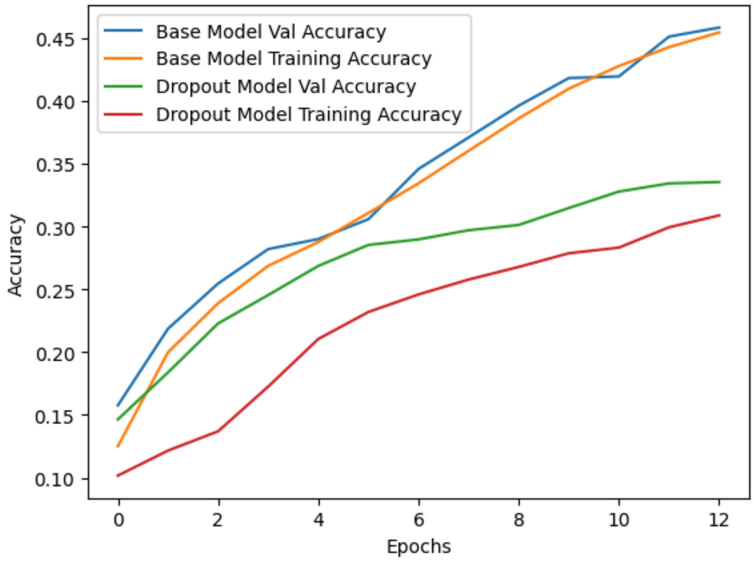
```
print("DLVS LAB")
print("exp 5")
```

```
DLVS LAB
exp 5
```

```
import tensorflow as tf
from tensorflow.keras import layers, models, datasets
import matplotlib.pyplot as plt

# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = datasets.cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  # Normalize pixel values

# One-hot encode the labels
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Build the base CNN model
def build_model(dropout_rate=0.0):
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='leaky_relu', input_shape=(32, 32, 3)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='tanh'),
        layers.Flatten(),
        layers.Dense(64, activation='sigmoid'),
        layers.Dropout(dropout_rate),  # Apply dropout
        layers.Dense(10, activation='softmax')
    ])
    return model

# Compile and train the model
def train_and_evaluate_model(model):
    model.compile(optimizer='SGD',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    history = model.fit(x_train, y_train, epochs=13,
                        validation_split=0.2, batch_size=64)
    return history

# Initial model without dropout
model_base = build_model()
history_base = train_and_evaluate_model(model_base)

# Model with dropout to prevent overfitting
model_dropout = build_model(dropout_rate=0.5)
history_dropout = train_and_evaluate_model(model_dropout)

# Plot training history
def plot_history(histories, labels):
    for history, label in zip(histories, labels):
        plt.plot(history.history['val_accuracy'], label=f'{label} Val Accuracy')
        plt.plot(history.history['accuracy'], label=f'{label} Training Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()

plot_history([history_base, history_dropout], ['Base Model', 'Dropout Model'])

# Evaluate the models on the test set
print("Base Model Performance:")
base_eval = model_base.evaluate(x_test, y_test)
print("Dropout Model Performance:")
dropout_eval = model_dropout.evaluate(x_test, y_test)
```

```
Epoch 1/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.1115 - loss: 2.3200 - val_accuracy: 0.1577 - val_loss: 2.2778
Epoch 2/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1880 - loss: 2.2612 - val_accuracy: 0.2187 - val_loss: 2.1462
Epoch 3/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2300 - loss: 2.0932 - val_accuracy: 0.2547 - val_loss: 2.0185
Epoch 4/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2640 - loss: 2.0035 - val_accuracy: 0.2821 - val_loss: 1.9667
Epoch 5/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2804 - loss: 1.9639 - val_accuracy: 0.2901 - val_loss: 1.9349
Epoch 6/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3070 - loss: 1.9132 - val_accuracy: 0.3059 - val_loss: 1.9253
Epoch 7/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3282 - loss: 1.8683 - val_accuracy: 0.3459 - val_loss: 1.8179
Epoch 8/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3517 - loss: 1.8098 - val_accuracy: 0.3710 - val_loss: 1.7524
Epoch 9/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3786 - loss: 1.7400 - val_accuracy: 0.3963 - val_loss: 1.6926
Epoch 10/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4034 - loss: 1.6772 - val_accuracy: 0.4183 - val_loss: 1.6141
Epoch 11/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4253 - loss: 1.6073 - val_accuracy: 0.4196 - val_loss: 1.6136
Epoch 12/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4421 - loss: 1.5596 - val_accuracy: 0.4513 - val_loss: 1.5233
Epoch 13/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4558 - loss: 1.5212 - val_accuracy: 0.4585 - val_loss: 1.4980
Epoch 1/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.0985 - loss: 2.4494 - val_accuracy: 0.1465 - val_loss: 2.2923
Epoch 2/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.1207 - loss: 2.2972 - val_accuracy: 0.1839 - val_loss: 2.2790
Epoch 3/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1269 - loss: 2.2814 - val_accuracy: 0.2229 - val_loss: 2.2287
Epoch 4/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.1631 - loss: 2.2228 - val_accuracy: 0.2455 - val_loss: 2.0963
Epoch 5/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2057 - loss: 2.1168 - val_accuracy: 0.2687 - val_loss: 2.0256
Epoch 6/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2300 - loss: 2.0594 - val_accuracy: 0.2854 - val_loss: 1.9854
Epoch 7/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2463 - loss: 2.0250 - val_accuracy: 0.2898 - val_loss: 1.9576
Epoch 8/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2540 - loss: 2.0020 - val_accuracy: 0.2971 - val_loss: 1.9269
Epoch 9/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2647 - loss: 1.9802 - val_accuracy: 0.3013 - val_loss: 1.9173
Epoch 10/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2754 - loss: 1.9508 - val_accuracy: 0.3148 - val_loss: 1.8872
Epoch 11/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2781 - loss: 1.9333 - val_accuracy: 0.3279 - val_loss: 1.8437
Epoch 12/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.2982 - loss: 1.8992 - val_accuracy: 0.3344 - val_loss: 1.8138
Epoch 13/13
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3062 - loss: 1.8689 - val_accuracy: 0.3355 - val_loss: 1.8286
```




```
Base Model Performance:
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.4692 - loss: 1.4872
Dropout Model Performance:
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.3482 - loss: 1.8264
```

```
print(history_base.history.keys())
print(history_dropout.history.keys())
```

```
dict_keys(['accuracy', 'loss'])
dict_keys(['accuracy', 'loss'])
```

```
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model


import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define paths
base_dir = '/kaggle/input/chest-xray-pneumonia/chest_xray'
train_dir = os.path.join(base_dir, 'train')
val_dir = os.path.join(base_dir, 'val')
test_dir = os.path.join(base_dir, 'test')


train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=False
)


val_test_datagen = ImageDataGenerator(rescale=1.0/255.0)


train_gen = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=64,
    class_mode='binary',
    shuffle=True
)

val_gen = val_test_datagen.flow_from_directory(
    val_dir,
    target_size=(224, 224),
    batch_size=64,
    class_mode='binary'
)

test_gen = val_test_datagen.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    batch_size=64,
    class_mode='binary'
)
```

```
Found 5216 images belonging to 2 classes.
Found 16 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

