ViTs:

```
import os
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
import numpy as np

# Parameters
IMAGE_SIZE = 128  # Resize ISAR images to 128x128
PATCH_SIZE = 16
NUM_CLASSES = 7
BATCH_SIZE = 32
EPOCHS = 30
AUTOTUNE = tf.data.AUTOTUNE

# Load ISAR Data
data_dir = "C:/Users/Varsha/OneDrive/Desktop/DRDO/uploaded_dataset"

train_datagen = ImageDataGenerator(
   validation_split=0.2,
   rescale=1./255,
   horizontal_flip=True,
   vertical_flip=True
)

train_generator = train_datagen.flow_from_directory(
   data_dir,
   target_size=(IMAGE_SIZE, IMAGE_SIZE),
   color_mode='grayscale',
   class_mode='categorical',
   subset='training',
   batch_size=BATCH_SIZE
)

val_generator = train_datagen.flow_from_directory(
   data_dir,
   target_size=(IMAGE_SIZE, IMAGE_SIZE),
   color_mode='grayscale',
   class_mode='categorical',
   subset='validation',
   batch_size=BATCH_SIZE
)

# Patch Embedding Layer
class PatchEmbedding(layers.Layer):
   def __init__(self, num_patches, projection_dim):
      super(PatchEmbedding, self).__init__()
```

```
        self.projection = layers.Dense(projection_dim)
        self.position_embedding = layers.Embedding(input_dim=num_patches,
output_dim=projection_dim)

    def call(self, patch):
        positions = tf.range(start=0, limit=tf.shape(patch)[-2], delta=1)
        embedded = self.projection(patch) + self.position_embedding(positions)
        return embedded

# ViT model
def build_vit(input_shape=(IMAGE_SIZE, IMAGE_SIZE, 1),
              patch_size=PATCH_SIZE,
              num_classes=NUM_CLASSES,
              projection_dim=64,
              transformer_layers=4,
              num_heads=4,
              mlp_dim=128):

    inputs = layers.Input(shape=input_shape)
    # Resize image to ensure it's divisible by patch size
    x = layers.Resizing(IMAGE_SIZE, IMAGE_SIZE)(inputs)
    x = layers.Conv2D(filters=projection_dim,
                      kernel_size=patch_size,
                      strides=patch_size,
                      padding='valid')(x)

    # Flatten patches
    x = layers.Reshape((-1, projection_dim))(x)

    # Positional + Linear Embedding
    embedding_layer = PatchEmbedding(num_patches=x.shape[1], projection_dim=projection_dim)
    x = embedding_layer(x)

    # Transformer blocks
    for _ in range(transformer_layers):
        # Layer norm 1
        x1 = layers.LayerNormalization(epsilon=1e-6)(x)
        # Multi-head Self Attention
        attention = layers.MultiHeadAttention(num_heads=num_heads, key_dim=projection_dim)(x1, x1)
        x2 = layers.Add()([attention, x])

        # Layer norm 2
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        # MLP block
        mlp = layers.Dense(mlp_dim, activation='gelu')(x3)
        mlp = layers.Dense(projection_dim)(mlp)
        x = layers.Add()([mlp, x2])
```

```
    # Classification head
    x = layers.LayerNormalization(epsilon=1e-6)(x)
    x = layers.GlobalAveragePooling1D()(x)
    x = layers.Dense(mlp_dim, activation='relu')(x)
    x = layers.Dropout(0.3)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)

    model = models.Model(inputs=inputs, outputs=outputs)
    return model

# Build and compile
vit_model = build_vit()
vit_model.compile(optimizer=Adam(1e-3), loss='categorical_crossentropy', metrics=['accuracy'])
vit_model.summary()

# Train
history = vit_model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=EPOCHS,
)

# Save model
vit_model.save("vit_isar_model.keras")
# Evaluate model on validation data
loss, accuracy = vit_model.evaluate(val_generator)
print(f"Validation Accuracy: {accuracy * 100:.2f}%")
```

dcnn_final.py:

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
import tensorflow as tf
import pandas as pd
import keras
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import PIL
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D,Flatten, Dropout, Activation, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.utils import load_img
import math
from flask import Flask,render_template,request
from keras.callbacks import ModelCheckpoint,EarlyStopping,LearningRateScheduler
from keras.optimizers import Adam, SGD, RMSprop, Ftrl,Adagrad
```

