Lab10 Image classification from embeddings

In [4]:
```
ID = 1631938
Name = 'MIN SOE HTUT'
```

Load the Data

In [5]:
```
import pandas as pd
import numpy as np
from google.colab import drive
drive.mount('/content/drive')
df=pd.read_csv('/content/drive/MyDrive/dataset/data.csv')
df_test=pd.read_csv('/content/drive/MyDrive/dataset/newdata.csv')
```

```
Mounted at /content/drive
```

Check the Data

In [6]:
```
# Check the shape and columns of the training data
print("Training Data:")
print(df.shape)
print(df.info())

# Check the first few rows of the data
print(df.head())

# Check the structure of the test data
print("Test Data (Unlabeled):")
print(df_test.shape)
print(df_test.info())
print(df_test.head())
```

```
Training Data:
(14034, 2050)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14034 entries, 0 to 14033
Columns: 2050 entries, a0 to target
dtypes: float64(2048), int64(1), object(1)
memory usage: 219.5+ MB
None
         a0        a1        a2        a3        a4        a5        a6  \
0  0.987159 -0.629449 -0.006023 -0.347239  1.676458 -0.322336  0.295421
1 -0.621512 -0.486168 -0.627894 -0.899259 -0.489736 -0.180663 -0.585231
2 -0.461637 -0.586819 -0.058349  0.412567  0.024197  1.308825  0.187696
3  2.017703 -0.434376 -0.612436  0.670076 -0.762519 -0.692070  0.700587
4  2.285501 -0.960575  0.400247  1.382798 -1.239082  0.598040  0.445940

         a7        a8        a9  ...     a2040     a2041     a2042     a2043
\
0  0.599619 -0.401561 -0.764479  ... -0.936275 -1.105490 -0.688952 -0.606492
1 -0.219178 -0.273466 -0.063870  ... -0.402456 -1.000294 -0.401711  0.065169
2 -1.148081  0.485641  0.057516  ...  1.404732 -0.253244 -0.252336 -0.163078
3 -0.130921 -0.418634  0.306285  ... -0.946080 -0.624846  0.495533 -0.525775
4  0.310546 -0.556349  0.065078  ...  0.785869  0.099995  2.000137 -0.687619

      a2044     a2045     a2046     a2047       file  target
0 -0.351768 -0.495843 -0.923672 -0.439941  14986.jpg       0
1 -0.612849  1.329306 -0.189771  0.209548   3138.jpg       0
2 -0.785985  4.627949 -0.609841 -0.831458   1700.jpg       0
3 -0.308397  1.141025  1.933065 -0.388813  16257.jpg       0
4  0.121015  2.718197  1.353535  0.821876   2863.jpg       0

[5 rows x 2050 columns]
Test Data (Unlabeled):
(3000, 2048)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Columns: 2048 entries, a0 to a2047
dtypes: float64(2048)
memory usage: 46.9 MB
None
         a0        a1        a2        a3        a4        a5        a6  \
0  1.772767  0.019048 -0.778882 -0.623594  0.017255  1.187788  0.483584
1  0.774805 -0.512778 -0.725175 -0.114472  0.273637  0.607259 -0.420410
2  0.171195 -0.684613 -0.065444  0.224897 -0.887273  0.419544  0.132605
3  0.158327 -0.512745 -0.392124 -0.037202 -0.022930 -0.009357 -0.154215
4 -0.611404 -0.158482 -0.110539  0.014812 -0.549536  1.024947 -0.184827

         a7        a8        a9  ...     a2038     a2039     a2040     a2041
\
0 -1.521608 -0.572495  0.354573  ... -0.674792 -0.456242 -0.045960  0.005846
1  0.208231  0.379005 -0.039611  ... -0.561232  0.498043 -0.087619 -0.607783
2 -0.506722 -0.922514  0.466029  ...  1.272501 -0.362401 -0.746261 -0.420286
3 -0.471399 -0.018809 -0.593472  ... -1.052285  0.794199 -0.869422 -1.156542
4 -0.986893 -0.615515 -0.698443  ...  3.610997  0.993858 -0.093286 -0.070229

      a2042     a2043     a2044     a2045     a2046     a2047
0  1.321807  1.011435  0.762783  1.094154 -1.099026  0.234245
1 -0.601328 -0.332610 -0.457010  0.906401  0.867925 -0.283792
```

```
2  0.533070 -1.381722 -0.937565  2.841209  1.510577 -0.439566
3  0.079826  0.588280  0.435322  0.478258 -0.935483 -0.159332
4 -0.301869 -0.368704 -0.644924  1.019486 -0.611050 -0.458587

[5 rows x 2048 columns]
```