```
images, labels = next(train_gen)
print("Image batch shape:", images.shape)
print("Label batch shape:", labels.shape)
```

```
Image batch shape: (64, 224, 224, 3)
Label batch shape: (64,)
```

```
def create_resnet18_fine_tune(input_shape=(224, 224, 3), num_classes=2):
    base_model = tf.keras.applications.ResNet50(
        weights='imagenet',
        include_top=False,
        input_shape=input_shape
    )

    base_model.trainable = True

    x = layers.GlobalAveragePooling2D()(base_model.output)
    x = layers.Dense(512, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(256, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=outputs)
    return model

model = create_resnet18_fine_tune(input_shape=(224, 224, 3), num_classes=2)
```

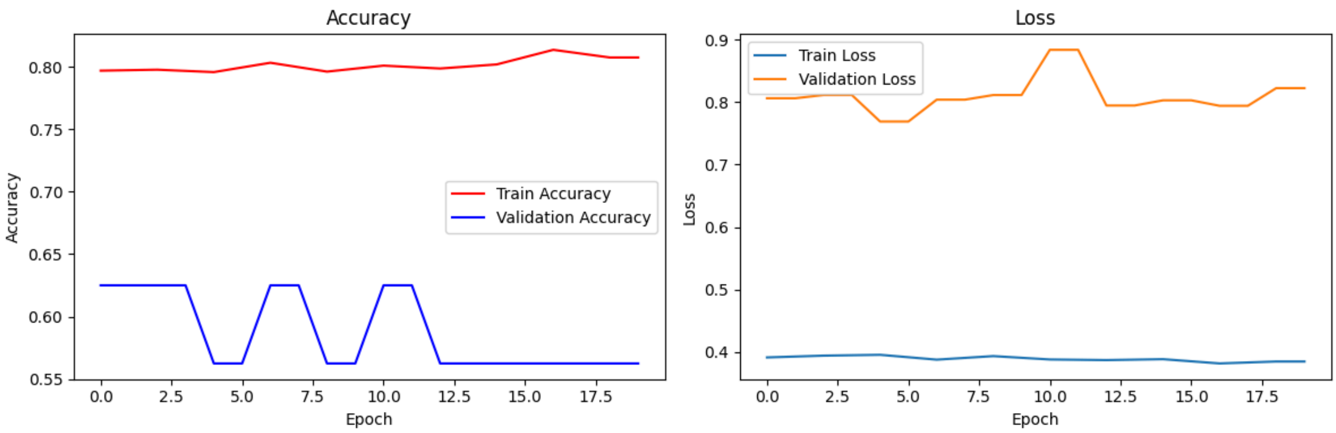
```
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    train_gen,
    validation_data=val_gen,
    epochs=10,
    steps_per_epoch=len(train_gen),
    validation_steps=len(val_gen),
    verbose=1
)
```

```
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 44s 51ms/step - accuracy: 0.0970 - loss: 2.3038 - val_accuracy: 0.0994 - val_loss: 2.3027
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 29s 47ms/step - accuracy: 0.0990 - loss: 2.3028 - val_accuracy: 0.1023 - val_loss: 2.3028
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 30s 47ms/step - accuracy: 0.0986 - loss: 2.3027 - val_accuracy: 0.0933 - val_loss: 2.3027
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 30s 47ms/step - accuracy: 0.1002 - loss: 2.3026 - val_accuracy: 0.0973 - val_loss: 2.3027
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 31s 49ms/step - accuracy: 0.0972 - loss: 2.3027 - val_accuracy: 0.0933 - val_loss: 2.3030
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 31s 48ms/step - accuracy: 0.0988 - loss: 2.3027 - val_accuracy: 0.0933 - val_loss: 2.3030
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 31s 48ms/step - accuracy: 0.0997 - loss: 2.3027 - val_accuracy: 0.0933 - val_loss: 2.3030
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 31s 49ms/step - accuracy: 0.1007 - loss: 2.3026 - val_accuracy: 0.0933 - val_loss: 2.3029
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 30s 48ms/step - accuracy: 0.0986 - loss: 2.3027 - val_accuracy: 0.0933 - val_loss: 2.3029
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 30s 48ms/step - accuracy: 0.1001 - loss: 2.3027 - val_accuracy: 0.0933 - val_loss: 2.3028
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.0987 - loss: 2.3027
Test Accuracy: 0.10
```

