UNIVERSITY OF TRIESTE

DEPARTMENT OF ENGINEERING AND ARCHITECTURE

Computer Vision and Pattern Recognition

CVPR 2024 CNN classifier - Project Report

Professor:
Felice Andrea Pellegrino

Student:
Matteo Crosariol

Academic year 2024/2025

Contents

1 Problem Statement

This project requires the implementation of an image classifier based on the bag-of-words approach. The provided dataset (from [Lazebnik et al., 2006][1]), contains 15 categories (office, kitchen, living room, bedroom, store, industrial, tall building, inside city, street, highway, coast, open country, mountain, forest, suburb), and is already divided in training set and test set. Samples are shown in Fig. 1.

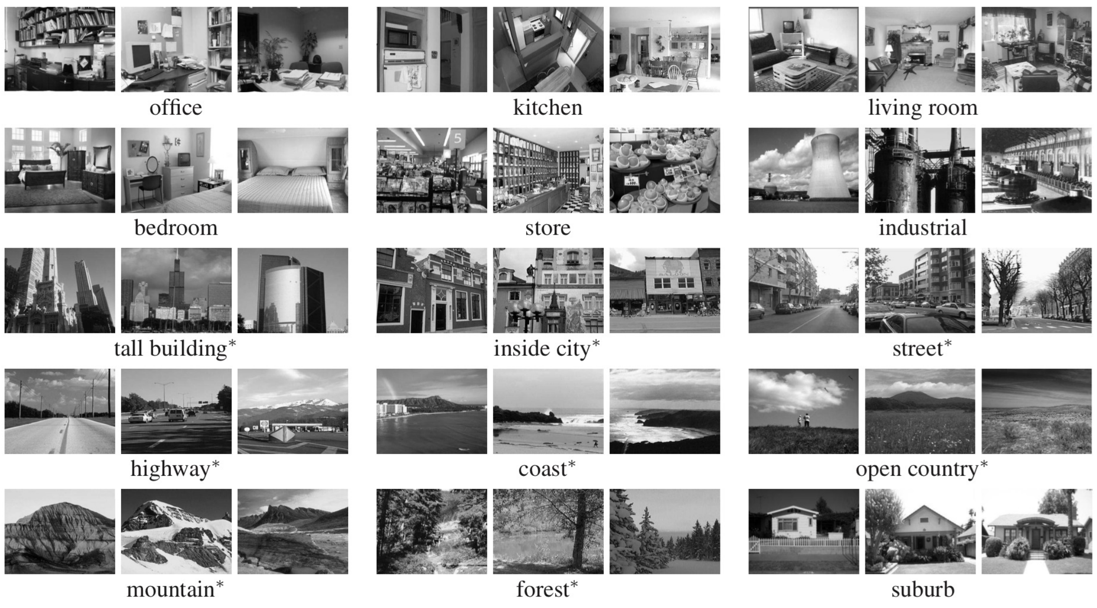

Figure 1: Examples of images from each of the 15 categories of the provided dataset (the same as [Lazebnik et al., 2006]).

The project mainly is divided in three parts:

1. **Base CNN Classifier**: Initialization and training of a CNN model with a fixed architecture.

2. **Improved CNN Classifier**: Optimization of the previous result by making use of data augmentation, regularization and the employment an ensemble of networks (ten), trained independently.

3. **Transfer learning based solution**: Improvement on the previous results through transfer learning, by finetuning a pre-trained model to the given dataset.

2 Base CNN Classifier

2.1 Task specification

The goal of this task is to train a shallow network from scratch according to the following specifications:

- the architecture of the network is shown in Table 1;
- since the input image is 64x64 it is needed to resize the images in order to feed them to the network;
- split the provided training set in 85% for actual training set and 15% to be used as validation set;
- employ the stochastic gradient descent with momentum optimization algorithm;
- use minibatches of size 32;
- set the initial bias values to 0 and use initial weights drawn from a Gaussian distribution having a mean of 0 and a standard deviation of 0.01;

#	Type	Size
1	Image Input	64x64x1 images
2	Convolution	8 3x3 convolutions with stride 1
3	ReLu	
4	Max Pooling	2x2 max pooling with stride 2
5	Convolution	16 3x3 convolutions with stride 1
6	ReLu	
7	Max Pooling	2x2 max pooling with stride 2
8	Convolution	32 3x3 convolutions with stride 1
9	ReLu	
10	Fully Connected	15
11	Softmax	softmax
12	Classification Output	crossentropyex

Table 1: Layout of the CNN used in the Base CNN Classifier

2.2 Implementation

2.2.1 Dataset loading and transformations definition

```
# Define transformations and datasets
transform = transforms.Compose([
    transforms.Resize((64, 64)),
    transforms.Grayscale(),
    transforms.ToTensor(), #conversion to tensor.
    transforms.Lambda(lambda x: x * 255)
])
```

This transformation rescales the images anisotropically to a size of 64x64, converts them to grayscale (since they will be loaded as RGB images due to the use of ImageFolder) ,converts them to a tensor and then scales the values back to [0,255] since ToTensor converts the image to a sensor with values in the range [0,1].

