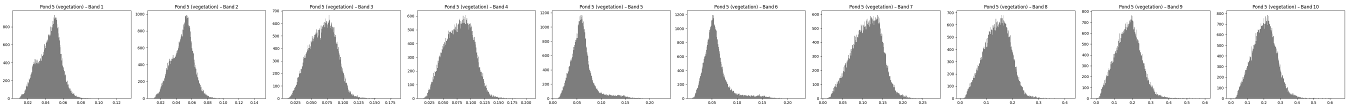
```
import rasterio
import numpy as np
import matplotlib.pyplot as plt
from itertools import combinations
import cv2
from skimage.filters import threshold_otsu, threshold_multiotsu   # ← NEW

# -------------------------------------------------
# 1   – Histogram helper (unchanged)
# -------------------------------------------------
def plot_histograms(path, title):
    with rasterio.open(path) as src:
        bands = src.read()       # (B, H, W)
    B = bands.shape[0]
    fig, ax = plt.subplots(1, B, figsize=(5*B, 4))
    for i in range(B):
        ax[i].hist(bands[i].ravel(), bins=256, color='gray')
        ax[i].set_title(f'{title} – Band {i+1}')
    plt.tight_layout(); plt.show()
    return bands

# -------------------------------------------------
# 2   – Score a band or band-combo with single-Otsu
#       (same logic as earlier)
# -------------------------------------------------
def otsu_score(bands, idxs):
    img = bands[idxs[0]] if len(idxs) == 1 \
          else np.mean(bands[idxs].astype(np.float32), axis=0)
    img = cv2.normalize(img, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
    thr = threshold_otsu(img)
    bin_ = img > thr
    fg, bg = img[bin_], img[~bin_]
    score = np.inf if fg.size == 0 or bg.size == 0 else np.var(fg)+np.var(bg)
    return score, img, idxs

# -------------------------------------------------
# 3   – Load POND 5, pick “best” image
# -------------------------------------------------
POND5 = '/content/pond5_stacked_05m.tif'
pond5 = plot_histograms(POND5, 'Pond 5 (vegetation)')

B = pond5.shape[0]
candidates = [otsu_score(pond5, [i]) for i in range(B)]
candidates += [otsu_score(pond5, list(c)) for c in combinations(range(B), 2)]

best_score, best_img, best_idxs = min(candidates, key=lambda x: x[0])

# -------------------------------------------------
# 4   – ★ Three-level segmentation with multi-Otsu
# -------------------------------------------------
# Get two thresholds → three classes (0,1,2)
thr1, thr2 = threshold_multiotsu(best_img, classes=3)
regions = np.digitize(best_img, bins=(thr1, thr2))   # 0 = darkest, 2 = brightest

# You may need to *confirm visually* which label is which.
# Typical assumption for NIR-rich stacks:
#   class 0 = outside (land/roads) – darkest
#   class 1 = pond water – mid-intensity
#   class 2 = vegetation – brightest
# If that mapping is wrong, just swap the colour lines below.

# -------------------------------------------------
# 5   – Colour map: outside = red, water = blue, veg = green
# -------------------------------------------------
palette = {
    0: (255,   0,   0),   # outside – red
    1: (  0,   0, 255),   # water   – blue
    2: (  0, 255,   0)    # veg     – green
}
h, w = best_img.shape
rgb = np.zeros((h, w, 3), dtype=np.uint8)
for cls, colour in palette.items():
    rgb[regions == cls] = colour

# -------------------------------------------------
# 6   – Display
# -------------------------------------------------
plt.figure(figsize=(8, 8))
plt.imshow(rgb)
lbl = ' & '.join([f'Band {i+1}' for i in best_idxs])
plt.title(f'3-class Multi-Otsu on {lbl}\nthresholds: {thr1}, {thr2}   |   best score: {best_score:.2f}')
```

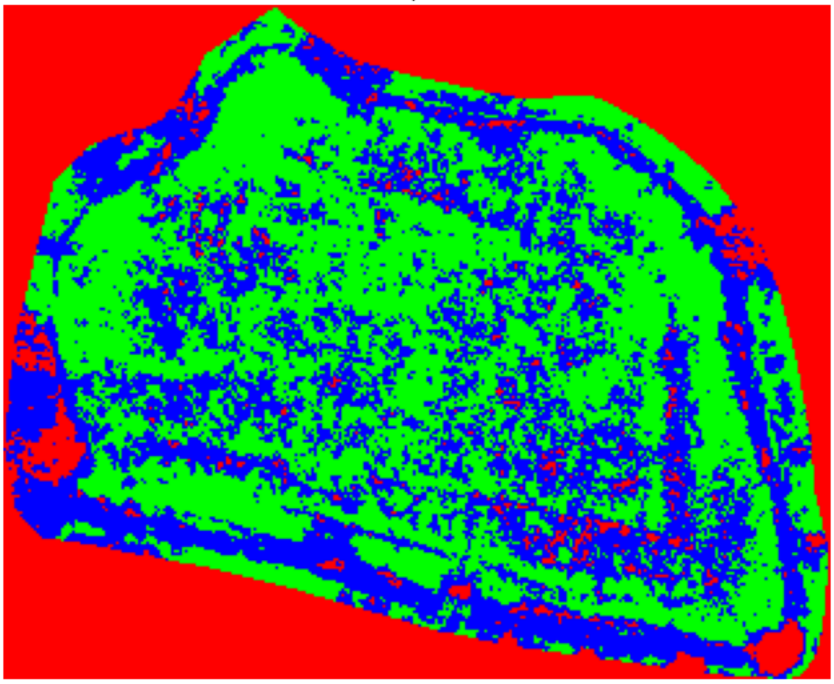
What can I help you build?

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



```
<ipython-input-74-3690427619>:29: RuntimeWarning: invalid value encountered in cast
  img = cv2.normalize(img, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
```

3-class Multi-Otsu on Band 2
thresholds: 27, 70 | best score: 574.61

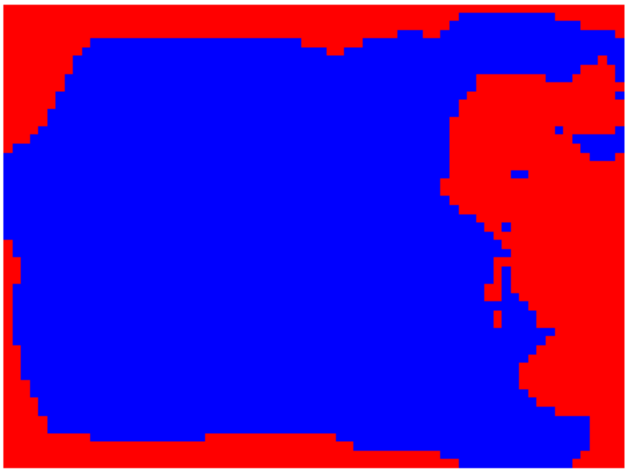


```
# ------------------------------------------------------------------
# 0.  Imports
# ------------------------------------------------------------------
import rasterio
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from itertools import combinations
import cv2
from skimage.filters import threshold_otsu, threshold_multiotsu
from sklearn.decomposition import FastICA
from pathlib import Path

# ------------------------------------------------------------------
# 1.  Config --- tweak here if band order ≠ standard Landsat/Sentinel
# ------------------------------------------------------------------
# Pick the (0-based) indices of RED and NIR bands in your 10-band stack
RED_BAND   = 3   # e.g. Band-4  (adjust!)
NIR_BAND   = 7   # e.g. Band-8  (adjust!)

DATA_DIR   = Path('/content')
FILES      = {
    'pond11': DATA_DIR / 'pond11_stacked_05m.tif',   # no vegetation
    'pond3' : DATA_DIR / 'pond3_stacked_05m.tif',    # no vegetation
    'pond5' : DATA_DIR / 'pond5_stacked_05m.tif'     # vegetation present
}

# Colour map for quick visual checks
COLS_3 = {0:(255,  0,  0), 1:(  0,  0,255), 2:(  0,255,  0)}   # o(land)=red, i(water)=blue, v=green
COLS_2 = {0:(255,  0,  0), 1:(  0,  0,255)}                    # o=red, i=blue

# ------------------------------------------------------------------
# 2.  Helper functions
# ------------------------------------------------------------------
def load_stack(path):
    with rasterio.open(path) as src:
        return src.read().astype(np.float32)          # shape (B, H, W)

def best_single_or_pair(bands):
    """Pick the band (or 2-band mean) with the lowest single-Otsu score."""
```

```
        B, H, W = bands.shape
        def score(img):
            t = threshold_otsu(img)
            fg, bg = img[img>t], img[img<=t]
            return np.inf if fg.size==0 or bg.size==0 else np.var(fg)+np.var(bg)
        best_img, best_idxs, best_score = None, None, np.inf
        for i in range(B):
            img = cv2.normalize(bands[i], None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
            s   = score(img)
            if s < best_score: best_img, best_idxs, best_score = img, [i], s
        for i,j in combinations(range(B),2):
            img = cv2.normalize(np.mean(bands[[i,j]],axis=0), None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
            s   = score(img)
            if s < best_score: best_img, best_idxs, best_score = img, [i,j], s
        return best_img, best_idxs

    def segment_three(img):
        """multi-Otsu → 3 labels (0,1,2)"""
        t1,t2 = threshold_multiotsu(img, classes=3)
        return np.digitize(img, bins=(t1,t2))             # shape (H, W)

    def segment_two(img):
        """single Otsu → 2 labels (0,1)"""
        t = threshold_otsu(img)
        return (img > t).astype(np.uint8)                 # shape (H, W)

    def show_rgb(mask, palette, title):
        h,w = mask.shape
        rgb = np.zeros((h,w,3), np.uint8)
        for k,c in palette.items():
            rgb[mask==k] = c
        plt.figure(figsize=(6,6)); plt.imshow(rgb); plt.title(title); plt.axis('off'); plt.show()

    def spectral_indices(stack):
        """Return 3×H×W array with NDVI, NDWI, MSAVI."""
        red, nir  = stack[RED_BAND], stack[NIR_BAND]
        swir      = stack[NIR_BAND-1]                    # crude pick for SWIR
        eps       = 1e-6
        ndvi  = (nir  - red)  / (nir  + red + eps)
        ndwi  = (nir  - swir) / (nir  + swir + eps)
        msavi = (2*nir + 1 - np.sqrt((2*nir + 1)**2 - 8*(nir - red))) / 2
        return np.stack([ndvi, ndwi, msavi])             # shape (3, H, W)

    # ------------------------------------------------------------------
    # 3.  Load all stacks
    # ------------------------------------------------------------------
    stacks = {name: load_stack(path) for name,path in FILES.items()}
    B, H, W = next(iter(stacks.values())).shape           # assume all equal size

    # ------------------------------------------------------------------
    # 4.  ICA (fit on 1 % random pixels from ALL images)
    # ------------------------------------------------------------------
    sample_px = []
    rng = np.random.default_rng(0)
    for s in stacks.values():
        flat = s.reshape(B, -1).T                        # (Npx, B)
        idx  = rng.choice(flat.shape[0], int(0.01*flat.shape[0]), replace=False)
        sample_px.append(flat[idx])
    sample_px = np.vstack(sample_px)
    sample_px = sample_px[~np.isnan(sample_px).any(axis=1)]  # remove rows with NaNs

    ica = FastICA(n_components=1, random_state=0, whiten='unit-variance')
    ica.fit(sample_px)


    # helper to get ICA-1 image
    def ica_image(stack):
        flat = stack.reshape(stack.shape[0], -1).T  # (Npx, B)
        valid_mask = ~np.isnan(flat).any(axis=1)
        comp = np.zeros(flat.shape[0], dtype=np.float32)
        comp[valid_mask] = ica.transform(flat[valid_mask])[:, 0]
        h, w = stack.shape[1:]
        return comp.reshape(h, w)

    # ------------------------------------------------------------------
    # 5.  SEGMENT + VISUALISE
    # ------------------------------------------------------------------
    masks = {}
    for name, stack in stacks.items():
        img_best, idxs = best_single_or_pair(stack)
        if name == 'pond5':                     # 3-class
            mask = segment_three(img_best)
```