Split Data into Features and Labels

In [7]:

```
# Remove 'file' column from the training data
X = df.iloc[:, :-2]  # All columns except 'file' and 'target'
y = df['target']     # Target labels

print(f"Features shape: {X.shape}, Labels shape: {y.shape}")
```

```
Features shape: (14034, 2048), Labels shape: (14034,)
```

Logistic Regression

In [11]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

# Define hyperparameters to search
params = {'C': [0.01, 0.1, 1, 10], 'max_iter': [1000]}

# Initialize Logistic Regression
log_reg = LogisticRegression()

# Grid search with 10-fold CV
grid_log = GridSearchCV(log_reg, param_grid=params, cv=10, scoring='accuracy',
n_jobs=-1)
grid_log.fit(X, y)

# Print best parameters and accuracy
print("Logistic Regression Best Params:", grid_log.best_params_)
print("Logistic Regression Best Accuracy:", grid_log.best_score_)

# Create a DataFrame to store all hyperparameter combinations and their accura
cies
log_results = pd.DataFrame(grid_log.cv_results_)

# Display only relevant columns: params and mean_test_score (accuracy)
log_results_table = log_results[['param_C', 'param_max_iter', 'mean_test_scor
e']]
print("Logistic Regression Hyperparameter Results Table:")
print(log_results_table)
```

```
Logistic Regression Best Params: {'C': 0.01, 'max_iter': 1000}
Logistic Regression Best Accuracy: 0.937507691089302
Logistic Regression Hyperparameter Results Table:
   param_C  param_max_iter  mean_test_score
0     0.01            1000         0.937508
1     0.10            1000         0.929740
2     1.00            1000         0.926605
3    10.00            1000         0.925607
```

Confusion Matrix & Classification Report for Logistic Regression

In [12]:
```
from sklearn.metrics import confusion_matrix, classification_report

# Make predictions using the best logistic regression model
y_pred_log = grid_log.best_estimator_.predict(X)

# Generate confusion matrix
print("Confusion Matrix (Logistic Regression):\n", confusion_matrix(y, y_pred_
log))

# Generate classification report
print("Classification Report (Logistic Regression):\n", classification_report
(y, y_pred_log))
```

```
Confusion Matrix (Logistic Regression):
 [[2476    0    0    0    0   36]
 [   0 2368   14    0    0    0]
 [   0   21 2170    0    0    0]
 [   6    0    1 2266    0    1]
 [   3    0    0    0 2268    0]
 [  71    0    0    1    0 2332]]
Classification Report (Logistic Regression):
               precision    recall  f1-score   support

           0       0.97      0.99      0.98      2512
           1       0.99      0.99      0.99      2382
           2       0.99      0.99      0.99      2191
           3       1.00      1.00      1.00      2274
           4       1.00      1.00      1.00      2271
           5       0.98      0.97      0.98      2404

    accuracy                           0.99     14034
   macro avg       0.99      0.99      0.99     14034
weighted avg       0.99      0.99      0.99     14034
```

k-Nearest Neighbors

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier

# Define hyperparameters for kNN
params = {'n_neighbors': [3, 5, 10, 20]}

# Initialize kNN
knn = KNeighborsClassifier()

# Perform grid search with 10-fold CV
grid_knn = GridSearchCV(knn, param_grid=params, cv=10, scoring='accuracy', n_j
obs=-1)
grid_knn.fit(X, y)

# Print best parameters and accuracy
print("kNN Best Params:", grid_knn.best_params_)
print("kNN Best Accuracy:", grid_knn.best_score_)

# Create a DataFrame for kNN hyperparameter combinations and accuracies
knn_results = pd.DataFrame(grid_knn.cv_results_)

# Display only relevant columns: params and mean_test_score (accuracy)
knn_results_table = knn_results[['param_n_neighbors', 'mean_test_score']]
print("kNN Hyperparameter Results Table:")
print(knn_results_table)
```

```
kNN Best Params: {'n_neighbors': 5}
kNN Best Accuracy: 0.9270321736287522
kNN Hyperparameter Results Table:
   param_n_neighbors  mean_test_score
0                  3         0.924753
1                  5         0.927032
2                 10         0.926890
3                 20         0.925751
```