After the splitting of the provided training set in 85% for actual training set and 15% to be used as validation set and the application of the transformations on the data, the three DataLoaders (train, validation and test) have been created.

```
full_training_data = datasets.ImageFolder(root="Dataset"+"/train")
test_dataset = datasets.ImageFolder(root="Dataset"+"/test")
full_training_data.transform=transform
test_dataset.transform=transform

train_size = int(SPLIT_RATIO_TRAINING * len(full_training_data))
val_size = len(full_training_data) - train_size
train_dataset, val_dataset = torch.utils.data.random_split(full_training_data,
[train_size, val_size])
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
validation_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=True)
```

2.2.2 Model implementation and initialization

The architecture of the network is implemented using PyTorch:

```
self.layers = nn.Sequential(
   nn.Conv2d(in_channels=1, out_channels=8,kernel_size=3,stride=1, padding=1),
   nn.ReLU(),
   nn.MaxPool2d(kernel_size=2, stride=2),
   nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3,stride=1, padding=1),
   nn.ReLU(),
   nn.MaxPool2d(kernel_size=2, stride=2),
   nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3,stride=1, padding=1),
   nn.ReLU(),
   nn.Flatten(),
   nn.Linear(in_features=self.last_input_size, out_features=15)
)
```

While the Convolutional and Linear layers are then initialized with the following code:

```
        if type(model) == nn.Conv2d or type(model) == nn.Linear:
            nn.init.normal_(model.weight, mean=0.0, std=0.01)
            if model.bias is not None:
                nn.init.constant_(model.bias, 0.0)
```

2.2.3 Training and Validation

Regarding the training the parameters used are:

- **Learning rate**:0.001. This value has been chosen after some trial, it's not too big to overshoot nor too small to slow down the training too much;
- **Momentum**: 0.9. This value helps the model converge faster and avoid getting stuck in local minima.

In addition, in order to stop the training before reaching the epochs limit the parameter **no_improvement_counter** is used. Its value is 15 and it describes the number of epochs without improvement after which the training is stopped using the validation loss as a stopping criterion.

Firs of all the model has to be initialized,after that the optimizer (stochastic gradient descent) and the loss function (cross entropy) are defined:

```
model = CNN_task_1()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001,momentum=0.9)
loss = torch.nn.CrossEntropyLoss()
```

For each epoch the model is trained, and the training and validation losses and accuracies are calculated in order track the best performance (based on validation), and save the model.

```
#TRAINING
      model.train(True)
      for x, y in iter(train_loader):
          optimizer.zero_grad()
          y_pred = model(x)
          l = loss(y_pred, y) #compute the loss
          l.backward() #backward pass
          optimizer.step() #update the weights

      training_loss, training_accuracy = calculate_loss_accuracy(model, train_loader,
       loss)

      #VALIDATION
      model.eval()
      with torch.no_grad():
          validation_loss, validation_accuracy = calculate_loss_accuracy(model,
           validation_loader, loss)
          validation_losses.append(validation_loss)
          validation_accuracies.append(validation_accuracy)
          if validation_loss < best_validation_loss:
              print(f"!!! NEW BEST MODEL FOUND")
              best_model = model.state_dict()
              best_validation_loss = validation_loss
              no_improvement_counter=0
          else:
              no_improvement_counter +=1
              if no_improvement_counter==15: #stop validation
                  break #exit here
```

2.2.4 Testing

Load the best model and evaluate the performance on the test set:

```
#LOAD THE BEST MODEL
model.load_state_dict(best_model)
torch.save(best_model, "models_1.pt")

#TEST MODEL
total = 0
correct = 0
all_predictions = []
all_labels = []
with torch.no_grad():
    for x_test, y_test in test_loader:
        y_pred_test = model(x_test)
        _, predicted = torch.max(y_pred_test.data, 1)
        total += y_test.size(0)
        correct += (predicted == y_test).sum().item()
        all_predictions.extend(predicted.numpy())
        all_labels.extend(y_test.numpy())

test_accuracy = 100 * correct / total
```