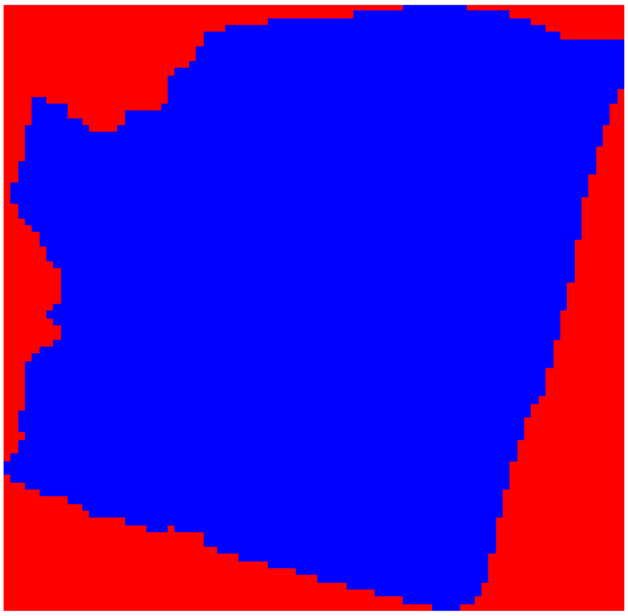
```
        show_rgb(mask, COLS_3, f'{name} – 3 classes (o/i/v)')
    else:                                   # 2-class
        mask = segment_two(img_best)        # 0,1  (we'll reuse 0=o,1=i)
        show_rgb(mask, COLS_2, f'{name} – 2 classes (o/i)')
    masks[name] = mask

# ------------------------------------------------------------------
# 6.  BUILD DATASET
# ------------------------------------------------------------------
dfs = []
for name, stack in stacks.items():
    mask          = masks[name]
    idx_flat      = mask.ravel()
    bands_flat    = stack.reshape(B, -1).T          # (Npx, 10)
    idx_flat_ica  = ica_image(stack).ravel()[:,None]
    idx_flat_idx3 = spectral_indices(stack).reshape(3, -1).T  # (Npx, 3)

    # Assemble DataFrame
    df = pd.DataFrame(bands_flat, columns=[f'band_{i+1}' for i in range(B)])
    df[['ndvi','ndwi','msavi']] = idx_flat_idx3
    df['ica1'] = idx_flat_ica
    # Map label ints → chars
    if name == 'pond5':                     # 3-way
        mapping = {0:'o', 1:'i', 2:'v'}
    else:
        mapping = {0:'o', 1:'i'}
    df['target'] = pd.Series(idx_flat).map(mapping)
    dfs.append(df)

dataset = pd.concat(dfs, ignore_index=True)
print(dataset.head())
print('Dataset shape:', dataset.shape)

# ------------------------------------------------------------------
# 7.  SAVE
# ------------------------------------------------------------------
dataset.to_parquet('pond_multispectral_dataset.parquet', index=False)
print('Saved → pond_multispectral_dataset.parquet')
```

pond11 – 2 classes (o/i)



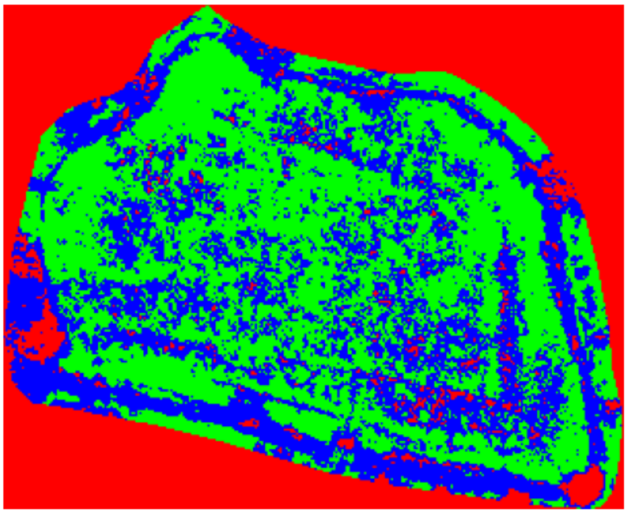
```
<ipython-input-75-3710634456>:52: RuntimeWarning: invalid value encountered in cast
  img = cv2.normalize(np.mean(bands[[i,j]],axis=0), None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
```

pond3 – 2 classes (o/i)



```
<ipython-input-75-3710634456>:48: RuntimeWarning: invalid value encountered in cast
  img = cv2.normalize(bands[i], None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
```

pond5 – 3 classes (o/i/v)



	band_1	band_2	band_3	band_4	band_5	band_6	band_7	band_8	band_9	\
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

	band_10	ndvi	ndwi	msavi	ica1	target
0	NaN	NaN	NaN	NaN	0.0	o
1	NaN	NaN	NaN	NaN	0.0	o
2	NaN	NaN	NaN	NaN	0.0	o

```
3      NaN   NaN   NaN    NaN   0.0      o
4      NaN   NaN   NaN    NaN   0.0      o
Dataset shape: (84903, 15)
Saved → pond_multispectral_dataset.parquet
```

```
# List of feature columns to fill NaNs in
features = [f'band_{i}' for i in range(1, 11)] + ['ndvi', 'ndwi', 'msavi', 'ica1']

# Create a copy of the original DataFrame to modify
df_filled = df.copy()

# For each column, compute the mean of rows where target == 'o'
for col in features:
    mean_val = df_filled.loc[df_filled['target'] == 'o', col].mean()
    df_filled.loc[(df_filled['target'] == 'o') & (df_filled[col].isna()), col] = mean_val
df_filled
```

	band_1	band_2	band_3	band_4	band_5	band_6	band_7	band_8	band_9	band_10	ndvi	ndwi	msavi	i
0	0.018978	0.020621	0.031033	0.036395	0.026126	0.025382	0.052075	0.074138	0.112891	0.144678	0.295267	0.148988	0.068745	
1	0.018978	0.020621	0.031033	0.036395	0.026126	0.025382	0.052075	0.074138	0.112891	0.144678	0.295267	0.148988	0.068745	
2	0.018978	0.020621	0.031033	0.036395	0.026126	0.025382	0.052075	0.074138	0.112891	0.144678	0.295267	0.148988	0.068745	
3	0.018978	0.020621	0.031033	0.036395	0.026126	0.025382	0.052075	0.074138	0.112891	0.144678	0.295267	0.148988	0.068745	
4	0.018978	0.020621	0.031033	0.036395	0.026126	0.025382	0.052075	0.074138	0.112891	0.144678	0.295267	0.148988	0.068745	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
73740	0.018978	0.020621	0.031033	0.036395	0.026126	0.025382	0.052075	0.074138	0.112891	0.144678	0.295267	0.148988	0.068745	
73741	0.018978	0.020621	0.031033	0.036395	0.026126	0.025382	0.052075	0.074138	0.112891	0.144678	0.295267	0.148988	0.068745	
73742	0.018978	0.020621	0.031033	0.036395	0.026126	0.025382	0.052075	0.074138	0.112891	0.144678	0.295267	0.148988	0.068745	
73743	0.018978	0.020621	0.031033	0.036395	0.026126	0.025382	0.052075	0.074138	0.112891	0.144678	0.295267	0.148988	0.068745	
73744	0.018978	0.020621	0.031033	0.036395	0.026126	0.025382	0.052075	0.074138	0.112891	0.144678	0.295267	0.148988	0.068745	