Confusion Matrix and Classification Report for kNN

In [ ]:
```
# Make predictions using the best kNN model
y_pred_knn = grid_knn.best_estimator_.predict(X)

# Generate confusion matrix
print("Confusion Matrix (kNN):\n", confusion_matrix(y, y_pred_knn))

# Generate classification report
print("Classification Report (kNN):\n", classification_report(y, y_pred_knn))
```

```
Confusion Matrix (kNN):
 [[2336    2    2    9    2  161]
 [   6 2275   90    6    5    0]
 [   6  159 2018    7    1    0]
 [  23    8    8 2228    0    7]
 [  40    2    4    3 2220    2]
 [ 155    4    7   14    7 2217]]
Classification Report (kNN):
               precision    recall  f1-score   support

           0       0.91      0.93      0.92      2512
           1       0.93      0.96      0.94      2382
           2       0.95      0.92      0.93      2191
           3       0.98      0.98      0.98      2274
           4       0.99      0.98      0.99      2271
           5       0.93      0.92      0.93      2404

    accuracy                           0.95     14034
   macro avg       0.95      0.95      0.95     14034
weighted avg       0.95      0.95      0.95     14034
```

Random Forest

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier

# Define hyperparameters for Random Forest
params = {'n_estimators': [50, 100], 'max_depth': [10, None]}

# Initialize Random Forest
rf = RandomForestClassifier()

# Perform grid search with 10-fold CV
grid_rf = GridSearchCV(rf, param_grid=params, cv=10, scoring='accuracy', n_job
s=-1)
grid_rf.fit(X, y)

# Print best parameters and accuracy
print("Random Forest Best Params:", grid_rf.best_params_)
print("Random Forest Best Accuracy:", grid_rf.best_score_)

# Create a DataFrame for Random Forest hyperparameter combinations and accurac
ies
rf_results = pd.DataFrame(grid_rf.cv_results_)

# Display only relevant columns: params and mean_test_score (accuracy)
rf_results_table = rf_results[['param_n_estimators', 'param_max_depth', 'mean_
test_score']]
print("Random Forest Hyperparameter Results Table:")
print(rf_results_table)
```

```
Random Forest Best Params: {'max_depth': 10, 'n_estimators': 100}
Random Forest Best Accuracy: 0.928885751533649
Random Forest Hyperparameter Results Table:
   param_n_estimators param_max_depth  mean_test_score
0                  50              10         0.924823
1                 100              10         0.928886
2                  50            None         0.926677
3                 100            None         0.928458
```

Confusion Matrix & Classification Report for Random Forest

In [ ]:
```python
# Make predictions using the best Random Forest model
y_pred_rf = grid_rf.best_estimator_.predict(X)

# Generate confusion matrix
print("Confusion Matrix (Random Forest):\n", confusion_matrix(y, y_pred_rf))

# Generate classification report
print("Classification Report (Random Forest):\n", classification_report(y, y_p
red_rf))
```

```
Confusion Matrix (Random Forest):
 [[2487    1    0    3    0   21]
 [   0 2364   18    0    0    0]
 [   0   43 2148    0    0    0]
 [  13    1    1 2259    0    0]
 [  11    0    1    0 2259    0]
 [ 123    2    1    5    2 2271]]
Classification Report (Random Forest):
               precision    recall  f1-score   support

           0       0.94      0.99      0.97      2512
           1       0.98      0.99      0.99      2382
           2       0.99      0.98      0.99      2191
           3       1.00      0.99      0.99      2274
           4       1.00      0.99      1.00      2271
           5       0.99      0.94      0.97      2404

    accuracy                           0.98     14034
   macro avg       0.98      0.98      0.98     14034
weighted avg       0.98      0.98      0.98     14034
```

XGBoost

In [ ]:
```
from xgboost import XGBClassifier