2.3 Results

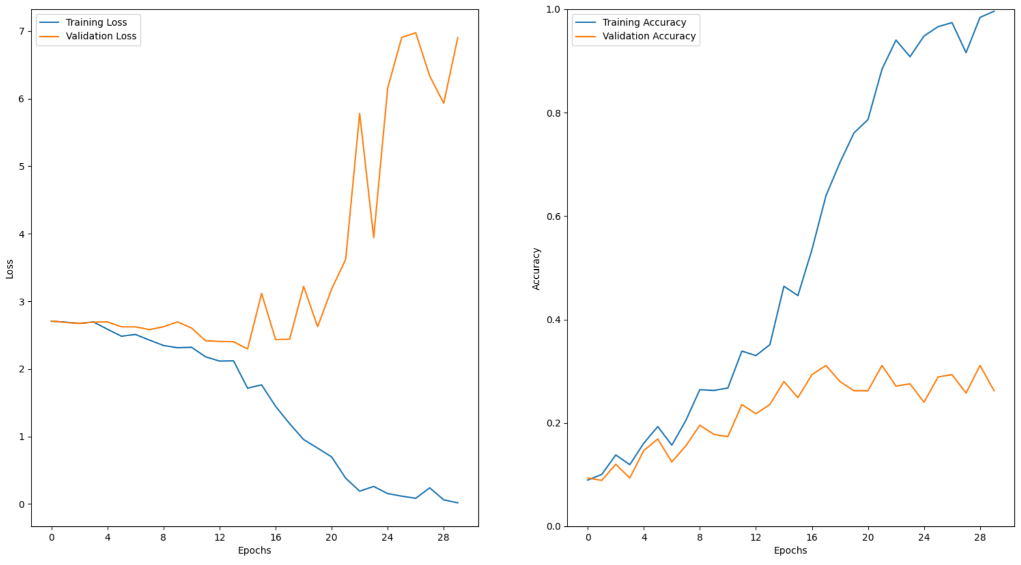

Figure 2: Training and validation losses and accuracies of the base CNN classifier.

It can be seen that after few epochs while the training accuracy decrease, the validation accuracy start increasing,so the model quickly starts overfitting.

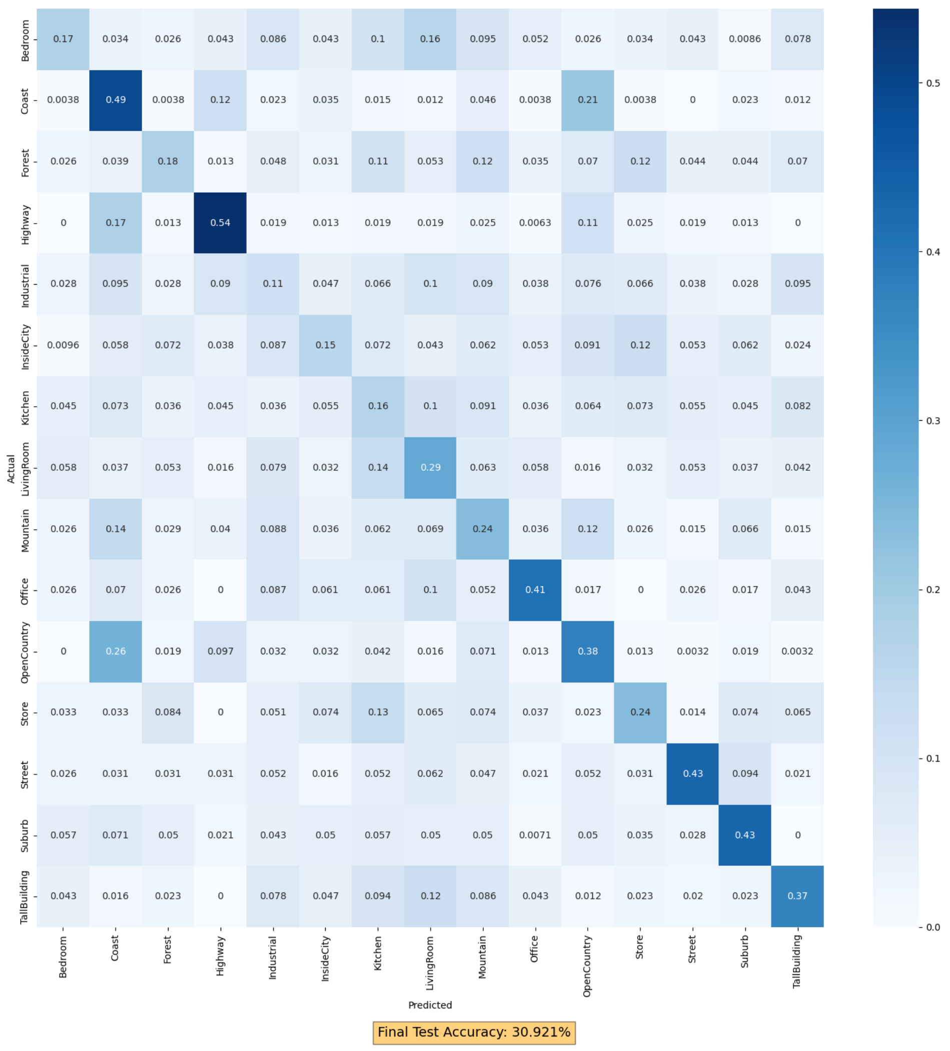

Figure 3: Confusion matrix of the base CNN classifier on the test set.

The test accuracy of the model is about 30%. From the confusion matrix, it's clear that the model is more precise in categorizing images that belong to certain classes, in fact it seems like the model is more precise in categorizing images referred to open spaces respect to the others.