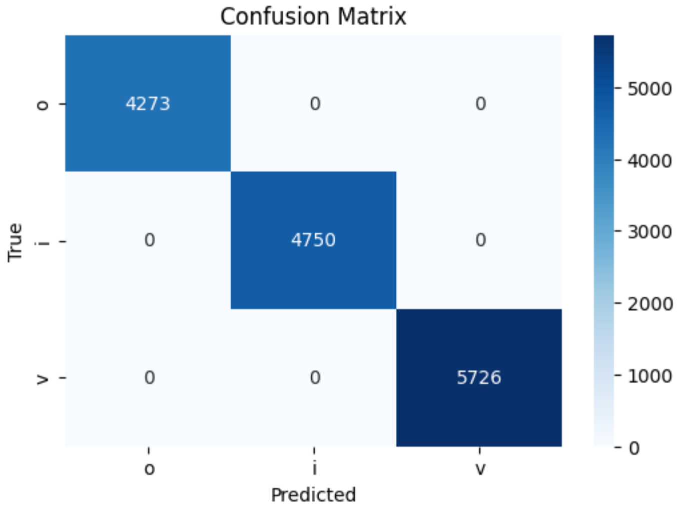
73745 rows × 15 columns

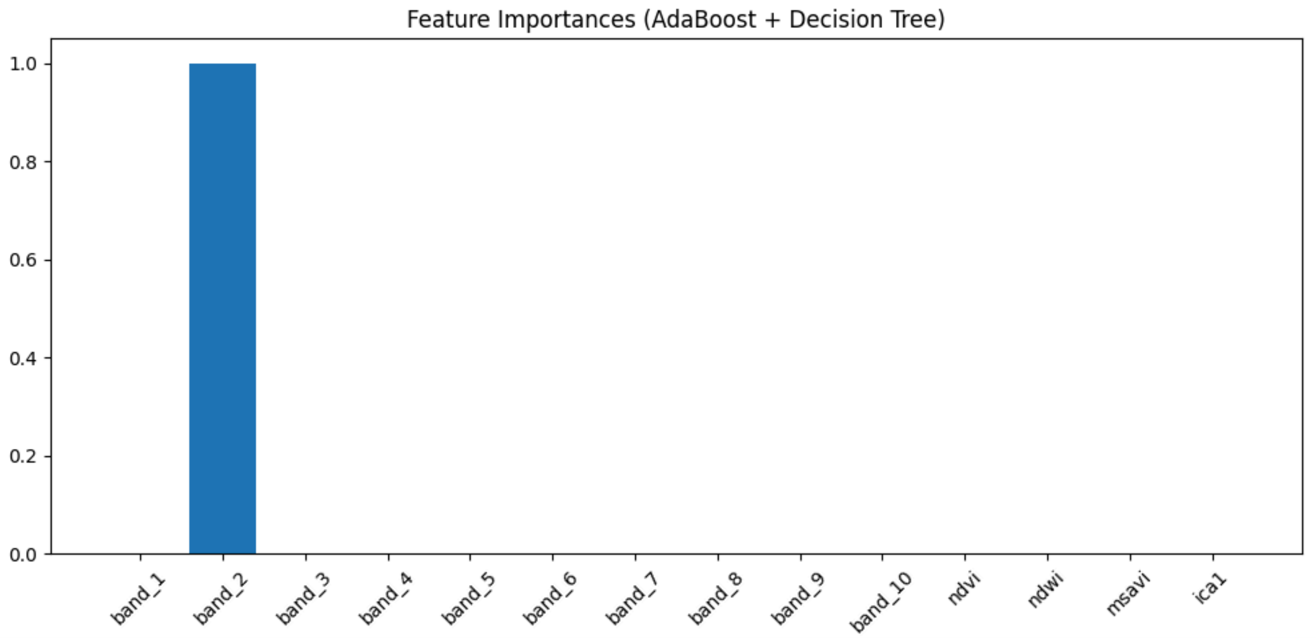
Next steps: Generate code with df_filled | View recommended plots | New interactive sheet

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
df_all=df_filled.copy()
# Define features and target
feature_cols = [f'band_{i+1}' for i in range(10)] + ['ndvi', 'ndwi', 'msavi', 'ica1']
X = df_all[feature_cols].values
y = df_all['target'].values  # 'o', 'i', 'v'

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# AdaBoost classifier with Decision Tree as base
base_tree = DecisionTreeClassifier(max_depth=3, random_state=42)
ada = AdaBoostClassifier(
    estimator=base_tree,
    n_estimators=100,
    learning_rate=1.0,
    algorithm='SAMME',  # Enables multi-class classification
    random_state=42
)

# Train
ada.fit(X_train, y_train)

# Evaluate
y_pred = ada.predict(X_test)
print("Classification Report:\n", classification_report(y_test, y_pred))

# Confusion Matrix
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues',
            xticklabels=['o', 'i', 'v'], yticklabels=['o', 'i', 'v'])
```

```
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()

# Feature Importance
importances = ada.feature_importances_
plt.figure(figsize=(10, 5))
plt.bar(range(len(importances)), importances, tick_label=feature_cols)
plt.xticks(rotation=45)
plt.title("Feature Importances (AdaBoost + Decision Tree)")
plt.tight_layout()
plt.show()
```

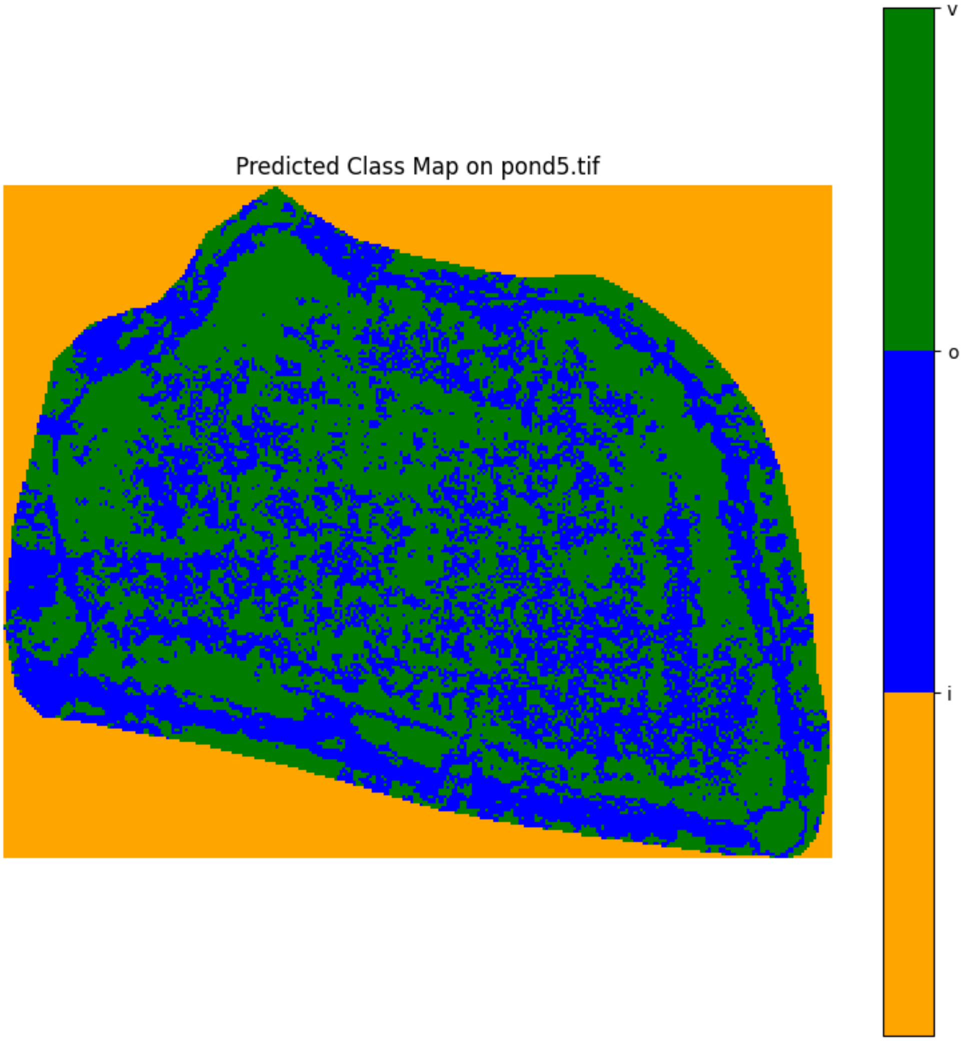
```
Classification Report:
              precision    recall  f1-score   support

           i       1.00      1.00      1.00      4273
           o       1.00      1.00      1.00      4750
           v       1.00      1.00      1.00      5726

    accuracy                           1.00     14749
   macro avg       1.00      1.00      1.00     14749
weighted avg       1.00      1.00      1.00     14749
```






```
import rasterio
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# === 1. Read TIF ===
def read_tif(path):
    with rasterio.open(path) as src:
```

