Value
0
1
2
3
4
5
6
7
8
Step

# EXPERIMENT-10

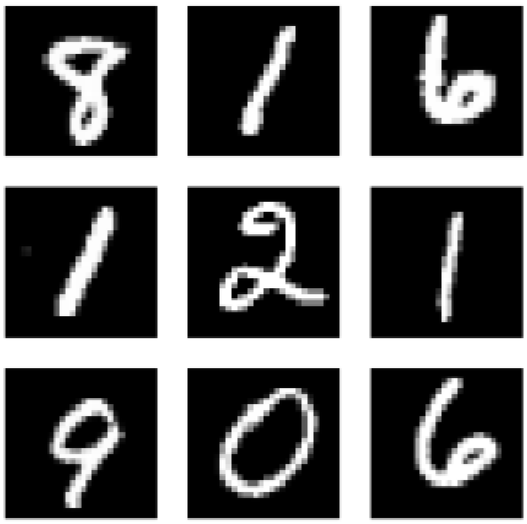
## Module name: Advanced GANs

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot as plt
import numpy as np

(X_train, y_train), (_, _) = mnist.load_data()
X_train = X_train.reshape((X_train.shape[0], 28, 28, 1)).astype('float32') /
255.0

datagen1 = ImageDataGenerator(featurewise_center=True,
featurewise_std_normalization=True)
datagen1.fit(X_train)

for X_batch, _ in datagen1.flow(X_train, y_train, batch_size=9,
shuffle=True):
    plt.figure(figsize=(5,5))
    for i in range(9):
        plt.subplot(3,3,i+1)
        plt.imshow(X_batch[i].reshape(28,28), cmap='gray')
        plt.axis('off')
    plt.suptitle("Feature Standardization")
    plt.show()
    break

datagen2 = ImageDataGenerator(zca_whitening=True)
datagen2.fit(X_train)

for X_batch, _ in datagen2.flow(X_train, y_train, batch_size=9,
shuffle=True):
    plt.figure(figsize=(5,5))
```

```
    for i in range(9):
        plt.subplot(3,3,i+1)
        plt.imshow(X_batch[i].reshape(28,28), cmap='gray')
        plt.axis('off')
    plt.suptitle("ZCA Whitening")
    plt.show()
    break

datagen3 = ImageDataGenerator(horizontal_flip=True, vertical_flip=True)

for X_batch, _ in datagen3.flow(X_train, y_train, batch_size=9,
shuffle=True):
    plt.figure(figsize=(5,5))
    for i in range(9):
        plt.subplot(3,3,i+1)
        plt.imshow(X_batch[i].reshape(28,28), cmap='gray')
        plt.axis('off')
    plt.suptitle("Random Flips")
    plt.show()
    break
```

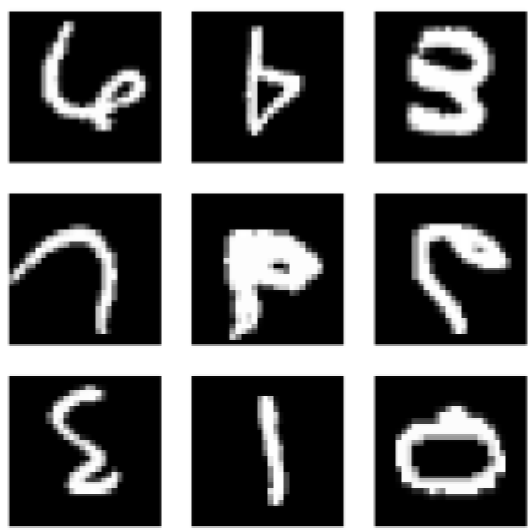
# OUPUT:

Feature Standardization



ZCA Whitening

Random Flips



# EXPERIMENT- 11

**Module Name:** Capstone Project
**Exercise:** Complete the requirements given in the capstone project.

**Object Classification for Automated CCTV**

**Problem Description:**

Nowadays, surveillance has become an essential part of any industry for safety and monitoring.
Recent developments in technology like computer vision and machine learning have brought significant advancements in various automatic surveillance systems.

Generally, CCTV cameras run continuously, consuming large amounts of memory.
One of the industries has decided to adopt **Artificial Intelligence (AI)** for automating CCTV recording.

The idea is to **customize the CCTV operation based on object detection.**
The industry plans to automate the CCTV system so that recording will start only when specific objects are recognized and categorized as belonging to predefined classes.

By using this method, continuous recording is avoided, thereby **reducing memory usage** and **improving efficiency.**

**Problem to be Solved:**

You are asked to analyze this industry requirement and develop a feasible solution that helps the company automate CCTV-based image classification.

## Instructions for Problem Solving:

As a deep learning developer, design a **best-performing model** by training a neural network with **60,000 training samples.**

- Use all test image samples to verify if the product is labeled correctly.
- You can use **TensorFlow** or **Keras** for downloading the dataset and building the model.
- Fine-tune hyperparameters and perform model evaluation.
- Substantiate your solution with insights for better visualization and provide a report on model performance.

## Data Set Description:

Initially, to test the model, you can use the benchmark dataset — **Fashion-MNIST** — before deploying it.
This dataset is a standard benchmark dataset that can be loaded directly.

## Dataset details:

- **Size of training set:** 60,000 images
- **Number of samples/class:** 6,000 images
- **Image size:** Each image is a 28×28 grayscale image. Each pixel has a single value (1–255) representing its intensity (lightness or darkness).
- **Number of classes:** 10 classes

## Training and Test Data Details:

- Each row represents a separate image.
- Column 1: Class label
- Remaining columns: Pixel numbers (total 784).
- Each pixel value ranges between **0 and 255**.
- Total columns: **785**

## Each Image Belongs to One of the Following Classes:

1. Cars
2. Birds
3. Cats

4. Deer
5. Dog
6. Frog
7. Horses
8. Planes
9. Trucks
10. Airplanes

# EXPERIMENT- 12

**Module Name:** Capstone Project

**Exercise:** Complete the requirements given in the capstone project.