3 Improved CNN Classifier

The goal of this task is to improve the previous result, according to different techniques and modifications, in fact this task has been divided in three different subtasks:

1. **Data augmentation**: applied to the training set in order to improve the generalization of the model.
2. **Regularization and Architectural changes:** : applied to the model in order to improve the performance and to prevent overfitting.
3. **Ensemble of networks**: Use an ensemble of networks (ten), trained independently in order to improve the performance.

3.1 Data augmentation

3.1.1 Task specification

The following data augmentation techniques have been applied to the training set:

- Random crop: The image is randomly cropped first into a 180x180 image;
- Rescaling: The image is resized to 64x64;
- Random horizontal flip: The image is flipped horizontally with a probability of 0.5.
- Random rotation: The image is rotated by a random angle between -15 and 15 degrees;

For which regarding the learning rate parameter and the momentum parameter are 0.001 and 0.9 respectively. In order to stop the training before reaching the epochs limit the parameter **no_improvement_counter** is used and Its value is 25.

3.1.2 Implementation

According to the techniques described above the transformation applied to the training set are:

```
    data_augmentation_transform = transforms.Compose([
     transforms.RandomCrop(180),
     transforms.Resize((64, 64)),
     transforms.RandomHorizontalFlip(p=0.5),
     transforms.RandomRotation(degrees=15),
     transforms.Grayscale(),
     transforms.ToTensor(),
 ])
```

3.1.3 Result

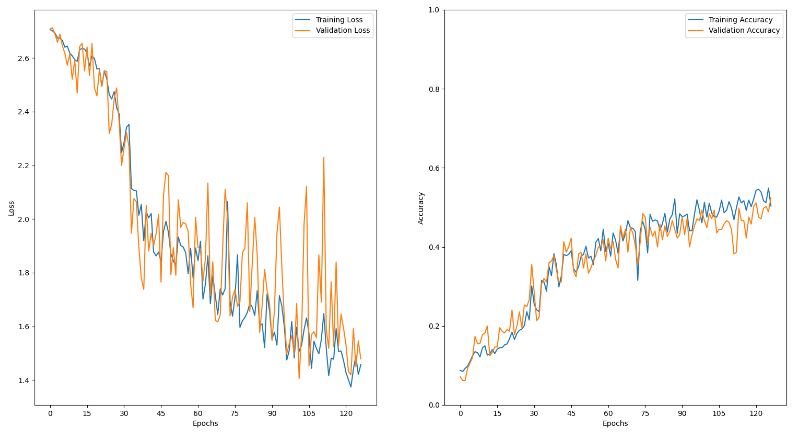

Figure 4: Training and validation losses and accuracies of the improved CNN classifier with only data augmentation.

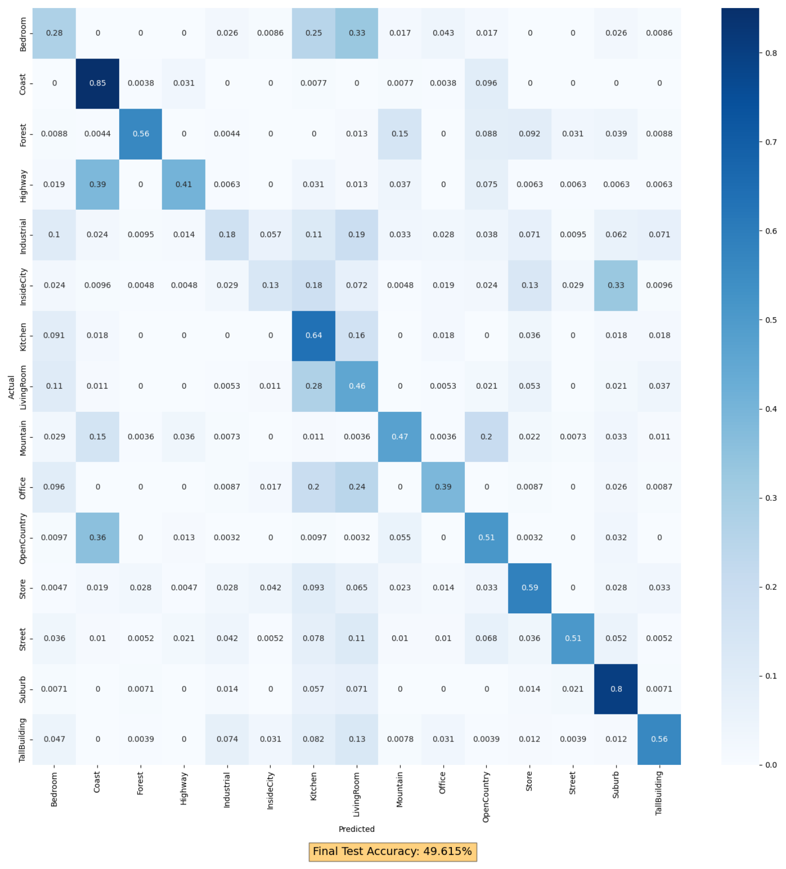

Figure 5: Confusion matrix of the improved CNN classifier with only data augmentation.