```
inp_num_classes=7
inp_epochs=30
end_epoch=inp_epochs
inp_batch_size_train=32
inp_batch_size_test=12
inp_initial_learning_rate=0.1
decay=inp_initial_learning_rate/inp_epochs
inp_patience=20
inp_save_freq=inp_batch_size_test//1
inp_show_graphics=1
lr_list=[]

trdata=ImageDataGenerator(
   validation_split=0.2,
   horizontal_flip=True,
   vertical_flip=True,
   rescale=1.0/255.0
)
traindata=trdata.flow_from_directory(
   directory="C:/Users/Varsha/OneDrive/Desktop/DRDO/uploaded_dataset",
   target_size=(200,500),
   color_mode="grayscale",
   batch_size=inp_batch_size_train,
   subset="training",
   class_mode="categorical",
   shuffle=True
)
testdata=trdata.flow_from_directory(
   directory="C:/Users/Varsha/OneDrive/Desktop/DRDO/uploaded_dataset",
   target_size=(200,500),
   color_mode="grayscale",
   batch_size=inp_batch_size_test,
   subset="validation",
   class_mode="categorical",
   shuffle=False
)
model=Sequential()
model.add(Conv2D(input_shape=(200,500,1),filters=2,kernel_size=(3,3),padding="same"))
model.add(Activation("relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model.add(Conv2D(filters=4,kernel_size=(3,3),padding="same"))
model.add(Activation("relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model.add(Conv2D(filters=1,kernel_size=(3,3),padding="same"))
model.add(Activation("relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
```

```
model.add(Conv2D(filters=1,kernel_size=(3,3),padding="same"))
model.add(Activation("relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model.add(Flatten())
model.add(Dense(units=64))
model.add(Activation("relu"))
model.add(Dropout(0.1))
model.add(Dense(units=inp_num_classes))
model.add(Activation("softmax"))
model.summary()

opt=Adagrad(learning_rate=inp_initial_learning_rate)
model.compile(optimizer=opt,loss=keras.losses.categorical_crossentropy,metrics=['accuracy'])
dcnn=model.fit(traindata,
        validation_data=testdata,
        epochs=inp_epochs,
        steps_per_epoch=traindata.samples//inp_batch_size_train,
        validation_steps=testdata.samples//inp_batch_size_test,)
model.save("dcnn_model.keras")
scores=model.evaluate(testdata,steps=testdata.samples//inp_batch_size_test,verbose=1)
print("Accuracy:%.2f%%"%(scores[1]*100))
Y_pred=model.predict(testdata,testdata.samples//inp_batch_size_test,verbose=1)
y_pred=np.argmax(Y_pred,axis=1)
class_labels=list(testdata.class_indices.keys())
print("class labels:",class_labels)
print(type(class_labels),len(class_labels))

from sklearn.metrics import classification_report,confusion_matrix
cm=confusion_matrix(testdata.classes,y_pred)
print("confusion matrix:\n",cm)

cr=classification_report(testdata.classes,y_pred,target_names=class_labels)
print("classification report:\n",cr)

df_cm=pd.DataFrame(cm,range(len(class_labels)),range(len(class_labels)))
plt.figure(figsize=(10,8))
sns.set(font_scale=1.4)
sns.heatmap(df_cm,annot=True,annot_kws={"size":12})
if(inp_show_graphics==1):
   plt.show()
```

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
```

```
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 <title>Upload Dataset - ISAR Ship Classification</title>
 <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}" />
</head>
<body>
 <div class="logo-container">
  <img class="logo" src="{{ url_for('static', filename='logo.png') }}" alt="DRDO Logo" />
 </div>

 <div class="container">
  <header>
   <h1>DRDO Internship Project</h1>
   <h2>ISAR Ship Type Classification System</h2>
  </header>

  <section class="form-section">
   <h3>Upload Dataset</h3>
   <form action="/" method="post" enctype="multipart/form-data">
    <div class="form-group">
     <label for="model_type">Select Model Type:</label>
     <select name="model_type" id="model_type" required>
      <option value="" disabled selected>Select a model</option>
      <option value="cnn">CNN</option>
      <option value="dcnn">DCNN</option>
      <option value="vit">ViTs</option>
     </select>
    </div>

    <div class="form-group">
     <label for="dataset">Upload ZIP Dataset:</label>
     <input type="file" id="dataset" name="dataset" accept=".zip" required />
    </div>

    <button type="submit" class="submit-button">Upload and Train</button>
   </form>

   {% with messages = get_flashed_messages() %}
    {% if messages %}
     <div class="flash-messages">
      {% for message in messages %}
       <p>{{ message }}</p>
      {% endfor %}
     </div>
    {% endif %}
   {% endwith %}
  </section>
 </div>
</body>
```

```
</html>
```