```
import os
import random
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models, callbacks, utils
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

seed = 1234
os.environ['PYTHONHASHSEED'] = str(seed)
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
random.seed(seed)
np.random.seed(seed)
tf.random.set_seed(seed)

BATCH_SIZE = 64
EPOCHS = 50
IMG_SHAPE = (32, 32, 3)
NUM_CLASSES = 10
MODEL_SAVE_PATH = "cctv_cifar10_model.keras"
HISTORY_PLOT = "training_history.png"
CM_PLOT = "confusion_matrix.png"

cifar10_labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog',
'horse', 'ship', 'truck']
```

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
y_train = y_train.flatten()
y_test = y_test.flatten()

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
y_train_cat = utils.to_categorical(y_train, NUM_CLASSES)
y_test_cat = utils.to_categorical(y_test, NUM_CLASSES)

data_augment = tf.keras.Sequential([
   layers.RandomFlip("horizontal"),
   layers.RandomRotation(0.08),
   layers.RandomZoom(0.08)
])

def build_model(input_shape=IMG_SHAPE, n_classes=NUM_CLASSES):
   inputs = layers.Input(shape=input_shape)
   x = data_augment(inputs)
   x = layers.Conv2D(32, (3,3), padding='same', activation='relu')(x)
   x = layers.Conv2D(32, (3,3), padding='same', activation='relu')(x)
   x = layers.MaxPooling2D((2,2))(x)
   x = layers.BatchNormalization()(x)
   x = layers.Conv2D(64, (3,3), padding='same', activation='relu')(x)
   x = layers.Conv2D(64, (3,3), padding='same', activation='relu')(x)
   x = layers.MaxPooling2D((2,2))(x)
   x = layers.BatchNormalization()(x)
   x = layers.Conv2D(128, (3,3), padding='same', activation='relu')(x)
   x = layers.Conv2D(128, (3,3), padding='same', activation='relu')(x)
   x = layers.MaxPooling2D((2,2))(x)
   x = layers.BatchNormalization()(x)
   x = layers.GlobalAveragePooling2D()(x)
   x = layers.Dense(128, activation='relu')(x)
   x = layers.Dropout(0.4)(x)
   outputs = layers.Dense(n_classes, activation='softmax')(x)
   return models.Model(inputs=inputs, outputs=outputs, name="cctv_cnn")
```

```
model = build_model()
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
loss='categorical_crossentropy', metrics=['accuracy'])
es = callbacks.EarlyStopping(monitor='val_loss', patience=8,
restore_best_weights=True, verbose=0)
rlrop = callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=4, min_lr=1e-6, verbose=0)
mcp = callbacks.ModelCheckpoint(MODEL_SAVE_PATH,
monitor='val_accuracy', save_best_only=True, verbose=0)

history = model.fit(x_train, y_train_cat, validation_split=0.1,
epochs=EPOCHS, batch_size=BATCH_SIZE, callbacks=[es, rlrop, mcp],
shuffle=True, verbose=2)
model.save(MODEL_SAVE_PATH)

test_loss, test_acc = model.evaluate(x_test, y_test_cat, verbose=2)
print(f"Test loss: {test_loss:.4f}  Test accuracy: {test_acc:.4f}")

def plot_history(history, filename=HISTORY_PLOT):
    plt.figure(figsize=(12,4), dpi=150)
    plt.subplot(1,2,1)
    plt.plot(history.history['accuracy'], label='Train Acc')
    plt.plot(history.history['val_accuracy'], label='Val Acc')
    plt.xlabel('Epoch'); plt.ylabel('Accuracy'); plt.title('Training vs Validation
Accuracy')
    plt.legend()
    plt.subplot(1,2,2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.xlabel('Epoch'); plt.ylabel('Loss'); plt.title('Training vs Validation
Loss')
    plt.legend()
    plt.tight_layout()
    plt.savefig(filename, dpi=300, bbox_inches='tight')
    plt.show()
```