The test accuracy of the model is about 49% which is good improvement from the previous case and also now there is no more overfitting. Implementing some data augmentation that lead to improve generalization is a good point of start for improving the model.

3.2 Regularization and Architectural changes

3.2.1 Task specification

In adding to the data augmentation presented before in this subtask there are the following arrangements:

- batch normalization [Ioffe and Szegedy, 2015][2] : add batch normalization layers before the reLU layers;
- dropout: add some dropout layer to improve regularization, the dropout rate has been set to 0.1;
- Data normalization: The image is normalized with mean 0.5 and standard deviation 0.5;

For which regarding the learning rate parameter and the momentum parameter are 0.001 and 0.9 respectively. In order to stop the training before reaching the epochs limit the parameter **no_improvement_counter** is used and Its value is 25.

3.2.2 Implementation

The architecture of this new improved network is:

```
self.layers = nn.Sequential(
    nn.Conv2d(in_channels=1, out_channels=8, kernel_size=3, stride=1, padding=1),
    nn.BatchNorm2d(8),
    nn.ReLU(),
    nn.Dropout(.1),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3, stride=1, padding=1),
    nn.BatchNorm2d(16),
    nn.ReLU(),
    nn.Dropout(.1),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1),
    nn.BatchNorm2d(32),
    nn.ReLU(),
    nn.Flatten(),
    nn.Linear(in_features=self.last_input_size, out_features=15)
)
```

3.2.3 Result

The test accuracy of the model is about 57% which is good improvement from the previous case and there is no overfitting. From the confusion matrix, it's clear that the model is more precise but it seems that the model has still some difficult to classify the images that belongs to similar category, for examble the interior spaces like bedroom, office, kitchen,store and living room.

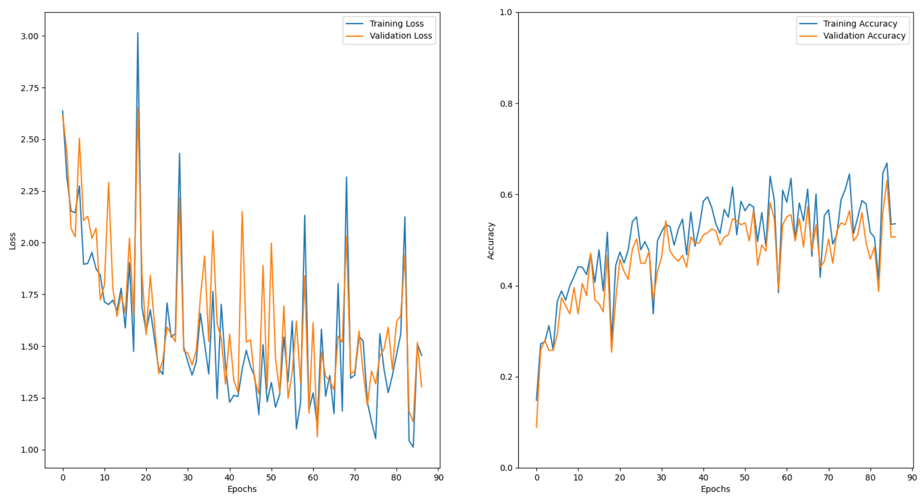

Figure 6: Training and validation losses and accuracies of the improved CNN classifier.