Train.html:

```
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <title>training results</title>
    <link rel="stylesheet" href="{{url_for('static',filename='style.css')}}">
  </head>
  <body>
    <div class="logo-container">
      <img class="logo" src="{{url_for('static',filename='logo.png')}}" alt="logo">
    </div>
    <div class="container">
      <h2>model training completed</h2>
      <p>Accuracy:<strong>{{accuracy}}</strong></p>
      <h3>Confusion Matrix</h3>
      <table>
        {% for row in confusion_matrix%}
        <tr>
          {% for item in row %}
          <td>{{item}}</td>
          {% endfor %}
        </tr>
        {% endfor %}

      </table>
      <h3>Classification report</h3>
      <pre>{{classification_report}}</pre>
      <h4>training completed</h4>
      <p><strong>trained model saved at:</strong></p>
      <a href="{{url_for('static',filename='improved_image_model.keras')}}" download>download
model</a>
      <a href="/test"><button type="submit">proceed to testing</button></a>
    </div>
  </body>
</html>
```

Test.html:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,initial-scale=1.0">
  <title>Test model</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
```

```
<body>
  <div class="logo-container">
    <img class="logo" src="{{ url_for('static', filename='logo.png') }}" alt="logo">
  </div>
  <div class="container">
    <h1>Upload model and dataset for testing</h1>
    <form action="/test" method="post" enctype="multipart/form-data">
      <label for="model-file">Choose model file (.keras):</label>
      <input type="file" name="model_file" id="model-file" accept=".keras" required>
      <br><br>
      <label for="dataset">Choose dataset (.zip):</label>
      <input type="file" name="dataset_zip" id="dataset" accept=".zip" required>
      <br><br>
      <button type="submit">Upload and Test</button>
    </form>

    {% if show_results %}
      <h2>Test Results</h2>
      <h3>Accuracy: {{ accuracy }}</h3>

      <h3>Confusion Matrix</h3>
      <div class="matrix-container">
        <table class="confusion-matrix">
          <tr>
            <th></th>
            {% for label in class_labels %}
            <th>Predicted: {{ label }}</th>
            {% endfor %}
          </tr>
          {% for i, row in confusion_matrix %}
          <tr>
            <th>Actual: {{ class_labels[i] }}</th>
            {% for value in row %}
            <td>{{ value }}</td>
            {% endfor %}
          </tr>
          {% endfor %}
        </table>
      </div>

      <h3>Classification Report</h3>
      <table class="classification-report">
        <tr>
          <th>Class</th>
          <th>Precision</th>
          <th>Recall</th>
          <th>F1-Score</th>
          <th>Support</th>
```

```
            </tr>
            {% for label in class_labels %}
            <tr>
              <td>{{ label }}</td>
              <td>{{ "%.2f"|format(classification_report[label]['precision']) }}</td>
              <td>{{ "%.2f"|format(classification_report[label]['recall']) }}</td>
              <td>{{ "%.2f"|format(classification_report[label]['f1-score']) }}</td>
              <td>{{ classification_report[label]['support'] }}</td>
            </tr>
            {% endfor %}
          </table>
        {% endif %}

        <br>
        <form action="/predict" method="get">
          <button type="submit">Go to prediction</button>
        </form>
      </div>
    </body>
    </html>
```

Predict.html:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,initial-scale=1.0">
  <title>Image Prediction</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
  <div class="container">
    <h2>Upload a model and an image for prediction</h2>
    <form action="/predict" method="post" enctype="multipart/form-data">
      <label for="model-file">Choose model file (.keras):</label>
      <input type="file" name="model_file" id="model-file" accept=".keras" required>
      <br><br>

      <label for="imagefile">Choose image (.png):</label>
      <input type="file" name="imagefile" id="imagefile" accept=".png" required>
      <br><br>

      <button type="submit">Predict</button>
    </form>

    {% if prediction and selected_file %}
    <h3>Prediction: <strong>{{ prediction }}</strong></h3>
```

```
      <img src="{{ url_for('static', filename='uploads/' + selected_file) }}" alt="Uploaded image"
class="uploaded_img" style="max-width: 200px;">
      {% endif %}

      <br><br>
      <a href="/"><button>Go back</button></a>
   </div>
</body>
</html>
```