```
        img = src.read()  # (bands, height, width)
    return img

pond5_img = read_tif("/content/pond5_stacked_05m.tif")  # update path if needed
bands, height, width = pond5_img.shape

# === 2. Reshape image to (H*W, bands) ===
X_img = pond5_img.reshape(bands, -1).T  # shape = (H*W, bands)

# === 3. Compute NDVI, NDWI, MSAVI, ICA1 ===
def compute_indices(X):
    red = X[:, 3]   # band_4
    nir = X[:, 4]   # band_5
    green = X[:, 2] # band_3
    ndvi = (nir - red) / (nir + red + 1e-6)
    ndwi = (green - nir) / (green + nir + 1e-6)
    msavi = (2 * nir + 1 - np.sqrt((2 * nir + 1)**2 - 8 * (nir - red))) / 2
    ica1 = X[:, 0] - X[:, 1]  # Simplified ICA1
    return np.column_stack((ndvi, ndwi, msavi, ica1))

indices = compute_indices(X_img)
X_img_full = np.hstack((X_img[:, :10], indices))  # total 14 features

# === 4. Remove NaNs and Predict ===
valid_mask = ~np.isnan(X_img_full).any(axis=1)
X_valid = X_img_full[valid_mask]
y_pred = ada.predict(X_valid)

# === 5. Create full label array with predictions ===
full_preds = np.full(X_img_full.shape[0], 'unknown', dtype=object)
full_preds[valid_mask] = y_pred
pred_labels = full_preds.reshape(height, width)

# === 6. Convert labels to integers for imshow ===
label_to_index = {label: idx for idx, label in enumerate(ada.classes_)}
index_to_label = {idx: label for label, idx in label_to_index.items()}

# Integer map for display
int_pred_image = np.full((height, width), -1)
for label, idx in label_to_index.items():
    int_pred_image[pred_labels == label] = idx

# === 7. Define colormap and plot ===
label_to_color = {'o': 'blue', 'v': 'green', 'i': 'orange'}
cmap = ListedColormap([label_to_color[label] for label in ada.classes_])

plt.figure(figsize=(10, 10))
plt.imshow(int_pred_image, cmap=cmap, interpolation='nearest')
cbar = plt.colorbar(ticks=range(len(ada.classes_)))
cbar.ax.set_yticklabels(ada.classes_)
plt.title("Predicted Class Map on pond5.tif")
plt.axis('off')
plt.show()

# === 8. Save prediction results as DataFrame ===
feature_cols = [f'band_{i}' for i in range(1, 11)] + ['ndvi', 'ndwi', 'msavi', 'ica1']
X_valid_full = X_img_full[valid_mask]
predicted_targets = y_pred

df_finaltest = pd.DataFrame(X_valid_full, columns=feature_cols)
df_finaltest['predicted_target'] = predicted_targets
df_finaltest['target_index'] = df_finaltest['predicted_target'].map(label_to_index)

# === 9. Preview and (optional) Save ===
print(df_finaltest.head())
print("✅ Final shape:", df_finaltest.shape)
print("Target distribution:\n", df_finaltest['predicted_target'].value_counts())

# Optional: Save to CSV
# df_finaltest.to_csv("pond5_predictions.csv", index=False)
```




```
       band_1    band_2    band_3    band_4    band_5    band_6    band_7  \
0  0.055036  0.067229  0.086199  0.102831  0.144363  0.146924  0.181491
1  0.059119  0.071893  0.090143  0.112069  0.172246  0.176670  0.204375
2  0.046861  0.056922  0.076444  0.088725  0.101295  0.102353  0.150220
3  0.046303  0.056298  0.074399  0.086562  0.101043  0.102837  0.147016
4  0.043621  0.052772  0.070700  0.082634  0.096481  0.097757  0.139786

     band_8    band_9   band_10      ndvi      ndwi     msavi      ica1  \
0  0.200284  0.235005  0.266785  0.168013 -0.252270  0.068047 -0.012193
1  0.216358  0.237850  0.254103  0.211655 -0.312902  0.096433 -0.012773
2  0.180934  0.235593  0.279309  0.066151 -0.139816  0.021282 -0.010062
3  0.174896  0.225639  0.270441  0.077191 -0.151868  0.024597 -0.009995
4  0.166395  0.214040  0.253529  0.077308 -0.154206  0.023685 -0.009151

  predicted_target  target_index
0                v             2
1                v             2
2                v             2
3                v             2
4                v             2
✅ Final shape: (52303, 16)
Target distribution:
 predicted_target
v    28629
i    21364
o     2310
Name: count, dtype: int64
```

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.metrics.pairwise import euclidean_distances

# Step 0: Copy original full dataset
df_main_full = df_filled.copy()

# === 1. Filter only 'i' target rows ===
df_main = df_main_full[df_main_full['target'] == 'o'].reset_index(drop=True)

# === 2. Load and combine turbidity CSVs ===
csv_paths = ["turb_11.csv", "pond_5.csv", "pond3_turb.csv"]
df_turb = pd.concat([pd.read_csv(p) for p in csv_paths], ignore_index=True)
```

```
# === 3. Rename B1-B10 to band_1 to band_10 ===
rename_dict = {f'B{i}': f'band_{i}' for i in range(1, 11)}
df_turb.rename(columns=rename_dict, inplace=True)

# === 4. Check required columns ===
band_cols = [f'band_{i}' for i in range(1, 11)]
assert all(col in df_turb.columns for col in band_cols + ['content'])

# === 5. Normalize ONLY turbidity dataset ===
scaler = StandardScaler()
X_turb_scaled = scaler.fit_transform(df_turb[band_cols])  # normalized turbidity bands
X_main = df_main[band_cols].values  # original, unnormalized band values

# === 6. Normalize turbidity features to match with raw df_main ===
X_main_scaled = scaler.transform(X_main)  # transformed into same scale as turbidity

# === 7. Compute distance matrix ===
dist_matrix = euclidean_distances(X_turb_scaled, X_main_scaled)

# === 8. Greedy one-to-one matching ===
assigned_main_indices = set()
matched_rows = []

for i in range(len(df_turb)):
    dists = dist_matrix[i]

    # Mask already-used indices
    dists[list(assigned_main_indices)] = np.inf

    min_index = np.argmin(dists)
    if dists[min_index] != np.inf:
        assigned_main_indices.add(min_index)

        # ✅ Keep original row with all columns
        matched_row = df_main.iloc[min_index].copy()

        # ✅ Add turbidity value from CSV
        matched_row['turbidity'] = df_turb.iloc[i]['content']

        matched_rows.append(matched_row)

# === 9. Final matched DataFrame ===
df_matches = pd.DataFrame(matched_rows)

print("Matched turbidity samples:", df_matches.shape[0])
print(df_matches.head())
```

```
Matched turbidity samples: 69
         band_1    band_2    band_3    band_4    band_5    band_6    band_7  \
17748  0.020848  0.023737  0.029142  0.036738  0.042878  0.041629  0.046649
17747  0.021633  0.023953  0.028846  0.035299  0.042694  0.041408  0.047271
8730   0.021757  0.023940  0.030792  0.037833  0.039818  0.037202  0.044095
17749  0.020093  0.023008  0.029246  0.036593  0.041144  0.039452  0.045202
17873  0.020895  0.023526  0.029989  0.037947  0.045114  0.043655  0.052645

         band_8    band_9   band_10      ndvi      ndwi     msavi      ica1  \
17748  0.045648  0.029580  0.039977  0.108150 -0.010843  0.016581  0.311732
17747  0.045937  0.031642  0.041377  0.130947 -0.014309  0.019846  0.311732
8730   0.042844  0.029444  0.048524  0.062114 -0.014395  0.009311  0.311733
17749  0.044794  0.029658  0.041261  0.100758 -0.004532  0.015266  0.311733
17873  0.053248  0.040488  0.050371  0.167778  0.005696  0.028385  0.311728

      target  turbidity
17748      o      49.44
17747      o      69.22
8730       o      44.83
17749      o      37.11
17873      o      34.48
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but Star
  warnings.warn(
```

```
# Raster and remote sensing
!pip install rasterio

# Image processing
!pip install opencv-python scikit-image

# Machine learning
!pip install scikit-learn

# Dataframe and storage
```

```
!pip install pandas pyarrow  # pyarrow is needed to save parquet files
```

```
Collecting rasterio
  Downloading rasterio-1.4.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.1 kB)
Collecting affine (from rasterio)
  Downloading affine-2.4.0-py3-none-any.whl.metadata (4.0 kB)
Requirement already satisfied: attrs in /usr/local/lib/python3.11/dist-packages (from rasterio) (25.3.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from rasterio) (2025.4.26)
Requirement already satisfied: click>=4.0 in /usr/local/lib/python3.11/dist-packages (from rasterio) (8.2.1)
Collecting cligj>=0.5 (from rasterio)
  Downloading cligj-0.7.2-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: numpy>=1.24 in /usr/local/lib/python3.11/dist-packages (from rasterio) (2.0.2)
Collecting click-plugins (from rasterio)
  Downloading click_plugins-1.1.1-py2.py3-none-any.whl.metadata (6.4 kB)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.11/dist-packages (from rasterio) (3.2.3)
Downloading rasterio-1.4.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (22.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 22.2/22.2 MB 26.6 MB/s eta 0:00:00
Downloading cligj-0.7.2-py3-none-any.whl (7.1 kB)
Downloading affine-2.4.0-py3-none-any.whl (15 kB)
Downloading click_plugins-1.1.1-py2.py3-none-any.whl (7.5 kB)
Installing collected packages: cligj, click-plugins, affine, rasterio
Successfully installed affine-2.4.0 click-plugins-1.1.1 cligj-0.7.2 rasterio-1.4.3
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Requirement already satisfied: scikit-image in /usr/local/lib/python3.11/dist-packages (0.25.2)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.11/dist-packages (from opencv-python) (2.0.2)
Requirement already satisfied: scipy>=1.11.4 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (1.15.3)
Requirement already satisfied: networkx>=3.0 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (3.5)
Requirement already satisfied: pillow>=10.1 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (11.2.1)
Requirement already satisfied: imageio!=2.35.0,>=2.33 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (2.37.0)
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (2025.6.1)
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (24.2)
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.11/dist-packages (from scikit-image) (0.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.15.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: pyarrow in /usr/local/lib/python3.11/dist-packages (18.1.0)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

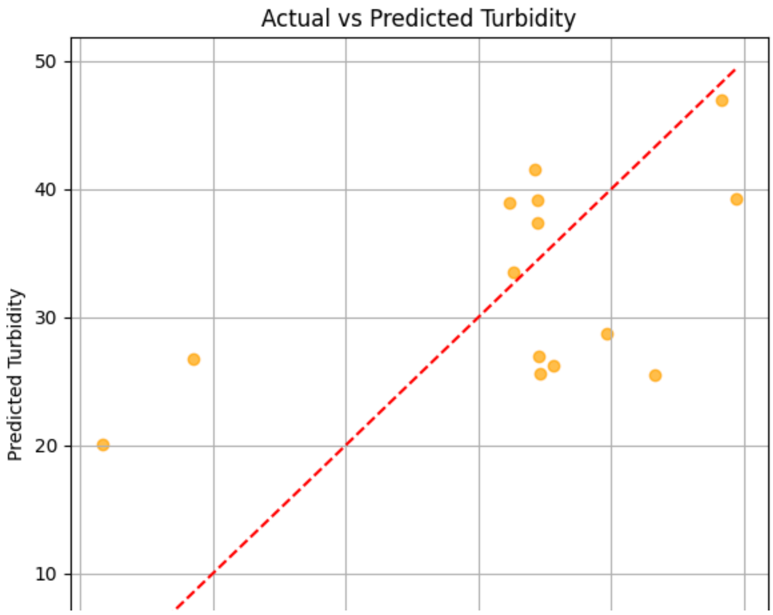
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
from scipy.spatial import cKDTree
from scipy.ndimage import uniform_filter

# === 1. Prepare Data ===
df = df_matches.copy()

# Ensure only numeric columns
df = df.select_dtypes(include=[np.number])

# Features and target
feature_cols = [col for col in df.columns if col != 'turbidity']
X = df[feature_cols].values
y = df['turbidity'].values

# === 2. Train/Test Split ===
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# === 3. Scaling ===
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# === 4. Model ===
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

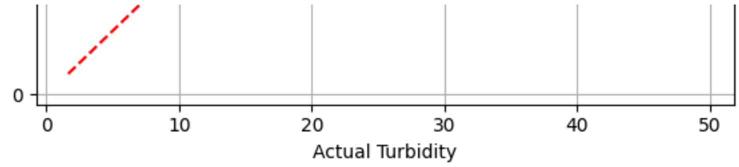
```
# === 5. Train ===
early_stop = EarlyStopping(patience=10, restore_best_weights=True)
model.fit(X_train_scaled, y_train,
          validation_split=0.2,
          epochs=100,
          batch_size=16,
          callbacks=[early_stop],
          verbose=1)