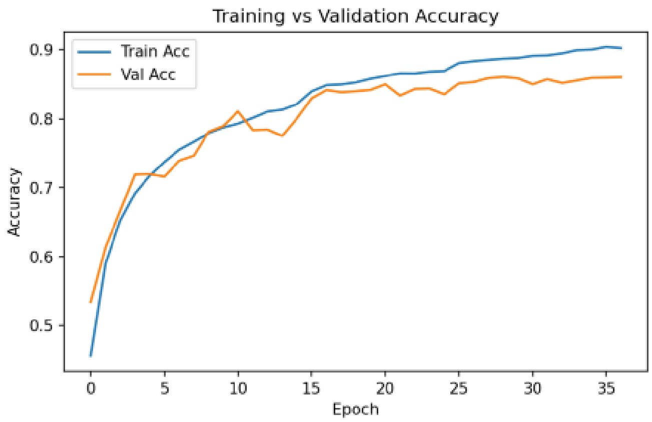
```
plot_history(history)

y_pred_prob = model.predict(x_test, verbose=0)
y_pred = np.argmax(y_pred_prob, axis=1)
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(10,8), dpi=200)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=cifar10_labels, yticklabels=cifar10_labels)
plt.xlabel('Predicted'); plt.ylabel('True'); plt.title('Confusion Matrix')
plt.tight_layout()
plt.savefig(CM_PLOT, dpi=300, bbox_inches='tight')
plt.show()

print(classification_report(y_test, y_pred, target_names=cifar10_labels))

def show_sample_predictions(x, y_true, y_pred, class_names, n=12):
   plt.figure(figsize=(12,6), dpi=200)
   indices = np.random.choice(range(len(x)), size=n, replace=False)
   for i, idx in enumerate(indices):
      plt.subplot(3, 4, i+1)
      plt.imshow((x[idx] * 255).astype('uint8'))
      plt.title(f"True: {class_names[y_true[idx]]}\nPred:
{class_names[y_pred[idx]]}", fontsize=8)
      plt.axis('off')
   plt.tight_layout()
   plt.show()

show_sample_predictions(x_test, y_test, y_pred, cifar10_labels, n=12)

def predict_image(img_array):
   img = img_array.astype('float32') / 255.0
   img = np.expand_dims(img, axis=0)
   prob = model.predict(img, verbose=0)[0]
   label = np.argmax(prob)
   return cifar10_labels[label], float(prob[label])
```

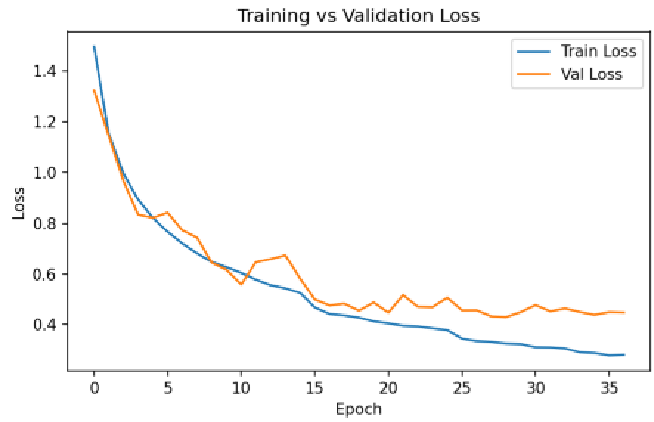
# OUTPUT:

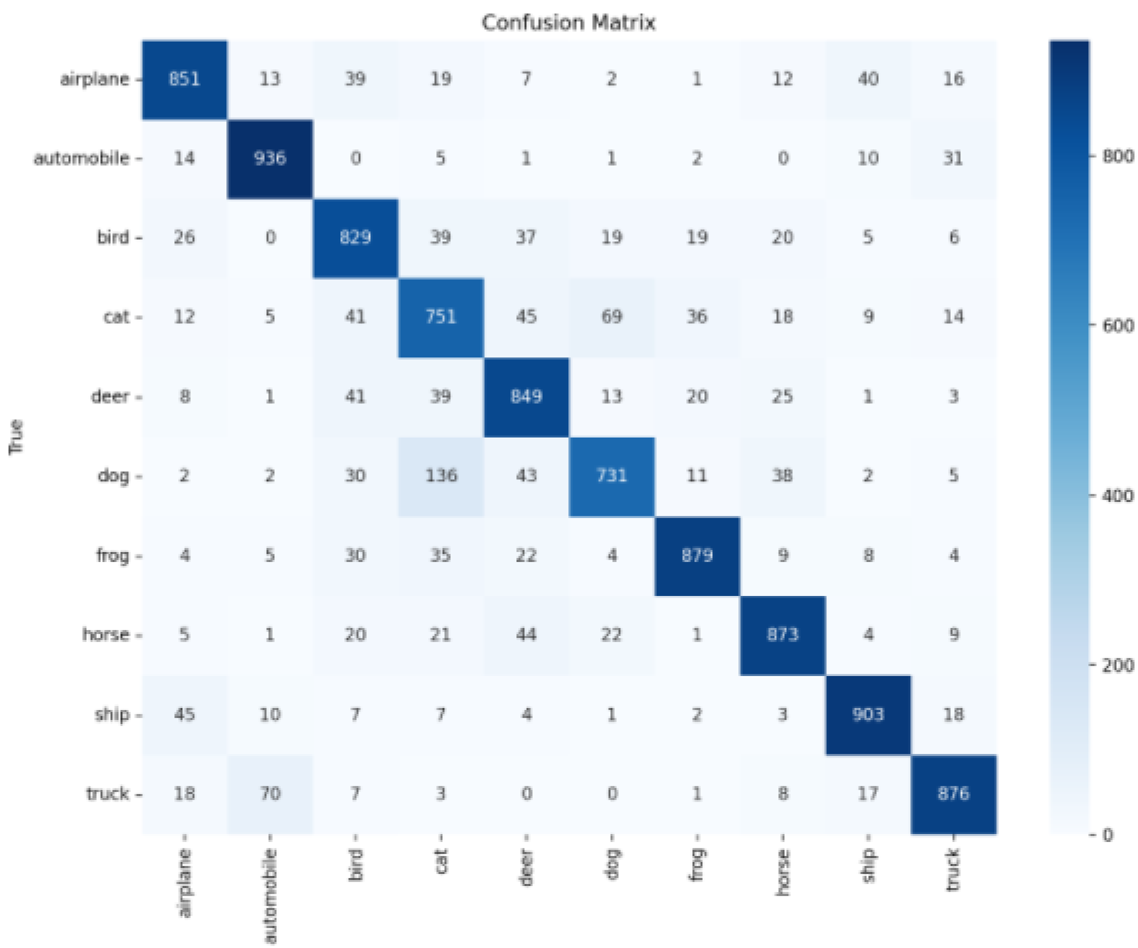
```
Epoch 1/50
704/704 - 14s - 19ms/step - accuracy: 0.4562 - loss: 1.4949 - val_accuracy: 0.5340 -
val_loss: 1.3230 - learning_rate: 1.0000e-03
Epoch 2/50
704/704 - 9s - 13ms/step - accuracy: 0.5904 - loss: 1.1528 - val_accuracy: 0.6136 -
val_loss: 1.1451 - learning_rate: 1.0000e-03
Epoch 3/50
704/704 - 9s - 13ms/step - accuracy: 0.6523 - loss: 0.9952 - val_accuracy: 0.6662 -
val_loss: 0.9653 - learning_rate: 1.0000e-03
Epoch 4/50
704/704 - 9s - 13ms/step - accuracy: 0.6919 - loss: 0.8927 - val_accuracy: 0.7192 -
val_loss: 0.8336 - learning_rate: 1.0000e-03
Epoch 5/50
704/704 - 9s - 13ms/step - accuracy: 0.7172 - loss: 0.8211 - val_accuracy: 0.7198 -
val_loss: 0.8216 - learning_rate: 1.0000e-03
Epoch 6/50
704/704 - 9s - 13ms/step - accuracy: 0.7372 - loss: 0.7665 - val_accuracy: 0.7160 -
val_loss: 0.8418 - learning_rate: 1.0000e-03
Epoch 7/50
704/704 - 10s - 14ms/step - accuracy: 0.7548 - loss: 0.7199 - val_accuracy: 0.7390 -
val_loss: 0.7721 - learning_rate: 1.0000e-03
Epoch 8/50
704/704 - 9s - 13ms/step - accuracy: 0.7665 - loss: 0.6822 - val_accuracy: 0.7464 -
val_loss: 0.7434 - learning_rate: 1.0000e-03
Epoch 9/50
704/704 - 9s - 13ms/step - accuracy: 0.7790 - loss: 0.6472 - val_accuracy: 0.7808 -
val_loss: 0.6446 - learning_rate: 1.0000e-03
Epoch 10/50
704/704 - 9s - 13ms/step - accuracy: 0.7874 - loss: 0.6270 - val_accuracy: 0.7894 -
val_loss: 0.6164 - learning_rate: 1.0000e-03
Epoch 11/50
704/704 - 9s - 13ms/step - accuracy: 0.7927 - loss: 0.6043 - val_accuracy: 0.8106 -
val_loss: 0.5592 - learning_rate: 1.0000e-03
Epoch 12/50
704/704 - 10s - 15ms/step - accuracy: 0.8012 - loss: 0.5780 - val_accuracy: 0.7834 -
val_loss: 0.6469 - learning_rate: 1.0000e-03
Epoch 13/50
704/704 - 9s - 12ms/step - accuracy: 0.8103 - loss: 0.5561 - val_accuracy: 0.7842 -
val_loss: 0.6592 - learning_rate: 1.0000e-03
Epoch 14/50
704/704 - 9s - 13ms/step - accuracy: 0.8130 - loss: 0.5443 - val_accuracy: 0.7750 -