# Define hyperparameters for XGBoost
params = {'n_estimators': [50], 'max_depth': [3], 'learning_rate': [0.1]}

# Initialize XGBoost (running on CPU)
xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', tree_meth
od='hist')

# Perform grid search with 10-fold CV
grid_xgb = GridSearchCV(xgb, param_grid=params, cv=10, scoring='accuracy', n_j
obs=-1)

grid_xgb.fit(X, y)

# Print best parameters and accuracy
print("XGBoost Best Params (CPU):", grid_xgb.best_params_)
print("XGBoost Best Accuracy (CPU):", grid_xgb.best_score_)

# Create a DataFrame for XGBoost hyperparameter combinations and accuracies
xgb_results = pd.DataFrame(grid_xgb.cv_results_)

# Display only relevant columns: params and mean_test_score (accuracy)
xgb_results_table = xgb_results[['param_n_estimators', 'param_max_depth', 'par
am_learning_rate', 'mean_test_score']]
print("XGBoost Hyperparameter Results Table (CPU):")
print(xgb_results_table)
```

```
/usr/local/lib/python3.10/dist-packages/joblib/externals/loky/process_executo
r.py:752: UserWarning: A worker stopped while some jobs were given to the exe
cutor. This can be caused by a too short worker timeout or by a memory leak.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [1
5:38:33] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)

XGBoost Best Params (CPU): {'learning_rate': 0.1, 'max_depth': 3, 'n_estimato
rs': 50}
XGBoost Best Accuracy (CPU): 0.9226157115501378
XGBoost Hyperparameter Results Table (CPU):
   param_n_estimators  param_max_depth  param_learning_rate  mean_test_score
0                  50                3                  0.1         0.922616
```

Confusion Matrix & Classification Report XGBoost

In [ ]:
```
# Make predictions using the best XGBoost model
y_pred_xgb = grid_xgb.best_estimator_.predict(X)

# Generate confusion matrix
print("Confusion Matrix (XGBoost):\n", confusion_matrix(y, y_pred_xgb))

# Generate classification report
print("Classification Report (XGBoost):\n", classification_report(y, y_pred_xg
b))
```

```
Confusion Matrix (XGBoost):
 [[2383    2    3   16    5  103]
 [   5 2262  109    3    3    0]
 [   6  131 2040   12    2    0]
 [  19    2   15 2229    3    6]
 [  19    2    6    3 2241    0]
 [ 223   10   10   29   11 2121]]
Classification Report (XGBoost):
               precision    recall  f1-score   support

           0       0.90      0.95      0.92      2512
           1       0.94      0.95      0.94      2382
           2       0.93      0.93      0.93      2191
           3       0.97      0.98      0.98      2274
           4       0.99      0.99      0.99      2271
           5       0.95      0.88      0.92      2404

    accuracy                           0.95     14034
   macro avg       0.95      0.95      0.95     14034
weighted avg       0.95      0.95      0.95     14034
```

Fully Connected Neural Network

In [8]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
import pandas as pd

# Define hyperparameters for Fully Connected Neural Network
params = {'hidden_layer_sizes': [(128,), (256, 128)], 'alpha': [0.0001, 0.00
1]}

# Initialize Fully Connected Neural Network Classifier
mlp = MLPClassifier(max_iter=1000)

# Perform grid search with 10-fold CV
grid_mlp = GridSearchCV(mlp, param_grid=params, cv=10, scoring='accuracy', n_j
obs=-1)
grid_mlp.fit(X, y)

# Print best parameters and accuracy
print("MLP Best Params:", grid_mlp.best_params_)
print("MLP Best Accuracy:", grid_mlp.best_score_)

# Create a DataFrame for Fully Connected Neural Network hyperparameter combina
tions and accuracies
mlp_results = pd.DataFrame(grid_mlp.cv_results_)

# Display only relevant columns: params and mean_test_score (accuracy)
mlp_results_table = mlp_results[['param_hidden_layer_sizes', 'param_alpha', 'm
ean_test_score']]
print("MLP Hyperparameter Results Table:")
print(mlp_results_table)
```

```
MLP Best Params: {'alpha': 0.0001, 'hidden_layer_sizes': (256, 128)}
MLP Best Accuracy: 0.9363676838195728
MLP Hyperparameter Results Table:
  param_hidden_layer_sizes  param_alpha  mean_test_score
0                   (128,)       0.0001         0.934657
1               (256, 128)       0.0001         0.936368
2                   (128,)       0.0010         0.934728
3               (256, 128)       0.0010         0.933589
```