# === 6. Evaluate ===
y_pred = model.predict(X_test_scaled).flatten()
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
print(f"\n✅ RMSE: {rmse:.2f}")
print(f"✅ R² Score: {r2:.2f}")

# === 7. Plot Actual vs Predicted ===
plt.figure(figsize=(6, 6))
plt.scatter(y_test, y_pred, alpha=0.7, color='orange')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--')
plt.xlabel("Actual Turbidity")
plt.ylabel("Predicted Turbidity")
plt.title("Actual vs Predicted Turbidity")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Epoch 1/100
/usr/local/lib/python3.11/dist-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)
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 125ms/step - loss: 1645.8031 - mae: 34.7727 - val_loss: 2324.0813 - val_mae: 46.2151
Epoch 2/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 183ms/step - loss: 1360.8484 - mae: 30.4533 - val_loss: 2284.9187 - val_mae: 45.7851
Epoch 3/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - loss: 1348.8174 - mae: 30.7002 - val_loss: 2245.1157 - val_mae: 45.3436
Epoch 4/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 1364.1516 - mae: 30.8779 - val_loss: 2202.6404 - val_mae: 44.8669
Epoch 5/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 1356.6392 - mae: 30.5598 - val_loss: 2156.2974 - val_mae: 44.3415
Epoch 6/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 1425.5378 - mae: 31.2606 - val_loss: 2104.4031 - val_mae: 43.7430
Epoch 7/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - loss: 1244.0411 - mae: 28.9866 - val_loss: 2046.2571 - val_mae: 43.0621
Epoch 8/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 1163.7769 - mae: 28.5674 - val_loss: 1982.1854 - val_mae: 42.2963
Epoch 9/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step - loss: 1083.7002 - mae: 26.3700 - val_loss: 1910.5731 - val_mae: 41.4229
Epoch 10/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - loss: 933.9391 - mae: 24.5520 - val_loss: 1829.8915 - val_mae: 40.4132
Epoch 11/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - loss: 990.2904 - mae: 25.8546 - val_loss: 1738.6708 - val_mae: 39.2324
Epoch 12/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 1009.5316 - mae: 26.1920 - val_loss: 1638.7887 - val_mae: 37.8894
Epoch 13/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 895.0751 - mae: 24.4818 - val_loss: 1531.5724 - val_mae: 36.3800
Epoch 14/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 862.8043 - mae: 24.2354 - val_loss: 1417.4539 - val_mae: 34.6869
Epoch 15/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 776.0776 - mae: 23.0349 - val_loss: 1298.0126 - val_mae: 32.8007
Epoch 16/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 665.5249 - mae: 21.4240 - val_loss: 1176.1960 - val_mae: 30.7250
Epoch 17/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - loss: 571.3956 - mae: 20.0964 - val_loss: 1055.4900 - val_mae: 28.4809
Epoch 18/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 188ms/step - loss: 577.8799 - mae: 20.0135 - val_loss: 939.1425 - val_mae: 26.0624
Epoch 19/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 408.5050 - mae: 16.8373 - val_loss: 832.3201 - val_mae: 23.5533
Epoch 20/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 321.1229 - mae: 14.6834 - val_loss: 738.0323 - val_mae: 21.5254
Epoch 21/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 311.3287 - mae: 14.4489 - val_loss: 655.6818 - val_mae: 20.3895
Epoch 22/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - loss: 307.1066 - mae: 14.8666 - val_loss: 588.3413 - val_mae: 19.8069
Epoch 23/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 276.3326 - mae: 14.2131 - val_loss: 544.1807 - val_mae: 19.3246
Epoch 24/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 260.4209 - mae: 14.0729 - val_loss: 512.9642 - val_mae: 18.8167
Epoch 25/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 225.3896 - mae: 13.8114 - val_loss: 493.6674 - val_mae: 18.3864
Epoch 26/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 208.5513 - mae: 12.9392 - val_loss: 480.9887 - val_mae: 18.0211
Epoch 27/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 229.9738 - mae: 13.8197 - val_loss: 474.7039 - val_mae: 17.7821
Epoch 28/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 221.7210 - mae: 13.7175 - val_loss: 467.2331 - val_mae: 17.5506
Epoch 29/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 166ms/step - loss: 230.1257 - mae: 13.8127 - val_loss: 461.3074 - val_mae: 17.3931
Epoch 30/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - loss: 226.8496 - mae: 13.5241 - val_loss: 456.7155 - val_mae: 17.2859
Epoch 31/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 190.6105 - mae: 12.1236 - val_loss: 445.7677 - val_mae: 17.1056
Epoch 32/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - loss: 249.3001 - mae: 14.4548 - val_loss: 438.7461 - val_mae: 17.0105
Epoch 33/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 231.6146 - mae: 13.7597 - val_loss: 430.5535 - val_mae: 16.8618
Epoch 34/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 177.2030 - mae: 12.3098 - val_loss: 424.5123 - val_mae: 16.7231
Epoch 35/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 169ms/step - loss: 184.1577 - mae: 11.7655 - val_loss: 422.0305 - val_mae: 16.6580
Epoch 36/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - loss: 187.3906 - mae: 11.9074 - val_loss: 417.4289 - val_mae: 16.5353
Epoch 37/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 216.4276 - mae: 12.4363 - val_loss: 410.6857 - val_mae: 16.3535
Epoch 38/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 190.0291 - mae: 12.4759 - val_loss: 407.7015 - val_mae: 16.2677
Epoch 39/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 199.0084 - mae: 12.4459 - val_loss: 402.5987 - val_mae: 16.1266
Epoch 40/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 193.5599 - mae: 12.5052 - val_loss: 398.6683 - val_mae: 16.0156
Epoch 41/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 182.1917 - mae: 11.9113 - val_loss: 394.2201 - val_mae: 15.8908
Epoch 42/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - loss: 172.7359 - mae: 11.6857 - val_loss: 389.4951 - val_mae: 15.7535
Epoch 43/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 164.4454 - mae: 11.3825 - val_loss: 382.5678 - val_mae: 15.5508
Epoch 44/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - loss: 190.3598 - mae: 12.0765 - val_loss: 376.7264 - val_mae: 15.3685
```