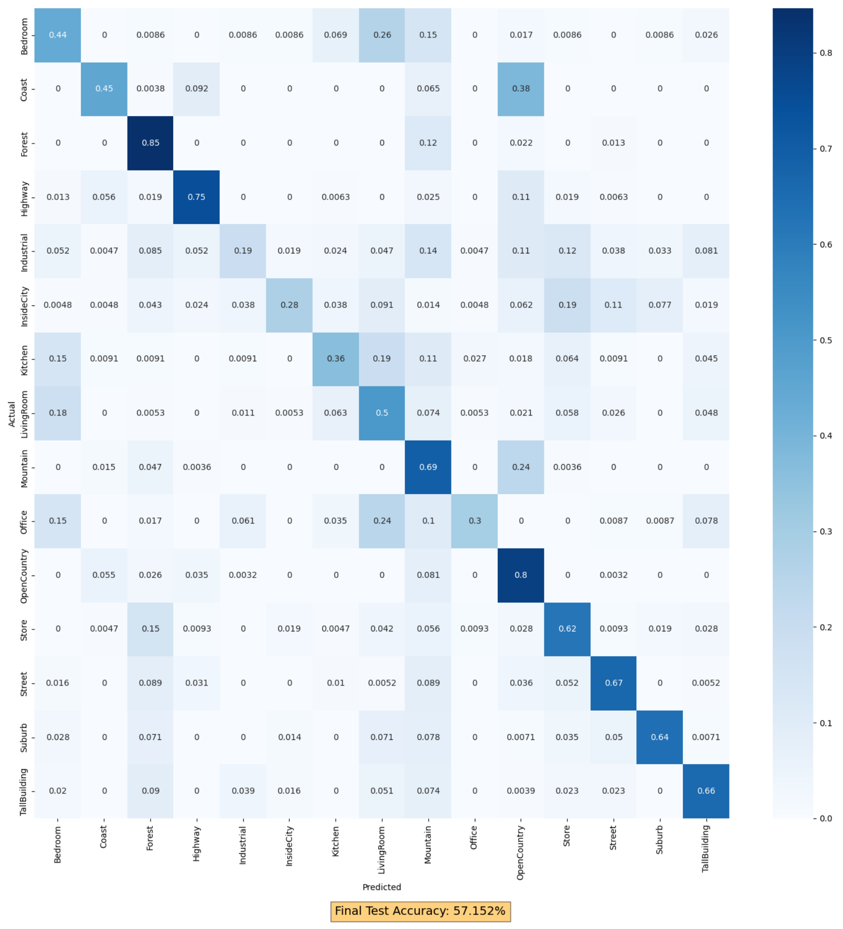

Figure 7: Confusion matrix of the improved CNN classifier.

3.3 Ensemble of networks

3.3.1 Task specification

The goal of this subtask is to employ an ensemble of networks (ten), trained independently and to use the arithmetic average of the outputs to assign the class, as in [Szegedy et al., 2015].[3]. The network architecture used is the same as the one of the previous subtask.

For which regarding the learning rate parameter and the momentum parameter are 0.001 and 0.9 respectively. In order to stop the training before reaching the epochs limit the parameter **no_improvement_counter** is used and Its value is 50.

3.3.2 Implementation

The architecture of this new improved network is:

```
class ensemble(nn.Module):
    def __init__(self, number_of_networks : int):
        super().__init__()

        self.models = nn.ModuleList([
            CNN_task_2() for _ in range(number_of_networks)])

    def forward(self, x):
        outputs = [model(x) for model in self.models]
        return sum(outputs) / len(outputs)
        )
```

And after that there is the training of the networks independently:

```
model =ensemble(number_of_networks)
no_improvement_counter_limit= int (50)

loss = torch.nn.CrossEntropyLoss()
optimizer = [torch.optim.SGD(model_voter.parameters(), lr=0.002, momentum=0.9)
 for model_voter in model.models]


for epoch in range(EPOCHS_LIMIT):
    #TRAINING
    print('EPOCH {}:'.format(epoch+ 1))

    for i, model_voter in enumerate(model.models):
        #train one epoch
        model_voter.train(True)
        for x, y in iter(train_loader):
            optimizer[i].zero_grad()
            y_pred = model_voter(x)
            l = loss(y_pred, y) #compute the loss
            l.backward() #backward pass
            optimizer[i].step() #update the weights
```

3.3.3 Result

The test accuracy of the model is about 61%, a result that is slight better from the previous case test and still there is not overfitting. It follows from the matrix that the previous problem of categorization of the images belonging to interior space is still present.

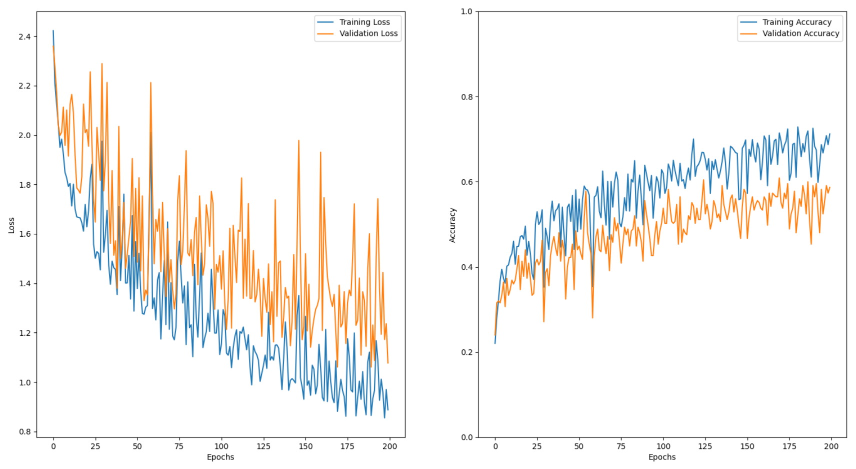

Figure 8: Training and validation losses and accuracies of the results of ten ensemble networks trained independently.