```
test_loss, test_accuracy = model.evaluate(test_gen, verbose=1)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
10/10 ━━━━━━━━━━━━━━━━━━━━ 5s 493ms/step - accuracy: 0.6046 - loss: 0.7142
Test Accuracy: 62.50%
```

```
import matplotlib.pyplot as plt

# Plot accuracy
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(X, label='Train Accuracy',color='red')
plt.plot(Y, label='Validation Accuracy',color='blue')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss
plt.subplot(1, 2, 2)
plt.plot(x, label='Train Loss')
plt.plot(y, label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```




```
## exp 6 practice
!pip install tensorflow
```

```
/opt/conda/lib/python3.10/pty.py:89: RuntimeWarning: os.fork() was called. os.fork() is incompatible with multithreaded code, and JA
  pid, fd = os.forkpty()
Requirement already satisfied: tensorflow in /opt/conda/lib/python3.10/site-packages (2.16.1)
Requirement already satisfied: absl-py>=1.0.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (0.5
Requirement already satisfied: google-pasta>=0.1.1 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (3.11.0)
Requirement already satisfied: libclang>=13.0.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes~=0.3.1 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (0.3.2)
Requirement already satisfied: opt-einsum>=2.3.2 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /opt/conda/lib/python3.10/site-packages (from tensorflow) (21.3)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /opt/conda/lib/py
Requirement already satisfied: requests<3,>=2.21.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.10/site-packages (from tensorflow) (70.0.0)
Requirement already satisfied: six>=1.12.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (1.16.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (1.62.2)
Requirement already satisfied: tensorboard<2.17,>=2.16 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (2.16.2)
Requirement already satisfied: keras>=3.0.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (3.3.3)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (0
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (1.26.4)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /opt/conda/lib/python3.10/site-packages (from astunparse>=1.6.0->tensorflow) (0
Requirement already satisfied: rich in /opt/conda/lib/python3.10/site-packages (from keras>=3.0.0->tensorflow) (13.7.1)
Requirement already satisfied: namex in /opt/conda/lib/python3.10/site-packages (from keras>=3.0.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /opt/conda/lib/python3.10/site-packages (from keras>=3.0.0->tensorflow) (0.11.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/conda/lib/python3.10/site-packages (from requests<3,>=2.21.0->tensor
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-packages (from requests<3,>=2.21.0->tensorflow) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/conda/lib/python3.10/site-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.10/site-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: markdown>=2.6.8 in /opt/conda/lib/python3.10/site-packages (from tensorboard<2.17,>=2.16->tensorflow
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /opt/conda/lib/python3.10/site-packages (from tensorboard<2
Requirement already satisfied: werkzeug>=1.0.1 in /opt/conda/lib/python3.10/site-packages (from tensorboard<2.17,>=2.16->tensorflow
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /opt/conda/lib/python3.10/site-packages (from packaging->tensorflow) (3.1
Requirement already satisfied: MarkupSafe>=2.1.1 in /opt/conda/lib/python3.10/site-packages (from werkzeug>=1.0.1->tensorboard<2.17,
Requirement already satisfied: markdown-it-py>=2.2.0 in /opt/conda/lib/python3.10/site-packages (from rich->keras>=3.0.0->tensorflow
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /opt/conda/lib/python3.10/site-packages (from rich->keras>=3.0.0->tensorfl
Requirement already satisfied: mdurl~=0.1 in /opt/conda/lib/python3.10/site-packages (from markdown-it-py>=2.2.0->rich->keras>=3.0.0
```

```
import tensorflow as tf
from tensorflow.keras import layers,models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt


img_size=(64,64)
batch_size=32


datagen=ImageDataGenerator(
    rescale=1.0/255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    validation_split=0.2,
```

```
    horizontal_flip=True
)
```

```
train_gen=datagen.flow_from_directory(
    '/kaggle/input/satellite-images-of-hurricane-damage',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)
val_gen=datagen.flow_from_directory(
    '/kaggle/input/satellite-images-of-hurricane-damage',
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)
```

```
Found 18400 images belonging to 4 classes.
Found 4600 images belonging to 4 classes.
```

```
## AlexNet model

model=models.Sequential([
    layers.Conv2D(32,(5,5),activation="relu",input_shape=(64,64,3)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,(5,5),activation="relu"),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(120,activation="relu"),
    layers.Dense(84,activation="relu"),
    layers.Dense(len(train_gen.class_indices),activation="softmax")
])
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy"])
```

```
history=model.fit(train_gen,validation_data=val_gen,epochs=8)
```

```
Epoch 1/8
/opt/conda/lib/python3.10/site-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` cl
  self._warn_if_super_not_called()