```
Epoch 45/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 181.5854 - mae: 12.0052 - val_loss: 372.1881 - val_mae: 15.2313
Epoch 46/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 174ms/step - loss: 200.1119 - mae: 12.7209 - val_loss: 366.9830 - val_mae: 15.0761
Epoch 47/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 165.7929 - mae: 11.0254 - val_loss: 364.3857 - val_mae: 14.9932
Epoch 48/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - loss: 155.8406 - mae: 10.7734 - val_loss: 363.6604 - val_mae: 14.9578
Epoch 49/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 172.7213 - mae: 11.3601 - val_loss: 363.7667 - val_mae: 14.9442
Epoch 50/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 216.5154 - mae: 12.3498 - val_loss: 366.9114 - val_mae: 15.0099
Epoch 51/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 117ms/step - loss: 176.5723 - mae: 11.2485 - val_loss: 368.2789 - val_mae: 15.0277
Epoch 52/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 178.1436 - mae: 11.4251 - val_loss: 368.9259 - val_mae: 15.0404
Epoch 53/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 145.5611 - mae: 10.2948 - val_loss: 366.1796 - val_mae: 14.9579
Epoch 54/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step - loss: 148.7740 - mae: 10.6648 - val_loss: 362.8835 - val_mae: 14.8571
Epoch 55/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step - loss: 193.9157 - mae: 11.9168 - val_loss: 358.2071 - val_mae: 14.6968
Epoch 56/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - loss: 190.9662 - mae: 11.2507 - val_loss: 353.1261 - val_mae: 14.5438
Epoch 57/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 141.8661 - mae: 10.1684 - val_loss: 348.1436 - val_mae: 14.4232
Epoch 58/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step - loss: 158.5423 - mae: 10.6387 - val_loss: 345.2148 - val_mae: 14.3736
Epoch 59/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step - loss: 152.9201 - mae: 10.3151 - val_loss: 336.5758 - val_mae: 14.1973
Epoch 60/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 134ms/step - loss: 187.0785 - mae: 11.4044 - val_loss: 332.2128 - val_mae: 14.1093
Epoch 61/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step - loss: 146.2780 - mae: 10.2969 - val_loss: 325.9442 - val_mae: 13.9777
Epoch 62/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 174.0542 - mae: 11.1105 - val_loss: 321.6655 - val_mae: 13.8921
Epoch 63/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - loss: 184.7262 - mae: 11.0597 - val_loss: 315.5412 - val_mae: 13.7569
Epoch 64/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - loss: 162.2555 - mae: 10.5815 - val_loss: 310.5764 - val_mae: 13.6467
Epoch 65/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 172.4526 - mae: 10.8720 - val_loss: 311.3691 - val_mae: 13.6751
Epoch 66/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 135.9098 - mae: 9.7135 - val_loss: 311.4195 - val_mae: 13.6824
Epoch 67/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - loss: 153.1813 - mae: 10.0096 - val_loss: 315.9845 - val_mae: 13.7955
Epoch 68/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - loss: 125.3285 - mae: 9.0267 - val_loss: 320.3325 - val_mae: 13.9017
Epoch 69/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 152.4604 - mae: 10.0883 - val_loss: 324.2410 - val_mae: 13.9898
Epoch 70/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - loss: 181.4370 - mae: 11.4458 - val_loss: 325.5277 - val_mae: 14.0162
Epoch 71/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step - loss: 177.6291 - mae: 10.9042 - val_loss: 326.4589 - val_mae: 14.0350
Epoch 72/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 121ms/step - loss: 178.2179 - mae: 10.7069 - val_loss: 321.8370 - val_mae: 13.9343
Epoch 73/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 148ms/step - loss: 151.6266 - mae: 10.2235 - val_loss: 316.2453 - val_mae: 13.8137
Epoch 74/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 145.1352 - mae: 9.9450 - val_loss: 313.9365 - val_mae: 13.7660
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 134ms/step
```

✅ RMSE: 10.58
✅ R² Score: 0.31

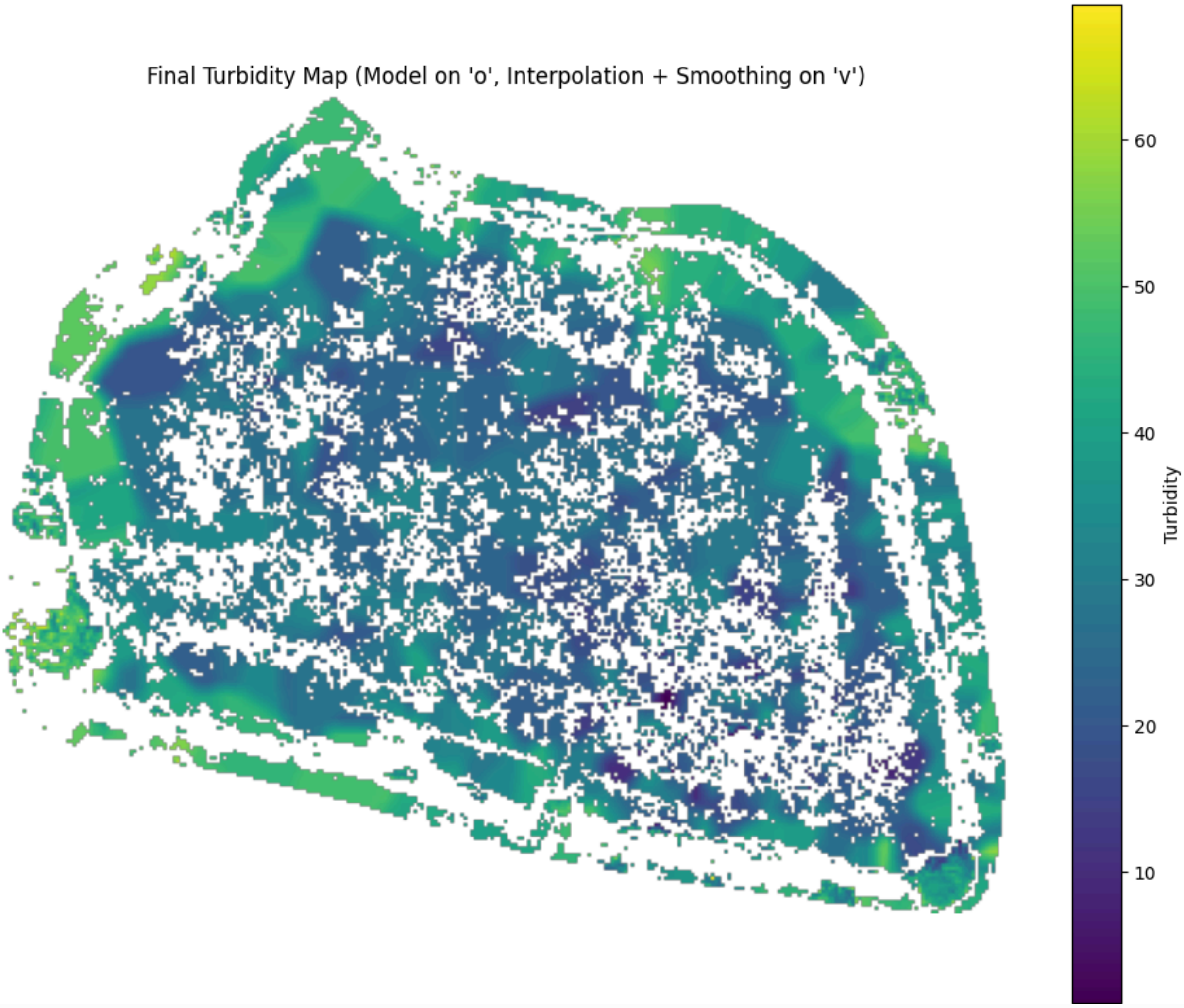



0
0
10
20
30
40
50
Actual Turbidity

```
# === 1. Setup ===
height, width = pred_labels.shape
n_pixels = height * width
feature_cols = [f'band_{i}' for i in range(1, 11)] + ['ndvi', 'ndwi', 'msavi', 'ica1']

# === 2. Predict on 'o' points ===
X_all = df_finaltest[feature_cols].values
X_all_scaled = scaler.transform(X_all)

o_mask = df_finaltest['predicted_target'] == 'o'
preds_raw = model.predict(X_all_scaled[o_mask]).flatten()

# === 3. Rescale predictions to y_train range ===
y_min, y_max = y_train.min(), y_train.max()
preds_min, preds_max = preds_raw.min(), preds_raw.max()

# avoid divide-by-zero if model gives flat output
if preds_max != preds_min:
    preds_scaled = (preds_raw - preds_min) / (preds_max - preds_min)  # scale to 0–1
    preds_rescaled = preds_scaled * (y_max - y_min) + y_min           # rescale to y range
else:
    preds_rescaled = np.full_like(preds_raw, y_min)  # all same if flat model output

# === 4. Build full prediction array ===
turbidity_preds_df = np.full(df_finaltest.shape[0], np.nan)
turbidity_preds_df[o_mask] = preds_rescaled

# === 5. Create flat turbidity array ===
valid_indices_flat = np.where(valid_mask.flatten())[0]
turbidity_flat = np.full(n_pixels, np.nan)
turbidity_flat[valid_indices_flat] = turbidity_preds_df