Style.css:

```
body{
   display:flex;
   justify-content: center;
   align-items: center;
   height: 100vh;
   margin: 0;
   font-family: Arial, sans-serif;
   background: linear-gradient(to right,#32c5ed98,#17457c);
   color: rgb(6,49,88);
   text-align: center;
   position: relative;
}
h1
{
   margin: 20px;
   font-size: 2.5rem;
   font-weight: bold;
}
h2{
   margin-bottom: 20px;
   font-size: 1.8rem;
   font-weight: bold;
   color: azure;
}
button{
   background-color: #05436da3;
   border: none;
   color: white;
   padding: 15px 32px;
   text-align: center;
   text-decoration: none;
   display: inline-block;
   font-size: 1rem;
   margin: 10px;
   cursor: pointer;
   border-radius: 5px;
   transition: background-color 0.3s ease;
```

```
}
button:hover{
   background-color: #1c4374;
}
table{
   margin:0 auto;
   border-collapse: collapse;
   width: 80%;
   background-color: white;
   color: black;
   margin-bottom: 20px;
}
th,td{
   padding: 10px;
   border: 1px solid #ddd;
}
pre
{
   text-align: left;
   background-color: white;
   color: black;
   padding: 10px;
   border-radius: 5px;
   white-space: pre-wrap;
   overflow-x: auto;
}
.logo
{
   position: absolute;
   top:50%;
   left:50%;
   transform: translate(-50%,-50%);
   opacity: 0.15;
   max-width: 100%;
   max-height: 100%;
   z-index: -1;
}
.content
{
   z-index: 1;
}
.flash-messages
{
   margin-top: 20px;
   color: #d9534f;
   font-weight: bold;
}
```

```
/* Confusion Matrix and Classification Report Styles */
.matrix-container {
    overflow-x: auto;
    margin: 20px 0;
}

.confusion-matrix, .classification-report {
    border-collapse: collapse;
    margin: 20px 0;
    background-color: white;
    box-shadow: 0 1px 3px rgba(0,0,0,0.2);
}

.confusion-matrix th, .confusion-matrix td,
.classification-report th, .classification-report td {
    padding: 8px 12px;
    border: 1px solid #ddd;
    text-align: center;
}

.confusion-matrix th, .classification-report th {
    background-color: #f5f5f5;
    font-weight: bold;
}

.confusion-matrix tr:nth-child(even),
.classification-report tr:nth-child(even) {
    background-color: #f9f9f9;
}

.confusion-matrix tr:hover,
.classification-report tr:hover {
    background-color: #f0f0f0;
}

/* Make the first column (class labels) stand out */
.confusion-matrix th:first-child,
.classification-report th:first-child {
    background-color: #e0e0e0;
    font-weight: bold;
}

/* Add some spacing between sections */
h2, h3 {
    margin-top: 30px;
    margin-bottom: 15px;
}
```

App.py:

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import tensorflow as tf
import pandas as pd
import keras
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import zipfile
import shutil
from tensorflow.keras import layers
from tensorflow.keras import models
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout, Activation
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from flask import Flask, render_template, request, redirect, url_for, flash, session
from tensorflow.keras.optimizers import Adam, Adagrad
from sklearn.metrics import classification_report, confusion_matrix
from werkzeug.utils import secure_filename

app = Flask(__name__)
UPLOAD_FOLDER = 'uploads'
MODEL_FOLDER = 'models'
app.secret_key = 'secret123'

inp_num_classes = 7
inp_epochs = 30
inp_batch_size_train = 32
inp_batch_size_test = 12
inp_initial_learning_rate = 0.001

os.makedirs(UPLOAD_FOLDER, exist_ok=True)
os.makedirs(MODEL_FOLDER, exist_ok=True)
model_path = os.path.join(MODEL_FOLDER, "improved_image_model.keras")

allowed_extensions = {'png'}

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in allowed_extensions

@app.route("/", methods=["GET", "POST"])
def upload_dataset():
    if request.method == "POST":
        file = request.files.get("dataset")
        model_type = request.form.get("model_type")
        session['model_type'] = model_type
```