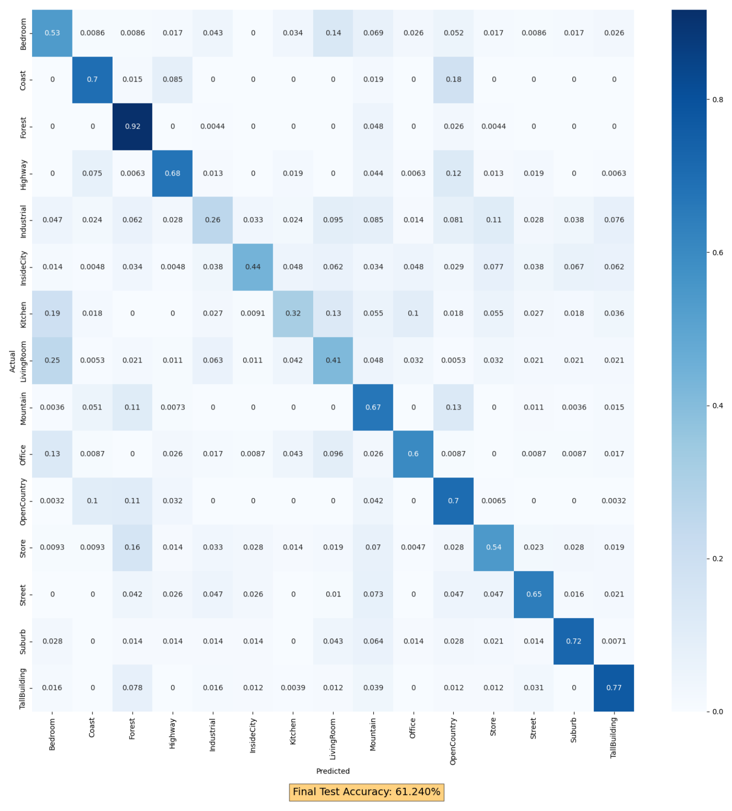

Figure 9: Confusion matrix of the results of ten ensemble networks trained independently on the test set.

4 Transfer learning based solution

Use transfer learning based on a pre-trained network,AlexNet [Krizhevsky et al., 2012] [4], in the following manner:

- freeze the weights of all the layers but the last fully connected layer and fine-tune the weights of the last layer based on the same train and validation sets employed before;

4.1 Task specification

According to the use of the AlexNet model our dataset need some transformation. The images need to be resized to 224x224, duplicated 3 times because of the architecture and characteristics of the model, and the images need to normalized with the mean and standard deviation of the dataset used for training the model.

For which regarding the learning rate parameter and the momentum parameter are 0.001 and 0.9 respectively. In order to stop the training before reaching the epochs limit the parameter **no_improvement_counter** is used. Its value is 25 and it describes the number of epochs without improvement after which the training is stopped using the validation loss as a stopping criterion.

4.2 Implementation

The data transform for the validation and test dataset is defined ad follows:

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.Grayscale(num_output_channels=3),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

while the data transform for the training set is defined as follows:

```
data_augmentation_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=15),
    transforms.RandomChoice([
        transforms.Resize((224, 224)),
        transforms.Resize((224, 224)),
        transforms.Compose([
            transforms.RandomCrop(180),
            transforms.Resize(224)
        ])
    ]),
    transforms.Grayscale(num_output_channels=3),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

For which regarding the model, its weights have to be frozen, except for the classifier?s. Its implementation is :

```
def __init__(self, mean_initialization : float = 0.0,
std_initialization : float = 0.01):
    super().__init__()

    self.mean_initialization = mean_initialization
    self.std_initialization = std_initialization