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1735312235.449208     125 service.cc:145] XLA service 0x7ab068005fc0 initialized for platform CUDA (this does not guarar
I0000 00:00:1735312235.449264     125 service.cc:153]   StreamExecutor device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1735312235.449271     125 service.cc:153]   StreamExecutor device (1): Tesla T4, Compute Capability 7.5
  3/575 ━━━━━━━━━━━━━━━━━━━━ 35s 61ms/step - accuracy: 0.4253 - loss: 1.3630I0000 00:00:1735312239.095749     125 device_compiler.h
575/575 ━━━━━━━━━━━━━━━━━━━━ 107s 176ms/step - accuracy: 0.4205 - loss: 1.1798 - val_accuracy: 0.4348 - val_loss: 1.1549
Epoch 2/8
575/575 ━━━━━━━━━━━━━━━━━━━━ 53s 91ms/step - accuracy: 0.4247 - loss: 1.1612 - val_accuracy: 0.4348 - val_loss: 1.1540
Epoch 3/8
575/575 ━━━━━━━━━━━━━━━━━━━━ 52s 90ms/step - accuracy: 0.4354 - loss: 1.1592 - val_accuracy: 0.4348 - val_loss: 1.1546
Epoch 4/8
575/575 ━━━━━━━━━━━━━━━━━━━━ 51s 88ms/step - accuracy: 0.4233 - loss: 1.1549 - val_accuracy: 0.4348 - val_loss: 1.1541
Epoch 5/8
575/575 ━━━━━━━━━━━━━━━━━━━━ 51s 88ms/step - accuracy: 0.4362 - loss: 1.1519 - val_accuracy: 0.4348 - val_loss: 1.1554
Epoch 6/8
575/575 ━━━━━━━━━━━━━━━━━━━━ 51s 88ms/step - accuracy: 0.4300 - loss: 1.1591 - val_accuracy: 0.4348 - val_loss: 1.1553
Epoch 7/8
575/575 ━━━━━━━━━━━━━━━━━━━━ 50s 86ms/step - accuracy: 0.4330 - loss: 1.1529 - val_accuracy: 0.4348 - val_loss: 1.1556
Epoch 8/8
575/575 ━━━━━━━━━━━━━━━━━━━━ 50s 86ms/step - accuracy: 0.4321 - loss: 1.1568 - val_accuracy: 0.4348 - val_loss: 1.1546
```

```
for i in range(8):
    print("accuracy:- ",history.history["accuracy"][i]," val_accuracy:- ",history.history["val_accuracy"][i])
```

```
accuracy:-  0.4238043427467346  val_accuracy:-  0.43478259444236755
accuracy:-  0.4321739077568054  val_accuracy:-  0.43478259444236755
accuracy:-  0.4288586974143982  val_accuracy:-  0.43478259444236755
accuracy:-  0.43173912167549133  val_accuracy:-  0.43478259444236755
accuracy:-  0.43141305446624756  val_accuracy:-  0.43478259444236755
accuracy:-  0.4338586926460266  val_accuracy:-  0.43478259444236755
accuracy:-  0.43478259444236755  val_accuracy:-  0.43478259444236755
accuracy:-  0.43478259444236755  val_accuracy:-  0.43478259444236755
```

```
# RCNN exp 9
```

```
import torch
from torchvision.models.detection import fasterrcnn_resnet50_fpn
from torchvision.transforms import functional as F
from PIL import Image
import matplotlib.pyplot as plt
import torchvision.transforms as T
import cv2
import numpy as np
```

```
model = fasterrcnn_resnet50_fpn(pretrained=True)
```

```
model.eval()
```

```
FasterRCNN(
  (transform): GeneralizedRCNNTransform(
      Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      Resize(min_size=(800,), max_size=1333, mode='bilinear')
  )
  (backbone): BackboneWithFPN(
    (body): IntermediateLayerGetter(
      (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
      (bn1): FrozenBatchNorm2d(64, eps=0.0)
      (relu): ReLU(inplace=True)
      (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
      (layer1): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(64, eps=0.0)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(64, eps=0.0)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(256, eps=0.0)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): FrozenBatchNorm2d(256, eps=0.0)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(64, eps=0.0)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(64, eps=0.0)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(256, eps=0.0)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(64, eps=0.0)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(64, eps=0.0)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(256, eps=0.0)
          (relu): ReLU(inplace=True)
        )
      )
      (layer2): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(128, eps=0.0)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(128, eps=0.0)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(512, eps=0.0)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
            (1): FrozenBatchNorm2d(512, eps=0.0)
          )
        )
```

```
COCO_CLASSES = [
"__background__", "person", "bicycle", "car", "motorcycle", "airplane", "bus",
"train", "truck", "boat", "traffic light", "fire hydrant", "N/A", "stop sign",
"parking meter", "bench", "bird", "cat", "dog", "horse", "sheep", "cow", "elephant",
"bear", "zebra", "giraffe", "N/A", "backpack", "umbrella", "N/A", "N/A", "handbag",
"tie", "suitcase", "frisbee", "skis", "snowboard", "sports ball", "kite", "baseball bat",
"baseball glove", "skateboard", "surfboard", "tennis racket", "bottle", "N/A", "wine glass",
"cup", "fork", "knife", "spoon", "bowl", "banana", "apple", "sandwich", "orange", "broccoli",
"carrot", "hot dog", "pizza", "donut", "cake", "chair", "couch", "potted plant", "bed", "N/A",
"dining table", "N/A", "N/A", "toilet", "N/A", "TV", "laptop", "mouse", "remote", "keyboard",
"cell phone", "microwave", "oven", "toaster", "sink", "refrigerator", "N/A", "book", "clock",
"vase", "scissors", "teddy bear", "hair drier", "toothbrush"
]
```

```
image=Image.open("/kaggle/input/dog-image/WhatsApp Image 2024-12-13 at 16.56.36.jpeg").convert("RGB")

transform=T.Compose([
    T.ToTensor()
])

transformed_img=transform(image)

with torch.no_grad():
    predictions=model([transformed_img])

image_np=np.array(image)
confidence_threshold=0.5

boxes=predictions[0]['boxes']
labels=predictions[0]['labels']
scores=predictions[0]['scores']

print(boxes)
```