```
        if not file or file.filename == "":
            flash("No dataset selected")
            return redirect(request.url)
        shutil.rmtree(UPLOAD_FOLDER, ignore_errors=True)
        os.makedirs(UPLOAD_FOLDER)
        dataset_path = os.path.join(UPLOAD_FOLDER, file.filename)
        file.save(dataset_path)
        with zipfile.ZipFile(dataset_path, "r") as zip_ref:
            zip_ref.extractall(UPLOAD_FOLDER)
        os.remove(dataset_path)
        extracted_root = os.path.join(UPLOAD_FOLDER, os.listdir(UPLOAD_FOLDER)[0])
        if os.path.exists(extracted_root):
            for subfolder in os.listdir(extracted_root):
                shutil.move(os.path.join(extracted_root, subfolder), UPLOAD_FOLDER)
            shutil.rmtree(extracted_root)
        flash("Dataset uploaded successfully. Training started.")
        return redirect(url_for("train_model"))
    return render_template("index.html")

@app.route("/train", methods=["GET"])
def train_model():
    model_type = session.get('model_type', 'cnn')

    if model_type == 'dcnn':
        trdata = ImageDataGenerator(
            validation_split=0.2,
            horizontal_flip=True,
            vertical_flip=True,
            rescale=1.0/255.0
        )
        traindata = trdata.flow_from_directory(
            directory=UPLOAD_FOLDER,
            target_size=(200, 500),
            color_mode="grayscale",
            batch_size=inp_batch_size_train,
            subset="training",
            class_mode="categorical",
            shuffle=True
        )
        testdata = trdata.flow_from_directory(
            directory=UPLOAD_FOLDER,
            target_size=(200, 500),
            color_mode="grayscale",
            batch_size=inp_batch_size_test,
            subset="validation",
            class_mode="categorical",
            shuffle=False
        )
```

```
model = Sequential()
model.add(Conv2D(input_shape=(200, 500, 1), filters=2, kernel_size=(3, 3), padding="same"))
model.add(Activation("relu"))
model.add(MaxPool2D(pool_size=(2, 2), strides=(2, 2)))

model.add(Conv2D(filters=4, kernel_size=(3, 3), padding="same"))
model.add(Activation("relu"))
model.add(MaxPool2D(pool_size=(2, 2), strides=(2, 2)))

model.add(Conv2D(filters=1, kernel_size=(3, 3), padding="same"))
model.add(Activation("relu"))
model.add(MaxPool2D(pool_size=(2, 2), strides=(2, 2)))

model.add(Conv2D(filters=2, kernel_size=(3, 3), padding="same"))
model.add(Activation("relu"))
model.add(MaxPool2D(pool_size=(2, 2), strides=(2, 2)))

model.add(Flatten())
model.add(Dense(units=64))
model.add(Activation("relu"))
model.add(Dropout(0.1))
model.add(Dense(units=inp_num_classes))
model.add(Activation("softmax"))

opt = Adagrad(learning_rate=0.1)
model.compile(optimizer=opt, loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])

history = model.fit(
   traindata,
   validation_data=testdata,
   epochs=inp_epochs,
   steps_per_epoch=traindata.samples // inp_batch_size_train,
   validation_steps=testdata.samples // inp_batch_size_test,
)
model.save(os.path.join(MODEL_FOLDER, "dcnn_model.keras"))

scores = model.evaluate(testdata, steps=testdata.samples // inp_batch_size_test, verbose=1)
accuracy = "Accuracy:%.2f%%" % (scores[1] * 100)
Y_pred = model.predict(testdata, testdata.samples // inp_batch_size_test, verbose=1)
y_pred = np.argmax(Y_pred, axis=1)
class_labels = list(testdata.class_indices.keys())
conf_matrix = confusion_matrix(testdata.classes, y_pred)
class_report = classification_report(testdata.classes, y_pred, target_names=class_labels)
return render_template('train.html', accuracy=accuracy,
                    confusion_matrix=conf_matrix,
                    classification_report=class_report,
                    show_result=True)
```