val_loss: 0.6740 - learning_rate: 1.0000e-03
Epoch 15/50
704/704 - 9s - 13ms/step - accuracy: 0.8212 - loss: 0.5256 - val_accuracy: 0.8004 -
val_loss: 0.5823 - learning_rate: 1.0000e-03
Epoch 16/50
704/704 - 9s - 13ms/step - accuracy: 0.8397 - loss: 0.4688 - val_accuracy: 0.8296 -
val_loss: 0.5001 - learning_rate: 5.0000e-04
Epoch 17/50
704/704 - 9s - 12ms/step - accuracy: 0.8486 - loss: 0.4430 - val_accuracy: 0.8416 -
val_loss: 0.4767 - learning_rate: 5.0000e-04
```

```
Epoch 18/50
704/704 - 10s - 14ms/step - accuracy: 0.8497 - loss: 0.4375 - val_accuracy: 0.8384 -
val_loss: 0.4837 - learning_rate: 5.0000e-04
Epoch 19/50
704/704 - 9s - 13ms/step - accuracy: 0.8526 - loss: 0.4283 - val_accuracy: 0.8398 -
val_loss: 0.4561 - learning_rate: 5.0000e-04
Epoch 20/50
704/704 - 9s - 13ms/step - accuracy: 0.8579 - loss: 0.4133 - val_accuracy: 0.8418 -
val_loss: 0.4884 - learning_rate: 5.0000e-04
Epoch 21/50
704/704 - 10s - 14ms/step - accuracy: 0.8619 - loss: 0.4054 - val_accuracy: 0.8500 -
val_loss: 0.4483 - learning_rate: 5.0000e-04
Epoch 22/50
704/704 - 9s - 12ms/step - accuracy: 0.8657 - loss: 0.3957 - val_accuracy: 0.8336 -
val_loss: 0.5175 - learning_rate: 5.0000e-04
Epoch 23/50
704/704 - 9s - 13ms/step - accuracy: 0.8656 - loss: 0.3932 - val_accuracy: 0.8432 -
val_loss: 0.4713 - learning_rate: 5.0000e-04
Epoch 24/50
704/704 - 9s - 13ms/step - accuracy: 0.8681 - loss: 0.3860 - val_accuracy: 0.8438 -
val_loss: 0.4697 - learning_rate: 5.0000e-04
Epoch 25/50
704/704 - 10s - 14ms/step - accuracy: 0.8691 - loss: 0.3789 - val_accuracy: 0.8354 -
val_loss: 0.5068 - learning_rate: 5.0000e-04
Epoch 26/50
704/704 - 9s - 13ms/step - accuracy: 0.8807 - loss: 0.3456 - val_accuracy: 0.8512 -
val_loss: 0.4571 - learning_rate: 2.5000e-04
Epoch 27/50
704/704 - 9s - 13ms/step - accuracy: 0.8834 - loss: 0.3361 - val_accuracy: 0.8532 -
val_loss: 0.4573 - learning_rate: 2.5000e-04
Epoch 28/50
704/704 - 9s - 13ms/step - accuracy: 0.8852 - loss: 0.3328 - val_accuracy: 0.8588 -
val_loss: 0.4332 - learning_rate: 2.5000e-04