# === 6. Interpolate missing 'v' pixels ===
pred_labels_flat = pred_labels.flatten()
v_mask_flat = pred_labels_flat == 'v'
coords = np.column_stack(np.unravel_index(np.arange(n_pixels), (height, width)))
filled = turbidity_flat.copy()

from scipy.spatial import cKDTree

iteration = 0
max_iterations = 10
while np.any(np.isnan(filled[v_mask_flat])) and iteration < max_iterations:
    iteration += 1
    known_mask = ~np.isnan(filled)
    target_mask = v_mask_flat & np.isnan(filled)

    known_coords = coords[known_mask]
    known_vals = filled[known_mask]
    target_coords = coords[target_mask]

    tree = cKDTree(known_coords)
    _, nearest_idx = tree.query(target_coords)
    filled[target_mask] = known_vals[nearest_idx]

# === 7. Smoothing (optional, can skip for debugging) ===
from scipy.ndimage import uniform_filter

turbidity_map = filled.reshape(height, width)
v_mask_image = (pred_labels == 'v')

nan_mask = np.isnan(turbidity_map)
valid_img = (~nan_mask).astype(float)
vals_no_nan = np.where(nan_mask, 0, turbidity_map)

sum_vals = uniform_filter(vals_no_nan, size=5)
sum_counts = uniform_filter(valid_img, size=5)
smoothed_all = sum_vals / np.maximum(sum_counts, 1e-6)

# write smoothed values into only 'v' pixels
turbidity_map[v_mask_image] = smoothed_all[v_mask_image]
filled = turbidity_map.flatten()

# === 8. Final prediction DataFrame ===
rows, cols = np.unravel_index(np.arange(n_pixels), (height, width))
df_all = pd.DataFrame({
    'row': rows,
    'col': cols,
    'target': pred_labels_flat,
    'predicted_turbidity': filled
})
df_turbidity_pred = df_all[df_all['target'].isin(['o', 'v'])].reset_index(drop=True)
```

```
print(df_turbidity_pred.head())
print("✅ Final prediction table shape:", df_turbidity_pred.shape)
print("Target counts:\n", df_turbidity_pred['target'].value_counts())
print("NaNs in output:", df_turbidity_pred['predicted_turbidity'].isna().sum())

# === 9. Final Map Plot ===
plt.figure(figsize=(10, 8))
plt.imshow(turbidity_map, cmap='viridis')
plt.colorbar(label="Turbidity")
plt.title("Final Turbidity Map (Model on 'o', Interpolation + Smoothing on 'v')")
plt.axis('off')
plt.tight_layout()
plt.show()
```

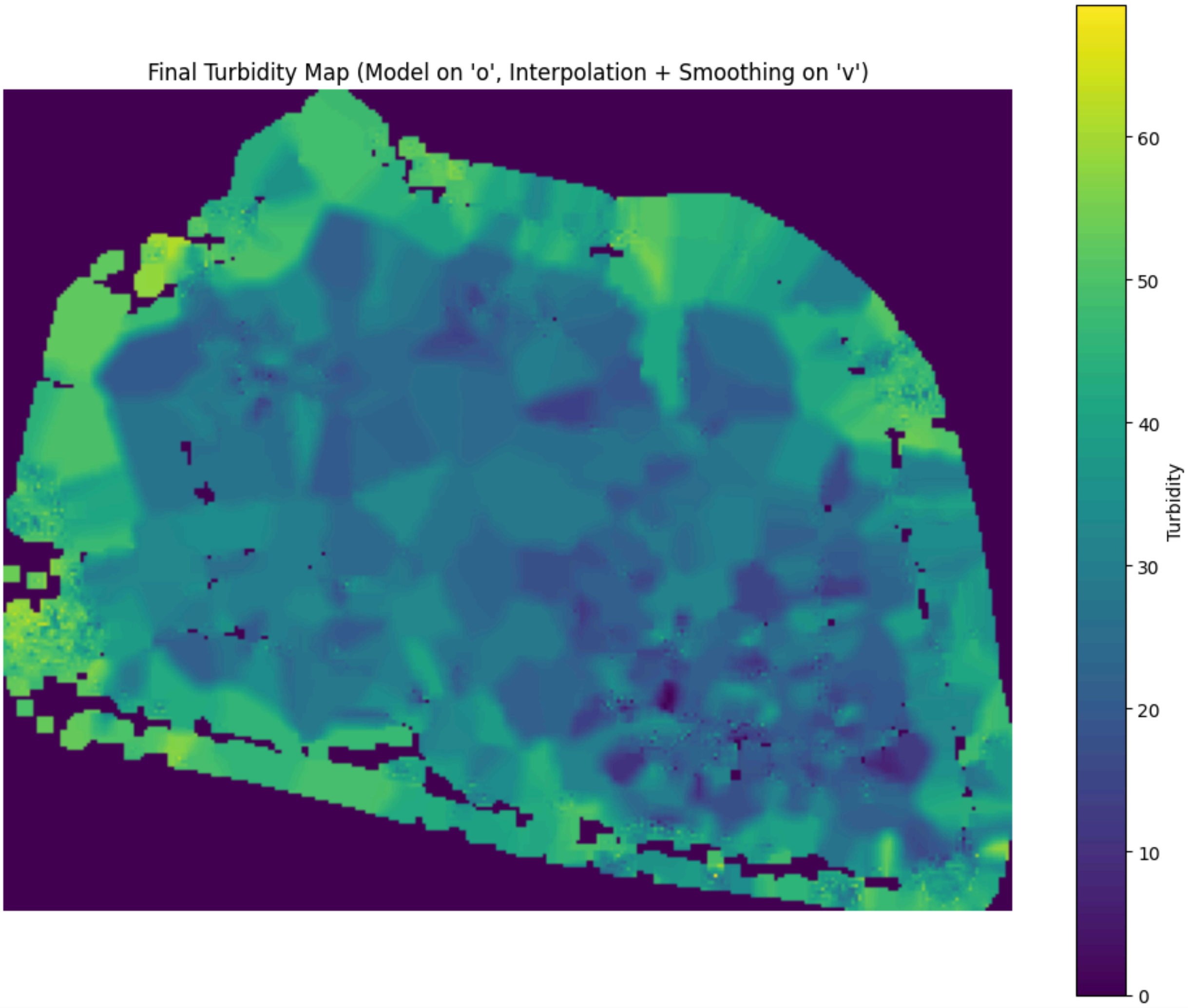
```
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
   row  col target  predicted_turbidity
0    1   98      v            48.095191
1    1   99      v            48.095191
2    2   96      v            48.107997
3    2   97      v            48.100393
4    2   98      v            48.095191
✅ Final prediction table shape: (30939, 4)
Target counts:
 target
v    28629
o     2310
Name: count, dtype: int64
NaNs in output: 0
```




```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.spatial import cKDTree
from scipy.ndimage import uniform_filter

# === 1. Setup ===
height, width = pred_labels.shape
n_pixels = height * width
feature_cols = [f'band_{i}' for i in range(1, 11)] + ['ndvi', 'ndwi', 'msavi', 'ica1']
```

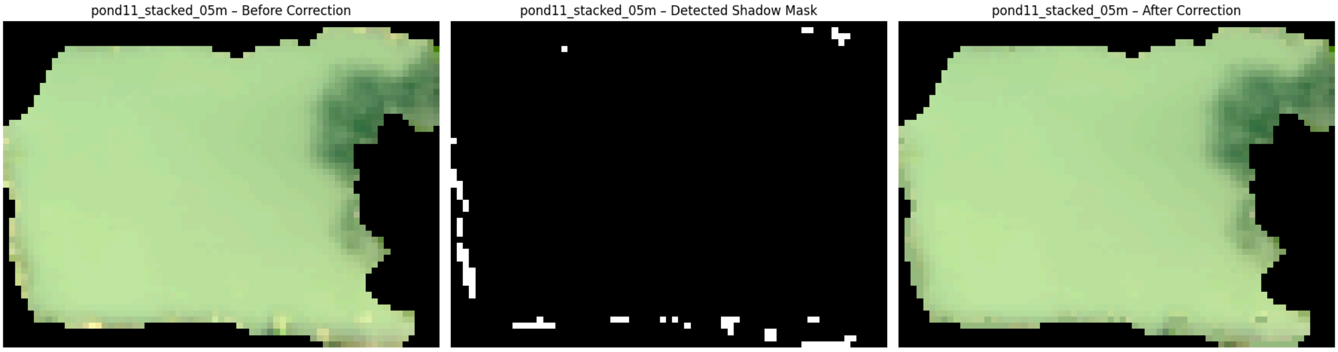
```
# === 2. Predict on 'o' points ===
X_all = df_finaltest[feature_cols].values
X_all_scaled = scaler.transform(X_all)

o_mask = df_finaltest['predicted_target'] == 'o'
preds_raw = model.predict(X_all_scaled[o_mask]).flatten()

# === 3. Rescale predictions to y_train range ===
y_min, y_max = y_train.min(), y_train.max()
preds_min, preds_max = preds_raw.min(), preds_raw.max()

if preds_max != preds_min:
    preds_scaled = (preds_raw - preds_min) / (preds_max - preds_min)
    preds_rescaled = preds_scaled * (y_max - y_min) + y_min
else:
    preds_rescaled = np.full_like(preds_raw, y_min)

# === 4. Create full turbidity array ===
turbidity_flat = np.full(n_pixels, np.nan)
pred_labels_flat = pred_labels.flatten()