```
    elif model_type == 'vit':
        IMAGE_SIZE = 128
        PATCH_SIZE = 16
        NUM_CLASSES = inp_num_classes
        BATCH_SIZE = inp_batch_size_train
        EPOCHS = inp_epochs
        projection_dim = 64
        transformer_layers = 4
        num_heads = 4
        mlp_dim = 128

        trdata = ImageDataGenerator(
            validation_split=0.2,
            horizontal_flip=True,
            vertical_flip=True,
            rescale=1.0/255.0
        )
        traindata = trdata.flow_from_directory(
            directory=UPLOAD_FOLDER,
            target_size=(IMAGE_SIZE, IMAGE_SIZE),
            color_mode="grayscale",
            batch_size=BATCH_SIZE,
            subset="training",
            class_mode="categorical",
            shuffle=True
        )
        testdata = trdata.flow_from_directory(
            directory=UPLOAD_FOLDER,
            target_size=(IMAGE_SIZE, IMAGE_SIZE),
            color_mode="grayscale",
            batch_size=BATCH_SIZE,
            subset="validation",
            class_mode="categorical",
            shuffle=False
        )

        class PatchEmbedding(layers.Layer):
            def __init__(self, num_patches, projection_dim):
                super(PatchEmbedding, self).__init__()
                self.projection = layers.Dense(projection_dim)
                self.position_embedding = layers.Embedding(input_dim=num_patches,
output_dim=projection_dim)

            def call(self, patch):
                positions = tf.range(start=0, limit=tf.shape(patch)[-2], delta=1)
                embedded = self.projection(patch) + self.position_embedding(positions)
                return embedded
```

```
def build_vit(input_shape=(IMAGE_SIZE, IMAGE_SIZE, 1),
              patch_size=PATCH_SIZE,
              num_classes=NUM_CLASSES,
              projection_dim=projection_dim,
              transformer_layers=transformer_layers,
              num_heads=num_heads,
              mlp_dim=mlp_dim):

    inputs = layers.Input(shape=input_shape)
    x = layers.Resizing(IMAGE_SIZE, IMAGE_SIZE)(inputs)
    x = layers.Conv2D(filters=projection_dim,
                      kernel_size=patch_size,
                      strides=patch_size,
                      padding='valid')(x)
    x = layers.Reshape((-1, projection_dim))(x)

    embedding_layer = PatchEmbedding(num_patches=x.shape[1], projection_dim=projection_dim)
    x = embedding_layer(x)

    for _ in range(transformer_layers):
        x1 = layers.LayerNormalization(epsilon=1e-6)(x)
        attention = layers.MultiHeadAttention(num_heads=num_heads, key_dim=projection_dim)(x1,
x1)
        x2 = layers.Add()([attention, x])

        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        mlp = layers.Dense(mlp_dim, activation='gelu')(x3)
        mlp = layers.Dense(projection_dim)(mlp)
        x = layers.Add()([mlp, x2])

    x = layers.LayerNormalization(epsilon=1e-6)(x)
    x = layers.GlobalAveragePooling1D()(x)
    x = layers.Dense(mlp_dim, activation='relu')(x)
    x = layers.Dropout(0.3)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)

    return models.Model(inputs=inputs, outputs=outputs)

vit_model = build_vit()
vit_model.compile(optimizer=Adam(1e-3), loss='categorical_crossentropy', metrics=['accuracy'])

checkpoint_cb = ModelCheckpoint(os.path.join(MODEL_FOLDER, 'vit_model.keras'),
                                save_best_only=True, monitor='val_loss')
early_stopping_cb = EarlyStopping(monitor='val_loss', patience=10)

history = vit_model.fit(
    traindata,
    validation_data=testdata,
```

```
        epochs=EPOCHS,
        callbacks=[checkpoint_cb, early_stopping_cb],
        steps_per_epoch=traindata.samples // BATCH_SIZE,
        validation_steps=testdata.samples // BATCH_SIZE
    )

    scores = vit_model.evaluate(testdata, steps=testdata.samples // BATCH_SIZE)
    accuracy = "Accuracy:%.2f%%" % (scores[1] * 100)
    Y_pred = vit_model.predict(testdata, testdata.samples // BATCH_SIZE)
    y_pred = np.argmax(Y_pred, axis=1)
    class_labels = list(testdata.class_indices.keys())
    conf_matrix = confusion_matrix(testdata.classes, y_pred)
    class_report = classification_report(testdata.classes, y_pred, target_names=class_labels)
    return render_template('train.html', accuracy=accuracy,
                           confusion_matrix=conf_matrix,
                           classification_report=class_report,
                           show_result=True)