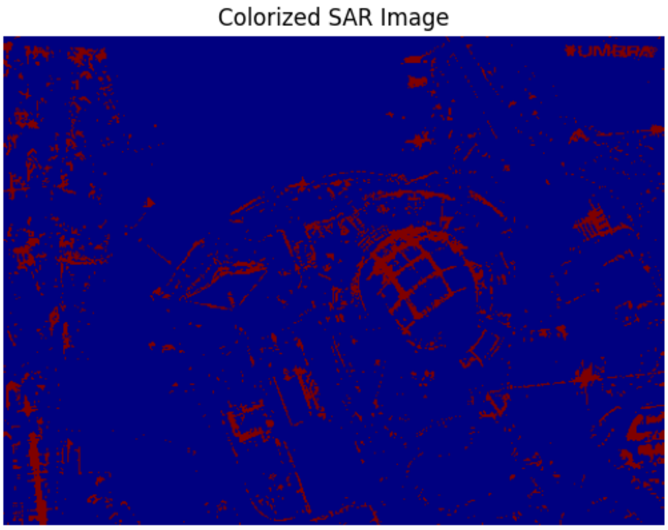
```
tensor([[1.5768e+02, 2.8703e+02, 4.5723e+02, 6.8609e+02],
        [4.5749e+02, 1.9869e+00, 4.9859e+02, 2.0636e+01],
        [1.6059e+02, 3.1262e+02, 4.7750e+02, 6.8124e+02],
        [2.9099e+02, 2.8718e+02, 4.5319e+02, 6.4728e+02],
        [3.1947e-01, 2.9270e+02, 1.1975e+02, 5.1460e+02]])
```

```
for i,box in enumerate(boxes):
    if scores[i]>confidence_threshold:
        x1, y1, x2, y2 = map(int, box)
        label_index = labels[i].item()
        label_name = COCO_CLASSES[label_index] # Get the label name from COCO classes
        score = scores[i].item() # Extract the confidence score
        # Draw a rectangle around the detected object
        cv2.rectangle(image_np, (x1, y1), (x2, y2), (0, 255, 0), 2)
        # Add a label with the class name and confidence score
        text = f"{label_name}: {score:.2f}"
        cv2.putText(image_np, text, (x1, y1 + 10),
        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)

plt.figure(figsize=(23,20))
plt.imshow(image_np)
plt.axis('off')
plt.title('Object Detection results')
```

Text(0.5, 1.0, 'Object Detection results')

SAR Image

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

def colorize_sar_image(sar_image_path, output_image_path):
    # Load the input JPEG image (grayscale)
    sar_image = cv2.imread(sar_image_path, cv2.IMREAD_GRAYSCALE)

    # Normalize the SAR image to the range [0, 1]
    sar_image_normalized = cv2.normalize(sar_image, None, 0, 1, cv2.NORM_MINMAX)

    # Apply a colormap for colorization (e.g., Jet colormap)
    colorized_image = cv2.applyColorMap((sar_image_normalized * 255).astype(np.uint8), cv2.COLORMAP_JET)

    # Save and display the colorized image
    cv2.imwrite(output_image_path, colorized_image)

    plt.imshow(cv2.cvtColor(colorized_image, cv2.COLOR_BGR2RGB))
    plt.title("Colorized SAR Image")
    plt.axis("off")
    plt.show()

# Example usage:
sar_image_path = '/kaggle/input/sar-images/sar_imageexample.jpg'  # Replace with your input JPEG image path
output_image_path = '/kaggle/working/colorized_sar_image.jpg'     # Output path for the colorized image in JPEG format

colorize_sar_image(sar_image_path, output_image_path)
```



Experiment 1 Design SLP

```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```python
iris = load_iris()
X=iris.data
y=iris.target
```

```python
y = np.where(y<=0,-1,1)
```

```python
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

```python
X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

```
((120, 4), (120,), (30, 4), (30,))
```

```
def trainSLP(X,y,epochs=1000,lr=0.01,lp=0.01):

    n_samples,n_features=X.shape
    weights = np.zeros(n_features)
    bias = 0.0

    y_ = np.where(y<=0,-1,1)

    for _ in range(epochs):

        for idx in range(n_samples):

            xi=X[idx]
            yi=y_[idx]
            cond = yi*(np.dot(xi,weights)-bias)>=1
            if(cond):
                weights-=lr*(2*lr*weights)
            else:
                weights-=lr*(2*lr*weights-np.dot(xi,yi))
                bias-=lr*yi

    return weights,bias


def predict(X,weights,bias):

    y_pred=np.dot(X,weights)-bias
    return np.sign(y_pred)


weights,bias=trainSLP(X_train,y_train)


weights,bias
```

```
(array([ 0.05494269, -0.43607723,  0.97027903,  0.49443919]),
 1.8300000000000014)
```