Confusion Matrix & Classification Report for Fully Connected Neural Network

In [9]:
```
from sklearn.metrics import confusion_matrix, classification_report

# Make predictions using the best Fully Connected Neural Network model
y_pred_mlp = grid_mlp.best_estimator_.predict(X)

# Generate confusion matrix
print("Confusion Matrix Fully Connected Neural Network:\n", confusion_matrix
(y, y_pred_mlp))

# Generate classification report
print("Classification Report Fully Connected Neural Network:\n", classificatio
n_report(y, y_pred_mlp))
```

```
Confusion Matrix Fully Connected Neural Network:
 [[2477    0    0    1   17   17]
 [   0 2378    4    0    0    0]
 [   0   54 2135    0    1    1]
 [   0    0    0 2273    0    1]
 [   0    0    0    0 2271    0]
 [   7    1    1    1    8 2386]]
Classification Report Fully Connected Neural Network:
               precision    recall  f1-score   support

           0       1.00      0.99      0.99      2512
           1       0.98      1.00      0.99      2382
           2       1.00      0.97      0.99      2191
           3       1.00      1.00      1.00      2274
           4       0.99      1.00      0.99      2271
           5       0.99      0.99      0.99      2404

    accuracy                           0.99     14034
   macro avg       0.99      0.99      0.99     14034
weighted avg       0.99      0.99      0.99     14034
```

Identify Misclassified Examples and Get Probabilities

In [16]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2

# Get the predicted probabilities
y_probs_log = grid_log.best_estimator_.predict_proba(X)

# Identify the misclassified examples
y_pred_log = grid_log.best_estimator_.predict(X)
misclassified_indices = np.where(y_pred_log != y)[0]

# Create a function to find the worst misclassified examples for each class
def find_worst_misclassified(y_true, y_pred, y_probs, misclassified_indices):
    worst_misclassified = {}
    for class_label in np.unique(y_true):
        # Get indices of misclassified examples for this class
        class_indices = np.where(y_true == class_label)[0]
        class_misclassified = np.intersect1d(misclassified_indices, class_indi
ces)

        if len(class_misclassified) > 0:
            # Find the example with the lowest probability for its correct cla
ss
            lowest_prob_index = class_misclassified[np.argmin(y_probs[class_mi
sclassified, class_label])]
            worst_misclassified[class_label] = (lowest_prob_index, y_probs[low
est_prob_index, class_label])

    return worst_misclassified

# Get the worst misclassified examples
worst_misclassified_log = find_worst_misclassified(y, y_pred_log, y_probs_log,
misclassified_indices)

# Print the details of the worst misclassified examples for each class
print("Worst Misclassified Examples (Logistic Regression):")
for class_label, (index, prob) in worst_misclassified_log.items():
    true_label = y[index]
    predicted_label = y_pred_log[index]
    print(f"Class {class_label}:")
    print(f"  - Index: {index}")
    print(f"  - True Label: {true_label}")
    print(f"  - Predicted Label: {predicted_label}")
    print(f"  - Probability of Correct Label: {prob:.4f}")
    print(f"  - File: {df.iloc[index]['file']}\n")
```

```
Worst Misclassified Examples (Logistic Regression):
Class 0:
  - Index: 582
  - True Label: 0
  - Predicted Label: 5
  - Probability of Correct Label: 0.0126
  - File: 16636.jpg

Class 1:
  - Index: 4200
  - True Label: 1
  - Predicted Label: 2
  - Probability of Correct Label: 0.2156
  - File: 3440.jpg

Class 2:
  - Index: 5196
  - True Label: 2
  - Predicted Label: 1
  - Probability of Correct Label: 0.0092
  - File: 10899.jpg