else:  # default CNN
    trdata = ImageDataGenerator(
        validation_split=0.2,
        horizontal_flip=True,
        vertical_flip=True,
        rescale=1.0/255.0
    )
    traindata = trdata.flow_from_directory(
        directory=UPLOAD_FOLDER,
        target_size=(200, 500),
        color_mode="grayscale",
        batch_size=inp_batch_size_train,
        subset="training",
        class_mode="categorical",
        shuffle=True
    )
    testdata = trdata.flow_from_directory(
        directory=UPLOAD_FOLDER,
        target_size=(200, 500),
        color_mode="grayscale",
        batch_size=inp_batch_size_test,
        subset="validation",
        class_mode="categorical",
        shuffle=False
    )

    model = Sequential()
    model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(200, 500, 1), activation='relu'))
    model.add(MaxPool2D(pool_size=(2, 2)))
    model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
```

```
        model.add(MaxPool2D(pool_size=(2, 2)))
        model.add(Flatten())
        model.add(Dense(128, activation='relu'))
        model.add(Dropout(0.5))
        model.add(Dense(inp_num_classes, activation='softmax'))

        model.compile(optimizer=Adam(inp_initial_learning_rate),
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])

        history = model.fit(
          traindata,
          validation_data=testdata,
          epochs=inp_epochs,
          steps_per_epoch=traindata.samples // inp_batch_size_train,
          validation_steps=testdata.samples // inp_batch_size_test,
        )

        model.save(model_path)

        scores = model.evaluate(testdata, steps=testdata.samples // inp_batch_size_test)
        accuracy = "Accuracy:%.2f%%" % (scores[1] * 100)
        Y_pred = model.predict(testdata, testdata.samples // inp_batch_size_test)
        y_pred = np.argmax(Y_pred, axis=1)
        class_labels = list(testdata.class_indices.keys())
        conf_matrix = confusion_matrix(testdata.classes, y_pred)
        class_report = classification_report(testdata.classes, y_pred, target_names=class_labels)

        return render_template('train.html', accuracy=accuracy,
                               confusion_matrix=conf_matrix,
                               classification_report=class_report,
                               show_result=True)

@app.route("/test", methods=["GET", "POST"])
def test_model():
    if request.method == "POST":
        file = request.files.get("test_image")
        if not file or file.filename == "":
            flash("No image selected for testing")
            return redirect(request.url)
        if not allowed_file(file.filename):
            flash("Only PNG images allowed")
            return redirect(request.url)
        filename = secure_filename(file.filename)
        test_image_path = os.path.join(UPLOAD_FOLDER, filename)
        file.save(test_image_path)