# Fill 'o' pixels directly
o_indices = np.where(pred_labels_flat == 'o')[0]
turbidity_flat[o_indices] = preds_rescaled

# === 5. Interpolate missing 'v' pixels ===
v_indices = np.where(pred_labels_flat == 'v')[0]
coords = np.column_stack(np.unravel_index(np.arange(n_pixels), (height, width)))
filled = turbidity_flat.copy()

iteration = 0
max_iterations = 10
while np.any(np.isnan(filled[v_indices])) and iteration < max_iterations:
    iteration += 1
    known_mask = ~np.isnan(filled)
    target_mask = pred_labels_flat == 'v'
    target_mask &= np.isnan(filled)

    known_coords = coords[known_mask]
    known_vals = filled[known_mask]
    target_coords = coords[target_mask]

    tree = cKDTree(known_coords)
    _, nearest_idx = tree.query(target_coords)
    filled[target_mask] = known_vals[nearest_idx]

# === 6. Smoothing ===
turbidity_map = filled.reshape(height, width)
v_mask_image = (pred_labels == 'v')

nan_mask = np.isnan(turbidity_map)
valid_img = (~nan_mask).astype(float)
vals_no_nan = np.where(nan_mask, 0, turbidity_map)

sum_vals = uniform_filter(vals_no_nan, size=5)
sum_counts = uniform_filter(valid_img, size=5)
smoothed_all = sum_vals / np.maximum(sum_counts, 1e-6)

# Fill 'v' pixels and remaining NaNs with smoothed values
turbidity_map[v_mask_image] = smoothed_all[v_mask_image]
turbidity_map[nan_mask] = smoothed_all[nan_mask]

# === 7. Final Map Plot ===
plt.figure(figsize=(10, 8))
plt.imshow(turbidity_map, cmap='viridis')
plt.colorbar(label="Turbidity")
plt.title("Final Turbidity Map (Model on 'o', Interpolation + Smoothing on 'v')")
plt.axis('off')
plt.tight_layout()
plt.show()
```

73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step




```
import numpy as np
import rasterio
import matplotlib.pyplot as plt
import cv2
import os
from skimage.morphology import footprint_rectangle, opening, closing

def remove_shadows_lsi_micasense(tif_path, struct_elem_size=1):
    def rgb_to_h_i(rgb_img):
        hsv = cv2.cvtColor(rgb_img, cv2.COLOR_RGB2HSV)
        h = hsv[:, :, 0].astype(np.float32) / 180
        i = np.mean(rgb_img.astype(np.float32), axis=2) / 255.0
        return h, i

    with rasterio.open(tif_path) as src:
        img = src.read(masked=True).filled(np.nan)  # Convert MaskedArray to writable float array
        profile = src.profile.copy()

    # Correct band mapping
    B = img[1]   # 475 nm
    G = img[3]   # 560 nm
    R = img[5]   # 668 nm
    NIR = img[9] # 842 nm

    rgb = np.stack([R, G, B], axis=-1).astype(np.float32)
    rgb_norm = (rgb - np.nanmin(rgb)) / (np.nanmax(rgb) - np.nanmin(rgb) + 1e-6)
    rgb_norm[np.isnan(rgb_norm)] = 0  # Replace NaNs for safety

    h, i = rgb_to_h_i((rgb_norm * 255).clip(0, 255).astype(np.uint8))
    nir_norm = (NIR - np.nanmin(NIR)) / (np.nanmax(NIR) - np.nanmin(NIR) + 1e-6)
    nir_norm[np.isnan(nir_norm)] = 0

    isi = nir_norm * ((i - h) / (i + h + 1e-6))
    lsi = np.log1p(isi)

    lsi[np.isnan(lsi)] = 0
    lsi_8bit = cv2.normalize(lsi, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
```

```
    _, shadow_mask = cv2.threshold(lsi_8bit, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    shadow_mask = shadow_mask.astype(bool)

    from skimage.morphology import footprint_rectangle  # Already imported

    kernel = footprint_rectangle((struct_elem_size, struct_elem_size))

    shadow_mask = opening(shadow_mask, kernel)
    shadow_mask = closing(shadow_mask, kernel)

    shadow_free = rgb_norm.copy()
    for c in range(3):
        band = shadow_free[:, :, c].copy()  # avoid modifying read-only
        mean_val = np.nanmean(band[~shadow_mask])
        band[shadow_mask] = mean_val
        shadow_free[:, :, c] = band

    return rgb_norm, shadow_free, shadow_mask, img, profile

def visualize_and_save_corrected(tif_path, output_dir='/content/shadow_corrected'):
    os.makedirs(output_dir, exist_ok=True)
    label = os.path.splitext(os.path.basename(tif_path))[0]

    rgb_before, rgb_after, mask, img_all, profile = remove_shadows_lsi_micasense(tif_path)

    # === Visualization
    fig, axs = plt.subplots(1, 3, figsize=(18, 6))

    axs[0].imshow(np.clip(rgb_before ** 0.5, 0, 1))
    axs[0].set_title(f'{label} - Before Correction')
    axs[0].axis('off')

    axs[1].imshow(mask, cmap='gray')
    axs[1].set_title(f'{label} - Detected Shadow Mask')
    axs[1].axis('off')

    axs[2].imshow(np.clip(rgb_after ** 0.5, 0, 1))
    axs[2].set_title(f'{label} - After Correction')
    axs[2].axis('off')

    plt.tight_layout()
    plt.show()

    # === Save corrected image
    corrected_clean = np.nan_to_num(rgb_after, nan=0.0)
    corrected_uint16 = np.clip(corrected_clean * 65535, 0, 65535).astype(np.uint16)

    img_all[5] = corrected_uint16[:, :, 0]  # Red
    img_all[3] = corrected_uint16[:, :, 1]  # Green
    img_all[1] = corrected_uint16[:, :, 2]  # Blue

    profile.update({
        'count': 10,
        'dtype': rasterio.uint16,
        'nodata': 0
    })

    output_path = os.path.join(output_dir, f"{label}_shadow_removed.tif")
    with rasterio.open(output_path, 'w', **profile) as dst:
        dst.write(img_all)

    print(f"✅ Saved corrected image → {output_path}")

# === Example usage
visualize_and_save_corrected("/content/pond11_stacked_05m.tif")
```




```
✅ Saved corrected image → /content/shadow_corrected/pond11_stacked_05m_shadow_removed.tif
/usr/local/lib/python3.11/dist-packages/numpy/_core/_asarray.py:126: RuntimeWarning: invalid value encountered in cast
  arr = array(a, dtype=dtype, order=order, copy=None, subok=subok)
```

```
import rasterio
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# === 1. Read TIF ===
def read_tif(path):
    with rasterio.open(path) as src:
        img = src.read()  # (bands, height, width)
    return img

pond5_img = read_tif("/content/pond5_stacked_05m.tif")  # update path if needed
bands, height, width = pond5_img.shape

# === 2. Reshape image to (H*W, bands) ===
X_img = pond5_img.reshape(bands, -1).T  # shape = (H*W, bands)

# === 3. Compute NDVI, NDWI, MSAVI, ICA1 ===
def compute_indices(X):
    red = X[:, 3]   # band_4
    nir = X[:, 4]   # band_5
    green = X[:, 2] # band_3
    ndvi = (nir - red) / (nir + red + 1e-6)
    ndwi = (green - nir) / (green + nir + 1e-6)
    msavi = (2 * nir + 1 - np.sqrt((2 * nir + 1)**2 - 8 * (nir - red))) / 2
    ica1 = X[:, 0] - X[:, 1]  # Simplified ICA1
    return np.column_stack((ndvi, ndwi, msavi, ica1))

indices = compute_indices(X_img)
X_img_full = np.hstack((X_img[:, :10], indices))  # total 14 features

# === 4. Remove NaNs and Predict ===
valid_mask = ~np.isnan(X_img_full).any(axis=1)
X_valid = X_img_full[valid_mask]
y_pred = ada.predict(X_valid)

# === 5. Create full label array with predictions ===
full_preds = np.full(X_img_full.shape[0], 'unknown', dtype=object)
full_preds[valid_mask] = y_pred
pred_labels = full_preds.reshape(height, width)

# === 6. Convert labels to integers for imshow ===
label_to_index = {label: idx for idx, label in enumerate(ada.classes_)}
index_to_label = {idx: label for label, idx in label_to_index.items()}

# Integer map for display
int_pred_image = np.full((height, width), -1)
for label, idx in label_to_index.items():
    int_pred_image[pred_labels == label] = idx

# === 7. Define colormap and plot ===
label_to_color = {'o': 'blue', 'v': 'green', 'i': 'orange'}
cmap = ListedColormap([label_to_color[label] for label in ada.classes_])

plt.figure(figsize=(10, 10))
plt.imshow(int_pred_image, cmap=cmap, interpolation='nearest')
cbar = plt.colorbar(ticks=range(len(ada.classes_)))
cbar.ax.set_yticklabels(ada.classes_)
plt.title("Predicted Class Map on pond5.tif")
plt.axis('off')
plt.show()
```

```
# === 8. Save prediction results as DataFrame ===
feature_cols = [f'band_{i}' for i in range(1, 11)] + ['ndvi', 'ndwi', 'msavi', 'ica1']
X_valid_full = X_img_full[valid_mask]
predicted_targets = y_pred

df_finaltest_shadow = pd.DataFrame(X_valid_full, columns=feature_cols)
df_finaltest_shadow['predicted_target'] = predicted_targets
df_finaltest_shadow['target_index'] = df_finaltest['predicted_target'].map(label_to_index)

# === 9. Preview and (optional) Save ===
print(df_finaltest_shadow.head())
print("✅ Final shape:", df_finaltest_shadow.shape)
print("Target distribution:\n", df_finaltest_shadow['predicted_target'].value_counts())

# Optional: Save to CSV
# df_finaltest.to_csv("pond5_predictions.csv", index=False)
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.