Epoch 29/50
704/704 - 9s - 13ms/step - accuracy: 0.8869 - loss: 0.3265 - val_accuracy: 0.8606 -
val_loss: 0.4307 - learning_rate: 2.5000e-04
Epoch 30/50
704/704 - 9s - 13ms/step - accuracy: 0.8879 - loss: 0.3242 - val_accuracy: 0.8586 -
val_loss: 0.4506 - learning_rate: 2.5000e-04
Epoch 31/50
704/704 - 9s - 13ms/step - accuracy: 0.8911 - loss: 0.3120 - val_accuracy: 0.8500 -
val_loss: 0.4779 - learning_rate: 2.5000e-04
Epoch 32/50
704/704 - 9s - 13ms/step - accuracy: 0.8916 - loss: 0.3112 - val_accuracy: 0.8572 -
val_loss: 0.4533 - learning_rate: 2.5000e-04
Epoch 33/50
704/704 - 9s - 13ms/step - accuracy: 0.8946 - loss: 0.3070 - val_accuracy: 0.8518 -
val_loss: 0.4647 - learning_rate: 2.5000e-04
Epoch 34/50
704/704 - 9s - 13ms/step - accuracy: 0.8991 - loss: 0.2915 - val_accuracy: 0.8554 -
val_loss: 0.4509 - learning_rate: 1.2500e-04
Epoch 35/50
704/704 - 9s - 13ms/step - accuracy: 0.9000 - loss: 0.2884 - val_accuracy: 0.8592 -
val_loss: 0.4396 - learning_rate: 1.2500e-04
Epoch 36/50
```

```
704/704 - 9s - 13ms/step - accuracy: 0.9039 - loss: 0.2790 - val_accuracy: 0.8596 -
val_loss: 0.4503 - learning_rate: 1.2500e-04
Epoch 37/50
704/704 - 9s - 13ms/step - accuracy: 0.9022 - loss: 0.2812 - val_accuracy: 0.8602 -
val_loss: 0.4489 - learning_rate: 1.2500e-04
313/313 - 1s - 4ms/step - accuracy: 0.8478 - loss: 0.4654
Test loss: 0.4654  Test accuracy: 0.8478
```




```
precision    recall  f1-score   support
```

| | | | | |
|---|---|---|---|---|
| airplane | 0.86 | 0.85 | 0.86 | 1000 |
| automobile | 0.90 | 0.94 | 0.92 | 1000 |
| bird | 0.79 | 0.83 | 0.81 | 1000 |
| cat | 0.71 | 0.75 | 0.73 | 1000 |
| deer | 0.81 | 0.85 | 0.83 | 1000 |
| dog | 0.85 | 0.73 | 0.79 | 1000 |
| frog | 0.90 | 0.88 | 0.89 | 1000 |
| horse | 0.87 | 0.87 | 0.87 | 1000 |
| ship | 0.90 | 0.90 | 0.90 | 1000 |
| truck | 0.89 | 0.88 | 0.88 | 1000 |
| accuracy | | | 0.85 | 10000 |
| macro avg | 0.85 | 0.85 | 0.85 | 10000 |
| weighted avg | 0.85 | 0.85 | 0.85 | 10000 |