        model_type = session.get('model_type', 'cnn')
```

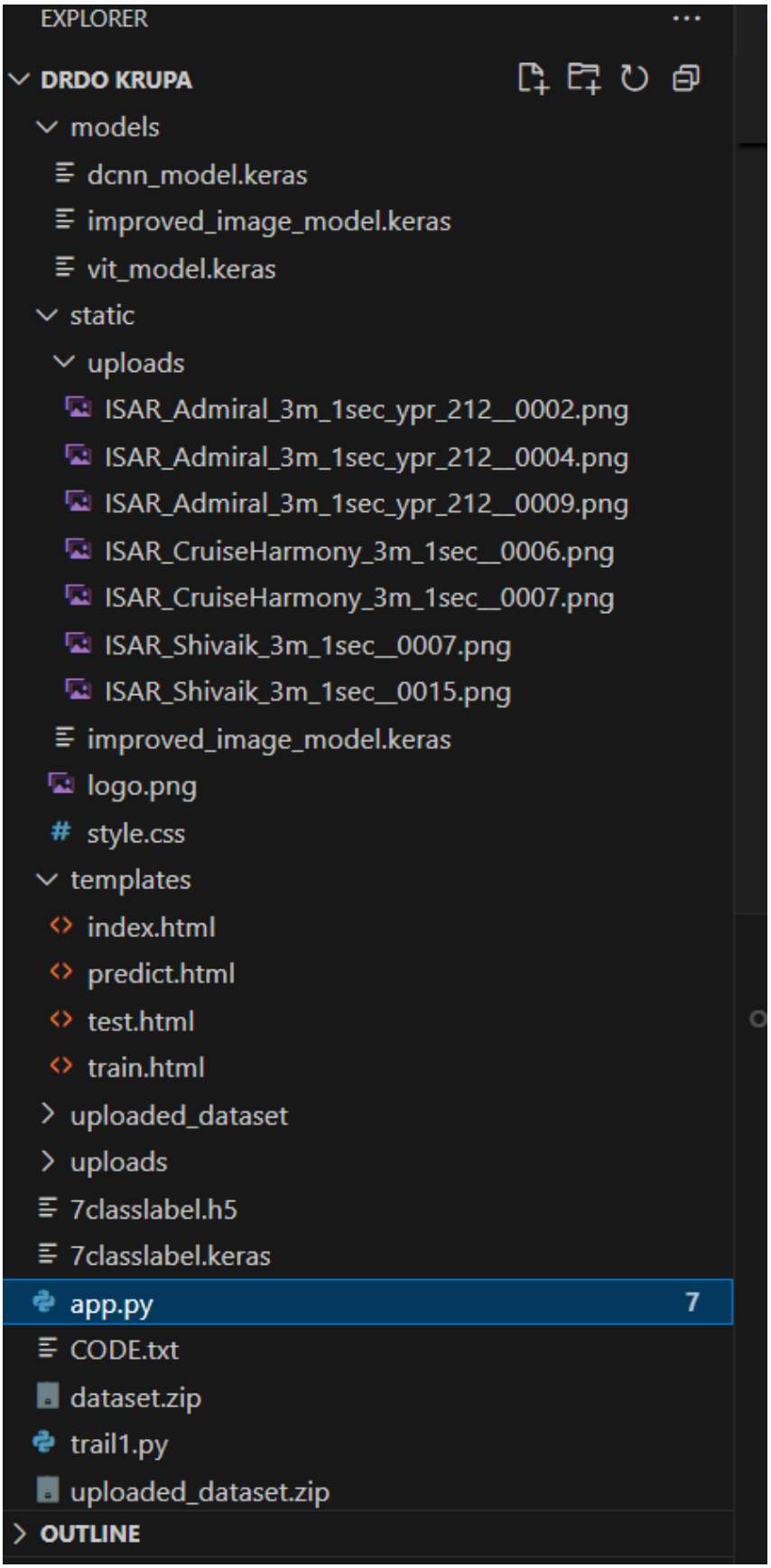
```
        if model_type == 'dcnn':
            model = load_model(os.path.join(MODEL_FOLDER, "dcnn_model.keras"))
            img = load_img(test_image_path, color_mode="grayscale", target_size=(200, 500))
            x = img_to_array(img) / 255.0
            x = np.expand_dims(x, axis=0)  # batch dimension
            preds = model.predict(x)
            predicted_class = np.argmax(preds, axis=1)[0]
            # You can map predicted_class to class names if you save class indices in session or file.
            return f"Predicted class index: {predicted_class}"

        elif model_type == 'vit':
            model = load_model(os.path.join(MODEL_FOLDER, 'vit_model.keras'))
            img = load_img(test_image_path, color_mode="grayscale", target_size=(128, 128))
            x = img_to_array(img) / 255.0
            x = np.expand_dims(x, axis=0)
            preds = model.predict(x)
            predicted_class = np.argmax(preds, axis=1)[0]
            return f"Predicted class index: {predicted_class}"

        else:  # default CNN
            model = load_model(model_path)
            img = load_img(test_image_path, color_mode="grayscale", target_size=(200, 500))
            x = img_to_array(img) / 255.0
            x = np.expand_dims(x, axis=0)
            preds = model.predict(x)
            predicted_class = np.argmax(preds, axis=1)[0]
            return f"Predicted class index: {predicted_class}"

    return render_template("test.html")

if __name__ == "__main__":
    app.run(debug=True)
```


EXPLORER
DRDO KRUPA
models
dcnn_model.keras
improved_image_model.keras
vit_model.keras
static
uploads
ISAR_Admiral_3m_1sec_ypr_212__0002.png
ISAR_Admiral_3m_1sec_ypr_212__0004.png
ISAR_Admiral_3m_1sec_ypr_212__0009.png
ISAR_CruiseHarmony_3m_1sec__0006.png
ISAR_CruiseHarmony_3m_1sec__0007.png
ISAR_Shivaik_3m_1sec__0007.png
ISAR_Shivaik_3m_1sec__0015.png
improved_image_model.keras
logo.png
style.css
templates
index.html
predict.html
test.html
train.html
uploaded_dataset
uploads
7classlabel.h5
7classlabel.keras
app.py
7
CODE.txt
dataset.zip
trail1.py
uploaded_dataset.zip
OUTLINE