Class 3:
  - Index: 7119
  - True Label: 3
  - Predicted Label: 0
  - Probability of Correct Label: 0.0992
  - File: 6337.jpg

Class 4:
  - Index: 10574
  - True Label: 4
  - Predicted Label: 0
  - Probability of Correct Label: 0.1373
  - File: 1705.jpg

Class 5:
  - Index: 13975
  - True Label: 5
  - Predicted Label: 0
  - Probability of Correct Label: 0.0254
  - File: 13189.jpg
```

Plot the Worst Misclassified Examples

In [ ]:

```
# Unzip the file to the dataset directory without showing all the output
!unzip -q /content/drive/MyDrive/dataset/all.zip -d /content/drive/MyDrive/dataset/unzipped/
```

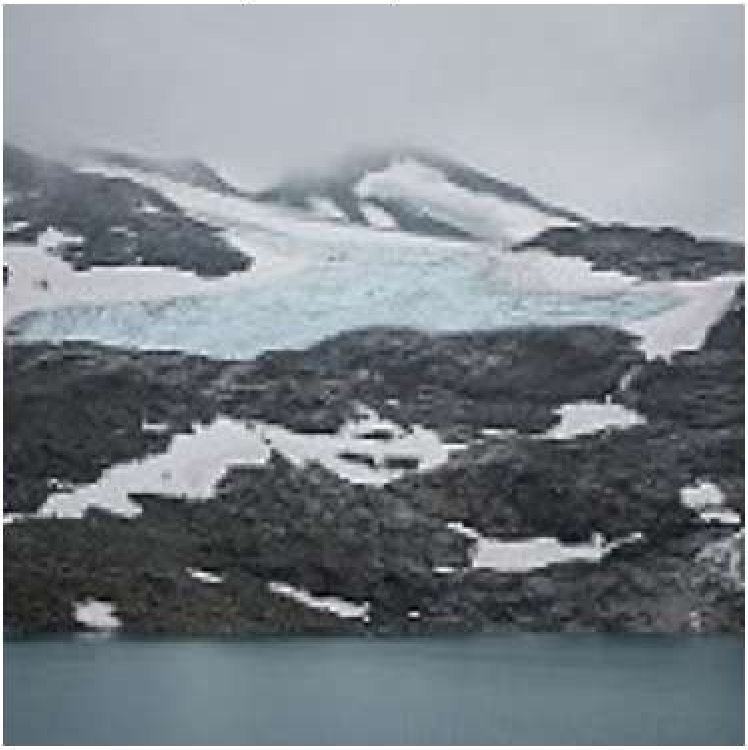
```
replace /content/drive/MyDrive/dataset/unzipped/all/9733.jpg? [y]es, [n]o,
[A]ll, [N]one, [r]ename: n
replace /content/drive/MyDrive/dataset/unzipped/all/14147.jpg? [y]es, [n]o,
[A]ll, [N]one, [r]ename: N
```

In [31]:
```
import matplotlib.pyplot as plt
import cv2
import os

# Function to display images with misclassification details
def plot_misclassified_image(index, true_label, predicted_label, prob):
    filename = df.iloc[index]['file']
    img_path = os.path.join('/content/drive/MyDrive/dataset/unzipped/all', fil
ename)
    if os.path.exists(img_path):
        img = cv2.imread(img_path)
        if img is not None:
            plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
            plt.title(f"True: {true_label}, Pred: {predicted_label}, Prob: {pr
ob:.4f}")
            plt.axis('off')
            plt.show()
        else:
            print(f"Could not load image data: {img_path}")
    else:
        print(f"File does not exist: {img_path}")

# Plot the worst misclassified example for each class
for class_label, (index, prob) in worst_misclassified_log.items():
    true_label = y[index]
    predicted_label = y_pred_log[index]
    plot_misclassified_image(index, true_label, predicted_label, prob)
```

True: 0, Pred: 5, Prob: 0.0126



True: 1, Pred: 2, Prob: 0.2156

True: 2, Pred: 1, Prob: 0.0092

True: 3, Pred: 0, Prob: 0.0992

True: 4, Pred: 0, Prob: 0.1373

True: 5, Pred: 0, Prob: 0.0254