```
prediction = predict(X_test,weights,bias)


from sklearn.metrics import accuracy_score,f1_score

accuracy = accuracy_score(prediction,y_test)
f1=f1_score(prediction,y_test)
print("Accuracy is: ",accuracy)
print("F1 score is: ",f1)
```

```
Accuracy is:  1.0
F1 score is:  1.0
```

```
## Experiment 2 Design MLP


import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.utils import to_categorical


## Normalize the data
iris = load_iris()
X = iris.data
y = iris.target

# One-hot encode the labels
y = to_categorical(y)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)



def build_MLP(activation_function='relu', optimizer='adam'):
    model = Sequential([
        tf.keras.layers.Dense(16, input_shape=(X_train.shape[1],), activation=activation_function),  # Correct input_shape
        tf.keras.layers.Dense(8, activation=activation_function),
        tf.keras.layers.Dense(y_train.shape[1], activation='softmax')
```

```
    ])
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
    return model



activation_functions=['relu','sigmoid','tanh']
optimizers=['adam','rmsprop','sgd']

for act in activation_functions:
    for opt in optimizers:
        # print("Using activation function ",act," , optimizer ",opt)

        model = build_MLP(act,opt)
        history = model.fit(X_train,y_train,epochs=25,batch_size=32,verbose=0)

        loss,accuracy = model.evaluate(X_test,y_test,verbose=0)
        print("Accuracy with the combination Activation_Function = ",act," and Optimizer = ",opt," is ",accuracy)
```

```
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1737120951.182437     216 service.cc:145] XLA service 0x783074005d70 initialized for platform CUDA (this does not guarar
I0000 00:00:1737120951.182561     216 service.cc:153]   StreamExecutor device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1737120951.182584     216 service.cc:153]   StreamExecutor device (1): Tesla T4, Compute Capability 7.5
I0000 00:00:1737120952.386501     216 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the l
Accuracy with the combination Activation_Function =  relu  and Optimizer =  adam  is  0.800000011920929
Accuracy with the combination Activation_Function =  relu  and Optimizer =  rmsprop  is  0.800000011920929
Accuracy with the combination Activation_Function =  relu  and Optimizer =  sgd  is  0.8333333134651184
Accuracy with the combination Activation_Function =  sigmoid  and Optimizer =  adam  is  0.7666666507720947
Accuracy with the combination Activation_Function =  sigmoid  and Optimizer =  rmsprop  is  0.699999988079071
Accuracy with the combination Activation_Function =  sigmoid  and Optimizer =  sgd  is  0.3333333432674408
Accuracy with the combination Activation_Function =  tanh  and Optimizer =  adam  is  0.9666666388511658
Accuracy with the combination Activation_Function =  tanh  and Optimizer =  rmsprop  is  0.8999999761581421
Accuracy with the combination Activation_Function =  tanh  and Optimizer =  sgd  is  0.8666666746139526
```

```
## Experiment 3 - Design and classify a 32x32 images using keras


import tensorflow as tf
from tensorflow.keras import models,layers
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical


(X_train, y_train), (X_test, y_test) = cifar10.load_data()


X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)


def build_model(input_shape,num_classes):

    model = models.Sequential([
        layers.Dense(256,input_shape=input_shape,activation='relu'),
        layers.Flatten(),
        layers.Dense(128,activation = 'relu'),
        layers.Dense(64,activation='relu'),
        layers.Dense(num_classes,activation='softmax')
    ])

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


model = build_model(input_shape=(32,32,3),num_classes=10)
```

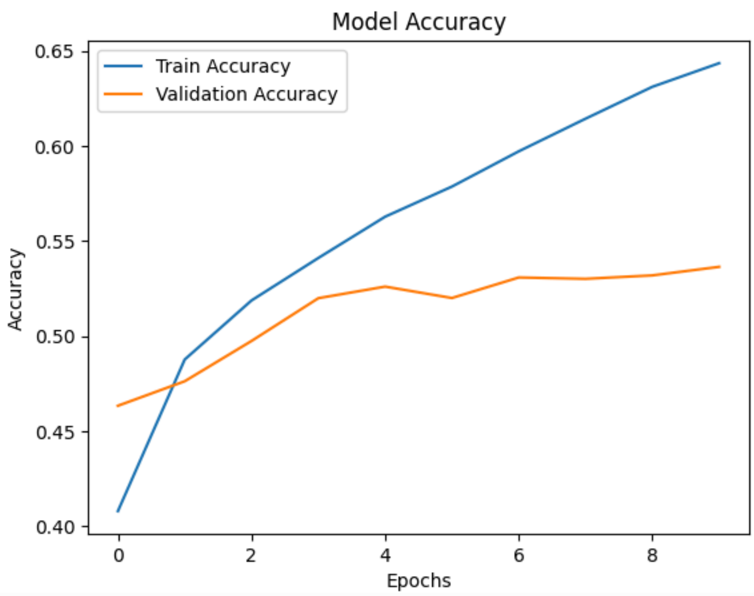
```
/opt/conda/lib/python3.10/site-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` arg
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
history = model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=10,batch_size=64)

loss,accuracy = model.evaluate(X_test,y_test)

print("Accuracy is : ",accuracy)
```

```
Epoch 1/10
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1737121815.734218    2395 service.cc:145] XLA service 0x792f60005df0 initialized for platform CUDA (this does not guarar
I0000 00:00:1737121815.734261    2395 service.cc:153]   StreamExecutor device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1737121815.734265    2395 service.cc:153]   StreamExecutor device (1): Tesla T4, Compute Capability 7.5
```