    self.alexnet = models.alexnet(weights=AlexNet_Weights.DEFAULT)
    self.alexnet.classifier[6] = nn.Linear(4096, 15)
    # Freeze all layers except the last one:
    for param in self.alexnet.parameters():
        param.requires_grad = False
    self.alexnet.classifier[6].apply(self._init_weights)
    self.alexnet.classifier[6].requires_grad = True
    self.alexnet.classifier[6].weight.requires_grad = True
    self.alexnet.classifier[6].bias.requires_grad = True
```

4.3 Result

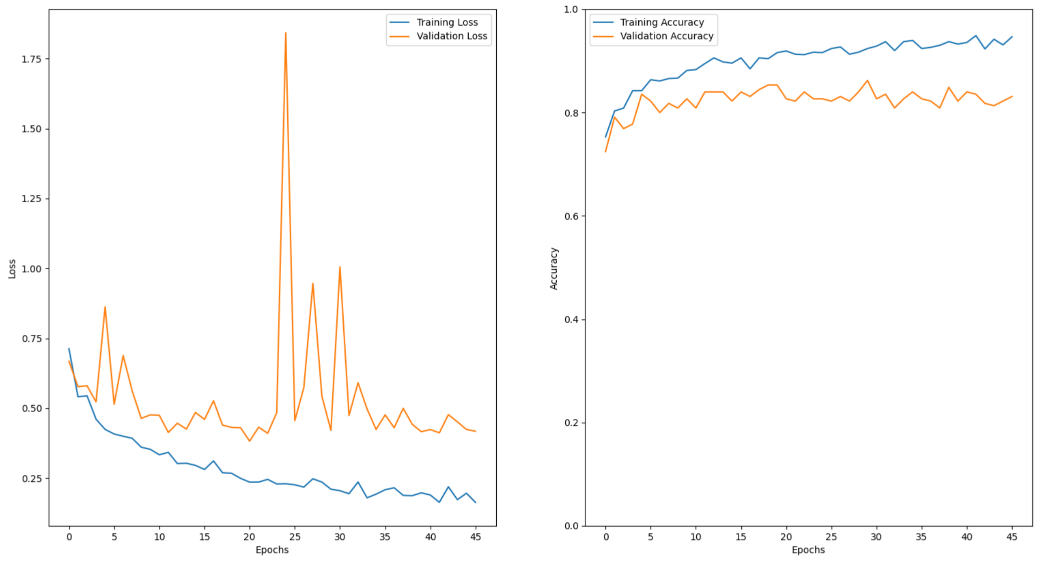

Figure 10: Training and validation losses and accuracies of the CNN classifier with AlexNet as a pretrained network.

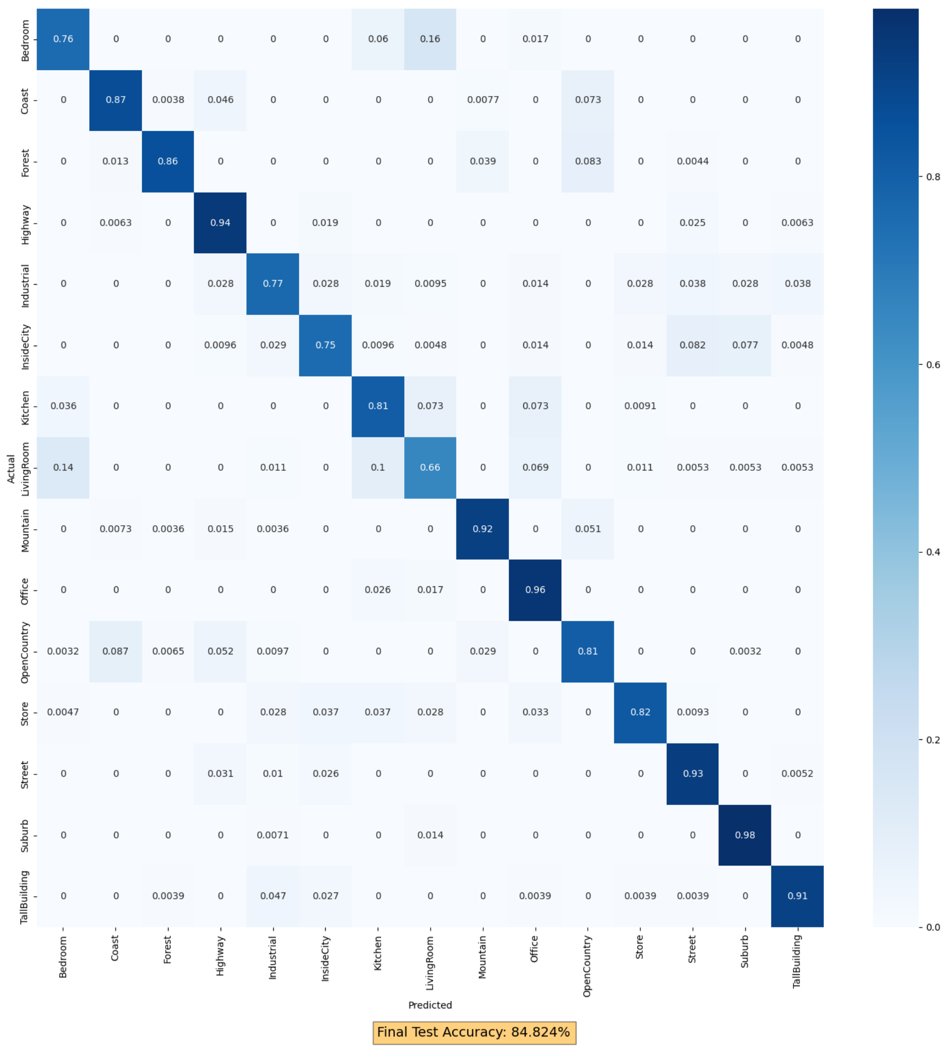

Figure 11: Confusion matrix of the CNN classifier with AlexNet as a pretrained network on the test set.

It's a very good improvement from the previous tasks, in fact the test accuracy of the model is about 85% and the model converges quickly. From the confusion matrix it can be recognised that the model has a slight difficult classification problem with the two classes bedroom and living room and with coast and open country.

References

[1] [Lazebnik et al., 2006] Lazebnik, S., Schmid, C., and Ponce, J. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06),*, volume 2, pages 2169-2178.IEEE.

[2] [Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167.*

[3] [Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[4] [Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*,pages 1097?1105.,