```
  15/782 ━━━━━━━━━━━━━━━━━━━━ 9s 12ms/step - accuracy: 0.1362 - loss: 5.3591I0000 00:00:1737121817.962873    2395 device_compiler.h::
782/782 ━━━━━━━━━━━━━━━━━━━━ 16s 16ms/step - accuracy: 0.3426 - loss: 2.0585 - val_accuracy: 0.4633 - val_loss: 1.5171
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.4828 - loss: 1.4730 - val_accuracy: 0.4762 - val_loss: 1.4675
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.5200 - loss: 1.3641 - val_accuracy: 0.4974 - val_loss: 1.4129
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.5412 - loss: 1.2987 - val_accuracy: 0.5199 - val_loss: 1.3727
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 11s 13ms/step - accuracy: 0.5629 - loss: 1.2356 - val_accuracy: 0.5260 - val_loss: 1.3558
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.5740 - loss: 1.1966 - val_accuracy: 0.5200 - val_loss: 1.3781
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 11s 13ms/step - accuracy: 0.5990 - loss: 1.1385 - val_accuracy: 0.5308 - val_loss: 1.3607
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 11s 13ms/step - accuracy: 0.6145 - loss: 1.0874 - val_accuracy: 0.5301 - val_loss: 1.3597
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 11s 14ms/step - accuracy: 0.6337 - loss: 1.0485 - val_accuracy: 0.5319 - val_loss: 1.3589
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 11s 14ms/step - accuracy: 0.6481 - loss: 0.9980 - val_accuracy: 0.5364 - val_loss: 1.3703
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.5352 - loss: 1.3720
Accuracy is :  0.5364000201225281
```

```
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'],label='Train Accuracy')
plt.plot(history.history['val_accuracy'],label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




```
## Experiment 4:- Design a CNN Model to train a Multi Class Image Dataset and predict new Image
```

```
import tensorflow as tf
from tensorflow.keras import models,layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os


base_dir = '/kaggle/input/chest-xray-pneumonia/chest_xray'
train_dir = os.path.join(base_dir, 'train')
val_dir = os.path.join(base_dir, 'val')
test_dir = os.path.join(base_dir, 'test')


train_datagen=ImageDataGenerator(
    rescale=1.0/255,
    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=1.0/255
```

```
)
train_datagenerator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(128,128),
    batch_size=64,
    class_mode='categorical'
)
validation_datagenerator = validation_datagen.flow_from_directory(
    val_dir,
    target_size=(128,128),
    batch_size=64,
    class_mode='categorical'
)
```

```
Found 5216 images belonging to 2 classes.
Found 16 images belonging to 2 classes.
```

```
def build_cnn(input_size,num_classes):
    model = models.Sequential([
        layers.Conv2D(32,(3,3),activation='relu',input_shape=input_shape),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(64,(3,3),activation='relu'),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(128,(3,3),activation='relu'),
        layers.MaxPooling2D((2,2)),
        layers.Flatten(),
        layers.Dense(128,activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(num_classes,activation='softmax')
    ])

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


num_classes=len(train_datagenerator.class_indices)
input_shape =(128,128,3)


model = build_cnn(input_shape,num_classes)
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
history = model.fit(
    train_datagenerator,
    epochs=20,
    validation_data=validation_datagenerator
)
```

```
Epoch 1/20
/opt/conda/lib/python3.10/site-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` cl
  self._warn_if_super_not_called()
82/82 ━━━━━━━━━━━━━━━━━━━━ 88s 914ms/step - accuracy: 0.7061 - loss: 0.6127 - val_accuracy: 0.5000 - val_loss: 1.3723
Epoch 2/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 57s 635ms/step - accuracy: 0.7954 - loss: 0.4253 - val_accuracy: 0.6875 - val_loss: 0.7269
Epoch 3/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 57s 629ms/step - accuracy: 0.8514 - loss: 0.3370 - val_accuracy: 0.5625 - val_loss: 0.8134
Epoch 4/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 56s 631ms/step - accuracy: 0.8805 - loss: 0.2754 - val_accuracy: 0.8750 - val_loss: 0.3611
Epoch 5/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 57s 631ms/step - accuracy: 0.8836 - loss: 0.2831 - val_accuracy: 0.7500 - val_loss: 0.5700
Epoch 6/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 57s 636ms/step - accuracy: 0.8955 - loss: 0.2449 - val_accuracy: 0.8125 - val_loss: 0.3649
Epoch 7/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 56s 619ms/step - accuracy: 0.8953 - loss: 0.2352 - val_accuracy: 0.6875 - val_loss: 0.6916
Epoch 8/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 56s 626ms/step - accuracy: 0.8968 - loss: 0.2430 - val_accuracy: 0.6875 - val_loss: 0.8280
Epoch 9/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 57s 636ms/step - accuracy: 0.9105 - loss: 0.2168 - val_accuracy: 0.6250 - val_loss: 0.6837
Epoch 10/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 57s 637ms/step - accuracy: 0.9276 - loss: 0.1813 - val_accuracy: 0.6250 - val_loss: 0.9006
Epoch 11/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 56s 626ms/step - accuracy: 0.9090 - loss: 0.2255 - val_accuracy: 0.5000 - val_loss: 1.2247
Epoch 12/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 56s 627ms/step - accuracy: 0.9130 - loss: 0.2070 - val_accuracy: 0.6250 - val_loss: 0.9053
Epoch 13/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 56s 627ms/step - accuracy: 0.9183 - loss: 0.2058 - val_accuracy: 0.8125 - val_loss: 0.5057
Epoch 14/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 56s 629ms/step - accuracy: 0.9197 - loss: 0.1922 - val_accuracy: 0.7500 - val_loss: 0.7850
Epoch 15/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 56s 623ms/step - accuracy: 0.9357 - loss: 0.1679 - val_accuracy: 0.6250 - val_loss: 0.9429
Epoch 16/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 57s 633ms/step - accuracy: 0.9341 - loss: 0.1782 - val_accuracy: 0.7500 - val_loss: 0.6731
```

```
Epoch 17/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 56s 629ms/step - accuracy: 0.9314 - loss: 0.1687 - val_accuracy: 0.6875 - val_loss: 0.9717
Epoch 18/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 56s 627ms/step - accuracy: 0.9271 - loss: 0.1839 - val_accuracy: 0.8125 - val_loss: 0.3908
Epoch 19/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 56s 625ms/step - accuracy: 0.9291 - loss: 0.1797 - val_accuracy: 0.8125 - val_loss: 0.3338
Epoch 20/20
82/82 ━━━━━━━━━━━━━━━━━━━━ 56s 624ms/step - accuracy: 0.9329 - loss: 0.1653 - val_accuracy: 0.6875 - val_loss: 0.5269
```

```
loss, accuracy = model.evaluate(validation_datagenerator)
print(f"Validation Accuracy: {accuracy * 100:.2f}%")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 159ms/step - accuracy: 0.6875 - loss: 0.5269
Validation Accuracy: 68.75%
```

```
### Experiment 10 Segmentation
```

```
import numpy as np
import tensorflow as tf
from keras.api.utils import to_categorical
from keras.api.datasets import cifar10
```

```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

X_train = X_train.astype("float32") / 255.0
X_test = X_test.astype("float32") / 255.0
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```

```
y_train_seg = (X_train.mean(axis=-1) > 0.5).astype(int)
y_test_seg = (X_test.mean(axis=-1) > 0.5).astype(int)

y_train_seg = y_train_seg[:, :, :, np.newaxis]
y_test_seg = y_test_seg[:, :, :, np.newaxis]
```

```
from keras.api import Model, Input
from keras.api.layers import Conv2D, MaxPooling2D, UpSampling2D, concatenate

def create_unet(input_size=(32, 32, 3)):

    inputs = Input(input_size)

    # Downsampling
    c1 = Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)
    p1 = MaxPooling2D((2, 2))(c1)

    c2 = Conv2D(64, (3, 3), activation='relu', padding='same')(p1)
    p2 = MaxPooling2D((2, 2))(c2)

    # Bottleneck
    c3 = Conv2D(128, (3, 3), activation='relu', padding='same')(p2)

    # Upsampling
    u1 = UpSampling2D((2, 2))(c3)
    m1 = concatenate([u1, c2])

    c4 = Conv2D(64, (3, 3), activation='relu', padding='same')(m1)
    u2 = UpSampling2D((2, 2))(c4)
    m2 = concatenate([u2, c1])

    c5 = Conv2D(32, (3, 3), activation='relu', padding='same')(m2)

    outputs = Conv2D(1, (1, 1), activation='sigmoid')(c5)

    return Model(inputs, outputs)
```

```
# Compile the model
unet_model = create_unet()

unet_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

unet_model.fit(X_train, y_train_seg, validation_data=(X_test, y_test_seg), epochs=10, batch_size=32)
```

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 6ms/step - accuracy: 0.9378 - loss: 0.1443 - val_accuracy: 0.9738 - val_loss: 0.0545
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9875 - loss: 0.0318 - val_accuracy: 0.9904 - val_loss: 0.0232
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9917 - loss: 0.0215 - val_accuracy: 0.9948 - val_loss: 0.0150
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9935 - loss: 0.0165 - val_accuracy: 0.9961 - val_loss: 0.0121
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9951 - loss: 0.0130 - val_accuracy: 0.9971 - val_loss: 0.0098
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9962 - loss: 0.0106 - val_accuracy: 0.9969 - val_loss: 0.0089
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9968 - loss: 0.0091 - val_accuracy: 0.9982 - val_loss: 0.0070
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9971 - loss: 0.0082 - val_accuracy: 0.9973 - val_loss: 0.0071
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9973 - loss: 0.0081 - val_accuracy: 0.9989 - val_loss: 0.0057
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9979 - loss: 0.0064 - val_accuracy: 0.9991 - val_loss: 0.0051
```

```python
import matplotlib.pyplot as plt

pred = unet_model.predict(X_test[:5])

# Display images and masks

for i in range(5):
  plt.subplot(1, 3, 1)

  plt.title("Input Image")
  plt.imshow(X_test[i])

  plt.subplot(1, 3, 2)
  plt.title("Ground Truth Mask")
  plt.imshow(y_test_seg[i].squeeze(), cmap='gray')

  plt.subplot(1, 3, 3)
  plt.title("Predicted Mask")
  plt.imshow(pred[i].squeeze(), cmap='gray')

  plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 490ms/step
```